

プログラミング実習

第8回：簡単なアルゴリズム

清水 哲也 (shimizu@info.shonan-it.ac.jp)

今回の授業内容

- 前回の課題の解答例
- 素数判定法
- ユークリッドの互除法
- 場合の数とアルゴリズム
- 課題

今回の授業内容は以下の本の内容を抜粋しました

問題解決のための「アルゴリズム×数学」が基礎からしっかり身につく本

前回の課題の解答例

前回の課題の解答例

SAの学生さんによる解答例です。

<https://shimizu-lab.notion.site/2024-11826a533567807390dcfa0a5e288e15?pvs=4>

素数判定法

素数判定法

自然数 N が素数であるかどうかを判定する問題を扱います

素数判定法はいろんなアルゴリズムがありますがここでは以下の 2 つを紹介します

- 単純な素数判定法
- 高速な素数判定法

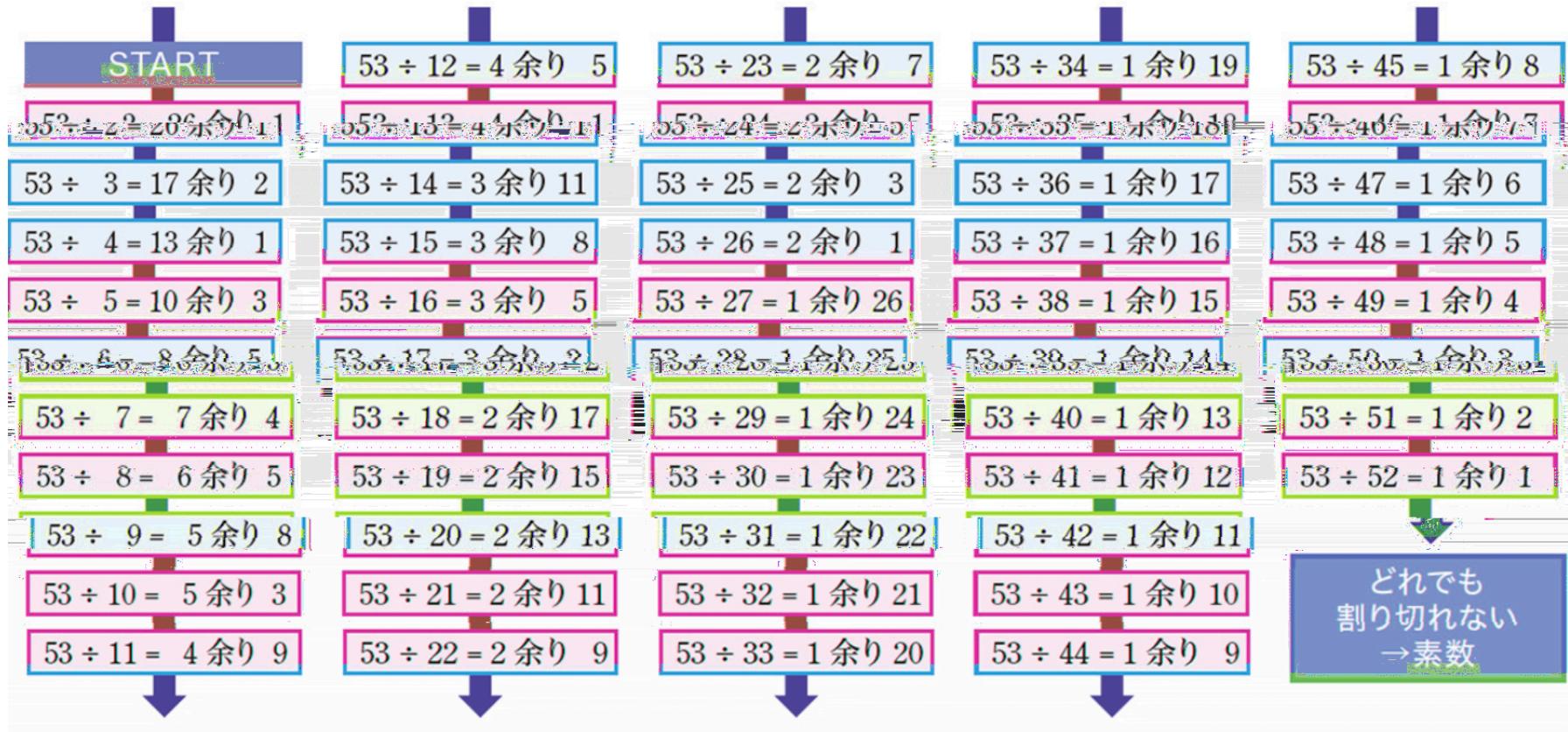
単純な素数判定法

例：53 が素数であるかを判定してみましょう

- とても単純な方法を考えると 2 から 52 まで割り切れるかどうかを調べる方法があります
- デメリットとしては計算に時間がかかることがあります
- 一般の整数 N についても同じように 2 から $N - 1$ まで割り切れるかどうかをしらべることで素数判定を行うことができます
- このアルゴリズムの計算量は $O(N)$ になりかなり時間がかかります

単純な素数判定法

例：53 が素数であるかを判定してみましょう



単純な素数判定法

```
static boolean isPrime(int N) {
    N を整数とし, 素数であれば返す
    for (int i = ; i <= N - ; i++) {
        if (N % i == )
            return false;
    }
    return true;
}
```

