

Part2. アルゴリズムと計算量

アルゴリズムを工夫することで、問題の計算時間や使用メモリを大幅に削減することができる。

1. 計算量
2. すべてを調べるアルゴリズム
3. 累積和
4. 二分探索法
5. 深さ優先探索、幅優先探索
6. 動的計画法
7. 分割統治法

2-1. アルゴリズムと計算量

・ アルゴリズム

アルゴリズムとは問題を解くための手順のことである。複数桁の計算を行うときに使う「筆算」は人間が使うアルゴリズムの一種だし、「右手を壁につけて歩き続ければ迷路を脱出できる」というのも迷路を解くためのアルゴリズムの1つである。

コンピュータにアルゴリズムを指示し、実行させるのがプログラムである。アルゴリズムによって速いもの、遅いもの、メモリをたくさん使うもの、使わないものなど良し悪しがある。コンピュータは一秒間に何億回という計算速度と、何億文字という記憶領域を持っているものの、下手なアルゴリズムを使うとすぐにその能力の限界に達してしまう。

・ 計算量

アルゴリズムの速さを表す1つの尺度が計算量である。計算量は、入力サイズ n に対して、どれだけの計算時間がかかるかを O 記号を使って表す。 n に比例する時間がかかるなら $O(n)$ 、 n の二乗に比例するなら $O(n^2)$ 、 n の階乗に比例するなら $O(n!)$ といった具合である。直感的には、「プログラムの中で一番多く実行される(一番深いループの中にいる)文が何回実行されるか」を考えるといい。

(1) $O(n)$ アルゴリズムの例

- ・ n 個の数 $x[0], x[1], \dots, x[n-1]$ のなかに、ある数 a が含まれているか調べる。

```
for(int i=0; i<n; i++){  
    if(x[i]==a){  
        return 1;  
    }  
}
```

この if 文は n 回実行されるので、これは $O(n)$ アルゴリズムである。

- ・ ある数 a の n 乗を計算する。

```
float x = 1;  
for(int i=0; i<n; i++){  
    x *= a;  
}
```

これも $x*=a;$ が n 回実行されるので、 $O(n)$ である。

なお、 $O(2n)$ とか $O(10n)$ みたいな比例係数は考えないで、こういうのも全て n に比例しているので $O(n)$ と書く。

(2) $O(n^2)$ アルゴリズムの例

- ・ n 個の数 $x[0], x[1], \dots, x[n-1]$ を小さい順に並び替える。

```
for(int i=0; i<n; i++){  
    x[i]から x[n-1]までで一番小さい数を探して、x[i]と入れ替える  
}
```

for 文の内側の、一番小さい数を探す、という作業にも n に比例する時間がかかるので、結局 n^2 に比例する時間がかかることになる。

(3) $O(n!)$ アルゴリズムの例

- ・ $1, 2, \dots, n$ を並び替えてできる n 桁の整数で、ある数 a で割り切れるものはいくつあるか。

$1, 2, \dots, n$ の並び替えは $n!$ 通りあるので、全部調べれば $O(n!)$ の計算が必要である。

- ・ n 個の町をまわって、出発地点に戻ってくる道のなかで、最も移動距離の短い町をまわる順番を探す。

これも町 $1, 2, \dots, n$ をまわる場合の数は 1 から n までの並び替えの場合の数と同じなので、 $O(n!)$ の計算が必要である。

(4) $O(\log(n))$ アルゴリズムの例

$\log(x)$ とは、 2 の $\log(x)$ 乗が x になるような数である。 $\log(2)=1, \log(4)=2, \log(64)=6$ である。 n が 1000000 くらいになっても $\log(1000000)$ は 20 くらいなので、非常に高速なアルゴリズムといえる。 $O(\log n)$ と略記することも多い。

- ・ ある数 a の n 乗を計算する。

a を n 回掛け算するのでは $O(n)$ にかかってしまうのだが、実は $a^{2^x} = a^x * a^x$ という性質を使うと、例えば a^8 を求めるには a^4 が分かればよく、 a^4 は a^2 、 a^2 は a が分かればいいので、 a^2, a^4, a^8 の 3 回の計算で a が求まることが分かる。これは a が 1024 でも、 $2, 4, 8, 16, 32, 64, 128, 256, 512, 1024$ と 10 回の計算で求めることができる。これは $O(\log n)$ アルゴリズムである。

2-2. すべてを調べるアルゴリズム

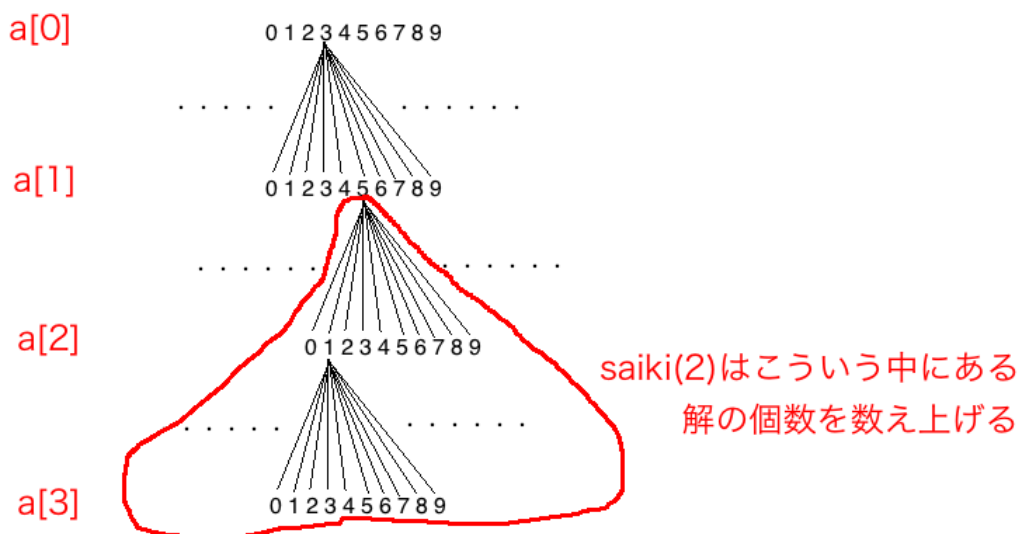
「問題を解く」というのは、「正解の候補」の中から正解を選び出す作業といえる。 $X^2 + 2X + 1 = 0$ を解け、とかいうのも、 X が-3とか0.12とか無数にある実数のなかから、この方程式をみたすものを選び出せ、と見る事が出来る。

