CHAPTER 10

Space–Time Tradeoffs

An important question in the study of computation is how best to use the registers of a CPU and/or the random-access memory of a general-purpose computer. In most computations, the number of registers (space) available is insufficient to hold all the data on which a program operates and registers must be reused. If the space is increased, the number of computation steps (time) can generally be reduced. This is an example of a space–versus–time tradeoff. In this chapter we examine tradeoffs between the number of storage locations and computation time using the pebble game and the branching program model.

The pebble game assumes that computations are done with straight-line programs in a data-independent fashion. Each such program is modeled by a directed acyclic graph. A pebble on a vertex indicates that its value is in a register. The goal of the game is to pebble the output vertices of the graph with numbers of pebbles (space) and steps (time) that are minimal, that is, neither can be reduced without increasing the other.

A branching program models data-dependent computation under the assumption that input variables assume a bounded number of values. Such a program is defined by a directed acyclic multigraph (there may be more than one edge between vertices) that specifies the order in which inputs are read. Time is the length of the longest path in a multigraph and space is the logarithm of its number of vertices.

For both models we present techniques to derive lower bounds on the exchange of space S for time T. For most problems examined here these exchanges are of the form $ST = \Omega(n^2)$, where n is the size of the problem input. Upper bounds on ST are obtained by evaluating S and T for particular algorithms.

Because the branching program is more general than the pebble game, it is more difficult to obtain good lower bounds with it, and for this reason we begin with the pebble game. In addition, the pebble game is appropriate for problems such as integer multiplication, convolution, and matrix multiplication on which only straight-line programs are used. For other problems, such as merging and sorting, the algorithms used typically involve branching and for them the branching program is the better model.

We also exhibit extreme results for the pebble game by showing that the time to pebble some graphs goes from minimal to exponential in the size of the graphs when the number of pebbles changes by 1, a warning against trying too hard to minimize the number of CPU registers used in a computation.

10.1 The Pebble Game

The pebble game is a game played on directed acyclic graphs (DAGs), which capture the dependencies of straight-line programs studied in Chapters 2 and 6. Algorithms for many important problems, such as the FFT and matrix multiplication, are naturally computed by straight-line programs. In the pebble game pebbles are placed on vertices of a DAG to indicate that the value associated with a vertex resides in a register. Pebbles are placed on vertices in a data-independent order.

In this game a pebble can be placed on an input vertex at any time and on any non-input vertex whose immediate predecessor vertices carry pebbles. The goal of the game is to place pebbles on each output vertex. A pebble can be removed from a vertex, including an output vertex, at any time after it has been pebbled. These rules are summarized below.

The rules of the **pebble game** are the following:

- (Initialization) A pebble can be placed on an input vertex at any time.
- (Computation Step) A pebble can be placed on (or moved to) any non-input vertex only if all its immediate predecessors carry pebbles.
- (Pebble Deletion) A pebble can be removed at any time.
- (Goal) Each output vertex must be pebbled at least once.

Placement of a pebble on an input vertex models the reading of input data. Placement of a pebble on a non-input vertex corresponds to computing the value associated with the vertex. The removal of a pebble models the erasure or overwriting of the value associated with the vertex on which the pebble resides.

Allowing pebbles to be placed on input vertices at any time reflects the assumption that inputs are readily available. (The multi-level pebble game introduced in the next chapter models the case in which each access to secondary storage is expensive.) The condition that all predecessor vertices carry pebbles when a pebble is placed on a vertex models the natural requirement that an operation can be performed only after all arguments of the operation are located in main memory. Moving (or **sliding**) a pebble to a vertex from an immediate predecessor reflects the design of CPUs that allow the result of a computation to be placed in a memory location holding an operand.

A **pebbling strategy** is the execution of the rules of the pebble game on the vertices of a graph. We assign a step to each placement of a pebble, ignoring steps on which pebbles are removed, and number the steps consecutively from 1 to T, the **time** or number of steps in the strategy. The **space**, S, used by a pebbling strategy is the maximum number of pebbles it uses. The goal of the pebble game is to pebble a graph with values of space and time that are minimal; that is, the space cannot be reduced for the given value of time and vice versa. In general, it is not possible to minimize space and time simultaneously. We derive upper and lower bounds on the possible exchanges of space for time.

10.1.1 The Pebble Game Versus the Branching Program

As stated above, the branching program model introduced in Section 10.9 handles data-dependent computation, and is thus a more general model than the pebble game. However, there are three reasons to study the pebble game. First, the branching program assumes that

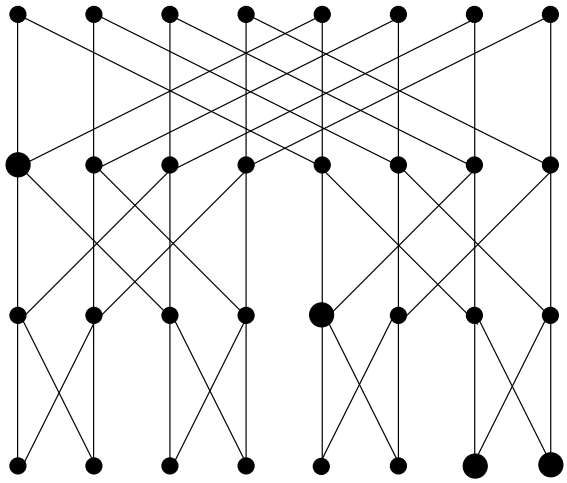


Figure 10.1 An FFT graph $F^{(3)}$ on $n = 2^3$ inputs. Input vertices are on the bottom; edges are directed upward. Four pebbles are shown on the graph when pebbling the leftmost output.

input variables are held in an auxiliary random-access machine so that it can access them in arbitrary order, a condition not imposed on pebble games. It follows that inputs to a pebble game can be fetched in advance, since the times at which they are needed are data-independent. Second, lower bounds on the exchange of space for time with branching programs are harder to obtain due to their increased flexibility. Third, straight-line programs are used in many problems, such as integer multiplication, convolution, matrix multiplication, and discrete Fourier transform, and the pebble game gives the relevant lower bounds. For other problems, such as sorting and merging, the branching program model is the model of choice since these problems are typically solved with branching programs. We expand upon this topic in Section 10.9.1.

10.1.2 Playing the Pebble Game

The pebble game is illustrated in Fig. 10.1 by pebbling the FFT graph $F^{(3)}$ with eight inputs and 24 non-input vertices. This graph has the property that the set of paths from input vertices to an output vertex forms a complete balanced binary tree. (See Fig. 10.2.) It follows that we can pebble the FFT graph by pebbling each of the trees. Since two of the eight outputs share the same tree at the next lower level, we can pebble two outputs at the same time.

Binary trees form an important class of graphs. A **complete balanced binary tree** of depth 4 is illustrated in Fig. 10.2. (The depth of a directed tree is the number of edges on the longest path from an input vertex to the output (or root) vertex.) This tree has 16 input vertices and one output vertex. A complete balanced binary tree of depth 0, $T(0)$, consists of a single vertex. A complete balanced binary tree of depth $d > 0$, $T(d)$, consists of a root vertex and two copies of $T(d-1)$ whose root vertices each have one edge directed from them to the root vertex of the full tree. Thus in Fig. 10.2 the complete balanced binary tree of depth four $T(4)$ is constructed of two copies of $T(3)$, which in turn are each constructed of two copies of $T(2)$, and so on. It follows by straightforward induction that a complete balanced binary tree of depth d has 2^d inputs and $2^{d+1} - 1$ vertices. (See Problem 10.8.)

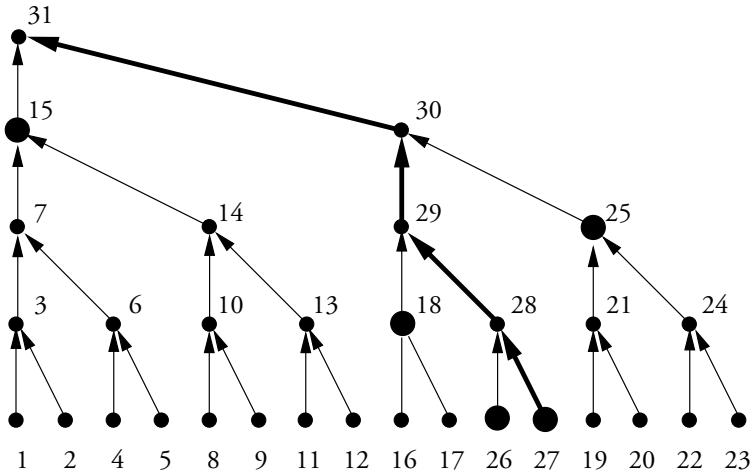


Figure 10.2 A complete balanced binary tree $T(4)$ of depth 4 on 16 inputs. At least five pebbles are needed to pebble it.

The binary tree of Fig. 10.2 can be pebbled with five pebbles by pebbling the vertices in the order shown. Five pebbles are needed at the time when vertex 27 is pebbled. After one pebble is moved to vertex 30, the two outputs of the FFT of Fig. 10.1 to which vertices 15 and 30 are attached can be pebbled. This tree-pebbling strategy can be repeated on all remaining outputs. It is a general strategy for pebbling complete balanced binary trees.

This pebbling strategy, explained in detail in the next section, demonstrates that an FFT graph on $n = 2^k$ inputs can be pebbled with no more pebbles than are needed to pebble the trees with n leaves contained within it, namely, $k + 1$. In the next section we show that this is the minimum number of pebbles needed to pebble a complete balanced binary tree on 2^k leaves. This FFT pebbling strategy for the graph in Fig. 10.1 pebbles each vertex on the third and fourth levels once, each vertex on the second level twice, and each vertex on the first level four times. It is clear that inputs must be repebbled if the minimum number of pebbles is used. This is an example of space–time tradeoff. We shall derive a lower bound on the exchange of space for time for this problem.

In the next section we also examine the minimum space required to pebble graphs. In the subsequent section we describe a graph that exhibits an extreme tradeoff. This graph requires a pebbling time exponential in the size of the graph when the minimum number of pebbles is used but can be pebbled with one move per vertex if one more pebble is available.

After studying extreme tradeoffs we define a flow property of functions that, if satisfied, implies a lower bound on the product $(S+1)T$ (or a related expression) involving the space S and time T needed to compute such functions. This test is used to show that many standard algorithms are optimal with respect to their use of space and time.

10.2 Space Lower Bounds

In this section we derive lower bounds on the **minimum space** $S_{\min}(G)$ needed to pebble a graph G for balanced binary trees, pyramids, and FFT graphs, a representative set of graphs.

Any pebbling strategy will need to use at least as many pebbles as this minimum value of space. It can be shown that no bounded-degree graph on n vertices requires more than $O(n/\log n)$ space (see Theorem 10.7.1) and that some graph requires space proportional to $n/\log n$ (see Theorem 10.8.1).

Complete balanced binary trees were introduced in the previous section. We now derive a lower bound on the space (number of pebbles) needed to pebble them.

LEMMA 10.2.1 *Any pebbling strategy for the complete balanced binary tree of depth k, $T(k)$, requires at least $S_{\min}(T(k)) = k+1$ pebbles and $2^{k+1}-1$ steps. There is a pebbling strategy of $T(k)$ that uses exactly this many pebbles and steps.*

Proof Proof of the lemma requires a proof that $k+1$ pebbles are necessary as well as a strategy that pebbles the tree with $k+1$ pebbles and makes one pebble placement per vertex. Let's first develop a pebbling strategy.

$T(0)$ obviously can be pebbled with one pebble in one step. Assume that $T(k-1)$ can be pebbled with k pebbles in 2^k-1 steps. To pebble $T(k)$, advance a pebble to the root of its left subtree (a copy of $T(k-1)$) using k pebbles and 2^k-1 steps. Leave a pebble on its root. Then pebble the right subtree of $T(k)$ using k pebbles and 2^k-1 steps. (A snapshot of $T(k)$ when the number of pebbles is maximal under this pebbling strategy is shown in Fig. 10.2.) Thus, $T(k)$ is pebbled in $2\times(2^k-1)+1 = 2^{k+1}-1$ steps with $k+1$ pebbles.

The lower bound is derived by showing that no pebbling strategy can use fewer than $k+1$ pebbles. The argument used is the following: initially no path to the root of the tree (or output) from input vertices carries a pebble because there are no pebbles on the graph. At the end of the computation a pebble resides on the root and all paths to the root carry pebbles. Therefore, there must be a first point in time at which there is a pebble on each path to the root. This must be a time at which a pebble is placed on an input vertex, thereby closing the last path from that input to the root. Such a path is highlighted in Fig. 10.2. Before a pebble is placed on the input vertex of this path, all other paths from input vertices to the root carry pebbles. Each of these paths enters the highlighted path via one edge. Thus, it follows that prior to the placement of this last pebble there is at least one pebble on the tree for each of the k edges on this path except for the input vertex. Consequently, at least $k+1$ pebbles are on the tree when the last pebble is placed on it. ■

The FFT graph on 2^k inputs, $F^{(k)}$, is defined recursively in terms of two sub-FFT graphs $F^{(k-1)}$ as shown in Section 6.7.2. It follows that this graph contains many copies of the tree $T(k)$ as a subgraph (see Problem 10.11) and that any pebbling strategy for $F^{(k)}$ requires at least $k+1$ pebbles. Many other straight-line computations involve tree computations.

A **pyramid graph** on m inputs, $P(m)$ ($P(6)$ is shown in Fig. 10.3), is obtained by slicing an $m\times m$ mesh into two parts along its diagonal, splitting all diagonal nodes (which are now inputs), and then directing edges from the diagonal vertices in one part to the one remaining unsplit corner vertex in this part of the graph. Edges are directed up, a convention we use throughout this chapter. $P(m)$ has $n = m(m+1)/2$ vertices. (See Problem 10.1.)

We apply to the pyramid graph $P(m)$ the lower bounding argument used in the preceding proof based on closing the last open path to the output vertex.

LEMMA 10.2.2 *Any pebbling strategy for the m-input, n-vertex ($n = m(m+1)/2$) pyramid graph $P(m)$ requires at least m pebbles; that is, a minimum space $S_{\min}(P(m)) = m \geq \sqrt{2n} -$*

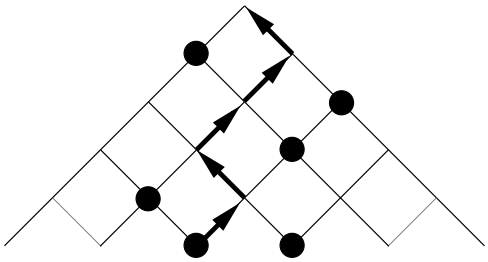


Figure 10.3 The pyramid graph on six inputs.

1. *There exists a pebbling strategy that pebbles $P(m)$ with m pebbles using one pebble placement per vertex.*

Proof The lower-bound proof again uses the fact that there is a first time at which all paths from an input to the output carry pebbles. Highlighted in Fig. 10.3 is a last path to carry a pebble. Prior to the placement of this last pebble, all paths to the output carry pebbles. Thus, with the placement of the last pebble there must be at least as many pebbles on the pyramid graph as there are vertices on a path from an input to the output, namely, m, and $m \geq \sqrt{2n} - 1$. (See Problem 10.1.)

With m pebbles, the vertices can be pebbled in levels by first placing pebbles on each of the m inputs. Pebbles are then advanced to vertices on the second level from left to right, and this process is repeated at all levels to complete the pebbling. Each vertex is pebbled once with this strategy. ■

In general, it is very hard to determine the minimum number of pebbles needed to pebble a graph. In terms of the complexity classes introduced in Chapter 8, we model this problem as a language consisting of strings each of which contains the description of a graph $G = (V, E)$, a vertex $v \in V$, and an integer S with the property that the vertex can be pebbled with S or fewer pebbles. The language of these strings is **PSPACE**-complete (see Section 8.12).

10.3 Extreme Tradeoffs

We now show that extreme space–time tradeoff behavior is possible. We do this by exhibiting a family of graphs, $H_1, H_2, \ldots, H_k, \ldots$ (Fig. 10.4), that requires a number of steps exponential in the size of the graph when the minimum number of pebbles is used but only one step per vertex when one more pebble is available. This illustrates that excessive minimization of the number of registers used by programs can be harmful!

H_1 has one input and one output vertex and an edge connecting them, as shown in Fig. 10.4. For $k \geq 2$ the kth graph, H_k, has $k + 1$ output vertices and is constructed from one copy of H_{k-1}, a tree (on the left) with k inputs, a two-level bipartite graph (on the top right) with k inputs and $k + 1$ outputs, and a chain of k vertices that connects the tree to the outputs of H_{k-1} and the open vertex. (A **bipartite graph** is a graph in which the vertices are partitioned into two sets and edges join vertices in different sets.)

We summarize our pebbling results for this family of graphs below. Here $n!$ is the factorial function with value $n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1$.

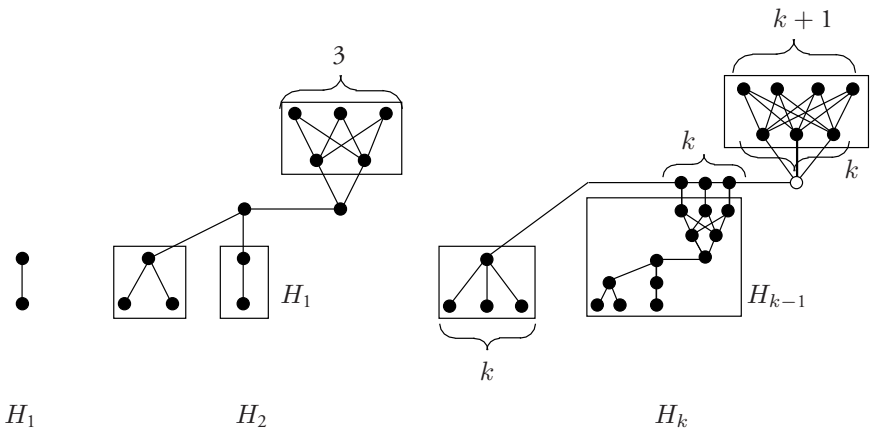


Figure 10.4 A family of graphs exhibiting an extreme tradeoff.

THEOREM 10.3.1 *The graph H_k has $N(k) = 2k^2 + 5k - 6$ vertices for $k \geq 2$. Any pebbling strategy for the graph H_k requires at least k pebbles, $k = \Theta(\sqrt{N(k)})$. Any strategy to pebble H_k with k pebbles requires at least $(k+1)!/2 = 2^{\Omega\left(\sqrt{N(k)}\log N(k)\right)}$ steps, whereas there exists a pebbling algorithm using $k+1$ pebbles that pebbles each vertex of H_k once.*

Proof Consider a pebbling strategy that uses $k+1$ pebbles to pebble H_k. For the case of $k = 1$, H_k can be completely pebbled with one move per vertex. This is also true for H_2 because we can move a pebble to the open vertex connected to the bipartite graph using two pebbles, from which we can advance two of our three pebbles to the bottom layer of the bipartite graph and have one additional pebble with which to pebble the output vertices. Note that this pebbling strategy allows us to pebble output vertices of H_2 from left to right with three pebbles.

Assume that we can pebble the outputs of H_{k-1} from left to right with k pebbles without pebbling any vertex more than once. Then to pebble H_k, advance a pebble to the root of the tree on the left and then pebble the outputs of H_{k-1} from left to right using k pebbles while keeping one additional pebble on the chain. Advance this pebble along the chain until it reaches the open vertex. At this point k pebbles can be advanced to the bottom row of vertices in the bipartite graph and the remaining pebble used to pebble outputs from left to right. This shows that our assumption holds.

The minimum number of pebbles needed to pebble H_k is at least k because at least this many are needed to pebble the tree on the left. To show that this value can be achieved, we give a recursive pebbling strategy. Observe that H_1 can be pebbled with $k = 1$ pebbles. To pebble H_k, assume that we can pebble any one output of H_{k-1} with $k-1$ pebbles. Advance a pebble to the root of the left tree and then advance it along the chain by pebbling output vertices of H_{k-1} from left to right with $k-1$ pebbles. Move a pebble to the open vertex and then to all vertices on one side of the bipartite graph. Any one output vertex can now be pebbled. However, doing so requires that one vertex on the bottom side of the bipartite graph lose its pebble. Thus, no other output vertex can be pebbled without repebbling the tree and all vertices of H_{k-1}.

As this pebbling strategy demonstrates, to pebble an output vertex, all k pebbles must move to the bottom of the bipartite graph, thereby removing all pebbles from other vertices of H_k. Let $M(k)$ be the number of pebble placements to pebble H_k with k pebbles. It follows that to pebble each of the $(k+1)$ outputs of H_k with k pebbles, we must pebble each output of H_{k-1} with $k-1$ pebbles. Thus,

$$\begin{aligned} M(k) &\geq (k+1) \times M(k-1) \\ &\geq (k+1)k(k-1)\cdots 3 \cdot 1 = (k+1)!/2 \end{aligned}$$

which provides the desired lower bound.

Let the graph H_k have $N(k)$ vertices. Then $N(1) = 2$, $N(2) = 12$ and $N(k) = N(k-1) + 4k + 3$ for $k \geq 3$. A straightforward proof by induction shows that $N(k) = 2k^2 + 5k - 6$ (see Problem 10.13).

To show that $M(k) \geq (k+1)!/2$ is exponential in $N(k) = 2k^2 + 5k - 6$, note that $p! = p \cdot (p-1) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$, which is at least $(p/2)^{(p/2)}$ since each of the first $p/2$ terms is at least $p/2$. Thus, $M(k) \geq .5[(k+1)/2]^{(k+1)/2}$ Also, it is easy to see that $N(k) \leq 3(k+1)^2$ for $k \geq 1$. Since this implies $\sqrt{N(k)/3} \leq (k+1)$, we have that

$$M(k) \geq .5 \left[(\sqrt{N(k)/3})/2\right]^{(\sqrt{N(k)/3})/2}$$

which is exponential in $N(k)$. ■

Many vertices in the graph H_k have a fan-in k. A new family $\{G_k\}$ of graphs with fan-in 2 can be obtained by replacing the tree on the left in H_k with the pyramid graph of Fig. 10.3 and replacing the bipartite graph on the top with a new graph (see Problem 10.14). This new graph exhibits an exponential jump in the time to pebble the graph but at a value of space that is the fourth root of the number of vertices in G_k.

10.4 Grigoriev's Lower-Bound Method

In this section we present a method for developing lower bounds on the exchange of space for time in the pebble game. These lower bounds are typically of the form $(S+1)T = \Omega(n^2)$, where S, T, and n are the space, time, and the size of the input to the problem, and are similar in spirit to those of Theorem 3.6.1. Because they assume a less general model of computation (the pebble game instead of the RAM), lower bounds are easier to derive.

The lower bounds use as a measure the maximum amount of information that can flow from a subset of the inputs to a subset of the outputs, and are much easier to derive than are lower bounds on circuit size for the circuit model. Although the results are stated for straight-line computations, they apply to all "input-output-oblivious" computations by finite-state machines: computations in which inputs are read and outputs produced at times independent of the values of the input variables. (See Problem 10.20.)

10.4.1 Flow Properties of Functions

We start by defining a flow property of functions. (See Fig. 10.5.) A function $f : \mathcal{A}^n \mapsto \mathcal{A}^m$ has a large information flow from input variables in X_1 to output variables in Y_1 if there are values for input variables in $X_0 = X - X_1$ such that many different values can be assumed by

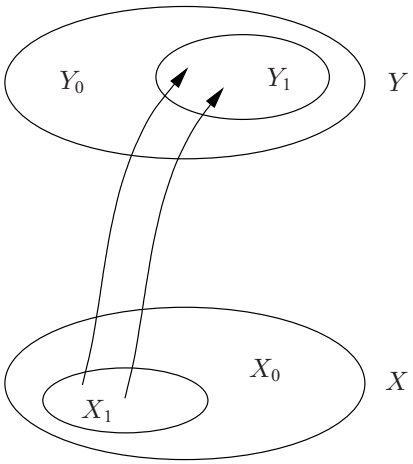


Figure 10.5 A function f that has a large information flow from input variables in X_1 to output variables in Y_1 for some values of input variables in $X_0 = X - X_1$.

outputs in Y_1 as inputs in X_1 range over all their $|\mathcal{A}|^{|X_1|}$ values. This flow property is also used in Section 12.7 to derive lower bounds on the exchange of area for time in the VLSI model of computation.

DEFINITION 10.4.1 *A function $f : \mathcal{A}^n \mapsto \mathcal{A}^m$ has a* $\boldsymbol{w(u,v)}$**-flow** *if for all subsets X_1 and Y_1 of its n input and m output variables, with $|X_1| \geq u$ and $|Y_1| \geq v$, there is a subfunction h of f obtained by making some assignment to variables of f not in X_1 (variables in X_0) and discarding output variables not in Y_1 such that h has at least $|\mathcal{A}|^{w(u,v)}$ points in the image of its domain.*

The exponent function $w(u, v)$ is a nondecreasing function of both of its arguments: increasing u, the number of variables that are allowed to vary, can only increase the number of values assumed by f; the same is true if v is increased.

An important class of functions are the (α, n, m, p)-independent functions defined below.

DEFINITION 10.4.2 *A function $f : \mathcal{A}^n \mapsto \mathcal{A}^m$ is an* $\boldsymbol{(\alpha, n, m, p)}$**-independent function** *for $\alpha \geq 1$ and $p \leq m$ if it has a $w(u, v)$-flow satisfying $w(u, v) > (v/\alpha) - 1$ for $n - u + v \leq p$.*

We illustrate the independence property of a function with matrix multiplication: we show that the function defined by the product of two $n \times n$ matrices is $(1, 2n^2, n^2, n)$-independent. In Section 10.5.4, we show that a stronger property holds for matrix multiplication.

The proof of the independence property of $n \times n$ matrices uses the permutation matrices described in Section 6.2. An $n \times n$ permutation matrix is obtained by permuting either the rows or columns of the $n \times n$ identity matrix. When a permutation matrix B multiplies another matrix A on the right (left) to produce AB (BA), it permutes the columns (rows) of A.

LEMMA 10.4.1 *The matrix multiplication function $f^{(n)}_{A\times B} : \mathcal{R}^{2n^2} \mapsto \mathcal{R}^{n^2}$ over the ring $\mathcal{R}$ is $(1, 2n^2, n^2, n)$-independent.*

Proof Let $C = AB$ be the product of $n \times n$ matrices A and B. Consider any set X_0 of input variables (entries of A and B) and any set Y_1 of output variables (entries of C) such that $|X_0| + |Y_1| = n$. The outputs in Y_1 fall into at most $|Y_1|$ columns of C and the inputs in X_0 fall into at most $|X_0|$ columns of A. It follows that at least $n - |X_0|$ columns of A contain only variables in X_1. Fix the entries in B so that it forms a permutation matrix that permutes the columns of A containing only elements in X_1 onto columns of C containing elements of Y_1. (We are free to make the best assignment of variables in B, whether in X_0 or X_1.) It follows that each output variable in Y_1 is assigned to an input variable of A in X_1 by this permutation. Thus these output variables are free to assume $|\mathcal{R}|^{|Y_1|}$ different values. Since this is more than $|\mathcal{R}|^{|Y_1|-1}$, it follows that $f^{(n)}_{A\times B}$ is $(1, 2n^2, n^2, n)$-independent. ■

As this result illustrates, for any set of y_1 outputs of the matrix multiplication function and any set of x_0 of its inputs satisfying $x_0 + y_1 \leq p$, there is some assignment to these inputs such that there is a large flow of information from the complementary set of inputs, X_1, to any set y_1 of its outputs.

10.4.2 The Lower-Bound Method in the Basic Pebble Game

The following theorem provides a lower bound on the exchange of space for time. Its proof uses a variant of the pigeonhole principle. Since the pebbling of vertices is assumed to occur sequentially, time is divided into intervals in which the number of output vertices pebbled, b, is chosen to be a small multiple of the number of pebbles, S, used in pebbling. The pigeonhole principle is used to show that a large number of inputs must be pebbled in each interval. In particular, we show that if the number of inputs pebbled inside an interval is small, the number of inputs outside the interval is large enough that there is a large flow from the inputs outside the interval to the outputs inside it. However, the flow cannot be any larger than can be supported by the number, S, of vertices carrying pebbles just before the interval. Thus, the number of input variables outside the interval is small, which implies that the number inside is large. That is, many inputs must be pebbled within each interval. Multiplying by the number of intervals in which b outputs are pebbled provides the lower bound.

THEOREM 10.4.1 *Let $f : \mathcal{A}^n \mapsto \mathcal{A}^m$ have an $w(u, v)$-flow and let it be realized by a straight-line program over a basis $\{h : \mathcal{A}^r \mapsto \mathcal{A}^s \mid r, s \geq 1\}$. For arbitrary $b \leq m$, every pebbling of every DAG for f requires space S and time T satisfying the inequality*

$$T \geq \lfloor m/b \rfloor (n - d)$$

where d is the largest integer such that $w(d, b) \leq S$.

