Algorithm And Tips For Competitive Programming

Tomoki Imai

2013

Contents

テンプレート

各種バッドノウハウを含む。

```
#include <iostream>
#include <cstdio>
#include <iomanip>
#include <vector>
#include <map>
#include <set>
#include <queue>
#include <bitset>
#include <stack>
#include <utility>
#include <numeric>
#include <algorithm>
#include <functional>
#include <cctype>
#include <complex>
```

```
16 #include <string>
17 #include <sstream>
18
19 using namespace std;
20
21 #define all(c) c.begin(),c.end()
22 #define rall(c) c.rbegin(),c.rend()
23 #define rep(i,n) for(int i=0;i<(n);i++)
24 #define tr(it,container) for(typeof(container.begin()) it = container.begin(); \
25                                                     it != container.end(); ++it)
26 #define mp(a,b) make_pair((a),(b))
27 #define eq ==
28
29 typedef long long ll;
30 typedef complex<double> point;
31 typedef pair<int,int> pii;
32
33 // → ↑ ← ↓
34 const int dx[] = {1,0,−1,0};
35 const int dy[] = {0,−1,0,1};
36
37
38 const double EPS = 1e−9;
39
40 int main(){
41
42     return 0;
43 }
```

算術型

int

基本中の基本。10^9くらい。こわいときにはlong longを使うことを推奨。

long long

大きい整数。10^{18}?くらい。

double

floatは使ってはだめ。

char

-128 ~ 127くらい。ちいさい。基本的には文字を入れるのに使う。vector<char>を vector<bool>の代わりに使ってもいい。

```
char c = 'a';
//cctypeが必要。大文字に変換する。すでに大文字のときは何も起こらない。
c = toupper(c);
//小文字に変換する。
c = tolower(c);
//vector<bool>の代わり。
vector<char> used(10,false);
```

bool

true(==1)とかfalse(==0)を入れるためだけに使う。ただしvector<bool>は使ってはいけない。

補助関数

上記の型に関する便利な関数。

床関数、天井関数、および四捨五入。返り値はdouble。cmathをincludeする。

```
double x = 0.3;
int f = floor(x); // -> 0
int c = ceil(x); // -> 1
int r = round(x) // -> 0
```

任意の場所で四捨五入したいときには、10^nをかけて、roundした後に、 10^nで割ればいい。

```
double x = 0.123456789;
//0.123
double r = round(x*1000) / 1000.0;
```

入出力

基本はcin,coutを使おう。

cin,cout

iostream,iomanipをincludeしておくこと。

cin

基本的な使い方について。

```
int n;
cin >> n;
vector<int> V(n);
rep(i,n) cin >> V[i];
```

こうすると、短く書ける。

入力の最後まで読む。

```
int n;
while(cin >> n){
    //処理
}
```

n=0のとき終わりとかの場合は、条件に&&n!=0とかをつける。

数値をカンマ区切りで読み込む。

```
int x,y;char c;
//cにカンマが入る
cin >> x >> c >> y;
```

冗長かもだけど、一番楽。

空白とか含めて一行読む。

```
string s;
getline(cin,s);
```

改行文字は、sに入らず、かつ読み捨てされる。 cinでは、改行文字は読み捨てないことに注意しよう。 つまり、数値<改行>文字列<改行>を読みたいときには、

```
int n;string s;
// 数値
cin >> n;
// 改行よみとばし
cin.ignore();
// 文字列
getline(cin,s);
```

とする。cinは改行文字を残すので、ignoreでそれを読み捨てないといけない。 また、ignoreの引数は読み捨てる文字数。引数なしの場合は1を渡したのと同等 の効果がある。

cout

有効数字等が設定されている問題は、必ず多めに出力すること。多めに出す分に は大丈夫。

基本の使い方

```
int n;vector<int> V(n);
cout << n << endl;
rep(i,n) cout << V[i] << "␣";
cout << endl;
```

以下主なiomanipの使い方

```
int n = 123;double d = 1.23;

//10進数 -> 123
cout << dec << n << endl ;

//8進数 -> 173
cout << oct << n << endl ;

//16進数 -> 7b
cout << hex << n << endl ;

//16進数かつ、大文字 -> 7B
cout << hex << uppercase << n << endl;

//10進数に戻す
cout << dec;

//幅が10になるようにする。デフォルトは右寄せ
// -> xxxxxxx123 (default)
cout << setfill('x') << setw(10) << right << n << endl;

// -> 123xxxxxxx
cout << setfill('x') << setw(10) << left << n << endl;
// -> 123yyyyyyy
cout << setfill('y') << setw(10) << n << endl;

//小数点以下10桁表示に。
cout << fixed << setprecision(10);

// -> 1.2300000000
cout << d << endl;
// -> 12.3000000000
cout << 10*d << endl;
```

```
//小数点の表示を元に戻す
std.unsetf(ios::fixed);
// —> 1.23
cout << d << endl;
```