このプログラムは `int` の範囲であれば実行できます

`int` の範囲は $-2,147,483,648 \sim 2,147,483,647$ なので, 例えば「2147483647」の素数判定をしてみましょう

これぐらいの数であればほぼタイムラグなく実行できます

javaで扱える整数型

javaで扱える整数型は以下の4種類があります

データ型	ビット数	値
byte	8 bit	-128 - 127 (3桁)
short	16 bit	-32768 - 32767 (5桁)
int	32 bit	-2147483648 - 2147483647 (10桁)
long	64 bit	-9223372036854775808 - 9223372036854775807 (19桁)

より大きい整数を扱うためにはデータ型を `int` から `long` に変更する必要があります
試しに「9999999999989」の素数判定を後でやってみてください
すぐに終わらないので注意！！(私の環境では約32分かかりました)

高速な素数判定法

単純な素数判定法は 2 から $N - 1$ まで全部を調べているため N が大きくなるとその分計算に時間がかかるてしまう

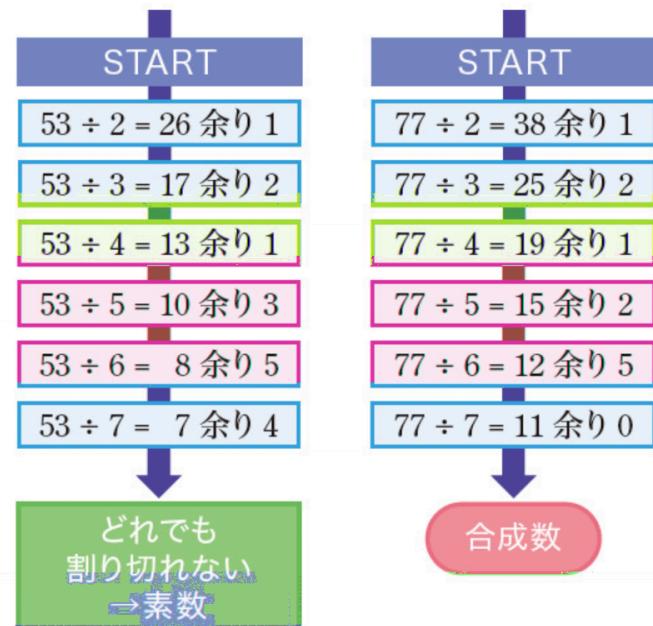
実は 2 から $N - 1$ まで全部を調べる必要はなく 2 から \sqrt{N} まで調べて割り切れなければ N は素数だと言えます

これが正しいことを示すのは省略します

高速な素数判定法

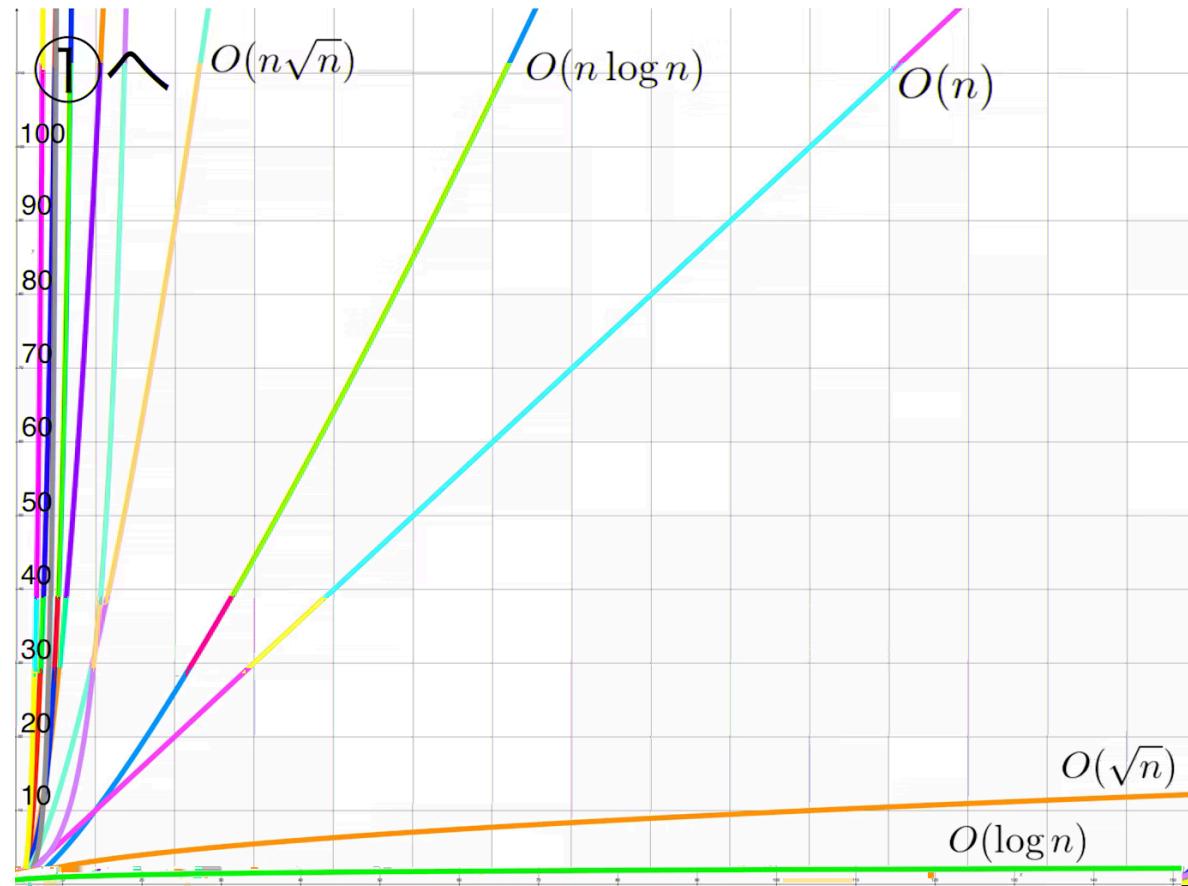
例えば53の場合 $\sqrt{53} = 7.28\cdots$ なので2, 3, 4, 5, 6, 7まで割り切れなければ「53は素数」といえます