Proof Assume that $G = (V, E)$ is pebbled with $S \geq 1$ pebbles in $T \geq 1$ steps. Let $T_I \leq T$ be the number of times that input vertices are pebbled. (This is generally more than the number of input variables.)

Given a pebbling of G with S pebbles, group the consecutive pebbling steps into intervals, the first $\lfloor m/b \rfloor$ of which contain b pebbled outputs and one of which contains $m - b(\lfloor m/b \rfloor)$ pebbled outputs.

Consider an arbitrary interval $\mathcal{I}$ in which b outputs are pebbled. Let Y_1 be these outputs and let x_0 and x_1 be the number of inputs pebbled inside and outside the interval, respectively. By definition, there is an assignment to the x_0 inputs such that that the $b = |Y_1|$

outputs have at least $|\mathcal{A}|^{w(x_1,b)}$ different values. If $w(x_1, b) > S$, the outputs Y_1 assume more values than can be taken by the S pebbles in use just prior to the start of $\mathcal{I}$. Because the values of variables in Y_1 are determined by the inputs pebbled in $\mathcal{I}$, which are fixed, and the values under the S pebbles, this contradicts the definition of f. It follows that x_1 can be no larger than d, where d is the largest value such that $w(d, b) \leq S$. Thus the number of inputs pebbled in $\mathcal{I}$, x_0, satisfies $x_0 \geq (n - d)$.

Since there are $\lfloor m/b \rfloor$ intervals in which b outputs are pebbled, the number of times that inputs are pebbled, T_I, is at least $\lfloor m/b \rfloor (n - d)$. ■

Grigoriev [120] established the above theorem for $(1, n, m, p)$-independent functions. We restate as a corollary a slightly revised version of his theorem for (α, n, m, p)-independent functions.

COROLLARY 10.4.1 *Let $f : \mathcal{A}^n \mapsto \mathcal{A}^m$ be (α, n, m, p)-independent and let it be realized by a straight-line program over a basis $\{h : \mathcal{A}^r \mapsto \mathcal{A}^s \mid r, s \geq 1\}$. Every pebbling of every DAG for f requires space S and time T satisfying the inequality*

$$\lceil \alpha(S + 1) \rceil T \geq mp/4$$

Proof An (α, n, m, p)-independent function on n inputs has a $w(u, v)$-flow satisfying $w(u, v) > (v/\alpha) - 1$ for $n - u + v \leq p$, where $x_0 = n - u \geq 0$. Since b can be freely chosen, let $b = \lceil \alpha(S + 1) \rceil$. Thus, $(b/\alpha) - 1 \geq S$ for $(n - d) + b \leq p$, which contradicts the requirement that $w(d, b) \leq S$. It follows that $(n - d) + b > p$ or that $(n - d) \geq p - \lceil \alpha(S + 1) \rceil$. With the inequality $\lfloor m/x \rfloor \geq (m - x + 1)/x$ (see Problem 10.2), the following lower bound follows from Theorem 10.4.1:

$$T \geq \frac{(m - \lceil \alpha(S + 1) \rceil + 1)(p - \lceil \alpha(S + 1) \rceil)}{\lceil \alpha(S + 1) \rceil}$$

Since $p \leq m$, if $\lceil \alpha(S + 1) \rceil \leq p/2$, the desired lower bound follows. On the other hand, if $\lceil \alpha(S + 1) \rceil \geq p/2$, $\lceil \alpha(S + 1) \rceil T \geq mp/2$ since $T \geq m$. ■

It is possible that a function $f : \mathcal{A}^n \mapsto \mathcal{A}^m$ is not (α, n, m, p)-independent but a subfunction $g : \mathcal{A}^r \mapsto \mathcal{A}^s$ is (α, r, s, p)-independent for $r \leq n$ and $s \leq m$. (Subfunctions are defined in Section 2.4.) As shown in Problem 10.18, the lower bound for the subfunction g applies to f.

Lower bounds on space–time exchanges can also be derived using properties of the graphs to be pebbled. For example, if a graph contains a superconcentrator (defined in Section 10.8), lower bounds on the product can be derived on $(S + 1)T$ in terms of the number of inputs of the graph. (See Problem 10.28.)

As mentioned at the beginning of this section, Theorem 10.4.1 is much more general that it appears. In Problem 10.20 the reader is asked to show that the lower bound holds for "input-output-oblivious" finite-state machines, FSMs that compute functions but read their inputs and produce their outputs at data-independent times. Problem 10.21 asks the reader to establish that pebblings of straight-line computations can be translated directly into computations by finite-state machines.

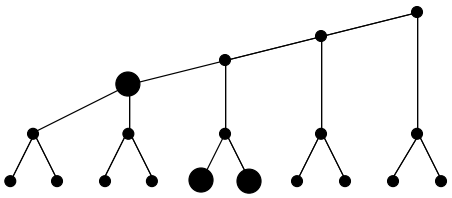


Figure 10.6 Pebbling an inner product graph with three pebbles.

10.4.3 First Matrix Multiplication Bound

The Grigoriev lower-bound method is well illustrated by matrix multiplication. We established its independence property in Section 10.4.1. In this section we apply it to Corollary 10.4.1. The upper bound stated in the following theorem follows from the development of an algorithm for matrix multiplication that uses three pebbles and executes at most $4n^3$ steps. This algorithm, based on the standard matrix multiplication algorithm of Section 6.2.2, forms each of the n^2 inner products defined by the product of two $n \times n$ matrices using three pebbles, as suggested in Fig. 10.6, and $4n - 1$ steps.

THEOREM 10.4.2 *Every pebbling strategy for straight-line programs computing the matrix multiplication function* $f_{A\times B}^{(n)} : \mathcal{B}^{2n^2} \mapsto \mathcal{B}^{n^2}$ *for* $n \times n$ *matrices requires space* S *and time* T *satisfying the following inequality:*

$$(S+1)T \geq n^3/4$$

The standard algorithm for multiplying $n \times n$ *matrices uses space and time satisfying*

$$(S+1)T = 16\,n^3$$

Those familiar with fast non-standard matrix multiplication algorithms such as Strassen's fast matrix algorithm (Section 6.3) may find this result surprising. Whereas one learns that the standard matrix multiplication algorithm is not optimal with respect to computation time, the above result states that the standard matrix multiplication algorithm is nearly optimal with respect to the space–time product.

In Section 10.5.4 we specialize Theorem 10.4.1 to the flow properties of matrix multiplication, giving a stronger result: that the space and time for matrix multiplication must satisfy the inequality $ST^2 = \Omega(n^6)$.

10.5 Applications of Grigoriev's Method

Given the above results, to derive a lower bound on $\lceil \alpha(S+1) \rceil T$ using Corollary 10.4.1 it suffices to establish the independence property of a function. We apply this idea in this section to convolution, cyclic shifting, integer multiplication, matrix-vector multiplication, matrix inversion, and solving linear equations. We apply related arguments to derive lower bounds for the discrete Fourier transform and merging. Finally, we apply Theorem 10.4.1 to derive a lower bound on space–time exchanges for matrix-matrix multiplication that improves upon the bound of Section 10.4.3. Where possible we also derive upper bounds on space–time tradeoffs.

10.5.1 Convolution

The wrapped convolution on strings of length n over the ring $\mathcal{R}$, $f^{(n)}_{\text{wrapped}} : \mathcal{R}^{2n} \mapsto \mathcal{R}^n$, is defined in Problem 6.19. It can be characterized by the following product of a circulant matrix with a vector (see Section 6.2):

$$\begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{n-1} \end{bmatrix} = \begin{bmatrix} u_0 & u_{n-1} & u_{n-2} & \cdots & u_1 \\ u_1 & u_0 & u_{n-1} & \cdots & u_2 \\ & & & \ddots & \\ u_{n-2} & u_{n-3} & u_{n-4} & \cdots & u_{n-1} \\ u_{n-1} & u_{n-2} & u_{n-3} & \cdots & u_0 \end{bmatrix} \times \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{n-1} \end{bmatrix} \tag{10.1}$$

Lemma 10.5.1 demonstrates $(2, 2n, n, n/2)$-independence for the wrapped convolution $f^{(n)}_{\text{wrapped}} : \mathcal{R}^{2n} \mapsto \mathcal{R}^n$ function by showing that for any set X_0 of inputs there is a way to put $|Y_1|/2$ of the inputs in $X - X_0$ into a one-to-one correspondence with $|Y_1|/2$ entries in any set Y_1 of outputs. This is established by setting one component of $\boldsymbol{v}$ to 1 and the rest to 0.

LEMMA 10.5.1 *For n even, the wrapped convolution $f^{(n)}_{\text{wrapped}} : \mathcal{R}^{2n} \mapsto \mathcal{R}^n$ over the ring $\mathcal{R}$ is $(2, 2n, n, n/2)$-independent.*

Proof Consider subsets X_0 and Y_1 of the inputs X and outputs Y of $f^{(n)}_{\text{wrapped}}$ satisfying $|X_0| + |Y_1| = p = n/2$. For $f^{(n)}_{\text{wrapped}}$ to be $(2, 2n, n, n/2)$-independent, there must be an assignment to input variables in X_0 such that the output variables in Y_1 have more than $|\mathcal{R}|^{(|Y_1|/2)-1}$ distinct values as the input variables of $f^{(n)}_{\text{wrapped}}$ in $X_1 = X - X_0$ range over all possible values.

As shown above, $f^{(n)}_{\text{wrapped}}$ is defined by a matrix-vector product $\boldsymbol{w} = M\boldsymbol{v}$, M a circulant matrix, in which each row (column) is a cyclic shift of the first row (column). Let $e = |X_0 \cap \{u_0, u_1, \ldots, u_{n-1}\}|$. Thus, every row of M contains the same number e of entries from X_0. Also, $n - e$ inputs are in $X_1 = X - X_0$. The entries in X_1 are free to vary.

Each output in Y_1 corresponds to a row of M. The number of instances of input variables from X_1 in these rows is $|Y_1|(n - e)$. Since these rows have n columns, there is some column, say the tth, containing at least the average number of instances from X_1. This average is $|Y_1|(1 - e/n) \geq |Y_1|/2$. (The instances of variables from X_1 in a column are distinct.) It follows that by choosing the tth component of $\boldsymbol{v}$, v_t, to be 1 and the others to be 0, at least $|Y_1|/2$ of the inputs in X_1 are mapped onto outputs in Y_1. Since these inputs (and outputs) can assume $|\mathcal{R}|^{|Y_1|/2}$ different values, it follows that $f^{(n)}_{\text{wrapped}}$ is $(2, 2n, n, n/2)$-independent. ■

This implies the lower bound stated below. The upper bound follows from the standard matrix-vector algorithm for the wrapped convolution using the observation that an inner product can be done with three pebbles, as suggested in Fig. 10.6.

THEOREM 10.5.1 *The time T and space S required to pebble any straight-line program for the standard or wrapped convolution must satisfy the following inequality:*

$$(S + 1)T \geq n^2/16$$

This lower bound can be achieved to within a constant multiplicative factor for $S = O(1)$.

10.5.2 Cyclic Shifting

The **cyclic shifting function** $f^{(n)}_{\text{cyclic}} : \mathcal{B}^{n+\lceil \log n \rceil} \mapsto \mathcal{B}^n$ defined in Section 2.5.2 is a subfunction of many functions, including integer multiplication and squaring (see Section 2.9.5), integer reciprocal (see Section 2.10.1), and powers of integers (see Problems 2.34 and 2.35).

Cyclic shifting is another good example of a problem for which a lower bound on the exchange of space and time exists. The method used to establish the independence properties of this function can be generalized to the class of transitive functions. (See Problem 10.22.)

We redefine $f^{(n)}_{\text{cyclic}}$ here. Let $k = \lceil \log n \rceil$. The input variables of $f^{(n)}_{\text{cyclic}}$ are segmented into two groups, an n-tuple $\boldsymbol{x} = (x_{n-1}, \ldots, x_1, x_0)$ of **value variables** and a k-tuple $\boldsymbol{s} = (s_{k-1}, \ldots, s_1, s_0)$ of **control variables**. The control variables specify the integer $|\boldsymbol{s}|$:

$$|\boldsymbol{s}| = s_{k-1}2^{k-1} + \cdots + s_1 2^1 + s_0$$

$|\boldsymbol{s}|$ is the number of places by which the value inputs must be shifted left cyclically to produce the output n-tuple $\boldsymbol{y} = (y_{n-1}, \ldots, y_1, y_0)$. That is, $f^{(n)}_{\text{cyclic}}(\boldsymbol{x}, \boldsymbol{s}) = (\boldsymbol{y})$, where

$$y_j = x_{(j-|\boldsymbol{s}|) \bmod n} \text{ for } 0 \leq j \leq \lceil \log n \rceil - 1 \tag{10.2}$$

A circuit to implement $f^{(n)}_{\text{cyclic}}$ is given in Section 2.5.2 that cyclically shifts $\boldsymbol{x}$ left by 2^j places for each of those values of j, $0 \leq j \leq \lceil \log n \rceil - 1$, such that $s_j = 1$.

The independence properties of the cyclic function are shown by demonstrating that some permutation of the input vector $\boldsymbol{x}$ aligns unselected inputs with selected outputs.

LEMMA 10.5.2 *$f^{(n)}_{\text{cyclic}} : \mathcal{B}^{n+\lceil \log n \rceil} \mapsto \mathcal{B}^n$ is $(2, n + \lceil \log n \rceil, n, n/2)$-independent.*

Proof Consider subsets X_0 and Y_1 of the inputs X and outputs Y of $f^{(n)}_{\text{cyclic}}$ satisfying $|X_0| + |Y_1| = p = n/2$. For $f^{(n)}_{\text{cyclic}}$ to be $(2, n + \lceil \log n \rceil, n, n/2)$-independent, there must be an assignment to input variables in X_0 such that the output variables in Y_1 have more than $|\mathcal{B}|^{(|Y_1|/2)-1}$ distinct values as the input variables of $f^{(n)}_{\text{cyclic}}$ in $X_1 = X - X_0$ range over all possible values.

Let X_0 contain e elements from $\boldsymbol{x}$. Let $y_i \in Y_1$. As $\boldsymbol{s}$ runs through all possible shift values, y_i is made equal to every one of the inputs in $\boldsymbol{x}$. For $n - e$ of these shifts y_i is set equal to an input in $X_1 = X - X_0$. (For example, if $n = 6$ and $e = 2$, say with $X_1 = \{x_0, x_3, x_4, x_5\}$ and $Y_1 = \{y_2, y_3, y_5\}$, then as $\boldsymbol{s}$ ranges over all of its values, each of the three y_i in Y_1 is assigned four different variables in X_1.) Thus, the number of input variables assigned to outputs, summed over all cyclic shifts, is $|Y_1|(n - e)$. Since there are n cyclic shifts, for some shift the number of variables in X_1 that are matched with outputs in Y_1 is at least the average of this quantity; that is, at least $|Y_1|(1 - e/n) \geq |Y_1|/2$. Thus, some shift sets at least $|Y_1|/2$ inputs in X_1 to outputs in Y_1. Since these outputs can assume $|\mathcal{B}|^{|Y_1|/2}$ different values, it follows that $f^{(n)}_{\text{cyclic}}$ is $(2, n + \lceil \log n \rceil, n, n/2)$-independent. ■

THEOREM 10.5.2 *Every pebbling strategy for straight-line programs computing the cyclic shifting function $f^{(n)}_{\text{cyclic}} : \mathcal{B}^{n+\lceil \log n \rceil} \mapsto \mathcal{B}^n$ requires space S and time T satisfying the inequality*

$$(S + 1)T \geq n^2/16$$

An algorithm exists to compute $f^{(n)}_{\text{cyclic}}$ *that uses space* $O(n)$ *and time* $O(n \log n)$*, namely, that satisfies the inequality*

$$(S+1)T = O(n^2 \log n)$$

Proof We leave the upper-bound proof to the reader. (See Problem 10.30.) ■

We now apply this result to integer multiplication.

10.5.3 Integer Multiplication

To apply Grigoriev's method to the binary integer multiplication function $f^{(n)}_{\text{mult}} : \mathcal{B}^{2n} \mapsto \mathcal{B}^{2n}$ of Section 2.9, we assemble a collection of results to show that with the proper encoding of one of its two arguments, $f^{(n)}_{\text{mult}}$ computes the logical shifting function $f^{(n)}_{\text{shift}}$ (see Lemma 2.9.1) and when n is even the logical shifting function $f^{(n)}_{\text{shift}}$ contains the cyclic shift function $f^{(n/2)}_{\text{cyclic}}$ as a subfunction (see Lemma 2.5.2). Thus, $f^{(n)}_{\text{mult}}$ contains $f^{(n/2)}_{\text{cyclic}}$ as a subfunction. We use this fact to obtain a lower bound on the space–time product for integer multiplication.

THEOREM 10.5.3 *Let* n *be even. Every pebbling strategy for straight-line programs computing the binary integer multiplication function* $f^{(n)}_{\text{mult}} : \mathcal{B}^{2n} \mapsto \mathcal{B}^{2n}$ *requires space* S *and time* T *satisfying the following inequality:*

$$(S+1)T \geq n^2/64$$

An algorithm exists for multiplying n*-bit integers using space* $O(\log^2 n)$ *and time* $O(n^2)$*, namely, that satisfies*

$$(S+1)T = O(n^2 \log^2 n)$$

Proof The lower-bound argument is given above. The upper bound follows from a pebbling of an integer multiplication circuit to multiply n-bit binary integers $\boldsymbol{u}$ and $\boldsymbol{v}$. The circuit is based on the following standard expansion of their product:

$$\begin{array}{cccccc}
 & & v_3u_0 & v_2u_0 & v_1u_0 & v_0u_0 \\
 & v_3u_1 & v_2u_1 & v_1u_1 & v_0u_1 & 0 \\
v_3u_2 & v_2u_2 & v_1u_2 & v_0u_2 & 0 & 0 \\
v_3u_3 & v_2u_3 & v_1u_3 & v_0u_3 & 0 & 0
\end{array}$$

To construct a circuit we use the observation that the number of 1s in the jth column is the jth component, w_j, of the convolution $\boldsymbol{w} = \boldsymbol{u} \otimes \boldsymbol{v}$. (See Section 6.7.4.)

To compute w_j we use the counting circuit $f^{(n)}_{\text{count}} : \mathcal{B}^n \mapsto \mathcal{B}^{\lceil \log n \rceil}$ of Section 2.11 on n inputs to count the number of 1s among the products $u_r v_s$ of the Boolean variables u_r and v_s in the sum

$$w_j = \sum_{r+s=j} u_r * v_s \quad \text{for } 0 \leq j \leq 2n-2$$

To compute the $2n$-bit product we add the binary representations for $w_0, w_1, \ldots, w_{2n-2}$ in a set of $(2n-1)$ ripple adders, adding w_j to the sum $\sigma(j) = \sum_{0 \leq i \leq j-1} w_i 2^i$, as suggested in Fig. 10.7, where we omit the counting circuits used to compute the values of $w_0, \ldots, w_{2n-2}$.

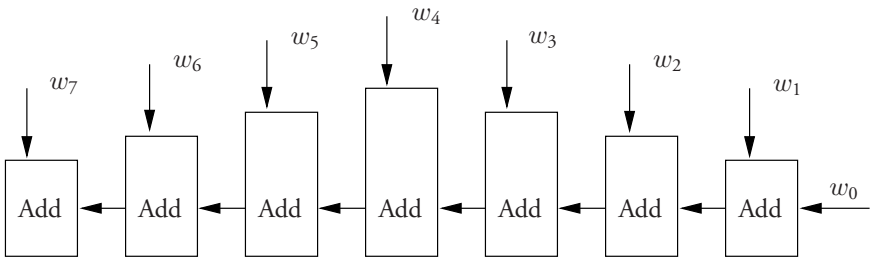


Figure 10.7 A multiplication circuit that can be pebbled in $O(n^2)$ time and $O(\log^2 n)$ space. The counting circuits that generate $w_0, w_1, \ldots, w_{2n-2}$ are not shown.

Each counting function can be pebbled with $O(n)$ steps using $O(\log^2 n)$ pebbles without repebbling vertices. (See Problem 10.10.) After the counting circuit is pebbled, pebbles remain on their outputs until their values have been used elsewhere in the multiplication circuit.

The value of w_j is represented by a k-tuple, $k \leq \lceil \log_2 n \rceil$. The value of $\sigma(j)$ is represented by at most $\lceil \log_2(n(2^j - 1)) \rceil \leq j + \lceil \log_2 n \rceil$ bits since it is the sum of at most n j-bit binary numbers. Because w_j is added after the first j bits, the pebbles on these bits can be discarded. Only $\lceil \log_2 n \rceil$ bits of the running sum and a like number for w_j are needed to hold values on the inputs to the ripple adder. A fixed additional number of pebbles suffices to pebble the internal vertices of the adder. On completion of the sum only $\lceil \log_2 n \rceil$ pebbles are needed. They are used to hold the portion of the running sum that is used in the next stage of addition.

For each value of j, $0 \leq j \leq 2(n-1)$, $O(\log n)$ steps are executed in the ripple adder and $O(n)$ steps are executed in a counting circuit. Consequently, $O(\log^2 n)$ pebbles and $O(n^2)$ time suffice to compute the product of n-bit binary numbers. ■

In Section 10.13.2 we show that a lower bound of $\Omega(n^2/\log^2 n)$ applies under the branching program model. The stronger lower bound of $\Omega(n^2)$ derived here reflects the extra constraints imposed on the pebble game, namely that inputs are read and computations performed at data-independent times.

Similar results apply to the squaring function $f^{(n)}_{\text{square}}$ since, as shown in Lemma 2.9.2, $f^{(3n+1)}_{\text{square}}$ contains $f^{(n)}_{\text{mult}}$ as a subfunction. (See Problem 10.32.)

Similar results also apply to the reciprocal function $f^{(n)}_{\text{recip}} : \mathcal{B}^n \mapsto \mathcal{B}^n$ since, as shown in Lemma 2.10.1, $f^{(n)}_{\text{recip}}$ contains as a subfunction the squaring function $f^{(m)}_{\text{square}}$ for $m = \lfloor n/12 \rfloor - 1$. (See Problem 10.33.)

10.5.4 Matrix Multiplication

In this section we show that the matrix multiplication function is richer than the other functions examined above in that it exhibits a stronger space–time lower bound than given in Theorem 10.4.2. After we derive a lower bound on the function $w(u, v)$ we specialize Theorem 10.4.1 to this case, thereby deriving the stronger lower bound.

LEMMA 10.5.3 *The matrix multiplication function $f^{(n)}_{A\times B} : \mathcal{R}^{2n^2} \mapsto \mathcal{R}^{n^2}$ over the ring $\mathcal{R}$ has a $w(u,v)$-flow, where $w(u,v)$ satisfies the following lower bound:*

$$w(u,v) \geq (v - (2n^2 - u)^2/4n^2)/2$$

Proof Let $C = AB$ be the product of $n \times n$ matrices A and B. We establish this result by using characteristic functions to identify the outputs in C in Y_1 and the inputs in A and B in X_1, as indicated below. Here the indices i and j range over $0 \leq i, j \leq n-1$:

$$\sigma_{i,j} = \begin{cases} 1 & c_{i,j} \in Y_1 \\ 0 & \text{otherwise} \end{cases} \qquad \alpha_{i,j} = \begin{cases} 1 & a_{i,j} \in X_1 \\ 0 & \text{otherwise} \end{cases}$$

$$\beta_{i,j} = \begin{cases} 1 & b_{i,j} \in X_1 \\ 0 & \text{otherwise} \end{cases}$$

Let $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ denote the matrices $[\alpha_{i,j}]$, $[\beta_{i,j}]$, and $[\sigma_{i,j}]$, respectively. Denote by $|\boldsymbol{A}|$, $|\boldsymbol{B}|$, and $|\boldsymbol{C}|$ the number of 1s in the three corresponding matrices. Note that $|\boldsymbol{A}| + |\boldsymbol{B}| = |X_1|$ and $|\boldsymbol{C}| = |Y_1|$.

The kth $n \times n$ **cyclic permutation matrix** $P(k)$ is the $n \times n$ identity matrix in which the rows are rotated cyclically $k-1$ times. For example, the following 3×3 matrix is $P(3)$.

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Let D be an $n \times n$ matrix. The matrix $P(k)D$ consists of the rows of D shifted cyclically down $k-1$ places. Similarly, the matrix $DP(k)$ consists of the columns of D shifted cyclically left $k-1$ places.

Let $\boldsymbol{B}(k)$ be the matrix $\boldsymbol{B}$ obtained by multiplication on the left by $A = P(k)$. Similarly, let $\boldsymbol{A}(k)$ be the matrix $\boldsymbol{A}$ obtained by multiplication on the right by $B = P(k)$. Then, a 1 value for the (i,j) entry in $\boldsymbol{A}(k)$ and $\boldsymbol{B}(k)$ identifies a variable in X_1 that is mapped to an output variable of C through its multiplication by $P(k)$.

Let $\boldsymbol{D}$ and $\boldsymbol{E}$ be $n \times n$ matrices whose entries are drawn from the set $\{0,1\}$. We denote by $\boldsymbol{D} \cap \boldsymbol{E}$ the $n \times n$ matrix whose (i,j) entry is 1 if $d_{i,j} = e_{i,j} = 1$. Similarly, let $\boldsymbol{D} \cup \boldsymbol{E}$ be the $n \times n$ matrix whose (i,j) entry is 1 if either $d_{i,j} = 1$ or $e_{i,j} = 1$. The following identity applies:

$$|\boldsymbol{D} \cup \boldsymbol{E}| + |\boldsymbol{D} \cap \boldsymbol{E}| = |\boldsymbol{D}| + |\boldsymbol{E}| \tag{10.3}$$

Since $|\boldsymbol{D} \cup \boldsymbol{E}| \leq n^2$ for $n \times n$ matrices, the following inequality holds:

$$|\boldsymbol{D} \cap \boldsymbol{E}| \geq |\boldsymbol{D}| + |\boldsymbol{E}| - n^2 \tag{10.4}$$

Also, since $|\boldsymbol{D} \cap \boldsymbol{E}| \geq 0$ we have

$$|\boldsymbol{D}| + |\boldsymbol{E}| \geq |\boldsymbol{D} \cup \boldsymbol{E}| \tag{10.5}$$

The $w(u,v)$-flow of matrix multiplication is large if for some choice of r or s $|\boldsymbol{C} \cap \boldsymbol{A}(r)|$ or $|\boldsymbol{C} \cap \boldsymbol{B}(s)|$ is large. This follows because choosing A to be the rth cyclic permutation

makes many variables of B in X_1 match entries in C in Y_1, or choosing B to be the sth cyclic permutation makes many variables of A in X_1 match entries in C in Y_1. When an input and output variable match, the latter assumes the value of the former. Thus, all the variation in the former is reflected in the latter.

Let $Q = |\boldsymbol{C} \cap \boldsymbol{A}(r)| + |\boldsymbol{C} \cap \boldsymbol{B}(s)|$. Then the $w(u, v)$-flow is at least $Q/2$. Applying (10.5) and then (10.4) to Q, we have the following inequalities:

$$Q \geq |\boldsymbol{C} \cap (\boldsymbol{A}(r) \cup \boldsymbol{B}(s))\,| \geq |\boldsymbol{C}| + |\boldsymbol{A}(r) \cup \boldsymbol{B}(s)| - n^2$$

Applying (10.3) to $|\boldsymbol{A}(r) \cup \boldsymbol{B}(s)|$ yields the following lower bound on Q:

$$Q \geq |\boldsymbol{C}| + |\boldsymbol{A}(r)| + |\boldsymbol{B}(s)| - |\boldsymbol{A}(r) \cap \boldsymbol{B}(s)| - n^2 \tag{10.6}$$

But $|\boldsymbol{C}| = |Y_1|$, $|\boldsymbol{A}(r)| = |\boldsymbol{A}|$, $|\boldsymbol{B}(s)| = |\boldsymbol{B}|$, and $|\boldsymbol{A}| + |\boldsymbol{B}| = |X_1|$. We now show that there are values for r and s such that $|\boldsymbol{A}(r) \cap \boldsymbol{B}(s)|$ is at most $|\boldsymbol{A}||\boldsymbol{B}|/n^2$.

