4. 貪欲法・動的計画法 (DP)

佐々木 佑

1 貪欲法 (Greedy Algorithm)

貪欲法 (Greedy Algorithm) とは, 1 つのルールに従って, 貪欲にその場での最善なものを選択することを繰り返すアルゴリズムです. 欲張り法・グリーディ算法ともいいます.

1.1 硬貨の問題

硬貨の問題

1 円玉, 5 円玉, 10 円玉, 50 円玉, 100 円玉, 500 円玉がそれぞれ C_1, C_5, C_{10}, C_{50}, C_{100}, C_{500} 枚ずつあります. できるだけ少ない枚数の硬貨で A 円を支払いたいと考えています. 何枚の硬貨を出す必要があるでしょうか? なお, そのような支払い方は少なくとも 1 つは存在します. 入力は 1 行目に C_1, C_5, C_{10}, C_{50}, C_{100}, C_{500} が, 2 行目に A が与えられます.

制約

$0 \leq C_1, C_5, C_{10}, C_{50}, C_{100}, C_{500} \leq 10^9$

$0 \leq A \leq 10^9$

入力

```
3 2 1 3 0 2
620
```

出力 (500 円玉*1 枚, 50 円玉*2 枚, 10 円玉*1 枚, 5 円玉*2 枚 の 6 枚)

```
6
```

とても日常的な簡単な問題です. これは直感のとおり, 次のように解けば正しい答えが求まります.

- 500 円玉をできるだけ多く使い,
- 残った額に対し 100 円玉をできるだけ多く使い,
- 残った額に対し 50 円玉をできるだけ多く使い,
- 残った額に対し 10 円玉をできるだけ多く使い,
- 残った額に対し 5 円玉をできるだけ多く使い,
- 残った額を 1 円玉で払う.

あるいはシンプルに

- 大きな額の硬貨から優先的に使う.

というように表現できます.

プログラム例

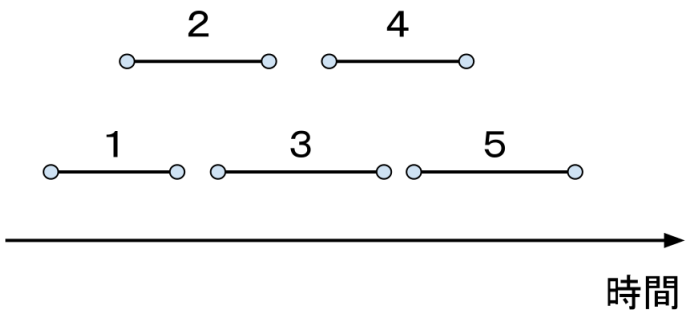
```
#include <iostream>
#include <algorithm>
using namespace std;

// コインの金額
const int V[6] = {1, 5, 10, 50, 100, 500};

// 入力
int C[6]; // C[0] = C_1, C[1] = C_5, C[2] = C_10, ...
int A;

int main(){
    for(int i=0 ; i < 6 ; i++ ){
        cin >> C[i];
    }
    cin >> A;

    int ans = 0;
    for(int i=5 ; i >= 0 ; i-- ){
        int m = A / V[i];
        int t = min( m , C[i] ); // コイン i を使う枚数
        A -= t * V[i];
        ans += t;
    }
    cout << ans << endl;
}
```

1.2 区間スケジューリング問題

今度は, 次のような問題を考えてみましょう.

区間スケジューリング問題

n 個の仕事があります. 各仕事は, 時間 s_i に始まり, 時間 t_i に終わります. あなたは各仕事について, 参加するかしないかを選ばなければなりません. 仕事に参加するならば, その仕事のはじめから終わりまで参加しなければなりません. また, 参加する仕事の時間帯が重なってはなりません. (開始の瞬間と終了の瞬間だけが重なるのも許されません)



できるだけ多くの仕事に参加したとき何個の仕事に参加できるでしょう.
入力の形式は次のようになっています.

