確率 DP を極めよう

確率と期待値の問題解説

tsutaj (@_TTJR_)

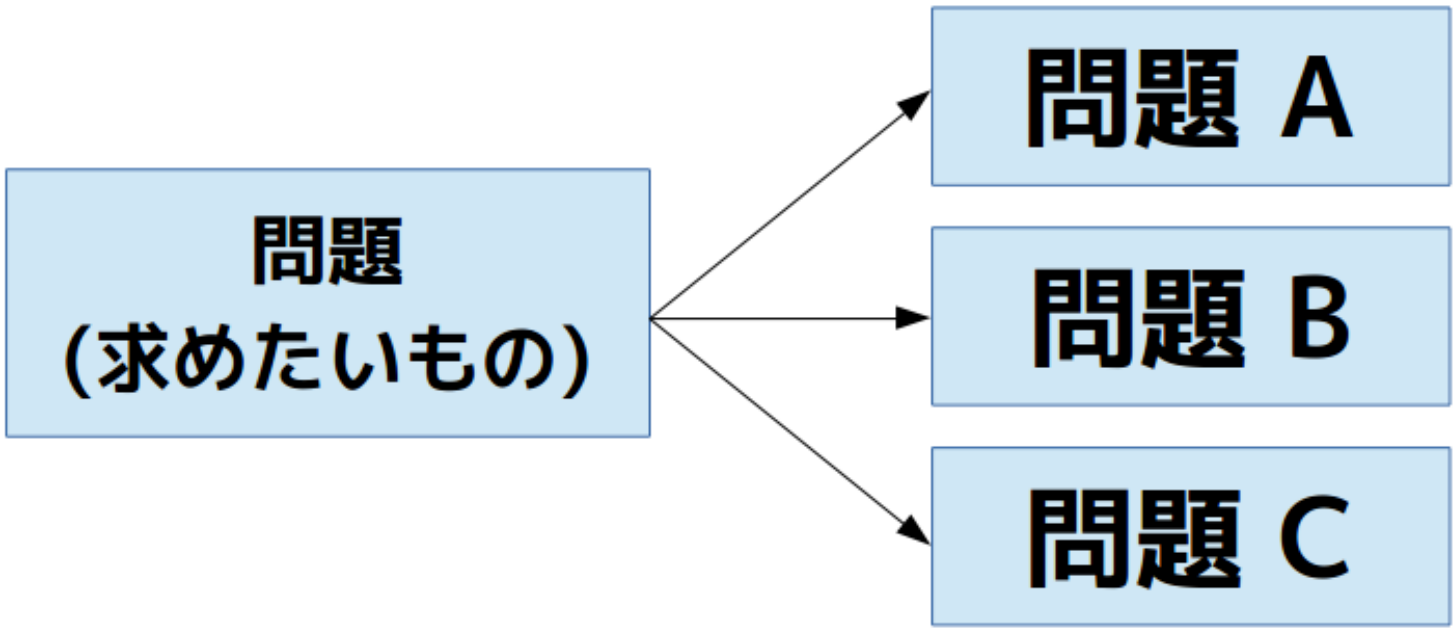
Hokkaido University B4

February 20, 2018

動的計画法とは

動的計画法 (Dynamic Programming)

- 問題を複数の部分問題に分けて，それを統合して解く手法
- 同じとみなせる状態を見極めて，まとめる
- 汎用性がめちゃめちゃある分野



たいせつなこと

DP を解く上で大切なこと

- 持つべき状態を見極めろ！
  - これが一番大事と言っても過言ではなさそう
  - 訓練が必要だと思います
- 典型を知っておけ！
  - 初見で解くのも限界があります
  - ある程度型を知っておくのは大事です
- 余事象を考えろ！
  - 問題で問われた条件に「当てはまらないもの」を数えたほうが，見通しが良い場合があります

というわけで，今回で確率 DP の典型を知りましょう
※ 一部，DP じゃない問題もあります

確率

初級編なので，まずは基本的事項のおさらい

確率

- 事象 A が起こる確率 $P(A) = \frac{A \text{ が起こる頻度}}{\text{全事象の頻度和}}$
  - 例: 6 面サイコロを 1 回振って 5 以上が出る確率 $P(x \geq 5) = \frac{1+1}{6} = \frac{1}{3}$
- 事象 A 以外が起こる確率 $P(\bar{A}) = 1 - P(A)$ (余事象)
  - 例: 2 回のコイントスで表が少なくとも 1 回出る確率: $1 - \left(\frac{1}{2}\right)^2 = \frac{3}{4}$
    - これは，「2 回とも裏が出る」の余事象

条件付き確率

- ある事象 B が起こるという条件下で事象 A が起こる確率 $P(A|B) = \frac{P(A \cap B)}{P(B)}$
  - 例: 1 回目のコイントスで表が出たとき，もう 1 回コイントスすることで 2 回連続表になる確率 $P(2 \text{ 回連続表} \,|1 \text{ 回目が表}) = \frac{1/2 \times 1/2}{1/2} = \frac{1}{2}$

出た目の和 (確率編)

問題 (出た目の和 (確率編))

1 から 6 までの整数が等確率に出るサイコロが 1 つある．このサイコロを N 回振るとき，出た目の数の和が K 以上になる確率を求めよ．

- $1 \leq N \leq 2 \times 10^3$
- $1 \leq K \leq 6 \times N$

サンプル

入力 1:	1 4	出力 1:	0.500000000000
入力 2:	3 5	出力 2:	0.981481481481

- 何を状態として持つべき？
- 状態から状態への遷移はどうなる？

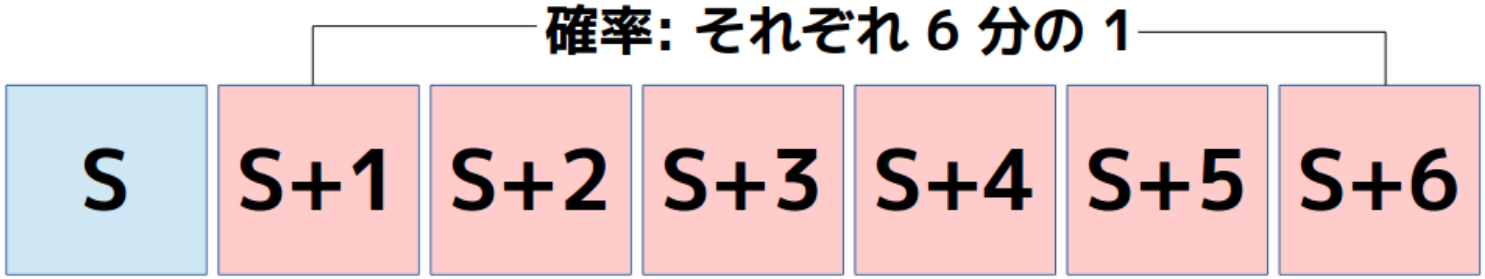
出た目の和 (確率編)

問題 (出た目の和 (確率編))

1 から 6 までの整数が等確率に出るサイコロが 1 つある．このサイコロを N 回振るとき，出た目の数の和が K 以上になる確率を求めよ．

- $1 \leq N \leq 2 \times 10^3$
- $1 \leq K \leq 6 \times N$

- 何を状態として持つべき？
  - i 回サイコロを振ったとき，出た目の数の和が j になる確率
- 状態から状態への遷移はどうなる？
  - サイコロを振ったときに出た目の数だけ，出た目の数の和が増える
  - サイコロの目 x が出る確率は $\frac{1}{6}$



出た目の和 (確率編)

解答コード

- 実際は誤差の関係で，こんな大きい制約では出ないかもです
- 本当にコレを書くとメモリ不足かもしれないので，適宜節約

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;

const int MAX_N = 2010;
double dp[MAX_N][6*MAX_N];
int main() {
    int N, K; scanf("%d%d", &N, &K);
    dp[0][0] = 1.0;
    for(int i=0; i<N; i++) {
        for(int j=0; j<=K; j++) {
            for(int k=1; k<=6; k++) {
                dp[i+1][min(K, j+k)] += dp[i][j] / 6.0;
            }
        }
    }
    printf("%.12f\n", dp[N][K]);
    return 0;
}
```

コイントス

問題 (コイントス)

コイントスを N 回行うとき，表または裏が K 回以上連続で出る確率を求めよ．

- $1 \leq N \leq 2 \times 10^3$
- $1 \leq K \leq N$

出典: エレガントな解法，エレファントな解法 ～モンテカルロ法を添えて～ (改) ▸ Link

サンプル

入力 1:	3 3	出力 1:	0.250000000000
入力 2:	100 10	出力 2:	0.086659044348

- 何を状態として持つべき？
- 状態から状態への遷移はどうなる？

コイントス

問題 (コイントス)

コイントスを N 回行うとき，表または裏が K 回以上連続で出る確率を求めよ．

- $1 \leq N \leq 2 \times 10^3$
- $1 \leq K \leq N$

出典: エレガントな解法，エレファントな解法 ～モンテカルロ法を添えて～ (改) ▸ Link

- 何を状態として持つべき？
  - i 回サイコロを振ったとき，i 回目の結果が j 連続目である確率
- 状態から状態への遷移はどうなる？
  - 直前まで x 連続 → 直前と同じ結果なら $x+1$ 連続，異なれば 1 連続
  - 「K 回以上連続」の余事象 → 「K 回未満連続」
    - x $(1 \leq x < K)$ 回連続する確率の総和を出して，それを 1 から引こう
  - 表と裏を区別する必要なし (直前と同じか違うかだけわかれば良い)

コイントス

解答コード

