担当：　富井尚志　(tommy@ynu.ac.jp)

「アルゴリズムとデータ構造」講義日程

~~1. 基本的データ型~~
~~2. 基本的制御構造~~
~~3. 変数のスコープルール，関数~~
~~4. 配列を扱うアルゴリズムの基礎(1)．最大値，最小値~~
~~5. 配列を扱うアルゴリズムの基礎(2)．重複除去，集合演算，ポインタ~~
~~6. ファイルの扱い~~
~~7. 整列(1)．単純挿入整列・単純選択整列・単純交換整列~~
~~8. 整列(2)．マージ整列・クイック整列~~
~~9. 再帰的アルゴリズムの基礎．再帰におけるスコープ．ハノイの塔など．~~
10. バックトラックアルゴリズム．8 王妃問題など．　← 本日の内容
11. 線形リストを扱うアルゴリズム
12. 木構造を扱うアルゴリズム(1) 基礎
13. 木構造を扱うアルゴリズム(2) 挿入，削除，バランスなど．
14. ハッシング
15. その他のアルゴリズム

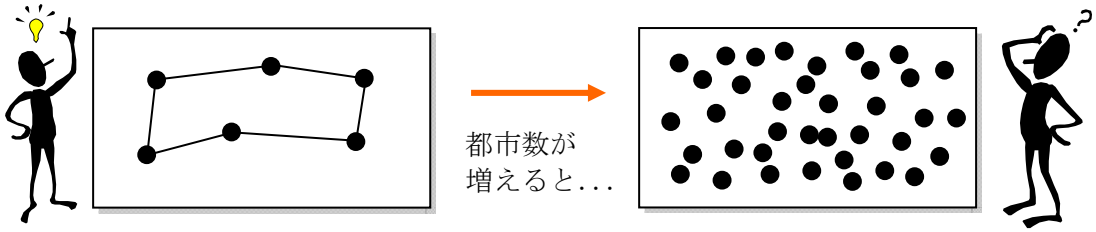
第 10 回「再帰的アルゴリズム（ﾊﾞｯｸﾄﾗｯｸｱﾙｺﾞﾘｽﾞﾑ）」

☆ 一般問題解決（人工知能の基礎）

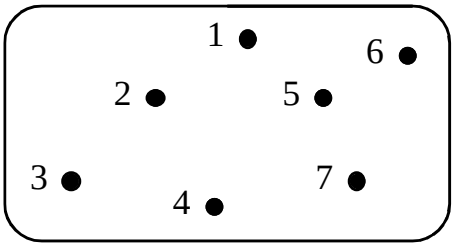
- 定まった計算規則に従うのではなく，「**試行錯誤**」を用いる．
- 試行錯誤をいくつかの部分作業に分解．多くの場合再帰的 → 「**探索**」

例）巡回セールスマン問題における部分作業への分解（試験範囲外）

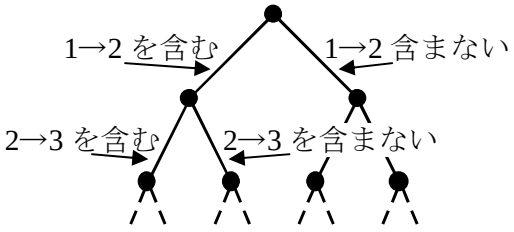
- 全都市を 1 回だけ巡る巡回路の中で総和が最小の経路長のものを求める問題．
- 都市数が増えると組み合わせの数が爆発し，非常に難解な問題になる．