n
s_1 t_1
s_2 t_2
...
s_n t_n

制約

$1 \leq n \leq 100000$
$1 \leq s_i < t_i \leq 10^9$

入力

```
5
1 3
2 5
4 7
6 9
8 10
```

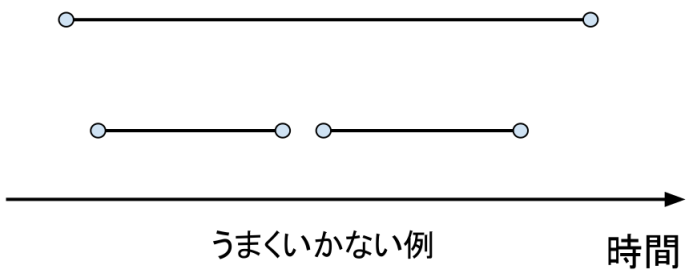
出力 (仕事 1,3,5 を選択)

```
3
```

この問題も貪欲法で解くことができます. しかし, 今度は前の問題ほどシンプルではありません. いろいろな貪欲法のアルゴリズムを考えることができます. 例えば最も思いつきやすいもののひとつに次のようなものがあるでしょう.

- 選べる仕事の中で開始時間が最も早いものを選ぶのを繰り返す.

このアルゴリズムでうまくいく場合もありますが, 例えば次のような場合, このアルゴリズムでは最適な解を求めることができません.



このように, 正しいルールを選ばないと, 間違ったアルゴリズムになってしまいます.
この問題では

- 選べる仕事の中で終了時間が最も早いものを選ぶのを繰り返す.

という方法で最適解を求めることができます. このアルゴリズムが正しい理由は自分で考えてみてください.

プログラム例

```
#include <iostream>
#include <map>
#include <algorithm>
using namespace std;

int main(){
    int n, s[100001], t[100001];
    // 仕事をソートするための pair の配列
    pair<int,int> itv[100001];

    cin >> n;
    for(int i=0 ; i < n ; i++ ){
        cin >> s[i] >> t[i];
    }

    // pair は辞書順にソートされる
    // 終了時間の早い順にしたいため, 終了時間 t[i] を first に、開始時間 s[i] を second に入れる
    for(int i=0 ; i < n ; i++ ){
        itv[i].first = t[i];
        itv[i].second = s[i];
    }
    sort( itv , itv + n );

    // time は最後に選んだ仕事の終了時間
    int ans = 0, time=0;
    for(int i=0 ; i < n ; i++ ){
        if( time < itv[i].second ){
            ans++;
            time = itv[i].first;
        }
    }

    cout << ans << endl;
}
```

貪欲法で解ける問題

AOJ No.0031 : Weight
AOJ No.0521 : Change
AOJ No.1052 : Old Bridges
AOJ No.1063 : Watchin' TVA
AOJ No.2216 : Summer of KMC

2　動的計画法 (DP: Dynamic Programming)

動的計画法 (DP: Dynamic Programming) とは, ある問題を小さな問題に分割しその計算結果を次の計算結果で使う手法です. 動的計画法はナップザック問題や経路の問題などを解くときに利用されます. 分割統治法がトップダウン的な手法であるのに対し, 動的計画法はボトムアップ的な手法といえます. 分割統治法の説明はここでは割愛します.

2.1　硬貨の問題

硬貨の問題

1 円玉、7 円玉、12 円玉があり, できるだけ少ない枚数で A 円を支払うとき何枚の硬貨を出す必要があるでしょう。

制約

$0 \leq A \leq 1000$

入力

```
14
```

出力 (7 円玉*2 枚)

```
2
```

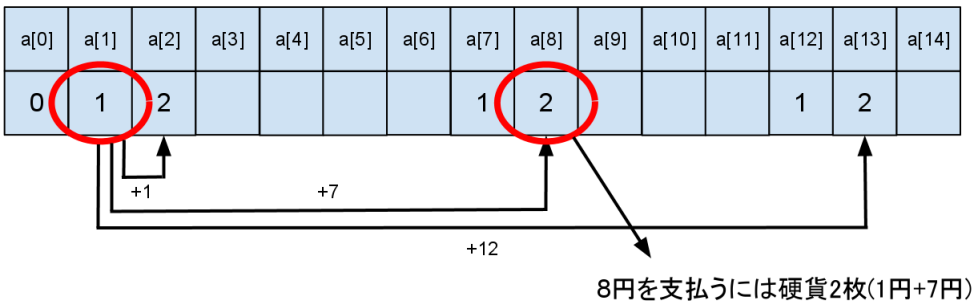
1 円玉と 7 円玉と 12 円玉だけがある場合, 14 円を払うには, 7 円玉 2 枚で払うのが最適です. しかし, この問題を大きな金額の硬貨を優先的に使うような貪欲法を使うと 12+1+1 の 3 枚になってしまい最適解を求めることができません. 最適解を見つけるにはいささか工夫が必要です.