さて、正解の候補から正解を選び出すということは、正解の候補をすべて調べあげれば正解を見つけることが出来るということである。この二次方程式のような正解の候補が無数にある場合は地球が減びても終わらないのだが、正解の候補が少ない時には候補をぜんぶ列挙して正解かどうかひとつひとつ確かめるというのが最も基本のアルゴリズムとなる。

(1) A0J 0008 (問題セット→Volume 0→8 番) を解け

4つの0から9までの数の和で合計が n となるものの数を答えよということなので、0から9までの4つの数の組すべてについて、合計が n となるかどうか確認し、数をカウントすればよさそう。候補の数は $10 \times 10 \times 10 \times 10 = 10000$ なので、一秒に一千万計算できるコンピュータには余裕だ。



これは4重ループでふつうに解けるのだが、その解き方はPart1-6の問題でもうやったはずなので、ここでは再帰呼び出しで解いてみよう。右のページの空欄を埋めるといい。



この問題では入力データがいくつあるのか与えられていない。（一行目でその個数が与えられることも多いのだが）

そういうときは、この33行目にあるような `if(cin>>n)` のように書くテクニックが便利で、`cin>>n` は実は `n` に入力を入れるだけでなく、入力が成功したか失敗したかを返していて、入力データが来なかったら失敗となる。失敗の場合はこの `if` 文が `else` にまわり、そのときは `break` して無限ループを終わらせる。

```

1  #include <iostream>
2  using namespace std;
3
4  int n;
5  int a[4];
6
7  // 問題文のa,b,c,dをa[0],a[1],a[2],a[3]とする
8  // kが4なら、a[0]+a[1]+a[2]+a[3]==nかどうかを調べて
9  // 満たしていたら1を返す。そうでなければ0。
10 // もしkが3以下ならa[k]を0から9まで調べてその都度saiki(k+1)を呼び出す。
11 // それらから返ってくる値の合計が、
12 // a[k-1]までを固定し、a[k]以降を0から9まで動かした時の
13 // 場合の数となるので、これをreturnする。
14 int saiki(int k){
15     if(k==4){
16         
17
18
19
20     }else{
21         int sum = 0;
22         
23
24
25
26         return sum;
27     }
28 }
29
30
31 int main(){
32     for(;;){
33         if(cin >> n){ // 入力が終わったらelse側にまわる
34             cout << saiki(0) << endl;
35         }else{
36             break;
37         }
38     }
39
40     return 0;
41 }

```

(2) A0J 1045 “Split up!” を解こう

Part1-9 にものせたので、そっちで解いたひとは飛ばしていい。

0 がグループ A、1 がグループ B とする

配列 $g[n]$ を用意して、

0 0 0	全員グループ A
0 0 1	3 人目だけグループ B
0 1 0	2 人目だけグループ B
...	

のようにしてすべての場合を試す事を考える。

今 n は最大 20 なので、

0 0 0 ... 0 (20 個)

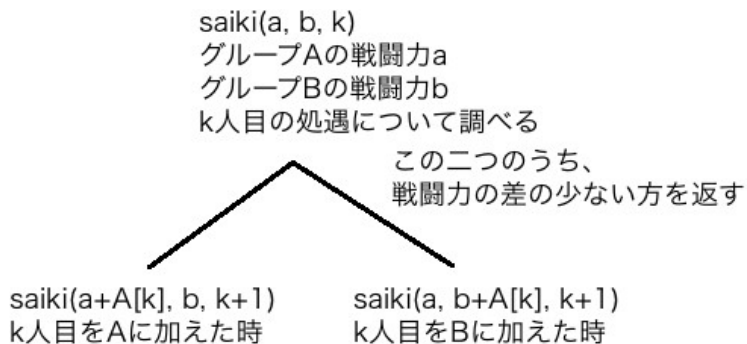
から

1 1 1 ... 1

までの場合の数は $2 \times 2 \times \dots \times 2$ (20 個の積) $\div 1,000,000$ (百万) 通りある。複数データセットあることを考えると、秒速一千万計算のコンピュータでぎりぎり終わるくらいだろうか。

再帰呼び出しは次の図のイメージでやるとよいだろう。

再帰の最下層では、引数 a と b の差の絶対値を return すればいい。



K 会夏期講習 2013 情報講座

こんな感じで、すべてを調べるアルゴリズムは、計算量 m^n とか、 $n!$ とかになることが多い。そしてこれらの値は、 m が 2 でも $n \leq 20$ くらいまでしか扱えず、 $n!$ なら $n \leq 10$ 程度である。

n がそれより大きい問題では、すべてを調べるんじゃなくて、工夫して調べる候補を減らすんだな、と考えたほうがいいだろう。

(メモ用)

2-3. 累積和

A0J 0516 (問題セット→Volume 5→16 番) を開こう。

これは 2006 年度の情報オリンピック本選 (ちょうど僕が受けていた頃) の問題だ。

問題文の入出力例が混乱しているので、ここに正しいのを載せる
複数のデータセットがあって、0 0 が来たら終了だ。

入力は、無限ループで各周回の最初に n, k を受け取って、0, 0 だったら break で脱出すればいい。

Sample Input	Sample Output
① 5 3 2 5 -4 10 3	11 4 11
② 3 1 3 1 4	
③ 4 2 1 5 9 2 0 0	