```cpp
#include <cstdio>
using namespace std;
double dp[2010][2010];
int main() {
    int N, K; scanf("%d%d", &N, &K);
    dp[1][1] = 1;
    for(int i=1; i<N; i++) {
        for(int j=0; j<K; j++) {
            // K 連続は数えてはいけないので捨てる
            if(j+1 < K) dp[i+1][j+1] += dp[i][j] / 2.0;
            dp[i+1][1] += dp[i][j] / 2.0;
        }
    }

    double sum = 0.0;
    for(int i=0; i<K; i++) sum += dp[N][i];
    printf("%.12f\n", 1 - sum);
    return 0;
}
```

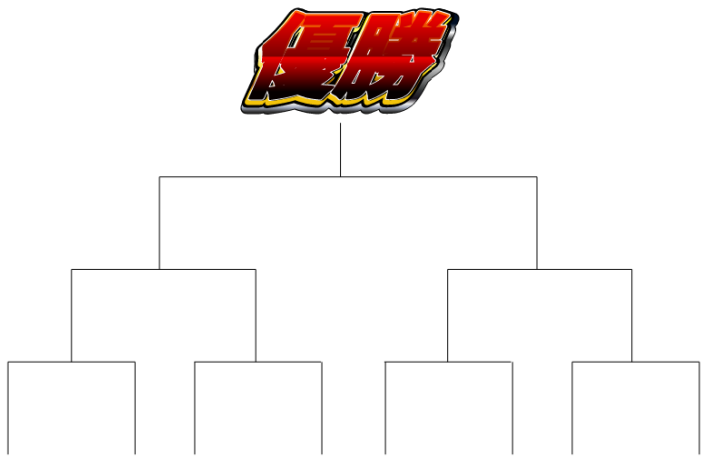
トーナメント

問題 (トーナメント)

2^K 人の参加者で K ラウンドのトーナメント戦を行う．それぞれの人にはレート R_i がついており，人 P と Q が対戦するとき，P が勝つ確率は $\frac{1}{1+10^{(R_Q-R_P)/400}}$ である．それぞれの人について優勝する確率を求めよ．

- $1 \leq K \leq 10$
- $0 \leq R_i \leq 4000$

出典: Typical DP Contest: C ▸ Link



2^K 人が参加、K ラウンド

トーナメント

- 人 i が優勝する → 人 i が K ラウンド目を終えても生き残っている

トーナメント

- 人 i が優勝する → 人 i が K ラウンド目を終えても生き残っている
- $\mathrm{dp}[i][k] :=$ 人 i が k ラウンド目を終えた時に生き残っている確率
  - はじめは全員生き残っているので，$\mathrm{dp}[i][0] = 1.0$
  - 人 i が人 j と対戦した時に勝つ確率を P_{ij} と，k ラウンド目において i と対戦する可能性のある相手の集合を S_{ik} と表すとき，

$$\mathrm{dp}[i][k] = \sum_{j \in S_{ik}} \overbrace{\mathrm{dp}[i][k-1] \times \mathrm{dp}[j][k-1]}^{\text{対戦が実現する可能性}} \times P_{ij}$$

  という更新式になる
  - (対戦する可能性のある相手の集合を出すのがちょっと面倒かも)

トーナメント

- 人 i が優勝する → 人 i が K ラウンド目を終えても生き残っている
- $\mathrm{dp}[i][k] :=$ 人 i が k ラウンド目を終えた時に生き残っている確率
  - はじめは全員生き残っているので，$\mathrm{dp}[i][0] = 1.0$
  - 人 i が人 j と対戦した時に勝つ確率を P_{ij} と，k ラウンド目において i と対戦する可能性のある相手の集合を S_{ik} と表すとき，

$$\mathrm{dp}[i][k] = \sum_{j \in S_{ik}} \overbrace{\mathrm{dp}[i][k-1] \times \mathrm{dp}[j][k-1]}^{\text{対戦が実現する可能性}} \times P_{ij}$$

    という更新式になる
  - (対戦する可能性のある相手の集合を出すのがちょっと面倒かも)
- 答えは $\mathrm{dp}[i][K]$

トーナメント

解答コード

```cpp
#include <cstdio>
#include <cmath>
int N, K, R[1050];
double dp[1050][15];
int main() {
    scanf("%d", &K); N = 1 << K;
    for(int i=0; i<N; i++) {
        scanf("%d", &R[i]);
        dp[i][0] = 1.0;
    }
    for(int k=1; k<=K; k++) {
        for(int i=0; i<N; i++) {
            int lb = (i >> k) << k;
            for(int j=lb; j<lb+(1<<k); j++) {
                if((i >> (k-1) & 1) == (j >> (k-1) & 1)) continue;
                double prob_mtc = dp[i][k-1] * dp[j][k-1];
                double prob_win = 1.0 / (1 + pow(10, (R[j]-R[i]) / 400.0));
                dp[i][k] += prob_mtc * prob_win;
            }
        }
    }
    for(int i=0; i<N; i++) printf("%.12f\n", dp[i][K]);
    return 0;
}
```

期待値

期待値

- 確率変数 X: 値 x_i が p_i の確率で出る変数
- X の期待値 $E[X] = \sum_{i=1}^{n} x_i \times P(X = x_i)$
  - 例: サイコロを 1 回振って出る目の期待値 $E_1 = \frac{1+2+3+4+5+6}{6} = 3.5$
- 期待値の線形性: $E[X + Y] = E[X] + E[Y]$
  - X と Y が独立でなくても成立！
  - 例: サイコロを 10 回振って出る目の期待値 $E_{10} = 10 \times E_1 = 35$
- X と Y が独立である場合，$E[XY] = E[X]E[Y]$

条件付き期待値

- 確率変数 Y の値が y であるという条件下での X の期待値
  $E[X|Y = y] = \sum_x x \times \frac{P(X=x, Y=y)}{P(Y=y)} = f(y)$ (y の関数)
  - 1 つめのサイコロで 2 が出たとき，2 つのサイコロの目の積の期待値
    $E = \frac{1\times2+2\times2+\cdots+6\times2}{6} = 7$

出た目の和 (期待値編)

問題 (出た目の和 (期待値編))

1 から 6 までの整数が等確率に出るサイコロが 1 つある．出た目の数の和が K 以上になるまでこのサイコロを振るとき，サイコロを振る回数の期待値を求めよ．

- $1 \leq K \leq 10^5$

サンプル

入力 1:	1	出力 1:	1.000000000000
入力 2:	5	出力 2:	1.852623456790

- 何を状態として持つべき？
- 状態から状態への遷移はどうなる？

出た目の和 (期待値編)

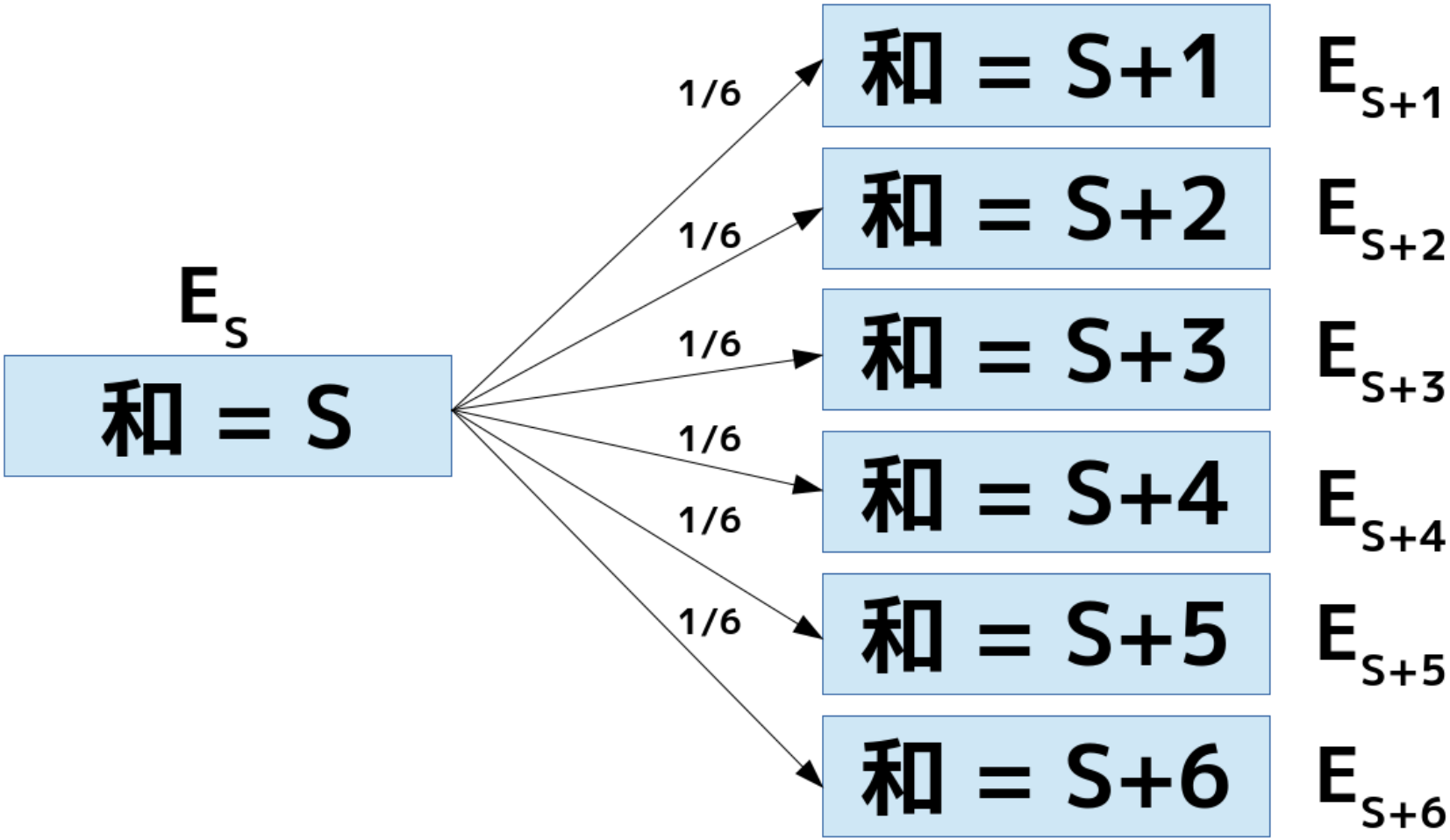
問題 (出た目の和 (期待値編))

1 から 6 までの整数が等確率に出るサイコロが 1 つある．出た目の数の和が K 以上になるまでこのサイコロを振るとき，サイコロを振る回数の期待値を求めよ．

- $1 \leq K \leq 10^5$

- 何を状態として持つべき？
  - 和が S (状態 S) から K 以上にするために必要な回数の期待値 E_S
- 状態から状態への遷移はどうなる？
  - 和が K 以上の状態では，この期待値は明らかに 0
  - 今の和が S で，出た目が x であるという条件下での期待値を考えよう
    - 状態 S でサイコロを振って x が出た場合，状態 $S + x$ に遷移
    - 状態 $S + x$ から K 以上にするために必要な回数の期待値は E_{S+x}
    - 出た目が x であるときの条件付き期待値は，S から $S + x$ にするまでに 1 回要していることから $E_{S+x} + 1$ で，その確率は $\frac{1}{6}$

出た目の和 (期待値編)



次の状態にするために 1 回サイコロを振っている
→ 今の期待値 = 次の状態の期待値 + 1

出た目の和 (期待値編)

よって，それぞれの期待値について以下が成立

$$E_S = \begin{cases} 0 & (S \geq K) \\ \sum_{x=1}^{6} \frac{E_{S+x}+1}{6} & (S < K) \end{cases} \tag{1}$$

- 後ろから決定していくので，E_{K-1} から順に求める
- 答えは E_0

出た目の和 (期待値編)

解答コード

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;

const int MAX_K = 100010;
double dp[MAX_K];
int main() {
    int K; scanf("%d", &K);
    for(int i=K-1; i>=0; i--) {
        for(int k=1; k<=6; k++) {
            dp[i] += (dp[i+k] + 1.0) / 6.0;
        }
    }
    printf("%.12f\n", dp[0]);
    return 0;
}
```

