

# Contents

## 1 Reminder

- 1.1 Bug List . . . . .
- 1.2 OwO . . . . .

## 2 Basic

- 2.1 Default . . . . .
- 2.2 Vimrc . . . . .
- 2.3 Run.sh . . . . .
- 2.4 Stress . . . . .
- 2.5 PBDS . . . . .
- 2.6 Random . . . . .

## 3 Python

- 3.1 I/O . . . . .
- 3.2 Decimal . . . . .

## 4 Data Structure

- 4.1 Heavy Light Decomposition . . . . .
- 4.2 Skew Heap . . . . .
- 4.3 Leftist Heap . . . . .
- 4.4 Persistent Treap . . . . .
- 4.5 Li Chao Tree . . . . .

## 5 DP

- 5.1 Aliens . . . . .

## 6 Graph

- 6.1 Bellman-Ford + SPFA . . . . .
- 6.2 BCC - AP . . . . .
- 6.3 BCC - Bridge . . . . .
- 6.4 SCC - Tarjan . . . . .
- 6.5 Eulerian Path - Undir . . . . .
- 6.6 Eulerian Path - Dir . . . . .
- 6.7 Hamilton Path . . . . .
- 6.8 Kth Shortest Path . . . . .

## 7 String

- 7.1 Rolling Hash . . . . .
- 7.2 Trie . . . . .
- 7.3 KMP . . . . .
- 7.4 Z Value . . . . .
- 7.5 Manacher . . . . .
- 7.6 Suffix Array . . . . .
- 7.7 SA-IS . . . . .
- 7.8 Minimum Rotation . . . . .
- 7.9 Aho Corasick . . . . .

## 8 Geometry

- 8.1 Basic Operations . . . . .
- 8.2 InPoly . . . . .
- 8.3 Sort by Angle . . . . .
- 8.4 Line Intersect Check . . . . .
- 8.5 Line Intersection . . . . .
- 8.6 Convex Hull . . . . .
- 8.7 Polygon Area . . . . .
- 8.8 Pick's Theorem . . . . .
- 8.9 Minimum Enclosing Circle . . . . .
- 8.10 Closest Pair of Points . . . . .
- 8.11 PolyUnion . . . . .
- 8.12 Minkowski Sum . . . . .

## 9 Number Theory

- 9.1 Pollard's rho . . . . .
- 9.2 Miller Rabin . . . . .
- 9.3 Fast Power . . . . .
- 9.4 Extend GCD . . . . .
- 9.5 Mu + Phi . . . . .
- 9.6 Other Formulas . . . . .

## 10 Linear Algebra

- 10.1 Gaussian-Jordan Elimination . . . . .
- 10.2 Determinant . . . . .

## 11 Flow / Matching

- 11.1 Dinic . . . . .
- 11.2 ISAP . . . . .
- 11.3 MCMF . . . . .
- 11.4 Hopcroft-Karp . . . . .
- 11.5 Cover / Independent Set . . . . .
- 11.6 KM . . . . .

## 12 Combinatorics

- 12.1 Catalan Number . . . . .
- 12.2 Burnside's Lemma . . . . .

## 13 Special Numbers

- 13.1 Fibonacci Series . . . . .
- 13.2 Prime Numbers . . . . .

# 1 Reminder

## 1.1 Bug List

- 沒開 long long
- 陣列戳出界／陣列開不夠大
- 寫好的函式忘記呼叫
- 變數打錯
- 0-base / 1-base
- 忘記初始化
- == 打成 =
- <= 打成 <+
- dp[i] 從 dp[i-1] 轉移時忘記特判 i > 0
- std::sort 比較運算子寫成 < 或是讓 = 的情況為 true
- 漏 case
- 線段樹改值懶標初始值不能設為 0
- DFS 的時候不小心覆寫到全域變數

## 1.2 OwO

- Enjoy The Game!