- 解法の例：部分的な経路に着目して分割して求める．



都市 1～7 の分布

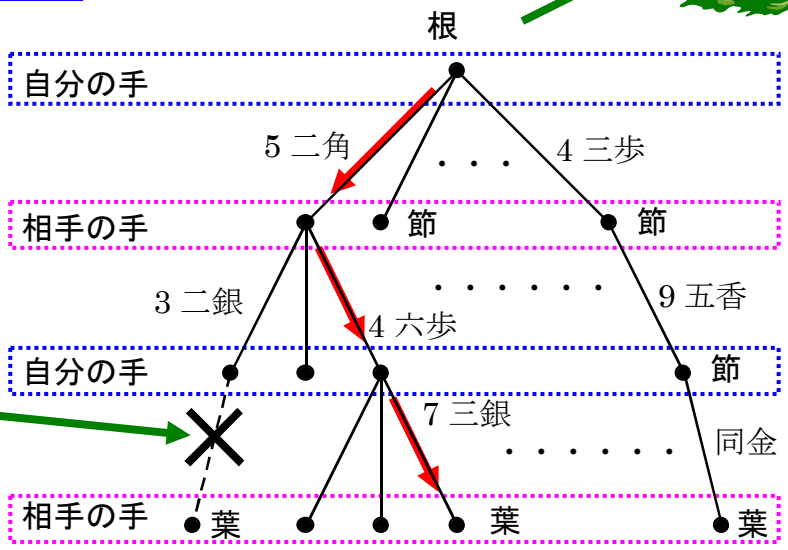


経路で部分問題に分割して解法

・ 「**探索の木**」を徐々に構成．多くの問題では指数関数的に成長
　→ 高速化のため「**発見的手法**」による「**枝刈(えだがり)**」が必要

例）ゲーム木探索（試験範囲外）

- 例えば**将棋**では現在の局面で指せる手の中から最善の手を選択する（＝木の探索(**右図**)）．
- 「見込みのない局面」以下は探しても無駄（＝**枝刈**）．
- 局面を評価する関数の設計が重要．



☆ バックトラックアルゴリズム --- 探索のためのアルゴリズム

・ 「**バックトラック(backtrack)**」＝「**後戻り**」

問題を解いている途中の部分問題において可能な**解の候補**がいくつかあったとき，

1) まずそのうちの一つを選択し，その部分問題の「仮の」解にする．
2) 問題の続きを解いてみる．
3) 成功ならそのまま．失敗なら，1) での選択を「破棄」し，別の候補を「仮の」解にし，2)へ．

・ 上記 2) の部分が**再帰的**になる．

・ 図式

```
try( )
{
    候補選択の初期化;
  do{
      次の候補の選択;
    if  (仮の解として適する) {
        解の一部として記録;
      if  (まだ解くべき問題が残っている) {
          try( 次の段階 );
          if  (失敗) 先の記録を破棄;
      }
    }
  } while (解がみつからない || 次の候補がある)
}
```

"try"は勝手に付けた関数名

自分を呼んでいる（＝再帰）

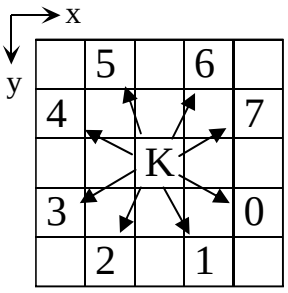
"または"

☆ 例 1　騎士巡回（Knight tour）

(a) n*n のチェス盤の座標 (x0,y0) にナイト（騎士）が置かれている.

(b) ナイトの移動規則に従って全てのマス目をちょうど 1 回だけ訪問する指し手を探す.

・ナイト(K)の動き：下図の 0 番から 7 番まで 8 通りの移動方法がある.



日本の将棋で言うと“桂馬”みたいな動きなんだな．これが.

knight.c

☆ 全解探索

・与えられた問題の全ての解を求める.

・図式

```
try( )
{
    for ( k=0; k<m; k++ ){
        k 番目の候補を選択;
        if  (仮の解として適する) {
            解の一部として記録;
            if  (まだ解くべき問題が残っている) {
                try (次の段階);
            else
                完全解の一つとして印刷（結果保存）
            }
            先の記録を破棄;
        }
    }
}
```

自分を呼んでいる（＝再帰）

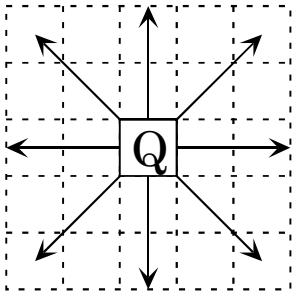
☆ 例 2　n クイーン(n-queen)

nqueen.c

(a) n×n のチェス盤と n 個のクイーン（女王）がある.

(b) 全てのクイーンを互いにとることができない（効き筋にならない）位置関係になるように配置する.

※ クイーン（Q）は「同じ列」「同じ行」「同じ対角線(2 つ)」のコマをとることができる.



