包除原理

解ける数え上げの範囲を広げよう

tsutaj (@_TTJR_)

Hokkaido University M1

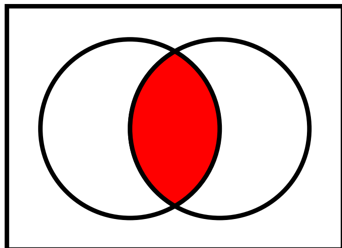
October 16, 2018

1. 入門編
   - 包除原理とは
   - 包除原理の証明
2. 包除原理の問題・初級編
   - オイラーの ϕ 関数
   - Uncommon
   - Ball and Boxes 3
   - Iahub and Permutations
3. 包除原理の問題・中級編
   - LCM Rush
   - Enumeration
   - 天下一ボディービルコンテスト
4. 包除原理の問題・上級編
   - 出席番号 (2)
   - Rotated Palindromes
   - Everything on It
5. 練習問題

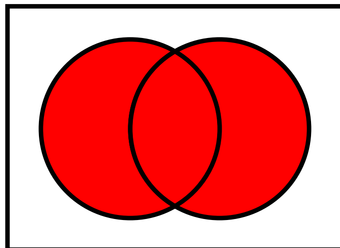
包除原理とは？

- 集合の「積集合」と「和集合」，求めるのはどちらが簡単？

積集合 (intersection)

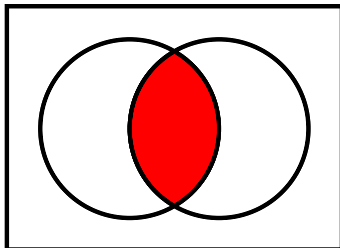


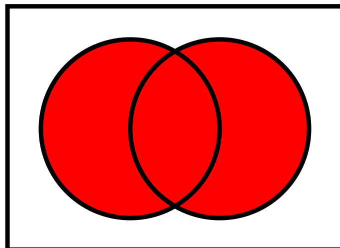
和集合 (union)



包除原理とは？

- 集合の「積集合」と「和集合」，求めるのはどちらが簡単？
- 一般に積集合のほうが簡単
  - 積集合 (intersection) は複数の条件でフィルタリングされたものを集めた結果なので，直接求めやすい
  - 和集合 (union) は複数の条件のうち，どれか 1 つでも当てはまっているものを集めた結果なので，求めにくい

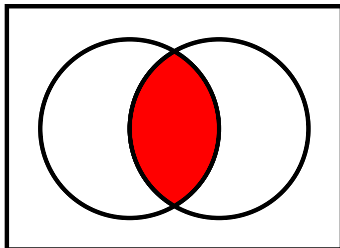




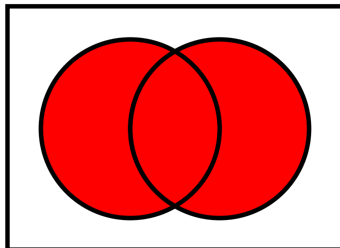
包除原理とは？

- 集合の「積集合」と「和集合」，求めるのはどちらが簡単？
- 一般に積集合のほうが簡単
  - 積集合 (intersection) は複数の条件でフィルタリングされたものを集めた結果なので，直接求めやすい
  - 和集合 (union) は複数の条件のうち，どれか 1 つでも当てはまっているものを集めた結果なので，求めにくい
- 直接求めにくい和集合を，積集合の組み合わせで求めよう
- ここで出てくるのが，包除原理！

積集合 (intersection)

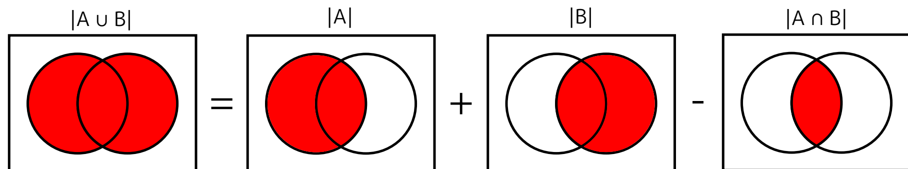


和集合 (union)



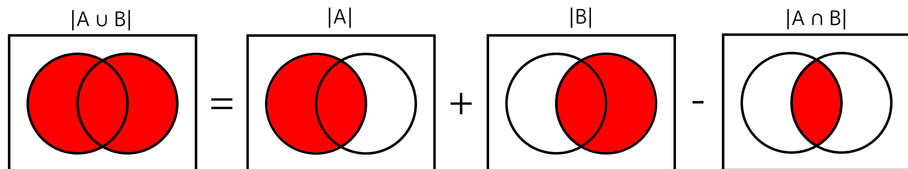
2 つの集合に対する包除原理

- 2 つの集合の和集合を求めるには？
- 下図のように足し引きすればよい
  - $|A| + |B|$ をすると $|A \cap B|$ が 2 回足されてしまうため，$|A \cap B|$ を引く



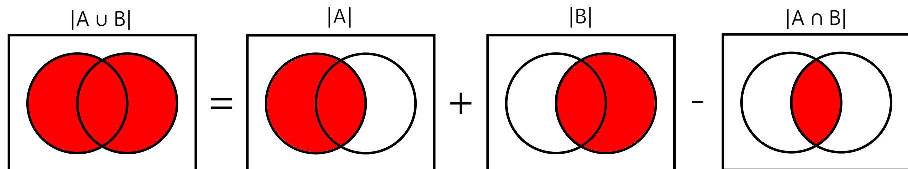
2 つの集合に対する包除原理

- 2 つの集合の和集合を求めるには？
- 下図のように足し引きすればよい
  - $|A| + |B|$ をすると $|A \cap B|$ が 2 回足されてしまうため，$|A \cap B|$ を引く
- 実際の例
  - 30 以下の自然数で，3 で割れるか 5 で割れるような数はいくつ？



2 つの集合に対する包除原理

- 2 つの集合の和集合を求めるには？
- 下図のように足し引きすればよい
  - $|A| + |B|$ をすると $|A \cap B|$ が 2 回足されてしまうため，$|A \cap B|$ を引く
- 実際の例
  - 30 以下の自然数で，3 で割れるか 5 で割れるような数はいくつ？
- 2 つの集合の場合は簡単だけど，N 個の集合になったらどうなる？
- 次ページの式のようにかける



N 個の集合に対する包除原理

包除原理

N 個の集合 $A_1, A_2, \ldots, A_N$ の和集合に属する要素数

$$
\begin{aligned}
|A_1 \cup \cdots \cup A_N| &= \sum_{k=1}^{N} (-1)^{k-1} \times (k \text{ 個の「かつ」の総和}) \\
&= \sum_{i} |A_i| - \sum_{i<j} |A_i \cap A_j| + \sum_{i<j<k} |A_i \cap A_j \cap A_k| - \cdots
\end{aligned}
$$

- これは本当に成り立つの？？
- $A_1 \cup \cdots \cup A_N$ をいくつかの領域に分割し，各領域について一度だけ足されていれば OK
- 任意の m に付いて，集合 m 個の共通部分が一度だけ足されているか見てみよう

包除原理の証明

二項定理による証明 (集合 m 個の共通部分が一度だけ足されている？)

- 各項における足し引きの考察

包除原理の証明

二項定理による証明 (集合 m 個の共通部分が一度だけ足されている？)

- 各項における足し引きの考察
  - 右辺の第一項 (集合 1 個の足し引き) において m 回足される
  - 右辺の第二項 (集合 2 個の足し引き) において ${}_mC_2$ 回引かれる
  - ... k が奇数の場合，右辺の第 k 項において ${}_mC_k$ 回足される
  - ... k が偶数の場合，右辺の第 k 項において ${}_mC_k$ 回引かれる

包除原理の証明

二項定理による証明 (集合 m 個の共通部分が一度だけ足されている？)

- 各項における足し引きの考察
  - 右辺の第一項 (集合 1 個の足し引き) において m 回足される
  - 右辺の第二項 (集合 2 個の足し引き) において ${}_mC_2$ 回引かれる
  - ... k が奇数の場合，右辺の第 k 項において ${}_mC_k$ 回足される
  - ... k が偶数の場合，右辺の第 k 項において ${}_mC_k$ 回引かれる
- 以上を踏まえると，合計で $\sum_{k=1}^{m}(-1)^{k-1}{}_mC_k$ 回足される
- $\sum_{k=1}^{m}(-1)^{k-1}{}_mC_k$ が 1 と等しければ証明おわり

包除原理の証明

二項定理より，

$$
\begin{aligned}
0 &= (1-1)^m = \sum_{k=0}^{m} 1^{m-k}(-1)^k {}_mC_k \\
&= -\sum_{k=0}^{m} (-1)^{k-1} {}_mC_k (-1\text{ の指数をいじって符号反転}) \\
&= 1 - \sum_{k=1}^{m} (-1)^{k-1} {}_mC_k (k=0\text{ だけ抜き出す})
\end{aligned}
$$

- $1 - \sum_{k=1}^{m} (-1)^{k-1} {}_mC_k = 0$ だから，$\sum_{k=1}^{m} (-1)^{k-1} {}_mC_k = 1$
- よって全領域について一度だけ足されている！

包除原理の証明

二項定理より，

$$
\begin{aligned}
0 &= (1-1)^m = \sum_{k=0}^{m} 1^{m-k}(-1)^k {}_mC_k \\
&= -\sum_{k=0}^{m}(-1)^{k-1}{}_mC_k(-1 \text{ の指数をいじって符号反転}) \\
&= 1-\sum_{k=1}^{m}(-1)^{k-1}{}_mC_k(k=0 \text{ だけ抜き出す})
\end{aligned}
$$

- $1-\sum_{k=1}^{m}(-1)^{k-1}{}_mC_k = 0$ だから，$\sum_{k=1}^{m}(-1)^{k-1}{}_mC_k = 1$
- よって全領域について一度だけ足されている！
- 色々長くなりましたが，結局「奇数個の集合の intersection は足し，偶数個なら引く」ことをすればよいです

包除原理

- 証明に関する補足
  - 数学的帰納法でも示すことができます
  - 詳細は省略します (ググったら出てくるので，気になったら調べてね)
- これで包除原理の入門は終わりです
- ここからは実際に問題を考察してみましょう！
  - 初級では，簡単目の典型を扱います
  - 中級では，高速ゼータ変換・メビウス変換や DP を交えた少し難しい問題を扱います
  - 上級では，考察が難しい問題を扱います

オイラーの ϕ 関数

オイラーの ϕ 関数

正の整数 n について，1 から n までの自然数のうち n と互いに素なものの個数を求めよ．

- $1 \leq n \leq 10^9$

出典: AOJ NTL_1_D Euler's Phi Function ▸ Link

サンプル

入力 1 :	N = 6	出力: 2
入力 2 :	N = 1000000	出力: 400000

- どのような集合を考えて数え上げたら良いか？

オイラーの ϕ 関数

- 互いに素なもの → n が持つどの素因数も持たない
- 余事象を考えて，「互いに素でないもの」を数えることを考える
  - 互いに素でないもの → n が持つ素因数を少なくとも 1 つ持つ

オイラーの ϕ 関数

- 互いに素なもの → n が持つどの素因数も持たない
- 余事象を考えて，「互いに素でないもの」を数えることを考える
  - 互いに素でないもの → n が持つ素因数を少なくとも 1 つ持つ
- n が持つ素因数を $p_1, p_2, \ldots, p_m$ と置く
- p_1 を素因数として持つ n 以下の自然数の集合を P_1 と定義
  - 以下同様に $P_2, \ldots, P_m$ を定義
- 「互いに素でない」自然数を表す集合はどう書ける？

オイラーの ϕ 関数

- 互いに素なもの → n が持つどの素因数も持たない
- 余事象を考えて，「互いに素でないもの」を数えることを考える
  - 互いに素でないもの → n が持つ素因数を少なくとも 1 つ持つ
- n が持つ素因数を $p_1, p_2, \ldots, p_m$ と置く
- p_1 を素因数として持つ n 以下の自然数の集合を P_1 と定義
  - 以下同様に $P_2, \ldots, P_m$ を定義
- 「互いに素でない」自然数を表す集合はどう書ける？
  - n の素因数を少なくとも 1 つ持つんだから・・・？
  - $P_1 \cup P_2 \cup \cdots \cup P_m$ と書ける！
  - この和集合の要素数は，包除原理を使って求められる