たとえば, A 円を小さくしてみます. コインの金額は同じままで, A-1 円の最適な払い方をまず計算してみます. そしたらそこから簡単に A 円の最適な払い方も求まったりしないでしょうか? 残念ながら, 難しそうです. A-1 円の払い方を応用して A 円を払うなんて, あとに 1 円玉を付け足すくらいしか手がありません. しかし, それが最適になるとは限りません.

諦めずにさらに問題を小さくします. A-1 円の最適な払い方, A-2 円の最適な払い方, ... , 3 円の最適な払い方, 2 円の最適な払い方, 1 円の最適な払い方, と全部計算したらどうでしょう. 計算結果は配列などに値をメモするとよいでしょう.

1 円と 7 円と 12 円の場合は硬貨 1 枚で支払うことができるのは明らかです.

2 円は, (1 円玉+1 円玉) で硬貨 2 枚で支払うことができます. 同様に 8 円は (1 円玉+7 円玉) で, 13 円は (1 円玉+12 円玉) で硬貨 2 枚で支払うことができます.



このようにして 1 円から 14 円までの最適解を順に求めていきます. 途中 i 円を支払う硬貨の枚数でさらに少ない枚数を見つけた場合は値を更新しましょう.

最終的に 14 円を支払うときは最小で 2 枚の硬貨 (7 円玉+7 円玉) という答えが求まります. 問題が 1 円玉, 7 円玉, 12 円玉でなく C_1 円玉, C_2 円玉, ... ,C_n 円玉になった場合については自分で考えてみましょう.

動的計画法では「ひとまわり小さい問題の答えから大きな問題の答えを作れるかどうか」 に焦点を絞ることで解が求まるか考えることが大切です.

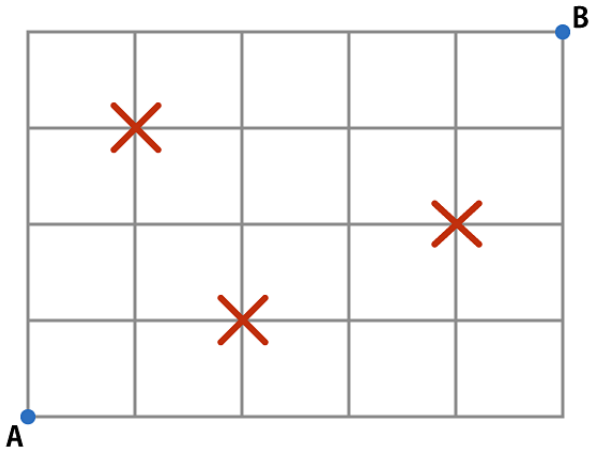
プログラム例

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

const int INF = 1e+8;

int main(){
    int A, dp[1020] = {0};
    cin >> A;

    // 初期化
    for(int i=1 ; i <= A ; i++ ){
        if( i == 1 || i == 7 || i == 12 )
            dp[i] = 1; // 1 円、7 円、12 円は硬貨 1 枚で支払うので 1 で初期化
        else
            dp[i] = INF; // 解が求まっていない i 円を INF で初期化
    }