Consider the following sum:

$$S = \sum_{r=1}^{n} \sum_{s=1}^{n} |\boldsymbol{A}(r) \cap \boldsymbol{B}(s)|$$

Since $\boldsymbol{A}(r)$ and $\boldsymbol{B}(s)$ are formed by the rth and sth cyclic shift of columns of $\boldsymbol{A}$ and rows of $\boldsymbol{B}$ respectively, each 1 in $\boldsymbol{A}$ is aligned once with each 1 in $\boldsymbol{B}$. It follows that

$$S = |\boldsymbol{A}||\boldsymbol{B}|$$

As a consequence, there are some r and s such that $|\boldsymbol{A}(r) \cap \boldsymbol{B}(s)|$ is at most S/n^2. Applying this result in (10.6), we have the following lower bound on Q:

$$Q \geq |Y_1| + |\boldsymbol{A}| + |\boldsymbol{B}| - |\boldsymbol{A}||\boldsymbol{B}|/n^2 - n^2$$

Since $|X_1| = |\boldsymbol{A}| + |\boldsymbol{B}|$ is fixed, the above lower bound on Q is minimized by maximizing $|\boldsymbol{A}||\boldsymbol{B}|$ under variation of $|\boldsymbol{A}|$. This maximum occurs when $|\boldsymbol{A}| = |X_1|/2$. Consequently we have the following lower bound on Q:

$$Q \geq |Y_1| - n^2 \left(1 - \frac{|X_1|}{2n^2}\right)^2$$

Since $w(u, v) \geq Q/2$ for $u = |X_1|$ and $v = |Y_1|$, we have desired the conclusion. ■

We now apply this result and Theorem 10.4.1 to derive a stronger result for matrix multiplication than was obtained earlier using its $(1, 2n^2, n^2, n)$-independence property.

THEOREM 10.5.4 *Every pebbling strategy for straight-line programs computing the matrix multiplication function $f^{(n)}_{A\times B} : \mathcal{B}^{2n^2} \mapsto \mathcal{B}^{n^2}$ for $n \times n$ matrices requires space S and time T satisfying the following inequality:*

$$ST^2 \geq n^6/3$$

The standard algorithm for multiplying $n \times n$ matrices uses space and time satisfying

$$ST^2 = 48\, n^6$$

Proof From Lemma 10.5.3 we have that the matrix multiplication function has a $w(u,v)$-flow, where

$$w(u,v) \geq (v-(2n^2-u)^2/4n^2)/2$$

Applying Theorem 10.4.1 to this problem with $b=3S$, we seek the largest integer d such that $w(d,b) \leq S$, which must satisfy the bound

$$\left(3S-(2n^2-d)^2/4n^2\right)/2 \leq S$$

This implies that $(2n^2-d) \geq 2n\sqrt{S}$. From Theorem 10.4.1, the time to pebble the graph satisfies

$$\begin{aligned} T &\geq 2\sqrt{S}n\lfloor n^2/3S \rfloor \\ &\geq 2\sqrt{S}n(n^2-3S+1)/3S \end{aligned}$$

If $S \leq n^2/27$, $T \geq (16\sqrt{2}n^3)/(27\sqrt{S})$ or $ST^2 \geq (.35)n^6$. On the other hand, since $T \geq 3n^2$ just to pebble inputs and outputs, if $S > n^2/27$, then $ST^2 \geq n^6/3$. ■

10.5.5 Discrete Fourier Transform

The discrete Fourier transform (DFT) is defined in Section 6.7.3. We derive upper and lower bounds on the space–time product needed to compute this function.

LEMMA 10.5.4 *The n-point DFT function $F_n : \mathcal{R}^n \mapsto \mathcal{R}^n$ over a commutative ring $\mathcal{R}$ is $(2,n,n,n/2)$-independent for n even.*

Proof As shown in equation (6.23), the DFT is defined by the matrix-vector product $[w^{ij}]\boldsymbol{a}$, where $[w^{ij}]$ is a Vandermonde matrix. To show that the DFT function is $(2,n,n,n/2)$-independent, consider any set Y_1 of outputs (corresponding to rows of $[w^{ij}]$) and any set X_0 of inputs (corresponding to columns) whose values are to be fixed judiciously, where $p=|X_0|+|Y_1|=n/2$. We show that the outputs in Y_1 have at least $|\mathcal{R}|^{|Y_1|/2}$ values as we vary over the remaining inputs.

It is straightforward to show that the submatrix of $[w^{ij}]$ defined by any $|Y_1|$ rows and any $|Y_1|$ consecutive columns is non-singular. (Its determinant is that of another Vandermonde matrix. Show this by letting the row and column indices be $r_1, r_2, \ldots, r_{|Y_1|}$ and $s, s+1, \ldots, s+|Y_1|-1$, respectively, and demonstrating that $w^{r_i s}$ can be factored out of the ith row when computing its determinant.) Our goal is to show that some consecutive group of columns corresponds to at least $|Y_1|/2$ inputs of $\boldsymbol{a}$ in X_1.

Divide the n columns of $[w^{ij}]$ into $\lceil n/|Y_1| \rceil$ groups of consecutive columns with $|Y_1|$ inputs in each group except possibly the last, which may have fewer. There are $n-|X_0|$ inputs that may vary. Since there are $\lceil n/|Y_1| \rceil$ groups, by an averaging argument some group contains at least $(n-|X_0|)/\lceil n/|Y_1| \rceil$ of these inputs. Since $\lceil n/|Y_1| \rceil \leq (n+|Y_1|-1)/|Y_1|$, we show that $(n-|X_0|)/\lceil n/|Y_1| \rceil > |Y_1|/2$ for $p=n/2$. Observe that $(n-|X_0|)/(n+|Y_1|-1) \geq 1/2$ if $2n-2|X_0| \geq n+|Y_1|-1$ or $n \geq |X_0|+p-1$, which holds because $|X_0| \leq p \leq n/2$.

Since the submatrix defined by k consecutive columns and any k rows where $\lceil |Y_1|/2 \rceil \leq k \leq |Y_1|$ is non-singular, it follows that any subset of $\lceil |Y_1|/2 \rceil$ columns has full rank. Thus, the submatrix contains a non-singular $\lceil |Y_1|/2 \rceil \times \lceil |Y_1|/2 \rceil$ matrix. When all inputs outside

of these columns are set to zero, the $\lceil |Y_1|/2 \rceil$ outputs have $|\mathcal{R}|^{\lceil |Y_1|/2 \rceil}$ values, or F_n is $(2, n, n, n/2)$-independent. ■

The space–time lower bound stated below follows from Corollary 10.4.1.

THEOREM 10.5.5 *To pebble any straight-line program for the n-point DFT over a commutative ring $\mathcal{R}$ requires space S and time T satisfying the following:*

$$(S+1)T \geq n^2/16$$

when n is even. The FFT graph on $n = 2^d$ inputs can be pebbled with space S and time T satisfying the upper bound

$$T \leq 4n^2/(S - \log_2 n) + n \log_2 S$$

Thus, $(S+1)T = O(n^2)$ when $2 \log_2 n \leq S \leq (n/\log_2 n) + \log_2 n$.

Proof This lower bound can be achieved up to a constant factor by a pebbling strategy for the FFT algorithm, as we now show. Denote with $F^{(d)}$ the n-point FFT graph (it has n inputs), $n = 2^d$. (Figures. 6.1, 6.7, and 10.8 show 4-point, 16-point, and 32-point FFT graphs.) Inputs are at level 0 and outputs are at level d. We invoke Lemma 6.7.4 to decompose $F^{(d)}$ at level $d - e$ into a set of top 2^{d-e} 2^e-point FFT graphs above the split, $\{F^{(e)}_{t,j} \mid 1 \leq j \leq 2^e\}$, and a set of 2^e 2^{d-e}-point FFT graphs below the split, $\{F^{(d-e)}_{b,j} \mid 1 \leq j \leq 2^e\}$, as suggested in Fig. 10.8. In this figure the vertices and edges have been grouped together as recognizable FFT graphs and surrounded by shaded boxes. The edges between boxes identify vertices that are common to pairs of FFT subgraphs.

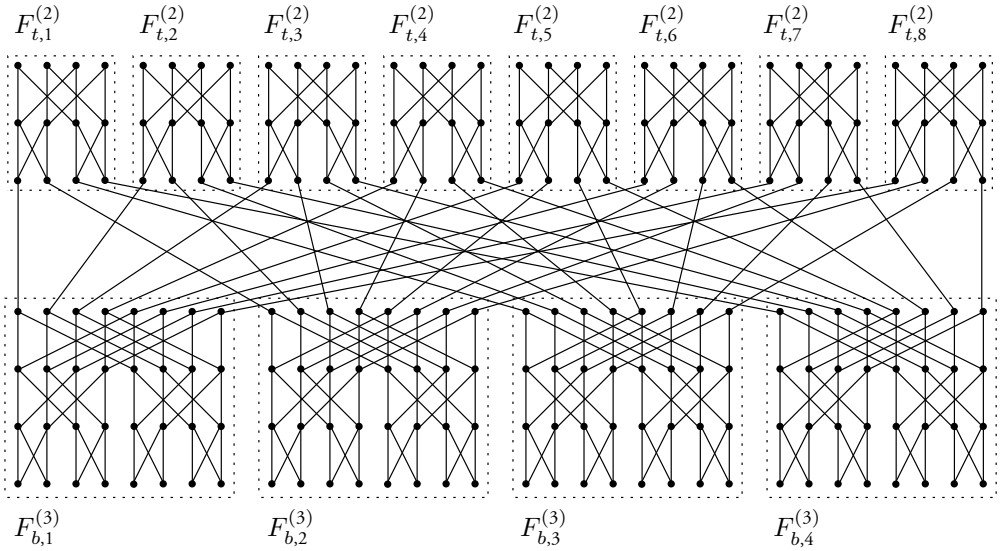


Figure 10.8 Decomposition of the FFT graph $F^{(5)}$ into four copies of $F^{(3)}$ and eight copies of $F^{(2)}$. Edges between bottom and top sub-FFT graphs are fictitious; they identify overlapping vertices between sub-FFT graphs.

A good strategy for pebbling the vertices of an FFT graph is to pebble the top FFT graphs $\{F_{t,j}^{(e)} \mid 1 \leq j \leq 2^{d-e}\}$ individually. The vertices of a top FFT graph in Fig. 10.8 are highlighted. To pebble its inputs, which are output vertices of FFT graphs below the split, it suffices to pebble the subtrees rooted at these vertices. (They are also highlighted.) Such subtrees are completely balanced binary trees with 2^{d-e} inputs. Thus, $d-e+1$ pebbles and $2^{d-e+1}-1$ pebble placements suffice to place a pebble on the root of one such subtree. If these subtrees are pebbled in sequence, pebbles can be left on the inputs to a 2^e-point FFT graph $F^{(e)}$ above the split using at most $2^e + d - e$ pebbles and $2^e(2^{d-e+1}-1)$ pebble placements. Since $2^e + 1$ pebbles and $e2^e$ pebble placements suffice to pebble $F^{(e)}$ level by level without repebbling vertices, it follows that all instances of $F^{(e)}$ above the split can be pebbled using a total of $T = 2^d(2^{d-e+1} + e - 1)$ pebble placements and $S = 2^e + d - e$ pebbles.

We now derive an upper bound on T by deriving upper and lower bounds on the value of e satisfying $S = 2^e + d - e$. Because $S \geq 2^e$, we have $e \leq \log_2 S$. Let e_0 be the smallest integer such that $2^{e_0+1} + d \geq S$. Then, $2^{e_0} + d - e_0 \leq S$ and $e \geq e_0$. Consequently, $2^e \geq (S-d)/2$, from which we have

$$T = 2^d(2^{d-e+1} + e - 1) \leq 4\frac{2^{2d}}{(S-d)} + 2^d \log_2 S$$

Finally, $\log_2 S \leq 2^d/(S-d) \leq 2\,2^d/S$ when $2d \leq S \leq (2^d/d) + d$, from which the desired conclusion follows. ■

10.5.6 Merging Networks

In this section we consider networks of comparators to merge two sorted lists. Such networks were described in Section 6.8 and an example was given, Batcher's (m, p) bitonic merging network.

A **comparator element** computes the function $\otimes : \mathcal{A}^2 \mapsto \mathcal{A}^2$ that returns the maximum and minimum of its two arguments, that is, $\otimes(a, b) = (\max(a, b), \min(a, b))$.

LEMMA 10.5.5 *Consider a comparator-based merging network that merges two sorted lists of n distinct elements $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ $(x_i \leq x_{i+1})$ and $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)$ $(y_i \leq y_{i+1})$ to produce the sorted list $\boldsymbol{z} = (z_1, z_2, \ldots, z_{2n})$ of $2n$ outputs $(z_i \leq z_{i+1})$. There must be r vertex-disjoint paths from any r inputs in $\boldsymbol{x}$ to the outputs in $\boldsymbol{z}$ to which they are mapped by the network.*

Proof Working backwards from the r selected outputs, we see that each output exits from the comparator elements to which it is attached via a disjoint path, as suggested for three outputs in Fig. 10.9. Extending this argument to the remainder of the network establishes the result. ■

We next show that inputs can be given values to cause a merging network to shift its values in a fashion that permits the derivation of a space–time lower bound.

THEOREM 10.5.6 *Any straight-line comparator-based program that merges two sorted lists of n elements requires space S and time T satisfying*

$$ST = \Omega(n^2)$$

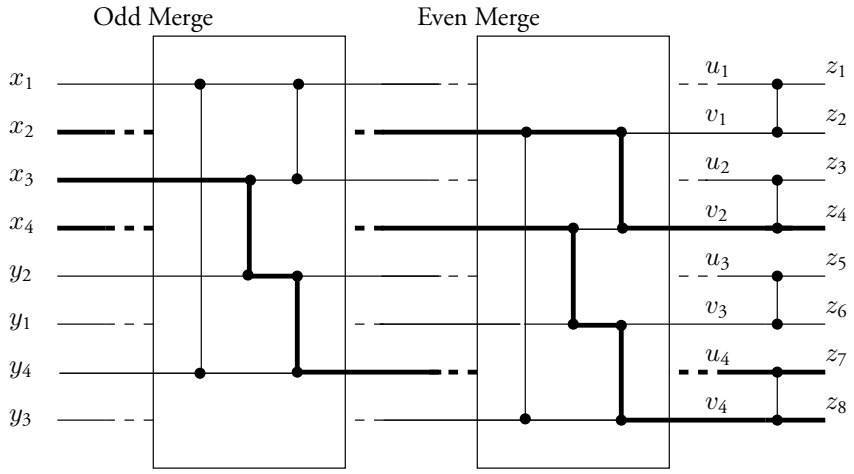


Figure 10.9 Movement of an ordered subset of the items through Batcher's bitonic merge algorithm.

This lower bound can be achieved to within a constant multiplicative factor when $2\log_2 n \leq S \leq (n/\log_2 n) + \log_2 n$.

Proof Let n be divisible by 2. Any consecutive $n/2$ inputs in $\boldsymbol{x}$ can be shifted to the middle $n/2$ positions in $\boldsymbol{z}$ through a judicious choice of values for $\boldsymbol{y}$. To see this, observe that the first $k = n - n/4 - l$ components of $\boldsymbol{y}$, $l \leq n/2$, can be chosen to be less than the first l components of $\boldsymbol{x}$ with the remaining $n - k$ components of $\boldsymbol{y}$ chosen to be larger than the first $l + n/2$ components of $\boldsymbol{x}$. This will cause elements in positions $l+1, l+2, \ldots, l+n/2$ to shift into positions $n - n/4 + 1, \ldots, n + n/4$.

Since coalescing vertices in a graph reduces neither the time nor space needed to pebble it, coalesce input vertices assigned to $\boldsymbol{x}$ whose indices are equivalent modulo $n/2$. By Lemma 10.5.5, the new graph has $n/2$-vertex disjoint paths between the new inputs and the $n/2$ outputs in positions $l+1, l+2, \ldots, l+n/2$ for each of the $n/2$ cyclic permutations. It follows that the argument applied to the cyclic shifting function (Lemma 10.5.2) applies to this function. Thus, the merging network computes a function containing a subfunction that is $(2, n/2, n/2, n/4)$-independent. The lower bound follows from Corollary 10.4.1.

As shown in Section 6.8, the graph of Batcher's bitonic merging network is an FFT graph. Thus, the upper bounds given in Theorem 10.5.5 apply. ■

10.6 Worst-Case Tradeoffs for Pebble Games*

In this section we show that degree-d graphs on n vertices can be pebbled with $O(n/\log n)$ pebbles (Theorem 10.7.1) and that some graphs require this many (Theorem 10.8.1). These results do not answer the question of how bad the space–time tradeoff can be for an arbitrary graph. To address this question we must make it precise. Lengauer and Tarjan [196] state it as follows: is there a value for the space S, say, $S_J(n)$, such that for positive constants $c_1(d)$ and $c_2(d)$ if $S \leq c_1(d)S_J(n)$, some graph on n vertices requires time superpolynomial in

n to pebble it, whereas for $S \geq c_2(d)S_J(n)$ all graphs on n vertices can be pebbled with a polynomial number of steps? They show that there is such a **jump value for space** and that $S_J(n) = \Theta(n/\log\log n)$. Since all graphs on n vertices can be pebbled with $O(n/\log n)$ space, their result shows there exist graphs on n vertices that require time exponential in n when pebbled with this number of pebbles.

10.7 Upper Bounds on Space*

We establish upper bounds on space for the class $\mathcal{G}(n, d)$ of directed acyclic graphs on n vertices that have maximum in-degree d and out-degree 2. We limit the out-degree to 2 because many straight-line programs with fan-out $k > 2$ (and their associated DAGs) can be reorganized so that each computation with fan-out k can be replaced by a binary tree of replicating subcomputations in which edges are directed from the root to the leaves. This at most doubles the number of vertices in the graph. (See Problem 10.12.)

THEOREM 10.7.1 *Let $\mathcal{G}(n, d)$ be graphs with n vertices, in-degree d, and out-degree 2 for d fixed. Then $S_{\min}(n, d)$, the minimum space needed to pebble any DAG in $\mathcal{G}(n, d)$, satisfies $S_{\min}(n, d) = O(n/\log n)$.*

Proof Let $E_{\min}(p, d)$ be the minimum number of edges in any graph in $\mathcal{G}(n, d)$ that requires p pebbles in the pebble game. We show that $E_{\min}(p, d) \geq cp\log_2 p$ for some constant $c > 0$. From this it follows that

$$p \leq 2(E_{\min}(p, d)/c)/\log_2(E_{\min}(p, d)/c)$$

when $p \geq 2$ and $E_{\min}(p, d) \geq 2c$. (See Problem 10.3.)

Consider a graph $G = (V, E)$ in $\mathcal{G}(n, d)$ with $|E|$ edges. The number of edges incident on vertices is $2|E|$. Since each vertex has at most $d + 2$ incident edges, $2|E| \leq (d + 2)|V| = (d + 2)n$. The upper bound on the number of pebbles, p, follows from this fact and the previous discussion.

Let $G = (V, E)$ in $\mathcal{G}(n, d)$ require p pebbles. An edge in E is a pair of vertices (u, v). Let $V_1 \subseteq V$ be vertices that can be pebbled with $p/2$ or fewer pebbles. Let $V_2 = V - V_1$. Thus, every vertex in V_2 requires more than $p/2$ pebbles. Let E_i, $i = 1, 2$, be the set of edges both of whose endpoints are in V_i. Let $G_i = (V_i, E_i)$. Let $A = E - (E_1 \cup E_2)$; that is, A is the set of edges joining vertices in V_1 and V_2.

We now show that there exists a vertex in G_2 that requires more than $p/2 - d$ pebbles if the pebble game is played on G_2 only. Suppose not. Then we show that every vertex in G can be pebbled with fewer than p pebbles. Certainly every vertex in V_1 can be pebbled with fewer than p pebbles. Consider vertices in V_2. We show they can be pebbled with fewer than p pebbles, thereby establishing a contradiction.

Let $\nu \in V_2$ be pebbled with $p/2 - d$ or fewer pebbles when G_2 alone is pebbled. In pebbling ν as part of the complete graph G, we may need to pebble a vertex $\omega \in V_2$ some of whose immediate predecessors are in V_1. As we encounter such vertices ω, advance a pebble to each of ω's predecessors in V_1 one at at time until all predecessors of ω are pebbled. After pebbling a predecessor in V_1, remove pebbles in V_1 not on such predecessors. When all of ω's predecessors in V_1 have been pebbled, pebble ω itself using one of the $p/2 - d$ or fewer pebbles reserved for pebbling on V_2. This strategy uses at most $p/2 + d - 1$ pebbles on vertices in V_1, at most $d - 1$ for all but the last predecessor in V_1 and at most $p/2$

for the last such predecessor, and at most $p/2 - d$ pebbles on vertices in V_2, for a total of at most $p - 1$. This is a contradiction. It follows that G_2 requires at least $p/2 - d + 1$ pebbles when pebbled alone and must have at least $E_{\min}(p/2 - d + 1, d)$ edges. Note that $E_{\min}(p/2 - d + 1, d) \geq E_{\min}(p/2 - d, d)$.

There is also some vertex in G_1 that requires at least $p/2 - d$ vertices, as we show. By assumption every vertex in V_1 must be pebbled. Suppose that each can be pebbled with $p/2 - d - 1$ pebbles. There must be a vertex η in V_2 all of whose predecessors are in V_1. (If not, we can always move backward from a vertex in V_2 to one of its immediate predecessors in V_2, a process that must terminate since the finite acyclic graph does not have a cycle.) Thus, the vertex η can be pebbled with $p/2 - 1$ pebbles using the pebbling strategy described in the preceding paragraph for ω, contradicting the definition of V_2. It follows that G_1 must have at least $E_{\min}(p/2 - d, d)$ edges.

Consider now the set of edges A connecting vertices in V_1 and V_2. If $|A| \geq p/4$, $E_{\min}(p, d) \geq 2E_{\min}(p/2 - d, d) + |A|$ because both G_1 and G_2 have $E_{\min}(p/2 - d, d)$ edges. If $|A| < p/4$, pebbles can be placed on the endpoints of edges of A in V_1 using at most $p/2 + p/4 - 1 \leq 3p/4$ pebbles, with the strategy for ω given above. If we leave at most $p/4$ pebbles on these vertices, $3p/4$ pebbles are available to pebble the vertices in V_2. If V_2 does not require at least $3p/4$ pebbles, we have a contradiction to the assumption that p pebbles are needed. Thus, there must be an output vertex μ that requires at least $3p/4$ pebbles, for if not, none of its predecessors can require more.

We show that a graph requiring at least $3p/4$ pebbles has a subgraph with at least $p/(4d)$ fewer edges that requires at least $p/2$ pebbles. To see this, observe that some predecessor of the output vertex μ requires at least $3p/4 - d$ pebbles. Delete μ and all its incoming edges to produce a subgraph with at least one fewer edge requiring at least $3p/4 - d$ pebbles. Repeat this process $p/(4d)$ times to produce the desired result. It follows that G_2 has at least $E_{\min}(p/2, d) + p/(4d)$ edges.

Thus, when either $|A| \geq p/4$ or $|A| < p/4$, at least $2E_{\min}(p/2 - d, d) + p/(4d)$ edges are required, and

$$E_{\min}(p, d) \geq 2E_{\min}(p/2 - d, d) + \frac{p}{4d}$$

The solution to this recurrence is $E_{\min}(p, d) \geq cp \log p$ for some constant $c \geq 1/8d$ and a sufficiently large value of p. ■

10.8 Lower Bound on Space for General Graphs*

Now that we have established that every graph in $\mathcal{G}(n, d)$ can be pebbled with $O(n/\log n)$ pebbles, we show that for all n there exists a graph $G(n)$ in $\mathcal{G}(n, d)$ whose minimum space requirement is at least $c_5 n/\log n$ for some constant $c_5 > 0$.

The graph $G(n)$ is obtained from a recursively constructed graph $H(k)$ on 2^k inputs and 2^k outputs, $n/2 < 2^k \leq n$, by adding $n - 2^k$ vertices and no edges. The graph $H(k)$ is composed of two copies of $H(k - 1)$ and two copies of an n-superconcentrator, which is defined below.

DEFINITION 10.8.1 *An n-**superconcentrator** is a directed acyclic graph $G = (V, E)$ with n input vertices and n output vertices and the property that for any r inputs and any r outputs,*

$1 \leq r \leq n$, there are r vertex-disjoint paths in G connecting these inputs and outputs. (Paths are **vertex-disjoint** *if they have no vertices in common.)*

For $n = 2^k$ Valiant [342] has shown the existence of n-superconcentrators $SC(k)$ that have 2^k inputs, 2^k outputs, and $c2^k$ edges. Since his graphs have in-degree greater than 2, replace vertices with in-degree $d > 2$ with binary trees of d leaves, thereby at most doubling the size of the graph. (See Problem 10.12.) This provides the following result.

LEMMA 10.8.1 *For some constant $c > 0$ and each integer k and $n = 2^k$ there exists an n-superconcentrator $SC(k)$ with $c2^k$ vertices.*

We let $H(8) = SC(8)$. For $k \geq 8$ we construct $H(k+1)$ recursively from two copies of $H(k)$, two copies of $SC(k)$, and extra edges, as suggested in Fig. 10.10. Here edges are directed from left to right. The 2^k output vertices of the first (leftmost) copy of $SC(k)$ (called $SC_1(k)$) are identified with the 2^k input vertices of the first copy of $H(k)$ (called $H_1(k)$), the 2^k output vertices of $H_1(k)$ are identified with the 2^k input vertices of the second copy of $H(k)$ (called $H_2(k)$), and the 2^k output vertices of $H_2(k)$ are identified with the 2^k input vertices of the second copy of $SC(k)$ (called $SC_2(k)$). In addition, we introduce 2^{k+1} new input vertices and 2^{k+1} new output vertices. The first (topmost) half of the new inputs (called I_t) are connected via individual edges to the inputs of $SC_1(k)$. The second (bottommost) half of the new inputs (called I_b) are also connected via individual edges to the inputs of $SC_1(k)$. The new inputs are connected individually to the new outputs. Finally, each output of $SC_2(k)$ is connected via individual edges to two new output vertices, one each in the top (called O_t) and bottom half (called O_b) of the new outputs.

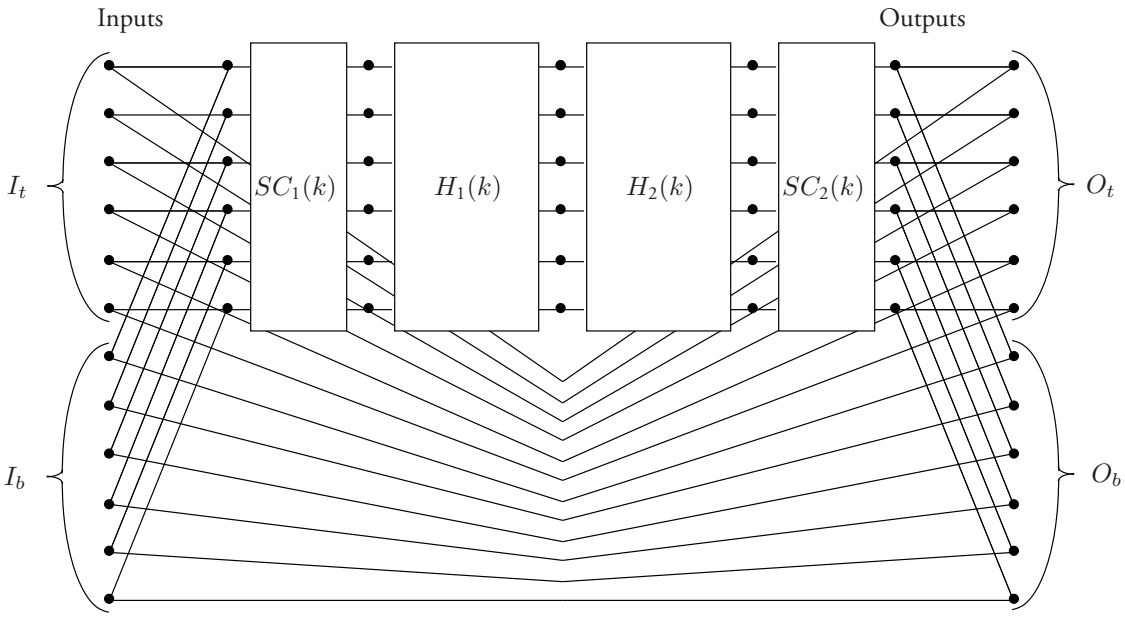


Figure 10.10 A graph $H(k+1)$ requiring large minimum space.

The graph $H(k)$ has $n(k) = |H(k)|$ vertices, where $n(k)$ satisfies the following:

$$n(8) = c2^8$$
$$n(k+1) = 2n(k) + (2c+4)2^k$$

The solution to the recurrence is $n(k) = (k-7)c2^k + (k-8)2^{k+1}$, as can be shown directly. The graph $H(k)$ is in $\mathcal{G}(n(k), 2)$.

Important subgraphs of $H(k+1)$ have the superconcentrator property, as we now show. This result is applied in the subsequent lemma to derive bounds on the amount of space used to pebble outputs of $H(k+1)$.

LEMMA 10.8.2 *The subgraphs of $H(k+1)$ on 2^k inputs and 2^k outputs defined by vertices and edges on paths from either inputs in I_t or inputs in I_b to the outputs of SC_1 and $H_1(k)$ have the 2^k-superconcentrator property.*

Proof The superconcentrator property applies to the outputs of $SC_1(k)$ by definition. Note that the jth input of $H_1(k)$ is connected to its jth output by an individual edge for $1 \leq j \leq 2^k$. Thus, any r outputs of $H_1(k)$ have vertex-disjoint paths to the corresponding inputs of $H_1(k)$. By the superconcentrator property of $SC_1(k)$, there are vertex-disjoint paths from these outputs of $SC_1(k)$ to any r of its inputs. These statements obviously apply to inputs in I_t and I_b. ■

Our goal is to show that pebbling the graph $H(k)$ requires a number of pebbles proportional to $n(k)/\log n(k)$. To do this we establish the following stronger condition, which implies the desired result.