いずれの方向へも無制限に遠方まで行くことができる.

・騎士巡回のプログラムと異なり，盤面自身ではなく，“各「行」「列」「対角線(2 つ)」にクイーンが置かれているか否か”という情報を管理する.

☆ 最適値探索

- ある尺度=「**評価関数**」によって決められた序列の下で，一番良い解を求める．
- 全ての解を生成し，それらの生成過程において最適解を一つ保持する．
- 図式

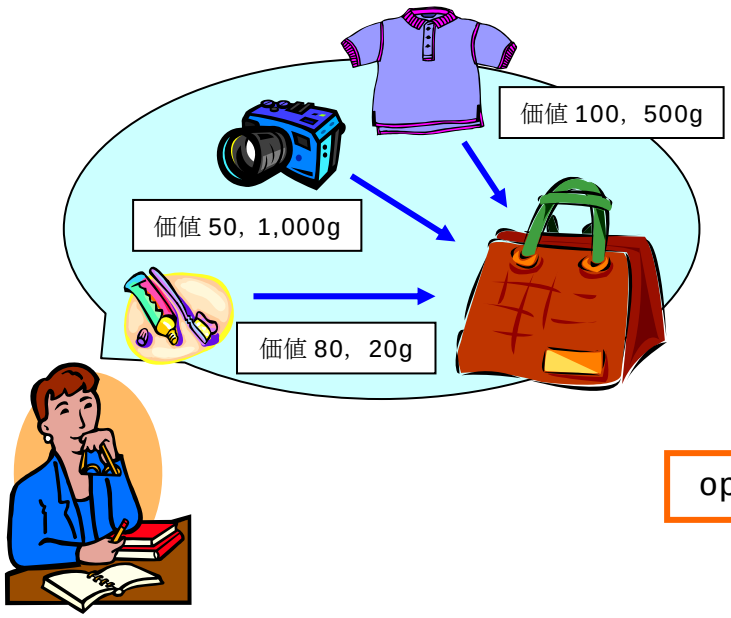
```
try(i)
{
    /* i 番目の選択肢を選択候補に加える場合 */
    if  (加えても条件を満足する) {
        i 番目の選択肢を選択候補に加える;
        if  (i<n)
            try(i+1);          ← 自分を呼んでいる（＝再帰）
        else
            最適解かどうかのチェック;
        i 番目の選択肢を選択候補から取り除く;
    }
    /* i 番目の選択肢を選択候補に加えない場合 */
    if  (加えなくても条件を満足する) {
        if  (i<n)
            try(i+1);          ← 自分を呼んでいる（＝再帰）
        else
            最適解かどうかのチェック;
    }
}
```

☆ 例 3　旅行カバンに物をつめる（ナップザック問題）

(a) n 個の品物を旅行カバンの中に入れて旅行する．持って歩ける重量には制限がある

(b) 個々の品物には，**重さ** と **価値（重要度）** の情報が与えられている．

(c) n 個の品物からいくつか選択して，制限重量以下で価値の総和が最大になるようにする．



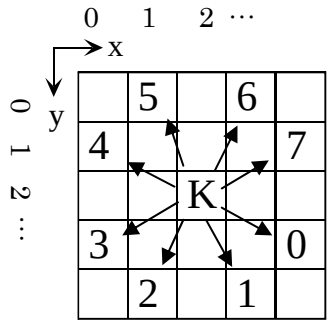
optimum.c

```
/*************************************************************
    「アルゴリズムとデータ構造」
    サンプルプログラム knight.c
    <<バックトラックアルゴリズムの例: ナイトツアー>>
    copyright (c)  T.Mori <mori@forest.dnj.ynu.ac.jp>
*************************************************************/
#include <stdio.h>
#define TRUE      1
#define FALSE     0
#define N         8       /* 盤面の大きさ */
#define NSQR      64      /* 盤面の大きさの 2 乗 */
/*  h[x][y]は，チェス盤面を表す.
    h[x][y]=0     マス目(x,y)は訪問されていない
    h[x][y]=k     マス目(x,y)は第 k 番目の指し手で訪問された．1 ≦ k ≦ N*N
 */
int h[N][N];

int try(int i, int x, int y);
void move_to(int i, int x, int y);
void undo(int x, int y);
void print_knights(void);

int main(void)
{
    int i,j;

    /* 盤面データ初期化 */
    for( i=0; i<N; i++ )
        for( j=0; j<N; j++ )
            h[i][j] = 0;
    /* 出発点を (0,0) にする */
    h[0][0] = 1;
    if (try(2,0,0))  /* 解が見つかったなら */
        print_knights();
    else
        printf("解なし¥n");
    return 0;
}

/*  dx[m],dy[m]は，ナイトの移動の仕方を X 軸 Y 軸それぞれの相対距離
    (どれだけ移動するか)で表したもの．8 通りあるので，0 ≦ m ≦ 7．*/
#define C   8
int dx[C] = { 2, 1,-1,-2,-2,-1, 1, 2};
int dy[C] = { 1, 2, 2, 1,-1,-2,-2,-1};

/* ナイトを進める．i-1 手まではすでに盤面 h[ ][ ]に記入済みとする.
    i : 次の指し手の番号(何手目か)
    x : 現在のナイトの x 座標
```