オイラーの ϕ 関数

つまり，解法をまとめると

1. n の素因数を列挙する
2. 素因数によって定められる集合を 1 個以上選択し，その積集合の要素数を足し引き
   - 「互いに素でない」自然数を数え上げる
   - あり得る集合の選択方法を全て試す必要があるので，bit 演算と相性〇
3. n から，上で求めた値を引く
   - 「互いに素である」自然数を数え上げる

実装は次ページで紹介

オイラーの ϕ 関数

実装 (C++)

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int N, origN; cin >> N; origN = N;
    vector<int> div;
    for(int d=2; d*d<=N; d++) {
        if(N % d == 0) {
            div.push_back(d);
            while(N % d == 0) N /= d;
        }
    }
    if(N != 1) div.push_back(N);

    int M = div.size(), ans = 0;
    // N と互いに素でないものを数え上げる
    for(int bit=1; bit<(1<<M); bit++) {
        int popcnt = __builtin_popcount(bit);
        int mul = 1;
        for(int i=0; i<M; i++) {
            if(bit >> i & 1) mul *= div[i];
        }
        // 集合の数による場合分け
        if(popcnt % 2 == 1) ans += origN / mul;
        else                ans -= origN / mul;
    }
    // 互いに素なもの = 全体から素でないものを引いたもの
    ans = origN - ans;
    cout << ans << endl;
    return 0;
}
```

Uncommon

Uncommon

N 個の異なる整数 $a_1, \ldots, a_N$ と整数 M が与えられるので，1 以上 M 以下のそれぞれの整数 i について，$a_1, \ldots, a_N$ のうち互いに素であるものの個数を求めてください．

- $1 \leq N, M \leq 10^5$
- $1 \leq a_i \leq 10^5$
- a_i は全て異なる
- 入力値はすべて整数

出典: 「みんなのプロコン 2018」決勝 オープンコンテスト A ▸ Link

- 先ほどの「オイラーの ϕ 関数」で用いたアイデアが役に立つと思います

Uncommon

- 集合の作り方は，先ほどの問題と全く同じ！
  - 集合の中身が「n 以下の自然数」ではなく，「与えられた整数 $a_1, \ldots a_N$」に変化しただけ
  - というわけで求め方の手順は省略します
- 1 から M までの全ての数字について答えなければならないけど，計算量は大丈夫？
  - $M \leq 10^5$ であることから，素因数の数はとても少ない (6 個程度)
  - これくらいならば普通にループを回せるので問題ない
- 素因数列挙を呼ぶことが多くなるので，関数化すると良いかも

実装 (C++)

```
#include <iostream>
#include <vector>
using namespace std;
vector<int> getPrimes(int val) {
    vector<int> res;
    for(int k=2; k*k<=val; k++) {
        if(val % k == 0) {
            res.push_back(k);
            while(val % k == 0) val /= k;
        }
    }
    if(val > 1) res.push_back(val);
    return res;
}
int main() {
    int N, M, cnt[100010] = {}; cin >> N >> M;
    for(int i=0; i<N; i++) {
        int val; cin >> val;
        vector<int> div = getPrimes(val);
        for(int bit=1; bit<(1<<div.size()); bit++) {
            int mul = 1; // val の素因数の掛けあわせ
            for(int i=0; i<div.size(); i++) if(bit >> i & 1) mul *= div[i];
            cnt[mul]++;
        }
    }
    for(int val=1; val<=M; val++) {
        vector<int> div = getPrimes(val); int ans = N;
        for(int bit=0; bit<(1<<div.size()); bit++) {
            int mul = 1;
            for(int i=0; i<div.size(); i++) if(bit >> i & 1) mul *= div[i];
            ans += cnt[mul] * (__builtin_popcount(bit) % 2 ? (-1) : (+1)); // 包除
        }
        cout << ans << endl;
    }
    return 0;
}
```

Ball and Boxes 3

Ball and Boxes 3

以下の条件を満たす，n 個の区別できるボールを k 個の区別できる箱に入れる方法は何通りあるか？

- どのボールも，必ずいずれかの箱に入れる
- どの箱にも 1 つ以上のボールを入れる

- $1 \leq n \leq 10^3$
- $1 \leq k \leq 10^3$

出典: AOJ DPL_5_C Ball and Boxes 3 Link

サンプル

```
入力 1 :  N = 4, K = 3    出力:  36
入力 2 :  N = 10, K = 3   出力:  55980
```

Ball and Boxes 3

- これも，最初に余事象を考えてみよう
  - 「どの箱にもボールが 1 つ以上入っている」の余事象は？

Ball and Boxes 3

- これも，最初に余事象を考えてみよう
  - 「どの箱にもボールが 1 つ以上入っている」の余事象は？
  - 「少なくとも 1 つの箱について，ボールが入っていない」

Ball and Boxes 3

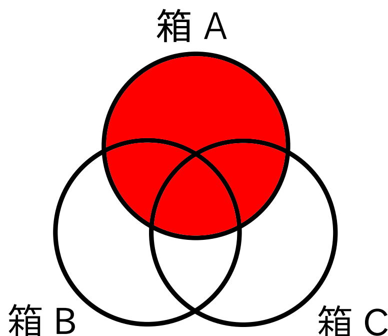
- これも，最初に余事象を考えてみよう
  - 「どの箱にもボールが 1 つ以上入っている」の余事象は？
  - 「少なくとも 1 つの箱について，ボールが入っていない」
- ボールが絶対に入らないところを決め打ちしたい
  - 先ほどの問題とほぼ同じ解き方になる！
  - しかし今回は集合の数がとても多い (10^3 個) ので，bit 演算で全部試すのは不可能！

Ball and Boxes 3

- これも，最初に余事象を考えてみよう
  - 「どの箱にもボールが 1 つ以上入っている」の余事象は？
  - 「少なくとも 1 つの箱について，ボールが入っていない」
- ボールが絶対に入らないところを決め打ちしたい
  - 先ほどの問題とほぼ同じ解き方になる！
  - しかし今回は集合の数がとても多い (10^3 個) ので，bit 演算で全部試すのは不可能！
- 対称性 を使おう！！

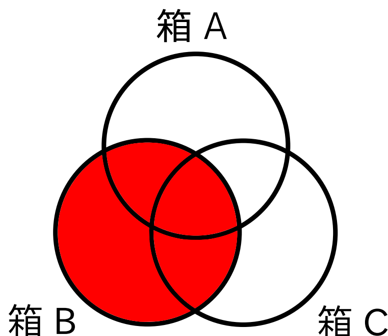
Ball and Boxes 3

- 例: n 個のボールを $k=3$ 個の箱に入れる
- 3 個の箱のうち，絶対に入れないところを 1 個決めて場合の数を計算
- 箱 A に絶対にボールを入れないようにする場合の数は？
  - 入れる場所の候補が $k-1$ 通りで，それを n 回やるので・・・
  - $(k-1)^n = 2^n$ 通り



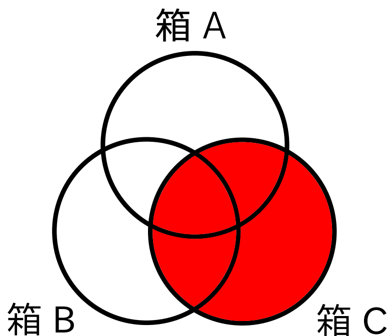
Ball and Boxes 3

- 例: n 個のボールを $k = 3$ 個の箱に入れる
- 3 個の箱のうち，絶対に入れないところを 1 個決めて場合の数を計算
- 箱 B に絶対にボールを入れないようにする場合の数は？
  - 入れる場所の候補が $k - 1$ 通りで，それを n 回やるので・・・
  - $(k - 1)^n = 2^n$ 通り



Ball and Boxes 3

- 例: n 個のボールを $k = 3$ 個の箱に入れる
- 3 個の箱のうち，絶対に入れないところを 1 個決めて場合の数を計算
- 箱 C に絶対にボールを入れないようにする場合の数は？
  - 入れる場所の候補が $k - 1$ 通りで，それを n 回やるので・・・
  - $(k-1)^n = 2^n$ 通り



Ball and Boxes 3

- 絶対に入れない箱の個数が同じならば，場合の数は同じ
  - 先ほどの例では，箱 A のみ・箱 B のみ・箱 C のみに絶対にボールを入れないようにする場合の数は全て等しかった
- combination を使うと，これらをまとめて処理できる！！
  - 先ほどの例だと，${}_3C_1 2^n = 3 \times 2^n$ 通り，とまとめて求められる
  - 箱の個数に対して指数個あった状態が，線形個に改善された！

対称性

全体で n 個ある集合から k 個選びとった集合同士の intersection の要素数が全て等しいとき，combination でまとめて計算可能である性質

Ball and Boxes 3

実装 (C++)

- 余談ですが，この制約下なら DP でも解けます (詳細は略)

```cpp
#include <iostream>
using namespace std;

using ll = long long int;
const ll MOD = 1000000007;
const int MAXN = 2000;
ll mod_pow(ll N, ll K) {… // 累乗 (略)
}
ll comb[MAXN + 10][MAXN + 10];
void init_comb() {… // 組み合わせ配列初期化 (略)
}

int main() {
    init_comb();
    ll N, K, ans = 0; cin >> N >> K;

    // 絶対に空にする箱の個数を決める
    for(int i=1; i<=K; i++) {
        int rest = K - i;
        ll val = comb[K][i] * mod_pow(rest, N) % MOD; // 対称性よりまとめて計算可能
        if(i % 2 == 1) ans = (ans + val      ) % MOD;
        else           ans = (ans - val + MOD) % MOD;
    }
    // 「どの箱にも 1 つ以上」の制約がなければ k^n 通り
    ans = (mod_pow(K, N) - ans + MOD) % MOD;
    cout << ans << endl;
    return 0;
}
```

Iahub and Permutations

Iahub and Permutations

長さ N の順列 $\{a_1, a_2, \ldots, a_n\}$ が，一部穴あきの状態で与えられる．順列の任意の要素に対して $a_i \neq i$ になるように順列内の穴を埋めることを考える．作れる順列は何通りあるか？

- $2 \leq N \leq 2,000$

出典: Codeforces Round #198 E Link

サンプル

入力 1 : `N = 5, a = {E, E, 4, 3, E}` 出力: 2

- 1, 2, 5 個目に穴が開いている
- 条件を満たす順列は $\{2, 5, 4, 3, 1\}$ と， $\{5, 1, 4, 3, 2\}$

Iahub and Permutations

- この問題も，先ほどの問題と考え方はほぼ同じ！
- 余事象を考えよう

Iahub and Permutations

- この問題も，先ほどの問題と考え方はほぼ同じ！
- 余事象を考えよう
  - 「$a_i = i$ である箇所が少なくとも 1 箇所以上存在する」順列

Iahub and Permutations

- この問題も，先ほどの問題と考え方はほぼ同じ！
- 余事象を考えよう
  - 「$a_i = i$ である箇所が少なくとも 1 箇所以上存在する」順列
- $a_i = i$ になる可能性のある場所はいくつあるか考える
  - 「穴の個数」と等しくはないので注意！！
  - 例えば順列が $\{\mathsf{E}, 1, \mathsf{E}\}$ の場合，穴は 2 つだが $a_i = i$ になる可能性がある要素は 3 のみなので 1 個

Iahub and Permutations

- この問題も，先ほどの問題と考え方はほぼ同じ！
- 余事象を考えよう
  - 「$a_i = i$ である箇所が少なくとも 1 箇所以上存在する」順列
- $a_i = i$ になる可能性のある場所はいくつあるか考える
  - 「穴の個数」と等しくはないので注意！！
  - 例えば順列が $\{\mathsf{E}, 1, \mathsf{E}\}$ の場合，穴は 2 つだが $a_i = i$ になる可能性がある要素は 3 のみなので 1 個
- $a_i = i$ になる要素の個数を決め打ちすると，これは対称性が成り立つので，まとめて処理できる！
- あとは組み合わせを足し引きすれば良い

Iahub and Permutations

よくある (と思われる) 質問

- 結局，$a_i = i$ になる可能性のある場所はどう数えるの？

Iahub and Permutations

よくある (と思われる) 質問

- 結局，$a_i = i$ になる可能性のある場所はどう数えるの？
    - i 番目のインデックスが空いており，かつ穴でない要素として自然数 i が使われていなければ，$a_i = i$ になる可能性がある

Iahub and Permutations

よくある (と思われる) 質問

- 結局，$a_i = i$ になる可能性のある場所はどう数えるの？
  - i 番目のインデックスが空いており，かつ穴でない要素として自然数 i が使われていなければ，$a_i = i$ になる可能性がある
- $a_i = i$ になる要素の個数を k 個と決め打ちした後は，どう足せばよいの？

Iahub and Permutations

よくある (と思われる) 質問

- 結局，$a_i = i$ になる可能性のある場所はどう数えるの？
  - i 番目のインデックスが空いており，かつ穴でない要素として自然数 i が使われていなければ，$a_i = i$ になる可能性がある
- $a_i = i$ になる要素の個数を k 個と決め打ちした後は，どう足せばよいの？
  - 以下，穴の個数を A とし，$a_i = i$ になる可能性がある要素の個数を B とする
    1. B 個の候補から k 個を持ってきたので ${}_BC_k$ 通り
    2. 穴の個数は A 個だったが，そのうち k 個は埋まっているので，残りの要素の配置パターンが $(A-k)!$ 通り
    3. あとは，k の偶奇を見て足し引き

実装 (C++)

- 余談ですが，このような順列は撹乱順列と呼ばれます