基本的には、引数のあるマニピュレータの効果は保存される。

scanf,printf

C++では、cstdioをinclude。複雑な書式とかが必要なときにはこっちを使うといいかもしれない。

scanf

基本的な使い方

```
int n;char tmp[256];
scanf("%d\n",&n);
gets(tmp);
```

stringに直接いれるのはだめ。scanfはcinと同様に改行を残す。getlineするならcin.ignore。getsするなら、直前のscanfで改行を読んでおく必要がある。また、scanfで改行を読むのでなく、直後にgetc(stdin)してもいい。

printf

基本的な使い方

```
int n = 100;
printf("n␣is␣%d\n",n);
```

scanfとほとんど同様の使い方ができる。

指定子	出力書式
%c	文字
%s	文字列
%d	符号付き１０進整数
%u	符号なし10進
%f	10進浮動小数点数
%o	符号なし8進
%x	符号なし16進(Xなら大文字)
%%	%記号

Table 1: 書式指定子

高速化

以下のコードをmain関数の最初に書くことで、cin,coutの速度が2倍程度になる。

```
ios::sync_with_stdio(false);
cin.tie(0);
```

ただし、このコードはstdioとの同期を切るという意味なので、これを使うとき にはprintfやscanfを使用してはだめ。

std::vector

ここでは、配列のSTL版である、vectorの使いかたについて書く。 ここに書かれている関数は、string等にも用いることができるものが多い。 ちなみに、vector<bool>は使ってはいけない。bitsetや、vector<char>をつかうこと。 また、all(vector)は、vector.begin(),vector.end()とdefineしている。

基本

```
//要素数10で、初期値は−1にする。
vector<int> vec(10,−1);
//vecの最初から3つの要素をコピーする。
vector<int> newvec(vec.begin(),vec.begin()+3);
//vecの最初から3つの要素を削除する。
vec.erase(vec.begin(),vec.begin()+3);
```

並び換え

sort

C++のsortは、$O(n \log n)$で、introsort。何も指定しない場合には昇順にソートされる。注意すべきなのは、C++11では、最悪ケースで$O(n \log n)$となること。C++03では特に何も制限はないが、g++ならば$O(n \log n)$である。

```
//昇順（sort(vec.begin(),vec.end())）(2,1,3) −> (1,2,3)
sort(all(vec));
//降順（ただ単純にreverseしてもいい）(2,1,3) −> (3,2,1)
sort(all(vec),greater<int>());
```

第三引数には関数を渡すことができる。

```
bool comp(const int& a ,const int& b){
    return abs(a) < abs(b);
}
int main(){
    vector<int> vec(10);
    //絶対値が小さい順にソート
    sort(all(vec),comp);
}
```

stable_sort

sortとちがって、同じ優先順位の要素の順番は保存される。最悪計算量は$O(n \log^2 n)$である。

```
stable_sort(all(vec),comp);
```

unique

隣あう同じ要素を一つにまとめる。eraseすることに注意。

```
int ints[] = {1,1,2,1,1};
vector<int> vec(ints,ints+5);
vec.erase(unique(all(vec)),vec.end());
// 1 2 1
rep(i,vec.size()) cout << vec[i] << endl;
```

rotate

rotateは第二引数の場所を先頭にするように回転する。

```
int[] ints = {1,2,3,4,5};
vector<int> vec(ints,ints+5);
rotate(vec.begin(),vec.begin()+1),vec.end()); //2,3,4,5,1
rotate(vec.begin(),vec.end()−1,vec.end()); //5,1,2,3,4
```

next_permutation

順列をすべて列挙する。$N!$個なので、それなりの勢いで大きくなる。章末の付録参照。

```
vector<int> V = {3,2,1,4};
//ソートすること。
sort(all(V));

do{
   for(int i=0;i<V.size();i++){
       cout << V[i] << "␣";
   }
   cout << endl;
}while(next_permutation(all(V)));
```

探索

全探索

全部しらべる。

```
int linear_search(vector<int> V,int val){
    rep(i,V.size()){
        if(V[i] == val) return i;
    }
    return -1;
}
```

二分探索

ある条件を満たす最小のものを探す。ただし単調増加な物にしかつかえない。 叙述関数をPとすると、

```
double lower = 0,upper = 1000000;
for(int i=0;i<200;i++){
    double m = (lower+upper) / 2;
    if(P(m)){
        upper = m;
    }else{
        lower = m;
    }
}
```

とすると、upperに求めたい値がはいる。もしみつからなかった場合には、値は 変わらない。なので、lower,upperには極端な値を設定すること。200という回数 は、すこし多め。100で十分。対象がvectorの場合は以下のように書ける。