i : 次の手の番号, (x,y):現在の座標,
戻り値:解発見／未発見(=true／false)

i 手目の指し手として座標(x,y)にナイトを進める

座標(x,y)に進めた直前の指し手を取り消す

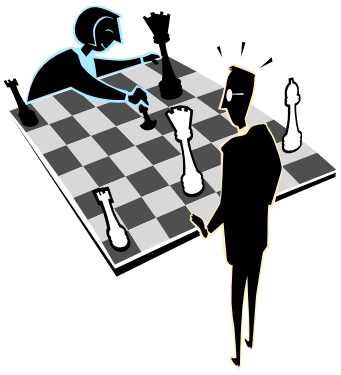


“マス目(0,0)は第１番目の指し手で訪問された”

```
49      y : 現在のナイトの y 座標
50      戻り値 : 真偽値(TRUE または FALSE).
51              解が見つかった場合---TRUE, 見つからない場合---FALSE  */
52
53  int try( int i, int x, int y )
54  {
55      int k= -1;  /* ナイトの可能な行き先の候補を表す. ループ内では 0 ≦ k ≦ 7 */
56      int ok;     /* 成功したか否かを表す真偽値 */
57      int x1,y1;  /* 次の行き先の X 座標 Y 座標 */
58
59      do {
60          k++;
61          ok = FALSE;
62          x1=x+dx[k]; y1=y+dy[k]; /* ナイトの可能な行き先のうち k 番目を選択 */
63          if ( 0<=x1 && x1<N && 0<=y1 && y1<N && h[x1][y1]==0 ) {
64                  /* 座標(x1,y1)が盤面内でまだ訪れたことがないのなら */
65              move_to( i, x1, y1 );  /* そこに進めてみる */
66              if ( i < NSQR ) {      /* まだ行っていないマス目があるなら */
67                  ok = try( i+1, x1, y1);     /* (i+1)手目以降を試みる */
68                  if ( !ok )  undo(x1, y1);  /* 失敗したら現在の指し手を止める */
69              }
70              else
71                  ok = TRUE;
72          }
73      } while ( !ok  &&  k < C-1 );
74      /* 解が見つかっておらず, ナイトの可能な行き先がまだあるときは繰り返す.  */
75      return( ok );
76  }
77
78  /* i 手目の指し手として座標(x,y)にナイトを進める */
79  void move_to( int i, int x, int y )
80  {
81      h[x][y] = i;
82  }
83
84  /* 座標(x,y)に進めた直前の指し手を取り消す */
85  void undo( int x, int y )
86  {
87      h[x][y] = 0;
88  }
89
90  /* 盤面を印刷する */
91  void print_knights( void )
92  {
93      int i, j;
94
95      for( j=0; j<N; j++ ){
96          for( i=0; i<N; i++ )
```

h[ ][ ]は大域変数

i は現在の手数を表している



```
97                printf(" %2d", h[i][j] );
98            printf("¥n");
99        }
100   }
101
102
103   【実行結果】
104   (盤面上の各マス目に，何手目でそのマスにナイトが訪れたかが記される)
105   ------------------------------------------------------------
106     1 38 59 36 43 48 57 52
107    60 35  2 49 58 51 44 47
108    39 32 37 42  3 46 53 56
109    34 61 40 27 50 55  4 45
110    31 10 33 62 41 26 23 54
111    18 63 28 11 24 21 14  5
112     9 30 19 16  7 12 25 22
113    64 17  8 29 20 15  6 13
114
```

2 桁で表示

2 桁に統一して見易く表示.