例えば77の場合 $\sqrt{77} = 8.77\cdots$ なので2, 3, 4, 5, 6, 7, 8まで調べますが、7で割り切れるので「77は合成数」といえます



高速な素数判定法

このアルゴリズムの計算量は $O(\sqrt{N})$ となり、 $O(N)$ 比較したグラフを示します



<https://qiita.com/kokorinosoba/items/1c7400ca6c740fe9f1ee>

高速な素数判定法

この方法をプログラムにしてみましょう

```
static boolean isPrime(long N) {
    N を 2 以上の整数とし、N が素数であれば true を返す
    for( long i = 2 ; i * i <= N; i++) {
        if(N % i == 0) return false;
    }
    return true;
}
```

このプログラムであれば、「999999999989」の素数判定法も計算時間がわずか数ms程度になるとおもいます

ユークリッドの互除法

ユークリッドの互除法

自然数 A と B の最大公約数を求める問題を扱います

素数判定法と同様に2種類のアルゴリズムを実装してみます

- 単純なアルゴリズム
- 効率的なアルゴリズム：ユークリッドの互除法

ユークリッドの互除法を利用すると計算量は $O(\log(A + B))$ となります

単純なアルゴリズム

33 と 88 の最大公約数を計算してみましょう

ここで明らかなことは、答えが 33 以下になることです

この条件を利用して単純なアルゴリズムは 1 から 33 までの数を使って、33 と 88 両方が割り切れるかどうか調べる方法とします

$$33 \div 1 = 33 \text{ 余り } 0 \quad 88 \div 1 = 88 \text{ 余り } 0$$

$$33 \div 2 = 16 \text{ 余り } 1 \quad 88 \div 2 = 44 \text{ 余り } 0$$

$$33 \div 3 = 11 \text{ 余り } 0 \quad 88 \div 3 = 29 \text{ 余り } 1$$

$$33 \div 4 = 8 \text{ 余り } 1 \quad 88 \div 4 = 22 \text{ 余り } 0$$

$$33 \div 5 = 6 \text{ 余り } 3 \quad 88 \div 5 = 17 \text{ 余り } 3$$

$$33 \div 6 = 5 \text{ 余り } 3 \quad 88 \div 6 = 14 \text{ 余り } 4$$

$$33 \div 7 = 4 \text{ 余り } 5 \quad 88 \div 7 = 12 \text{ 余り } 4$$

$$33 \div 8 = 4 \text{ 余り } 1 \quad 88 \div 8 = 11 \text{ 余り } 0$$

$$33 \div 9 = 3 \text{ 余り } 6 \quad 88 \div 9 = 9 \text{ 余り } 7$$

$$33 \div 10 = 3 \text{ 余り } 3 \quad 88 \div 10 = 8 \text{ 余り } 8$$

$$33 \div 11 = 3 \text{ 余り } 0 \quad 88 \div 11 = 8 \text{ 余り } 0$$

$$33 \div 12 = 2 \text{ 余り } 9 \quad 88 \div 12 = 7 \text{ 余り } 4$$

$$33 \div 13 = 2 \text{ 余り } 7 \quad 88 \div 13 = 6 \text{ 余り } 10$$

$$33 \div 14 = 2 \text{ 余り } 5 \quad 88 \div 14 = 6 \text{ 余り } 4$$

$$33 \div 15 = 2 \text{ 余り } 3 \quad 88 \div 15 = 5 \text{ 余り } 13$$

$$33 \div 16 = 2 \text{ 余り } 1 \quad 88 \div 16 = 5 \text{ 余り } 8$$

$$33 \div 17 = 1 \text{ 余り } 16 \quad 88 \div 17 = 5 \text{ 余り } 3$$

$$33 \div 18 = 1 \text{ 余り } 15 \quad 88 \div 18 = 4 \text{ 余り } 16$$

$$33 \div 19 = 1 \text{ 余り } 14 \quad 88 \div 19 = 4 \text{ 余り } 12$$

$$33 \div 20 = 1 \text{ 余り } 13 \quad 88 \div 20 = 4 \text{ 余り } 8$$

$$33 \div 21 = 1 \text{ 余り } 12 \quad 88 \div 21 = 4 \text{ 余り } 4$$

$$33 \div 22 = 1 \text{ 余り } 11 \quad 88 \div 22 = 4 \text{ 余り } 0$$

$$33 \div 23 = 1 \text{ 余り } 10 \quad 88 \div 23 = 3 \text{ 余り } 19$$

$$33 \div 24 = 1 \text{ 余り } 9 \quad 88 \div 24 = 3 \text{ 余り } 16$$

$$33 \div 25 = 1 \text{ 余り } 8 \quad 88 \div 25 = 3 \text{ 余り } 13$$

$$33 \div 26 = 1 \text{ 余り } 7 \quad 88 \div 26 = 3 \text{ 余り } 10$$

$$33 \div 27 = 1 \text{ 余り } 6 \quad 88 \div 27 = 3 \text{ 余り } 7$$

$$33 \div 28 = 1 \text{ 余り } 5 \quad 88 \div 28 = 3 \text{ 余り } 4$$

$$33 \div 29 = 1 \text{ 余り } 4 \quad 88 \div 29 = 3 \text{ 余り } 1$$

$$33 \div 30 = 1 \text{ 余り } 3 \quad 88 \div 30 = 2 \text{ 余り } 28$$

$$33 \div 31 = 1 \text{ 余り } 2 \quad 88 \div 31 = 2 \text{ 余り } 26$$

$$33 \div 32 = 1 \text{ 余り } 1 \quad 88 \div 32 = 2 \text{ 余り } 24$$

$$33 \div 33 = 1 \text{ 余り } 0 \quad 88 \div 33 = 2 \text{ 余り } 22$$

両方割り切れた最大の数は 11
→最大公約数は 11

単純なアルゴリズム

正の整数の最大公約数を返すメソッド
は（最大公約数）の略

```
static long GCD(long A, long B) {  
    long Answer = ;  
  
    for (long i = ; i <= Math.min(A, B); i++) {  
        if (A % i == 0 && B % i == 0) {  
            Answer = i;  
        }  
    }  
    return Answer;  
}
```

ここで `Math.min(A,B)` は A と B のうち小さい値を返すメソッドです

このプログラムでは余りの計算を $2 \times \min(A, B)$ 回行う必要があり、あまり効率的ではありません

100000000000と123450000000で試してみてください

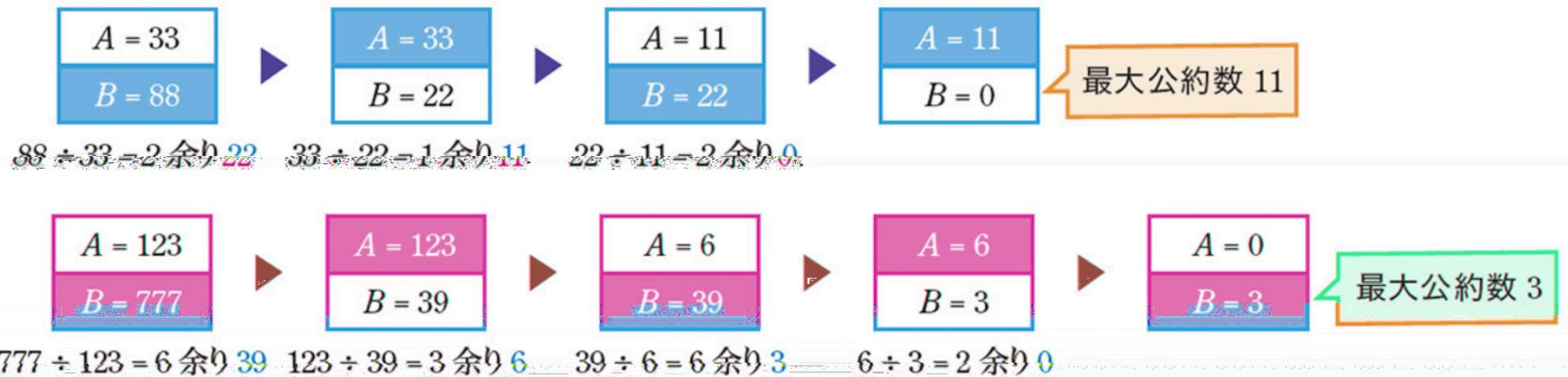
効率的なアルゴリズム：ユークリッドの互除法

以下の方法を利用すると2つの正整数の最大公約数を高速に計算することができます

1. 大きいほうの数を「大きいほうを小さいほうで割った余り」に書き換えるという操作を繰り返す
2. 片方が0になったら操作を終了する。もう片方の数が最大公約数である

効率的なアルゴリズム：ユークリッドの互除法

この方法で 33 と 88 の最大公約数, 123 と 777 の最大公約数をそれぞれ計算すると以下のようになります



このアルゴリズムを ユークリッドの互除法 といいます

A と B の最大公約数を求めるときの計算量は $O(\log(A + B))$ であるため, A, B が 10^{18} 程度であってもある程度一瞬で計算できます

効率的なアルゴリズム：ユークリッドの互除法

```
static long EuclideanGCD(long A, long B) {
    while (A >= 0 && B >= 0) {
        if (A < B) {
            B = B % A;      A < B の場合、大きい方をき換える
        } else {
            A = A % B;      A >= B の場合、大きい方をき換える
        }
    }
    if (A >= 0) {
        return A;
    }
    return B;
}
```