LEMMA 10.8.3 *Let $c_1 = 14/256$, $c_2 = 3/256$, $c_3 = 34/256$, and $c_4 = 1/256$. To pebble at least $c_1 2^k$ outputs of $H(k)$ in any order from an initial placement of at most $c_2 2^k$ pebbles requires there be a time interval $[t_1, t_2]$ during which at least $c_3 2^k$ inputs are pebbled and at least $c_4 2^k$ pebbles remain on the graph.*

Proof The proof is by induction on k with $k = 8$ as the base case. For the base case, consider pebbling $c_1 2^k = 14$ outputs during a time interval $[0, t]$ from an initial placement of no more than $c_2 2^k = 3$ pebbles.

By Problem 10.27 any four outputs of $SC(8)$ are connected via pebble-free paths to $256 - 3 = 253$ inputs. At least one of these four outputs, say v, has pebble-free paths to $64 = \lceil 253/4 \rceil$ inputs. Let $t_1 - 1$ be the last time at which all 64 of these inputs have pebble-free paths to v. Let t_2 be the last time at which a pebble is placed on these 64 inputs. During the time interval $[t_1, t_2]$ at least $64 \geq c_3 2^k$ inputs are pebbled and at least one pebble remains on the graph; that is, at least $c_4 2^k$ pebbles remain. This establishes the base case.

Now assume the conditions of the lemma (our inductive hypothesis) hold for k. We show they hold for $k+1$. Assume that at least $c_1 2^{k+1}$ outputs of $H(k+1)$ are pebbled in any order from an initial placement of at most $c_2 2^{k+1}$ pebbles during a time interval $[t_a, t_b]$.

We consider four cases including the following two cases. There is an interval $[t_1, t_2] \subseteq [t_a, t_b]$ during which at least $c_2 2^k$ pebbles are always on the graph and at least $c_3 2^k$ outputs of either (1) $SC_1(k)$, or (2) $H_1(k)$ are pebbled. By Lemma 10.8.2 the subgraph of $H(k+1)$ consisting of paths from I_t (and I_b) to the outputs of each of these graphs constitutes a 2^k-superconcentrator. This is the only fact about these two cases that we use. Without loss of generality, we show the hypothesis holds for the first of them.

The graph consisting of paths from inputs in I_t to the outputs of $SC_1(k)$ constitutes a 2^k-superconcentrator. Prior to time t_a there are at most $c_2 2^{k+1}$ pebbles on the graph and during the interval $[t_1, t_2]$ there are at least $c_2 2^k$ (but at most $c_2 2^{k+1}$) pebbles on the graph. Thus, there is a latest time t_0 before t_1 when there are at most $c_2 2^{k+1}$ pebbles on the graph. Since $c_3 2^k \geq c_2 2^{k+1} + 1$ outputs of $SC_1(k)$ are pebbled in the interval $[t_1, t_2]$ (and in the interval $[t_0, t_2]$), by Problem 10.27 at time t_0 there are at least $2^k - c_2 2^{k+1} \geq c_3 2^k$ inputs in I_t (and in I_b) that are connected by pebble-free paths to the pebbled outputs of $SC_1(k)$. Thus, at least $c_3 2^{k+1}$ inputs in I_t and I_b are connected via pebble-free paths to the pebbled outputs of $SC_1(k)$. In $[t_0, t_1 - 1]$ there are at least $c_2 2^{k+1}$ pebbles continuously on the graph, whereas there are at least $c_2 2^k$ pebbles during $[t_1, t_2]$. Since $c_2 2^k \geq c_4 2^{k+1}$, the number continuously on the graph in $[t_1, t_2]$ is at least $c_4 2^{k+1}$ and we have the desired conclusion for $H(k+1)$.

In the third case, there is an interval $[t_1, t_2] \subseteq [t_a, t_b]$ during which at least $c_1 2^k$ outputs of the full graph $H(k+1)$ are pebbled and at least $c_2 2^k$ pebbles are always on the graph. This implies that during $[t_1, t_2]$ either $c_1 2^k/2$ outputs in O_t or in O_b are pebbled, which in turn implies that at least $c_1 2^k/2$ outputs of $SC_2(k)$ are pebbled. Since $c_1 2^k/2 \geq c_2 2^{k+1} + 1$ (at most $c_2 2^{k+1}$ pebbles are on $H(k+1)$), it follows from Problem 10.27 that at least $2^k - c_2 2^{k+1} \geq c_3 2^k$ inputs in I_t (or I_b) are connected via pebble-free paths to the pebbled outputs of $SC_2(k)$. The total number of such inputs is $c_3 2^{k+1}$. Since $c_2 2^k \geq c_4 2^{k+1}$, there are at least $c_4 2^{k+1}$ pebbles on the graph continuously during $[t_1, t_2]$ and we have the desired conclusion.

In the fourth case none of the previous cases hold. Since $c_1 2^{k+1}$ outputs of $H(k+1)$ are pebbled during $[t_a, t_b]$, there is an earliest time $t_1 \in [t_a, t_b]$ such that $c_1 2^k$ outputs of $H(k+1)$ are pebbled in the interval $[t_a, t_1 - 1]$. Since the third case does not hold, there is a time $t_2 \leq t_1$ such that fewer than $c_2 2^k$ pebbles are on the graph at $t_2 - 1$ and at least $c_1 2^k$ outputs of $H(k+1)$ are pebbled in the interval $[t_2, t_b]$. It follows that at least $c_1 2^k/2$ outputs of $SC_2(k)$ are pebbled during this interval. Since $c_1 2^k/2 \geq c_2 2^k + 1$, it follows from Problem 10.27 that at least $2^k - c_2 2^k \geq c_3 2^k$ inputs to $SC_2(k)$ (which are outputs to $H_2(k)$) are connected via pebble-free paths to the pebbled outputs of $SC_2(k)$ and must be pebbled during $[t_2, t_b]$. Since $c_3 2^k \geq c_1 2^k$, by the inductive hypothesis there is an interval $[t_d, t_e] \subseteq [t_2, t_b]$ during which at least $c_3 2^k$ inputs of $H_2(k)$ (which are outputs of $H_1(k)$) are pebbled and $c_4 2^k$ pebbles reside continuously on $H_2(k)$.

Since the second case does not hold, by an argument paralleling that given in the preceding paragraph there must be a time $t_3 \in [t_d, t_e]$ such that at most $c_3 2^k/2$ outputs of $H_1(k)$ are pebbled during $[t_d, t_3 - 1]$ and fewer than $c_2 2^k$ pebbles reside on $H(k+1)$ at $t_c - 1$. Thus, during $[t_3, t_e]$ at least $c_3 2^k/2 \geq c_1 2^k$ outputs of $H_1(k)$ are pebbled from an initial configuration of fewer than $c_2 2^k$ pebbles. By the inductive hypothesis there is an interval $[t_f, t_g] \subseteq [t_3, t_e]$ during which at least $c_3 2^k$ inputs of $H_1(k)$ (which are outputs of $SC_1(k)$) are pebbled and $c_4 2^k$ pebbles reside on $H_1(k)$ continuously.

Since the first case does not hold, again paralleling an earlier argument there must be a time $t_4 \in [t_f, t_g]$ such that at most $c_3 2^k/2$ outputs of $SC_1(k)$ are pebbled during $[t_f, t_4 - 1]$ and fewer than $c_2 2^k$ pebbles reside on $H(k+1)$ at $t_4 - 1$. Thus, during $[t_4, t_g]$ at least $c_3 2^k/2 \geq c_2 2^k + 1$ outputs of $SC_1(k)$ are pebbled from an initial configuration of fewer than $c_2 2^k$ pebbles. By Problem 10.27 at least $2^k - c_2 2^k \geq c_3 2^k$ inputs of $SC_1(k)$ are connected via pebble-free paths to the pebbled outputs. Thus at least $c_3 2^k$ corresponding inputs in both I_t and I_b must be pebbled for a total of at least $c_3 2^{k+1}$ inputs.

Since at least $c_4 2^k$ pebbles reside continuously on both $H_1(k)$ during $[t_d, t_e]$ and on $H_2(k)$ during $[t_f, t_g]$ and $[t_f, t_g] \subseteq [t_d, t_e]$, it follows that $c_4 2^k + c_4 2^k = c_4 2^{k+1}$ reside continuously on $H(k+1)$ during $[t_f, t_g]$. ■

We are now ready to show the existence of a graph on n vertices that requires $\omega(n/\log n)$ minimal space.

THEOREM 10.8.1 *For integers $n \geq 1$ there exists a graph $G(n)$ in $\mathcal{G}(n, d)$ that requires minimum space $S_{\min}(G(n)) \geq c_5 n/\log n$ for some constant $c_5 > 0$.*

Proof For $n \geq 2^8$, let k be the largest integer such that $n(k) \leq n$; that is, $n(k) \leq n < n(k+1)$. Construct the graph $G(n)$ by adding $n - n(k)$ vertices and no edges to the graph $H(k)$. An optimal pebbling strategy for $G(n)$ pebbles the added vertices one at a time using one pebble, after which $H(k)$ is pebbled. From Lemma 10.8.3 it follows that pebbling $H(k)$ requires at least $c_4 2^k$ pebbles, since at least this many must reside on the graph at one time. Since $n(k+1) \leq 4n(k)$ for $k \geq 8$ and $c \geq 2$, it follows that $n/4 \leq n(k) \leq n$. This implies that $2^k \leq n$ and $k \leq \log_2 n$ and that $n/4 \leq k(c+2)2^k \leq (\log_2 n)(c+2)2^k$. From this we have $2^k \geq c_5 n/\log_2 n$, where $c_5 = 1/(4c+8)$. The conclusion follows by observing that at least $(c_4 c_5)n/\log_2 n$ pebbles are needed to pebble $G(n)$. ■

10.9 Branching Programs

The general branching program is a serial computational model that permits data-dependent computation, unlike the pebble game. A branching program is a directed graph consisting of a single starting vertex and in which vertices are labeled with predicates. Each vertex has one outgoing edge for each value of its predicate. (See, for example, Figs. 10.11 and 10.12.) Time in this model is the number of queries performed, and computations other than queries are not counted. The space used by a branching program is the base-2 logarithm of the number of vertices in its graph. Lower bounds on space and input time obtained with the branching program apply to within constant multiplicative factors to the pebble game and the RAM model. (See Section 10.9.1.)

As noted in Section 10.1.1, since the branching program reads inputs in a less constrained manner than the straight-line program, it may be possible to solve some problems with branching programs using less space or time than in the pebble game. As a consequence, space–time lower bounds for branching programs may be smaller than for the pebble game. Thus, if a problem is going to be solved with straight-line programs, such as an algebraic circuit, it is better to use lower bounds derived with the pebble game unless the branching program gives the same lower bounds. In particular, branching programs give smaller space–time lower bounds for integer multiplication and shifting (see Section 10.13.2) than does the pebble game.

We examine two kinds of branching programs in this section, general branching programs and decision branching programs.

DEFINITION 10.9.1 *A* **multigraph** *is a graph that may have more than one edge between two vertices. A* **directed multigraph** *is a multigraph in which each edge has a direction. A* **directed acyclic multigraph** *(DAM) is a multigraph with no directed cycles. A* **rooted directed acyclic multigraph** *is a multigraph with a* **root vertex***, a vertex with no edges directed into it, and is such that every vertex can be reached via some path from the root. A* **sink vertex** *has no edges directed away from it.*

A **branching program** $\mathcal{P}$ *with input variables* $\boldsymbol{x}$ *over the set* $\mathcal{A}$ *and output variables* $\boldsymbol{y}$ *over the set* $\mathcal{F}$ *is a rooted directed acyclic multigraph that has a query* $q(\boldsymbol{x})$ *associated with each vertex except for sink vertices and has a query outcome associated with every edge directed away from a vertex. Each edge may also carry as a label the values of some output variables, with the proviso that each output variable is assigned exactly one value along any one path from the root to a sink vertex.*

The **decision branching program** is a special kind of branching program in which the queries $q(\boldsymbol{x})$ compare two variables and produce either the two outcomes $\{\leq, >\}$ or the three outcomes $\{<, =, >\}$. Figure 10.11 shows an example of a decision branching program that merges two 2-element sorted lists (u_1, u_2) and (v_1, v_2) ($u_1 \leq u_2$ and $v_1 \leq v_2$) by using queries that compare the values of two input variables. Each vertex in the example has two out-directed edges corresponding to the results of the query. The outputs appear in sorted order along a path from the root to a leaf.

A **decision tree** is a decision branching program whose DAM (directed acyclic multigraph) is a tree. A decision tree may be constructed for a sequential comparison-based sorting algorithm, such as Batcher's odd-even merging algorithm of Section 6.8, by associating the first comparison with the root, the second comparisons with the roots of the left and right subtrees, etc.

DEFINITION 10.9.2 *A* **computation on a branching program** $\mathcal{P}$ *is a traversal of the unique path in the DAM from the root to a leaf determined by the values of the input variables in* $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ *over the set* $\mathcal{A}$*. The* **output of the computation** *is the sequence of output values in* $\boldsymbol{y} = (y_1, y_2, \ldots, y_m)$ *over the set* $\mathcal{F}$ *encountered on the edges of the path traversed.*

A function $f^{(n)} : \mathcal{A}^n \mapsto \mathcal{F}^m$ *with input variables in* $\boldsymbol{x}$ *and output variables in* $\boldsymbol{y}$*, namely*

$$f^{(n)}(x_1, x_2, \ldots, x_n) = (y_1, y_2, \ldots, y_m)$$

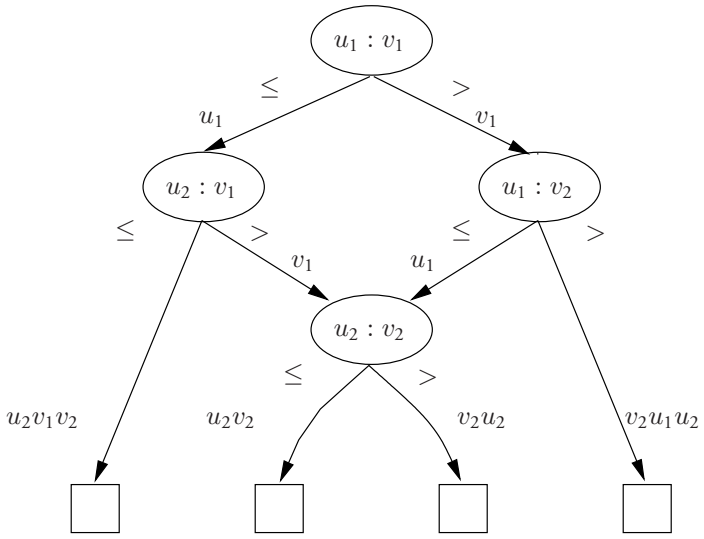


Figure 10.11 A decision branching program that merges the lists (u_1, u_2) and (v_1, v_2) when $u_1 \leq u_2$ and $v_1 \leq v_2$.

is **computed** *by $\mathcal{P}$ if for each value of $\boldsymbol{x}$ the correct value of each output variable appears exactly once on each path from the root to a leaf.*

The **time associated with a computation** *is the length of the path traversed by the computation. The* **computation time T of a branching program** *is the length of its longest path.*

In Fig. 10.11 the computation associated with the input values $(u_1, u_2, v_1, v_2) = (2, 4, 1, 3)$ takes the right branch out of the root and produces the output value $v_1 = 1$, takes the left branch at the next vertex and produces $u_1 = 2$, and takes the right branch at the last vertex and produces $v_2 = 3$ and $u_2 = 4$. The output of this computation is the sorted sequence 1, 2, 3, 4, as expected. This branching program merges the two sorted lists. Each sink vertex corresponds to one of the four ways of merging the two lists. The computation time of this branching program is 3.

Branching programs that compare elements at vertices are well suited to merging and sorting but are not of the most general type.

DEFINITION 10.9.3 *A* **general branching program** *$\mathcal{P}$ with input variables $\boldsymbol{x}$ over a finite set $\mathcal{A}$ has a query of the form $x_i = ?$ associated with a variable x_i at each vertex. It also has one edge directed away from the vertex for each value of x_i. A general branching program is* **non-redundant** *if along each path from the root to a leaf a query $x_i = ?$ appears at most once.*

The general branching program is also known as a **binary decision diagram (BDD)**. BDD's are widely used in the computer-aided design (CAD) of circuits for Boolean functions.

A general branching program that convolves two short binary sequences over the integers is shown in Fig. 10.12. (Convolution is defined in Section 6.7.4.) A computation leaves the left branch of a vertex when the associated variable has value 0 and the right branch when it

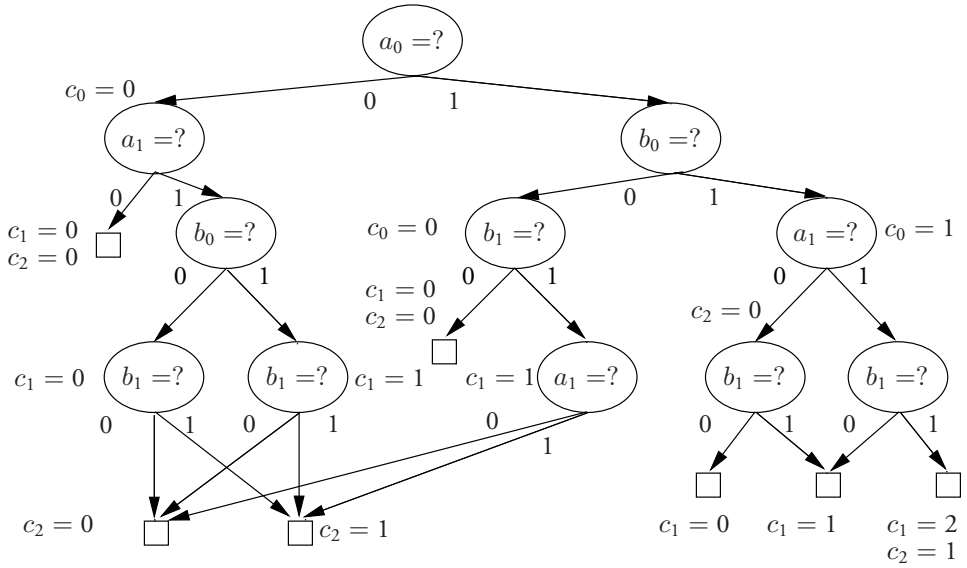


Figure 10.12 A general branching program to compute the convolution of two sequences (a_0, a_1) and (b_0, b_1).

has value 1. This branching program computes the convolution $\boldsymbol{c} = \boldsymbol{a} \otimes \boldsymbol{b}$ of the sequences $\boldsymbol{a} = (a_0, a_1)$ and $\boldsymbol{b} = (b_0, b_1)$; that is,

$$c_0 = a_0 b_0, \quad c_1 = a_0 b_1 + a_1 b_0, \quad c_2 = a_1 b_1$$

The performance of a branching program is also measured by its space complexity.

DEFINITION 10.9.4 *The* **space** *used by branching program $\mathcal{P}$ is the base-2 logarithm of the number of vertices in its directed acyclic multigraph.*

As shown in the next section, this definition permits a lower bound on the space complexity used by any reasonable general-purpose computer model equipped with a random-access read-only memory for its input data.

The following lemma demonstrates that every decision branching program can be simulated by a general branching program, thereby showing the latter to be more general than the former. (See Problem 10.35.)

LEMMA 10.9.1 *Every decision branching program with variables over a finite set $\mathcal{A}$ with computation time T and space S can be simulated by a general branching program with computation time $2T$ and space $S + \log(|\mathcal{A}| + 1)$.*

This result is proved by constructing a general branching program to simulate a comparison operator and substituting it for the comparison operator in a decision branching program. (See Problem 10.35.) The graph that results from this construction is explicitly a multigraph.

While Lemma 10.9.1 establishes that decision branching programs are no more powerful than general branching programs, this does not imply that general branching programs require less space. In fact, the space complexity of a given decision branching program is independent of the size of the set $\mathcal{A}$ over which the variables are defined; this is not true for general branching programs.

If space complexity is not an issue, a **tree program** can be constructed. This is a branching program whose DAM is a tree. The following recursive procedure converts a branching program to a tree program: a) If any immediate descendant of the root has more than one edge directed into it, make as many copies of the submultigraph rooted at that descendant as there are entering edges and direct exactly one edge into each. b) Apply this procedure recursively to each of the submultigraphs until leaf vertices are reached. This procedure does not change the length of any path in the original DAM or the computation time.

The notions of space and time can be generalized to average time and space when a probability distribution is defined on input values. (See Problem 10.37.)

Below we present a key lemma used to derive lower bounds on the space–time product. This lemma is stated for **normal-form branching programs**, general branching programs whose DAMs are **level multigraphs**, that is, multigraphs in which each vertex has a level and adjacent vertices are in adjacent levels. An example of such a graph is shown in Fig. 10.13.

LEMMA 10.9.2 *If there is a general branching program of space S and computation time T for a function f, then there is a normal-form branching program for f that has space $2S$ and computation time T.*

Proof To convert a general branching program to a normal-form branching program, create $T + 1$ copies of the general branching program, one for each time step including the zeroth.

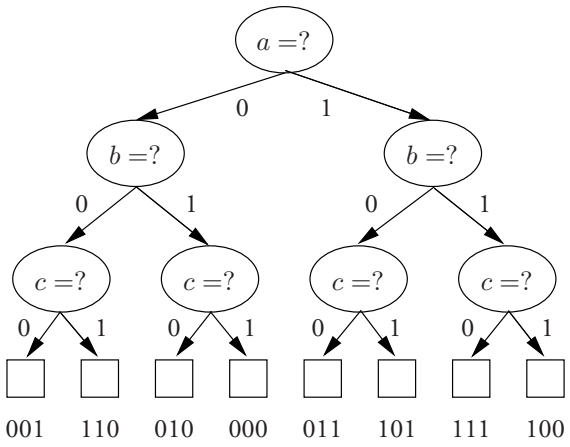


Figure 10.13 A normal-form tree program for table lookup. It has one path for each value of the input.

Delete the original edges and add an edge from vertex u in the ith copy to vertex v in the $(i+1)$st copy if there was an edge between u and v in the original graph. Now delete all edges and vertices that are not reached from the root of the zeroth branching program. (See Fig. 10.14.)

This procedure increases the number of vertices by at most a factor of T, thereby increasing the space by adding at most $\log T$. However, a branching program with space S has 2^S vertices. Thus, the length of the longest path through the program T cannot exceed 2^S, or $S + \log T \leq 2S$. ■

Generally the space S used for a branching program computation will be large by comparison with $\log T$, in which case the space bounds for normal-form branching programs and general branching programs will differ by at most a constant factor.

In the rest of this chapter when we speak of a branching program we mean a general branching program.

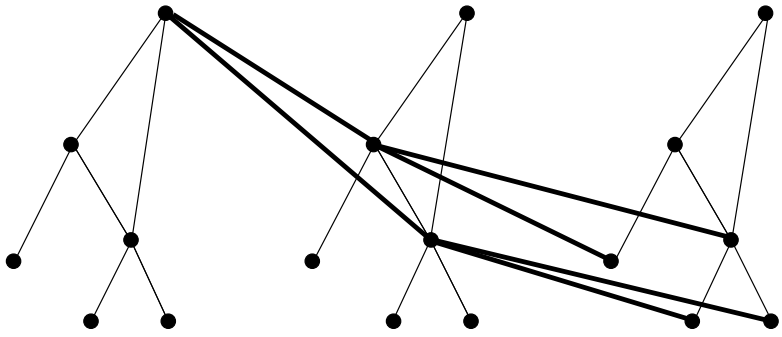


Figure 10.14 Construction of a normal-form general branching program as a level multigraph.

We close this section by describing a normal-form tree program for **table lookup**, an important programming tool that can be used to compute an arbitrary function $f^{(n)} : \mathcal{A}^n \mapsto \mathcal{A}^m$ on n variables whose value is an m-tuple. Each of the n variables is read and the value of the function is found in a table. This is simulated by a tree program with branching factor $|\mathcal{A}|$ in which the variables are read in succession until they are all read, at which point the value of the function is provided. An example of such a tree program for a function $f^{(3)} : \mathcal{B}^3 \mapsto \mathcal{B}^3$ is shown in Fig. 10.13. There is one path through the tree for each of the possible $|\mathcal{A}|^n$ assignments to the n inputs. The sink vertices are labeled by the appropriate m-tuple. Such table-lookup tree programs have computation time n and space proportional to $n \log |\mathcal{A}|$ since they have $(|\mathcal{A}|^{n+1}-1)/(|\mathcal{A}|-1)$ vertices with A edges per vertex except for those at the lowest level.

10.9.1 Branching Programs and Other Models

We begin this section with a comparison of branching programs and pebble games and conclude with a brief comparison of branching programs and the RAM model of computation.

The pebble model assumes that computation is serial and straight-line. If all algorithms used for a particular problem are of this type, the pebble game is the appropriate model, especially if the lower bounds on space–time exchanges are larger than those provided by the branching program model. (All algorithms used today for integer multiplication are straight-line and the lower bounds on the space–time product for this problem are larger with the pebble game than with the branching program model.) If the two models give the same lower bounds, then we can invoke Lemma 10.9.3 to derive lower bounds on the space–time exchanges for pebbling from those for branching programs when $\log_2 T_{\mathcal{P}}$ is small by comparison with $S_{\mathcal{P}}$, where $T_{\mathcal{P}}$ and $S_{\mathcal{P}}$ are the time and space used by the pebbling model.

Data-dependent reading of inputs may allow the branching program to perform a computation more quickly than the pebbling model. For example, merging requires a space–time product that is quadratic in the length of the input strings with the pebble game but only linear in the branching program. (See Section 10.10.2.) This demonstrates that the branching program is a much more natural model for this problem.

If the lower bounds derived with the branching program are comparable in strength to those offered by the pebbling model, as is true for most of the problems considered in this chapter, straight-line programs are the better model for these problems. But the extra flexibility offered by branching programs means that when their results are comparable to those provided by the pebble game, one must work harder to obtain them. (See Sections 10.11 and 10.12.)

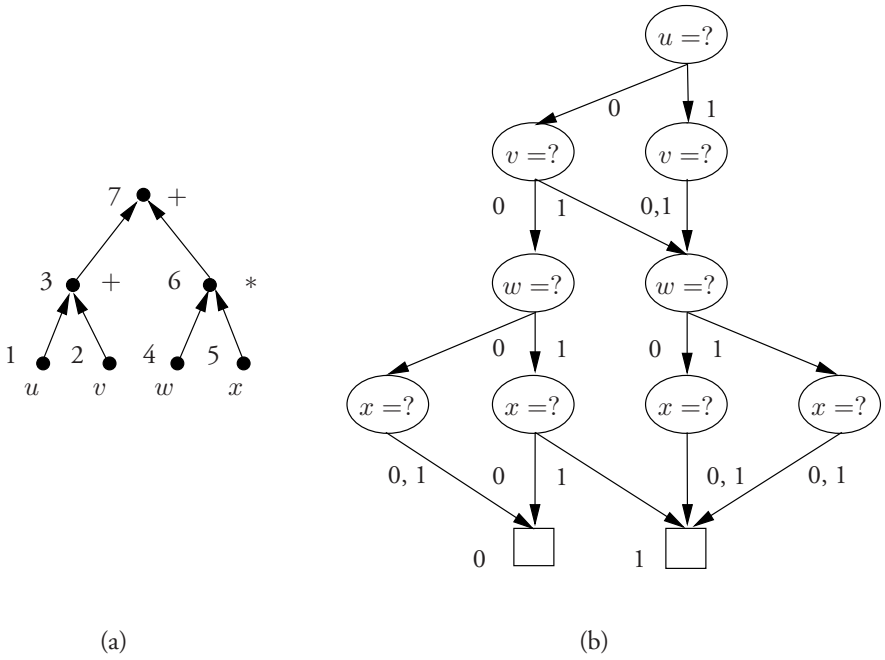
The branching program measures the time to read inputs but ignores the time for computations and the production of outputs. By contrast, the pebble game measures the time to read inputs, perform computations, and produce outputs. Although the time for computations generally cannot be ignored, the methods available today to derive lower bounds for both models are based on the time spent reading inputs. But while for many problems the time to read inputs dominates computation time for many values of space, when space is large the pebbling model has the potential to give larger lower bounds than the branching program model. For example, no way is known to compute the n-point DFT with fewer than $\Theta(n \log n)$ steps, the number used by the FFT algorithm, although in the limit of large space the branching program gives a lower bound on space proportional to n.