素直に考えると、全ての連続する k 個の和を調べて、そのなかで一番大きいもの

$n=12, k=6$

```

3 1 4 1 5 9 2 6 5 3 5 8
23
28

```

を探せば良い。連続する k 個の数の和をとるには $O(k)$ の計算量がかかる。それが n 箇所あるので、この場合の計算量は $O(nk)$ だ。

さて、 n, k の最大値を見ると、

正整数 $n (1 \leq n \leq 100000)$ 正整数 $k (1 \leq k \leq n)$

らしい。つまり $nk \leq 10,000,000,000$ 百億は 1 秒では終わらない！

K 会夏期講習 2013 情報講座

さて、それではどうすればいいか考えてみよう。

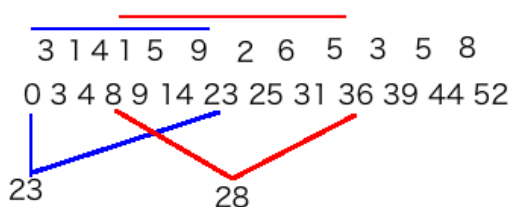
10 分くらい考えて分かんなかったら次のページをめくろう。

この問題は、累積和と呼ばれるアルゴリズムで解くことができる。

3 1 4 1 5 9 2 6 5 3 5 8
0 3 4 8 9 14 23 25 31 36 39 44 52

この2行目は、1行目の数列に対してある操作をしたものである。それがなにか分かるだろうか？

このような数列を用意すると、



$O(k)$ がかかっていた連続する k 個の和を求める処理が、その両側にある2つの数を引き算するだけで終わらせることができる！

つまり、 $O(nk)$ だった計算量は $O(n)$ まで落ち、無事制限時間内に終わらせられるのである。

※二次元累積和

累積和は次のような性質であるが、

累積和の各数は、それより手前の数の総計を表している

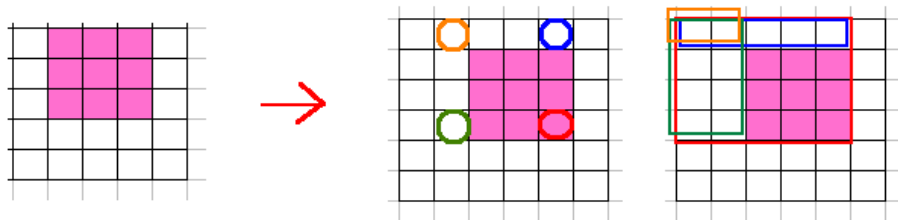
3 1 4 1 5 9 2 6 5 3 5 8
0 3 4 8 9 14 23 25 31 36 39 44 52

これを二次元に拡張することも出来る。



各数は、それより左上側の長方形領域の数の合計を表す

こうすると、二次元配列上の長方形領域の数の合計が、その二次元累積和配列内の4つの数を足したり引いたりするだけで求めることができる。



この図では、**ピンクの領域の合計 = 赤 青 緑 + オレンジ** である。

これを利用するのが

・ A0J 0560 “Planetary Exploration”

なので、ぜひ解いてみよう。情報オリンピックではなぜか良く出てくるアルゴリズムなのである。

2-4. 二分探索法

n 個の要素の数列の中に、ある数があるかどうか調べる方法を考える。数列を左から順番に調べていくと、最悪 n 回、平均 $n/2$ 回比較を行わないといけないので、 $O(n)$ かかる。

ここで、数列が小さい順に並んでいたらどうだろうか。「数列のまんなかを調べて、目的の数よりも小さければ、次に右側の領域を、大きければ左側の領域を調べる」という操作を再帰的に繰り返すことで $O(\log n)$ まで高速化することができるのである。

数列

1	2	5	7	8	11	13	16	20	21
---	---	---	---	---	----	----	----	----	----

 のなかに、20 があるか調べる

手順と、10 があるか調べる手順について、 $O(n)$ アルゴリズムと二分法を比較してみよう。

・ 左から順に調べていく $O(n)$ アルゴリズム

1	2	5	7	8	11	13	16	20	21
↑	↑	↑	↑	↑	↑	↑	↑	↑	
1	2	3	4	5	6	7	8	9	

20 を見つけたところで終了

1	2	5	7	8	11	13	16	20	21
↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
1	2	3	4	5	6	7	8	9	10

10 は見つからなかった

・ 二分法 $O(\log n)$

1	2	5	7	8	11	13	16	20	21
				↑				↑	↑
				1				2	3

1. $x[0]$ から $x[9]$ までのどこかにある
 2. $x[5]$ から $x[0]$ までのどこかにある
 3. $x[8]$ から $x[9]$ までのどこかにある
- 20 を見つけたので終了

1	2	5	7	8	11	13	16	20	21
				↑	↑			↑	
				1	3			2	

1. $x[0]$ から $x[9]$ までのどこかにある
 2. $x[5]$ から $x[0]$ までのどこかにある
 3. $x[5]$ から $x[6]$ までのどこかにある
- どこにもないので終了

二分法では、調べる領域が n 個 $\rightarrow n/2$ 個 $\rightarrow n/4$ 個... と減っていくので、 n が 1024 でも 10 回の比較で探索を終わらせることができる。従って、このアルゴリズムの計算量は $O(\log n)$ である。

(1) $O(n)$ アルゴリズムの実装

AOJ Lesson, Algorithms and Data Structures の、4-Search の A “Search I” を開こう。

二つ目の数列 T (要素数 $q \leq 500$) のなかで、一つ目の数列 S (要素数 $n \leq 10000$) 含まれているものの個数を調べるだけである。なにも難しいことはない。