```cpp
#include <iostream>
using namespace std;
using ll = long long int;
const int MAXN = 2010;
const ll MOD = 1000000007;
int usedIdx[MAXN+10], usedVal[MAXN+10];
ll comb[MAXN+10][MAXN+10], perm[MAXN+10];
void init_comb_perm() { // 組み合わせと階乗 (略)
}
int main() {
    init_comb_perm();
    // 穴あきの個数と、a_i = i になるかもしれない個数を調べる
    ll N, cntFixed = 0, cntEmpty = 0, ans = 0; cin >> N;
    for(int i=0; i<N; i++) {
        int val; cin >> val; val--;
        if(val >= 0) usedVal[val] = true;
        else         usedIdx[i  ] = true, cntEmpty++;
    }
    for(int i=0; i<N; i++) if(usedIdx[i] and !usedVal[i]) cntFixed++;

    // 余事象を考え、a_i = i になる個数を決め打ち
    for(int i=1; i<=cntFixed; i++) {
        int add = comb[cntFixed][i] * perm[cntEmpty-i] % MOD;
        if(i % 2 == 1) ans = (ans + add      ) % MOD;
        else           ans = (ans - add + MOD) % MOD;
    }
    ans = (perm[cntEmpty] - ans + MOD) % MOD;
    cout << ans << endl;
    return 0;
}
```

包除原理の問題・中級編

ここからは中級編です！

- 少し高度な考察を要する包除原理
  - 包除原理の前のパートが重かったり，言い換えが大変だったり
- 高速ゼータ変換・高速メビウス変換
- DP が絡む包除原理

こちらについて触れていきます

LCM Rush

LCM Rush

2 つの正整数 N, K が与えられるので，1 以上 N 以下のすべての整数 i について $\mathrm{LCM}(i, K)$ を求め，その合計を求めよ．ここで，$\mathrm{LCM}(a, b)$ とは a と b の最小公倍数を指す．

- $1 \leq N, K \leq 10^9$

出典: AtCoder Beginner Contest 020 D ▸ Link

サンプル

入力 1 : `N = 4, K = 2` 出力: `14`

- $\mathrm{LCM}(1,2)+\mathrm{LCM}(2,2)+\mathrm{LCM}(3,2)+\mathrm{LCM}(4,2) = 2+2+6+4 = 14$

- どのようなアプローチで解いていくか？
- (ヒント: LCM をぐっとにらむと・・・？)

LCM Rush

- 求めるもの → $\sum_{i=1}^{N} \mathrm{LCM}(i, K)$

LCM Rush

- 求めるもの → $\sum_{i=1}^{N} \mathrm{LCM}(i, K)$
- まず，LCM だと扱いづらいので GCD (最大公約数) に変えてみよう
  - $\mathrm{LCM}(a, b) = \frac{ab}{\mathrm{GCD}(a,b)}$
  - 求めるもの 改 → $K \sum_{i=1}^{N} \frac{i}{\mathrm{GCD}(i,K)}$

LCM Rush

- 求めるもの → $\sum_{i=1}^{N} \mathrm{LCM}(i, K)$
- まず，LCM だと扱いづらいので GCD (最大公約数) に変えてみよう
  - $\mathrm{LCM}(a, b) = \frac{ab}{\mathrm{GCD}(a,b)}$
  - 求めるもの 改 → $K \sum_{i=1}^{N} \frac{i}{\mathrm{GCD}(i,K)}$
- ここで，GCD の取りうる値の個数について考えてみよう
  - 取りうる値は，K の約数に限られる
  - $K \leq 10^9$ より，この個数は最大でも $1,344$ 個

LCM Rush

- 求めるもの → $\sum_{i=1}^{N} \mathrm{LCM}(i, K)$
- まず，LCM だと扱いづらいので GCD (最大公約数) に変えてみよう
  - $\mathrm{LCM}(a, b) = \frac{ab}{\mathrm{GCD}(a,b)}$
  - 求めるもの 改 → $K \sum_{i=1}^{N} \frac{i}{\mathrm{GCD}(i,K)}$
- ここで，GCD の取りうる値の個数について考えてみよう
  - 取りうる値は，K の約数に限られる
  - $K \leq 10^9$ より，この個数は最大でも $1,344$ 個
- 同じ GCD を取るもの同士は，まとめて計算できそう
  - 注目する GCD の値を g をおく
  - $\sum_{1 \leq i \leq N, \mathrm{GCD}(i,K)=g} \frac{i}{g} = \sum_{1 \leq i \leq \lfloor \frac{N}{g} \rfloor, \mathrm{GCD}(i, \frac{K}{g})=1} i$
  - $K \sum_{1 \leq i \leq \lfloor \frac{N}{g} \rfloor, \mathrm{GCD}(i, \frac{K}{g})=1} i$ をまとめて計算できると嬉しい
    - これが高速に求められるのなら，元の式を GCD の種類ごとに分解することで高速に処理可能

LCM Rush

- GCD が同じ要素の和は，どのように高速に求められる？

LCM Rush

- GCD が同じ要素の和は，どのように高速に求められる？
- これは初級編でやった包除と実質同じ (総和パートを工夫すれば OK)

GCD が同じ要素の和を包除原理で求める例

- $\mathrm{GCD}(12, i) = 1$ になる，12 以下の自然数 i の和 (答えは 24)
  - 1 から 12 までの総和は $\frac{12(12+1)}{2} = 78$
  - 素因数は 2 と 3
  - 2 を素因数として持つ要素の和 → $2 + 4 + 6 + 8 + 10 + 12 = 42$
    - 要素の数は $\frac{12}{2} = 6$ 個 (初項 2 公差 2 の等差数列状)
    - 総和は $2 \times \frac{6(6+1)}{2} = 42$
  - 3 を素因数として持つ要素の和 → $3 + 6 + 9 + 12 = 30$
    - 要素の数は $\frac{12}{3} = 4$ 個 (初項 3 公差 3 の等差数列状)
    - 総和は $3 \times \frac{4(4+1)}{2} = 30$
  - $2, 3$ を素因数として持つ要素の和 → $6 + 12 = 18$
  - 以上を踏まえて $78 - 42 - 30 + 18 = 24$ (確かに一致)

LCM Rush

ここまでをまとめると・・・

1. 元の式を変形して，LCM ではなく GCD で考える
2. K との GCD のパターン数が少ないことを利用して式を分解し，GCD ごとにまとめて処理
3. 要素の和の包除原理も，これまで紹介した包除原理 + 少し数学を頑張ると可能

次に具体的な実装をのせます

実装 (C++)

```cpp
#include <iostream>
#include <vector>
using namespace std;
using ll = long long int;
const ll MOD = 1000000007;
vector<int> getPrimes(int val) { // val の素因数列挙 (略)
}
// 1 \leq i \leq N であって GCD(i, K) = 1 である i の総和
ll solve(ll N, ll K) {
    vector<int> div = getPrimes(K); ll M = div.size(), res = 0;
    for(int bit=0; bit<(1<<M); bit++) {
        ll mul = 1; // K の全ての素因数の掛け合わせを試す
        for(int i=0; i<M; i++) if(bit >> i & 1) mul *= div[i];
        // mul を約数として持つ数の総和を求め、足し引きする
        ll d = N / mul, val = d * (d+1) / 2 % MOD * mul % MOD;
        (res += val * (__builtin_popcount(bit) % 2 ? (-1) : (+1)) + MOD) %= MOD;
    }
    return res;
}
int main() {
    ll N, K, ans = 0; cin >> N >> K;
    for(int i=1; i*i<=K; i++) {
        if(K % i == 0) { // すべての約数について処理
                    (ans += K * solve(N/i    , K/i)) %= MOD;
            if(K/i != i) (ans += K * solve(N*i/K , i   )) %= MOD;
        }
    }
    cout << ans << endl;
    return 0;
}
```

Enumeration

n 個の整数 $a_1, a_2, \ldots, a_n$ と n 個の整数 $p_1, p_2, \ldots, p_n$, 整数 m が与えられる．k 番目の整数 a_k を p_k % の確率で選ぶという操作を各 k について行い，いくつか整数を選び出す．m 以下の自然数の中で，選ばれた整数の少なくとも 1 つで割り切れるものの個数の期待値を求めよ．

- $1 \leq n \leq 20$
- $1 \leq m \leq 10^{18}$
- $1 \leq a_k \leq 10^{18}$
- $1 \leq p_k \leq 99$

出典: AOJ 2446 Link

サンプル

入力 1： n=2, m=15, a={3, 5}, p={50, 50} 出力: 3.75

Enumeration

- n 個の整数の集合を W とおき，これに対し
  - W の部分集合を S とする
  - S の元の少なくとも 1 つで割り切れる自然数の個数を V_S とする
  - 集合が S になる確率を P_S とする
- 求める期待値 E は以下のように書ける

Enumeration

- n 個の整数の集合を W とおき，これに対し
  - W の部分集合を S とする
  - S の元の少なくとも 1 つで割り切れる自然数の個数を V_S とする
  - 集合が S になる確率を P_S とする
- 求める期待値 E は以下のように書ける
  - $E = \sum_{S \subseteq W} P_S * V_S$
- この期待値を求めるには，どんな計算が必要？

Enumeration

- n 個の整数の集合を W とおき，これに対し
  - W の部分集合を S とする
  - S の元の少なくとも 1 つで割り切れる自然数の個数を V_S とする
  - 集合が S になる確率を P_S とする
- 求める期待値 E は以下のように書ける
  - $E = \sum_{S \subseteq W} P_S * V_S$
- この期待値を求めるには，どんな計算が必要？
  - P_S の計算 (これは簡単)
  - V_S の計算 (これはちょっとたいへん)
- V_S は「少なくとも 1 つ」からもわかるように和集合である
  - 包除原理で処理できそう
  - 計算量は大丈夫？

Enumeration

- n 個の整数の集合を W とおき，これに対し
  - W の部分集合を S とする
  - S の元の少なくとも 1 つで割り切れる自然数の個数を V_S とする
  - 集合が S になる確率を P_S とする
- 求める期待値 E は以下のように書ける
  - $E = \sum_{S \subseteq W} P_S * V_S$
- この期待値を求めるには，どんな計算が必要？
  - P_S の計算 (これは簡単)
  - V_S の計算 (これはちょっとたいへん)
- V_S は「少なくとも 1 つ」からもわかるように和集合である
  - 包除原理で処理できそう
  - 計算量は大丈夫？ → 実は大丈夫じゃない・・・
- 計算量を考えてみよう
  - W の部分集合は 2^n 通り
  - 部分集合それぞれについて包除原理を適用しなければならない
  - 全ての部分集合に対して，その部分集合を見ることは全体で $O(3^n)$ かかり，これだと遅い

Enumeration

- 包除原理パートを何とかして高速にやりたい
- ここで登場: 「高速ゼータ変換」や「高速メビウス変換」！
  - 詳細は後で説明しますが，高速メビウス変換は包除原理と相性良いです
- これらはどのような変換・操作なのか？？

高速ゼータ変換・高速メビウス変換

高速ゼータ変換・高速メビウス変換

- 以下の関数を用意
  - $f(S) :=$ 集合 S に対して定義される関数
- 高速ゼータ変換により以下の関数が高速に求められる
  1. $g(S) = \sum_{S \subseteq T} f(T)$
     - S を部分集合として持つ集合 T の関数値 $f(T)$ の総和
  2. $g'(S) = \sum_{T' \subseteq S} f(T')$
     - S の任意の部分集合 T' の関数値 $f(T')$ の総和
- 高速メビウス変換により以下の関数が高速に求められる
  3. $f(S) = \sum_{S \subseteq T} g(T) \times (-1)^{|T \setminus S|}$
     - 1. の逆変換に相当するもの
  4. $f(S) = \sum_{T' \subseteq S} g'(T') \times (-1)^{|S \setminus T'|}$
     - 2. の逆変換に相当するもの
- 真面目にやると $O(3^n)$ である処理が $O(n2^n)$ になる

高速ゼータ変換・高速メビウス変換

どのコードも非常に簡潔に書ける

1. $g(S) = \sum_{S \subseteq T} f(T)$ (高速ゼータ変換)
```
for(int i=0; i<N; i++) for(int bit=0; bit<(1<<N); bit++) {
    if(!(bit >> i & 1)) func[bit] += func[bit ^ (1<<i)];
}
```

2. $g'(S) = \sum_{T' \subseteq S} f(T')$ (高速ゼータ変換)
```
for(int i=0; i<N; i++) for(int bit=0; bit<(1<<N); bit++) {
    if(bit >> i & 1)    func[bit] += func[bit ^ (1<<i)];
}
```

3. $f(S) = \sum_{S \subseteq T} g(T) \times (-1)^{|T \setminus S|}$ (高速メビウス変換)
```
for(int i=0; i<N; i++) for(int bit=0; bit<(1<<N); bit++) {
    if(!(bit >> i & 1)) func[bit] -= func[bit ^ (1<<i)];
}
```

4. $f(S) = \sum_{T' \subseteq S} g'(T') \times (-1)^{|S \setminus T'|}$ (高速メビウス変換)
```
for(int i=0; i<N; i++) for(int bit=0; bit<(1<<N); bit++) {
    if(bit >> i & 1)    func[bit] -= func[bit ^ (1<<i)];
}
```

高速ゼータ変換・高速メビウス変換

- コードが短いが，実際にはどういうことをしている？
- 高速ゼータ変換の 2 つについて，動きを見ていこう
- 高速メビウス変換も実質同じ動きなので，興味があれば実験してね
- 説明のため，全体集合に属する要素数は 3 個とする

高速ゼータ変換・高速メビウス変換

高速ゼータ変換 $g(S) = \sum_{S \subseteq T} f(T)$ についてみていこう