    // 1 円から A 円までの解を動的計画法ですべて求める
    for(int i=1 ; i <= A ; i++ ){
        if( dp[i] != INF ){
            dp[i+1] = min( dp[i+1] , dp[i]+1 );
            dp[i+7] = min( dp[i+7] , dp[i]+1 );
            dp[i+12] = min( dp[i+12] , dp[i]+1 );
        }
    }
    cout << dp[A] << endl;
}
```

2.2 経路の問題

経路の問題

下図のような格子状の道があります. A 点から B 点に向かって遠回りせずに移動したいとき, 移動する経路は何通りあるでしょう?
ただし縦 h, 横 w とし, 通れない交差点が n 個とします.



制約

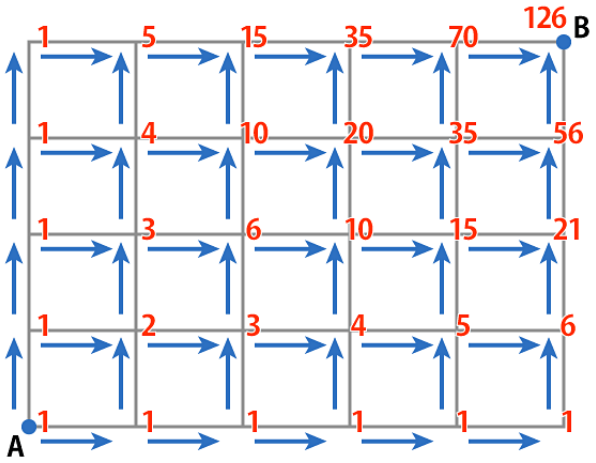
$1 \leq w, h \leq 16$

この問題では探索による全列挙で解くという方法が考えられます.
しかし, この方法では大きなサイズになると解くことができません. 計算量を考えると, 右に行くか上に行くかの選択を $h + w$ 回行うことになるので, ざっと $O\left(2^{h+w}\right)$ 程度の計算量となります. 2^{30} でだいたい 10 億ですので, たった 15×15 程度のサイズでも, かなりの時間が掛かってしまうことが予想できるわけです. これではあまりに時間が掛かり過ぎてしまうので別の方法を考える必要があります.

今回の例では,

- (1 つ下の点にたどり着く経路の数) + (1 つ左の点にたどり着く経路の数) = (その点に辿り着く経路の数)

となっていることに気づきます. (1 つ下の点にたどり着く経路の数) と (1 つ左の点にたどり着く経路の数) が分かれば, その点にたどり着く経路の数は容易に計算できます. 左下から順番に, 左と下の数字を足し算していくだけで, 簡単に答えを求めることができます.



この場合の計算量を考えると, 各地点において, 下と左から足し算しているだけですので, $O\left(h \times w\right)$ となります.

問題 School Road(AOJ 0515)

太郎君の住んでいる JOI 市は， 南北方向にまっすぐに伸びる a 本の道路と，東西方向にまっすぐに伸びる b 本の道路により， 碁盤の目の形に区分けされている．南北方向の a 本の道路には，西から順に $1, 2, ..., a$ の番号が付けられている．また，東西方向の b 本の道路には，南から順に $1, 2, ..., b$ の番号が付けられている．西から i 番目の南北方向の道路と，南から j 番目の東西方向の道路が交わる交差点を (i, j) で表す．太郎君は，交差点 $(1, 1)$ の近くに住んでおり， 交差点 (a, b) の近くの JOI 高校に自転車で通っている．自転車は道路に沿ってのみ移動することができる．太郎君は，通学時間を短くするため， 東または北にのみ向かって移動して通学している．現在， JOI 市では， n 個の交差点 $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ で工事を行っている．太郎君は工事中の交差点を通ることができない．

太郎君が交差点 $(1, 1)$ から交差点 (a, b) まで，工事中の交差点を避けながら，東または北にのみ向かって移動して通学する方法は何通りあるだろうか．太郎君の通学経路の個数 m を求めるプログラムを作成せよ．

データセットは複数与えられ, 各データセットの形式は以下のようになっています. 入力の終わりはゼロを 2 つ含む行で示されます.

a b

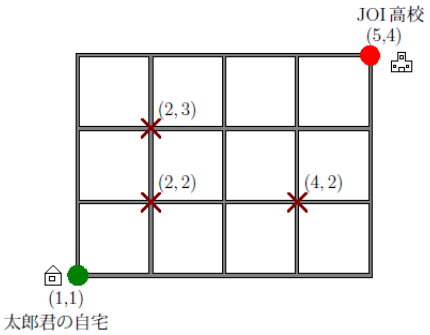
n

x_1 y_1

x_2 y_2

...

x_n y_n



制約

$1 \leq w, h \leq 16$

$1 \leq n \leq 40$

入力