(途中経過を表示したもの)

```
  1  0  0  0  0  0  0  0
  0  0  2  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
==== 2 (1th)
  1  0  0  0  0  0  0  0
  0  0  2  0  0  0  0  0
  0  0  0  0  3  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
==== 3 (2th)
  1  0  0  0  0  0  0  0
  0  0  2  0  0  0  0  0
  0  0  0  0  3  0  0  0
  0  0  0  0  0  0  4  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
==== 4 (3th)
```

(途中省略)

```
  1  0  0  0  0  0  0  0
  0  0  2  0  0  0  0  0
  0 32  0  0  3  0  0  0
 34  0  0 27  0  0  4  0
 31 10 33  0  0 26 23  0
 18 35 28 11 24 21 14  5
  9 30 19 16  7 12 25 22
  0 17  8 29 20 15  6 13
==== 35 (34th)
  1  0  0  0  0  0  0  0
  0  0  2  0  0  0  0  0
  0 32  0  0  3  0  0  0
 34  0  0 27  0  0  4  0
 31 10 33  0  0 26 23  0
 18 35 28 11 24 21 14  5
  9 30 19 16  7 12 25 22
 36 17  8 29 20 15  6 13
==== 36 (35th)
backtracked
  1  0  0  0  0  0  0  0
  0  0  2  0  0  0  0  0
  0 32  0  0  3  0  0  0
 34  0 36 27  0  0  4  0
 31 10 33  0  0 26 23  0
 18 35 28 11 24 21 14  5
  9 30 19 16  7 12 25 22
  0 17  8 29 20 15  6 13
==== 36 (36th)
```

(途中省略)

```
  1  0 47 52  0  0 49  0
 46  0  2  0 48 51  0  0
  0 32 43 38  3  0 41 50
 34 45 36 27 42 39  4  0
 31 10 33 44 37 26 23 40
 18 35 28 11 24 21 14  5
  9 30 19 16  7 12 25 22
  0 17  8 29 20 15  6 13
==== 52 (52th)
  1  0 47 52  0  0 49  0
 46 53  2  0 48 51  0  0
  0 32 43 38  3  0 41 50
 34 45 36 27 42 39  4  0
 31 10 33 44 37 26 23 40
 18 35 28 11 24 21 14  5
  9 30 19 16  7 12 25 22
  0 17  8 29 20 15  6 13
==== 53 (53th)
backtracked
backtracked
  1  0 47  0  0  0 49 52
 46  0  2  0 48 51  0  0
  0 32 43 38  3  0 41 50
 34 45 36 27 42 39  4  0
 31 10 33 44 37 26 23 40
 18 35 28 11 24 21 14  5
  9 30 19 16  7 12 25 22
  0 17  8 29 20 15  6 13
==== 52 (54th)
backtracked
backtracked
backtracked
  1  0 47  0  0  0 49  0
 46  0  2  0 48  0  0  0
  0 32 43 38  3 50 41  0
 34 45 36 27 42 39  4  0
 31 10 33 44 37 26 23 40
 18 35 28 11 24 21 14  5
  9 30 19 16  7 12 25 22
  0 17  8 29 20 15  6 13
==== 50 (55th)
```

(途中省略)

```
==== 55 (241457th)
backtracked
backtracked
backtracked
backtracked
  1  0 47 44 41  0  0 52
 46 43  2  0 48  0 40  0
  0 32 45 42  3 38 51  0
 34  0 36 27  0 49  4 39
 31 10 33  0 37 26 23 50
 18 35 28 11 24 21 14  5
  9 30 19 16  7 12 25 22
  0 17  8 29 20 15  6 13
==== 52 (241458th)
```

(以下省略)