この問題では数列 S が小さい順に並んでいるわけではないので、 $O(n)$ の探索 (左端から全て調べる) アルゴリズムしかつかえない。

数列 T の各要素について、数列 S 上で $O(n)$ 探索を行うので、計算量は $O(qn)$ である。 $qn \leq 5,000,000$ なので、まあぎりぎり間に合いそうだとわかる。

(2) 二分探索の実装

これが解けたら、この次の問題 B “Search II” を開こう。前と違うのは、 **S が小さい順に並んでいること**と、 $n \leq 100000$, $q \leq 50000$ となったことである。 $O(qn)$ のアルゴリズムではどうみても間に合わないが、試しに(1)と同じプログラムで送信してみよう。Time Limit Exceeded と出るはずだ。

さて、この問題は S が小さい順に並んでいるので二分探索をすることができる。

二分法命令 `int bsearch(int l, int r, int t)` を作ろう。この命令は、 $S[l]$, $S[l+1]$, ..., $S[r-2]$, $S[r-1]$ のなかに、 t が含まれているかを調べる命令である。探索領域が $S[r]$ まででなく、 $S[r-1]$ までであるところがコツである。次の方針で実装する。

- (i) $r-l$ が 1 以下のとき、探索領域が残り 1 つ以下になっている。 $S[l]$ が t と等しければ、見つかっているので l を返す。等しくなかったら、見つからなかったということなので、 -1 を返す。
- (ii) $S[l]$ から $S[r-1]$ までのなかの、まんなかの数と t を比較したい。まず、変数 mid を用意して、ここに $(l+r)/2$ を入れる。
- (iii) $S[mid]$ が t より大きければ、 t は $S[l]$ から $S[mid-1]$ までの間にあるはずなので、`bsearch(l, mid-1, t)` を再帰呼び出しする。
そうでなければ、 t は $S[mid]$ から $S[r-1]$ までにあるので、`bsearch(mid, r, t)` を呼び出す。
- (iv) 呼び出した結果を `return` する。

(3) 二分法を使うと、ルートを求める計算が高速でできる。
 ルートを求めるというのは、 $y=x*x$ をみたす x を探すということである。 x を実数とすると解の候補は無限にあってしまうのだが、小数点以下 3 桁の精度で探す、とすれば、ルート 2 を求めたいとすれば

0.001	→ x^2 は 0.000001	2 より小さい
0.002	0.000004	
0.003	0.000009	
...		
1.414	1.999396	2 より小さい
1.415	2.002225	2 を超えた！ここが答えだ！

と探していけば、答えにたどり着ける。これは $O(n)$ 探索に相当する。

さて、ここで注目して欲しいのが、 x を 0.001, 0.002 ... と増やしていった時、 x^2 も 0.000001, 0.000004, ... と 小さい順に増えていく ということである。つまり、 x^2 を小さい順に並んだ数列だと思って、二分探索を行うことが出来る！

次のような再帰命令 `double root(double l, double r, double t)` を書け。

- ・ この命令は、 l 以上 r 未満の小数 x のなかで、 $x*x = t$ をみたす x を返す
- ・ さっきと同じように、 $x = (l+r)/2$ として、 $x*x > t$ なら左半分の領域で、そうでなければ右半分の領域で再帰呼び出しする。
- ・ r と l の差が 0.00001 未満になったら、そのときの r を正解として return する。

`#include <cmath>` を書くことで使える命令である `sqrt(x)` の結果と比較して答え合わせせよ。

(4) 二分法を使って $x^3 + 2x^2 + 3x + 4 = 5$ を解け。(0.275682...になるはず)

※このように二分法は方程式を解くのに便利だが、単調増加か単調減少（グラフが右上がりか右下がり）でないと使えないので注意。

(5) 二分法の応用

・ A0J 0181 “Persistence” を解こう

本を全部しまえる、本棚の最小の横幅を求めたいらしい。一見なにが二分探索なのかわからない。

ここで、次のように考えよう。

「本棚の横幅 w が与えられた時、本が全部入りきるかどうか答えよ」という質問を考える。これは w を 1 から順番に増やしていくと、

無理 無理 無理 ... 無理 OK OK OK OK ...

となるはずである。この初めて OK になる w が、元々の問題の答えである。なんかさっきのルートを求める問題と似てないだろうか？ w を 1 から順番に増やしていくのは $O(n)$ 探索だが、これは二分探索が使える問題である！

2-5. 深さ優先探索、幅優先探索

大事なんだけどこの講習では時間がないので省略する。
次のような問題で練習してみるといいかも

深さ優先探索

A0J 0207

“8 クイーン問題”

幅優先探索

A0J 1130

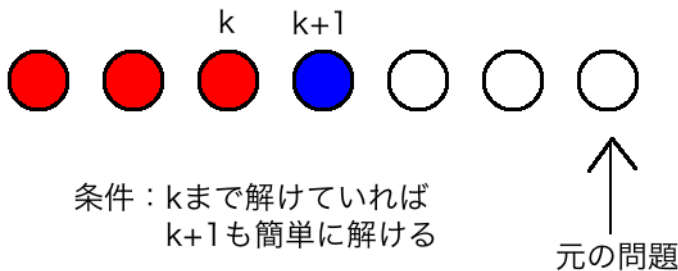
A0J 1046

A0J 0179

2-6. 動的計画法

この動的計画法と、次の分割統治法は、どちらも元の問題をより簡単に解ける小さな問題に分けて、それを利用して元の問題を解くという方針である。

・動的計画法のイメージ



このようなときに、 $k=1$ から順番に答えを求めていくのが動的計画法である。数学的帰納法とか、漸化式とかと関連が深い。

動的計画法はあらゆるプログラミングコンテストで、毎回必ず1問は出てくるくらい有名なアルゴリズムである。使えるようになっておいて損はない。「動的計画法」は長いので、以下 DP (Dynamic Programming) と呼ぶことにした。