# 2 Basic

## 2.1 Default

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4 using ll = long long;
5 using pii = pair<int, int>;
6 using pll = pair<ll, ll>;
7
8 #define endl '\n'
9
10 #define F first
11 #define S second
12 #define ep emplace
13 #define pb push_back
14 #define eb emplace_back
15 #define ALL(x) x.begin(), x.end()
16 #define SZ(x) (int)x.size()
17
18 namespace{
19     const int INF = 0x3f3f3f3f;
20     const ll LINF = 0x3f3f3f3f3f3f3f3f;
21
22     template<typename T> using V=vector<T>;
23     template<typename T1,typename T2=T1> using P = pair<T1,
24         T2>;
25
26     void _debug() {}
27     template<typename A,typename... B> void _debug(A a,B...
28         b){
29         cerr<<a<<' ',_debug(b...);
30     }
31     #define debug(...) cerr<<#__VA_ARGS__<<" ",_debug(
32         __VA_ARGS__),cerr<<endl;
33     template<typename T>
34     ostream& operator<<(ostream& os,const vector<T>& v){
35         for(const auto& i:v)
36             os<<i<<' ';
37         return os;
38     }
39 }
40
41 /*-----*/
42
43 const ll MOD = 1e9 + 7;
44 const int maxn = 2e5 + 5;
45
46 void init() {
47     ;
48 }
49
50 void solve() {
51     ;

```

```

49 }
50
51 /*
52
53
54 */
55
56 signed main() {
57     cin.tie(0), ios::sync_with_stdio(0);
58
59     int T = 1;
60     // cin >> T;
61     while (T--) {
62         init();
63         solve();
64     }
65
66     return 0;
67 }

```

## 2.2 Vimrc

```

1 syn on
2 se ai nu rnu ru cul mouse=a
3 se cin et ts=4 sw=4 sts=4
4 colo desert
5 set autochdir
6 no <F5> :!./a.out<CR>
7 no <F9> :!~/run.sh %:p:h %:p:t<CR>

```

## 2.3 Run.sh

```

1 clear
2 echo File Location: $1
3 echo File Name: $2
4 echo =====
5 echo Start compiling \"$2\"...
6 echo
7 g++ $1/$2 -std=c++20 -Ofast -Wall -Wextra -g -fsanitize
   =address,undefined -o$1/a.out
8 if [ \"$?\" -ne 0 ]
9 then
10     exit 1
11 fi
12 echo
13 echo Done compiling...
14 echo =====
15 echo Input file:
16 echo -----
17 cat $1/input
18 echo =====
19 declare startTime=`date +%s%N`
20 $1/a.out < $1/input > $1/output
21 declare endTime=`date +%s%N`
22 delta=`expr $endTime - $startTime`
23 delta=`expr $delta / 1000000`
24 echo "Program ended in $delta ms with the return value
   $?"
25 cat $1/output

```

## 2.4 Stress

```

1 g++ gen.cpp -o gen.out
2 g++ ac.cpp -o ac.out
3 g++ wa.cpp -o wa.out
4 for ((i=0;;i++))
5 do
6     echo "$i"
7     ./gen.out > in.txt
8     ./ac.out < in.txt > ac.txt
9     ./wa.out < in.txt > wa.txt
10    diff ac.txt wa.txt || break
11 done

```

## 2.5 PBDS

```

1 #include <bits/extc++.h>
2 using namespace __gnu_pbds;

```

```

3
4 // map
5 tree<int, int, less<>, rb_tree_tag,
   tree_order_statistics_node_update> tr;
6 tr.order_of_key(element);
7 tr.find_by_order(rank);
8
9 // set
10 tree<int, null_type, less<>, rb_tree_tag,
   tree_order_statistics_node_update> tr;
11 tr.order_of_key(element);
12 tr.find_by_order(rank);
13
14 // priority queue
15 __gnu_pbds::priority_queue<int, less<int> > big_q; //
   Big First
16 __gnu_pbds::priority_queue<int, greater<int> > small_q;
   // Small First
17 q1.join(q2); // join

```

## 2.6 Random

```

1 mt19937 gen(chrono::steady_clock::now().
   time_since_epoch().count());
2 uniform_int_distribution<int> dis(1, 100);
3 cout << dis(gen) << endl;
4 shuffle(v.begin(), v.end(), gen);

```

# 3 Python

## 3.1 I/O

```

1 import sys
2 input = sys.stdin.readline
3
4 # Input
5 def readInt():
6     return int(input())
7 def readList():
8     return list(map(int, input().split()))
9 def readStr():
10    s = input()
11    return list(s[:len(s) - 1])
12 def readVars():
13    return map(int, input().split())
14
15 # Output
16 sys.stdout.write(string)
17
18 # faster
19 def main():
20     pass
21 main()

```

## 3.2 Decimal

```

1 from decimal import *
2 getcontext().prec = 2500000
3 getcontext().Emax = 2500000
4 a,b = Decimal(input()),Decimal(input())
5 a*=b
6 print(a)

```

# 4 Data Structure

## 4.1 Heavy Light Decomposition