(参考： 8250731th　で終了)

```
/*************************************************************
     「アルゴリズムとデータ構造」
     サンプルプログラム　nqueen.c
     <<バックトラックアルゴリズムの例: n クイーン>>
     copyright (c)    T.Mori <mori@forest.dnj.ynu.ac.jp>
*************************************************************/
#include <stdio.h>
#define TRUE       1
#define FALSE      0
#define N          8

int q_pos[N];   /* q_pos[i]は, i 番めの列のクィーンの位置.  0 <= q_pos[i] <= N-1 */
int q_row[N];   /* q_row[j]は，j 番めの行にクィーンがいない場合，TRUE */
        /* q_se[k]は，k 番めの＼方向対角線にクィーンがいない場合，TRUE */
int q_se[2*N-1]; /* -(N-1)<=col-row<=(N-1) --> 0<=col-row+N-1<=2(N-1) */
        /* q_sw[l]は，l 番めの／方向対角線にクィーンがいない場合，TRUE */
int q_sw[2*N-1];   /* 0 <= col+row <= 2(N-1) */

int try( int col );
void try_all( int col );
void init_queen( void );
void set_queen( int row, int col );
void rm_queen( int row, int col );
void print_queen( void );

int main(void)
{
    int i;

    /* 最初の解を求めるプログラム例 */
    printf("N クイーン: 最初の解を求める例¥n");
    init_queen( );     /* 盤面データ初期化 */
    if ( try(0) )  print_queen( );     /* 配置を試みる. */
    else   printf("解なし¥n");

    /* すべての解を求めるプログラム例 */
    printf("N クイーン: 全ての解を求める例¥n");
    init_queen();    /* 盤面データ初期化 */
    try_all(0);
    return 0;
}

void init_queen(void)    /* 盤面データ初期化 */
{
    int i;

    for( i=0; i<N; i++ )  q_row[i]=TRUE;
    for( i=0; i<2*N-1; i++ ){  q_se[i] = TRUE;  q_sw[i] = TRUE;  }
```

8×8 マスであることを示す

クイーンを配置.

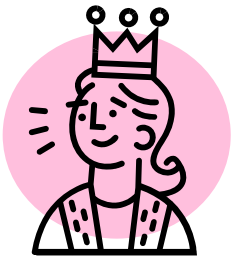
クイーンを配置. 全ての行を調べる.

盤面データ初期化

row 行 col 列にクイーンを配置

row 行 col 列のクイーンを取り除く

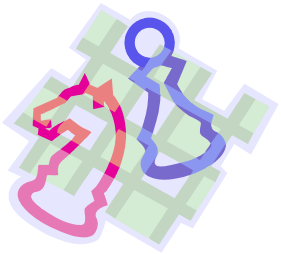
盤面を印刷



プログラムリストの行数を短くするため, 表記を短縮しています. 見づらくなるので真似をしないで下さい.

```
49  }
50  /* クイーンを配置する．col-1 列目まではすでに盤面に配置済みとする．
51     col ： 次の考慮すべき列,
52     戻り値 ： 真偽値( TRUE または FALSE ).
53              解が見つかった場合---TRUE, 見つからない場合---FALSE   */
54  int try( int col )
55  {
56      int row;   /* 何行目にクイーンを配置するか表す．ループ内では 0≦row≦N*/
57      int ok;    /* 成功したか否かを表す真偽値 */
58
59      row=-1;
60      do {
61          row++;
62          ok = FALSE;
63          if ( q_row[row] && q_se[col-row+N-1] && q_sw[col+row] ) {
64              /* row 行 col 列が他のクイーンの効き筋になっていなければ */
65              set_queen( row, col );  /* クイーンを配置 */
66              if ( col < N-1 ) {
67                  ok = try( col+1 );  /* 残りの列にも配置する */
68                  if ( !ok )  rm_queen( row, col );
69                                      /* 失敗なら取り除く(バックトラック) */
70              }  else  ok = TRUE;
71          }
72      } while( !ok  &&  row < N-1 );
73      return( ok );
74  }
75
76  /* 解の番号 */
77  int number = 0;
78  /*  try --- クイーンを配置する．col-1 列目まではすでに盤面に配置済みとする．
79      col ： 次の考慮すべき列   */
80  void try_all( int col )
81  {
82      int row, ok;
83
84      for( row=0; row<N; row++ ) { /* 全解探索なのですべての行を調べる */
85          if ( q_row[row] && q_se[col-row+N-1] && q_sw[col+row] ) {
86              set_queen( row, col );
87              if ( col < N-1 )  try_all( col+1 );
88              else {
89                  number++; printf("解番号 %d¥n", number);
90                  print_queen();
91              }  /* N 個の配置が終ったら，解が求まっているのでそれを表示 */
92              rm_queen( row, col );
93              /* 解が求まっても求まらなくてもクイーンを取り除く
94               (強制バックトラック) */
95          }
96      }
```

```
97  }
98
99  void set_queen( int row, int col )   /* row 行 col 列にクイーンを配置 */
100 {
101     q_pos[col] = row;                q_row[row] = FALSE;
102     q_se[col-row+N-1] = FALSE;    q_sw[col+row] = FALSE;
103 }
104
105 void rm_queen( int row, int col )   /* row 行 col 列のクイーンを取り除く */
106 {
107     q_row[row] = TRUE;
108     q_se[col-row+N-1] = TRUE;    q_sw[col+row] = TRUE;
109 }
110
111 void print_queen( void )   /* 盤面を印刷 */
112 {
113     int i,j;
114
115     for( i=0; i<N; i++ ) {
116         for( j=0; j<N; j++ )
117             if ( i == q_pos[j] )   printf(" Q");   else   printf(" .");
118         printf("¥n");
119     }
120     printf("¥n");
121 }
122
```