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main(){
    vector<int> v = {1,2,3,1,2,3,1,2,3};
    // ソートする必要あり。
    //  i  0 1 2 3 4 5 6 7 8
    // -> [1,1,1,2,2,2,3,3,3]
    sort(v.begin(),v.end());
    // 2以上の数値が初めて現れる場所 -> 3
    int lower = lower_bound(v.begin(),v.end(),2) - v.begin();
    // 2より大きい数値が初めて表われる場所 -> 6
    int upper = upper_bound(v.begin(),v.end(),2) - v.begin();
    // 2の個数
    int number_of_2 = upper - lower;
}
```

文字列操作

stringをincludeする。cctypeもいるかも。

std::string

charをラップしたテンプレートクラス。 基本的な使い方について

部分列

```
          //0123456789
string str("abcdefghij");
// 5番目以降
str.substr(5);    // "fghij"
// 5番目から3つ
str.substr(5,3); // "fgh"
//全部小文字にする
transform(s.begin(),s.end(),s.begin(),::tolower);
```

substrは一つの引数でそこから先全部、二つの場合は第一引数の位置から、第二 引数の数だけ持ってくる。

検索

stringには、いくつかのfindが定義されている。線形検索なので、早い検索が必 要なときには後述するKMP法やBM法を用いること。

- find 引数が最初に現れる位置
- rfind 引数が最後に表われる位置
- find_first_of 引数の文字のどれかが最初に表われる位置
- find_last_of 引数の文字のどれかが最後に表われる位置
- find_first_not_of 引数の文字のどれかではない文字が最初に表われる位置
- find_first_not_of 引数の文字のどれかではない文字が最後に表われる位置

第二引数として探すための最初の位置を指定できる。

Boyer Moore法

KMP法

stringstream

cinやcoutのようなstreamをstringを元に作成したりする。基本的には、string をフォーマットしたり、intやlongに、intやlongから変換するために使用する。

```
stringstream ss;
ss << 102;
string s;
ss >> s;
```

再帰下降構文解析

整数論

最大公約数,最小公倍数

ユークリッドの互除法を使う。intをlong longに置換してもいい。 $O(\log n)$

最大公約数

```
int gcd(int a,int b){
    return b==0 ? a : gcd(b,a%b);
}
```

最小公倍数

```
int lcm(int a,int b){
    return a*b / gcd(a,b);
}
```

mod

long longに入らないような答えのときにmodが登場する。

modの計算式について

$$a \equiv c \pmod{m}$$
$$b \equiv d \pmod{m}$$

の条件下では以下の式が成り立つ。

$$a + b \equiv c + d \pmod{m}$$
$$a - b \equiv c - d \pmod{m}$$
$$a \times b \equiv c \times d \pmod{m}$$

さらに、mが素数の場合、以下の関係が成り立つ。

$$a^m \equiv a \pmod{m}$$
$$a^{m-1} \equiv 1 \pmod{m}$$
$$a^{m-2} \equiv \frac{1}{a} \pmod{m}$$

つまり、aで割ることと、a^{m-2}を掛けることは同じである。
これは、$C(10000, 5000) \pmod{p}$といった式を計算する際、次の冪乗の演算と組みあわせて用いる。

冪乗のmod

いわゆるmod_pow。計算量は$O(\log n)$。

```
ll mod_pow(ll x,ll n,ll mod){
    if(n==0) return 1;
    ll ret = mod_pow(x * x % mod,n/2,mod);
    if(n%2== 1)  ret = ret * x % mod;
    return ret;
}
```

ちなみにC++のpowを使うときに、引数が整数で、返り値も整数であることを期待 するときには、上記のpowを使うべき。なぜならC++のpowは double,double->doubleな関数であるから。

素数

エラトステネスの篩

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

//上限より余裕を取ること。
const int M = 1e6+10;
bool isPrime[M];

void sieve(){
    for(int i=2;i<M;i++) isPrime[i] = true;
    for(int i=2;i*i < M;i++){
        if(!isPrime[i]) continue;
        for(int j=i*i;j<M;j+=i){
            isPrime[j] = false;
        }
    }
}

int main(){
    sieve();
    for(int i=0;i<M;i++){
        if(isPrime[i]) cout << i << endl;
    }
    return 0;
}
```