```

1 constexpr int maxn=2e5+5;
2 int arr[(maxn+1)<<2];
3 #define m ((l+r)>>1)
4 void build(V<int>& v, int i=1, int l=0, int r=maxn){
5     if((int)v.size()<=1) return;
6     if(r-l==1){arr[i]=v[l];return;}
7     build(v, i<<1, l, m), build(v, i<<1|1, m, r);
8     arr[i]=max(arr[i<<1], arr[i<<1|1]);
9 }

```

```

10 void modify(int p,int k,int i=1,int l=0,int r=maxn){
11     if(p<l||r<=p) return;
12     if(r-l==1){arr[i]=k;return;}
13     if(p<m) modify(p,k,i<<1,l,m);
14     else modify(p,k,i<<1|1,m,r);
15     arr[i]=max(arr[i<<1],arr[i<<1|1]);
16 }
17 int query(int ql,int qr,int i=1,int l=0,int r=maxn){
18     if(qr<=l||r<=ql) return 0;
19     if(ql<=l&&r<=qr) return arr[i];
20     if(qr<=m) return query(ql,qr,i<<1,l,m);
21     if(m<=ql) return query(ql,qr,i<<1|1,m,r);
22     return max(query(ql,qr,i<<1,l,m),query(ql,qr,i
        <<1|1,m,r));
23 }
24 #undef m
25 inline void solve(){
26     int n,q;cin>>n>>q;
27     V<int> v(n);
28     for(auto& i:v)
29         cin>>i;
30     V<V<int>> e(n);
31     for(int i=1;i<n;i++){
32         int a,b;cin>>a>>b,a--,b--;
33         e[a].emplace_back(b);
34         e[b].emplace_back(a);
35     }
36     V<int> d(n,0),f(n,0),sz(n,1),son(n,-1);
37     F<void(int,int)> dfs1=
38     [&](int x,int pre){
39         for(auto i:e[x]) if(i!=pre){
40             d[i]=d[x]+1,f[i]=x;
41             dfs1(i,x),sz[x]+=sz[i];
42             if(!~son[x]||sz[son[x]]<sz[i])
43                 son[x]=i;
44         }
45     };dfs1(0,0);
46     V<int> top(n,0),dfn(n,-1),rnk(n,0);
47     F<void(int,int)> dfs2=
48     [&](int x,int t){
49         static int cnt=0;
50         dfn[x]=cnt++,rnk[dfn[x]]=x,top[x]=t;
51         if(!~son[x]) return;
52         dfs2(son[x],t);
53         for(auto i:e[x])
54             if(!~dfn[i]) dfs2(i,i);
55     };dfs2(0,0);
56     V<int> dfnv(n);
57     for(int i=0;i<n;i++)
58         dfnv[dfn[i]]=v[i];
59     build(dfnv);
60     while(q--){
61         int op,a,b;cin>>op>>a>>b;
62         switch(op){
63             case 1:{
64                 modify(dfn[a-1],b);
65             }break;
66             case 2:{
67                 a--,b--;
68                 int ans=0;
69                 while(top[a]!=top[b]){
70                     if(d[top[a]]>d[top[b]]) swap(a,b);
71                     ans=max(ans,query(dfn[top[b]],dfn[b]+1)
72                         );
73                     b=f[top[b]];
74                 }
75                 if(dfn[a]>dfn[b]) swap(a,b);
76                 ans=max(ans,query(dfn[a],dfn[b]+1));
77                 cout<<ans<<endl;
78             }break;
79         }
80     }

```

## 4.2 Skew Heap

```

1 struct node{
2     node *l,*r;
3     int v;
4     node(int x):v(x){
5         l=r=nullptr;

```

```

6     }
7 };
8 node* merge(node* a,node* b){
9     if(!a||!b) return a?:b;
10    // min heap
11    if(a->v>b->v) swap(a,b);
12    a->r=merge(a->r,b);
13    swap(a->l,a->r);
14    return a;
15 }

```

## 4.3 Leftist Heap

```

1 struct node{
2     node *l,*r;
3     int d, v;
4     node(int x):d(1),v(x){
5         l=r=nullptr;
6     }
7 };
8 static inline int d(node* x){return x?x->d:0;}
9 node* merge(node* a,node* b){
10    if(!a||!b) return a?:b;
11    // min heap
12    if(a->v>b->v) swap(a,b);
13    a->r=merge(a->r,b);
14    if(d(a->l)<d(a->r))
15        swap(a->l,a->r);
16    a->d=d(a->r)+1;
17    return a;
18 }

```

## 4.4 Persistent Treap