```
for(int i=0; i<N; i++) for(int bit=0; bit<(1<<N); bit++) {
    if(!(bit >> i & 1)) func[bit] += func[bit ^ (1<<i)];
}
```

最初は，自身の f の値のみが入っている

- $g\{000\} : f\{000\}$
- $g\{001\} : f\{001\}$
- $g\{010\} : f\{010\}$
- $g\{011\} : f\{011\}$
- $g\{100\} : f\{100\}$
- $g\{101\} : f\{101\}$
- $g\{110\} : f\{110\}$
- $g\{111\} : f\{111\}$

高速ゼータ変換 $g(S) = \sum_{S \subseteq T} f(T)$ についてみていこう

```
for(int i=0; i<N; i++) for(int bit=0; bit<(1<<N); bit++) {
    if(!(bit >> i & 1)) func[bit] += func[bit ^ (1<<i)];
}
```

i = 0 のループ終了後 (0 ビット目が違うものが足される)

- $g\{000\} : f\{000\} + f\{001\}$
- $g\{001\} : f\{001\}$
- $g\{010\} : f\{010\} + f\{011\}$
- $g\{011\} : f\{011\}$
- $g\{100\} : f\{100\} + f\{101\}$
- $g\{101\} : f\{101\}$
- $g\{110\} : f\{110\} + f\{111\}$
- $g\{111\} : f\{111\}$

高速ゼータ変換・高速メビウス変換

高速ゼータ変換 $g(S) = \sum_{S \subseteq T} f(T)$ についてみていこう

```
for(int i=0; i<N; i++) for(int bit=0; bit<(1<<N); bit++) {
    if(!(bit >> i & 1)) func[bit] += func[bit ^ (1<<i)];
}
```

i = 1 のループ終了後 (1 ビット目が違うものが足される)

- $g\{000\} : f\{000\} + f\{001\} + f\{010\} + f\{011\}$
- $g\{001\} : f\{001\} + f\{011\}$
- $g\{010\} : f\{010\} + f\{011\}$
- $g\{011\} : f\{011\}$
- $g\{100\} : f\{100\} + f\{101\} + f\{110\} + f\{111\}$
- $g\{101\} : f\{101\} + f\{111\}$
- $g\{110\} : f\{110\} + f\{111\}$
- $g\{111\} : f\{111\}$

高速ゼータ変換 $g(S) = \sum_{S \subseteq T} f(T)$ についてみていこう

```
for(int i=0; i<N; i++) for(int bit=0; bit<(1<<N); bit++) {
    if(!(bit >> i & 1)) func[bit] += func[bit ^ (1<<i)];
}
```

i = 2 のループ終了後 (2 ビット目が違うものが足される)

- $g\{000\} : f\{000\} + f\{001\} + f\{010\} + f\{011\} + f\{100\} + f\{101\} + f\{110\} + f\{111\}$
- $g\{001\} : f\{001\} + f\{011\} + f\{101\} + f\{111\}$
- $g\{010\} : f\{010\} + f\{011\} + f\{110\} + f\{111\}$
- $g\{011\} : f\{011\} + f\{111\}$
- $g\{100\} : f\{100\} + f\{101\} + f\{110\} + f\{111\}$
- $g\{101\} : f\{101\} + f\{111\}$
- $g\{110\} : f\{110\} + f\{111\}$
- $g\{111\} : f\{111\}$

高速ゼータ変換・高速メビウス変換

高速ゼータ変換 $g'(S) = \sum_{T' \subseteq S} f(T')$ についてみていこう

```
for(int i=0; i<N; i++) for(int bit=0; bit<(1<<N); bit++) {
    if(bit >> i & 1)    func[bit] += func[bit ^ (1<<i)];
}
```

最初は，自身の f の値のみが入っている

- $g'\{000\} : f\{000\}$
- $g'\{001\} : f\{001\}$
- $g'\{010\} : f\{010\}$
- $g'\{011\} : f\{011\}$
- $g'\{100\} : f\{100\}$
- $g'\{101\} : f\{101\}$
- $g'\{110\} : f\{110\}$
- $g'\{111\} : f\{111\}$

高速ゼータ変換 $g'(S) = \sum_{T' \subseteq S} f(T')$ についてみていこう

```
for(int i=0; i<N; i++) for(int bit=0; bit<(1<<N); bit++) {
    if(bit >> i & 1)    func[bit] += func[bit ^ (1<<i)];
}
```

i = 0 のループ終了後 (0 ビット目が違うものが足される)

- $g'\{000\} : f\{000\}$
- $g'\{001\} : f\{000\} + f\{001\}$
- $g'\{010\} : f\{010\}$
- $g'\{011\} : f\{010\} + f\{011\}$
- $g'\{100\} : f\{100\}$
- $g'\{101\} : f\{100\} + f\{101\}$
- $g'\{110\} : f\{110\}$
- $g'\{111\} : f\{110\} + f\{111\}$

高速ゼータ変換・高速メビウス変換

高速ゼータ変換 $g'(S) = \sum_{T' \subseteq S} f(T')$ についてみていこう

```
for(int i=0; i<N; i++) for(int bit=0; bit<(1<<N); bit++) {
    if(bit >> i & 1)    func[bit] += func[bit ^ (1<<i)];
}
```

i = 1 のループ終了後 (1 ビット目が違うものが足される)

- $g'\{000\} : f\{000\}$
- $g'\{001\} : f\{000\} + f\{001\}$
- $g'\{010\} : f\{000\} + f\{010\}$
- $g'\{011\} : f\{000\} + f\{001\} + f\{010\} + f\{011\}$
- $g'\{100\} : f\{100\}$
- $g'\{101\} : f\{100\} + f\{101\}$
- $g'\{110\} : f\{100\} + f\{110\}$
- $g'\{111\} : f\{100\} + f\{101\} + f\{110\} + f\{111\}$

高速ゼータ変換 $g'(S) = \sum_{T' \subseteq S} f(T')$ についてみていこう

```
for(int i=0; i<N; i++) for(int bit=0; bit<(1<<N); bit++) {
    if(bit >> i & 1)    func[bit] += func[bit ^ (1<<i)];
}
```

i = 2 のループ終了後 (2 ビット目が違うものが足される)

- $g'\{000\} : f\{000\}$
- $g'\{001\} : f\{000\} + f\{001\}$
- $g'\{010\} : f\{000\} + f\{010\}$
- $g'\{011\} : f\{000\} + f\{001\} + f\{010\} + f\{011\}$
- $g'\{100\} : f\{000\} + f\{100\}$
- $g'\{101\} : f\{000\} + f\{001\} + f\{100\} + f\{101\}$
- $g'\{110\} : f\{000\} + f\{010\} + f\{100\} + f\{110\}$
- $g'\{111\} : f\{000\} + f\{001\} + f\{010\} + f\{011\} + f\{100\} + f\{101\} + f\{110\} + f\{111\}$

Enumeration

- 結論として，包除原理パートを高速メビウス変換で高速化することによって解ける
- 数が非常に大きく，普通に LCM するとオーバーフローするので，そこだけ注意が必要
- 次のページで実装を示します

Enumeration

実装 (C++)