素数のリストが欲しかったら、適当に突っ込むこと。 実際には$O(n \log \log n)$だけれど、大体$O(n)$だと思っていい。

素因数分解

コンビネーション

くみあわせ。

コンビネーションの数

```
#include <iostream>
#include <vector>

using namespace std;
typedef long long ll;

//いい感じのやつ（n=61まで大丈夫）
ll combi2(int n,int r){
    if(n < r) return 0;
    ll ret = 1;
    for(int i=0;i<r;i++){
        ret *= n-i;
        ret /= i+1;
    }
    return ret;
}

// これはまちがっています。要修正
// http://stackoverflow.com/questions/3537360/calculating-binomial-coefficient-nck-for-large-
ll mod_combi(ll n,ll r,ll mod){
    if(n < r) return 0;
    ll ret = 1;
    for(int i=0;i<r;i++){
        ret = (ret * mod_pow(i+1,mod-2,mod)) % mod;
    }
    for(int i=0;i<r;i++){
        ret = (ret * (n-i)) % mod;
    }
    return ret;
}

//パスカルの三角形（n=66まで大丈夫）
//たくさん必要になるときはこっちのほうがいい。
ll combi3(int n,int r){
    int N = n+1;
    vector<vector<ll> > memo(N,vector<ll>(N,0));
    memo[0][0] = 1;
```

```
    for(int i=1;i<N;i++){
        memo[i][0] = memo[i−1][0];
        for(int j=1;j<N;j++){
            memo[i][j] = memo[i−1][j−1] + memo[i−1][j];
        }
    }
    return memo[n][r];
}
```

列挙

```
int main(){
    int N,M;
    cin >> N >> M;

    vector<int> numbers(N);
    for(int i=0;i<N;i++) numbers[i] = i;

    stack<pair<int,vector<int> > > stack;
    stack.push(make_pair(0,vector<int>()));

    vector<vector<int> > combis;
    while(!stack.empty()){
        int lower = stack.top().first;
        vector<int> choose = stack.top().second;
        stack.pop();

        if(choose.size() == M){
            combis.push_back(choose);
            continue;
        }

        for(int i=lower;i<N−M+choose.size()+1;i++){
            vector<int> cop = choose;
            cop.push_back(numbers[i]);
            stack.push(make_pair(i+1,cop));
        }
    }

    cout << "size␣:" << combis.size() << endl;
    for(int i=0;i<combis.size();i++){
        for(int j=0;j<combis[i].size();j++){
            cout << combis[i][j] << "␣";
        }
        cout << endl;
    }
```

```
    return 0;
}
```

ヨセフス数

乱数

XORShiftをつかったらうれしいかもしれない。

```
unsigned long xor128(){
    static unsigned long x=123456789,y=362436069,z=521288629,w=88675123;
    unsigned long t;
    t=(x^(x<<11));x=y;y=z;z=w; return( w=(w^(w>>19))^(t^(t>>8)) );
}
```

行列

基本要素

正方行列用 //いつかなおす。

```
// 適宜intにしたりすること。
typedef vector<vector<ll> > ll_mat;
```

基本演算

かけ算とmod。たしざんはcoming soon.

```
namespace std{
    ll_mat operator*(const ll_mat& lhs,const ll_mat& rhs){
        int N = lhs.size();
        ll_mat ret(N,vector<ll>(N));
        for(int row=0;row<N;row++){
            for(int col=0;col<N;col++){
                int c = 0;
                for(int k=0;k<N;k++){
                    c+= rhs[row][k] * lhs[k][col];
                }
                ret[row][col] = c;
            }
        }
        return ret;
    }
```

```
    ll_mat operator%(ll_mat lhs,ll rhs){
        int N = lhs.size();
        ll_mat ret = lhs;
        for(int i=0;i<N;i++){
            for(int j=0;j<N;j++){
                ret[i][j] = ret[i][j] % rhs;
            }
        }
        return ret;
    }
};
```

基本操作

累乗

```
vector<vector<ll> >  mod_pow(vector<vector<ll> > x,ll n,ll mod){
    if(n==0){
        vector<vector<ll> > E(x.size(),vector<ll>(x.size()));
        for(int i=0;i<x.size();i++){
            E[i][i] = 1;
        }
        return E;
    }
    vector<vector<ll> > ret = mod_pow(x * x % mod,n/2,mod);
    if(n%2 == 1)  ret = ret * x % mod;
    return ret;
}
```

表示

```
void display_matrix(vector<vector<ll> > mat){
    for(int i=0;i<mat.size();i++){
        for(int j=0;j<mat[0].size();j++){
            cerr << mat[i][j] << "␣";
        }
        cerr << endl;
    }
}
```

ベクトルとのかけ算

一次元列ベクトルとのかけ算

```
vector<ll> mat_multi(vector<vector<ll> > lhs,vector<ll> rhs,int mod){
    vector<ll> ret(rhs.size());
    int N = lhs.size();
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            ret[i] = (ret[i] + lhs[i][j] * rhs[j]) % mod;;
        }
    }
    return ret;
}
```

テスト

AOJ 1327

動的計画法およびそれに似たやつら。(TODO)

LCS

Longest common sequence.

LIS

Longest increasing subsequence.