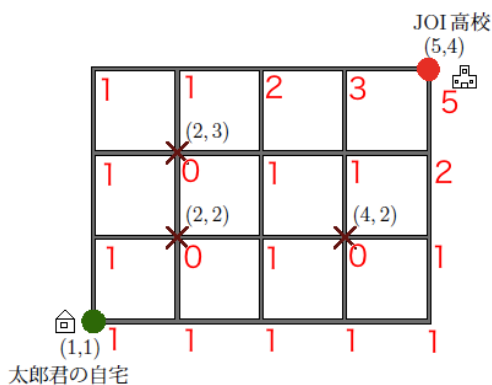
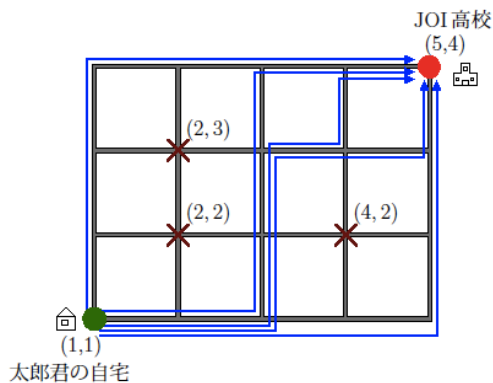
(1) A0J Lesson, Algorithms and Data Structures, 10-Dynamic Programming-A “Fibonacci Number” を解こう

問題文の定義のとおり再帰関数 $\text{fib}(n)$ を作ると、 $\text{fib}(n)$ は $\text{fib}(n-1)$ と $\text{fib}(n-2)$ という二つの再帰呼び出しを行うので、 $n=44$ ともなると 2^{44} とまではいかないもののそのくらい無茶な回数の計算を行ってしまう。そして TLE (Time Limit Exceeded)。

しかし逆に考えると、 $\text{fib}(n-1)$ と $\text{fib}(n-2)$ が分かっていたら $\text{fib}(n)$ は足すだけで求まるので、配列を作って $\text{fib}(0)$ から順番に埋めていけば $O(n)$ で $\text{fib}(n)$ を求めることが出来る。

(2) A0J 0515 “School Road” を解こう

これは情報オリンピックの予選問題だが、中学入試をした人とかは見たことがある問題かもしれない。さきほどのフィボナッチが一次元の DP だとすると、これは二次元の DP といえる。



右の赤い数字の意味と求め方が分ければ、もう解けるだろう。二次元配列を作って、それを順番に決定していく。「表を埋めていく」というのが動的計画法のイメージである。

(3) A0J 0528 “Common substring” と解こう

入力が複数データセットであることに注意（問題文を最後まで読むべし）
これもまた `if(cin >> s >> t)` のようにして入力の終わりを検知する。

次の表の意味が分かれば解ける。これも二次元の DP だ。

	E	C	A	D	A	D	A	B	R	B	C	R	D	A	R	A
A	0	0	1	0	1	0	1	0	0	0	0	0	0	1	0	1
B	0	0	0	0	0	0	0	2	0	1	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	3	0	0	1	0	0	1	0
A	0	0	1	0	1	0	1	0	0	0	0	0	0	1	0	2
C	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
A	0	0	2	0	1	0	1	0	0	0	0	0	0	1	0	1
D	0	0	0	3	0	2	0	0	0	0	0	0	1	0	0	0
A	0	0	1	0	4	0	3	0	0	0	0	0	0	2	0	1
B	0	0	0	0	0	0	0	4	0	1	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	5	0	0	1	0	0	1	0
A	0	0	1	0	1	0	1	0	0	0	0	0	0	1	0	2

K 会夏期講習 2013 情報講座

(4) 工夫のいる DP たち

次のページにヒントあり。自信のあるひとは見ないで解こう。

・ A0J 0157 “Russian Dolls”

・ A0J 0202 “At Boss’ s Expense”

※素数を求める必要あり。「エラトステネスのふるい」がわからないひとはまずこれを調べて素数リストを作れるようにせよ。

K 会夏期講習 2013 情報講座

・ Russian Dolls

両方をあわせて h の順でソートする
一次元 DP

h 順ソートで k 番目に小さい人形を一番外側とした時、そこに入れられる人形の最大数を $dp[k]$ とする。

$dp[k]$ は $dp[k-1]$ までが求まっていたら、 $dp[0]$ から $dp[k-1]$ までで、 k 番目の人形に入るものの値+1 のなかで最も大きいもの、として求められる。

・ At Boss's Expense

1 次元 DP

料理の値段が $A[0], A[1], \dots, A[n]$ とする

料理を組み合わせて価格 p が作れるかどうかは

$$p-A[0], p-A[1], \dots, p-A[n]$$

が作れるなら作ることが出来る。

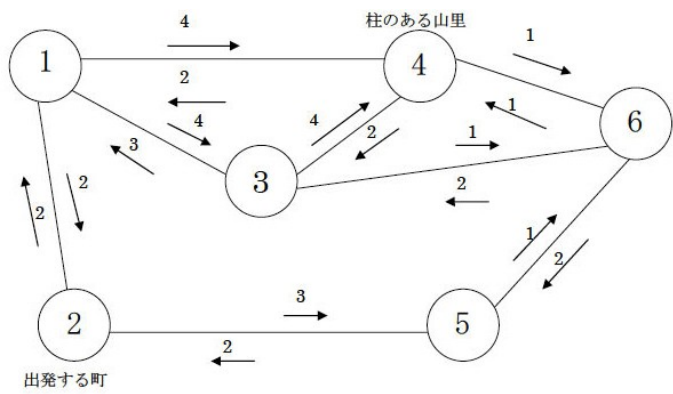
そして、価格の合計の最大値は 1000000 なので、この要素数を持った配列を用意して、1 円から x 円まで、それが作れるかどうか順に調べてこの配列を埋めていけばいい。