```cpp
#include <iostream>
#include <iomanip>
#include <algorithm>
using namespace std;
using ll = long long int;

ll lcm(ll A, ll B) {
    return (ll)min(2e18l+10, A / __gcd(A, B) * 1.0l * B);
}

ll N, M, A[25], dp[1 << 20]; double P[25], ans;
int main() {
    cin >> N >> M;
    for(int i=0; i<N; i++) cin >> A[i];
    for(int i=0; i<N; i++) cin >> P[i], P[i] /= 100.0;
    for(int bit=1; bit<(1<<N); bit++) {
        ll mul = 1;
        for(int i=0; i<N; i++) if(bit >> i & 1) mul = lcm(mul, A[i]);
        dp[bit] = M / mul;
    }
    // 高速メビウス変換
    // (対象とする集合の元のいずれかで割り切れる要素数を高速に計算)
    for(int i=0; i<N; i++) for(int bit=0; bit<(1<<N); bit++) {
        if(bit >> i & 1) dp[bit] -= dp[bit ^ (1 << i)];
    }
    for(int bit=0; bit<(1<<N); bit++) {
        double prob = 1.0;
        for(int i=0; i<N; i++) prob *= (bit >> i & 1 ? P[i] : 1 - P[i]);
        ans += prob * abs(dp[bit]);
    }
    cout << fixed << setprecision(12) << ans << endl;
    return 0;
}
```

天下一ボディービルコンテスト

天下一ボディービルコンテスト

長さ N の整数列 $A = \{a_1, a_2, \ldots, a_N\}$ と，全要素が 0 の長さ N の数列 B がある．数列 B に対して以下の操作を D 回行う．

- B 中の要素をいずれかひとつ選び，数値を 1 増加させる．

最終的にできる数列 B の任意の要素 b_i について，$b_i \geq a_i$ を満たすようにしたい．このような操作方法が何通りあるか求めよ．

- $1 \leq N \leq 30$
- $1 \leq D \leq 10^9$
- $1 \leq a_i \leq 30$

出典: 天下一プログラマーコンテスト 2013 決勝 D ▸ Link

サンプル

入力 1： N = 2, D = 3, A = {1, 1}　出力: 6

- $\{1,1,2\}$・$\{1,2,1\}$・$\{1,2,2\}$・$\{2,1,1\}$・$\{2,1,2\}$・$\{2,2,1\}$ の 6 通り

天下一ボディービルコンテスト

- 「任意の要素 b_i について $b_i \geq a_i$」の余事象を考えると？

天下一ボディービルコンテスト

- 「任意の要素 b_i について $b_i \geq a_i$」の余事象を考えると？
  - 「少なくとも 1 つの要素 b_i について $b_i < a_i$」
- $b_i < a_i$ となる要素の個数を決め打ち (j 個)
  - 残り $N - j$ 個に関してはどうでも良くなる

天下一ボディービルコンテスト

- 「任意の要素 b_i について $b_i \geq a_i$」の余事象を考えると？
  - 「少なくとも 1 つの要素 b_i について $b_i < a_i$」
- $b_i < a_i$ となる要素の個数を決め打ち (j 個)
  - 残り $N - j$ 個に関してはどうでも良くなる
- 対称性は成り立たなさそうだし，$N \leq 30$ なので愚直にやると TLE
- 動的計画法を用いて状態をまとめることを考えよう！！
  - 見るべき状態のパラメータを単純にし，まとめられるものはまとめる

天下一ボディービルコンテスト

- 包除原理でどんな情報が必要になるか考えて，DP を組み立てよう

天下一ボディービルコンテスト

- 包除原理でどんな情報が必要になるか考えて，DP を組み立てよう
  - 足し引き時は集合の要素数が重要になるので，それは覚えておきたい
  - 包除で考慮する要素に対して何回操作するかも知りたい

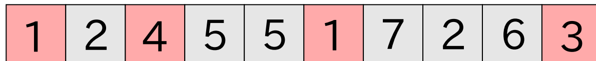
天下一ボディービルコンテスト

- 包除原理でどんな情報が必要になるか考えて，DP を組み立てよう
  - 足し引き時は集合の要素数が重要になるので，それは覚えておきたい
  - 包除で考慮する要素に対して何回操作するかも知りたい
- $\mathrm{dp}[i][j][k] :=$ i 番目までの要素のうち j 種類を包除内で考慮し，それらに対して合計 k 回操作する場合の数 とする
- この DP をすることによって，以下のように包除パートを処理可能

天下一ボディービルコンテスト

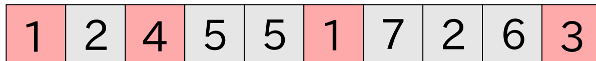
- 包除原理でどんな情報が必要になるか考えて，DP を組み立てよう
  - 足し引き時は集合の要素数が重要になるので，それは覚えておきたい
  - 包除で考慮する要素に対して何回操作するかも知りたい
- $\mathrm{dp}[i][j][k] :=$ i 番目までの要素のうち j 種類を包除内で考慮し，それらに対して合計 k 回操作する場合の数 とする
- この DP をすることによって，以下のように包除パートを処理可能
  - 包除で使用 j 種類・それ以外 $N-j$ 種類と分けられる
  - 操作は包除で使用する要素で k 回・それ以外で $D-k$ 回

数字決め打ち → 位置シャッフル のようなイメージ
※位置シャッフルとは、赤ブロックの間に灰ブロックを順番を守りながら挿入すること



天下一ボディービルコンテスト

- 包除原理でどんな情報が必要になるか考えて，DP を組み立てよう
  - 足し引き時は集合の要素数が重要になるので，それは覚えておきたい
  - 包除で考慮する要素に対して何回操作するかも知りたい
- $\mathrm{dp}[i][j][k] := i$ 番目までの要素のうち j 種類を包除内で考慮し，それらに対して合計 k 回操作する場合の数 とする
- この DP をすることによって，以下のように包除パートを処理可能
  - 包除で使用 j 種類・それ以外 $N - j$ 種類と分けられる
  - 操作は包除で使用する要素で k 回・それ以外で $D - k$ 回
  - 包除で使用する領域 (赤色) の組み合わせは DP で求められている
  - それ以外の領域 (灰色) では，まずどの要素に操作するかを全て決め，その後に赤領域の間に挿入するイメージで操作列を全通り列挙



天下一ボディービルコンテスト

$\mathrm{dp}[i][j][k]$ からの状態遷移はどうなるか？

- 包除原理内で i 番目の要素を考慮しない場合
  - 考慮する種類数も，包除で考慮する要素に対する操作回数も不変
  - $\mathrm{dp}[i+1][j][k]$ に対してそのまま足せば良い

天下一ボディービルコンテスト

$\mathrm{dp}[i][j][k]$ からの状態遷移はどうなるか？

- 包除原理内で i 番目の要素を考慮しない場合
  - 考慮する種類数も，包除で考慮する要素に対する操作回数も不変
  - $\mathrm{dp}[i+1][j][k]$ に対してそのまま足せば良い
- 包除原理内で i 番目の要素を考慮する場合
  - 考慮する種類数が増える
  - 包除で考慮する要素に対する操作回数は増えるか変わらない
  - i 番目の要素に対して x 回 $(0 \leq x < a_i)$ 操作したとする
    - 元々の条件を満たさないものを作りたいので操作は a_i 回未満
    - 操作回数の合計は $k+x$ 回になる
    - 既存の操作列のどこかに x 回の操作を挿入 $({}_{k+x}C_x$ 通り$)$
  - $\mathrm{dp}[i+1][j+1][k+x]$ に対して，${}_{k+x}C_x \mathrm{dp}[i][j][k]$ 通り足せば良い

天下一ボディービルコンテスト

この DP をすることによって，以下のように包除パートを処理可能 (再掲)

- 包除で使用 j 種類・それ以外 $N-j$ 種類と分けられる
- 操作は包除で使用する要素で k 回・それ以外で $D-k$ 回
- 包除で使用する領域 (赤色) の組み合わせは DP で求められている
- それ以外の領域 (灰色) では，まずどの要素に操作するかを全て決め，その後に赤領域の間に挿入するイメージで操作列を全通り列挙

包除で j 種類考慮、操作 k 回　　それ以外 N - j 種類、操作 D- k 回

1	4	1	3	2	5	5	7	2	6

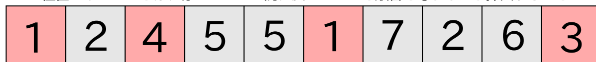
数字決め打ち → 位置シャッフル のようなイメージ
※位置シャッフルとは、赤ブロックの間に灰ブロックを順番を守りながら挿入すること

1	2	4	5	5	1	7	2	6	3

天下一ボディービルコンテスト

つまり，DP の結果を用いて答えを得るには？

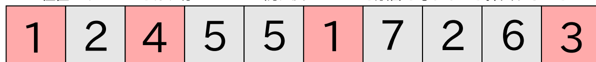
- 全ての j, k について以下を行う
  - $\mathrm{dp}[N][j][k]$ (包除 j 種類・操作 k 回) の組み合わせを持ってくる



天下一ボディービルコンテスト

つまり，DP の結果を用いて答えを得るには？

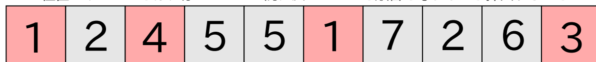
- 全ての j, k について以下を行う
  - $\mathrm{dp}[N][j][k]$ (包除 j 種類・操作 k 回) の組み合わせを持ってくる
  - 残った $D-k$ 回の各操作について，$N-j$ 種類のうちどれに割り当てるか決定
    - $N-j$ 通り取れるものが $D-k$ 個あるので，$(N-j)^{D-k}$



天下一ボディービルコンテスト

つまり，DP の結果を用いて答えを得るには？

- 全ての j, k について以下を行う
  - $\mathrm{dp}[N][j][k]$ (包除 j 種類・操作 k 回) の組み合わせを持ってくる
  - 残った $D-k$ 回の各操作について，$N-j$ 種類のうちどれに割り当てるか決定
    - $N-j$ 通り取れるものが $D-k$ 個あるので，$(N-j)^{D-k}$
  - $D-k$ 回の操作を k 回の操作列中に挿入する
    - 組み合わせは ${}_DC_k$ 通りある



天下一ボディービルコンテスト

つまり，DP の結果を用いて答えを得るには？

- 全ての j, k について以下を行う
  - $\mathrm{dp}[N][j][k]$ (包除 j 種類・操作 k 回) の組み合わせを持ってくる
  - 残った $D-k$ 回の各操作について，$N-j$ 種類のうちどれに割り当てるか決定
    - $N-j$ 通り取れるものが $D-k$ 個あるので，$(N-j)^{D-k}$
  - $D-k$ 回の操作を k 回の操作列中に挿入する
    - 組み合わせは ${}_DC_k$ 通りある
  - これで得られた値を，j の偶奇に応じて答えに足し引きする

包除で j 種類考慮、操作 k 回 ／ それ以外 N - j 種類、操作 D- k 回

1	4	1	3	2	5	5	7	2	6

数字決め打ち → 位置シャッフル のようなイメージ
※位置シャッフルとは、赤ブロックの間に灰ブロックを順番を守りながら挿入すること

1	2	4	5	5	1	7	2	6	3

実装 (C++)

```
// mod_pow, combination 省略
long long int dp[31][31][31*31], ans;
int main() {
    cin >> N >> D; init_comb(); // N, D はグローバルに置く
    // dp[何番目の数字まで見た][考慮に入れる要素 j 個][操作回数の合計 k 回] := 場合の数
    dp[0][0][0] = 1;
    for(int i=0; i<N; i++) {
        int val; cin >> val, sum += val;
        for(int j=0; j<=i; j++) {
            for(int k=0; k<sum; k++) {
                // 考慮しない
                (dp[i+1][j][k] += dp[i][j][k]) %= MOD;
                // 考慮する (take 個追加・0 個追加もわすれずに)
                for(int take=0; take<val; take++) {
                    if(k + take > D) continue; // あくまで D 回以内のもののみ考慮
                    (dp[i+1][j+1][k+take] += dp[i][j][k] * comb[k+take][take]) %= MOD;
                }
            }
        }
    }
    for(int j=0; j<=N; j++) for(int k=0; k<=min(D-1, sum); k++) {
        long long int val = dp[N][j][k];
        // 残っている操作回数は D - k 回、これの分を配置
        (val *= calcComb(k)) %= MOD; // D choose k (D は不変)
        // それぞれについて N - j 種類のうちのどれかにする
        (val *= mod_pow(N - j, D - k)) %= MOD;
        ans = (ans + val * (j % 2 ? -1 : 1) + MOD) % MOD;
    }
    cout << (D < sum ? 0 : ans) << endl;
    return 0;
}
```

包除原理の問題・上級編

- ここからは包除原理上級編です
- AtCoder で言うところの 1000 点前後の問題を扱います
- 考察が重い + 他のテクニックとの組み合わせが多いです
  - 動的計画法によって状態をまとめる
  - 約数系包除 などなど
- 説明が多くてスライドの枚数も多いですが，がんばりましょう

出席番号 (2)

長さ N の順列を作りたい．「i 番目はある値 A_i にできない」という制約が，全ての i について 1 つずつある．これを全て満たすような順列は何通りあるか？

- $1 \leq N \leq 5000$
- $0 \leq A_i \leq 4999$

出典: yukicoder No.243 Link

サンプル