```
5 4
3
2 2
2 3
4 2
5 4
3
2 2
2 3
4 2
0 0
```

出力

```
5
5
```

解答例

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

int main(){
    int a,b;
    while( cin >> a >> b , a || b ){
        int n, dp[20][20] = {0};
        dp[1][1] = 1;

        cin >> n;
        for(int i=0 ; i < n ; i++ ){
            int x,y;
            cin >> x >> y;
            dp[y][x] = -1; // 交差点 (x,y) までの経路の数が求まっていないので-1 で初期化する
        }

        for(int y=1 ; y <= b ; y++ ){
            for(int x=1 ; x <= a ; x++ ){
                if( dp[y][x] != -1 ){
                    if( y != 1 && dp[y-1][x] != -1 ){
                        dp[y][x] += dp[y-1][x];
                    }
                    if( x != 1 && dp[y][x-1] != -1 ){
                        dp[y][x] += dp[y][x-1];
                    }
                }
            }
        }
        // (a,b) までの経路の数を出力
        cout << dp[b][a] << endl;
    }
}
```

動的計画法で解ける問題 1 (数え上げるタイプ)

AOJ No.0168 : Kannondou
AOJ No.0515 : School Road
AOJ No.0547 : Commute routes
AOJ No.0557 : A First Grader
AOJ No.1105 : Unable Count
AOJ No.2186 : Heian-Kyo Walking

2.3 ナップサック問題

ナップサック問題

重さと価値がそれぞれ w_i, v_i であるような n 個の品物があります。これらの品物から、重さの総和が W を超えないように選んだときの、価値の最大値を求めなさい。
入力の形式は以下のようになっています。
$n\ W$
$w_1\ v_1$
$w_2\ v_2$
...
$w_n\ v_n$

制約
$1 \leq n \leq 100$
$1 \leq w_i, v_i \leq 100$
$1 \leq W \leq 10000$

入力

```
5 10
3 2
4 3
1 2
2 3
3 6
```

出力

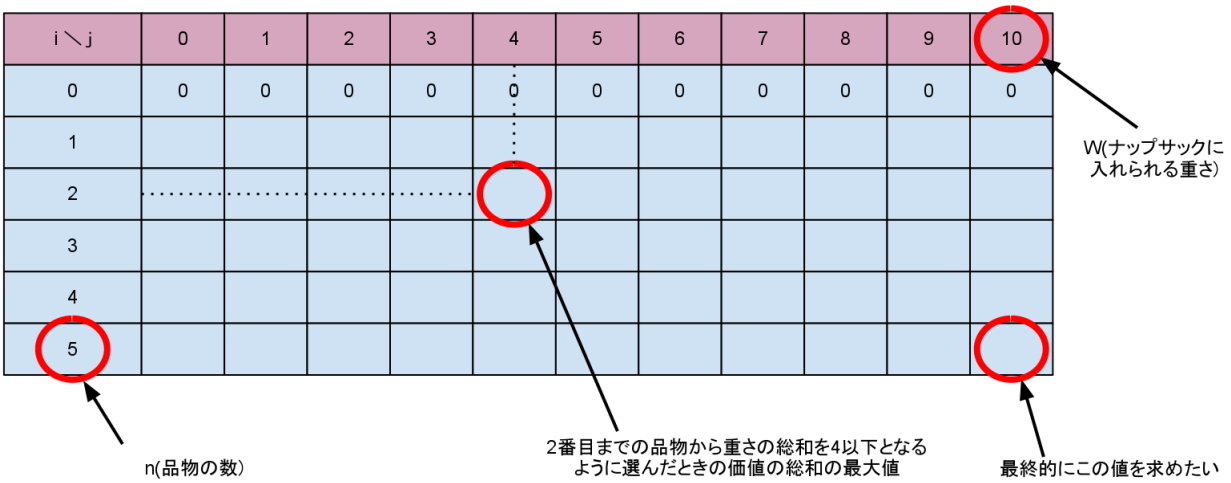
```
14
```

これはナップサック問題と呼ばれる有名な問題です．この問題も動的計画法で解くことができます．今回は，「合計の重さ」と「何番目の品物まで考えたか」という 2 つの要素に対し，「価値の合計の最大値」を格納します．

先ほどの入力例では品物は次のようになっています．

品物	1番	2番	3番	4番	5番
重さ (w)	3	4	1	2	3
価値 (v)	2	3	2	3	6

先ほどの入力例では重さ $W = 10$ ですので次のような二次元配列を用意するとよいでしょう．このような二次元配列を用意することで「合計の重さ」と「何番目の品物まで考えたか」という 2 つの要素に対し，「価値の合計の最大値」を格納することができます．