To simulate the pebbling of a DAG by a branching program we must give an **interpretation** to each vertex of the DAG: assign an operation to each non-input vertex and a variable as

well as values to each input vertex. Two different interpretations of a DAG may yield different branching programs. Of course, a DAG is pebbled without regard to the interpretation of vertices: the pebble-game lower bounds use only the fact that vertices can hold one of $|\mathcal{A}|$ values and do not depend explicitly on the interpretation given to their operator.

LEMMA 10.9.3 *Given a pebbling $\mathcal{P}$ of an interpreted directed acyclic graph G that uses $S_{\mathcal{P}}$ pebbles and $T_{\mathcal{P}}$ input steps to compute a function with operations over a finite set $\mathcal{A}$, there is a branching program with space $S_{\mathcal{P}} \log |\mathcal{A}| + \log (2T_{\mathcal{P}})$ and time $T_{\mathcal{P}}$ that computes the function computed by G. Thus, if $2T_{\mathcal{P}} \leq |\mathcal{A}|^{S_{\mathcal{P}}}$, simultaneous lower bounds on the space and time for a branching program for the function imply simultaneous lower bounds on space and time in the pebble game that differ by at most constant multiplicative factors.*

Proof We construct a branching program $\mathcal{Q}$ to simulate the pebbling $\mathcal{P}$ of a directed acyclic graph that uses $S_{\mathcal{P}}$ pebbles and $T_{\mathcal{P}}$ steps. (Figure 10.15 illustrates the construction of such a branching program.) Initially the branching program has a single vertex, the root, which is labeled with the first variable to be pebbled according to $\mathcal{P}$. Advance the first pebble as far as possible. Create a vertex in the branching program for each value of the operation or input covered by the first pebble. Label these new vertices with the name of the second input to be pebbled and attach an edge from the root vertex to these new vertices labeled with the corresponding value for the first input. Advance pebbles as far as possible according to $\mathcal{P}$ and create one new vertex in the branching program for each different tuple of values



(a) (b)

Figure 10.15 A general branching program (b) that simulates the pebbling of a DAG (a) in the vertex order 1, 2, 4, 3, 5, 6, 7. The DAG input variables are denoted u, v, w, and x and assume values in $\{0, 1\}$. $+$ denotes OR and $*$ denotes AND.

residing under the pebble(s) currently on the DAG. (In the example of Fig. 10.15, after placing a pebble on the second vertex we advance a pebble to the third vertex and remove all other pebbles. Thus, only two vertices are added to the branching program at this step.) Label the new vertices with the third input to be pebbled. Now repeat the above process by advancing pebbles as far as possible (in the example, pebbles now reside on the third and fourth vertices), add one new vertex for each tuple of pebbles on the DAG (four vertices are added), and connect edges from the previous to the current set of new vertices that conform to the values assumed at the vertices of the DAG. This process is repeated until all inputs have been pebbled.

Since the values of operations are always determined by the values under at most $S_{\mathcal{P}}$ pebbles, the number of new vertices added in $\mathcal{Q}$ with the pebbling of each new input vertex in G is most $|\mathcal{A}|^{S_{\mathcal{P}}}$. Since $T_{\mathcal{P}}$ input vertices of G are pebbled, it follows that $\mathcal{Q}$ has at most $T_{\mathcal{P}}|\mathcal{A}|^{S_{\mathcal{P}}} + 1 \leq 2T_{\mathcal{P}}|\mathcal{A}|^{S_{\mathcal{P}}}$ vertices, from which the conclusion follows. ■

A branching program can also simulate a computation by a general model of computation, such as the RAM discussed in Section 3.4, as we now show. Let the RAM have M b-bit words of memory and a finite number of b-bit words in its CPU. Consider any program for such a machine. Its **state** is determined by the values in its registers and memory locations. Thus the RAM has at most $O(2^{Mb})$ states. Let the **space used by a RAM** be the base-2 logarithm of the number of its states. Let the RAM execute T_{RAM} steps to read its inputs. We simulate this computation in the same fashion as with the pebble game. After reading an input variable, the branching program enters one of at most $O(2^{Mb})$ vertices corresponding to states of the RAM. Since the RAM reads inputs on T_{RAM} steps, the branching program also takes T_{RAM} steps and has at most $O(T_{\text{RAM}}2^{Mb})$ vertices or uses space of at most $O(Mb + \log T_{\text{RAM}})$. As long as Mb is larger than some multiple of $\log T_{\text{RAM}}$, simultaneous lower bounds on the time to read inputs and space of a branching program for a function computed by the RAM serve as lower bounds on the same quantities on the RAM. The following lemma summarizes this discussion.

LEMMA 10.9.4 *Given a RAM program that uses space S_{RAM} and T_{RAM} input steps to compute $f : \mathcal{A}^n \mapsto \mathcal{A}^m$ there is a branching program with space $O(S_{\text{RAM}} + \log(2T_{\text{RAM}}))$ and time T_{RAM} that computes f. Thus, if $2T_{\text{RAM}} \leq 2^{S_{\text{RAM}}}$, simultaneous lower bounds on the space and time for a branching program for the function imply simultaneous lower bounds on the space and time on the RAM that differ by at most constant multiplicative factors.*

10.10 Straight-Line Versus Branching Programs

In this section we show that some problems can use space and time more efficiently with branching programs than they can with the pebble game. We demonstrate this for the cyclic shifting function $f^{(n)}_{\text{cyclic}} : \mathcal{B}^{n+\lceil \log n \rceil} \mapsto \mathcal{B}^n$ introduced in Section 2.5.2 and the merging problem introduced in Section 6.8. However, for all of the other problems studied in this chapter the lower bounds obtained with these two models are the same up to constant multiplicative factors, except for integer multiplication, where the branching program bound is smaller by a factor of $\log^2 n$.

It is important to note, however, that the superiority of branching programs arises from the assumption that inputs can be read in a data-dependent fashion, an assumption that is

not available to straight-line programs. As we know from Problem 10.20, if branching is allowed but inputs must be read in a data-independent fashion by an input-output-oblivious finite-state machine, Theorem 10.4.1 applies. Thus, branching programs that read inputs in a data-independent fashion have no advantage over straight-line programs, at least in terms of lower bounds on space–time exchanges.

10.10.1 Efficient Branching Programs for Cyclic Shift

We present a branching program for $f^{(n)}_{\text{cyclic}}$ that uses space $S = O(logn)$ and time $T = n + \lceil \log n \rceil$; that is, $ST = O(n \log n)$, a product that is much less than the $\Theta(n^2)$ product required in the pebble game. (See Section 10.5.2.)

The function $f^{(n)}_{\text{cyclic}}$ has $n + \lceil \log n \rceil$ Boolean variables, $\lceil \log n \rceil$ control inputs, and n "value" inputs whose values are shifted by the amount specified by the control inputs. Our efficient branching program is a tree program (see Fig. 10.13) that reads the control inputs and selects one of n paths through the tree. (Note that $n \leq 2^{\lceil \log_2 n \rceil} \leq 2n$.) Each path corresponds to one of the n possible cyclic shifts of the n value inputs. Attached to a leaf of this tree is a chain of vertices, one per value input. These inputs appear in the order specified by the cyclic shift associated with the path. An input value is read and then produced as output at each of these n vertices. Since this branching program has at most $2n + 2n^2$ vertices, it has space $O(\log n)$. It uses time $n + \lceil \log n \rceil$.

If cyclic shifting is to be done by a straight-line program, say in hardware, then it is better to use the pebble game for lower bounds since this model applies to logic circuits and the results it provides are stronger. However, if the problem is to be executed in software, the branching program should be used unless the program is straight-line.

10.10.2 Efficient Branching Programs for Merging

Consider now the merging problem. In Section 10.5.6 we show that it requires an $\Omega(n^2)$ space–time product where n is the size of the input. However, when executed by a branching program it uses space $O(\log n)$ and time $O(n)$, as we show.

Figure 10.11 shows a "pyramid" decision branching program to merge two sequences of length two. It is straightforward to extend this decision branching program to sequences of length n, as suggested in Fig. 10.16. In this figure vertices are labeled by the number of elements that are removed from the two lists being merged before arriving at the vertex carrying the label. For example, prior to arriving at the vertex labeled $(2, 1)$, two elements have been removed from the left list and one from the right list. We assume that the lists to be merged each contain n elements. Thus, all the pyramid vertices below a vertex labeled with (n, k) or (k, n), $1 \leq k \leq n - 1$, are deleted because below such vertices no further comparisons are needed; the outputs produced are those on the list from which k values have been removed. Thus, we attach a chain of $n - k$ vertices, one for each of the input values at the end of the smaller list. If the root is at level 1, vertices labeled (n, k) and (k, n) are at level $n + k + 1 \leq 2n + 1$.

The number of vertices on level l of this decision branching program is at most l. Since $1 \leq l \leq 2n$, it has at most $\sum_{l=1}^{2n+1} l = (n + 1)(2n + 1)$ vertices. The space associated with this program is $O(\log(n + 1)(2n + 1))$. Since the length of the longest path in this program is $2n$, it has time $2n$ associated with it. From Lemma 10.9.2 it follows that merging can be

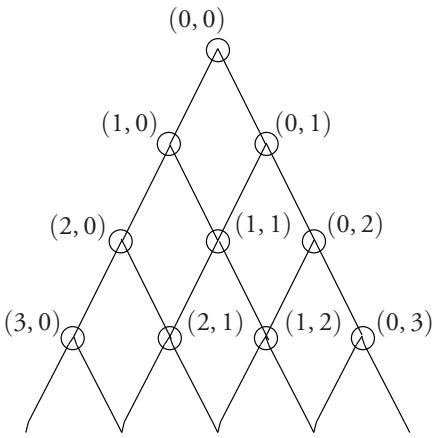


Figure 10.16 The top portion of a decision branching program to merge two sorted lists. The pair of integers at a vertex denotes the number of elements removed from the left and right lists by the program before arriving at the vertex carrying the pair.

realized by a general branching program with space $O(\log n) + \log |\mathcal{A}|$ and time $O(n)$ or a space–time product that is $O(n \log n)$, much smaller than the $O(n^2)$ space–time product that applies to the pebble game.

10.11 The Borodin-Cook Lower-Bound Method

In this section we generalize the method of Borodin and Cook [53] for deriving space-time lower bounds for branching programs. The conditions under which lower bounds can be derived are captured by a property of functions called $(\phi, \lambda, \mu, \nu, \tau)$-distinguishability, which is stronger than the flow property used to derive lower bounds on space-time tradeoffs for the pebble game. In fact, we show that a function that is $(1, \lambda, \mu, \nu, \tau)$-distinguishable is (α, n, m, p)-independent for the appropriate values of α, n, m, and p.

DEFINITION 10.11.1 *Let $\tau : \mathbb{N} \mapsto \mathbb{N}$ be a nondecreasing function. A function $f : \mathcal{A}^n \mapsto \mathcal{F}^m$ is* **$(\phi, \lambda, \mu, \nu, \tau)$-distinguishable** *for $0 \leq \phi, \lambda, \mu, \nu \leq 1$ if there is a set $\mathcal{D} \subset \mathcal{A}^n$ satisfying $|\mathcal{D}| \geq \phi |\mathcal{A}|^n$ such that for each assignment to a selection of $a \leq \lambda n$ input variables and each assignment to a selection of $b \leq \mu m$ output variables of f, $a \leq \tau(b)$, the number of input n-tuples consistent with the values of the a input variables that cause f to assume the given values for the b output variables is at most $|\mathcal{A}|^{n-a-\nu b}$.*

The meaning of this property for the function f is suggested by Fig. 10.17. For a fraction of ϕ of the input tuples ($\phi = 1$ is the normal case), when any a input variables and any b output variables of f are assigned values, the maximum number of input n-tuples that cause f to produce these output values is no more than $|\mathcal{A}|^{n-a-\nu b}$. This property is used below to derive a lower bound on the space-time product for branching programs. We use $\phi = 1$ for all problems considered below except for the unique elements problem.

This theorem also uses a version of the pigeonhole principle. Time is subdivided into intervals containing equal numbers of input queries. This has the effect of chopping the

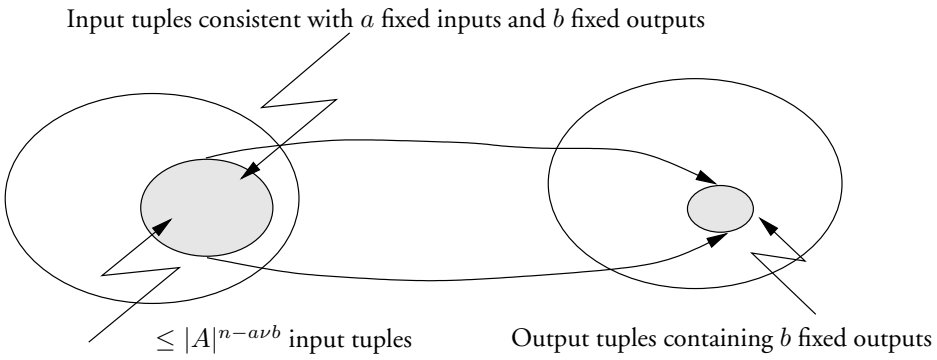


Figure 10.17 For a fraction of at least ϕ of the input n-tuples, an $(\phi, \lambda, \mu, \nu, \tau)$-distinguishable function f has an upper limit of $|\mathcal{A}|^{n-a-\nu b}$ on the number of input n-tuples consistent with an assignment of values to any a inputs and any b outputs of f when $a \leq \lambda n$, $b \leq \mu m$ and $a \leq \tau(b)$.

branching program up into layers (called stages in the proof). We reason that each input n-tuple follows a rich path through a layer that contains a large number of outputs. Because of the distinguishability property, an upper limit on the number of inputs can be associated with each rich path. It follows that there must be many rich paths or that the branching program must have a large number of vertices (and space).

THEOREM 10.11.1 *Let $f : \mathcal{A}^n \mapsto \mathcal{F}^m$ be $(\phi, \lambda, \mu, \nu, \tau)$-distinguishable for $\lambda \leq \mu$. Then the space S and time $T \geq n$ required by any general branching program $\mathcal{P}$ that computes f must satisfy*

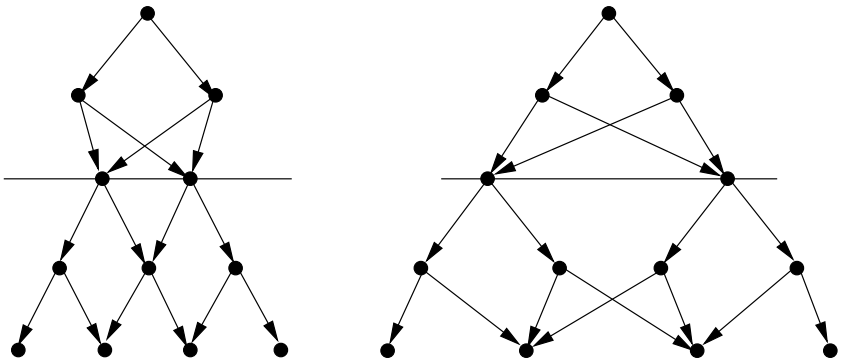
$$S \geq \frac{m\nu a}{4T} \log_2 |\mathcal{A}| + \frac{1}{2} \log_2 \phi$$

where $a \leq \lambda n$ is the largest integer satisfying $a \leq \tau(ma/2T)$ and $n > (\lceil 1/\lambda \rceil - 2)/(1 - \lambda(\lceil 1/\lambda \rceil - 1))$. (Note that $\log_2 \phi$ is a negative constant.)

Proof We show that $S \geq m\nu a/2T \log_2 |\mathcal{A}| + \log_2 \phi$ for normal-form branching programs and then invoke Lemma 10.9.2 to apply it to a general branching program with space $2S$ and time T.

The approach is to break $\mathcal{P}$ into $\sigma = \lceil (T+1)/(a+1) \rceil$ disjoint stages starting with the root at the zeroth level, each stage of which contains $a + 1$ levels, $a \leq \lambda n$, except possibly for the last, which may have fewer levels. ($\sigma \leq 2T/a$ since $T \geq n \geq 1$.) Each stage has depth a. Thus, the last row in one stage is the first row in the next stage. Each stage except for the first typically has multiple roots. (Figure 10.18(a) shows a branching program with $T = 5$ levels. Since $a = 2$, it is divided into $\sigma = \lceil (T + 1)/(a + 1) \rceil = 2$ layers by the horizontal line. Internal vertices belong to two layers.)

Using a modified version of the technique described on page 491 to create a tree program from a branching program, replace the branching program in each stage by a set of tree programs of depth a, shown in Fig. 10.18(b). Eliminate redundant queries on each path in each tree. Also, pad paths that do not have a queries on them with superfluous but non-redundant queries so that each path through each tree has the same length. A **superfluous**



(a) (b)

Figure 10.18 The transformation of a T-step branching program into a branching program with $\sigma = \lceil (T+1)/(a+1) \rceil$ layers in which each layer consists of a forest of trees.

query has all of its output edges directed to a single successor vertex. Also, move all tree outputs down to the leaves of these trees (which are also roots of trees in the next stage). Let $\mathcal{P}^*$ be the new branching program. Since the roots of trees in each stage are vertices in the original branching program, there are no more than 2^S trees.

Let $\boldsymbol{x}$ be one of the input n-tuples among the fraction ϕ for which $(\phi, \lambda, \mu, \nu, \tau)$-distinguishability is defined. The path through $\mathcal{P}^*$ defined by $\boldsymbol{x}$ passes through σ stages. Therefore, there must be at least one stage containing a tree path that produces at least $b = \lceil m/\sigma \rceil$ outputs (a **rich path**). (As shown in the last paragraph of this proof, $b \leq \lceil \mu m \rceil$ when $\lambda \leq \mu$ for sufficiently large n.) Thus, $\boldsymbol{x}$ defines at least one rich path. Let $a \leq \tau(b)$. Because the function $f : \mathcal{A}^n \mapsto \mathcal{F}^m$ is $(\phi, \lambda, \mu, \nu, \tau)$-distinguishable, each rich path can be associated with at most $|\mathcal{A}|^{n-a-\nu b}$ inputs. (This number is smaller if more than b outputs are produced.) Since there are at most 2^S trees and at most $|\mathcal{A}|^a$ paths through each tree, there are at most $2^S|\mathcal{A}|^a$ rich paths. Furthermore, two distinct rich paths (either the inputs queried or outputs produced are different) are associated with disjoint sets of input n-tuples. Thus, $2^S|\mathcal{A}|^a|\mathcal{A}|^{n-a-\nu b}$ cannot be less than the number of input n-tuples in question, from which the following inequality holds:

$$\phi|\mathcal{A}|^n \leq 2^S|\mathcal{A}|^a|\mathcal{A}|^{n-a-\nu b}$$

We conclude that

$$S \geq \nu b \log_2 |\mathcal{A}| + \frac{1}{2}\log_2 \phi$$

We replace $b = \lceil m/\sigma \rceil$ by its lower bound $ma/2T$. Since $\tau(b)$ is a nondecreasing function, the value of a satisfying $a \leq \tau(b)$ is not increased by replacing b by $ma/2T$. Thus, $S \geq \nu(ma/2T)\log_2 |\mathcal{A}| + \log_2 \phi$, subject to $a \leq \tau(ma/2T)$ and $a \leq \lambda n$.

We show there exists an integer n_α such that for $n > n_\alpha$ the condition $b \leq \lceil \mu m \rceil$ is met by the condition $\lambda \leq \mu$. Note that $b = \lceil m/\sigma \rceil$ is a nondecreasing function of a and a nonincreasing function of T since $\sigma = \lceil (T+1)/(a+1) \rceil$ is a nonincreasing

function of a and a nondecreasing function of T. Thus, b is largest when $T = n$ and $a = \lambda n$. It follows that b is largest when $\sigma = \lceil (n+1)/(\lambda n+1) \rceil \leq \lceil 1/\lambda \rceil$. If $n > (\lceil 1/\lambda \rceil - 2)/(1 - \lambda(\lceil 1/\lambda \rceil - 1))$, then $(n+1)/(\lambda n+1) > \lceil 1/\lambda \rceil - 1$, which implies that $\lceil (n+1)/(\lambda n+1) \rceil = \lceil 1/\lambda \rceil$. In other words, when $n > (\lceil 1/\lambda \rceil - 2)/(1 - \lambda(\lceil 1/\lambda \rceil - 1))$, b assumes a value of at most $\lceil m/\lceil 1/\lambda \rceil \rceil \leq \lceil \lambda m \rceil$. ■

COROLLARY 10.11.1 *Let $f : \mathcal{A}^n \mapsto \mathcal{F}^m$ be $(\phi, \lambda, \mu, \nu, \tau)$-distinguishable for $\lambda \leq \mu$ and $\tau(b) = n$. Then the space S and time T required by any normal-form branching program $\mathcal{P}$ that computes f must satisfy*

$$ST \geq \frac{mn\lambda\nu}{2} \log_2 |\mathcal{A}| + \log_2 \phi$$

when $T \geq n$ and $n > (\lceil 1/\lambda \rceil - 2)/(1 - \lambda(\lceil 1/\lambda \rceil - 1))$.

Proof The result follows from the observation that the maximum value of a in Theorem 10.11.1 is λn. ■

The connection between (α, n, m, p)-independence and $(1, \lambda, \mu, \nu, \tau)$-distinguishability is given below.

LEMMA 10.11.1 *If $f : \mathcal{A}^n \mapsto \mathcal{F}^m$ is $(1, \lambda, \mu, \nu, \tau)$-distinguishable, it is $(1/\nu, n, m, p)$-independent for $p = \min(\lambda n, \tau(\mu m)) + \mu m$.*

Proof Consider sets of a input and b output variables to f such that $a \leq \tau(b)$, $a \leq \lambda n$, and $b \leq \mu m$, or equivalently $a \leq \tau^*$, where $\tau^* = \min(\lambda n, \tau(\mu m))$ since $\tau(x)$ is nondecreasing in x. For any particular assignment to the a inputs, the input n-tuples that agree with this assignment but lead to different values for the b outputs must be disjoint, as suggested in Fig. 10.19. We show that for some assignment of values to the a inputs, the number of values assumed by the b outputs is more than $|\mathcal{A}|^{b/\alpha - 1}$ for $\alpha = 1/\nu$. Suppose not. Then there are at most $|\mathcal{A}|^{n-a-\nu b} |\mathcal{A}|^{\nu b - 1}$ input tuples for each assignment to the a inputs, or a total of at most $|\mathcal{A}|^{n-1}$ input tuples. Since f has $|\mathcal{A}|^n$ input tuples, we have a contradiction. Therefore, f is $(1/\nu, n, m, p)$-independent for $p = \tau^* + \mu m$. ■

The following lemma makes it easier to derive space-time lower bounds for branching programs. It uses the notions of subfunction (see Definition 2.4.2) and reduction (see Definition 2.4.1).

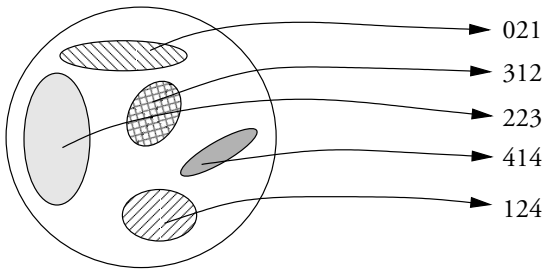


Figure 10.19 On the left are the points in the domain of f that map to individual output b-tuples when the values of a input variables are fixed.

LEMMA 10.11.2 *Let $g : \mathcal{A}^r \mapsto \mathcal{A}^s$ be a reduction of $f : \mathcal{A}^n \mapsto \mathcal{A}^m$ that is either a subfunction or a reduction obtained by restricting f to a subset of its domain. A lower bound to the space-time product ST on branching programs for g is also a lower bound for f.*

Proof Given any branching program for f, we can construct one for g that has no more vertices or longer paths as follows. If g is obtained by deleting outputs, delete these outputs from vertices in the branching program. This may allow the coalescing of vertices. If g is obtained by restricting the set of values that variables of f can assume, this may make some paths and subgraphs inaccessible and therefore removable. If g is obtained by giving two variables of f the same identity, this constrains the branching program and again may make some subgraphs inaccessible. In all cases neither the number of vertices nor the length of any path to a sink vertex is increased by the reduction of f to g. Thus, any lower bound to ST for g must be a lower bound for f. ■

10.12 Properties of "nice" and "ok" Matrices*

In this section we develop properties of matrices that are γ-nice or γ-ok, concepts we now introduce. (A matrix that is γ-nice is also γ-ok.) These properties are used in Section 10.13 to develop lower bounds on the exchange of space for time using the Borodin-Cook method. This section requires a knowledge of probability theory.

DEFINITION 10.12.1 *An $n \times m$ matrix A, $n \leq m$, is* **γ-nice** *for $0 < \gamma < 1/2$ if and only if for all $p \leq \lceil \gamma n \rceil$ and $q \geq n - \lceil \gamma n \rceil$ every $p \times q$ submatrix of A has rank p. Such a matrix is* **γ-ok** *if all such $p \times q$ submatrices have rank at least γp.*

As shown below, most matrices are γ-nice, a fact that is used in several places.

LEMMA 10.12.1 *At least a fraction $(1 - |\mathcal{A}|^{-1}(2/3)^{\gamma n})$ of the $|\mathcal{A}|^{n^2}$ $n \times n$ matrices over a subset $\mathcal{A}$ of a field, $|\mathcal{A}| \geq 2$, are γ-nice for some constant γ, $0 < \gamma < \frac{1}{2}$, independent of n and $\mathcal{A}$. This result also holds for $n \times n$ Toeplitz matrices, matrices $[t_{i,j}]$ with the property that $t_{i,j} = a_{i-j}$; that is, all elements on each diagonal are the same.*

Proof Let $r = \lceil \gamma n \rceil$ and $s = n - r$. The proof is established by deriving upper bounds on the number $N(r, s)$ of $r \times s$ matrices in an $n \times n$ matrix M and the probability $q(r, s)$ that any particular $r \times s$ matrix fails to contain a non-singular $r \times r$ submatrix (it fails to have **rank** r) when each entry in M is equally likely to be an element of $\mathcal{A}$. Since the probability of a union of events is at most the sum of the probabilities of the events, the probability that some $r \times s$ matrix fails to have rank r is at most $q(r, s)N(r, s)$.

It is straightforward to show that

$$N(r, s) = \binom{n}{r}^2$$

since an $r \times s$ submatrix of an $n \times n$ matrix is chosen by selecting a set of r rows and a set of s columns and each can be chosen in $\binom{n}{r}$ ways. (Note that $\binom{n}{s} = \binom{n}{r}$.) We now show that the binomial coefficient $\binom{n}{r}$ is at most $(n/r)^r e^r$. We use the fact that $n!/(n-r)! = n(n-1)\cdots(n-r+1) \leq n^r$ and the observation that $r^r/r!$ is a term in

the Taylor-series expansion of e^r, as stated below:

$$\binom{n}{r} = \frac{n!}{r!(n-r)!} \leq \frac{n^r}{r!} = \left(\frac{n}{r}\right)^r \frac{r^r}{r!} \leq \left(\frac{n}{r}\right)^r e^r$$

Later we show that $q(r,s) \leq \rho^{-s}|\mathcal{A}|^{r-1}$, where $\rho = |\mathcal{A}|^2/(2|\mathcal{A}|-1) \leq 2|\mathcal{A}|/3$, from which it follows that

$$\begin{aligned} q(r,s)N(r,s) &\leq |\mathcal{A}|^{-1}\left(\frac{en}{r}\right)^{2r}\rho^{-n}\rho^r|\mathcal{A}|^r \\ &\leq |\mathcal{A}|^{-1}\left(\frac{2}{3}\right)^r\left[\rho^{-n}\left(\frac{en|\mathcal{A}|}{r}\right)^{2r}\right] \end{aligned}$$

since $s = n - r$. Elementary calculus shows that $(e|\mathcal{A}|/r)^{2r}$ is an increasing function of r and that it has value 1 at $r = 0$. Since $r = \lceil \gamma n \rceil$ and $\rho \geq 4/3$, it follows that the quantity in square brackets is less than 1 for some value of $0 < \gamma < 1/2$, which is the desired conclusion.

We now give a proof by induction that $q(r,s)$ satisfies $q(r,s) \leq \rho^{-s}|\mathcal{A}|^{r-1}$. Clearly $q(1,1) \leq 1/|\mathcal{A}|$, since at most one entry in $\mathcal{A}$ is zero. This satisfies the bound. We now assume the inductive hypothesis holds for $q(r-1,s-1)$ and $q(r,s-1)$ and show that it holds for $q(r,s)$.

Consider an $r \times s$ matrix B. It has rank r if the submatrix consisting of the first $s-1$ columns has rank r. (This occurs with probability $1 - q(r,s-1)$.) If this is not the case, there are many other ways in which it can have rank r. In particular, this is true if the submatrix C consisting of the last $r-1$ rows and the first $s-1$ columns of B has rank $r-1$ (with probability $1 - q(r-1,s-1)$) and the element $b_{1,s}$ has an appropriate value (with probability at least $1 - 1/|\mathcal{A}|$), as we now show.