巡回セールスマン問題

bit演算をする。bitのループを先に回すこと。

データ構造

Union-Find

```
struct UnionFind{
    vector<int> par; // 親
    vector<int> rank; // 木の深さ
    UnionFind(int n){
        rep(i,n) par.push_back(i);
        rank = vector<int>(n);
```

```
    }
    // 親を探す
    int root(int x){
        if(par[x] == x){
            return x;
        }else{
            // 縮約
            return par[x] = root(par[x]);
        }
    }
    // x,yの含まれる集合を併合
    void unite(int x,int y){
        x = root(x);
        y = root(y);
        if(x==y) return;
        if(rank[x] < rank[y]){
            par[x] = y;
        }else{
            par[y] = x;
            if(rank[x] == rank[y]) rank[x]++;
        }
    }
    // 同じ集合にいるかどうか
    bool same(int x,int y){
        return root(x) == root(y);
    }
};
```

ヒープ

```
#include <queue>
#include <iostream>

using namespace std;

typedef pair<int,int> pii;

struct Comp{
    bool operator()(pii left,pii right){
        if(left.second < right.second) return true;
        else if(left.second == right.second and left.first > right.first) return true;
        else return false;
    };
};

int main(){
```

```
    // 何も書かないと降順。(おっきい方からでてくる。)
    // これは昇順(ちいさいほうから出てくる)にしたもの。
    priority_queue<int,vector<int>,greater<int> > Qi;
    //関数オブジェクトを使っていい感じにもできる。
    priority_queue<pii,vector<pii>,Comp> Q;
    Q.push(make_pair(1,2));
    Q.push(make_pair(2,2));
    Q.push(make_pair(3,2));
    while(not Q.empty()){
        cout << Q.top().first << "␣" << Q.top().second << endl;
        Q.pop();
    }
}
```

bitset

限られた大きさのvector<bool>を使いたいときに、bitsetを使うことができる。

```
#include <iostream>
#include <bitset>

using namespace std;
const int N = 10;

struct bit_cmp{
    bool operator() (const bitset<N> &left,const bitset<N> &right) {
        for(int i=N-1;i>=0;i--){
            if(left[i] < right[i]) return true;
            if(left[i] > right[i]) return false;
        }
        return false;
    }
};


int main(){
    //定数じゃないとダメ。最初は全部false
    bitset<N> bits;
    // すべての要素をtrue -> 1111111111
    bits.set();
    if(bits.all()) cout << "all" << endl;
    // 立っているbitの数 -> 10
    cout << bits.count() << endl;
    // すべての要素をfalse -> 0000000000
    bits.reset();
    if(bits.none()) cout << "none" << endl;
```

```
    //1番目の要素をtrue —> 0100000000
    bits.set(1);
    if(bits.any()) cout << "any" << endl;

    // 0110000000
    bits.set(2);
    //1番目の要素をfalse —> 0010000000
    bits.reset(1);

    if(bits[2]) cout << 2 << endl;
    cout << bits << endl;

    bitset<N> newbits;
    // 和を取る
    bits |= newbits;
    // 積を取る
    bits &= newbits;

    // 関数オブジェクトを作る必要アリ
    map<bitset<N>,int,bit_cmp> M;
}
```

グラフ

構成要素

隣接行列を使うか、vector<Edge>みたいのを使うかの二択。場合によってはNode みたいなのも使うかも。隣接行列を使うとメモリとか探すのとか重い。

```
struct Edge{
    int to,cost;
    Edge(int to,int cost) : to(to),cost(cost) {};
};
```

ベルマンフォード

O(N|E|)

```
const int INF = 1 << 30;
// s:始点,dist:距離,prev:最短経路木
bool bellman(const vector<vector<Edge> >& graph,int s,vector<int> &dist,vector<int> &prev){
    int n = graph.size();
```

```
    for(int i=0;i<n;i++) dist[i] = INF;
    dist[s] = 0;
    for(int i=0;i<n;i++) prev[i] = -1;

    bool neg_cycle = false;
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            for(int k=0;k<graph[j].size();k++){
                const Edge &e = graph[j][k];
                if(dist[e.to] > dist[e.from] + e.cost){
                    dist[e.to] = dist[e.from] + e.cost;
                    prev[e.to] = e.from;
                    if(i == n-1){
                        dist[e.to] = -INF;
                        neg_cycle = true;
                    }
                }
            }
        }
    }
    return !neg_cycle;
}
```

ダイクストラ

負の経路があったらダメ

```
#include <iostream>
#include <vector>
#include <queue>
#include <utility>
#include <algorithm>
#include <functional>

using namespace std;

typedef pair<int,int> pii;

struct Edge{
    int from,to,cost;
    Edge(int from,int to,int cost)
        : from(from),to(to),cost(cost) {};
};

int main(){
```

```
    int n,m;
    cin >> n >> m;
    vector<vector<Edge> > V(m);

    for(int i=0;i<n;i++){
        int a,b,cost;
        cin >> a >> b >> cost;
        V[a].push_back(Edge(a,b,cost));
        V[b].push_back(Edge(b,a,cost));
    }

    int ret = -1;
    int p,q;
    cin >> p >> q;
    vector<char> visited(m,false);
    //                 cost,where
    priority_queue<pii,vector<pii>, greater<pii> > Q;
    Q.push(make_pair(0,p));
    while(!Q.empty()){
        int cost,where;
        cost = Q.top().first;
        where = Q.top().second;
        Q.pop();
        if(visited[where]) continue;
        if(where == q){
             ret = cost;
             break;
        }
        visited[where] = true;
        for(int j=0;j<(int)V[where].size();j++){
            Q.push(make_pair(V[where][j].cost+cost,V[where][j].to));
        }
    }
    // 到達不能なときは-1
    cout << ret << endl;
    return 0;
}
```