このプログラムで A と B の大小関係によって行うべき操作が変わるので、`if` 文を用いて場合分けを行っています
更に効率的に実装する方法もあります

場合の数とアルゴリズム

場合の数とアルゴリズム

ここでは、階乗・二項係数・積の法則など、基本的な場合の数の公式について扱います

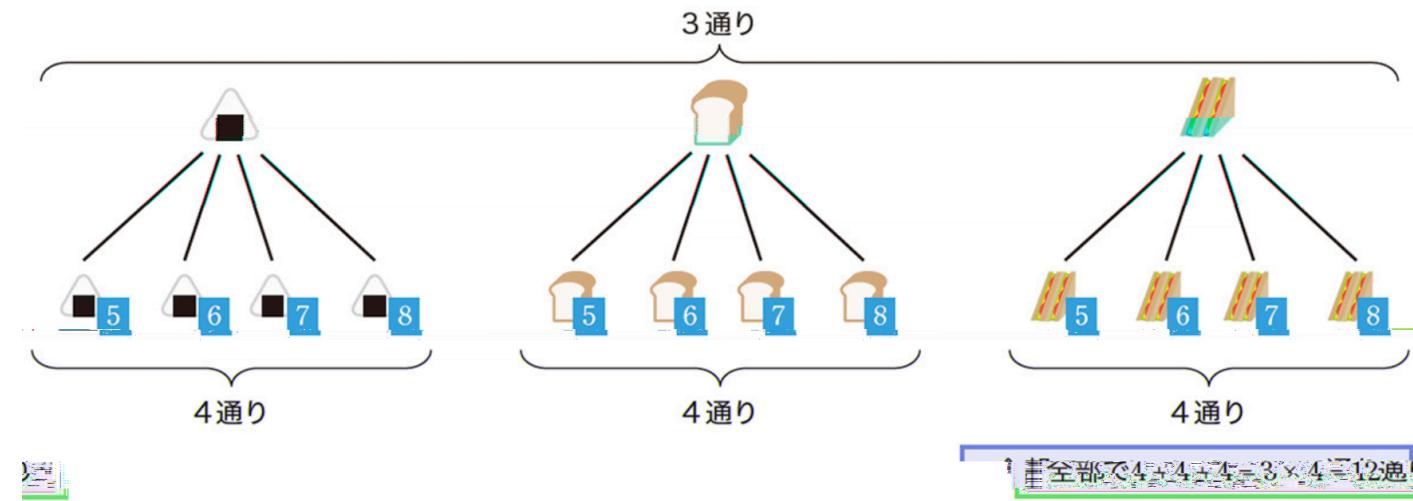
- 基本公式 1 : 積の法則
- 基本公式 2 : 積の法則の拡張
- 基本公式 3 : n 個のモノを並び替える方法の数は $n!$
- 基本公式 4 : n 個のモノから r 個を並べる方法は $_n P_r$
- 基本公式 5 : n 個のモノから r 個を選ぶ方法は $_n C_r$
- 応用例 1 : 買い物の方法の数
- 応用例 2 : 同色カードの組み合わせ
- 応用例 3 : 全探索の計算回数

基本公式 1：積の法則

事象 A の起こり方が N 通り、事象 B の起こり方が M 通りあるとき、事象 A, B の起こり方の組み合わせは全部で $NM(N \times M)$ 通りあります