最後に、作れる価格の中で、最大の素数を探す。

(5) ワーシャル - フロイド法

ワーシャルフロイド法はDPの応用で、最短経路を探す簡潔な手法である。一度理解してしまえば10行くらいで最短経路を求めることが出来てしまうという恐ろしい手法である。

最短経路問題とは
A0J 0117 “A reward for a Carpenter”を開こう。これは最短経路問題の典型的な問題である。



町同士がこのような通行料で繋がれているらしい。これは二次元配列を使って、次のように表すことが出来る。

		to					
		1	2	3	4	5	6
from	1	0	2	4	4	#	#
	2	2	0	#	#	3	#
	3	3	#	0	4	#	1
	4	2	#	2	0	#	1
	5	#	2	#	#	0	1
	6	#	#	2	1	2	0

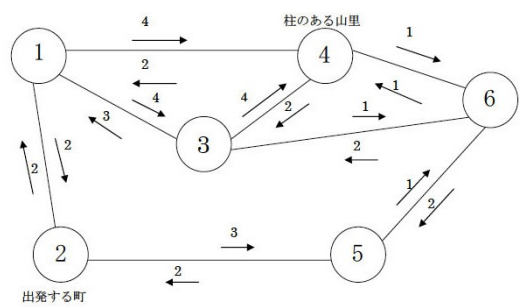
たとえば町3から町4への料金「4」は、この表では3行目の4列目となる。#は町から町に道がないことを表す。プログラム上では非常に大きい距離(1001001001とか)で繋がれているとすると便利だったりする。

この表の名前を dst とする。

さて、この表は町 a から町 b まで一本の道を使うといくらかかるかを表しているのだが、問題では複数本の道を経由して最安値となる経路を考えたい。

そこで、まず町 1 を経由するルートを考えてみる。

- 2 → 1 → 2 cost 4
- 2 → 1 → 3 cost 6
- 2 → 1 → 4 cost 6
- 3 → 1 → 2 cost 5
- 3 → 1 → 3 cost 7
- 3 → 1 → 4 cost 7
- 4 → 1 → 2 cost 4
- 4 → 1 → 3 cost 6
- 4 → 1 → 4 cost 6



経路 a → 1 → b の値段は、
dst[a][1] + dst[1][b]で求められる。
ここで、もし dst[a][1] + dst[1][b]のほうが元々あった経路 dst[a][b]よりも安かったら、町 a から町 b にもっと安く行けるということから、

```
if(dst[a][b] > dst[a][1] + dst[1][b]){
    dst[a][b] = dst[a][1] + dst[1][b];
}
```

このように dst[a][b]の値を更新する。a から b に既存の道よりも安い新たな道ができたと考えてもいい。元々 a, b 間に道がないときは、dst[a][b]にはとても大きな値が入っているのでこの if 文は必ず true になり、dst[a][b]は更新される。

次のようにしてすべての a, b 対について調べ上げると次の表になる。

```
for(int i=1; i<=6; i++){
    for(int j=1; j<=6; j++){
        if(dst[i][1]+dst[1][j] < dst[i][j]){
            dst[i][j] = dst[i][1]+dst[1][j];
        }
    }
}
```

左が更新前、右が更新後である。赤字の部分に変化している。

	1	2	3	4	5	6
1	0	2	4	4	#	#
2	2	0	#	#	3	#
3	3	#	0	4	#	1
4	2	#	2	0	#	1
5	#	2	#	#	0	1
6	#	#	2	1	2	0

	1	2	3	4	5	6
1	0	2	4	4	#	#
2	2	0	6	6	3	#
3	3	5	0	4	#	1
4	2	4	2	0	#	1
5	#	2	#	#	0	1
6	#	#	2	1	2	0

次に、町 2 を経由するルートを考える。ただし、元の路線図ではなく、先ほど更新して新しい道が追加された路線図を使う。

```
for(int i=1; i<=6; i++){
    for(int j=1; j<=6; j++){
        if(dst[i][2]+dst[2][j] < dst[i][j]){
            dst[i][j] = dst[i][2]+dst[2][j];
        }
    }
}
```

	1	2	3	4	5	6
1	0	2	4	4	#	#
2	2	0	6	6	3	#
3	3	5	0	4	#	1
4	2	4	2	0	#	1
5	#	2	#	#	0	1
6	#	#	2	1	2	0

	1	2	3	4	5	6
1	0	2	4	4	5	#
2	2	0	6	6	3	#
3	3	5	0	4	8	1
4	2	4	2	0	7	1
5	4	2	8	8	0	1
6	#	#	2	1	2	0

更新が行われるのは色の付いている 6 箇所である。

ここで、青い字になっている

3 → 2 → 5

をみると、元々の路線図では 3 から 2 への道が存在しないものの、この前にやった町 1 を経由するルートの更新操作により新たな道 3 → 2 ができたため、3 → 2 → 5 が通行可能になっていることがわかる。つまりこの 3 → 2 → 5 は

3 → 1 → 2 → 5

なのだ！ほかの青字のところも同様のことが起こっている。

つまり、町 1 を経由するルートの更新を終えた後に町 2 を経由するルートの更新を行うと、町 1 と 2 を両方経由するルート全ての最安値が調べられたことになる。これでどうして DP なのか分かっただろう。町 1 から k-1 までを経由するルートの更新ができているとき、上記の操作（すべての (a, b) 対について a → k → b のルートのコストを求め、安かったら更新する）を行うことで、町 k までを経由するルートの最安値が求められるのだ。k=1 から順番に n まで進めば、元の路線図上の全ての出発地、目的地ペアについてのルートの最安値が求められたことになる。

プログラムは次のような三重ループで書ける。3次元配列を使わなくていいのは、低いkのときに求めたデータを大きいkのときに再利用したりしないため、同じ表の上に上書きしていけばできるからである。

```
for(int k=1; k<=n; k++){
    for(int i=1; i<=n; i++){
        for(int j=1; j<=n; j++){
            if(dst[i][k]+dst[k][j] < dst[i][j]){
                dst[i][j] = dst[i][k]+dst[k][j];
            }
        }
    }
}
```