```
入力 1 :  N = 3, A = {0, 1, 2}      出力:  2
入力 2 :  N = 4, A = {3, 3, 3, 3}   出力:  0
```

出席番号 (2)

- まず条件を言い換えよう
  - 余事象「ある i について，数列の値が A_i と等しい」

出席番号 (2)

- まず条件を言い換えよう
  - 余事象「ある i について，数列の値が A_i と等しい」
- 集合を全て試すのは制約上不可能なので，状態をまとめたい
  - A_i と必ず等しくなる要素が k 個あるような状態数を求めればよさそう
  - これが分かれば，包除原理で答えを求められる
  - 対称性は成り立ちそうにない

出席番号 (2)

- まず条件を言い換えよう
  - 余事象「ある i について，数列の値が A_i と等しい」
- 集合を全て試すのは制約上不可能なので，状態をまとめたい
  - A_i と必ず等しくなる要素が k 個あるような状態数を求めればよさそう
  - これが分かれば，包除原理で答えを求められる
  - 対称性は成り立ちそうにない
- どうやったらこの状態数を求められるか？
  - 動的計画法で解決できる！
  - 次ページから詳細を説明します

出席番号 (2)

- 例えば，ある 1 つの数 x について条件を必ず違反させたいとき

1	0	5	0	2	4	1	3	3	3

出席番号 (2)

- 例えば，ある 1 つの数 x について条件を必ず違反させたいとき
- x が禁止されている要素数を数える (C_x 個あるとする)
- そのような要素のうち，いずれか 1 つに x を配置させる

例えば 1 について、必ず条件に違反するように配置する場合・・・
・ 1 が禁止されている要素の数を求め (下の例では 2 通り)、その内のいずれかに 1 を配置
・ あとの並びはどうでも良いので (N - 1)! を掛け合わせる

1	0	5	0	2	4	1	3	3	3

出席番号 (2)

- 例えば，ある 1 つの数 x について条件を必ず違反させたいとき
- x が禁止されている要素数を数える (C_x 個あるとする)
- そのような要素のうち，いずれか 1 つに x を配置させる
- それ以外の要素に関しては何が来ても良い
  - x だけもう配置先が決まっており，それ以外は未定
  - $(N-1)!$ 通りある

例えば 1 について、必ず条件に違反するように配置する場合・・・
・ 1 が禁止されている要素の数を求め (下の例では 2 通り)、その内のいずれかに 1 を配置
・ あとの並びはどうでも良いので (N - 1)! を掛け合わせる

1	0	5	0	2	4	1	3	3	3

出席番号 (2)

- 例えば，ある 1 つの数 x について条件を必ず違反させたいとき
- x が禁止されている要素数を数える (C_x 個あるとする)
- そのような要素のうち，いずれか 1 つに x を配置させる
- それ以外の要素に関しては何が来ても良い
  - x だけもう配置先が決まっており，それ以外は未定
  - $(N-1)!$ 通りある
- よって，x について必ず違反させるときの場合の数は $C_x(N-1)!$ 通り

例えば 1 について、必ず条件に違反するように配置する場合・・・
・ 1 が禁止されている要素の数を求め (下の例では 2 通り)、その内のいずれかに 1 を配置
・ あとの並びはどうでも良いので (N - 1)! を掛け合わせる

1	0	5	0	2	4	1	3	3	3

出席番号 (2)

- これは，必ず違反させたい数が複数個あっても同様にできる！
- なぜか？

1	0	5	0	2	4	1	3	3	3

出席番号 (2)

- これは，必ず違反させたい数が複数個あっても同様にできる！
- なぜか？
  - 任意の異なる 2 つの数 x, y について，x が禁止されている要素のインデックスと y が禁止されている要素のインデックスが被らない (つまり独立) ため

1 と 3 について、必ず条件に違反するように配置する場合・・・
・1 が禁止されている要素の数を求め (下の例では 2 通り)、その内のいずれかに 1 を配置
・3 が禁止されている要素の数を求め (下の例では 3 通り)、その内のいずれかに 3 を配置
・あとの並びはどうでも良いので (N - 2)! を掛け合わせる

1	0	5	0	2	4	1	3	3	3

出席番号 (2)

- これは，必ず違反させたい数が複数個あっても同様にできる！
- なぜか？
  - 任意の異なる 2 つの数 x, y について，x が禁止されている要素のインデックスと y が禁止されている要素のインデックスが被らない (つまり独立) ため
- したがって，違反させたい数が $x_1, x_2, \ldots, x_m$ である場合，これらを必ず違反させるときの場合の数は $\prod_{i=1}^{m} C_{x_i}(N-m)!$ 通り

1 と 3 について、必ず条件に違反するように配置する場合・・・
・1 が禁止されている要素の数を求め (下の例では 2 通り)、その内のいずれかに 1 を配置
・3 が禁止されている要素の数を求め (下の例では 3 通り)、その内のいずれかに 3 を配置
・あとの並びはどうでも良いので (N - 2)! を掛け合わせる

1	0	5	0	2	4	1	3	3	3

出席番号 (2)

- これらを踏まえて，どのような DP を組み立てれば良いか？

出席番号 (2)

- これらを踏まえて，どのような DP を組み立てれば良いか？
- $\mathrm{dp}[k]$:= 必ず違反させたい数が k 個のときの $\prod_{i=1}^{k} C_{x_i}$ の合計 とする
- これは $O(N^2)$ で実現可能
  - $\mathrm{dp}[i$ 番目まで見た][必ず違反する数が k 種類] の二次元でやるとする
  - $\mathrm{dp}[i][k]$ からの遷移を考えてみよう

出席番号 (2)

- これらを踏まえて，どのような DP を組み立てれば良いか？
- $\mathrm{dp}[k]$:= 必ず違反させたい数が k 個のときの $\prod_{i=1}^{k} C_{x_i}$ の合計 とする
- これは $O(N^2)$ で実現可能
  - $\mathrm{dp}[i$ 番目まで見た][必ず違反する数が k 種類] の二次元でやるとする
  - $\mathrm{dp}[i][k]$ からの遷移を考えてみよう
  - $i+1$ 番目の要素を必ず違反させたいとき
    - 今までの状態全てに対し $C_{x_{i+1}}$ を掛け合わせるイメージなので・・・
    - $\mathrm{dp}[i+1][k+1]$ に $\mathrm{dp}[i][k] \times C_{x_{i+1}}$ を足しこむ

出席番号 (2)

- これらを踏まえて，どのような DP を組み立てれば良いか？
- $\mathrm{dp}[k] :=$ 必ず違反させたい数が k 個のときの $\prod_{i=1}^{k} C_{x_i}$ の合計 とする
- これは $O(N^2)$ で実現可能
  - $\mathrm{dp}[i$ 番目まで見た$][$必ず違反する数が k 種類$]$ の二次元でやるとする
  - $\mathrm{dp}[i][k]$ からの遷移を考えてみよう
  - $i+1$ 番目の要素を必ず違反させたいとき
    - 今までの状態全てに対し $C_{x_{i+1}}$ を掛け合わせるイメージなので・・・
    - $\mathrm{dp}[i+1][k+1]$ に $\mathrm{dp}[i][k] \times C_{x_{i+1}}$ を足しこむ
  - そうでないとき
    - $\mathrm{dp}[i+1][k]$ に $\mathrm{dp}[i][k]$ を足しこむ

出席番号 (2)

- これらを踏まえて，どのような DP を組み立てれば良いか？
- $\mathrm{dp}[k]$:= 必ず違反させたい数が k 個のときの $\prod_{i=1}^{k} C_{x_i}$ の合計 とする
- これは $O(N^2)$ で実現可能
  - $\mathrm{dp}[i$ 番目まで見た$][$必ず違反する数が k 種類$]$ の二次元でやるとする
  - $\mathrm{dp}[i][k]$ からの遷移を考えてみよう
  - $i+1$ 番目の要素を必ず違反させたいとき
    - 今までの状態全てに対し $C_{x_{i+1}}$ を掛け合わせるイメージなので・・・
    - $\mathrm{dp}[i+1][k+1]$ に $\mathrm{dp}[i][k] \times C_{x_{i+1}}$ を足しこむ
  - そうでないとき
    - $\mathrm{dp}[i+1][k]$ に $\mathrm{dp}[i][k]$ を足しこむ
  - (本当に二次元配列でやるとメモリが足りないので注意)
    - 一次元配列とかで実装しましょう

出席番号 (2)

実装 (C++)