Consider a submatrix D consisting of some $r-1$ linearly independent columns of C. Consider the $r \times r$ submatrix of B consisting of these same $r-1$ columns and its last column. When the determinant of this matrix is expanded on the first row, the multiplier of $b_{1,s}$ is ± 1 times the determinant of D, which is non-zero. Thus, there is at most one value for $b_{1,s}$ that causes the determinant to be zero (the field element causing it to be zero may not be in the set $\mathcal{A}$) or at least $|\mathcal{A}| - 1$ values that cause it to be non-zero. Summarizing this result, we have the following lower bound:

$$\begin{aligned} 1 - q(r,s) &\geq 1 - q(r,s-1) + (1 - q(r-1,s-1))\left(1 - \frac{1}{|\mathcal{A}|}\right) \\ &\geq (1 - q(r,s-1))\frac{1}{|\mathcal{A}|} + (1 - q(r-1,s-1))\left(1 - \frac{1}{|\mathcal{A}|}\right) \end{aligned}$$

This implies that

$$\begin{aligned} q(r,s) &\leq q(r,s-1)\frac{1}{|\mathcal{A}|} + q(r-1,s-1)\left(1 - \frac{1}{|\mathcal{A}|}\right) \\ &\leq \rho^{-s+1}|\mathcal{A}|^{r-1}\frac{1}{|\mathcal{A}|}\left(2 - \frac{1}{|\mathcal{A}|}\right) \\ &\leq \rho^{-s}|\mathcal{A}|^{r-1} \end{aligned}$$

which is the desired conclusion.

The proof also holds for Toeplitz matrices (each element on a diagonal of the matrix is the same) because we reasoned only about the value of elements in the upper right-hand corner of submatrices that are on different diagonals. ■

The Kronecker product of matrices is used in Section 10.13.5 to derive a lower bound on the space-time product for matrix inversion.

DEFINITION 10.12.2 *The* **Kronecker product** *of two $n \times n$ matrices A and B is the $n^2 \times n^2$ matrix C, denoted $C = A \otimes B$, obtained by replacing the entry $a_{i,j}$ of A with the matrix $a_{i,j}B$.*

A Kronecker product $C = A \otimes B$ of matrices A and B is shown below:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \qquad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}, \qquad C = \begin{bmatrix} 5 & 6 & 10 & 12 \\ 7 & 8 & 14 & 16 \\ 15 & 18 & 20 & 24 \\ 21 & 24 & 28 & 32 \end{bmatrix}$$

The following property of the Kronecker product of two γ-nice matrices is used to derive the space-time lower bounds stated in Theorem 10.13.5.

LEMMA 10.12.2 *If A and B are both $n \times n$ γ-nice matrices for some $0 \leq \gamma \leq 1/2$, then $C = A \otimes B$ is an $n^2 \times n^2$ γ^2-ok matrix.*

Proof Number the rows and columns of A, B, and C consecutively from 0. For a matrix E, extend the notation $e_{i,j}$ for the entry in the ith row and jth column of E to $e_{I,J}$, by which we denote the submatrix of E consisting of the intersection of the rows in the set I and columns in the set J. Thus, if $I = \{i\}$ and $J = \{j\}$, then $e_{I,J} = e_{i,j}$.

To show that C is γ^2-ok, we must show that every $p \times q$ submatrix S of C satisfying $p \leq \lceil \gamma^2 n^2 \rceil$ and $q \geq n - \lceil \gamma^2 n^2 \rceil$ has rank at least $\gamma^2 p$. Such a matrix S can be represented as $S = c_{I,J}$ for index sets I and J, where $p = |I| \leq \lceil \gamma^2 n^2 \rceil$ and $q = |J| \geq n - \lceil \gamma^2 n^2 \rceil$. We assume that $\gamma n \geq 1$, since otherwise the result holds trivially.

The rth **block row** of C is the submatrix $[a_{r,0}B, a_{r,1}B, \ldots, a_{r,n-1}B]$ containing rows numbered $I_r = \{rn, rn+1, \ldots, rn+n-1\}$ and all n^2 columns.

Let $\Delta_r = I \cap \{rn, rn+1, \ldots, rn+n-1\}$ be the indices of the rows of S that fall into the rth block row. Choose a set $\Gamma \subset \{0, 1, 2, \ldots, n-1\}$ of size $|\Gamma| = \lceil \gamma n \rceil$ that maximizes the sum $T = \sum_{r \in \Gamma} |\Delta_r|$. Then, $T \geq \gamma p$ because the lower bound is achieved if the rows of S are uniformly distributed over the rows of C and T is larger if they are not.

Let $\Lambda_r = \Delta_r$ if $|\Delta_r| \leq \lceil \gamma n \rceil$ and let Λ_r consist of the smallest $\lceil \gamma n \rceil$ indices in Δ_r otherwise. Clearly, $|\Lambda_r| \geq |\Delta_r|\gamma$ because Δ_r is chosen from a set of size n. Call rows of C with indices in $\bigcup_{r \in \Gamma} \Lambda_r$ **blue rows**. There must be at least $\gamma^2 p$ blue rows because, if not,

$$\gamma^2 p > \sum_{r \in \Gamma} |\Lambda_r| \geq \sum_{r \in \Gamma} |\Delta_r|\gamma = \gamma T \geq \gamma^2 p$$

which is a contradiction.

We now show that the blue rows of S are linearly independent. Suppose not. Then there exist constants $\{\alpha_{r,s} \mid r \in \Gamma, s \in \Delta_r\}$ not all of which are zero such that the linear

combination of the blue rows of S is zero:

$$\sum_{r\in\Gamma}\sum_{s\in\Lambda_r}\alpha_{r,s}c_{nr+s,J}=\mathbf{0} \tag{10.7}$$

Here $\mathbf{0}$ is a column vector of zeros, one per blue row. Again, J is the set of columns of C in the submatrix S.

Column j of the $n\times n$ matrix B is *good* if it is associated with at least $(1-\gamma)n$ columns of S and is *bad* otherwise. Let G be the indices of the good columns in B and let $g=|G|$. Then there are $g\geq(1-\gamma)n$ good columns and $b\leq\gamma n$ bad columns in B ($g+b=n$) because, if not, $g\leq(1-\gamma)n-1$ and the number of columns altogether in S is at most $gn+b(1-\gamma)n$, which is an increasing function of g whose value is less than $n^2-\lceil\gamma^2\,n^2\rceil$ when $g\leq(1-\gamma)n-1$, which is less than the number of columns of S.

Since B has at least $g=|G|\geq(1-\gamma)n$ good columns and B is γ-nice, any set of up to $\lceil\gamma n\rceil$ rows are linearly independent. In particular, the rows of B indexed by Λ_r are linearly independent. This implies that

$$\sum_{s\in\Lambda_r}\alpha_{r,s}b_{s,G}\neq\mathbf{0}$$

where $\mathbf{0}$ is a zero column with $|\Lambda_r|$ rows. Thus, there must be a column index $t\in G$ such that

$$\sum_{s\in\Lambda_r}\alpha_{r,s}b_{s,t}\neq 0 \tag{10.8}$$

Let $K=\{j\mid nj+t\in J\}$ be the columns of S corresponding to the good column of B with index t. It follows that $|K|\geq\lfloor(1-\gamma)n\rfloor$.

Let $\boldsymbol{u}_i=c_{i,J\cap K}$, the intersection of the ith row of S with columns whose indices are in K. Similarly, let $\boldsymbol{v}_i$ be the intersection of the ith row of A with columns in K. It follows from the definition of C that $\boldsymbol{u}_{ni+j}=b_{j,t}\boldsymbol{v}_i$. From (10.7) we have that

$$\sum_{r\in\Gamma}\sum_{s\in\Lambda_r}\alpha_{r,s}c_{nr+s,J\cap K}=\mathbf{0}$$

$$\sum_{r\in\Gamma}\left(\sum_{s\in\Lambda_r}\alpha_{r,s}b_{s,t}\right)\boldsymbol{v}_r=\mathbf{0}$$

However, the rows $|\Gamma|$ rows $\boldsymbol{v}_r$ constitute a $\lceil\gamma n\rceil\times|K|$ submatrix of the γ-nice matrix A where $|K|\geq\lfloor(1-\gamma)n\rfloor$. Since its rows are linearly independent, each of the coefficients $\sum_{s\in\Lambda_r}\alpha_{r,s}b_{s,t}$ must be zero, contradicting the statement of (10.8). It follows that $C=A\otimes B$ is γ^2-ok. ■

10.13 Applications of the Borodin-Cook Method

In this section we illustrate the Borodin-Cook method of Section 10.11 by applying it to a variety of representative problems.

10.13.1 Convolution

The wrapped convolution function $f^{(n)}_{\text{wrapped}} : \mathcal{R}^{2n} \mapsto \mathcal{R}^n$ over the ring $\mathcal{R}$ (see Problem 6.19) of two sequences $\boldsymbol{u}$ and $\boldsymbol{v}$ is described by the matrix-vector product $C\boldsymbol{v}$ of a circulant matrix C in which $c_{i,j} = u_{(i-j) \bmod n}$, as shown in Section 10.5.1.

LEMMA 10.13.1 *For n even, the wrapped convolution $f^{(n)}_{\text{wrapped}} : \mathcal{R}^{2n} \mapsto \mathcal{R}^n$ over the ring $\mathcal{R}$ contains a subfunction $g^{(n)} : \mathcal{R}^{2n} \mapsto \mathcal{R}^{n/2}$ that is $(1, \gamma/2, \gamma/2, 1, 2n)$-distinguishable for some $0 < \gamma < 1/2$.*

Proof Writing C as a 2×2 matrix of $n/2 \times n/2$ matrices, we find that its (1,1) entry is an unrestricted Toeplitz matrix T. That is, each diagonal can contain a different element. Consider the subfunction of $f^{(n)}_{\text{wrapped}}$ defined by this submatrix. By Lemma 10.12.1, a fraction of at least $1 - (2/3)^{(\gamma/2)n}/|\mathcal{R}|$ of such matrices are γ-nice. By Definition 10.12.1, this implies that $\lceil(\gamma/2)n\rceil$ output variables assume $|\mathcal{R}|^{\lceil(\gamma/2)n\rceil}$ different values. If we fix the entries of T to be those of a γ-nice matrix, by Lemma 10.11.2 the lower bound on ST for matrix-vector multiplication with a Toeplitz matrix with n replaced by $n/2$ serves as a lower bound for the original problem. Since for large n most Toeplitz matrices are γ-nice, we have the desired conclusion. ■

Invoking Theorem 10.11.1, we have the space–time lower bound stated below. The upper bound follows from the design of a branching program to implement the inner product operation, as suggested by Fig. 10.6.

THEOREM 10.13.1 *There is an integer $n_0 > 0$ such that for n even and $n \geq n_0$, the time T and space S used by any general branching program for the wrapped convolution $f^{(n)}_{\text{wrapped}} : \mathcal{R}^{2n} \mapsto \mathcal{R}^n$ over the ring $\mathcal{R}$ must satisfy*

$$ST = \Omega(n^2 \log |\mathcal{R}|) \tag{10.9}$$

Branching programs exist that achieve the following bound for $\log |\mathcal{R}| \leq S \leq n \log |\mathcal{R}|$:

$$ST = O(n^2 \log n \log |\mathcal{R}|)$$

Proof Since the wrapped convolution function depends on $2n$ variables, it can be computed via table lookup with space $O(n \log |\mathcal{R}|)$ and time $O(n)$.

At the limit of small space, namely for $S = \Theta(\log |\mathcal{R}|)$, a branching program can be designed that computes the n inner products defined by the matrix-vector product of (10.1). An example of a branching program to compute the inner product of two 3-vectors is shown in Fig. 10.20. A branching program for the inner product of two n-tuples can be constructed that has $O(n|\mathcal{R}|^2)$ vertices and depth $O(n)$. Hence, a branching program to multiply a general $n \times n$ matrix by a vector can be constructed that has time $O(n^2)$ and space $O(\log n + \log |\mathcal{R}|)$.

To fill in the range between these extremes, let k divide n and note that the product of an $n \times n$ matrix by a column n-vector can be viewed as the product of an $n/k \times n/k$ matrix of $k \times k$ matrices with a column n/k-vector of column k-vectors. Since each product of a $k \times k$ submatrix by a k-vector is a function of $O(k)$ parameters, compute it with table lookup in time $O(k)$ and space $O(k \log |\mathcal{R}|)$. Add two of these matrix-vector products by

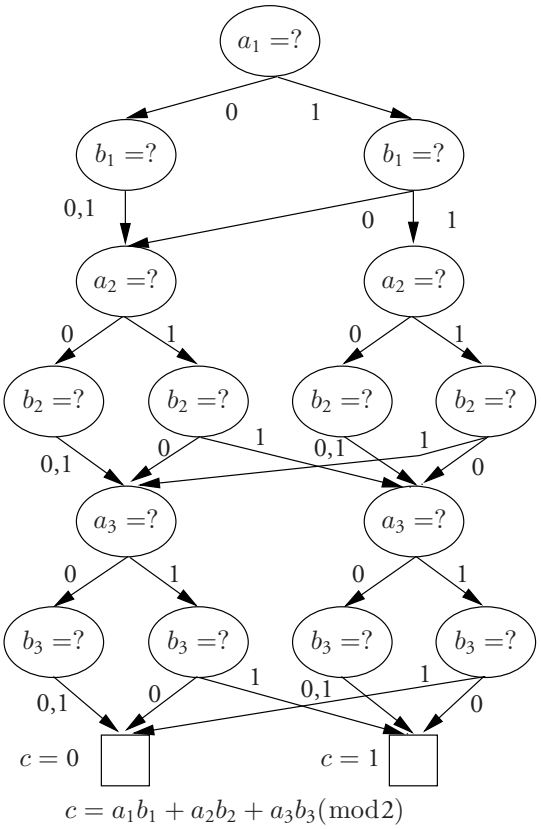


Figure 10.20 A branching program to compute the inner product of two 3-vectors over the set $\mathcal{R}$ of integers modulo 2.

rooting a table-lookup program at each of the $O(|\mathcal{R}|^k)$ final states of a first table-lookup program. Coalesce final states corresponding to the $|\mathcal{R}|^k$ sums of the two column k-vectors. This program has $O(|\mathcal{R}|^{2k})$ vertices or space $O(k \log |\mathcal{R}|)$ and time $O(k)$. n/k such stages increase the number of vertices and time each by a factor of n/k. Since this process is then repeated for each of the n/k rows of the block matrix, the space and time used are $O(k \log |\mathcal{R}| + \log(n/k))$ and $O(n^2/k)$, respectively. ■

10.13.2 Integer Multiplication

To derive space–time lower bounds for integer multiplication, we could invoke the reductions from this problem to cyclic shifting, as was done in Section 10.5.3. However, as shown in Section 10.10, the space–time product for cyclic shifting is only $O(n \log n)$. Thus, we are forced to use another reduction to obtain a strong space–time product lower bound, namely a reduction from integer multiplication to convolution.

Let $\mathbb{Z}_2$ be the ring of integers modulo 2. As shown in Problem 6.20, the integer multiplication function $f^{(n)}_{\text{mult}} : \mathcal{B}^{2n} \mapsto \mathcal{B}^{2n}$ contains the convolution function over $f^{(n/\log n)}_{\text{conv}} : \mathbb{Z}_2^{2n/\log n} \mapsto \mathbb{Z}_2^{2n/\log n}$. Thus, by Lemmas 10.11.2 and 10.13.1 the following holds:

THEOREM 10.13.2 *There is an integer $n_0 > 0$ such that for $n > n_0$ the time T and space S used by any general branching program for binary integer multiplication $f^{(n)}_{\text{mult}} : \mathcal{B}^{2n} \mapsto \mathcal{B}^{2n}$ must satisfy*

$$ST = \Omega(n^2/\log^2 n) \tag{10.10}$$

This lower bound can be achieved to within a factor of $O(\log^3 n)$ for space $\Omega(\log n) \leq S \leq O(n)$.

Proof Since the integer multiplication function depends on $2n$ variables, it can be computed via table lookup with space $O(n)$ and time $O(n)$, thereby meeting the lower bound to within a factor of $O(\log^2 n)$.

At the limit of small space, $S = \Theta(\log n)$, the integer multiplication algorithm of Section 10.5.3 provides a branching program. Since at most $\lceil \log_2 n \rceil$ bits suffice for the carry from one power of 2 to the next, a branching program based on this algorithm has at most $O(2^{\lceil \log_2 n \rceil})$ vertices at each of n^2 levels. Thus, this program uses time $O(n^2)$ and space $O(\log n)$, achieving the lower bound to within a factor of $O(\log n)$.

We sketch a procedure to fill in the range of space between these extremes and ask the reader to complete the details. (See Problem 10.39.) Assume that k divides n and represent each n-bit binary number as an (n/k)-component base-2^k number. As in the standard binary integer multiplication algorithm (where $k = 1$), form n/k (n/k)-component numbers through multiplication and shifting of consecutive base-2^k components, as suggested below:

$$\begin{array}{ccccccc}
 & & & v_3u_0 & v_2u_0 & v_1u_0 & v_0u_0 \\
 & & v_3u_1 & v_2u_1 & v_1u_1 & v_0u_1 & 0 \\
 & v_3u_2 & v_2u_2 & v_1u_2 & v_0u_2 & 0 & 0 \\
v_3u_3 & v_2u_3 & v_1u_3 & v_0u_3 & 0 & 0 & 0
\end{array}$$

Here u_r and v_s are base-2^k numbers. Multiply two such numbers through table lookup in time and space $O(k)$. Extend the algorithm for the base-2 case by replacing each subprogram that multiplies two binary numbers by the table lookup program to multiply base-2^k numbers. This new program adds products to a running sum of length $O(\log n)$ bits. Thus, it uses space $O(k + \log n)$ and time $O(n^2/k)$, giving a space–time product of $O(n^2 \log n)$ for $k \geq \log n$. ■

10.13.3 Matrix-Vector Product

The matrix-vector product function $f^{(n)}_{A\times\boldsymbol{x}} : \mathcal{R}^n \mapsto \mathcal{R}^n$ computes the n-tuple $\boldsymbol{y}$ from the n-tuple $\boldsymbol{x}$ for a fixed $n \times n$ matrix A over $\mathcal{R}$ according to the rule

$$\boldsymbol{y} = A\boldsymbol{x}$$

where $y_j = \sum_{k=0}^{n-1} a_{j,k}x_k$ for $0 \leq j \leq n-1$.

LEMMA 10.13.2 *Let A be a γ-ok $n \times n$ matrix over $\mathcal{R}$ for some $0 < \gamma < 1/2$. Then the matrix-vector product function $f^{(n)}_{A\times \boldsymbol{x}} : \mathcal{R}^n \mapsto \mathcal{R}^n$ is $(1, \gamma, \gamma, \gamma, \tau)$-distinguishable where $\tau(b) = n$.*

Proof To show that $f^{(n)}_{A\times \boldsymbol{x}}$ is $(1, \gamma, \gamma, \gamma, \tau)$-distinguishable, select any $a \leq \lceil \gamma n \rceil$ inputs and any $b \leq \lceil \gamma n \rceil$ outputs. If the ith input is chosen and it has value u_i, introduce the equation $x_i = u_i$. Let B be the $a \times n$ coefficient matrix defining these equations; that is, $B\boldsymbol{x} = \boldsymbol{u}$, where B contains the jth row of the $n \times n$ identity matrix if the jth variable is among the selected inputs.

Consider the $(n + a) \times n$ matrix $C = \begin{bmatrix} A \\ B \end{bmatrix}$. We show that it has rank $a + \gamma b$. The submatrix D of A consisting of the intersection of those columns not selected by inputs (of which there are $n - a \geq n - \lceil \gamma n \rceil$) and rows selected by outputs (of which there are b) has rank γb because A is γ-ok. Thus, γb of the $n - a$ columns of A not selected by inputs and the a non-zero columns of B are linearly independent. Thus, the submatrix E of C consisting of the selected rows of B and the rows of D has rank $a + \gamma b$.

The number of n-tuple input vectors $\boldsymbol{x}$ consistent with the linear system $E\boldsymbol{x} = \boldsymbol{d}$ is $|\mathcal{A}|^{n-a-\gamma b}$, as we show. Without loss of generality assume that the first $a+\gamma b$ columns of E (call it F) are linearly independent. (Permute the columns, if necessary, so that this is true.) Fix the values of the b realizable outputs. Then for each assignment to inputs corresponding to the last $n - (a + \gamma b)$ columns there are unique values for the first $a + \gamma b$ inputs, due to the non-singularity of F. Thus the number of assignments to the last $n - (a + \gamma b)$ columns that are consistent with values for the a inputs and b outputs is $|\mathcal{A}|^{n-a-\gamma b}$. ■

Invoking Corollary 10.11.1 yields the following result.

THEOREM 10.13.3 *Let A be a γ-ok $n \times n$ matrix over $\mathcal{R}$ for some $0 < \gamma < 1/2$. Then there is a constant $0 < \gamma < 1/2$ and an integer n_0 such that for $n \geq n_0$ the space S and time T used by any general branching program for the function $f^{(n)}_{A\times \boldsymbol{x}} : \mathcal{R}^n \mapsto \mathcal{R}^n$ must satisfy the following lower bound when $T \geq n$:*

$$ST = \Omega(n^2 \log |\mathcal{R}|)$$

This lower bound can be met to within a factor of $O(\log n)$ for $\log n \leq S \leq n$.

Proof The lower bound follows from the application of Theorem 10.11.1.

The matrix-vector product $A\boldsymbol{x}$ for an $n \times n$ matrix A can be done with a branching program for the standard algorithm as follows: Compute the inner product of the ith row with the column $\boldsymbol{x}$ for $1 \leq i \leq n$. The inner product of two n-tuples can be computed with a branching program having $O(n|\mathcal{R}|^2)$ vertices, as suggested in Fig. 10.20. (This is true even if A is not fixed.) n branching programs for inner products can be concatenated to form one branching program to multiply an $n \times n$ matrix with an n-vector. This branching program uses space $O(\log n + \log |\mathcal{R}|)$ and time $O(n^2)$, thereby meeting the lower bound to within a factor of $O(\log n)$.

A matrix-vector product for a fixed matrix (this case) can also be computed by table lookup in space $O(n \log |\mathcal{R}|)$ and time $O(n)$ since this function has n variables.

To bridge the gap between these two results, compute the matrix-vector product using a hybrid algorithm similar to that used for convolution in the proof of Theorem 10.13.1. ■

10.13.4 Matrix Multiplication*

The space–time lower-bound argument for matrix multiplication in the branching program model uses ideas similar to those used for matrix-vector multiplication.

LEMMA 10.13.3 *The matrix multiplication function* $f^{(n)}_{A\times B} : \mathcal{R}^{2n^2} \mapsto \mathcal{R}^{n^2}$ *over the ring* $\mathcal{R}$ *is* $(1, 1, 1, \gamma/4, \tau)$*-distinguishable for some* $0 < \gamma < 1/2$*, where* $\tau(b) = \gamma n\sqrt{b/2}$.

Proof Consider the subfunction of $f^{(n)}_{A\times B}$ obtained by choosing A and B from the set of $n \times n$ γ-nice matrices. By Lemma 10.11.2, a lower bound on the space–time product for this subfunction provides a lower bound to the matrix multiplication function.

Consider some $a \leq 2n^2$ selected inputs and some $b \leq n^2$ selected outputs such that $a \leq \tau(b)$; that is, $(a/\gamma n)^2 \leq b/2$. The outputs correspond to entries of the product matrix $C = A \times B$. Let row i of C be a **heavy row** if at least γn of the a selected inputs are in row i of A. Similarly, let column j of C be a **heavy column** if at least γn of the a selected inputs are in column j of B. A row or column of C is **light** otherwise. (See Fig. 10.21.)

There are at most $a/\gamma n$ heavy rows and $a/\gamma n$ heavy columns of C. We now show that either a) at least $b/4$ of the selected outputs fall into light rows of C or b) at least $b/4$ of the selected outputs fall into light columns of C. Suppose not. Then both statements are false and less than $b/4$ of the selected outputs fall into light rows and less than $b/4$ of the selected outputs fall into light columns of C. It follows that at least $3b/4$ of the selected outputs fall into heavy rows. Of these at most $(a/\gamma n)^2$ fall into heavy columns, since this is the maximum number of entries of C that could be in both heavy rows and columns. The remaining selected outputs in these rows (of which there are less than $b/4$) fall into light columns. However, because the entries in each row fall into either heavy or light columns, the number of selected outputs that are in heavy rows is less than $(a/\gamma n)^2 + b/4$. But this is less than $3b/4$ since $a \leq \tau(b) = \gamma n\sqrt{b/2}$, contradicting the stated hypothesis.

Without loss of generality, assume that b holds. (If not, a holds and at least $b/4$ selected outputs fall into light rows of C or into light columns of the transpose C^T.) Represent the

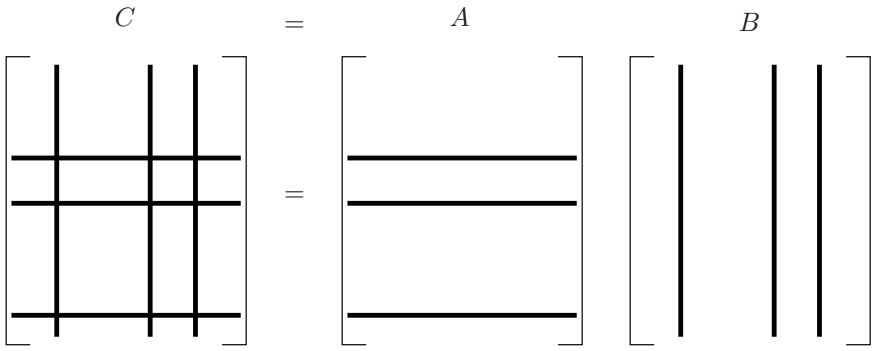


Figure 10.21 Identification of heavy rows and columns of matrices.

product $C = A \times B$ as follows:

$$\begin{bmatrix} A & & \\ & \ddots & \\ & & A \end{bmatrix} \begin{bmatrix} \boldsymbol{B^1} \\ \vdots \\ \boldsymbol{B^n} \end{bmatrix} = \begin{bmatrix} \boldsymbol{C^1} \\ \vdots \\ \boldsymbol{C^n} \end{bmatrix}$$

Here $\boldsymbol{B^i}$ and $\boldsymbol{C^i}$ are the ith columns of the matrices B and C, respectively. Let $\boldsymbol{B}$ and $\boldsymbol{C}$ denote the columns of these columns, respectively, and let $\boldsymbol{D}$ denote the block diagonal matrix on the left.

We show that at most $|\mathcal{R}|^{2n^2-a-\gamma b/4}$ of the matrix pairs (A, B) are consistent with any assignment to any set of a selected inputs and values of any b selected outputs.

Of the a selected inputs, let a_1 be drawn from A and a_2 be drawn from B, where $a = a_1 + a_2$. The number of γ-nice matrices A consistent with the a_1 selected inputs from A is at most $|\mathcal{R}|^{n^2-a_1}$. We now bound the number of matrices B that are consistent with the values of selected inputs and outputs.

Let A be fixed and γ-nice. Consider just the (at least $b/4$) selected outputs that fall into light columns of C. Every value for B consistent with the selected inputs and these outputs must satisfy the following linear equation:

$$\begin{bmatrix} E \\ F \end{bmatrix} \boldsymbol{B} = H\boldsymbol{B} = \begin{bmatrix} \boldsymbol{r} \\ \boldsymbol{c} \end{bmatrix}$$

Here E consists of the b rows of $\boldsymbol{D}$ corresponding to selected outputs and F is a submatrix of the $n^2 \times n^2$ identity matrix consisting of the a_2 rows corresponding to selected inputs in B. $\boldsymbol{c}$ is the column of values for the selected inputs in $\boldsymbol{B}$ and $\boldsymbol{r}$ is a column of selected outputs of C that fall into light columns. The number of values for B consistent with a fixed A and the values of the selected inputs and outputs is no more than the number of solutions $\boldsymbol{B}$ to these equations, since we are ignoring outputs in heavy rows.

We now show that H has rank at least $a_2 + \gamma b/4$. A column of H is **queried** if a column of E contains a selected input or the corresponding row of $\boldsymbol{B}$ contains a selected input. a_2 of these columns correspond to selected inputs in B and are linearly independent because the corresponding columns of F are linearly independent. Consider the unqueried columns of H. These columns in F are zero columns. Thus, consider these unqueried columns in E. Consider k rows in E that come from a common copy of A on the diagonal of $\boldsymbol{D}$. The column $\boldsymbol{B^i}$ of B corresponding to this copy of A is light (it has fewer than γn selected entries) because the corresponding column of C is chosen to be light. Thus, this copy of A has at least $n(1-\gamma)$ unqueried entries, or at least $n(1-\gamma)$ of its columns are unqueried.