HSI

問題 (HSI)

テストケースが N 個ある問題に対して，M ケースでは実行にそれぞれ 1900ms 要して $\frac{1}{2}$ の確率で正解し，残りの $N-M$ ケースでは実行にそれぞれ 100ms 要して必ず正解するプログラムを，以下の規則に従って提出する

- 提出した結果，M ケースのうちいずれかが不正解である場合は，再度提出する.
- 一度の提出で全てのテストケースに正解するまでこれを繰り返す.

Accepted するまでのプログラムの実行時間の総和の期待値を出力せよ.

- $1 \leq N \leq 100$
- $1 \leq M \leq \min(N, 5)$

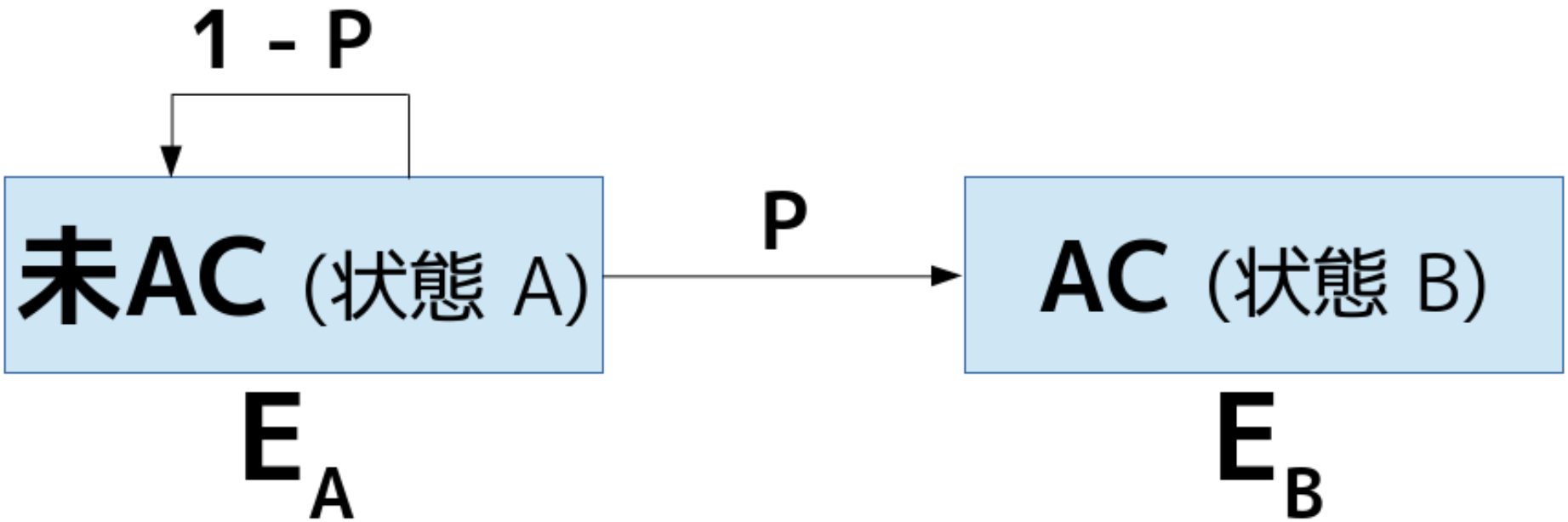
出典: AtCoder Regular Contest 085: C ▸ Link

サンプル

入力: 1 1 出力: 3800

HSI

まずは状態遷移の図を書こう

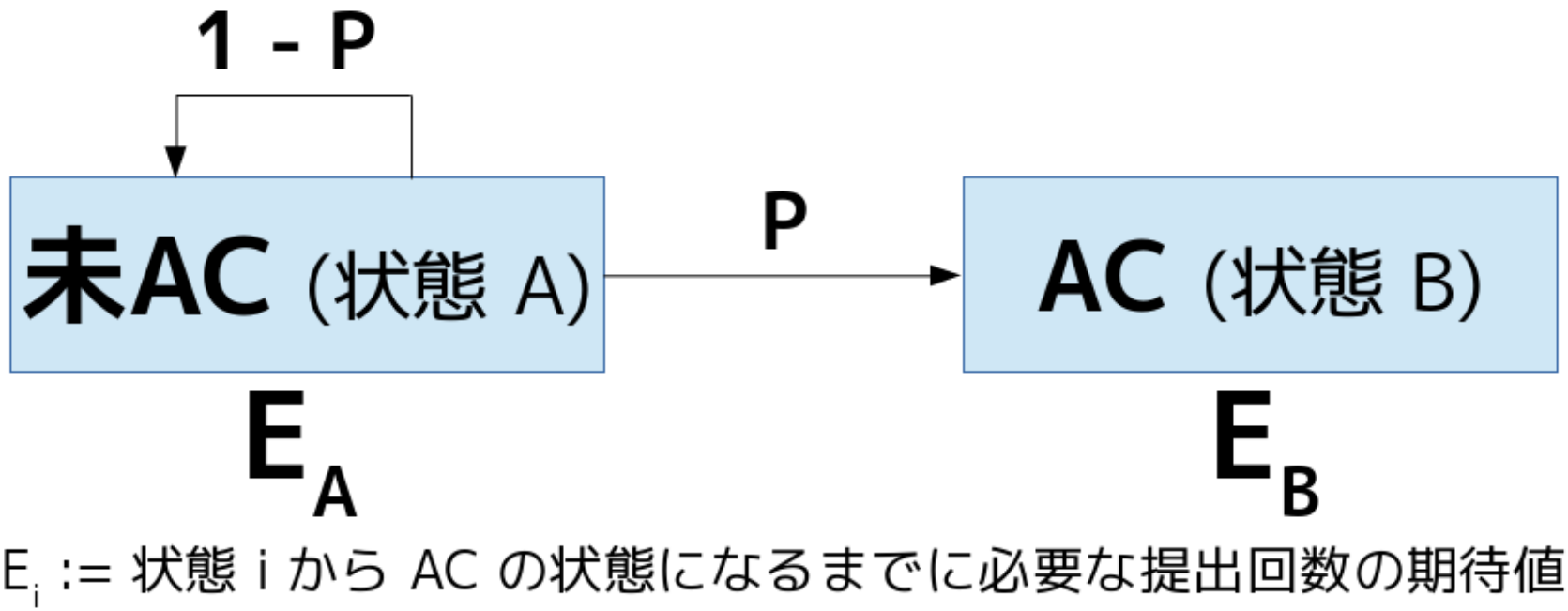


E_i := 状態 i から AC の状態になるまでに必要な提出回数の期待値

期待値 E_A, E_B それぞれについて立式しよう！

HSI

- $P = \left(\frac{1}{2}\right)^M$
- $E_B = 0$ (もう AC しているため)
- $E_A = (1-P) \times (E_A + 1) + P \times (E_B + 1) = (1-P)E_A + 1$
  - さらに変形し，$E_A = \frac{1}{P}$