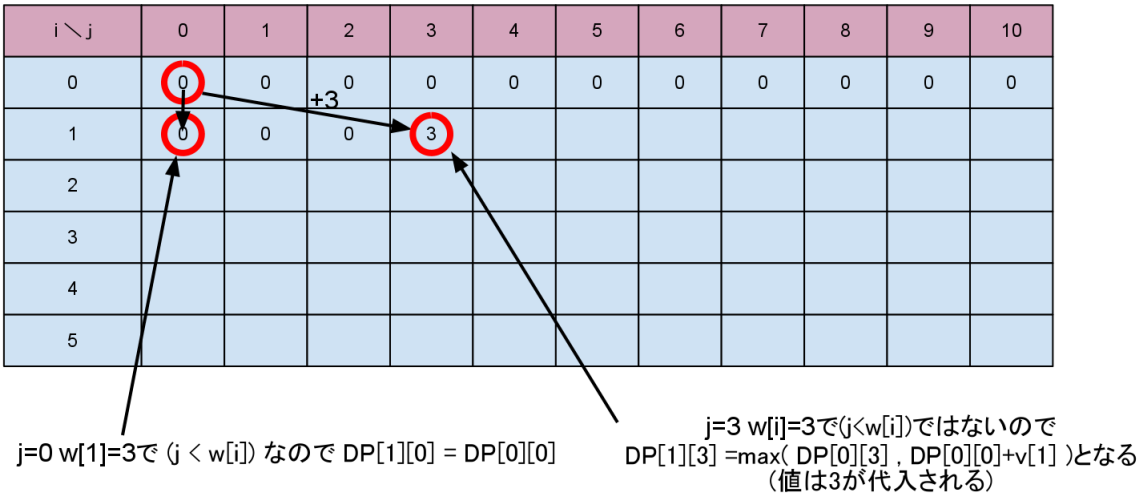
計算するときは

dp[i+1][j] := i 番目までの品物の重さがの総和が j 以下となるように選んだときの価値の総和の最大値

dp[0][j] = 0

$$\text{dp[i+1][j]} = \begin{cases} dp[i][j] & (j < w[i]) \\ \max(dp[i][j], dp[i][j-w[i]] + v[i]) & (\text{それ以外}) \end{cases}$$

という漸化式で計算していきます.



```
void solve(){
    for(int i=0 ; i < n ; i++ ){
        for(int j=0 ; j <= W ; j++ ){
            if( j < w[i] ){
                dp[i+1][j] = dp[i][j];
            }else{
                dp[i+1][j] = max( dp[i][j] , dp[i][j-w[i]] + v[i] );
            }
        }
    }
    cout << dp[n][W] << endl;
}
```

問題 A Thief(AOJ 0042)

泥棒が重さの総和 W まで耐えられる風呂敷を持っていて, 宝物がたくさん収蔵されている博物館に忍び込みました. 博物館には重さと価値がそれぞれ w_i, v_i であるような n 個の宝物があります.
重さの総和が W を超えない範囲で価値の総和が最大になるときの, お宝の価値総和と重さの総和を出力して終了するプログラムを作成してください. ただし, 価値の総和が最大になる組み合わせが複数あるときは, 重さの総和が小さいものを出力することとします.
データセットは複数与えられ, 各データセットの入力の形式は以下のようになっています. データセットの終わりは 0 が一つの行で示されます.
W
n
v_1, w_1
v_2, w_2
...
v_n, w_n
また出力の形式は以下のようにしてください.
Case データセットの番号:
風呂敷に入れたお宝の価値総和
そのときのお宝の重さの総和

制約
$1 \leq n \leq 1000$
$1 \leq v_i \leq 10000$
$1 \leq W, w_i \leq 1000$

入力

```
50
5
60,10
100,20
120,30
210,45
10,4
0
```

出力

```
Case 1:
220
49
```

解答例

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

// 入力
int W, n;
int v[1001], w[1001];

int dp[1001][1001];
int ans_weight;

void solve(){
    // 価値の最大値を求める
    for(int i=0 ; i < n ; i++ ){
        for(int j=0 ; j <= W ; j++ ){
            if( j < w[i] ){
                dp[i+1][j] = dp[i][j];
            }else{
                dp[i+1][j] = max( dp[i][j] , dp[i][j-w[i]] + v[i] );
            }
        }
    }
    // 価値の最大値 dp[n][W] となる最小の重さの総和を求める
    for(int j=0 ; j <= W ; j++ ){
        if( dp[n][j] == dp[n][W] ){
            ans_weight = j;
            break;
        }
    }
}