- 明日の朝食は、おにぎり・食パン・サンドイッチのいずれかである
- 明日の起きる時間は、5:00, 6:00, 7:00, 8:00のいずれかである

朝食を事象 A 、起床時間を事象 B とすると $3 \times 4 = 12$ 通りとなり、これを積の法則といいます

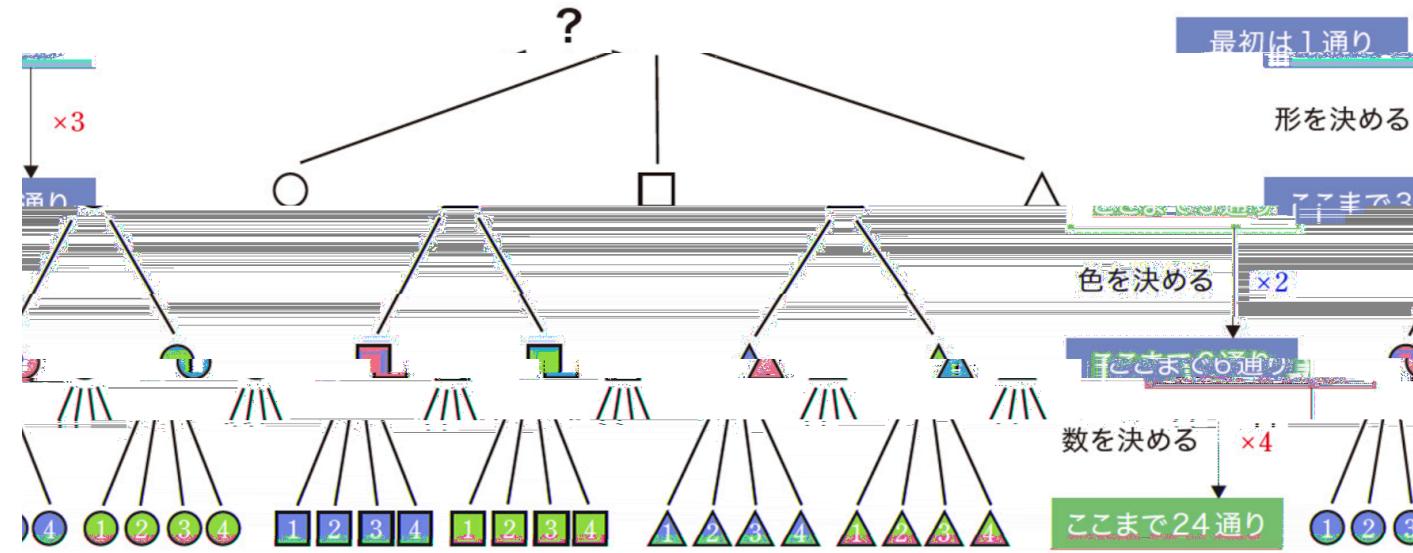


基本公式 2：積の法則の拡張

積の法則は事象が3以上の場合にも拡張することができます

- 形：円形，四角形，三角形のいずれか
- 色：赤・青のいずれか
- 記入する数字：1,2,3,4のいずれか

この場合も組み合わせの数は積の法則どおり $3 \times 2 \times 4 = 24$ 通りあります



基本公式 2：積の法則の拡張

選択肢が同じ数（例えば M 通り）の事象が N 個あるような組み合わせの数は

$$M \times M \times M \times \cdots \times M = M^N$$

通りあります