つまり，未 AC の状態から AC するまでに必要な回数の期待値は $\frac{1}{P} = 2^M$
(この「出るまで試行する」期待値はよく出てきます)

HSI

- 1 回の提出でかかる時間は簡単に分かる
  - M ケースで 1900ms，$N-M$ ケースで 100ms
    $\rightarrow 1900M + 100(N-M) = 100N + 1800M$
- 実行時間総和の期待値 = 実行時間 $\times$ 回数の期待値

解答コード

```cpp
#include <cstdio>
using namespace std;
int main() {
    int N, M; scanf("%d%d", &N, &M);
    int submit_time = 100*N + 1800*M;
    printf("%d\n", submit_time * (1 << M));
    return 0;
}
```

中級編

このセクションでは以下を扱います

- 期待値の線形性
- 状態をシンプルに見る訓練

慣れるまで難しいですが，どれも大事です

エラトステネスのざる

問題 (エラトステネスのざる)

2 から N までの整数からなるリスト A と，空のリスト B がある．
リスト A が空になるまで，「リスト A 内の最小の要素 x をリスト B に移動させ，リスト A から $2x, 3x, 4x, \ldots$ をリスト A から削除する」という操作を行う．
各操作で下線部の処理が行われる確率が p であるとき，最終的にリスト B にある要素数の期待値を求めよ．

- $2 \leq N \leq 10^6$
- $0 \leq p \leq 1$ (小数)

出典: yukicoder No.144 ▸ Link

サンプル

入力 1:	10 1.00000	出力 1:	4.000000000000
入力 2:	10 0.50000	出力 2:	5.750000000000

エラトステネスのざる

- $X(S) :=$ 集合 S のうち，素数として列挙される数の個数
  - 求めたいものは $E[X\{2,\ldots,N\}]$
  - $X\{2,\ldots,N\} = X\{2\} + \cdots + X\{N\}$
  - 期待値の線形性より，
    $E[X\{2,\ldots,N\}] = E[X\{2\}] + \cdots + E[X\{N\}] = P_2 + \cdots + P_N$
  - 各要素について，リスト B に移される確率を求めて足せば良い！
- ある要素 x がリスト A から選択されてリスト B に移されると，x の倍数が全てリスト A から消える
- 要素 x がリスト B に移されるように生き残るための条件
  - x の約数が全て生き残ること！
  - こうなるためには，x の約数が全て選択されない必要がある
  - 「選択されない」は「選択される」の余事象なので，確率は $1-p$
  - x の 1 以外の約数の数を d とすると，x が生き残る確率は $(1-p)^{d-1}$
- 各数字について約数がいくつあるか数えて，上で述べたように計算
- エラトステネスっぽくやると $O(N \log N)$
  - $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots \approx \log N$

エラトステネスのざる

解答コード

```cpp
#include <cstdio>
#include <cmath>
using namespace std;
int divisor[1000010];
int main() {
    int N; double p, ans = 0.0;
    scanf("%d%lf", &N, &p);
    for(int k=2; k<=N; k++) {
        for(int m=k; m<=N; m+=k) {
            divisor[m]++;
        }
        ans += pow(1-p, divisor[k]-1);
    }
    printf("%.12f\n", ans);
    return 0;
}
```

トリプルカードコンプ

適切に・シンプルに状態を管理する訓練となる問題です

問題 (トリプルカードコンプ)

- N 種類のカードを，各種類とも最低 3 枚ずつ集めたい
- 手持ちのカードによらず，1 枚カードを買うと各種類とも $\frac{1}{N}$ の確率で手に入る
- 初めにどの種類を何枚持っているかの情報が与えられるので，コンプリートするまでに買うことになるカードの枚数の期待値を求めよ
  - $1 \leq N \leq 100$
  - i 種類目のカードの所持数 $0 \leq A_i \leq 10$

出典: yukicoder No.108 ▸ Link

トリプルカードコンプ

- カードの種類が 100 種類もあるし，3 枚集めなければならない → ビットで管理するのは無理そう

トリプルカードコンプ

- カードの種類が 100 種類もあるし，3 枚集めなければならない → ビットで管理するのは無理そう
- カードを区別せず，「k 枚持っているカードが何種類あるか」を管理するとどうなる？

トリプルカードコンプ

- カードの種類が 100 種類もあるし，3 枚集めなければならない → ビットで管理するのは無理そう
- カードを区別せず，「k 枚持っているカードが何種類あるか」を管理するとどうなる？
  - k は 0 から 3 までの値を取る (3 以上は全て 3 にまとめる)
  - カードの種類数は一定 → どれか 1 つは引けば出るので 3 通りを覚える
  - 各 k において，カードの種類数は高々 100
  - これで状態数が $O(N^3)$ になって，できそう！

トリプルカードコンプ

- カードの種類が 100 種類もあるし，3 枚集めなければならない → ビットで管理するのは無理そう
- カードを区別せず，「k 枚持っているカードが何種類あるか」を管理するとどうなる？
  - k は 0 から 3 までの値を取る (3 以上は全て 3 にまとめる)
  - カードの種類数は一定 → どれか 1 つは引けば出るので 3 通りを覚える
  - 各 k において，カードの種類数は高々 100
  - これで状態数が $O(N^3)$ になって，できそう！
- より具体的には $\mathrm{dp}[X][Y][Z] :=$ 1 枚以上あるカードが X 種類，2 枚以上あるカードが Y 種類，3 枚以上あるカードが Z 種類ある状態からコンプリートするために必要な回数の期待値
  - 0 枚持っているカードの種類数は，「1 枚以上」の情報から計算

トリプルカードコンプ

- $\mathrm{dp}[X][Y][Z] := 1$ 枚以上あるカードが X 種類，2 枚以上あるカードが Y 種類，3 枚以上あるカードが Z 種類ある状態からコンプリートするために必要な回数の期待値
- これを (X, Y, Z) と略記するとき，状態遷移は以下のようになる

トリプルカードコンプ

- $\mathrm{dp}[X][Y][Z] :=$ 1 枚以上あるカードが X 種類，2 枚以上あるカードが Y 種類，3 枚以上あるカードが Z 種類ある状態からコンプリートするために必要な回数の期待値
- これを (X, Y, Z) と略記するとき，状態遷移は以下のようになる
  - 現在 0 枚持っているカードを引いた → 1 枚持っている種類数が増えるので $(X+1, Y, Z)$ に遷移
  - 現在 1 枚持っているカードを引いた → 2 枚持っている種類数が増えるので $(X, Y+1, Z)$ に遷移
  - 現在 2 枚持っているカードを引いた → 3 枚持っている種類数が増えるので $(X, Y, Z+1)$ に遷移

トリプルカードコンプ

- $\mathrm{dp}[X][Y][Z] :=$ 1 枚以上あるカードが X 種類，2 枚以上あるカードが Y 種類，3 枚以上あるカードが Z 種類ある状態からコンプリートするために必要な回数の期待値
- これを (X, Y, Z) と略記するとき，状態遷移は以下のようになる
  - 現在 0 枚持っているカードを引いた → 1 枚持っている種類数が増えるので $(X+1, Y, Z)$ に遷移
  - 現在 1 枚持っているカードを引いた → 2 枚持っている種類数が増えるので $(X, Y+1, Z)$ に遷移
  - 現在 2 枚持っているカードを引いた → 3 枚持っている種類数が増えるので $(X, Y, Z+1)$ に遷移
- (N, N, N) でコンプリート，この状態における期待値は 0
- 最初の状態からメモ化再帰で答えを出せば良い

トリプルカードコンプ

解答コード

```
int N;
double dp[110][110][110];
int cnt[4];

double solve(int x, int y, int z) {
    // メモ化
    if(dp[x][y][z] != -1) return dp[x][y][z];
    dp[x][y][z] = 0;
    // 2 枚以下しか持っていないものが出るまで引く回数の期待値
    double exp_num = 1.0 * N / (N - z);
    // (条件付き期待値) * (条件付き確率) の足しあわせ
    if(x != N) dp[x][y][z] += (solve(x+1, y, z) + exp_num) * (N - x) / (N - z);
    if(y != x) dp[x][y][z] += (solve(x, y+1, z) + exp_num) * (x - y) / (N - z);
    if(z != y) dp[x][y][z] += (solve(x, y, z+1) + exp_num) * (y - z) / (N - z);
    return dp[x][y][z];
}

signed main() {
    cin >> N;
    repq(i,0,N) repq(j,0,N) repq(k,0,N) dp[i][j][k] = -1;
    dp[N][N][N] = 0;

    rep(i,0,N) {
        int A; cin >> A;
        cnt[min(A, 3LL)]++;
    }
    repr(i,2,0) cnt[i] += cnt[i+1];

    printf("%.12f\n", solve(cnt[1], cnt[2], cnt[3]));
    return 0;
}
```

コイン

適切に・シンプルに状態を管理する訓練となる問題です

問題 (コイン)