【実行結果】

N クイーン:
最初の解を求める例

```
Q . . . . . .
. . . . . Q .
. . . . Q . .
. . . . . . Q
. Q . . . . .
. . . Q . . .
. . . . . Q .
. . Q . . . .
```

N クイーン:
全ての解を求める例

解番号 1

```
Q . . . . . .
. . . . . Q .
```

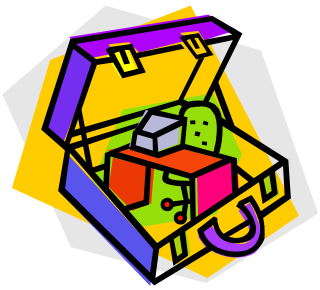
（長いので途中省略）

解番号 92

```
. . Q . . . . .
. . . . . Q . .
. . . Q . . . .
. Q . . . . . .
. . . . . . . Q
. . . . Q . . .
. . . . . Q . .
Q . . . . . .
```

```
1   /*************************************************************
2       「アルゴリズムとデータ構造」
3       サンプルプログラム　optimum.c
4       <<バックトラックアルゴリズムの例: 最適選択問題---旅行カバンの中身>>
5       copyright (c)  T.Mori <mori@forest.dnj.ynu.ac.jp>
6   *************************************************************/
7   #include <stdio.h>
8   #define TRUE        1
9   #define FALSE       0
10  #define NAMELEN     20
```

```
11  struct object {   /* 品物の情報 */
12      char name[20];   /* 名称 */
13      int value;          /* 旅をする上での価値 */
14      int weight;         /* 重さ (g) */
15  };
16  /* 旅行カバンにつめる物品の候補のリスト */
17  #define N   10
18  struct object a[N] ={        /* {名称, 価値, 重さ} */
19      {"本", 10, 500},{"傘", 90, 500},{"下着", 90, 300},{"ジャケット", 30, 1000},
20      {"薬", 100, 20},{"MD プレーヤ", 20, 400},{"地図", 70, 200},
21      {"宿チケット", 100, 10},{"航空券", 100, 10},{"洗面用具", 50, 300}   };
22  /* sel[]     --- カバンにつめる品物の集合を表す配列
23        品物 a[i] をカバンにつめる場合,      sel[i] == TRUE
24        品物 a[i] をカバンにつめるない場合, sel[i] == FALSE
25     optsel[] --- カバンにつめる品物の集合で, 最も価値の高い組合わせを表す配列
26        内容は sel[]と同じ構造をしている.   */
27  int sel[N], optsel[N];
28  /* limw: 制限重量, totv: 全ての品物の価値の合計,
29     maxv: 最も価値の高い品物の組み合わせ(optsel)における価値   */
30  int limw,totv,maxv;
31
32  void empty_set(int s[ ], int n );
33  void try( int i, int tw, int av );
34
35  int main(void)
36  {
37      int i;
38
39      /* 総価値の計算 */
40      totv = 0;
41      for(i=0; i<N; i++)
42          totv += a[i].value;
43      /* いくつかの制限重量で計算 */
44      for (limw=500; limw <=3000; limw += 500) {
45          /* 選択品物の情報初期化 */
46          maxv=0;   empty_set(sel,N);   empty_set(optsel,N);
47          try( 0, 0, totv);       /* 組合せを求める */
48          /* 結果の印刷 */
49          printf("制限重量 %4dg の場合: 総価値 %3d:",limw,maxv);