int main(){
    for(int t=1 ; scanf("%d", &W) , W ; t++ ){
        // 配列の初期化
        for(int i=0 ; i < 1001 ; i++ ){
            for(int j=0 ; j < 1001 ; j++ ){
                dp[i][j] = 0;
            }
        }

        scanf("%d", &n);
        for(int i=0 ; i < n ; i++ ){
            scanf("%d,%d", &v[i], &w[i]);
        }
        solve();
        printf("Case %d:\n", t);
        printf("%d\n%d\n", dp[n][W], ans_weight );
    }
}
```

2.4 最長共通部分列問題

最長共通部分列問題

2 つの文字列 $s_1s_2...s_n$ と $t_1t_2...t_m$ が与えられます. これら 2 つの共通部分列の長さの最大値を求めなさい. ただし, 文字列 $s_1s_2...s_n$ の部分列とは $s_1s_2...s_n$ からいくつかの要素を順序を維持して取出した列のことです. (連続でなくてもよい)

制約

$1 \leq n, m \leq 1000$

入力

```
4
4
abcd
becd
```

出力 (LCS は"bcd"で長さは 3)

```
3
```

この問題は**最長共通部分列 (LCS:Longest Common Subsequence)** と呼ばれる有名な問題です.
次のように定義してみましょう.

$dp[i][j] := s_1...s_i$ と $t_1..t_j$ に対する LCS の長さ (s の i 文字目までと t の j 文字目に対する LCS)

このように定義すると, $s_1...s_{i+1}$ と $t_1...t_{j+1}$ に対する共通部分列は,

- $s_{i+1} = t_{j+1}$ ならば, $s_1...s_i$ と $t_1..t_j$ に対する LCS の後ろに s_{i+1} をつなげたもの
- $s_1...s_i$ と $t_1..t_{j+1}$ に対する LCS
- $s_1...s_{i+1}$ と $t_1..t_j$ に対する LCS

のどれかであるので

$$dp[i+1][j+1] = \begin{cases} dp[i][j] + 1 & (s_{i+1} = t_{j+1}) \\ \max(dp[i][j+1], dp[i+1][j]) & (otherwise) \end{cases}$$

という漸化式が成り立ちます. この漸化式は計算量 $O(nm)$ で計算でき, 最長共通部分列の長さは $dp[n][m]$ となります.

例えば文字列 s ="abac" と t ="abaa" の最長共通部分列は"aba"でその長さは 3 となります. 二次元配列 dp を用いた計算結果は図 1 のようになります.

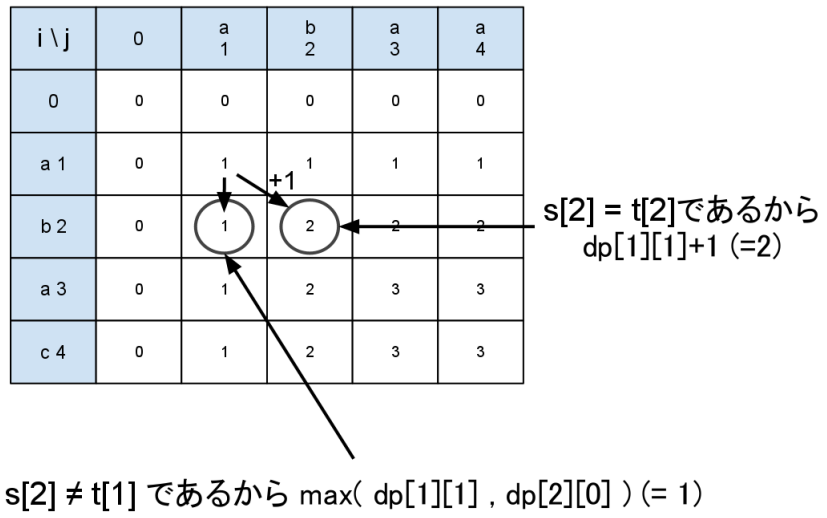


図 1: DP テーブル

プログラム例

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

// 入力
int n, m;
string s, t;

int dp[1001][1001];

void solve(){
    for(int i=0 ; i < n ; i++ ){
        for(int j=0 ; j < m ; j++ ){
            if( s[i] == t[j] ){
                dp[i+1][j+1] = dp[i][j] + 1;
            }else{
                dp[i+1][j+1] = max( dp[i][j+1] , dp[i+1][j] );
            }
        }
    }
}

int main(){
    cin >> n >> m;
    cin >> s >> t;
    solve();
    cout << dp[n][m] << endl;
}
```