ワーシャルフロイド

負の経路があってもOK。すべてのノードに対してすべてのノードへの距離を求める。もし負の閉路があったらiからiはマイナスになる。$O(|V|^3)$

```
// m はノードの個数。NOはでかい数
vector<vector<int> > V(m,vector<int>(m,NO));
```

```
// i->iは0にする。
rep(i,m){
    V[i][i] = 0;
}

for(int k=0;k<m;k++){
    for(int i=0;i<m;i++){
        for(int j=0;j<m;j++){
            V[i][j] = min(V[i][j],V[i][k]+V[k][j]);
        }
    }
}
```

最小全域木

プラム法による。$O(N^3)$だと思う。最小コストを求めるコードが以下。 ただし、vectorを使えばもっとよい。

```
typedef pair<int,int> pii;

int main(){
    int N;
    while(cin >> N){
        vector<vector<int> > M(N,vector<int>(N));
        // 距離行列を読みこむ
        rep(i,N) rep(j,N) cin >> M[i][j];
        vector<char> used(N,false);
        ll ret = 0;
        // cost , where.
        priority_queue<pii,vector<pii>,greater<pii> > Q;
        Q.push(mp(0,0));
        while(!Q.empty()){
            int cost = Q.top().first;
            int where = Q.top().second;
            Q.pop();
            if(used[where]) continue;
            used[where] = true;
            ret += cost;
            for(int i=0;i<N;i++){
                Q.push(mp(M[where][i],i));
            }
        }
        cout << ret << endl;
    }
    return 0;
```

```
}
```

最大流

Dinic法による。

```
struct Edge{
    int to,cap,rev;
    Edge(int to,int cap,int rev) : to(to),cap(cap),rev(rev) {};
};

void add_edge(vector<vector<Edge> >& E,int from,int to,int cap){
    E[from].push_back(Edge(to,cap,E[to].size()));
    E[to].push_back(Edge(from,0,E[from].size()-1));
}

vector<int> levels(vector<vector<Edge> > &E,int s){
    vector<int> level(E.size(),-1);
    level[s] = 0;
    queue<int> Q;
    Q.push(s);
    while(!Q.empty()){
        int v = Q.front();
        Q.pop();
        for(int i=0;i<E[v].size();i++){
            Edge &e = E[v][i];
            if(e.cap > 0 and level[e.to] == -1){
                level[e.to] = level[v]+1;
                Q.push(e.to);
            }
        }
    }
    return level;
}

int good_path(vector<vector<Edge> > &E,
        vector<int> &iter,
        vector<int> &level,
        int v,int t,int f){
    if(v == t) return f;
    for(int &i=iter[v];i<E[v].size();i++){
        Edge &e = E[v][i];
        if(e.cap > 0 and level[v] < level[e.to]){
            int d = good_path(E,iter,level,e.to,t,min(f,e.cap));
            if(d > 0){
```

```
                e.cap -= d;
                E[e.to][e.rev].cap += d;
                return d;
            }
        }
    }
    return 0;
}

int max_flow(vector<vector<Edge> > E,int s,int t){
    int flow = 0;
    const int INF = 1 << 30;
    while(true){
        vector<int> level = levels(E,s);
        if(level[t] < 0) return flow;
        vector<int> iter(E.size());
        int f;
        while((f=good_path(E,iter,level,s,t,INF)) > 0){
            flow += f;
        }
    }
}

int main(){
    int N,M;
    while(cin >> N >> M){
        // [0,N) is cow,[N,N+M) is barn.
        vector<vector<Edge> > E(N+M+2);
        int s = N+M;
        int t = N+M+1;
        for(int i=0;i<N;i++){
            add_edge(E,s,i,1);
        }
        for(int i=0;i<M;i++){
            add_edge(E,N+i,t,1);
        }

        for(int i=0;i<N;i++){
            int S;
            cin >> S;
            for(int j=0;j<S;j++){
                int k;
                cin >> k;
                k--;
                add_edge(E,i,N+k,1);
            }
```

```
        }

        cout << max_flow(E,s,t) << endl;
    }
    return 0;
}
```