- 裏表が区別でき，1 枚につき 1 つ正の整数が書かれた N 枚のコイン (コインはそれぞれ区別できる)
- これらのコインを無作為に ($N!$ 通りの組み合わせが等しく出るように) 一列に並べ，以下を実行
  1. すべてのコインを表向きにする
  2. 左端のコインから順に，現在見ているコインより右側にあるコインのうち，現在見ているコインに書かれている整数の倍数が書かれているコイン全ての表裏をひっくり返す
- この操作を実行したあとに，表を向いているコインの枚数の期待値を求めよ
  - $1 \leq N \leq 100$
  - i 番目のコインに書かれた整数 $1 \leq C_i \leq 10^9$

出典: AtCoder Beginner Contest 008: C ▸ Link

コイン

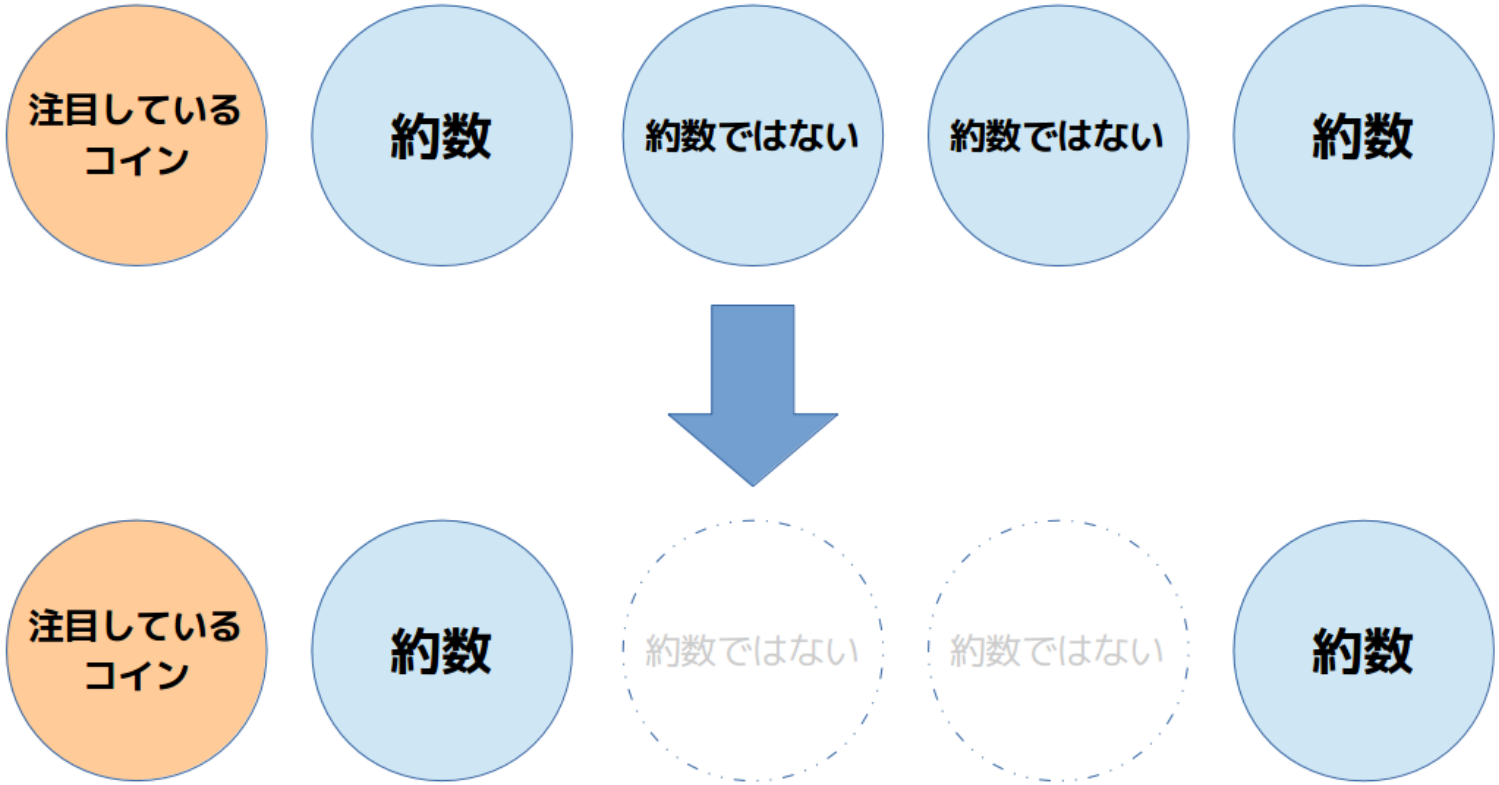
- 全部の場合を試すのはもちろん無理
- i 番目のコインを X_i，集合 S のうち表になっている枚数を $C(S)$ とするとき，$E[C(S)] :=$ 「S 内のコインで，表になる枚数の期待値」が求められれば良い
- 期待値の線形性より，

$$\begin{aligned} E[C\{X_1, \ldots, X_N\}] &= E[C\{X_1\} + \cdots + C\{X_N\}] \\ &= E[C\{X_1\}] + \cdots + E[C\{X_N\}] \\ &= P_{X_1} + \cdots + P_{X_N} \end{aligned}$$

  と変形できる
- よって，「i 番目のコインが表になる確率」を求めて，それを足しあわせたものが答え！

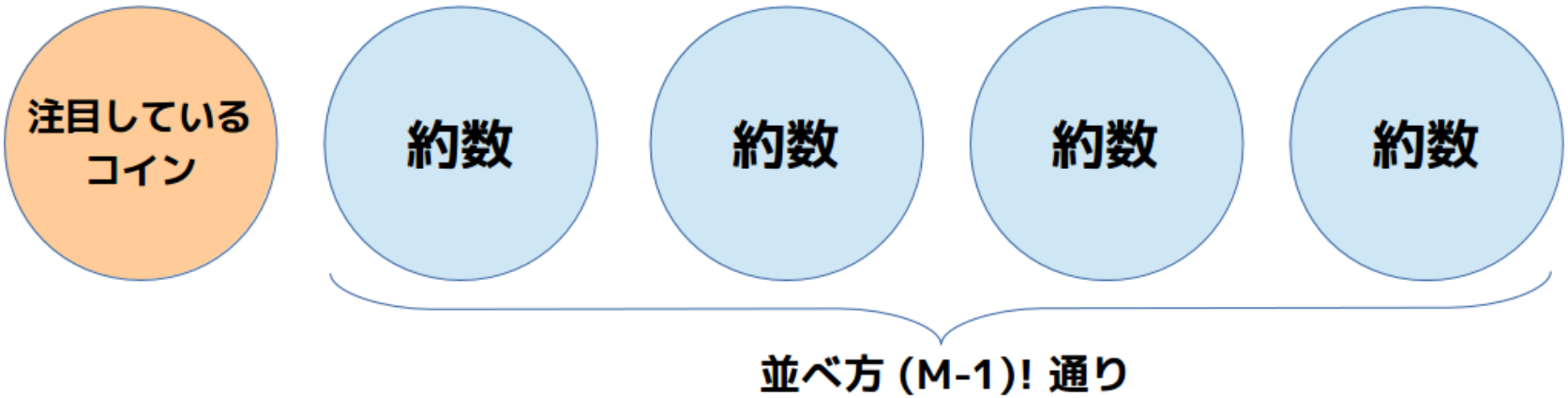
コイン

- i 番目のコインが表になる確率を求めよう
- 問題設定より，i 番目のコインに書かれている数字の約数が書かれているコインが左に偶数枚あれば表，そうでなければ裏
- i 番目のコインに対して，その約数しか答えに影響しないため，自分自身と約数コインのみを取り出した状態で考える



コイン

- いま注目しているコインと，その約数を 1 列に並べることを考える (説明のため全部で M 枚あるとする)
- 注目しているコインがどこに来ようとも，その他 $M-1$ 枚のコインの並び方は $(M-1)!$ 通り → 注目しているコインが左から何番目に来るかは等確率！
- よって，注目しているコインが表になる確率がわかる
  - M が偶数のとき，$\frac{1}{2}$
  - M が奇数のとき，$\frac{M+1}{2M}$
- これを足し合わせれば，終わり！



コイン

解答コード

```cpp
#include <cstdio>
int N, C[110], div[110];
int main() {
    scanf("%d", &N);
    for(int i=0; i<N; i++) scanf("%d", &C[i]);
    for(int i=0; i<N; i++) {
        for(int j=0; j<N; j++) {
            div[i] += !(C[i] % C[j]);
        }
    }
    double ans = 0;
    for(int i=0; i<N; i++) {
        if(div[i] % 2 == 0) ans += 1.0 / 2.0;
        else ans += (div[i] + 1.0) / (2.0 * div[i]);
    }
    printf("%.12f\n", ans);
    return 0;
}
```

上級編

このセクションでは以下を扱います

- もっと状態をシンプルに見る訓練
- 適切な状態遷移
- bit DP との組み合わせ

正直ここの内容は難しいので，一気に理解する必要はないです
(が，必ず後で追いましょう)

RareItems

問題 (RareItems)

N 個のアイテムが出るくじがある．i 番目のアイテムが出る頻度は f_i である (すなわち，i 番目が出る確率は $\frac{f_i}{\sum_{k=1}^{N} f_k}$)．$N$ 個すべてのアイテムを出すまでに必要な，くじを引く回数の期待値を求めよ．