Since A is γ-nice, the unqueried columns of this copy of it have rank at least $\min(k, \gamma n)$. Because there are no dependencies between columns in distinct copies of A in $\boldsymbol{D}$, the number of linearly independent unqueried columns of E is minimal if they all fall in as few common copies of A as possible, because then $\min(k, \gamma n) = \gamma n$. It follows that the unqueried columns of E have rank at least $\gamma b/4$. Since the queried columns have rank at least a_2, the columns of H have rank at least $a_2 + \gamma b/4$. It follows from an argument given in the proof of Lemma 10.13.2 that the number of solutions $\boldsymbol{B}$ to this system is at most $|\mathcal{R}|^{n^2-a_2-\gamma b/4}$. Since there are at most $|\mathcal{R}|^{n^2-a_1}$ matrices A that are γ-nice and consistent with the a_1 selected inputs in A, it follows that the number of pairs consistent with values of the selected inputs and outputs is at most $|\mathcal{R}|^{2n^2-a-\gamma b/4}$, the desired conclusion. ■

This result provides a lower bound on the space and time for matrix multiplication. The upper bound cited below is obtained by another hybrid algorithm that mixes a branching program for the standard algorithm with one for table lookup.

THEOREM 10.13.4 *There is an integer $n_0 > 0$ such that for $n > n_0$ the space S and time T needed to compute the matrix multiplication function $f^{(n)}_{A\times B} : \mathcal{R}^{2n^2} \mapsto \mathcal{R}^{n^2}$ over the ring $\mathcal{R}$ using a general branching program satisfies the inequality:*

$$ST^2 \geq \frac{\gamma^3}{16} n^6 \log_2 |\mathcal{R}|$$

for some $0 < \gamma < 1/2$ when $T \geq n^2$. This lower bound can be achieved up to a multiplicative factor of $O(\log n)$ for space in the range $\Omega(\log n + \log |\mathcal{A}|) \leq S \leq O(n \log |\mathcal{A}|)$.

Proof The lower bound follows from Theorem 10.11.1 and Lemma 10.13.3 by letting $a = \lfloor \gamma^2 n^4/4T \rfloor$, since this value of a satisfies the two conditions $a \leq \tau(ma/2T) = \gamma n\sqrt{ma/4T}$ and $a \leq 2n^2$ when $T \geq n^2$.

At the extreme of large space, namely $S = O(n^2)$, the upper bound follows from a branching program for table lookup that has one level for each of the $2n^2$ variables in the matrices A and B and the fact that there are $|\mathcal{R}|^{2n^2}$ pairs of such matrices over the ring $\mathcal{R}$. Consequently, the branching program has at most $O(|\mathcal{R}|^{2n^2})$ vertices and space $O(n^2 \log |\mathcal{R}|)$. It uses $O(n^2)$ steps.

At the extreme of small space, namely $S = \Omega(\log n + \log |\mathcal{A}|)$, we use a branching program for the standard matrix multiplication algorithm that forms n^2 inner products of rows and columns of the two matrices. As discussed in the proof of Theorem 10.13.3, a branching program can be constructed to form the inner product of two n-tuples that has $\Theta(n|\mathcal{R}|^2)$ vertices; that is, space $\Omega(\log n + \log |\mathcal{A}|)$ and time $O(n)$. Concatenating n^2 of these programs, one for each of the n^2 entries in the product matrix, we have a branching program with space $\Omega(\log n + \log |\mathcal{A}|)$ and time $O(n^3)$.

To fill in the gap between these extremes, the method applied in Theorem 10.13.3 can be used, as the reader can demonstrate. (See Problem 10.40.) ■

10.13.5 Matrix Inversion

As an intermediate step to deriving a space–time product lower bound on matrix inversion, we derive a lower bound for the product of three $n \times n$ matrices. This is done by first deriving an alternate representation for this product in terms of the Kronecker product of two matrices. Kronecker products are defined in Section 10.12.

LEMMA 10.13.4 *Let A, B, C, and D be $n \times n$ matrices over a commutative ring. The following two equations define the same set of mappings from entries of A, B, and C to entries in D:*

$$D = ABC$$
$$\boldsymbol{E} = (A \otimes C^T)\boldsymbol{B}$$

where $\boldsymbol{B}$ and $\boldsymbol{E}$ are $n^2 \times 1$ column vectors obtained by concatenating the transposes of the rows of the matrices B and D, respectively.

Proof Let $\boldsymbol{E} = (A \otimes C^T)\boldsymbol{B}$. The goal is to show that the results in the $n^2 \times 1$ column vector $\boldsymbol{E}$ are the same as those in the $n \times n$ matrix D but in a different order. In particular, we show that the $ni + j$ entry in the former, namely $e_{ni+j,1}$, is equal to the (i, j) entry in D, namely $d_{i,j}$.

Given a matrix F, let $f_{i,j}$ denote its entry in the ith row and jth column. Let $f_{i,-}$ and $f_{-,j}$ denote the ith row and jth column of F, respectively. Let rows and columns of matrices be numbered consecutively from zero.

The matrix $A \otimes C^T$ consists of blocks of n consecutive rows with the ith block containing $[a_{i,1}C^T, a_{i,2}C^T, \ldots, a_{i,n}C^T]$. Thus, the $ni + j$th entry of $\boldsymbol{E}$, namely $e_{ni+j,1}$, is the jth entry in the product $[a_{i,1}C^T, a_{i,2}C^T, \ldots, a_{i,n}C^T]\boldsymbol{B}$, as shown below, where $(c_{-,j})^T(b_{k,-})^T$ is the inner product of the row vector $(c_{-,j})^T$ with the column vector $(b_{k,-})^T$.

$$\begin{aligned} e_{ni+j,1} &= \sum_{k=0}^{n-1} a_{i,k}(c_{-,j})^T(b_{k,-})^T \\ &= \sum_{k=0}^{n-1}\sum_{l=0}^{n-1} a_{i,k}c_{l,j}b_{k,l} \\ &= \sum_{k=0}^{n-1}\sum_{l=0}^{n-1} a_{i,k}b_{k,l}c_{l,j} \\ &= d_{i,j} \end{aligned}$$

This is the desired conclusion. ■

With this as background, we state the space–time results to compute the product of three matrices.

THEOREM 10.13.5 *There is an integer $n_0 > 0$ such that for $n > n_0$ the time T and space S used by any general branching program to compute the product of three $n \times n$ matrices over a commutative ring $\mathcal{R}$ must satisfy the following inequality:*

$$ST = \Omega(n^4 \log |\mathcal{R}|)$$

Proof Given a general branching program to compute ABC, no more space or time are used when the matrices A and C are given specific values. Let them each be γ-nice for some $0 \leq \gamma \leq 1/2$. The existence of such matrices is established in Lemma 10.12.1. From Lemma 10.12.2 we know that the matrix $A \otimes C^T$ is γ^2-ok. The result follows from Theorem 10.13.3 since $A \otimes C^T$ is $n^2 \times n^2$. ■

We are now prepared to state space–time bounds for matrix inversion.

THEOREM 10.13.6 *There is an integer $n_0 > 0$ such that for $n > n_0$ the time T and space S used by any general branching program to compute the inverse of a non-singular $n \times n$ matrix over a commutative ring $\mathcal{R}$ must satisfy the following inequality:*

$$ST = \Omega(n^4 \log |\mathcal{R}|)$$

This lower bound can be achieved to within a multiplicative factor over the range $\Omega(n^2) \leq T \leq O(n^3)$.

Proof Let n be a multiple of 4. The lower bound follows by reducing matrix inversion to the computation of the product of three arbitrary $n/4 \times n/4$ matrices, as shown below:

$$\begin{bmatrix} I & -A & 0 & 0 \\ 0 & I & -B & 0 \\ 0 & 0 & I & -C \\ 0 & 0 & 0 & I \end{bmatrix}^{-1} = \begin{bmatrix} I & A & AB & ABC \\ 0 & I & B & BC \\ 0 & 0 & I & C \\ 0 & 0 & 0 & I \end{bmatrix}$$

The upper bound for $T = \Theta(n^2)$ is obtained by table lookup using an algorithm of the kind described in the proof of Theorem 10.13.3. For $T = \Theta(n^3)$, the matrix inversion algorithm based on the LDL^T decomposition of a symmetric positive definite matrix of Section 6.5.4 can be used. For intermediate values of time, a hybridized algorithm based on the inversion of block matrices provides the stated upper bound. ■

10.13.6 Discrete Fourier Transform

The discrete Fourier transform (DFT) and the fast Fourier transform algorithm are described in Sections 6.7.2 and 6.7.3. In this section we derive upper and lower bounds on space–time tradeoffs for this problem. The lower bound follows from the result for matrix-vector multiplication and the fact that the coefficient matrix for the DFT is $(1/4)$-ok.

LEMMA 10.13.5 *Consider the n-point DFT over a commutative ring that has a principal nth root of unity. It is defined as a matrix-vector product with $[w^{ij}]$ as its $n \times n$ coefficient matrix. This matrix is $(1/4)$-ok.*

Proof We use the fact, shown in Theorem 10.5.5, that the submatrix of $W = [w^{ij}]$ consisting of any k rows and any k consecutive columns is non-singular. We show that any $p \times q$ submatrix B of W, with $p \leq \lceil n/4 \rceil$ and $q \geq n - \lceil n/4 \rceil$, has rank at least $p/4$.

Let I denote the row indices of the submatrix B and let J denote its column indices. Let C be the submatrix of W with row indices in I. Divide the columns of C into $\lceil n/p \rceil$ groups each containing p columns except possibly the last which has at most p columns. We claim that some group has at least $p/2$ columns in common with B. Suppose not. Then every one of the $\lceil n/p \rceil$ groups has at most $(p-1)/2$ columns in common with B. Thus B has at most $\chi(p) = \lceil n/p \rceil (p-1)/2$ columns. We show that $\chi(p) < n - (n+3)/4 \leq n - \lceil n/4 \rceil$. But this is a contradiction because B has at least $n - \lceil n/4 \rceil$ columns. Since $\lceil n/p \rceil \leq (n+p-1)/p$, if $(n+p-1)(p-1)/2p < n - (n+3)/4$, the following holds after multiplying both sides by $2p$:

$$(n+p-1)(p-1) < \frac{3p(n-1)}{2} \quad \text{or}$$
$$-n+1 < p\left(\frac{(n+1)}{2} - p\right)$$

It suffices to show that the right-hand side of the last equation is positive. But $((n+1)/2) - p$ is positive since $p \leq \lceil n/4 \rceil \leq (n+3)/4 \leq (n+1)/2$ for $n \geq 1$. ■

THEOREM 10.13.7 *There is an integer $n_0 > 0$ such that for $n > n_0$ the n-point DFT over a commutative ring $\mathcal{R}$ requires space S and time T with a branching program satisfying the following*

lower bound:

$$ST = \Omega(n^2 \log |\mathcal{R}|)$$

This lower bound can be achieved to within a constant multiplicative factor.

Proof The upper bound follows by applying Lemma 10.9.3 and Theorem 10.5.5. ■

10.13.7 Unique Elements

We now derive a lower bound on the space–time product for the sorting problem by reducing sorting to the unique-elements problem. The unique elements problem takes a list of values and returns in any order a list of the non-repeated elements among them.

DEFINITION 10.13.1 *Let $\mathcal{R}$ be a set with at least n distinct elements. The function $f^{(n)}_{\text{unique}} : \mathcal{R}^n \mapsto 2^{\mathcal{R}^n}$ defines the* **unique elements** *problem where $2^{\mathcal{R}^n}$ is the power set of $\mathcal{R}^n$ and $f^{(n)}_{\text{unique}}(\boldsymbol{x})$ is the set of non-repeated elements in the input string $\boldsymbol{x}$.*

We emphasize that no order is imposed on the outputs of $f^{(n)}_{\text{unique}}$. Thus, if a set of values appears in the output, their position in the output does not matter.

From Lemma 10.11.2 it follows that a lower bound to ST can be derived by restricting the domain and discarding outputs. We restrict the domain by restricting each input variable to values in a subset $\mathcal{S} \subseteq \mathcal{R}$ containing n elements. We also restrict input tuples to the set $\mathcal{D}$ containing at least $n/(2e)$ unique values (e is the base of the natural logarithm). In the following lemma we show that $|\mathcal{D}| \geq |\mathcal{S}|^n/(2e-1) = \phi n^n$, where $\phi = 1/(2e-1)$. On inputs in $\mathcal{D}$ the function $f^{(n)}_{\text{unique}}$ has at least $n/(2e)$ unique outputs. We define the subfunction $f^{(n)}_{\text{restricted}} : \mathcal{S}^n \mapsto \mathcal{S}^m$, $m = n/(2e)$, of $f^{(n)}_{\text{unique}}$ to be the subfunction obtained by restricting its inputs to $\mathcal{D} \subset \mathcal{S}^n$ and deleting all but the first $n/(2e)$ outputs, which are all unique.

LEMMA 10.13.6 *Let $\mathcal{S}$ be a set of n elements. The fraction ϕ of the input n-tuples over $\mathcal{S}^n$ containing $n/(2e)$ or more unique elements exceeds $1/(2e-1)$.*

Proof We use simple probabilistic arguments. Assign each n-tuple over $\mathcal{S}^n$ probability $1/n^n$. Let $u(\boldsymbol{x})$ be the number of unique elements in $\boldsymbol{x}$. Let $X_i(\boldsymbol{x})$ have value 1 if the ith element of $\mathcal{S}$ occurs uniquely in $\boldsymbol{x}$ and value 0 otherwise. Then

$$u(\boldsymbol{x}) = \sum_{i=1}^{n} X_i(\boldsymbol{x})$$

Let $E[u]$ denote the average value of $u(\boldsymbol{x})$ (the sum of $u(\boldsymbol{x})$ over $\boldsymbol{x}$ weighted by its probability). Because the order of summation can be changed without affecting the sum, we have

$$E[u(\boldsymbol{x})] = \sum_{i=1}^{n} E[X_i(\boldsymbol{x})]$$

$E[X_i(\boldsymbol{x})]$ is also the probability that $X_i = 1$. If $X_i = 1$, then each of the other components of $\boldsymbol{x}$ can assume only one of $n-1$ values. Since the ith value can be in any one of n positions

among input variables and since for each position that it occupies there are $(n-1)^{n-1}$ ways to fill the remaining $n-1$ positions so that the ith value is unique, we have that $E[X_i] = f(n)$ where $f(n) = n(n-1)^{n-1}/n^n = (1-1/n)^n/(1-1/n)$. But $f(n)$ is a decreasing function of n, as is shown by calculating its derivative and using the inequality $(1-x) \leq e^{-x}$ (see Problem 10.5). The limit of $f(n)$ for large n is e^{-1} because in the limit of small x the function e^{-x} has value $1-x$. It follows that $E[u(\boldsymbol{x})] > n/e$.

Let $\pi = P_r[u(\boldsymbol{x}) \geq n/(2e)]$ be the fraction (or probability) of the input n-tuples for which $u(\boldsymbol{x}) \geq n/(2e)$). Because $u(\boldsymbol{x}) \leq n$, it follows that $\pi n + (1-\pi)n/(2e) \geq E[u(\boldsymbol{x})] \geq n/e$, from which we conclude that $\pi > 1/(2e-1)$. (This is known as **Markov's inequality**.) ■

LEMMA 10.13.7 *Let $|\mathcal{S}| = n$. Then $f^{(n)}_{\text{restricted}} : \mathcal{S}^n \mapsto \mathcal{S}^m$, $m = n/(2e)$, is $(\phi, \lambda, \mu, \nu, \tau)$-distinguishable for $\phi = 1/(2e-1)$, $\lambda = \mu = 1$, $\nu = (1-1/(2e))/\log_2 n$, and $\tau(b) = n$.*

Proof If $f^{(n)}_{\text{restricted}}$ is $(\phi, \lambda, \mu, \nu, \tau)$-distinguishable for $\phi = 1/(2e-1)$, $\lambda = \mu = 1/2$, $\nu = (1-1/(2e))/\log_2 n$, and $\tau(b) = n$, then for at least ϕn^n input tuples and any $a \leq \lambda n$ input and $b \leq \mu m$ output variables and specified values for them, $f^{(n)}_{\text{restricted}}$ has at most $n^{n-a-\nu b} = n^{n-a}e^{-(1-1/(2e))b}$ input n-tuples that are consistent with these assignments. The order of output values to $f^{(n)}_{\text{restricted}}$ is irrelevant.

Let $\mathcal{B}$ be the values of the b selected and specified unique outputs, $b \leq m$, and let $\mathcal{A}$ be the values of the a selected and specified input values. The k values in $\mathcal{B} - \mathcal{A}$ appear in input positions that are not specified. $r = n-k-a$ inputs are in neither $\mathcal{A}$ nor $\mathcal{B}$. We overestimate the number of patterns of inputs consistent with the a inputs and b outputs that are specified if we allow these a inputs to assume any value not in $\mathcal{B}$, since all values in $\mathcal{B}$ are unique. Thus, there are at most $(n-b)^r$ ways to assign values to these r inputs. The k values in $\mathcal{B} - \mathcal{A}$ are fixed, but their positions among the $r+k$ non-selected inputs are not fixed. Since there are $(r+k)!/r!$ ways for these ordered k values to appear among any specific ordering of the remaining r non-selected inputs (see Problem 10.6), the number Q of input patterns consistent with the selected and specified a inputs and b outputs satisfies the following inequality:

$$Q \leq \frac{(r+k)!}{r!}(n-b)^r$$

Here $r+k = n-a \leq n$ and $k \leq b$. Below we bound $(r+k)!/r!$ by $(r+k)^k$ and use the inequality $(1-x) \leq e^{-x}$:

$$\begin{aligned} Q \leq \ (r+k)^k(n-b)^r \ &\leq n^{r+k}\left(1-\frac{a}{n}\right)^k\left(1-\frac{b}{n}\right)^r \\ \leq n^{n-a}e^{-(ka/n+rb/n)} &\leq \ n^{n-a}e^{-(ka/n+(n-a-k)b/n)} \end{aligned}$$

The exponent $e(a,b,k) = ka/n + (n-a-k)b/n$ is a decreasing function of a whose smallest value is $(1-k/n)b$. In turn, this function is a decreasing function of k whose smallest value is $(1-b/n)b \geq (1-1/(2e))b$. As a consequence, we have

$$Q \leq n^{n-a}e^{-(1-1/(2e))b}$$

It follows that $f^{(n)}_{\text{restricted}}$ is $(\phi, \lambda, \mu, \nu, \tau)$-distinguishable for $\phi = 1/(2e-1)$, $\lambda = \mu = 1$, $\nu = (1-1/(2e))/\log_2 n$, and $\tau(b) = n$. ■

```
b := 0;
for j := 1 to ⌈n/S⌉
{b = (j − 1)S on the jth iteration.}
  begin
    for i := 1 to S
      C[i] := 0;
    for i := 1 to n
      if b ≤ x_i ≤ b + S then
        begin
          k := x_i − b;
          if C[k] < 2 then C[k] := C[k] + 1;
        end;
    for i := 1 to S
      if C[i] = 1 then print b + i;
    b := b + S;
  end
```

Figure 10.22 A RAM program for the unique-elements problem over the set $\{1, 2, \ldots, n\}$ when $n \geq S \geq O(\log n)$. The input to the program is the n-tuple $\boldsymbol{x}$ in which x_i is the ith entry. The program uses space $O(S)$.

Invoking Theorem 10.11.1, we have a quadratic space–time product lower bound. The RAM program for the unique elements problem given in Fig. 10.22 can be converted to a branching program to obtain an upper bound on the space–time product needed for this problem, as shown in Theorem 10.13.8.

THEOREM 10.13.8 *Let $|\mathcal{R}| \geq n$. There is an integer $n_0 > 0$ such that for $n \geq n_0$ and $S = \Omega(\log n)$ the time T and space S used by any general branching program for the unique elements function $f^{(n)}_{\text{unique}} : \mathcal{R}^n \mapsto 2^{\mathcal{R}^n}$ must satisfy*

$$ST = \Omega(n^2)$$

This lower bound can be met to within a constant multiplicative factor for inputs drawn from the set $\{1, 2, 3, \ldots, n\}$.

Proof The lower bound follows directly from Theorem 10.11.1. The upper bound follows from an analysis of the branching program that results from conversion of the RAM program in Fig. 10.22. The RAM program makes $\lceil n/S \rceil$ passes over the input data. On the jth pass the program examines input values in the range $[(j-1)S, \ldots, jS]$ and determines for each value whether there are zero, one, or more than one instances of it in the input.

The program uses an S-element one-dimensional array $C[1..S]$ that it initializes to zero at the beginning of each pass. If on the jth pass the ith input variable, x_i, is in the interval $[(j-1)S, \ldots, jS]$, the array element associated with it, namely $C[x_i - (j-1)S]$, is incremented unless it already has value 2. At the end of the jth pass, if the array element $C[i]$ has value 1, the program prints out the value $jS + i$, namely, the value of an input that appears only once in the input.

The reader is asked to show that the program of Fig. 10.22 can be converted to a branching program of space $O(S)$ and time $O(T)$. (See Problem 10.41.) ■

The program of Fig. 10.22 relies on the fact that input variables are drawn from the set $\{1, 2, 3, \ldots, n\}$. If the set from which they are drawn is much larger, say $\{1, 2, 3, \ldots, n^c\}$, $c > 1$, the outer loop is executed $O(n^c/S)$ times and its total running time is $O(n^c)$. Thus, the program is not optimal in this case.

10.13.8 Sorting

The sorting problem is described in Section 6.8. The general sorting problem is defined by a function $f^{(n)}_{\text{sort}} : \mathcal{R}^n \mapsto \mathcal{R}^n$ that rearranges the values of input variables so they are in descending order. Given a branching program for sorting, we show below that a branching program for the unique-elements problem can be obtained with a small additional amount of space. As a consequence, the space–time product lower bound for unique elements applies to the sorting problem. We also give a nearly matching upper bound.

THEOREM 10.13.9 *Let $|\mathcal{R}| \geq n$. There is an integer $n_0 > 0$ such that for $n \geq n_0$ and $S = \Omega(\log n)$ the time T and space S used by any general branching program for the sorting function $f^{(n)}_{\text{sort}} : \mathcal{R}^n \mapsto \mathcal{R}^n$ that reports its outputs in descending order must satisfy*

$$ST = \Omega(n^2)$$

This lower bound can be met to within a constant multiplicative factor for inputs drawn from the set $\{1, 2, 3, \ldots, n\}$.

Proof Given a branching program for $f^{(n)}_{\text{sort}}$ that uses space S, we use it to construct a branching program for $f^{(n)}_{\text{unique}}$ that uses space $S + O(\log n) = O(S)$. Since $f^{(n)}_{\text{unique}}$ requires space that is $\Omega(n^2/T)$, the same lower bound applies to sorting.

The branching program for $f^{(n)}_{\text{sort}}$ generates its sorted outputs in descending order. By analyzing the outputs the unique elements can be found. Store the last output l along with a bit b that is 1 if l is so far the only occurrence of this value and 0 otherwise. If the next output value is the same as l, set b to 0. If it is different from l and $b = 1$, produce l as an output, replace l with the last output, and set b to 1. Otherwise, do not produce an output.

Given a branching program Π for sorting, we describe a branching program for unique elements that uses modified copies of Π. If more than one output appears on some edge in Π, modify it (yielding Π^*) by replacing edges producing more than one output by a sequence of edges each producing one output separated by vertices testing an arbitrary input. This increases the number of vertices in Π by a factor of at most n and adds at most $\log_2 n$ to its space. Now make $2|\mathcal{R}|$ additional copies of Π^*, two for each value in $\mathcal{R}$, a "one" copy if the value is the first encountered in the sorted output and a "zero" copy if it is not.

Consider an edge in Π^* or one of its copies that produces an output (call it v). There are several cases to examine: the current copy of Π^* is a) the original copy, b) a "one" copy, or c) a "zero" copy. In case a), redirect the edge to the same vertex in the "one" copy of Π^* associated with v. In case b), if v is different from the value c associated with the current

copy of Π^*, output c and redirect the edge to the same vertex in the "one" copy of Π^* associated with v. In case c), if v is the same as the value associated with the current copy of Π^*, produce no output; otherwise also produce no output but redirect the edge to the same vertex in the "one" copy of Π^* associated with v. The new branching program has at most $2n + 1$ copies of Π^*, thereby increasing its space by an additive term of size $O(\log n)$. The lower bound on ST for the sorting problem follows.

The upper bound on ST for the sorting problem is obtained by constructing a family of branching programs, one for each value of S. We begin by constructing a "full" branching program for the case $S = \Theta(n)$. Let the variables in the input string be $x_1, x_2, \ldots, x_n$ and let them be tested in sequence. Thus, the root is labeled x_1 and has n successors, each of which tests x_2. There is one successor for each vertex labeled with x_2 for each way two numbers can be chosen with replacement from the set $\{1, 2, \ldots, n\}$. As shown in Problem 10.7, there are $N(n, k)$ ways in which k numbers can be drawn from a set of n elements with replacement where the order among the numbers is unimportant and

$$N(n, k) = \binom{n + k - 1}{k}$$

Thus, $N(n, 1) = n$ and $N(n, 2) = (n + 1)n/2$. The successors to vertices labeled x_2 are labeled x_3. They have $N(n, 3)$ successors, and so on. At the kth level there are $N(n, k)$ successors. Since $N(n, k) < 2^{n+k-1}$, it follows that for $k \leq n$ the above branching program has $O(2^{2n})$ vertices or space $S = \Theta(n)$. It also has time $T = n$ and space–time product $O(n^2)$.

To construct a branching program for space $S = O(n)$, we use $O(n/S)$ pruned copies of the full branching program described above. The idea behind the pruning is the following: we scan the input list looking for variables with values in the set $\{1, 2, \ldots, S\}$. If there are $O(S)$ of them, we record the number of values of each type and produce them in sorted order. However, if there are more than $O(S)$ elements in this range, as we examine additional inputs we reduce the size of the range so that only $O(S)$ space is used to carry the number of values of variables encountered. (This space is represented by $2^{O(S)}$ vertices in the branching program.) On each pass through the input either we reduce the size of the range by $O(S)$ or reduce the number of outputs that must be produced by the same amount. Thus, after $2n/S$ passes the input is sorted. Since each pass tests the value of each variable, the time is $O(n^2/S)$.

It is not difficult to convert the above schema into a branching program. The goal is to have no more than about 2^S vertices on each level of the branching program. The branching program will consist of $O(n/S)$ copies of the full branching program, each having n levels. Thus, the branching program will have $O(n^2 2^S/S)$ vertices or space $O(S)$.

We order vertices at each level in the branching program, placing those with smaller input values to the left. We remove vertices at the jth level that correspond to input values larger than S as well as those to the right of the first 2^S vertices on the jth level. Each edge in the first full branching program that is directed into a removed vertex is redirected to the root of the next copy of the branching program. The second copy of the full branching program is pruned to remove the vertices appearing in the first copy as well as those reached on inputs outside the range $[S + 1, S + 2, \ldots, 2S]$. The edges directed to removed vertices are redirected to the root of the third copy of the full branching program. A similar process is applied to each copy of the full branching program. ■

Problems

MATHEMATICAL PRELIMINARIES

10.1 Show that the the pyramid graph on m inputs, $P(m)$, has $m(m+1)/2$ vertices. Let $n = m(m+1)/2$. Show that $m \geq \sqrt{2n} - 1$.

10.2 Show that the following inequalities hold for integers m and x:

$$m/x \leq \lceil m/x \rceil \leq (m + x - 1)/x$$
$$(m - x + 1)/x \leq \lfloor m/x \rfloor \leq m/x$$

10.3 Suppose that $p \log_2 p \leq q$ for positive integers $p, q \geq 2$. Show that $p \leq 2q/\log_2 q$.

10.4 For n positive integers $x_1, x_2, \ldots, x_n$, show that the following inequality holds between the **geometric mean** on the left and the **arithmetic mean** on the right:

$$(x_1 x_2 \cdots x_n)^{1/n} \leq (x_1 + x_2 + \cdots + x_n)/n$$

10.5 Show that the inequality $(1 - x) \leq e^{-x}$ holds for $x \leq 1$.

10.6 Show that there are $(r + k)!/r!$ ways for k ordered values to appear among r distinct ordered items.

10.7 Show that there are $N(n, k) = \binom{n+k-1}{k} < 2^{n+k-1}$ ways to choose with repetition k numbers from a set $\mathcal{A}$ of size n where the order among the numbers is unimportant. Choosing with repetition means that a number can be chosen more than once.

Hint: Without loss of generality, let $\mathcal{A} = \{1, 2, \ldots, n\}$. Since order is unimportant, assume the chosen numbers are sorted. Let each chosen number be represented by a blue marker. Imagine placing the blue markers on a horizontal line. For $1 \leq i \leq n-1$, place a red marker between the last blue marker associated with the number i and the first blue marker associated with the number $i+1$, if any. This representation uniquely determines the number of elements of each type chosen. How many ways can the red markers be placed?