例えば、すべての要素が 1 または 2 である長さ 4 の数列 $A = (A_1, A_2, A_3, A_4)$ の個数は $2^4 = 24$ 通りです



基本公式 3： 個のモノを並び替える方法の数は

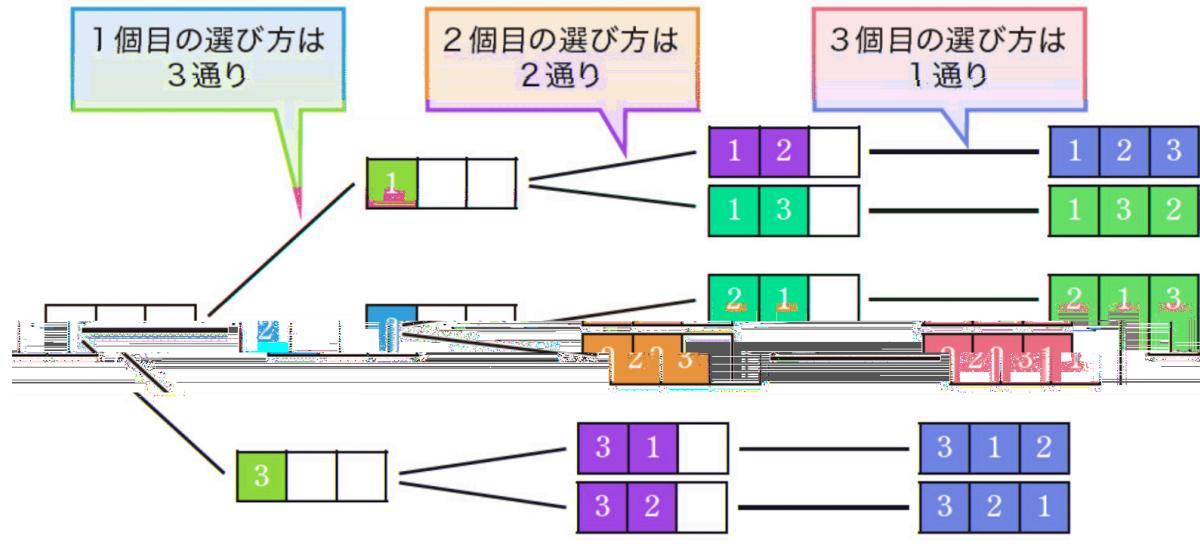
n 個のモノを並び替える方法は

$$n! = n \times (n - 1) \times (n - 2) \times \cdots \times 3 \times 2 \times 1$$

通りあります

例えば、3つの整数1, 2, 3を並び替える方法は $3! = 3 \times 2 \times 1 = 6$ 通りあります

方法が $3!$ 通りになる理由は以下の樹形図をみるとわかりやすいです



基本公式 4 : のモノから を並べる方法は

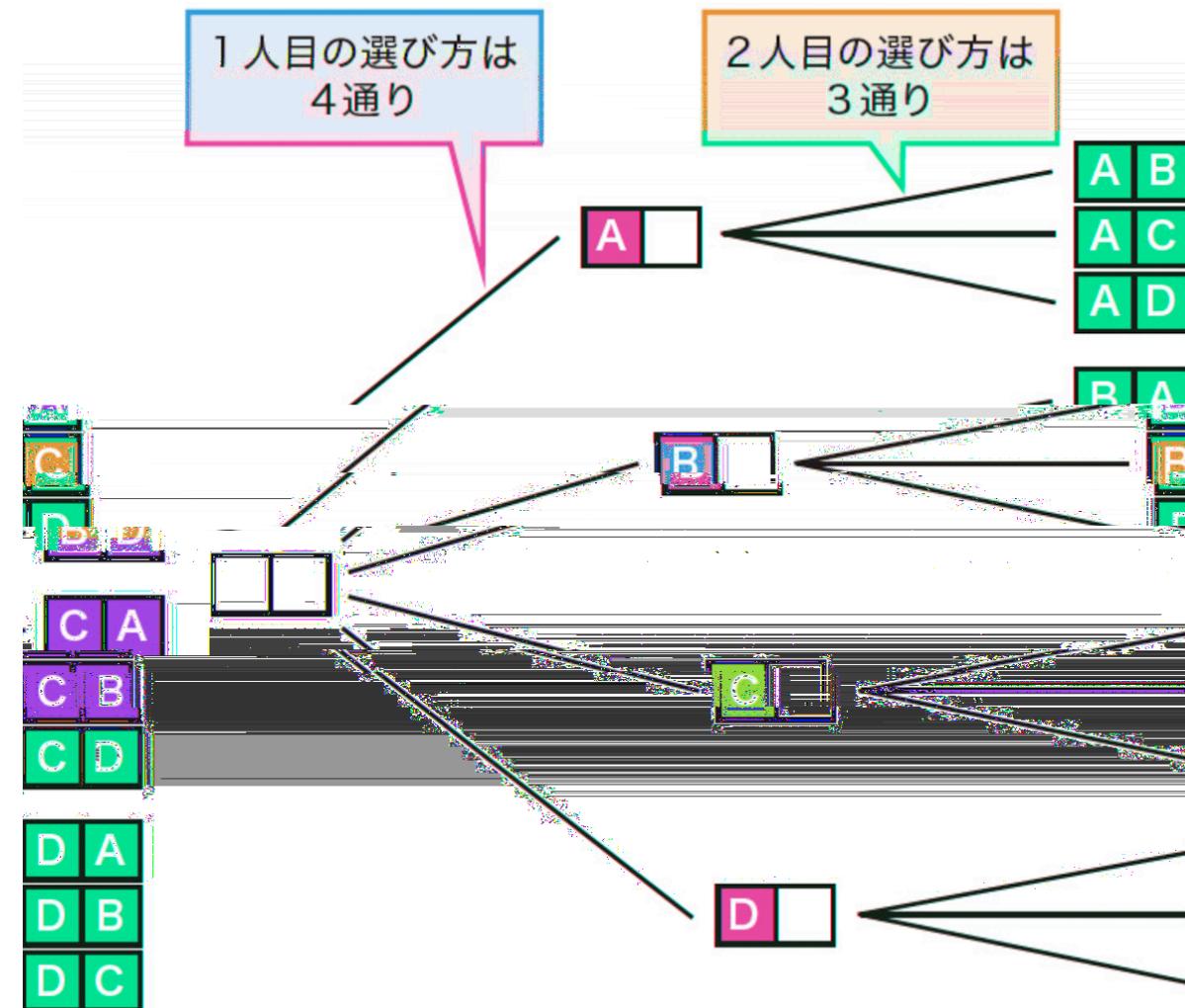
n 個のモノから r 個のモノを選び、これらを一列に並べる方法の数は以下の式で求めることができます