最小費用流

Primal-Dual法による。

```
typedef pair<int,int> pii;

struct Edge{
    int to,cap,cost,rev;
    Edge(int to,int cap,int cost,int rev)
        : to(to),cap(cap),cost(cost),rev(rev) {};
};

void add_edge(vector<vector<Edge> > &E,int from,int to,int cap,int cost){
    E[from].push_back(Edge(to,cap,cost,E[to].size()));
    E[to].push_back(Edge(from,0,-cost,E[from].size()-1));
}

// s -> t (flow f)
//  if cant, return -1.
int min_cost_flow(vector<vector<Edge> > E,int s,int t,int f){
    const int INF = 1 << 30;
    int ret = 0;
    // potential
    vector<int> h(E.size());
    vector<int> prevv(E.size());
    vector<int> preve(E.size());

    while(f > 0){
        vector<int> dist(E.size(),INF);
        dist[s] = 0;
        priority_queue<pii,vector<pii>,greater<pii> > que;
        que.push(make_pair(0,s));
        while(!que.empty()){
            pii p = que.top();
            que.pop();
            int pf = p.first,ps = p.second;
            if(dist[ps] < pf) continue;
            for(int i=0;i<E[ps].size();i++){
```

```
                Edge &e = E[ps][i];
                if(e.cap > 0 and dist[e.to] > dist[ps] + e.cost + h[ps] - h[e.to]){
                    dist[e.to] = dist[ps] + e.cost + h[ps] - h[e.to];
                    prevv[e.to] = ps;
                    preve[e.to] = i;
                    que.push(make_pair(dist[e.to],e.to));
                }
            }
        }
        if(dist[t] == INF){
            return -1;
        }
        for(int v=0;v<E.size();v++){
            h[v] += dist[v];
        }
        int d = f;
        for(int v=t;v!=s;v=prevv[v]){
            d = min(d,E[prevv[v]][preve[v]].cap);
        }
        f -= d;
        ret += d * h[t];
        for(int v=t;v!=s;v=prevv[v]){
            Edge &e = E[prevv[v]][preve[v]];
            e.cap -= d;
            E[v][e.rev].cap += d;
        }
    }
    return ret;
}

int main(){
    while(true){
        int N,M;
        cin >> N >> M;
        if(N == 0 and M == 0) break;
        vector<pair<int,int> > men;
        vector<pair<int,int> > houses;
        for(int h=0;h<N;h++){
            for(int w=0;w<M;w++){
                char x;
                cin >> x;
                if(x == 'H') houses.push_back(make_pair(h,w));
                else if(x == 'm') men.push_back(make_pair(h,w));
            }
        }
```

```
        vector<vector<Edge> > E(men.size()+houses.size()+2);
        int s = men.size()+houses.size();
        int t = s+1;

        for(int i=0;i<men.size();i++){
            add_edge(E,s,i,1,0);
            for(int j=0;j<houses.size();j++){
                int dist = abs(men[i].first - houses[j].first)
                    + abs(men[i].second - houses[j].second);
                add_edge(E,i,men.size()+j,1,dist);
            }
        }

        for(int i=0;i<houses.size();i++){
            add_edge(E,men.size()+i,t,1,0);
        }

        cout << min_cost_flow(E,s,t,men.size()) << endl;
    }
    return 0;
}
```

幾何

基本要素

```
#include <complex>
using namespace std;

typedef complex<double> point;
bool operator < (const point &lhs,const point &rhs){
    if(real(lhs) == real(rhs)){
        return imag(lhs) < imag(rhs);
    }else{
        return real(lhs) < real(rhs);
    }
}

double cross(const point &a,const point &b){
    return imag(conj(a)*b);
}
double dot(const point &a,const point &b){
    return real(conj(a)*b);
}
```

```
// 点の進行方向
int ccw(point a,point b,point c){
    b -= a;c -= a;
    if(cross(b,c) > 0) return +1;    // counter clockwise
    if(cross(b,c) < 0) return -1;    // clockwise
    if(dot(b,c) < 0) return +2;      // c -- a -- b
    if(norm(b) < norm(c)) return -2; // a -- b -- c
    return 0;
}

struct line : vector<point> {
    line(const point &a,const point &b){
        push_back(a);push_back(b);
    }
};

struct circle {
    point center;double r;
    circle(const point &center,double r) : center(center),r(r) {}
};
```

ゲーム

Nim

いくつかのコインの山がある。この中からプレイヤーは山を一つ選び、1個以上の任意の 数のコインを取る。最後のコインを取ったプレイヤーが勝ちである。この問題に 対しては以下のことが知られている。すべての山のxorをとったとき、それが0で あるとき、後攻の勝ち、それ以外のときは先攻の勝ち。

```
#include <iostream>
#include <vector>

using namespace std;

int main(){
    int xor_sum = 0;
    vector<int> coins = {1,2,4,5,6};
    for(size_t i=0;i<coins.size();i++){
        xor_sum = xor_sum ^ coins[i];
    }
    if(xor_sum != 0) cout << "First␣Player␣Win" << endl;
    else cout << "Second␣Player␣Win" << endl;
}
```