50          for( i=0; i<N; i++ )
51              if ( optsel[i] )   printf(" %s", a[i].name);
52          printf("¥n");
53      }
54      return 0;
55  }
56
57  /* 品物の集合を空にする */
58  void   empty_set( int s[ ], int n )
```



```
59  {   while(n>0) {   n--; s[n]=FALSE;   }   }
60  /* 品物の集合をコピーする */
61  void   copy_set( int d[ ], int s[ ], int n )
62  {   while(n>0) {   n--; d[n]=s[n];   }   }
63  /* 品物の集合に新しい品物を一つ加える */
64  void   add_member( int s[ ], int i )
65  {   s[i]=TRUE;   }
66  /* 品物の集合から品物一つ削除する */
67  void   delete_member( int s[ ], int i )
68  {   s[i]=FALSE;   }
69
70  /* 品物の組合せを求める. 品物(i-1)まではカバンにつめるかつめないかは決定済みとする.
71        i : 次に考慮する品物の番号, tw: これまで選んだ品物の総重量
72        av: 現在の選び方から, 今後まだ達成できうる価値 */
73  void try( int i, int tw, int av )
74  {
75      int av1;
76
77      /* 現在考慮すべき品物(番号 i)をカバンにつめる場合 */
78      if ( tw + a[i].weight <= limw ) {    /* 制限重量を越えていないのなら */
79          add_member( sel, i );            /* カバンにつめる品物の集合に登録 */
80          if ( i<N-1 )                     /* 全ての品物を試したわけでないのなら */
81              try( i+1, tw+a[i].weight, av );   /* 残り品物についても考える */
82          else if (av>maxv) { /* 全品物を試し終わり今求めた組合せの価値が高い */
83              maxv = av;
84              copy_set(optsel, sel, N);   /* その組合せを現在の最適解として登録 */
85          }
86          delete_member(sel,i); /* 別解のため品物 i をつめる品の集合から除く */
87      }
88      /* 現在考慮すべき品物(番号 i)をカバンにつめない場合 */
89      av1 = av-a[i].value; /* 品物 i をつめないので到達可能な価値はその分下がる. */
90      if (av1>maxv) { /* 到達可能な価値が今までの最適値より大きければ望みがある */
91          if ( i<N-1 )                     /* 全ての品物を試したわけでないのなら */
92              try( i+1, tw, av1 );    /* 残りの品物についても考える */
93          else {   /* 到達可能な価値が今までの最適値よりも大きく, 残りの品物も
94                      ないので, 最適解として登録 */
95              maxv = av1;   copy_set( optsel, sel, N );
96          }
97      }
98  }
```

【実行結果】

制限重量　500g の場合: 総価値 400: 下着 薬 宿チケット 飛行機チケット
制限重量 1000g の場合: 総価値 520: 下着 薬 地図 宿チケット 飛行機チケット 洗面用具
制限重量 1500g の場合: 総価値 610: 傘 下着 薬 地図 宿チケット 飛行機チケット 洗面用具
制限重量 2000g の場合: 総価値 630: 傘 下着 薬 MD プレーヤ 地図 宿チケット 飛行機チケット 洗面用具
制限重量 2500g の場合: 総価値 640: 傘 下着 ジャケット 薬 地図 宿チケット 飛行機チケット 洗面用具
制限重量 3000g の場合: 総価値 660: 傘 下着 ジャケット 薬 MD プレーヤ 地図 宿チケット 飛行機チケット 洗面用具