$${}_n P_r = \frac{n!}{(n-r)!} = n \times (n-1) \times (n-2) \times \cdots \times (n-r+1)$$

たとえば、人 A, B, C, D の中から2人を選び、並び順を決める方法は ${}_4 P_2 = 12$ 通りあります

「1人の選び方は n 通り」「2人の選び方は $n - 1$ 通り」と考えたうえで 積の法則 を適用させることで上式を導出することができます

基本公式 4 : 個のモノから 個を並べる方法は

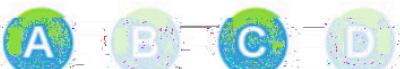


基本公式 5 :

基本公式 5 : n 個のモノから r 個を選ぶ方法は



並び順を考えると
 $A \rightarrow B / B \rightarrow A$



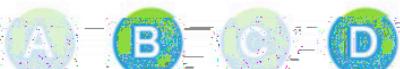
並び順を考えると
 $A \rightarrow C / C \rightarrow A$



並び順を考えると
 $A \rightarrow D / D \rightarrow A$



並び順を考えると
 $B \rightarrow C / C \rightarrow B$



並び順を考えると
 $B \rightarrow D / D \rightarrow B$



並び順を考えると
 $C \rightarrow D / D \rightarrow C$

応用例 1：買い物の方法の数

以下の問題を考えてみましょう

コンビニには $\square \blacksquare$ の品物が売られており、 $\blacksquare \blacksquare \blacksquare$ ($\square \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare$) の商品の値段は $\blacksquare \blacksquare$ 円です。異なる2つの品物を買う方法のうち、合計値段が500円となるものは何通りありますか。

制約： $\blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare$ $\blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare$ いずれか

商品の選び方を全探索する方法が思いつくと思います。しかし N 個の中から2個の品物を選ぶため、全部で $N C_2$ 通りの選び方があります。したがって計算量は $O(N^2)$ となり、効率が悪いです。