計算量は $O(n^3)$ であり、この問題は $n \leq 20$ で $20^3=8000$ なので余裕で間に合うだろう。

それではこのアルゴリズム（ワーシャルフロイド法）を使ってこの問題を解こう。

いくつか注意点がある。

1. 「1, 2, 3, 4」のような入力データを int 型の変数 a, b, c, d に入りたい時、cin を使うと空白や改行区切りになっているカタマリしか読んでくれないので、「1, 2, 3, 4」で1カタマリだと思われてしまつてうまく動作しない。そこでC言語入力関数 scanf を使う。
まずはプログラムの先頭で

```
#include <cstdio>
```

と書く。
これを入力したい所では

```
scanf( "%d, %d, %d, %d", &a, &b, &c, &d);
```

と書く。
scanf 関数の解説をするのはとても長くなってしまうのでこのテキストでは省く。「苦しんで覚えるC言語 <http://9cguides.appspot.com/>」の「キーボードからの入力」とかをみるといいんじゃないだろうか。
2. この問題では町の番号は1から始まるが、配列はa[0]から始まるので、配列の要素数は一つ多めに確保する必要がある。
3. ワーシャルフロイド法を終えた後の配列 dst[a][b]には町aから町bへの最安コストが入っていることになる。どうがんばってもaからbにいけないときには#が残っているはず（ただしこの問題ではそういうデータはなさそう）

K 会夏期講習 2013 情報講座

・ A0J 0144 “Packet Transportation” を解いてみよう。
入力形式が面倒ですね。

次のページにヒントあり

・ A0J 0526 “Boat Travel” を解いてみよう。
J01（情報オリンピック）予選の問題。
新しいルートができるたびにワーシャルフロイドで最短路を更新するのは間に合うだろうか？
なにか工夫が必要だろうか？

次のページにヒントあり

・ A0J 0144 “Packet Transportation” を解いてみよう。

```

1  #include <iostream>
2  using namespace std;
3
4  int n;
5  int dst[110][110];
6  int M = 1001001001; // de kai kazu
7
8  int main(){
9      cin >> n;
10     for(int i=0; i<=n; i++){
11         for(int j=0; j<=n; j++){
12             dst[i][j] = M;
13             if(i==j) dst[i][j] = 0;
14         }
15     }
16     for(int i=0; i<n; i++){
17         int r, k;
18         cin >> r >> k;
19         for(int j=0; j<k; j++){
20             int t;
21             cin >> t;
22             dst[r][t] = 1;
23         }
24     }
25
26
27
28
29
30
31
32
33
34
35     int p;
36     cin >> p;
37     for(int i=0; i<p; i++){
38         int s, d, v;
39         cin >> s >> d >> v;
40         if(dst[s][d] < v){
41             cout << dst[s][d]+1 << endl;
42         }else{
43             cout << "NA" << endl;
44         }
45     }
46
47     return 0;
48 }

```

・ A0J 0526 “Boat Travel” を解いてみよう。

毎回ワーシャルフロイドだと、毎回 $O(n^3)$ の計算を行うことになり間に合わないが、ワーシャルフロイド的発想を用いることは出来る。

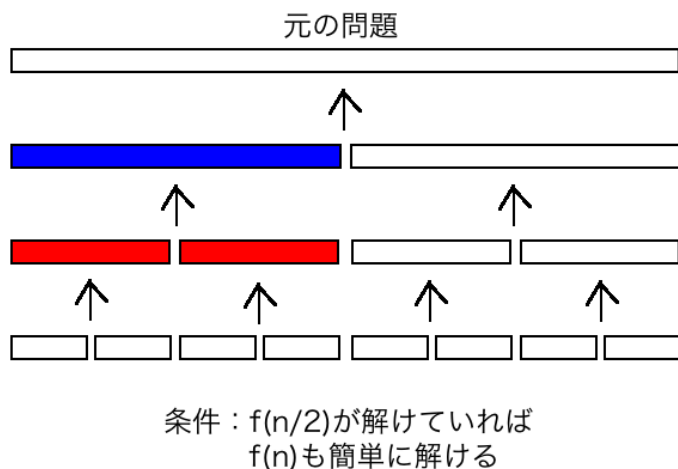
それは、

 a と b の間にコスト c のルートが出来た
 という情報が来たら、

 すべての i, j の組について、i→a→b→j というルートと i→b→a→j という
 ルートを考え、既存の dst[i][j] より安かったら更新
 というようにすれば毎回の計算量が $O(n^2)$ となり間に合う。

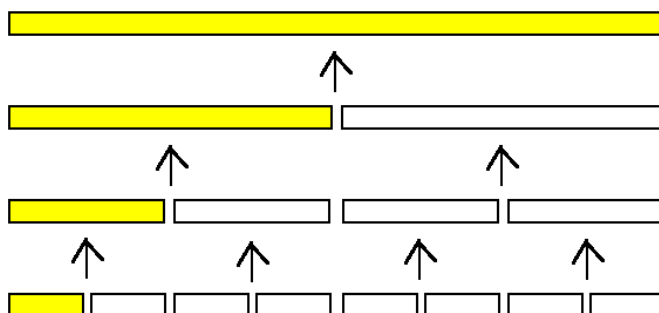
2-7. 分割統治法

DP が表を端から順番に埋めていくのに対し、分割統治法は、二つずつに分けて、それを統合していくイメージである。



といってもこれも問題を小問題に分割する解き方なので、DP と呼ぶ人もいる。計算量は、よく $n \log(n)$ になる。 $n^2 \log(n)$ とかもまあよくある。