```cpp
#include <iostream>
using namespace std;
using ll = long long int;
ll dp[5010], cnt[5010], MOD = 1000000007;
int main() {
    int N; cin >> N; dp[0] = 1;
    for(int i=0; i<N; i++) {
        int val; cin >> val;
        cnt[val]++;
    }
    // dp[k] := 制約に必ず違反する人数が k 人である場合の数
    for(int i=0; i<N; i++) for(int k=i; k>=0; k--) {
        (dp[k+1] += dp[k] * cnt[i]) %= MOD;
    }
    ll fact = 1, ans = 0, cnt = 0;
    for(int i=N; i>=0; i--) {
        // 制約に必ず違反する人数が i 人、その他 N - i 人
        // N - i 人に関してはどう並んでも良い
        ll val = dp[i] * fact % MOD;
        // 包除原理 (人数の偶奇で値を足し引き)
        if(i % 2 == 0) ans = (ans + val      ) % MOD;
        else           ans = (ans - val + MOD) % MOD;
        (fact *= (++cnt)) %= MOD;
    }
    cout << ans << endl;
    return 0;
}
```

Rotated Palindromes

Rotated Palindromes

長さが N で，各要素が 1 以上 K 以下で，数列全体が回文であるような数列 A に対して，次の操作を好きなだけ繰り返す.

- A の先頭要素を末尾へ移動 (つまり回転)

最終的な数列として考えられるものはいくつあるか？

- $1 \leq N, K \leq 10^9$

出典: AtCoder Regular Contest 064 F Link

サンプル

入力 1 :　N = 4, K = 2　出力:　6
入力 2 :　N = 1, K = 10　出力:　10

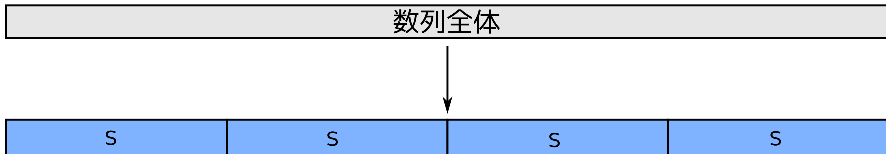
- 入力 1 について，次の 6 通り
  - $\{1,1,1,1\}$・$\{1,1,2,2\}$・$\{1,2,2,1\}$・$\{2,2,1,1\}$・$\{2,1,1,2\}$・$\{2,2,2,2\}$

Rotated Palindromes

- まず重要なアプローチとして，「約数を考慮すること」がある
  - 回転操作による影響を考慮すると，筋が良い考察

Rotated Palindromes

- まず重要なアプローチとして，「約数を考慮すること」がある
  - 回転操作による影響を考慮すると，筋が良い考察
- 回文をなす長さ N の数列全体が，ある長さ d の数列 S の繰り返しによって構成されるとする
  - d の候補は N の約数のみ (全体の長さがちょうど N なので)
  - 周期 d なので，回転操作を d 回すると元に戻る
  - S は，d 未満の周期を持たないもののみ考慮する
    - つまり、S の最小周期は d である
  - 数列の先頭 d 個と末尾 d 個は同一なので，S も回文



Rotated Palindromes

- 周期を d に固定したとする
- まず，長さが d であって回文になるような数列は何通りあるか？

Rotated Palindromes

- 周期を d に固定したとする
- まず，長さが d であって回文になるような数列は何通りあるか？
  - 回文なので，数を決定すべき場所は $\lceil \frac{d}{2} \rceil$ 箇所
  - 数は 1 以上 K 以下から選択可能であるため，$K^{\lceil \frac{d}{2} \rceil}$ 通り存在

Rotated Palindromes

- 周期を d に固定したとする
- まず，長さが d であって回文になるような数列は何通りあるか？
  - 回文なので，数を決定すべき場所は $\lceil \frac{d}{2} \rceil$ 箇所
  - 数は 1 以上 K 以下から選択可能であるため，$K^{\lceil \frac{d}{2} \rceil}$ 通り存在
- 長さ d であって， d 未満の周期を持たない数列は $K^{\lceil \frac{d}{2} \rceil}$ 通りか？

Rotated Palindromes

- 周期を d に固定したとする
- まず，長さが d であって回文になるような数列は何通りあるか？
  - 回文なので，数を決定すべき場所は $\lceil \frac{d}{2} \rceil$ 箇所
  - 数は 1 以上 K 以下から選択可能であるため，$K^{\lceil \frac{d}{2} \rceil}$ 通り存在
- 長さ d であって， d 未満の周期を持たない数列は $K^{\lceil \frac{d}{2} \rceil}$ 通りか？
  - これは嘘
  - このままだと d 未満の周期を持つ数列もカウントしてしまっている
    - 例えば $d = 4, K = 3$ である場合，$K^{\lceil \frac{d}{2} \rceil}$ 通りの数列の中には $\{1, 1, 1, 1\}$・$\{2, 2, 2, 2\}$・$\{3, 3, 3, 3\}$ といった，周期が 1 であるものも含まれている

Rotated Palindromes

- 周期を d に固定したとする
- まず，長さが d であって回文になるような数列は何通りあるか？
  - 回文なので，数を決定すべき場所は $\lceil \frac{d}{2} \rceil$ 箇所
  - 数は 1 以上 K 以下から選択可能であるため，$K^{\lceil \frac{d}{2} \rceil}$ 通り存在
- 長さ d であって，d 未満の周期を持たない数列は $K^{\lceil \frac{d}{2} \rceil}$ 通りか？
  - これは嘘
  - このままだと d 未満の周期を持つ数列もカウントしてしまっている
    - 例えば $d = 4, K = 3$ である場合，$K^{\lceil \frac{d}{2} \rceil}$ 通りの数列の中には $\{1, 1, 1, 1\}$・$\{2, 2, 2, 2\}$・$\{3, 3, 3, 3\}$ といった，周期が 1 であるものも含まれている
- したがって，$K^{\lceil \frac{d}{2} \rceil}$ から，d の約数長であるような回文をなす数列の総数を引く必要がある
- $\mathrm{num}[d] :=$ 最小周期が d である回文数列の総数 とする
  - $\mathrm{num}[d] = K^{\lceil \frac{d}{2} \rceil} - \sum_{d' | d, d' \neq d} \mathrm{num}[d']$

Rotated Palindromes

- N の各約数 d について，最小周期が d であるような回文数列の総数がわかった
- あとは，回転操作によってどれだけ状態数が増えるか考えよう

Rotated Palindromes

- N の各約数 d について，最小周期が d であるような回文数列の総数がわかった
- あとは，回転操作によってどれだけ状態数が増えるか考えよう
- 回転することによって，回文数列 A が別の回文数列 B と一致することはあるか？

Rotated Palindromes

- N の各約数 d について，最小周期が d であるような回文数列の総数がわかった
- あとは，回転操作によってどれだけ状態数が増えるか考えよう
- 回転することによって，回文数列 A が別の回文数列 B と一致することはあるか？
  - これは起こりうる
  - 例えば $A = \{1, 3, 3, 1\}$ で $B = \{3, 1, 1, 3\}$ であるとき，A に対して回転操作を 2 回行うと B と一致

Rotated Palindromes

- N の各約数 d について，最小周期が d であるような回文数列の総数がわかった
- あとは，回転操作によってどれだけ状態数が増えるか考えよう
- 回転することによって，回文数列 A が別の回文数列 B と一致することはあるか？
  - これは起こりうる
  - 例えば $A = \{1, 3, 3, 1\}$ で $B = \{3, 1, 1, 3\}$ であるとき，A に対して回転操作を 2 回行うと B と一致
- 回文数列に対して回転操作を行ったときに，別の回文数列と一致する条件を考えてみよう

結論から書くと，以下のようになります

最小周期が d の回文数列に対する回転操作

- d が奇数のとき，どのように回転しても他の回文数列と一致しない
- d が偶数のとき，回転によって一致する他の回文数列がただ 1 つ存在

事実だけ述べても不親切なので，証明してみましょう

Rotated Palindromes

- 回文数列 A に対して回転を施すことで B と一致する状況を考える
  - まず，明らかに A と B の最小周期は等しい (d とおく)
  - A は回転操作を d 回すると元に戻ることから，A に対して回転操作を $1, 2, \ldots, d-1$ 回行うことで得られる数列のいずれか 1 つのみが B と一致

Rotated Palindromes

- 回文数列 A に対して回転を施すことで B と一致する状況を考える
  - まず，明らかに A と B の最小周期は等しい (d とおく)
  - A は回転操作を d 回すると元に戻ることから，A に対して回転操作を $1, 2, \ldots, d-1$ 回行うことで得られる数列のいずれか 1 つのみが B と一致
- A に回転操作を t 回施して B と一致したとする
- 実は，A に逆回転操作を t 回施しても B と一致する！

Rotated Palindromes

- 回文数列 A に対して回転を施すことで B と一致する状況を考える
  - まず，明らかに A と B の最小周期は等しい (d とおく)
  - A は回転操作を d 回すると元に戻ることから，A に対して回転操作を $1, 2, \ldots, d-1$ 回行うことで得られる数列のいずれか 1 つのみが B と一致
- A に回転操作を t 回施して B と一致したとする
- 実は，A に逆回転操作を t 回施しても B と一致する！

$$
\begin{aligned}
(A\ \text{に}\ t\ \text{回逆操作した数列}) &= (\bar{A}\text{に}\ t\ \text{回逆操作した数列}) \\
&= \overline{(A\ \text{に}\ t\ \text{回操作した数列})} \\
&= \bar{B} \\
&= B
\end{aligned}
$$

Rotated Palindromes

- 回文数列 A に対して回転を施すことで B と一致する状況を考える
  - まず，明らかに A と B の最小周期は等しい (d とおく)
  - A は回転操作を d 回すると元に戻ることから，A に対して回転操作を $1, 2, \ldots, d-1$ 回行うことで得られる数列のいずれか 1 つのみが B と一致
- A に回転操作を t 回施して B と一致したとする
- 実は，A に逆回転操作を t 回施しても B と一致する！

$$
\begin{aligned}
(A \text{ に } t \text{ 回逆操作した数列}) &= (\bar{A}\text{に } t \text{ 回逆操作した数列}) \\
&= \overline{(A \text{ に } t \text{ 回操作した数列})} \\
&= \bar{B} \\
&= B
\end{aligned}
$$

- つまり，A に回転操作を $d-t$ 回施しても B と一致することがわかる

Rotated Palindromes

- 前ページより，$(A$ に t 回操作した数列$) = (A$ に $d-t$ 回操作した数列$)$

Rotated Palindromes

- 前ページより，(A に t 回操作した数列) $=$ (A に $d-t$ 回操作した数列)
- A に対して回転操作を $1, 2, \ldots, d-1$ 回行うことで得られる数列のいずれか 1 つのみが B と一致することから，$t = d - t$
- 変形して，$t = \frac{d}{2}$

Rotated Palindromes

- 前ページより，$(A$ に t 回操作した数列$) = (A$ に $d-t$ 回操作した数列$)$
- A に対して回転操作を $1, 2, \ldots, d-1$ 回行うことで得られる数列のいずれか 1 つのみが B と一致することから，$t = d - t$
- 変形して，$t = \frac{d}{2}$
- d も t も整数であるため，d が偶数の場合は回転することで一致する他の文字列がただひとつ存在し，奇数の場合は存在しない
- 以上を総合すると答えが出る！
  - 計算量は，N の約数の数を $d(N)$ と置くと，$O(d(N)^2)$
  - $N \leq 10^9$ の上では，$d(N)$ は最大 $1,344$ 個なので問題ない

実装 (C++)

- 約数による重複を除くための包除は「約数系包除」と呼ばれています