- $1 \leq N \leq 20$
- $1 \leq f_i \leq 1000$

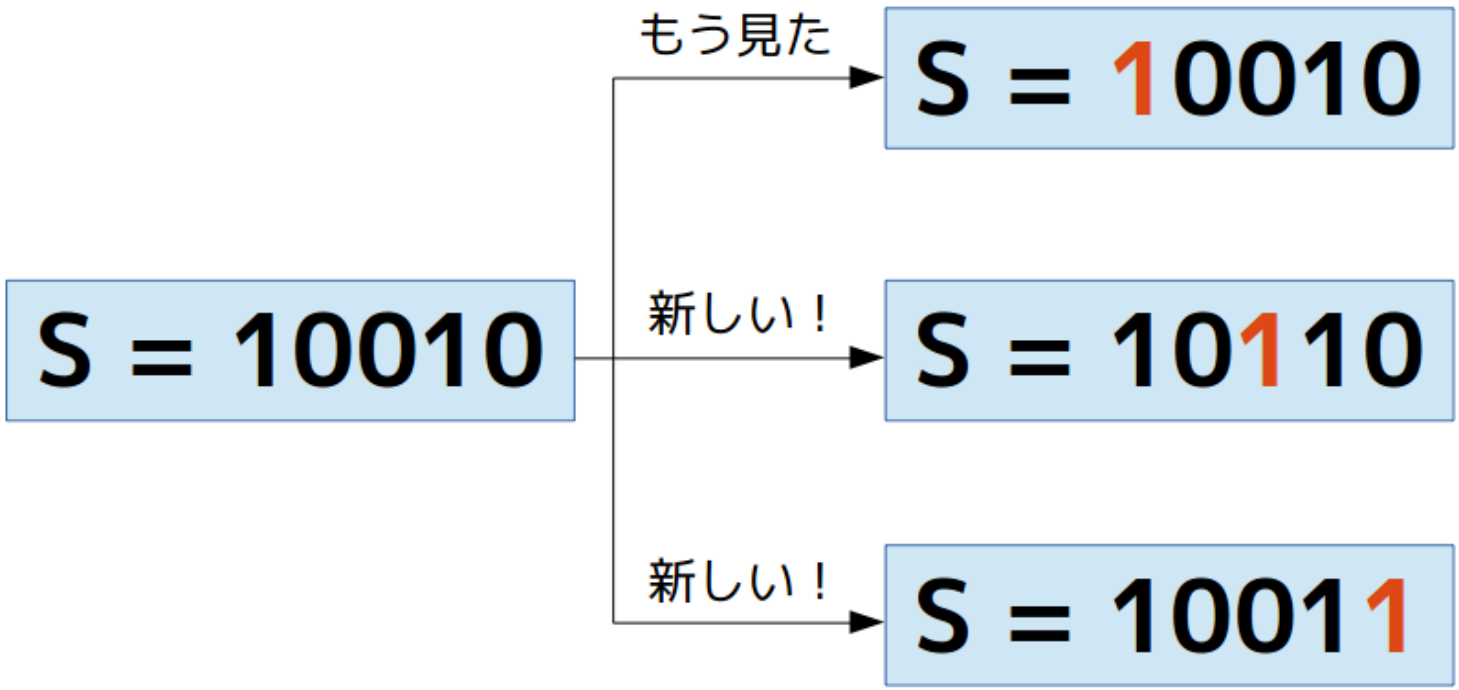
出典: TopCoder SRM 729 Div2 Hard ▸ Link

サンプル

入力 1:	`f = {1}`	出力 1:	`1.0`
入力 2:	`f = {2, 2}`	出力 2:	`3.0`

RareItems

- 持つべき状態 → $E_S :=$ 既に持っているアイテムの集合が S で，その状態からフルコンプするまでに必要な回数の期待値
- 集合を管理 → **bit DP** で解ける！
  - 例: アイテムは全 5 種類，現在 1 番目と 4 番目を所持 → 10010 と表現
- くじを引いたら，「まだ持ってないものが出る」か「もう持っているものが出る」のどちらか

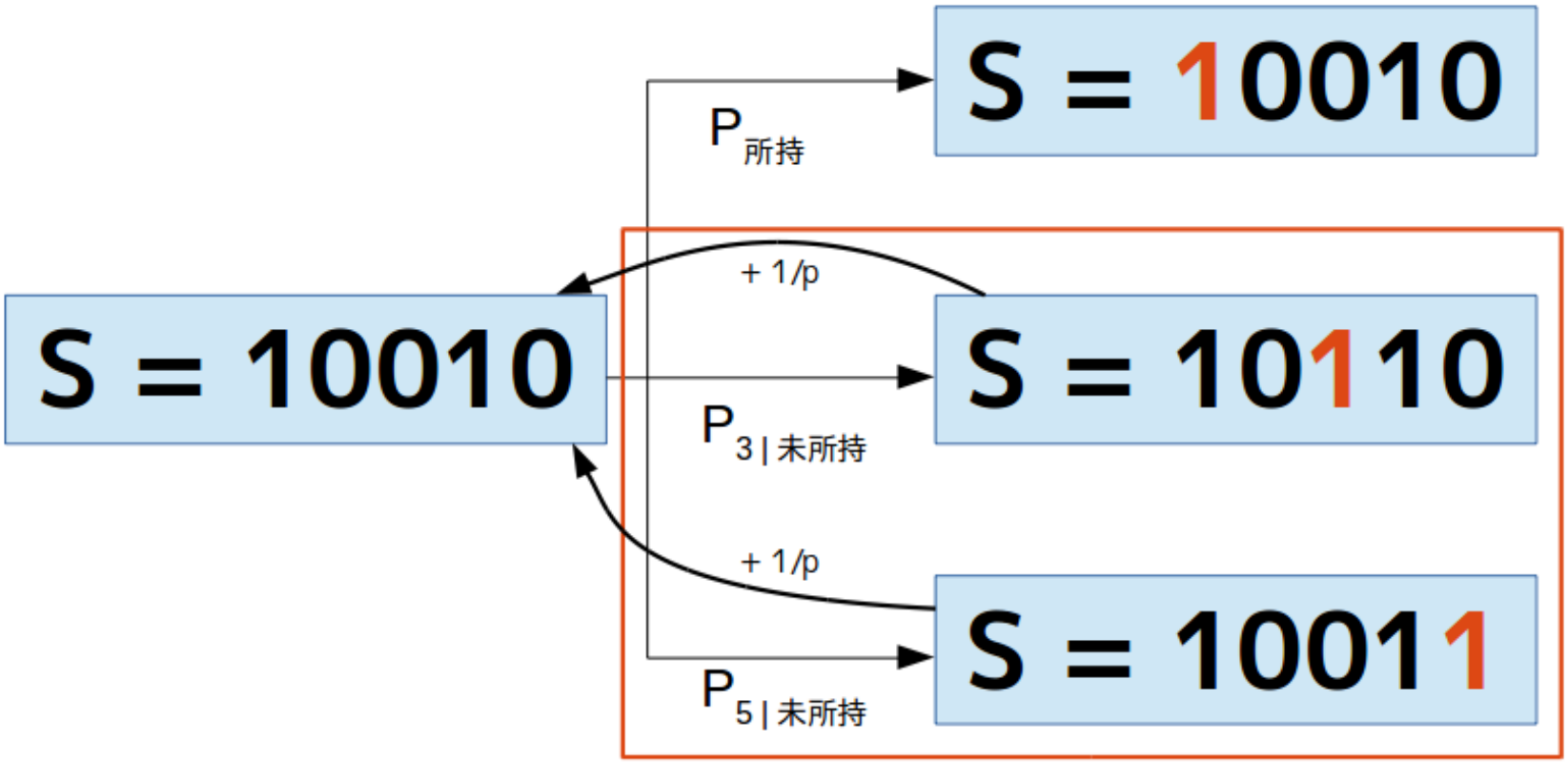


RareItems

- 現在の所持アイテム集合 → S，未所持アイテム → x
  - まだ持っていないので $S \cap \{x\} = \emptyset$
- 未所持アイテムを引く確率は？
  - $\frac{\text{(まだ持っていないアイテムの頻度和)}}{\text{(全てのアイテムの頻度和)}}$
  - この確率を p とおくと，先ほどの問題の知識を用いて，「未所持アイテムのうちどれか 1 つが出るまで必要な回数の期待値」は $\frac{1}{p}$ であることがわかる！
- 未所持アイテムを引いたとき，そのアイテムが x である確率は？
  - これはまさしく「条件付き確率」！
  - $\frac{f_x}{\text{(まだ持っていないアイテムの頻度和)}}$

RareItems

- これをコードで表現すれば良い
- ひとつ先の集合から今の状態へ値を渡すので，降順ループで処理



未所持アイテムを入手するまでの回数期待値 → 1/p
(新しい集合における回数期待値) + 1/p が、必要な回数

RareItems

解答コード

```cpp
double expectedPurchases(vector<int> frequency) {
    int N = frequency.size();
    // すべてのアイテムの頻度の和（全事象）
    int sum = accumulate(frequency.begin(), frequency.end(), 0);

    // dp[S] := 既に得たアイテムの集合が S であるとき、
    //          S からフルコンプするまでに必要な回数の期待値
    vector<double> dp(1<<N, 0.0);
    for(int bit=(1<<N)-1; bit>=0; bit--) {
        // 既に得ているアイテムの頻度和
        int already_get = 0;
        for(int i=0; i<N; i++) {
            if(bit >> i & 1) already_get += frequency[i];
        }

        // 未取得のアイテムの頻度和
        int sum_part = sum - already_get;

        if(sum_part) {
            // 未取得のアイテムが出るまで引く回数の期待値
            double exp_num = 1.0 * sum / sum_part;

            // 未取得のアイテムを何かひとつ引く → 集合を更新
            for(int i=0; i<N; i++) {
                if(!(bit >> i & 1)) {
                    // 未取得のアイテムの集合が決定しているとき、
                    // i 番目のアイテムを引く確率（条件付き確率）
                    double prob = 1.0 * frequency[i] / sum_part;
                    int nbit = bit | (1 << i);
                    dp[bit] += (dp[nbit] + exp_num) * prob;
                }
            }
        }
    }
    return dp[0];
}
```

ボール

問題 (ボール)

N 個のアイテムが一直線上に並んでいる．i 番目のものは座標 x_i に置かれている．すぬけ君が座標 x の点を目指してボールを投げると，$x-1, x, x+1$ のうちのいずれかに $\frac{1}{3}$ の確率で飛んでいき，そこにアイテムがあった場合は倒れる．最適な戦略でボールを投げたとき，アイテムを全て倒すのに必要なボールを投げる回数の期待値を求めよ．

- $1 \leq N \leq 16$
- $0 \leq x_i \leq 15, \quad x_i$ are pairwise distinct.