2.5 最小コストを求める

問題 Cicada(AOJ 2272)

家と学校が描かれた地図は長方形状になっており， $H \times W$ 個のブロックに分割される．一番左上のブロック $(0,0)$ に N 君の家があり， 一番右下のブロック $(W-1, H-1)$ に学校がある．彼は学校へまっすぐ登校したいので家の方に逆戻りすることはない (地図において，右か下のブロックにしか移動しない)．各ブロックにおいて， そこにいる蝉の数だけ不快になる．どのような道を選べば， 不快な度合いを最小化， つまり出会う蝉の数を最小に出来るだろうか.

制約

$2 \leq W, H \leq 50$

N 君の家と学校には蝉はいない.

ブロック中の蝉の数は 0 以上 9 以下である.

入力 1

```
3 3
023
321
120
```

出力 1

```
5
```

入力 2

```
5 6
000000
923450
000000
054329
000000
```

出力 2

```
4
```

この問題は右か下の蝉の少ない方に進むという貪欲法を使うと入力 2 のときにうまくいきません. この問題では動的計画法を使うと解くことができます.

`dp[y][x] := (x,y)` までにたどり着く最小の蝉の数

となるように $(0,0)$ から計算していきます. (x,y) の地点までにたどり着く最小の蝉の数は, $(x-1,y)$ か $(x,y-1)$ までの地点にたどり着く蝉の数の小さい方と (x,y) の地点にいる蝉の数を足した数となります.

プログラム例

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

int main(){
    int h, w, dp[51][51];
    string s[51];

    cin >> h >> w;
    for(int i=0 ; i < h ; i++ ){
```

```
        cin >> s[i];
    }
    for(int y=0 ; y < h ; y++ ){
        for(int x=0 ; x < w ; x++ ){
            dp[y][x] = 10000;
        }
    }
    dp[0][0] = 0;
    for(int y=0 ; y < h ; y++ ){
        for(int x=0 ; x < w ; x++ ){
            if( y != 0 ){
                dp[y][x] = min( dp[y][x] , dp[y-1][x] + (s[y][x]-'0') );
            }
            if( x != 0 ){
                dp[y][x] = min( dp[y][x] , dp[y][x-1] + (s[y][x]-'0') );
            }
        }
    }
    cout << dp[h-1][w-1] << endl;
}
```

動的計画法で解ける問題 2 (最大値や最小値を求めるタイプ)

AOJ No.0089 : The Shortest Path on A Rhombic Path
AOJ No.0191 : Baby Tree
AOJ No.0202 : At Boss's Expense
AOJ No.1126 : The Secret Number
AOJ No.2272 : Cicada

動的計画法で解ける問題 3 (未分類)

AOJ No.0157 : Russian Dolls
AOJ No.0541 : Walk
AOJ No.1167 : Pollock's conjecture
AOJ No.2011 : Gather the Maps!
AOJ No.2199 : Differential Pulse Code Modulation
AOJ No.2284 : The Legendary Sword
AOJ No.2420 : Anipero 2012

参考文献

[1] 秋葉拓哉, 岩田陽一, 北川宜稔: 『プログラミングコンテストチャレンジブック』, 毎日コミュニケーションズ (2010).

[2] 『最強最速アルゴリズマー養成講座:病みつきになる「動的計画法」　、その深淵に迫る』, http://www.itmedia.co.jp/enterprise/articles/1005/15/news002.html

[3] 『最強最速アルゴリズマー養成講座：アルゴリズマーの登竜門、「動的計画法・メモ化再帰」はこんなに簡単だった』, http://www.itmedia.co.jp/enterprise/articles/1003/06/news002.html

[4] 『アルゴリズムコンテストの挑み方 (2)』, http://www.kmonos.net/wlog/90.html