```
// mod_pow 略
int main() {
    ll N, K, ans = 0, inv2 = mod_pow(2, MOD-2); cin >> N >> K;
    vector<ll> div, num;
    for(int k=1; k*k<=N; k++) {
        if(N % k == 0) {
            div.push_back(k);
            if(N / k != k) div.push_back(N / k);
        }
    }
    sort(div.begin(), div.end()); // 約数を昇順に列挙
    int M = div.size(); num.resize(M);
    for(int i=0; i<M; i++) {
        num[i] = mod_pow(K, (div[i] + 1) / 2);
        // num[i] の約数における場合の数を引き、
        // num[i] を最小周期として持つ数列だけ数える (約数系包除)
        for(int j=0; j<i; j++) if(div[i] % div[j] == 0) {
            num[i] = (num[i] - num[j] + MOD) % MOD;
        }
        // 周期が 2 で割り切れるならば他 1 つと重複するため、2 で割る
        if(div[i] % 2 == 1) (ans += div[i] * num[i]) %= MOD;
        else (ans += div[i] * num[i] % MOD * inv2) %= MOD;
    }
    cout << ans << endl;
    return 0;
}
```

Everything on It

N 種類のトッピングができるラーメンがあり，各トッピングは乗せるか否か選べる．つまり 2^N 通りのラーメンを注文できる．このとき，以下の条件を満たすようなラーメンの組み合わせの数を M で割った余りを求めよ．ただし注文の順番は考慮しないものとする．

1. トッピングの組み合わせが全く同じラーメンを複数杯注文しない
2. 各トッピングは，注文したラーメンのうち 2 杯以上に乗っている

- $2 \leq N \leq 3000$
- $10^8 \leq M \leq 10^9 + 9$，$M$ は素数

出典: AtCoder Regular Contest 096 E ▸ Link

サンプル

入力 1 : n=2，m=1000000007 出力: 2

Everything on It

- 「各トッピングが 2 杯以上に乗っている」という条件は扱いづらい
  - 余事象「ある i 個のトッピングについて 1 杯以下に乗っている」
  - 選んだ i 個以外については何も制限がないことに注意！！

Everything on It

- 「各トッピングが 2 杯以上に乗っている」という条件は扱いづらい
  - 余事象「ある i 個のトッピングについて 1 杯以下に乗っている」
  - 選んだ i 個以外については何も制限がないことに注意！！
- トッピングを選ぶ個数 i を全部試そう
  - トッピングが N 種類あるので，0 から N まで試せば良い
  - トッピングの種類による組み合わせ数の違いはない → 対称性

Everything on It

- 「各トッピングが 2 杯以上に乗っている」という条件は扱いづらい
  - 余事象「ある i 個のトッピングについて 1 杯以下に乗っている」
  - 選んだ i 個以外については何も制限がないことに注意！！
- トッピングを選ぶ個数 i を全部試そう
  - トッピングが N 種類あるので，0 から N まで試せば良い
  - トッピングの種類による組み合わせ数の違いはない → 対称性
- 以上を踏まえると，以下のような式を計算すればよいことがわかる

問題の答えとなる式

$$\text{Answer} = \sum_{i=0}^{N} {}_NC_i (-1)^i \text{ways}(i)$$

- $\sum$ → i 種類のトッピングを 1 杯以下に乗せる (この i を全て試す)
- ${}_NC_i$ → 1 杯以下に乗せるのはどのトッピング？
- $\text{ways}(i)$ → i 種類のトッピングを必ず 1 杯以下に乗せる場合の数

Everything on It

- $\mathrm{ways}(i)$ はどう求めるの？ (ここが一番のポイント)

Everything on It

- $\mathrm{ways}(i)$ はどう求めるの？　(ここが一番のポイント)
- 注文したラーメン全体での各トッピングの注文回数を表にすると以下のようになる

i 種類: 必ず "0" か "1"　　N-i 種類: 制限なし

1	0	0	1	0	2	1	3	3	3

Everything on It

- ways(i) はどう求めるの？ (ここが一番のポイント)
- 注文したラーメン全体での各トッピングの注文回数を表にすると以下のようになる

i 種類: 必ず "0" か "1"　　N-i 種類: 制限なし

1	0	0	1	0	2	1	3	3	3

- 一旦 $N - i$ 種類のことは忘れて，i 種類について考える
- i 種類のトッピングを 1 杯以下に乗せる方法を数えたい

Everything on It

- ways(i) はどう求めるの？ (ここが一番のポイント)
- 注文したラーメン全体での各トッピングの注文回数を表にすると以下のようになる

i 種類: 必ず "0" か "1"　　N-i 種類: 制限なし

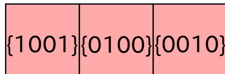
1	0	0	1	0	2	1	3	3	3

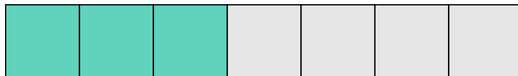
- 一旦 $N-i$ 種類のことは忘れて，i 種類について考える
- i 種類のトッピングを 1 杯以下に乗せる方法を数えたい
- 「同じトッピングのラーメンは複数注文できない」という制約に注意
  - 適当にやると，何もトッピングされていないものが複数登場する
  - どうやったら重複なく考慮できるのか？

Everything on It

- i 種類のトッピングを乗せるラーメンの杯数を j と決め打ちする
- すると，残り $N - i$ 種類に関して

1 杯以下にしか乗せられない
i 種類のトッピングを
j 杯に配分 (空ラーメンはダメ)





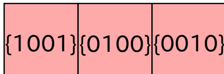
N-i 種類を自由に置く

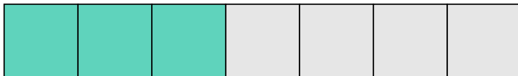
N-i 種類を、同じ組み合わせが
出ないように自由に置く

Everything on It

- i 種類のトッピングを乗せるラーメンの杯数を j と決め打ちする
- すると，残り $N-i$ 種類に関して
  - 最初 j 杯には，$N-i$ 種類を自由配置 (緑色領域)
    - i 種類のトッピングの配置の時点で互いに異なる組み合わせになるため，普通に 2^{N-i} 通りの置き方が許される

1 杯以下にしか乗せられない
i 種類のトッピングを
j 杯に配分 (空ラーメンはダメ)



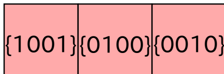


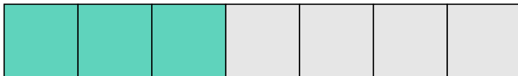
N-i 種類を自由に置く
N-i 種類を、同じ組み合わせが出ないように自由に置く

Everything on It

- i 種類のトッピングを乗せるラーメンの杯数を j と決め打ちする
- すると，残り $N-i$ 種類に関して
  - 最初 j 杯には，$N-i$ 種類を自由配置 (緑色領域)
    - i 種類のトッピングの配置の時点で互いに異なる組み合わせになるため，普通に 2^{N-i} 通りの置き方が許される
  - それ以降には，重複が出ないように $N-i$ 種類を自由配置 (灰色領域)
    - トッピングの組み合わせは 2^{N-i} 通り
    - それのうち，どれを注文するかも自由なので結局 $2^{2^{N-i}}$ 通り
- これを踏まえて，次ページで紹介するような状態を定義

1 杯以下にしか乗せられない
i 種類のトッピングを
j 杯に配分 (空ラーメンはダメ)





N-i 種類を自由に置く

N-i 種類を、同じ組み合わせが出ないように自由に置く

Everything on It

$\mathrm{ways}(i)$ を求めるために必要な式

$\mathrm{ways2}(i, j) :=$ 1 杯以下に乗せたい i 種類のトッピングを，以下の条件を満たして j 杯に乗せる場合の数

- i 種類のトッピングのうち，全く登場しないものがあっても良い
  - i 種類の各トッピングについて「0 杯」or「1 杯」に乗っていればよい
- j 杯の各ラーメンについて少なくとも 1 つのトッピングが乗っている

$\mathrm{ways}(i)$ と $\mathrm{ways2}(i)$ の関係

$$\mathrm{ways}(i) = 2^{2^{N-i}} \sum_{j=0}^{i} 2^{(N-i)j} \mathrm{ways2}(i, j)$$

- $\mathrm{ways2}(i, j)$ は動的計画法によって $O(N^2)$ でできる
- どのような遷移になるか？

Everything on It

- i 番目のトッピングまでを j 杯に乗せたとする → $\mathrm{ways2}(i, j)$
- ここに，$i + 1$ 番目のトッピングを新たに追加してみよう

Everything on It

- i 番目のトッピングまでを j 杯に乗せたとする → $\mathrm{ways2}(i, j)$
- ここに，$i + 1$ 番目のトッピングを新たに追加してみよう
  - トッピングを乗せない場合

Everything on It

- i 番目のトッピングまでを j 杯に乗せたとする → $\mathrm{ways2}(i, j)$
- ここに，$i+1$ 番目のトッピングを新たに追加してみよう
  - トッピングを乗せない場合
    - 状態としてなにも変わらない
    - $\mathrm{ways2}(i, j)$ 通りをそのまま持ってくれば良さそう

Everything on It

- i 番目のトッピングまでを j 杯に乗せたとする → $\mathrm{ways2}(i, j)$
- ここに，$i + 1$ 番目のトッピングを新たに追加してみよう
  - トッピングを乗せない場合
    - 状態としてなにも変わらない
    - $\mathrm{ways2}(i, j)$ 通りをそのまま持ってくれば良さそう
  - トッピングを乗せる場合

Everything on It

- i 番目のトッピングまでを j 杯に乗せたとする → $\mathrm{ways2}(i, j)$
- ここに，$i+1$ 番目のトッピングを新たに追加してみよう
  - トッピングを乗せない場合
    - 状態としてなにも変わらない
    - $\mathrm{ways2}(i, j)$ 通りをそのまま持ってくれば良さそう
  - トッピングを乗せる場合
    - 既存の j 杯のラーメンのどれかに乗せるか，新たなラーメンに乗せるか
    - j 杯のラーメンのどれかに乗せる場合，ラーメンは増えない
    - 新たなラーメンに乗せる場合，ラーメンは増える

Everything on It

- i 番目のトッピングまでを j 杯に乗せたとする → $\mathrm{ways2}(i, j)$
- ここに，$i + 1$ 番目のトッピングを新たに追加してみよう
  - トッピングを乗せない場合
    - 状態としてなにも変わらない
    - $\mathrm{ways2}(i, j)$ 通りをそのまま持ってくれば良さそう
  - トッピングを乗せる場合
    - 既存の j 杯のラーメンのどれかに乗せるか，新たなラーメンに乗せるか
    - j 杯のラーメンのどれかに乗せる場合，ラーメンは増えない
    - 新たなラーメンに乗せる場合，ラーメンは増える
- 以上を踏まえると以下のような遷移になる
  - $\mathrm{ways2}(i + 1, j)$ に，$\mathrm{ways2}(i, j) \times (j + 1)$ を配る
  - $\mathrm{ways2}(i + 1, j + 1)$ に，$\mathrm{ways2}(i, j)$ を配る

Everything on It

まとめると以下のようになる

問題の答えとなる式

$$\text{Answer} = \sum_{i=0}^{N} {}_N C_i (-1)^i \text{ways}(i)$$

ways(i) を求めるために必要な式

$$\text{ways2}(i, j) = \text{ways2}(i-1, j-1) + \text{ways2}(i-1, j) \times (j+1)$$

ways(i) と ways2(i) の関係

$$\text{ways}(i) = 2^{2^{N-i}} \sum_{j=0}^{i} 2^{(N-i)j} \text{ways2}(i, j)$$

実装 (C++)

```cpp
using ll = long long int;
// mod_pow と combination 略
ll dp[3010][3010], ways[3010], ans;
int main() {
    int N; cin >> N >> MOD; init();// MOD はグローバル
    // dp (ways2 を求める)
    dp[0][0] = 1;
    for(int i=1; i<=N; i++) for(int j=0; j<=i; j++) {
        if(j > 0) (dp[i][j] += dp[i-1][j-1]) %= MOD;
        (dp[i][j] += dp[i-1][j] * (j+1)) %= MOD;
    }
    // ways1 を求め、答えに足す
    for(int i=0; i<=N; i++) {
        for(int j=0; j<=i; j++) {
            (ways[i] += dp[i][j] * mod_pow(2, (N-i)*j)) %= MOD;
        }
        (ways[i] *= mod_pow(2, mod_pow(2, N-i, MOD-1))) %= MOD;
        ll val = ways[i] * comb[N][i] % MOD;
        if(i % 2 == 0) ans = (ans + val      ) % MOD;
        else           ans = (ans - val + MOD) % MOD;
    }
    cout << ans << endl;
    return 0;
}
```

練習問題

今までの内容が学べる問題を紹介．全部解いて包除に強くなろう！！
まずは簡単めなものを・・・

- yukicoder No. 316: もっと刺激的な FizzBuzz をください (入門) Link
  - 3 つの集合で包除原理を体験したいあなたに
- yukicoder No. 391: CODING WAR (初級) Link
  - Ball and Boxes 3 の制約強化版です．
- yukicoder No. 546: オンリー・ワン (初級) Link
  - 包除を適用する場所がちょっと特殊です．
- AtCoder Beginner Contest 003 D: AtCoder 社の冬 (初級) Link
  - 考え方は初級編でやったものと同様です．自力でやってみよう！

練習問題

以下の問題は，難易度的には中〜上級くらい？

- JOI 春合宿 2011: カードキー Link
- AOJ 2136: Webby Subway Link
- Kyoto University Programming Contest 2018 H: カラフル数列 Link
- yukicoder No.125: 悪の花弁 Link
- CSAcademy Round #71: Losing Nim Link
- University CodeSprint 5: Cube-loving Numbers Link
- CodeChef: Chef and Digits Link
- ウクーニャたんお誕生日コンテスト E: Couple Link

練習問題

以下はすべて AtCoder の問題です

- AtCoder Regular Contest 102 E: Stop. Otherwise... (700) Link
- AtCoder Regular Contest 087 F: Squirrel Migration (800) Link
- AtCoder Regular Contest 101 E: Ribbons on Tree (900) Link
- AtCoder Grand Contest 005 D: ～K Perm Counting (900) Link
- AtCoder Regular Contest 093 F: Dark Horse (1100) Link
- AtCoder Grand Contest 013 E: Placing Squares (1600) Link

練習問題

以下はすべて Codeforces の問題です

- Round #251 Div.2 E: Devu and Birthday Celebration ▸ Link
- Round #305 Div.1 C: Mike and Foam ▸ Link
- Round #313 Div.1 C: Gerald and Giant Chess ▸ Link
- Educational Codeforces Round 20 F: Coprime Subsequences ▸ Link
- Educational Codeforces Round 52 E: Side Transmutations ▸ Link

練習問題

以下はすべて Topcoder の問題です

- SRM 305 Div1 Hard: PowerCollector ▸ Link
- SRM 460 Div1 Easy: TheQuestionsAndAnswersDivOne ▸ Link
- SRM 466 Div1 Hard: DrawingBlackCrosses ▸ Link
- SRM 498 Div1 Hard: FoxJumping ▸ Link
- SRM 602 Div2 Hard: BlackBoxDiv2 ▸ Link
- SRM 603 Div1 Med: PairsOfStrings ▸ Link
- SRM 613 Div1 Med: RandomGCD ▸ Link
- SRM 626 Div1 Hard: ReflectiveRectangle ▸ Link
- 2018 TCO Algorithm Round 3B Med: TestProctoring ▸ Link

さいごに

- 字の誤り・内容の誤り・その他誤りあれば @_TTJR_ まで
- 質問等も @_TTJR_ までお願いします
- たくさんの方が包除原理の練習問題を提供してくれました．ありがとうございます！
- また，作成時に以下の記事をとても参考にしました (順不同)
  - 競技プログラミングにおける包除原理問題まとめ [hamayan さん] Link
  - ビットによる部分集合の列挙 [Sigmar さん] Link
  - 高速ゼータ変換 / 高速メビウス変換 [naoya_t さん] Link
  - AtCoder 版！ 蟻本 (上級編) [drken さん] Link
  - 数え上げテクニック集 [DEGwer さん] Link

- END -