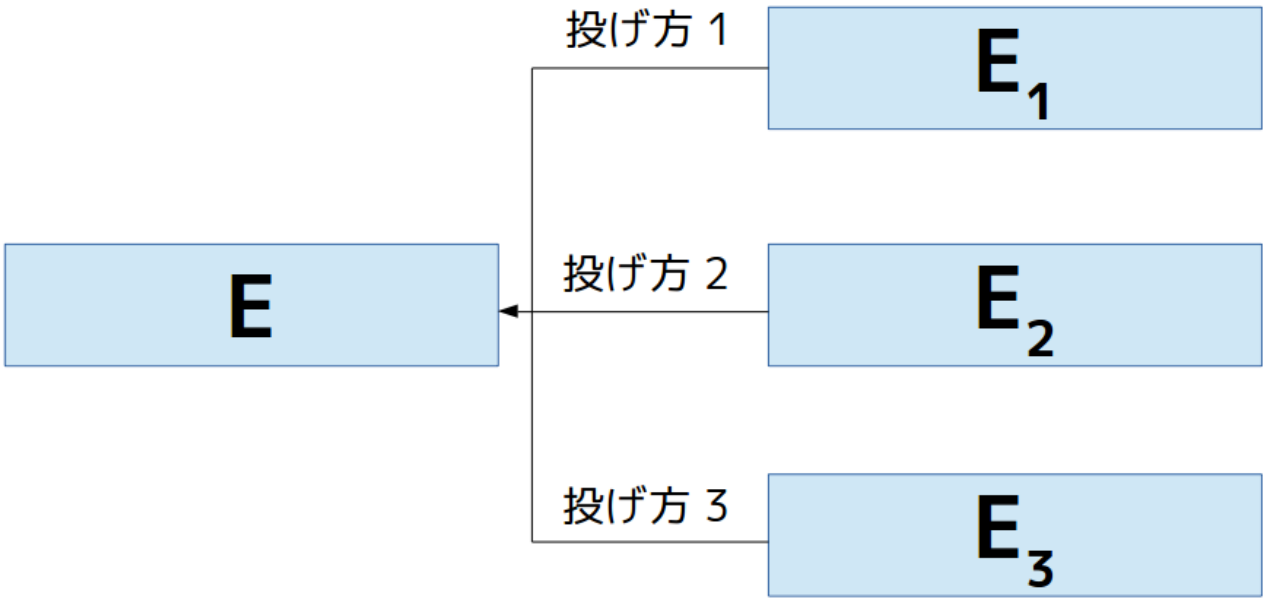
出典: Typical DP Contest: J ▸ Link

サンプル

入力 1:	`N = 2, x = {0, 2}`	出力 1:	`4.500000000`
入力 2:	`N = 5, x = {1, 3, 4, 2, 5}`	出力 2:	`8.986111111`

ボール

- N が小さいので，先ほどと同様に bit で管理すれば良さそう
- $\mathrm{dp}[S] :=$ まだ倒れていないアイテムの集合が S であるとき，その状態から全て倒すまでに必要な回数の期待値
  - $\mathrm{dp}[\emptyset] = 0$
  - どこにボールを投げるのが最適か？　→ 期待値が最小になるところに投げるのが最適！



期待値が最小になるような投げ方を選択しよう！

ボール

- ある座標 x をめがけてボールを投げてみよう
  - $x-1, x, x+1$ いずれにもアイテムがない場合は無駄なので考えない
- $x-1, x, x+1$ にアイテムが k 個 $(1 \leq k \leq 3)$ ある場合，
  - どれかのアイテムにあたるまで投げるとき，その回数の期待値 $\to \frac{3}{k}$ 回
  - 「どれかに当たる」という条件下で，あるアイテム a にあたる確率 $\to \frac{1}{k}$
- よって，座標 x めがけて投げるときの期待値は，

$$\mathrm{dp}[S]_x = \sum_a \left(\mathrm{dp}[S \cup \{a\}] + \frac{3}{k} \right) \times \frac{1}{k} \tag{2}$$

- これを全ての x について試した時の最小値を覚えておけば良い
  - $x = 0, 15$ は端なので試さなくても大丈夫

ボール

解答コード (1)

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;

const double INF = 1e100;
int N, init_bit = 0;
double dp[1 << 16];

double solve(int bit) {
    if(dp[bit] != INF) return dp[bit];
    for(int i=1; i<15; i++) {
        // どこに投げるかを決めて、何箇所にあたる可能性があるか
        int cnt = 0;
        for(int k=-1; k<=1; k++) {
            if(bit >> (i+k) & 1) cnt++;
        }
        // どこかにあたる可能性があるなら、投げてみる
        // 期待値を計算して、その最小値を取る
        // (最小値を出したときの座標に投げるのが最適な戦略)
        if(cnt) {
            double temp = 0.0;
            for(int k=-1; k<=1; k++) {
                if(!(bit >> (i+k) & 1)) continue;
                int nbit = bit ^ (1 << (i+k));
                // 何かにあたるまで投げる回数の期待値 → 3 / cnt
                temp += (solve(nbit) + 3.0 / cnt) / cnt;
            }
            dp[bit] = min(dp[bit], temp);
        }
    }
    return dp[bit];
}
```

ボール

解答コード (2)

```
34 int main() {
35     scanf("%d", &N);
36     for(int i=0; i<N; i++) {
37         int x; scanf("%d", &x);
38         init_bit |= (1 << x);
39     }
40     for(int i=0; i<(1<<16); i++) dp[i] = INF;
41     dp[0] = 0;
42     printf("%.12f\n", solve(init_bit));
43     return 0;
44 }
```

カードゲーム (Hard)

問題 (カードゲーム (Hard))

- 初期状態 → A 君のカード $A_1, \ldots, A_N$，B 君のカード $B_1, \ldots, B_N$
- 各カードには数字が書かれている
- 初期状態から，N 試合からなるゲームを行う
  - A，B 君ともに自分の手札からカードを 1 枚出す
  - 書かれている整数が大きいほうがその試合の勝者．勝者は 2 枚のカードに書かれた整数の和を得点として得る
  - A 君は使えるカードが 2 枚以上あればその中でもっとも小さい整数が書かれたカードを確率 P_A で出し，その他を選ぶ確率は全て等しい．B 君に対しても確率 P_B で同様
- A 君が得られる合計得点の期待値を求めよ
  - $1 \leq N \leq 20$
  - $0.500 \leq P_A, P_B \leq 0.999$
  - $1 \leq A_1, \ldots A_N, B_1, \ldots, B_N \leq 1000$，これらの数字は全て相異なる

出典: yukicoder No. 174 ▸ Link

カードゲーム (Hard)

- $\mathrm{dp}[S_A][S_B] :=$ A 君の持っているカードの集合が S_A で，B 君の持っているカードの集合が S_B であるときの，A 君が得られる合計得点の期待値
  - しかし，これでは $O(2^{2N} \times N^2)$ とかになるので無理！
- A 君，B 君がカードを出す規則は「自分の手札の状態」にのみ依存 (相手の状態は関係ない)
  - 独立に考えるとうまく行きそう！
- $\mathrm{dp}[S_X] :=$ X 君の持っているカードの集合が S_X になる確率 (何試合目かはビットの立っている数から計算可能なので，今回は DP の添字にしなくてもよい)
- この DP から，「誰が何試合目にどのカードを出すか」の確率を全部求めよう！
- あとは各試合について，どのカードを出したかを全通り試し，その確率から期待値を計算

カードゲーム (Hard)

解答コード (1)

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;

int N; double p[2];
int cards[2][20];
double rec[2][20][20];
double dp[1 << 20];

// dp を計算しつつ、誰が何試合目にどのカードを出す確率を求めよう
void solve(int t) {
    fill(dp, dp + (1 << N), 0);
    dp[(1 << N) - 1] = 1.0;
    for(int bit=(1<<N)-1; bit>=0; bit--) {
        bool fst = true;
        for(int i=0; i<N; i++) {
            // カードを何か 1 枚出す
            if(bit >> i & 1) {
                // 新しい集合 (nbit)、何試合目か (step)
                int nbit = bit ^ (1 << i), cnt = __builtin_popcount(bit);
                int step = N - cnt;
                if(cnt == 1) {
                    // 1 枚しかない場合
                    rec[t][i][step] += dp[bit];
                    dp[nbit] += dp[bit];
                }
                else {
                    // 複数枚ある場合 → 最小のカードなら確率 p
                    double prob = (fst ? p[t] : (1 - p[t]) / (cnt - 1));
                    rec[t][i][step] += dp[bit] * prob;
                    dp[nbit] += dp[bit] * prob;
                    fst = false;
                }
            }
        }
    }
}
```