応用例①：買い物の方法の数

全探索ではなく別の方法を考えます

合計値段が500円となるような買い物の仕方を考えると、以下の2つしかないことが分かります

方法A：100円の品物を1個、400円の品物を1個買う

方法B：200円の品物を1個、300円の品物を1個買う

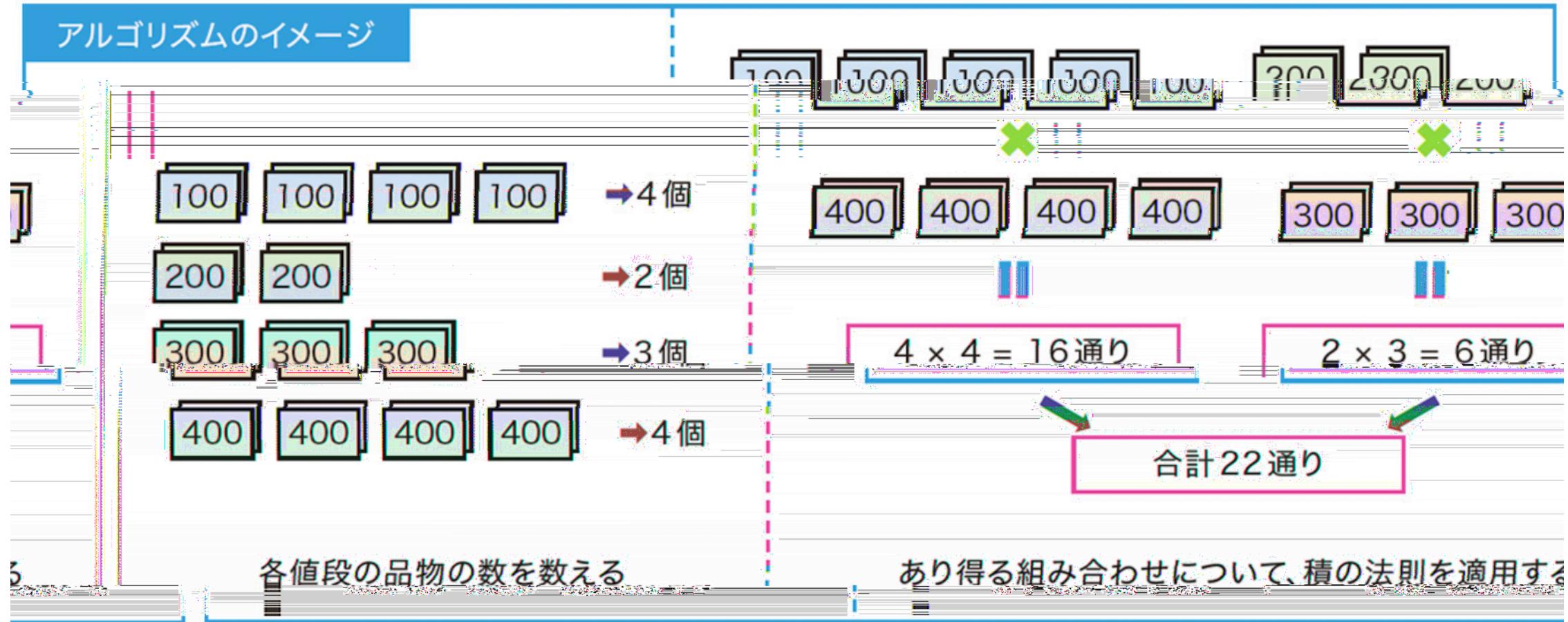
また、100円・200円・300円・400円の品物の数をそれぞれ a, b, c, d 個とするとき、積の法則より、各方法における買い方の数は次の通りです。

方法A： a 通り× b 通り = ad 通り

方法B： b 通り× c 通り = bc 通り

したがって、求める答えは $ad + bc$ となります。 a, b, c, d の値は計算量 $O(N)$ で数えられるので、 $N = 200000$ のケースでも1秒以内に答えを出すことができます

応用例 1：買い物の方法の数



応用例 2 : 同色カードの組み合わせ

カードの選び方の個数を求める問題です

□×のカードがあり、左から ××× (□××××××××) のカードの色は ××です。 ××× ×
のとき赤色、 ××× ×のとき黄色、 ××× ×のとき青色です。同じ色のカードを2枚選ぶ方法は何通りありますか。

制約： □××× ×× ××××××××× × × ×

カードの選び方を全探索する方法が思いつくでしょう。しかし2枚のカードを選ぶため、全探索アルゴリズムの計算量は $O(N^2)$ となってしまい、効率が悪いです

応用例 2：同色カードの組み合わせ

全探索ではなく別の方法を考えます

赤色・黄色・青色のカードの枚数をそれぞれ x, y, z 枚とするとき、以下のことが分かります。

- 赤色のカードを2枚選ぶ方法は $_x C_2$ 通りある
- 黄色のカードを2枚選ぶ方法は $_y C_2$ 通りある
- 青色のカードを2枚選ぶ方法は $_z C_2$ 通りある

したがって、求める答えは以下の通りになります。

$$_x C_2 + _y C_2 + _z C_2 = \frac{x(x - 1)}{2} + \frac{y(y - 1)}{2} + \frac{z(z - 1)}{2}$$

このように考えると、赤色・黄色・青色のカードの枚数をそれぞれ数えるだけで答えが分かれます。計算量は $O(N)$ であり、全探索に比べ大幅に効率が良いです

応用例 2：同色カードの組み合わせ

赤
黄
青

赤色：4枚
 $\rightarrow 4 \times 3 \div 2 = 6$ 通り

黄色：4枚
 $\rightarrow 4 \times 3 \div 2 = 6$ 通り

青色：3枚
 $\rightarrow 3 \times 2 \div 2 = 3$ 通り

合計： $6 + 6 + 3 = 15$ 通り

トの枚数を数える

公式に従って計算する

各色のカ

応用例 3：全探索の計算回数

5枚のカードを選ぶ問題です

□△のカードがあり、左から□△ (□△□△□△□△) □のカードには整数 □△が書かれています。カードを5枚選ぶ方法のうち、選んだカードに書かれた整数の和がちょうど□△となるものは何通りありますか。

制約： □△□△□△ □△□△□△□△ □△□△□△□△

この問題も全探索から考えていきましょう。今回は5枚のカードを選ぶため、全探索アルゴリズムの計算量は $O(N^5)$ です。単純計算をすると $100^5 = 10^{10}$ であるため、一見5秒以内に答えを出すように思えないかもしれません

応用例 3：全探索の計算回数

N 枚のカードから 5 枚を選ぶ方法は $_N C_5$ 通りであるため、 $N = 100$ の場合でもカードの選び方は

$${}_{100} C_5 = \frac{100 \times 99 \times 98 \times 97 \times 96}{5 \times 4 \times 3 \times 2 \times 1} = 75287520$$

通りしかありません。これは 10^9 を大幅に下回っており、5 秒以内に実行が終わると予測できます。

課題

課題

- 課題はMoodle上にあります
- 課題に書かれている問題に解答するプログラムを作成してください
- 作成したプログラムを実行して問題なく動作しているかを確認してください
- 動作確認が終わったら、プログラムファイル（`filename.java`）をMoodleに提出してください

提出期限は **11月18日(月) 20:00** まで