10.8 Show that a complete balanced binary tree on 2^{k-1} leaves has $2^k - 1$ vertices including leaves and that each path from a leaf to the root has $k - 1$ edges and k vertices.

THE PEBBLE GAME

10.9 Consider the circuit shown in Fig. 2.15. Treat each gate and each input vertex as a vertex. Give a good pebbling strategy for this graph.

10.10 Give a pebbling strategy for the m-input counting circuit in Fig. 2.21(b) that uses $O(\log^2 m)$ pebbles and $O(m)$ steps. Determine the minimum number of pebbles with which the circuit can be pebbled. Determine the number of steps needed with this minimal pebbling.

SPACE LOWER BOUNDS WITH PEBBLING

10.11 Consider the FFT graph $F^{(k)}$ on $m = 2^k$ inputs. Show that the subgraph connecting inputs to any one output is a complete binary tree on m leaves.

10.12 Consider a directed acyclic graph with n vertices, some of which have out-degree greater than 2. (a) Show that if each vertex of out-degree $k > 2$ is replaced by a binary tree with k leaves and edges directed from the root to the leaves, the number of vertices in the graph is at most doubled. (b) Show that replacing vertices with in-degree greater than 2 with binary trees also at most doubles the number of vertices in the graph.

EXTREME TRADEOFFS WITH PEBBLING

10.13 Let $N(k)$ be the number of vertices in the graph H_k discussed in Section 10.3. Show that the following recurrence holds for $N(k)$:

$$N(k) = N(k-1) + 4k + 3$$

Show that $N(k) = 2k^2 + 5k - 6$ for $k \geq 2$ since $N(2) = 12$.

10.14 Construct a new family $\{G_k\}$ of graphs with fan-in 2 at each vertex from the graphs $\{H_k\}$ by replacing the tree in Fig. 10.4 by a pyramid graph in k inputs and the bipartite graph with the graph E_k shown in Fig. 10.23. Show that each output of E_k can be pebbled with k pebbles but that after pebbling any one output there is at least one path without pebbles between the input and every other output. Show also that with $k+1$ pebbles E_k can be pebbled without repebbling any vertex.

Let $T_k(S)$ be the number of steps to pebble G_k with S pebbles. Using the above facts, show the following:

a) $N(k) = |G_k| = O(n^4)$
b) $S_{\min}(G_k) = k$
c) $T_k(k+1) = N(k)$
d) $T_k(k) = 2^{\Omega(N(k)^{1/4} \log N(k))}$

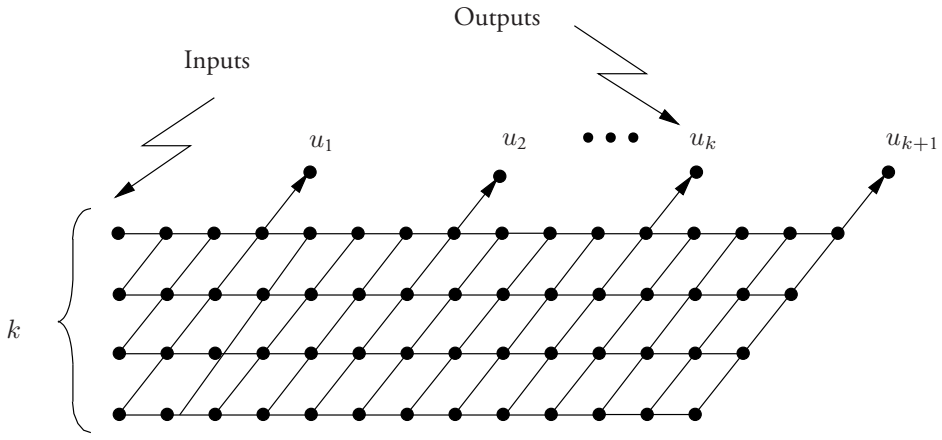


Figure 10.23 The graph E_k used in the construction of the family $\{G_k\}$.

SPACE–TIME LOWER BOUNDS WITH PEBBLING

10.15 Let A be a γ-nice $n \times n$ matrix over a ring $\mathcal{R}$ for some $0 < \gamma < 1/2$. Show that the matrix-vector multiplication function $f^{(n)}_{A\times\boldsymbol{x}} : \mathcal{R}^n \mapsto \mathcal{R}^n$ that maps the input n-tuple $\boldsymbol{x}$ to the output n-tuple $A\boldsymbol{x}$ is $(1, n^2 + n, n, \gamma n)$-independent.

10.16 Use Lemma 10.12.1 and the result of the previous problem to show that for almost all $n \times n$ matrices A every straight-line program for the matrix-vector multiplication function $f^{(n)}_{A\times\boldsymbol{x}} : \mathcal{R}^n \mapsto \mathcal{R}^n$ over the ring $\mathcal{R}$ requires space S and time T satisfying the inequality

$$(S + 1)T = \Omega(n^2)$$

Furthermore, show that a straight-line program for matrix-vector multiplication can be realized with space $S = 3$ and time $T = n(2n - 1)$, that is, with

$$(S + 1)T = O(n^2)$$

10.17 Linear systems are described in Section 6.2.2. A linear system of n equations in n unknowns $\boldsymbol{x}$ is defined by an $(n \times n)$-coefficient matrix A and an n-vector $\boldsymbol{b}$, as suggested below:

$$A\boldsymbol{x} = \boldsymbol{b} \tag{10.11}$$

The goal is to solve this equation for $\boldsymbol{x}$. If A is non-singular, such a solution exists for each vector $\boldsymbol{b}$. Let $f^{(n)}_{A^{-1}\times\boldsymbol{b}} : \mathcal{R}^{n^2+n} \mapsto \mathcal{R}^n$ denote the **linear system solver function** that maps the matrix A and the vector $\boldsymbol{b}$ onto the solution $\boldsymbol{x}$ when the matrix-vector multiplication is over the ring $\mathcal{R}$ and A is non-singular.

Show that every pebbling strategy for every straight-line program to compute the linear system solver function $f^{(n)}_{A^{-1}\times\boldsymbol{b}} : \mathcal{R}^{n^2} \mapsto \mathcal{R}^{n^2}$ over the ring $\mathcal{R}$ for n even requires space S and time T satisfying the following inequality:

$$(S + 1)T \geq n^3/24$$

Hint: Would it be possible to violate the lower bound on $(S+1)T$ for matrix inversion given in Problem 10.25 if a DAG for the linear system solver function can be pebbled with S pebbles in too few steps?

10.18 Let $f : \mathcal{A}^n \mapsto \mathcal{A}^m$ have $g : \mathcal{A}^r \mapsto \mathcal{A}^s$ as a subfunction. Show that if g is (α, r, s, p)-independent for $r \leq n$ and $s \leq m$, then so is f. Show that, as a consequence, the space S and time T needed to pebble the graph of a straight-line program for f satisfy the following inequality:

$$\lceil\alpha(S + 1)\rceil T \geq sp/4$$

10.19 Show that if a function is (α, n, m, p)-independent, it is also (α, n, m, q)-independent for $q \leq p$.

Hint: Consider the same set V of outputs in the two definitions.

10.20 A finite-state machine M computes the function $f_M^{(n)} : Q \times \Sigma^n \mapsto \Psi^n$ that maps the initial state in Q and an input string $\boldsymbol{x}$ of length n over the input alphabet Σ onto an output string $\boldsymbol{y}$ of the same length over the output alphabet Ψ. Such a machine can compute a function $f : \mathcal{A}^n \mapsto \mathcal{A}^n$ by associating inputs and outputs of f with inputs and outputs of $f_M^{(n)}$. A computation of an FSM M of a function f is **input-output oblivious** if the times at which inputs of f are read and its outputs produced are independent of the value of its input variables.

Show that Theorem 10.4.1 can be generalized from straight-line computations to computations by input-output-oblivious FSMs.

Hint: Try to parallel the proof of Theorem 10.4.1 using the FSM M instead of the pebble game. What correspondence can you make between the values under pebbles before the interval $\mathcal{I}$ and the state of M? Let $\log_2 |Q|$, where Q is the set of states of M, be the measure of space associated with it.

10.21 Give a design of an FSM that computes a function f from straight-line programs for it using a number of steps and storage locations proportional to the time and space used by a pebbling strategy for this straight-line program.

Hint: Design the FSM so that it receives the inputs provided to the pebbling strategy as well as instructions to specify which operations are performed on the inputs and temporary storage locations of the FSM.

TRANSITIVE FUNCTIONS

10.22 Many functions for which space–time lower bounds have been derived are transitive. Such functions have the property that for subsets X and Y of their inputs and outputs, respectively, $|X| = |Y| = n$, the (control) inputs not in X can be chosen so as to cause the outputs in Y to be equal to an arbitrary permutation drawn from the set $G(n)$ of the inputs in X. For example, the cyclic shifting function studied in Section 2.5.2 has a set of control inputs that specify the amount by which value inputs are permuted cyclically and assigned to the output variables.

DEFINITION 10.13.2 *Let $G(n)$ be a group of permutations of the integers $\mathbb{N}(n) = \{0, 1, 2, \ldots, n-1\}$. That is, if π is in $G(n)$, then $\pi : \mathbb{N}(n) \mapsto \mathbb{N}(n)$. We denote by $\pi(i)$ the integer to which integer i is mapped by π. A function $f_{G(n)} : \mathcal{A}^{n+s} \mapsto \mathcal{A}^n$, where $(y_{n-1}, \ldots, y_1, y_0) = f_{G(n)}(x_{n-1}, \ldots, x_1, x_0, c_{s-1}, \ldots, c_0)$, is said to have* **value inputs** *$x_{n-1}, \ldots, x_1, x_0$,* **control inputs** *$c_{s-1}, \ldots, c_0$, and* **outputs** *$y_{n-1}, \ldots, y_1, y_0$. Such a function is* **transitive of order** *n with respect to the group $G(n)$ if*

a) For each $0 \leq i \leq n-1$ and $0 \leq j \leq n-1$, there exists a permutation $\pi \in G(n)$ such that $\pi(i) = j$, and

b) For each $\pi \in G(n)$, there is an assignment to $c_{s-1}, \ldots, c_0$ such that $y_{\pi(i)} = x_i$ for $0 \leq i \leq n-1$.

Show that every transitive function of order n with respect to the permutation group $G(n)$, $f_{G(n)} : \mathcal{A}^{n+s} \mapsto \mathcal{A}^n$, is $(2, n+s, n, n/2)$-independent.

10.23 Show that the cyclic shifting function $f_{\text{cyclic}}^{(n)} : \mathcal{B}^{n+\lceil \log n \rceil} \mapsto \mathcal{B}^n$ defined in Section 2.5.2 is transitive of order n.

10.24 Consider the function $f^{(n)}_{PAQ} : \mathcal{R}^{3n^2} \mapsto \mathcal{R}^{n^2}$ whose value is the product PAQ of three $n \times n$ matrices P, A, and Q. Let P and Q be permutation matrices whose entries serve as control inputs. Show that $f^{(n)}_{PAQ}$ is transitive of order n^2.

10.25 The matrix inversion function $f^{(n)}_{M^{-1}} : \mathcal{R}^{n^2} \mapsto \mathcal{R}^{n^2}$ maps a non-singular $n \times n$ matrix over the ring $\mathcal{R}$ to its inverse. (See Section 6.3.) Show that $f^{(n)}_{M^{-1}}$ is $(2, n^2, n, n/2)$-independent.

Hint: Show that $f^{(2n)}_{M^{-1}}$ contains as a subfunction the function $f^{(n)}_{PAQ} : \mathcal{R}^{3n^2} \mapsto \mathcal{R}^{n^2}$ defined in Problem 10.24. In this connection consider the following identity, which holds when the $n \times n$ matrices R and S are non-singular:

$$M = \begin{bmatrix} R & A \\ 0 & S \end{bmatrix}^{-1} = \begin{bmatrix} R^{-1} & -R^{-1}AS^{-1} \\ 0 & S^{-1} \end{bmatrix}$$

PEBBLING SUPERCONCENTRATORS

10.26 Show that the graph consisting of two $n = 2^d$-input FFT graphs connected back to back (as shown in Fig. 10.24 with the second FFT graph reversed) is a superconcentrator. (Valiant [342] has shown the existence of n-superconcentrators with $O(n)$ vertices.)

Hint: Reason that there are unique vertex-disjoint paths from any r input vertices of this graph to any r consecutive vertices that are simultaneously outputs of the first FFT graph and the inputs to the reversed FFT graph. The first and last vertices are consecutive.

10.27 Prove that to pebble any $S+1$ outputs of an n-superconcentrator, $S+1 \leq n$, from an initial placement of S pebbles requires that at least $n - S$ different inputs be pebbled.

Hint: Suppose that at most $n - (S + 1)$ inputs are pebbled from an initial placement of S pebbles to pebble $S + 1$ outputs. Can you reason from the superconcentration

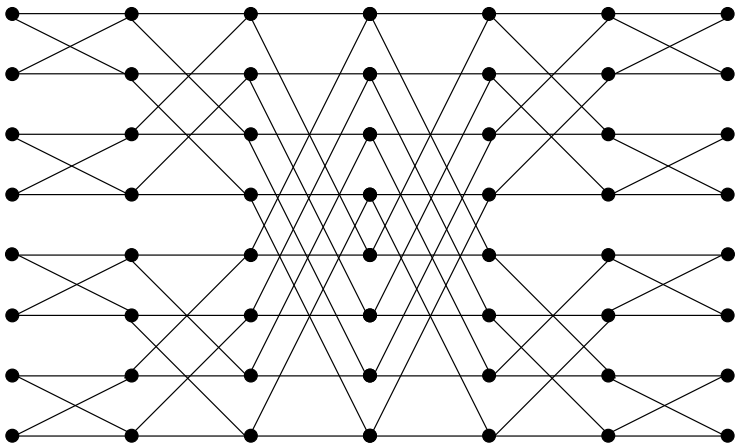


Figure 10.24 Two back-to-back FFT graphs form a superconcentrator.

property that $S + 1$ or more inputs cannot remain unpebbled since $S + 1$ outputs are pebbled?

10.28 Use the result of the previous problem to show that to pebble an n-superconcentrator with S pebbles in time T requires S and T to satisfy the following inequality:

$$(S + 1)T \geq \frac{n^2}{2}$$

Hint: As in the proof of Theorem 10.4.1, divide time up into consecutive intervals. Choose the intervals so that each has the same number of outputs pebbled during it. Apply the results of the previous problem to obtain a lower bound on the sum of the number of input and output vertices that are pebbled during the interval.

10.29 Show that the pebbling of two n-input back-to-back FFT graphs requires space and time that satisfy $S^2T = \Omega(n^3)$ and that this lower bound can be achieved up to a multiplicative factor.

Hint: From the proof of Lemma 10.5.4 it follows that to pebble any $2S$ outputs with S pebbles at least $n - S + 1$ inputs must be pebbled because if fewer inputs need be pebbled the outputs can have more values than is possible for the FFT.

APPLICATIONS OF THE GRIGORIEV LOWER BOUND

10.30 Show that there is a pebbling for a straight-line program for the cyclic shift function $f^{(n)}_{\text{cyclic}} : \mathcal{B}^{n+\lceil \log n \rceil} \mapsto \mathcal{B}^n$ examined in Section 10.5.2 for which $(S + 1)T = O(n^2 \log n)$.

Hint: Pebble the graph of the circuit described in Section 2.5.1. Construct a circuit for $f^{(n)}_{\text{cyclic}}$ that produces each output with $O(n \log n)$ gates.

10.31 Show that the binary addition function $f^{(n)}_{\text{add}}$ (see Section 2.7) can be realized by a straight-line program using space and time satisfying $ST = O(n)$.

10.32 Derive upper and lower bounds on the product $(S + 1)T$ for pebblings of circuits for the squaring function $f^{(n)}_{\text{square}}$ that are within a factor of $O(\log^2 n)$ of one another.

10.33 Derive good upper and lower bounds on the product $(S+1)T$ for pebblings of circuits for the reciprocal function $f^{(n)}_{\text{recip}}$.

10.34 In Section 6.5.3 a straight-line algorithm is given to invert an $n \times n$ triangular matrix. Construct another straight-line algorithm based on it that can be pebbled with $O(n)$ pebbles to produce outputs by columns in $O(n^3)$ steps under the assumption that the standard matrix multiplication algorithm is used for the matrix multiplication steps.

Hint: To produce outputs of a triangular matrix T by columns using the algorithm of Fig. 6.5, it is necessary to read the elements of $T_{2,1}$ by rows and produce the outputs of $T^{-1}_{2,2}$ by rows. Consider modifying this algorithm to generate the elements of the latter matrix first by rows and then by columns.

BRANCHING PROGRAMS

10.35 Give a proof of Lemma 10.9.1 by a) designing a general branching program to simulate a comparison operator and b) using this design in a complete branching program that simulates a decision branching program.

10.36 In Section 10.9 a procedure is given to convert a general branching program to a tree program without increasing the length of any path. Use this fact to show that every decision branching program with queries $\{\leq, =\}$ that sorts a list of n items requires worst-case time of at least $(n/2)\log(n/2)$ when n is even. Show that this lower bound can be achieved up to a constant multiplicative factor.

Hint: Show that every binary tree with m leaves must have a longest path of length at least $\log_2 m$ and determine the number of distinct leaves necessary in every decision branching program for sorting.

THE BORODIN-COOK LOWER-BOUND METHOD

10.37 The computation time of a branching program is the length of the longest path in its directed acyclic multigraph. Assume that a probability is assigned to each input $\boldsymbol{x}$ of length n. The **average computation time**, $\overline{T}$, of a branching program is the sum of the lengths of the paths associated with different inputs weighted by the probabilities of these inputs. To compute the average space of a branching program with k vertices, the integers in the set $\{1, 2, \ldots, k\}$ are assigned to the vertices of the branching program. The space associated with input $\boldsymbol{x}$ is the base-2 logarithm of the largest such integer encountered during the computation associated with $\boldsymbol{x}$. The average space associated with a numbering of vertices is the average of this logarithm. The **average space**, $\overline{S}$, associated with a branching program is the smallest average space over all numberings of vertices.

Given a probability distribution on inputs of length n, let $C_f(a, b)$ denote the maximum over all those tree branching programs of depth a of the probability that b of the m outputs of the function f are computed correctly. Show that Theorem 10.11.1 can be generalized to the above probabilistic setting.

Hint: If $\overline{T}$ is the average time of the branching program P, truncate the branching program at depth $2\overline{T}$, call the new program P^*, and show that P^* solves the problem solved by P with probability at least 1/2. Also, show that with probability at least $1/2$ there exists a rich path in some stage that produces $b = \lceil m/\sigma \rceil$ outputs. Let p_i be the probability that the subtree with root i in some stage correctly produces b outputs. Now develop an upper bound in terms of the p_i on the probability that some tree in some stage correctly produces b outputs.

APPLICATIONS OF THE BORODIN-COOK LOWER BOUND

10.38 Show that the branching program in Fig. 10.20 computes the inner product of two 3-element sequences over the **set of integers modulo-2**; that is, the integers $\{0, 1\}$ with the EXCLUSIVE-OR function for addition and the AND function for multiplication.

10.39 Complete the proof of Theorem 10.13.2 by filling in the details of the construction of a branching program for integer multiplication for the middle range of space.

10.40 Complete the proof of Theorem 10.13.4 by showing that two $n \times n$ matrices can be multiplied with a hybrid algorithm that combines table lookup with the standard matrix multiplication algorithm on $k \times k$ blocks to achieve space and time satisfying

$$ST^2 = O(n^3 \log |\mathcal{R}|)$$

10.41 Show that the RAM program described in Fig. 10.22 can be converted to a branching program of space $O(S)$ and time $O(T)$.

Chapter Notes

The first formal study of space–time tradeoffs was made by Cobham [73]. He considered computations on one-tape Turing machines using as a space measure the logarithm of the number of configurations, and obtained quadratic lower bounds on the space–time product to recognize strings representing palindromes and perfect squares.

The pebble-game model was implicitly used by Paterson and Hewitt [238] to study program schemas, uninterpreted graphs representing programs. They derived the space lower bound of Lemma 10.2.1, thereby demonstrating that recursive programs are more powerful than nonrecursive ones. Cook [75,79] asked how much space (how many pebbles) was needed to execute a program schema with n vertices and obtained the result for pyramids of Lemma 10.2.2, showing that the minimum space is at least $\Omega(\sqrt{n})$ for some schemas. The minimum-space question was answered by Hopcroft, Paul, and Valiant [139], who proved Theorem 10.7.1, and Paul, Tarjan, and Celoni [245], who obtained Theorem 10.8.1. The pebble model first formally appeared in [139]. Gilbert, Lengauer, and Tarjan [114] and Loui [204] have shown that the languages associated with minimal pebblings of DAGs (described at the end of Section 10.2) are **PSPACE**-complete.

In addition to studying the minimum space needed for a computation, researchers also examined tradeoffs between space and time. Paterson and Hewitt [238] studied the conversion of a linear recursive program schema into a non-recursive one and demonstrated that the time needed satisfies $T = \Omega(n^{1+1/(S-1)})$ for $S \geq 2$. (See Chandra [66] and Swamy and Savage [320]) for more details on this problem.)

A number of other authors have identified graphs exhibiting non-trivial exchanges of space for time. Pippenger [253] gave a graph on n vertices for which $T = \Omega(n \log \log n)$ when $S = O(n/\log n)$, and Savage and Swamy [292] demonstrated that the FFT graph requires S and T satisfying $ST = \Theta(n^2)$. (This is the first tradeoff result for a natural algorithm. Their upper bound is given in Theorem 10.5.5.) Later Tompa [332] and Reischuk [278] exhibited graphs requiring $T = \Omega(n \log n)$ and $T = \Omega(n \log^t n)$ for any integer t, respectively, when $S = \Theta(n/\log n)$.

Paul and Tarjan [244], Lingas [200], and van Emde Boas and van Leeuwen [348] gave graphs with T increasing from $O(n)$ to $T = 2^{\Omega(n^{1/2})}$, $T = 2^{\Omega(n^{1/3})}$, and $T = 2^{\Omega(n^{1/4} \log n)}$, respectively, when S drops by a constant amount from $S = O(n^{1/2})$, $S = O(n^{1/3})$ and $S = O(n^{1/4})$, respectively. Theorem 10.3.1 is from [348], as is Problem 10.14. Carlson and Savage [64] took a different tack and exhibited graphs for which T is superlinear, namely, $T = 2^{\Omega(\log n \log \log n)}$ over a range of values of S, namely, $\Omega(\log n) \leq S \leq O(n^{1/2}/\log n)$. References to the worst-case exchange of space for time are given in Section 10.6.

Grigoriev [120] gave the first space–time lower bounds that apply to all graphs for a problem (see Corollary 10.4.1), the essential idea of which is generalized in Theorem 10.4.1. Savage [290] introduced the $w(u, v)$-flow measure used in this version of a theorem to derive lower bounds on area–time tradeoffs for VLSI algorithms. Grigoriev [120] also established Theorem 10.4.2 and derived a tradeoff lower bound on polynomial multiplication that is equivalent to Theorem 10.5.1 on convolution. The improved version of Theorem 10.4.2, namely Theorem 10.5.4, is original with this book.

Lower bounds using the Grigoriev approach explicitly require that the sets over which functions are defined be finite. Tompa [330,331] eliminated the requirement for finite sets but required instead that functions be linear. Using concentrator properties of matrices deduced by Valiant [342], Tompa derived a lower bound on ST for superconcentrators that he applied to matrix-vector multiplication and polynomial multiplication. He developed a similar lower bound for the DFT. (See Abelson [2] for a generalization of some of these results to continuous functions.) The lower bound of Theorem 10.5.5 uses Tompa's DFT proof but does not require that straight-line programs be linear.

The result on cyclic shift (Theorem 10.5.2) is due to Savage [291]. (This paper also generalizes Grigoriev's model to I/O-oblivious FSMs, extends JáJá's [146] space–time lower bound for matrix inversion, and derives space–time lower bounds for transitive functions and banded matrices.) The result on integer multiplication (Theorem 10.5.3) is due to Savage and Swamy [293]. In [330] Tompa also obtained Theorem 10.5.6 on merging. Transitive functions defined in Problem 10.22 were introduced by Vuillemin [354].

In [332] Tompa examined the graph associated with the algorithm for transitive closure based on successive squarings described in Section 6.4 and demonstrated that it can be pebbled either in a polynomial number of steps or with small space, namely $O(\log^2 n)$, but not both. Carlson [61] demonstrated that algorithms for convolution based on FFT graphs (see Section 6.7.4) require that $T = \Theta(n^3/S^2 + n^2(\log n)/S)$, which doesn't come close to matching the lower bound of Theorem 10.5.1. However, through the judicious replacement of back-to-back FFT subgraphs in the standard convolution algorithm, Carlson [62] was able to achieve the bounds $T = \Theta(n \log S + n^2(\log S)/S)$, which are optimal over all FFT-based convolution algorithms and nearly as good as the $T = \Theta(n^2/S)$ bounds. (See also [63].) Carlson and Savage [65] explored for a number of problems the size of the smallest graphs that can be pebbled with a small number of pebbles and demonstrated a tradeoff between size and space.

Pippenger [250] has surveyed many of the results described above as well as those on the black-white pebble game described below.

Several extensions of the pebble game have been developed. One of these is the red-blue pebble game discussed in Chapter 11 and its generalization, the memory hierarchy game. Another is the black-white pebble game whose rules are the following: a) a black pebble can be placed on an input vertex at any time and on a non-input vertex only if its predecessors carry pebbles, whether white or black; b) a black pebble may be removed at any time; c) a white pebble can be placed on a vertex at any time; d) a white pebble can be removed only if all its predecessors carry pebbles. The placement of white pebbles models a non-deterministic guess. The removal of a white vertex is allowed only when the guess has been verified. Questions this game makes possible are whether the minimum space required for a graph is lower with the black-white pebble game than with the standard game and whether for a given amount of space, the time required is lower. The black-white game was introduced by Cook and Sethi

[78], who showed that the minimum space for the pyramid graph is at least $\sqrt{N/2}-1$. Meyer auf der Heide [221] proved that this minimum space is at most $\lceil n/2 \rceil + 2$ and established in general that any graph with minimum space n in the black-white game has minimum space at most $(n^2-n)/2+1$ in the standard game. The latter result is the pebbling analog of Savitch's theorem (Theorem 8.5.5).

Loui [205] and Meyer auf der Heide [221] have shown that the minimum space with the black-white game is at least one half that for the standard pebble game for balanced trees, a result extended by Lengauer and Tarjan [195] to all trees and then by Klawe [166]. Wilber [362] has exhibited an infinite family of graphs for which the black-white minimum space is smaller than the minimum space with the standard game by more than a constant factor.

All of the pebble games mentioned above are one-person games; that is, one person plays the game. A two-person game introduced by Venkateswaran and Tompa [351] models parallel complexity classes. Savage and Vitter [295] have also introduced a model of parallel pebbling.

Branching programs have been known as binary decision diagrams for at least 30 years [15], although their importance to CAD was recognized only in the last 10 or 12 years. (See [60]). Branching programs were proposed as a vehicle for studying space–time problems by Pippenger and first studied by Tompa [330], who cites Pippenger for Lemma 10.9.2. Borodin, Fischer, Kirkpatrick, Lynch, and Tompa [55] derived a lower bound of $ST = \Omega(n^2)$ to sort n items with decision branching programs. Borodin and Cook [53] formulated the same problem in terms of the general branching programs of Section 10.9 and developed the general framework used in Theorem 10.11.1.

Yesha [369] developed lower bounds on the space–time product with branching problems for the discrete Fourier transform (see Theorem 10.13.7) and matrix multiplication over restricted domains. Abrahamson [6] (see also [4]) derived the lower bound on ST^2 in Theorem 10.13.4, thereby improving upon the matrix multiplication bound of Yesha. He also extended the Borodin-Cook model to probabilistic branching programs (see Problem 10.37) and derived the lower bound on ST for convolution (Theorem 10.13.1), integer multiplication (Theorem 10.13.2), matrix-vector multiplication (Theorem 10.13.3), and matrix inversion (Theorem 10.13.6). He also developed a lower bound of $\Omega(n^3)$ on ST to compute the product PAQ of three $n \times n$ matrices, where P and Q are permutation matrices. Abrahamson has also studied Boolean matrix multiplication in the general branching program model [5]. Beame [34] has obtained the result of Theorem 10.13.8 showing that the unique elements problem requires that $ST = \Omega(n^2)$ for general branching programs, which implies the lower bound on sorting stated in Theorem 10.13.9.

In the comparison-based branching program model, Borodin, Fich, Meyer auf der Heide, Upfal, and Wigderson [54] derive the lower bound $ST = \Omega(n^{3/2}\sqrt{\log n})$ for the element-distinctness problem on n inputs. For the same computational model, Yao [368] improved this to $ST = \Omega(n^{2-\epsilon(n)})$, where $\epsilon(n)$ is a decreasing function of n.