カードゲーム (Hard)

解答コード (2)

```
39 int main() {
40     scanf("%d%lf%lf", &N, &p[0], &p[1]);
41     for(int i=0; i<N; i++) {
42         scanf("%d", &cards[0][i]);
43     }
44     for(int i=0; i<N; i++) {
45         scanf("%d", &cards[1][i]);
46     }
47     sort(cards[0], cards[0] + N);
48     sort(cards[1], cards[1] + N);
49     solve(0); solve(1);
50 
51     double ans = 0.0;
52     for(int step=0; step<N; step++) {
53         for(int x=0; x<N; x++) {
54             for(int y=0; y<N; y++) {
55                 // A 君の出した値の方が大きい場合のみ答えに反映
56                 if(cards[0][x] < cards[1][y]) continue;
57                 ans += (cards[0][x] + cards[1][y]) * rec[0][x][step] * rec[1][y][step];
58             }
59         }
60     }
61     printf("%.12f\n", ans);
62     return 0;
63 }
```

せんべい

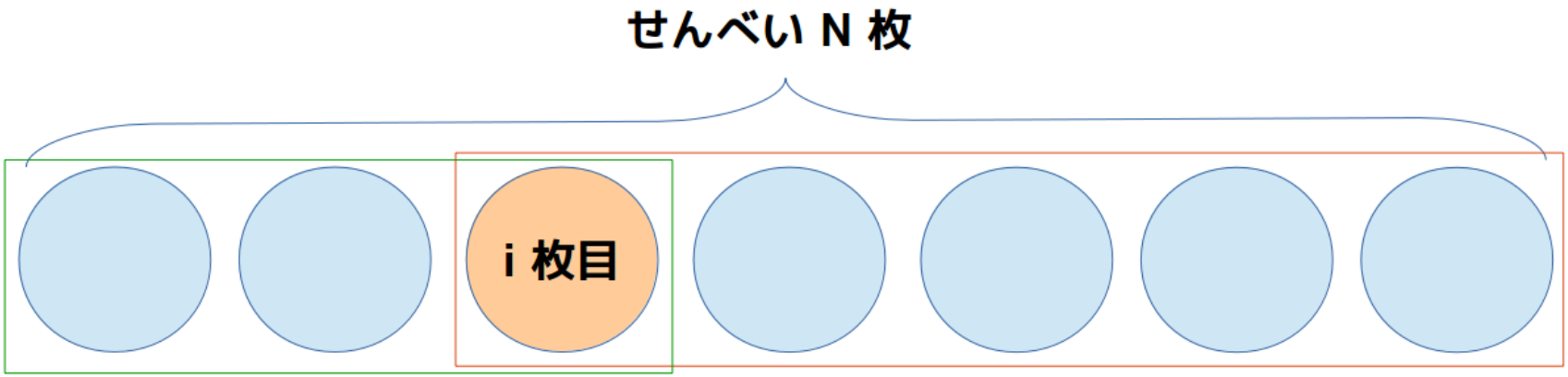
問題 (せんべい)

- 1 ～ N の番号のついた N 枚のせんべいをランダムに ($N!$ 通りから等確率) 1 枚ずつ配布
- シカは K 枚までしか食べられないが，番号 N のせんべいを食べたい
- i 枚目が配られるとき，シカはそれを食べるかどうかをその時点で選ぶ
- せんべいを見てもその番号はわからないが，前に見た (食べたもの含む) 全てのせんべいと大小比較できる
- 最適な戦略を取ったとき，シカが番号 N のせんべいを食べられる確率を求めよ
  - $1 \leq K \leq N \leq 1000$

出典: AtCoder Regular Contest 055: B ▸ Link

せんべい

- i 枚目 (0-indexed) に配られるせんべいについて考えよう
- i 枚目に配られたせんべいが今までで最大でなければ，取らないほうが良い (明らかに N でないため)
- 今までで最大なら，取るか取らないかのどちらか (最大値を採用)
  - i 枚目が N である確率 $p = \frac{1}{N-i}$
  - i 枚目が N でないが，今までで最大である確率 $q = \frac{1-p}{i+1}$
  - N である確率と，過去最高である確率が独立ではないことに注意！！



i 枚目が N である確率 **p** → **1/(N-i)**

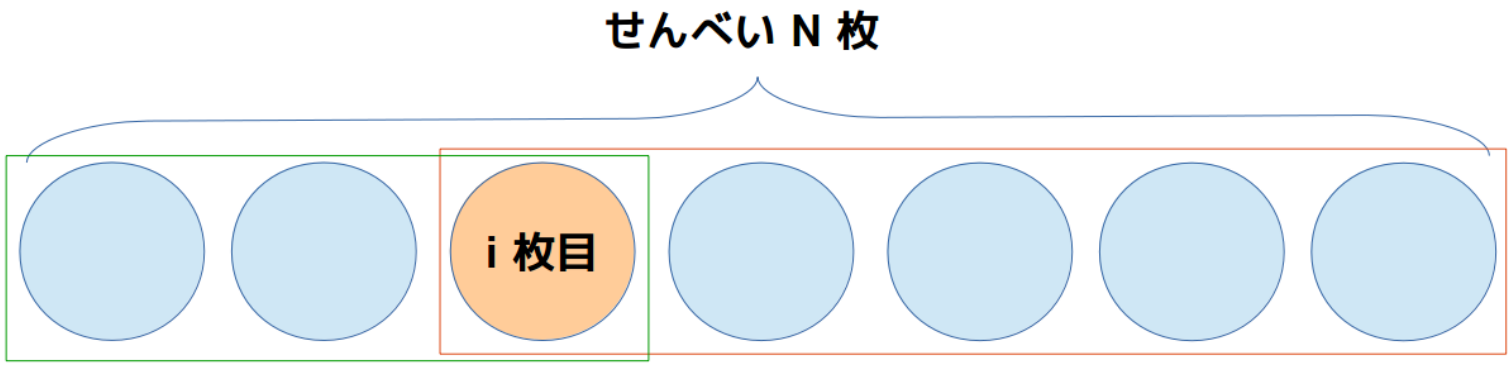
N ではないが今までで最大である確率 **q** → **(1–p) / (i+1)**

(0-indexed)

せんべい

- $\mathrm{dp}[i][k] := i$ 枚目まで配られ，今までに k 枚食べた状態から，番号が N のせんべいを食べられる確率
- $p = \frac{1}{N-i}, q = \frac{1-p}{i+1}$ とおくと，過去最高でない確率 r は $r = 1 - p - q$

$$\begin{aligned} \mathrm{dp}[i][k] &= r \times \mathrm{dp}[i+1][k] \\ &+ \max \left\{ \underbrace{p + q \times \mathrm{dp}[i+1][k+1]}_{\text{せんべいを取る}}, \underbrace{q \times \mathrm{dp}[i+1][k]}_{\text{せんべいを取らない}} \right\} \end{aligned}$$



i 枚目が N である確率 p → 1/(N-i)
N ではないが今までで最大である確率 q → (1–p) / (i+1)
(0-indexed)

せんべい

解答コード

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;

int N, K;
double dp[1010][1010];

double solve(int i, int k) {
    if(dp[i][k] != -1) return dp[i][k];
    if(i == N || k == K) return dp[i][k] = 0;
    // i 枚目が N である
    double is_n   = 1.0 / (N - i);
    // N ではないが、過去最高値である
    double fake_n = (1 - is_n) / (i + 1);
    // 取ったほうが不利にはたらく確率
    double notake = 1.0 - is_n - fake_n;

    // 有利に働かないなら取らないのが最適
    dp[i][k]  = notake * solve(i+1, k);
    // そうでなければ取るか取らないか (最大値を採用)
    dp[i][k] += max(is_n + fake_n * solve(i+1, k+1),
                          fake_n * solve(i+1, k  ));
    return dp[i][k];
}

int main() {
    scanf("%d%d", &N, &K);
    for(int i=0; i<=N; i++) {
        for(int k=0; k<=K; k++) {
            dp[i][k] = -1;
        }
    }
    printf("%.12f\n", solve(0, 0));
    return 0;
}
```

練習問題集

類題などなど．全部解いて確率に強くなろう！！
(もちろんコレ以外にも多数あります)

- (普通) **yukicoder No.58:** イカサマなサイコロ Link
- (普通) **AOJ 1277:** Minimal Backgammon Link
- (普通) **yukicoder No.75:** 回数の期待値の問題 Link
  - 出た目の和 (期待値編) と設定が似ていますが，こちらは和が K を超えると合計を 0 に戻し，和がちょうど K になるまで繰り返します．
  - 制約が大きくなったバージョンもあります
    (**yukicoder No.301:** サイコロで確率問題 **(1)** Link)
- (難) **yukicoder No.210:** 探し物はどこですか？ Link
  - 今回紹介できなかったアプローチです．制約をよく見てみましょう．
- (難) **AtCoder Regular Contest 016 C:** ソーシャルゲーム Link
  - RareItems の類題です．こちらはくじが複数種類あります．
- (難) **yukicoder No.155:** 生放送と BGM Link
  - 状態を 1 つ巻き戻すあの DP の練習にもなります (発展的内容)