いろんなデータ

階乗

N	$N!$
0	1
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800

数単位変換

N	日本語	英語
10^0	一	one
10^1	十	ten
10^2	百	hundred
10^3	千	thousand
10^4	万	ten thousand
10^5	十万	hundred thousand
10^6	百万	million
10^7	千万	ten million
10^8	億	hundred million
10^9	十億	billion
10^{10}	百億	ten billion
10^{11}	千億	hundred billion

bit

N	2^N	備考
0	1	
1	2	boolの大きさ
2	4	
3	8	
4	16	
7	128	charの最大値+1
8	256	unsigned charの最大値+1
16	65,534	
24	16,777,216	
31	2,147,483,648	intの最大値+1(about 2*10^9)
32	4,294,967,296	unsigned intの最大値+1
52	4,503,599,627,370,496	doubleのprecision
63	9,223,372,036,854,775,808	long longの最大値+1
64	18,446,744,073,709,551,616	unsigned long longの最大値+1(about 10^19)

アスキーコード

Char	Dec	Oct	Hex
(nul)	0	0000	0x00
(soh)	1	0001	0x01
(stx)	2	0002	0x02
(etx)	3	0003	0x03
(eot)	4	0004	0x04
(enq)	5	0005	0x05
(ack)	6	0006	0x06
(bel)	7	0007	0x07
(bs)	8	0010	0x08

(ht)	9	0011	0x09
(nl)	10	0012	0x0a
(vt)	11	0013	0x0b
(np)	12	0014	0x0c
(cr)	13	0015	0x0d
(so)	14	0016	0x0e
(si)	15	0017	0x0f
(dle)	16	0020	0x10
(dc1)	17	0021	0x11
(dc2)	18	0022	0x12
(dc3)	19	0023	0x13
(dc4)	20	0024	0x14
(nak)	21	0025	0x15
(syn)	22	0026	0x16
(etb)	23	0027	0x17
(can)	24	0030	0x18
(em)	25	0031	0x19
(sub)	26	0032	0x1a
(esc)	27	0033	0x1b
(fs)	28	0034	0x1c
(gs)	29	0035	0x1d
(rs)	30	0036	0x1e
(us)	31	0037	0x1f
(sp)	32	0040	0x20
!	33	0041	0x21
“	34	0042	0x22
#	35	0043	0x23
$	36	0044	0x24
%	37	0045	0x25
&	38	0046	0x26
’	39	0047	0x27

(	40	0050	0x28
)	41	0051	0x29
*	42	0052	0x2a
+	43	0053	0x2b
,	44	0054	0x2c
-	45	0055	0x2d
.	46	0056	0x2e
/	47	0057	0x2f
0	48	0060	0x30
1	49	0061	0x31
2	50	0062	0x32
3	51	0063	0x33
4	52	0064	0x34
5	53	0065	0x35
6	54	0066	0x36
7	55	0067	0x37
8	56	0070	0x38
9	57	0071	0x39
:	58	0072	0x3a
;	59	0073	0x3b
<	60	0074	0x3c
=	61	0075	0x3d
>	62	0076	0x3e
?	63	0077	0x3f
@	64	0100	0x40
A	65	0101	0x41
B	66	0102	0x42
C	67	0103	0x43
D	68	0104	0x44
E	69	0105	0x45
F	70	0106	0x46

G	71	0107	0x47
H	72	0110	0x48
I	73	0111	0x49
J	74	0112	0x4a
K	75	0113	0x4b
L	76	0114	0x4c
M	77	0115	0x4d
N	78	0116	0x4e
O	79	0117	0x4f
P	80	0120	0x50
Q	81	0121	0x51
R	82	0122	0x52
S	83	0123	0x53
T	84	0124	0x54
U	85	0125	0x55
V	86	0126	0x56
W	87	0127	0x57
X	88	0130	0x58
Y	89	0131	0x59
Z	90	0132	0x5a
[	91	0133	0x5b
	92	0134	0x5c
]	93	0135	0x5d
^	94	0136	0x5e
_	95	0137	0x5f
‘	96	0140	0x60
a	97	0141	0x61
b	98	0142	0x62
c	99	0143	0x63
d	100	0144	0x64
e	101	0145	0x65

f	102	0146	0x66
g	103	0147	0x67
h	104	0150	0x68
i	105	0151	0x69
j	106	0152	0x6a
k	107	0153	0x6b
l	108	0154	0x6c
m	109	0155	0x6d
n	110	0156	0x6e
o	111	0157	0x6f
p	112	0160	0x70
q	113	0161	0x71
r	114	0162	0x72
s	115	0163	0x73
t	116	0164	0x74
u	117	0165	0x75
v	118	0166	0x76
w	119	0167	0x77
x	120	0170	0x78
y	121	0171	0x79
z	122	0172	0x7a
{	123	0173	0x7b
|	124	0174	0x7c
}	125	0175	0x7d
~	126	0176	0x7e
(del)	127	0177	0x7f