問題によっては端の一つだけについて解けば、右側は端と同じなので計算しなくてよかったりすることもある。



たとえば、 a^n を求める問題では

$a, a^2, a^4, a^8, a^{16}, \dots$

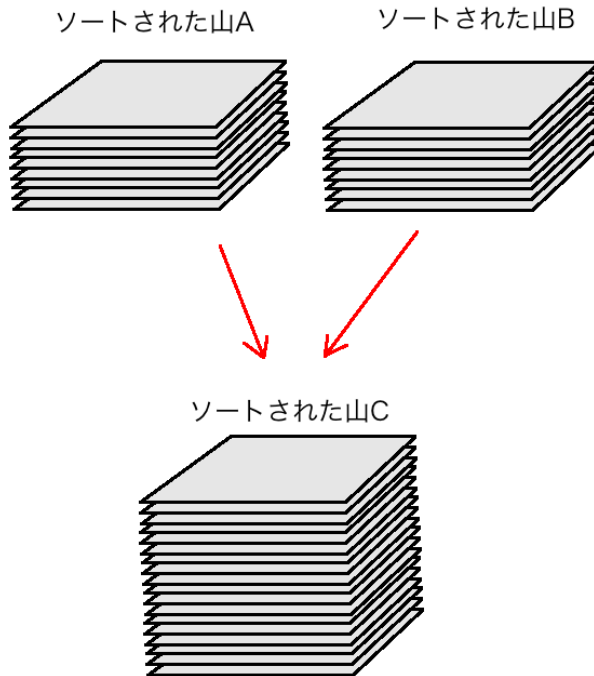
が求まればそれを組み合わせて a^n を作ることが出来る。

このときは計算量は非常に少なく $\log(n)$ になることが多い。

・ マージソート

バラバラに並んだ数を小さい順に並び替えることを「ソート」というが、ここではソートを行う高速なアルゴリズムである「マージソート」を学ぶ。この計算量は $O(n \log n)$ であり、二重ループによるソートの $O(n^2)$ と比較すると非常に速い。

まず、何か数字が書かれたカードがたくさんあって、それを並び替える事を考え



る。次の図のようにソートされた二つのカードの山があるとき、それをくっつけてソートされた山Cを作るには、

山Aと山Bの一番上（それぞれの山の中では一番小さいカード）を比較して、小さい方を山Cの一番上に持ってくる

という操作を繰り返せばよさそうである。

n 枚のカード全体をソートしたいときには、まずそれを二つの山にわけ、それぞれをさらに二つにわけ…とやって、上の操作で1枚と1枚から2枚を、2枚と2枚から4枚を…と統合していく。

・ A0J Lesson Algorithms and Data Structures → 5. Recursion and Divide and Conquer → B. Merge Sortを解こう

ここにはマージソートの擬似コード（プログラミング言語に近いが、もう少し人間に読みやすいようにデフォルメして書かれたもの（逆に空気の読めないコンピュータには読みづらい））が書かれており、その通りに実装せよとのことである。

問題文の「比較回数を数えろ」というのは、次の赤四角で囲った if 文の実行回数をカウントするということである。

```

Merge(A, left, mid, right)
  n1 = mid - left;
  n2 = right - mid;
  create array L[0...n1], R[0...n2]
  for i = 0 to n1-1
    do L[i] = A[left + i]
  for i = 0 to n2-1
    do R[i] = A[mid + i]
  L[n1] = SENTINEL
  R[n2] = SENTINEL
  i = 0;
  j = 0;
  for k = left to right-1
    if L[i] <= R[j]
      then A[k] = L[i]
        i = i + 1
      else A[k] = R[j]
        j = j + 1

Merge-Sort(A, left, right){
  if left+1 < right
    then mid = (left + right)/2;
    call Merge-Sort(A, left, mid)
    call Merge-Sort(A, mid, right)
    call Merge(A, left, mid, right)

```

配列Aはグローバル変数とする

Merge(left, mid, right)命令

配列AのA[left]からA[mid-1]までと、
A[mid]からA[right-1]までがソートされている時、
A[left]からA[right-1]までが
ソートされているようにする

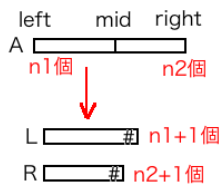
Merge-Sort(left, right)命令

A[left]からA[right-1]までに2つ以上の要素があったら、
leftとrightの中間で二つにわけ、
両者について再帰呼び出しする。
両者のソートが終わったら、
それらをMerge命令でマージして
A[left]からA[right-1]までをソートされた状態にする



Merge 命令の中身

① 1-9行目



Aをmidで分けて
二つの配列L, Rにコピー

は番兵(SENTINEL)といって、
配列Aのどの数よりも大きな数を、
L, Rの右端に追加しておく

② 10-17行目



小さい方を持ってくる動作を
n1+n2回繰り返す

番兵はどの数よりも大きいので
Rの中身を全部見終わっても
のところでストップされる
(配列の外に出て行かないで済む)

K 会夏期講習 2013 情報講座

・ A0J 0529 “Darts” を解こう

分割統治法は $O(n^2)$ が $O(n \log n)$ になったりと劇的に計算量が良くなることが多く、そのアルゴリズムも「これ思いついたひと天才じゃね？」と思えるくらい美しいものが多い。

プログラミングコンテストでもときどき出題され、時間内に解き方を思いつかないといけないうので結構難しい問題となる。この Darts は情報オリンピック本選に出た問題で、合宿の出場可否を分けた問題と言っても良い難問であった。

この問題は本選の時は部分点が設定されていて、 $N \leq 100$ で解ければ 20 点、 $N \leq 300$ で解ければ 50 点、 $N < 1000$ で解ければ満点であった。

A0J は多分部分点とかではなく、 $N < 1000$ で解けないと無慈悲に 0 点をつけてくるので、部分点をとりたいひとは情報オリンピックの web ページに行って、過去問の 2008 年度本選問題のページを開こう。そのときの採点データが公開されているので、それをダウンロードして手元でためしてみると良い。

解説も J01 のページに丁寧に書かれている。