```

1 struct node {
2     node *l, *r;
3     char c; int v, sz;
4     node(char x = '$'): c(x), v(mt()), sz(1) {
5         l = r = nullptr;
6     }
7     node(node* p) { *this = *p; }
8     void pull() {
9         sz = 1;
10        for (auto i : {l, r})
11            if (i) sz += i->sz;
12    }
13 } arr[maxn], *ptr = arr;
14 inline int size(node* p) {return p ? p->sz : 0;}
15 node* merge(node* a, node* b) {
16     if (!a || !b) return a ? : b;
17     if (a->v < b->v) {
18         node* ret = new(ptr++) node(a);
19         ret->r = merge(ret->r, b), ret->pull();
20         return ret;
21     }
22     else {
23         node* ret = new(ptr++) node(b);
24         ret->l = merge(a, ret->l), ret->pull();
25         return ret;
26     }
27 }
28 P<node*> split(node* p, int k) {
29     if (!p) return {nullptr, nullptr};
30     if (k >= size(p->l) + 1) {
31         auto [a, b] = split(p->r, k - size(p->l) - 1);
32         node* ret = new(ptr++) node(p);
33         ret->r = a, ret->pull();
34         return {ret, b};
35     }
36     else {
37         auto [a, b] = split(p->l, k);
38         node* ret = new(ptr++) node(p);
39         ret->l = b, ret->pull();
40         return {a, ret};
41     }
42 }

```

## 4.5 Li Chao Tree

```

1 constexpr int maxn = 5e4 + 5;
2 struct line {
3     ld a, b;
4     ld operator()(ld x) {return a * x + b;}
5 } arr[(maxn + 1) << 2];
6 bool operator<(line a, line b) {return a.a < b.a;}
7 #define m ((l+r)>>1)
8 void insert(line x, int i = 1, int l = 0, int r = maxn) {
9     {
10         if (r - l == 1) {
11             if (x(l) > arr[i](l))
12                 arr[i] = x;
13             return;
14         }
15         line a = max(arr[i], x), b = min(arr[i], x);
16         if (a(m) > b(m))
17             arr[i] = a, insert(b, i << 1, l, m);
18         else
19             arr[i] = b, insert(a, i << 1 | 1, m, r);
20     }
21 ld query(int x, int i = 1, int l = 0, int r = maxn) {
22     if (x < l || r <= x) return -numeric_limits<ld>::max();
23     if (r - l == 1) return arr[i](x);
24     return max({arr[i](x), query(x, i << 1, l, m), query(
25         x, i << 1 | 1, m, r)});
26 }
27 #undef m

```

## 5 DP

### 5.1 Aliens

```

1 int n; ll k;
2 vector<ll> a;
3 vector<pll> dp[2];
4 void init() {
5     cin >> n >> k;
6     Each(i, dp) i.clear(), i.resize(n);
7     a.clear(); a.resize(n);
8     Each(i, a) cin >> i;
9 }
10 pll calc(ll p) {
11     dp[0][0] = mp(0, 0);
12     dp[1][0] = mp(-a[0], 0);
13     FOR(i, 1, n, 1) {
14         if (dp[0][i-1].F > dp[1][i-1].F + a[i] - p) {
15             dp[0][i] = dp[0][i-1];
16         } else if (dp[0][i-1].F < dp[1][i-1].F + a[i] - p) {
17             {
18                 dp[0][i] = mp(dp[1][i-1].F + a[i] - p, dp[1][i-1].S+1);
19             } else {
20                 dp[0][i] = mp(dp[0][i-1].F, min(dp[0][i-1].S, dp[1][i-1].S+1));
21             }
22         } else if (dp[0][i-1].F - a[i] > dp[1][i-1].F) {
23             dp[1][i] = mp(dp[0][i-1].F - a[i], dp[0][i-1].S);
24         } else if (dp[0][i-1].F - a[i] < dp[1][i-1].F) {
25             dp[1][i] = dp[1][i-1];
26         } else {
27             dp[1][i] = mp(dp[1][i-1].F, min(dp[0][i-1].S, dp[1][i-1].S));
28         }
29     }
30     return dp[0][n-1];
31 }
32 void solve() {
33     ll l = 0, r = 1e7;
34     pll res = calc(0);
35     if (res.S <= k) return cout << res.F << endl, void();
36     while (l < r) {
37         ll mid = (l+r)>>1;
38         res = calc(mid);
39         if (res.S <= k) r = mid;
40         else l = mid+1;
41     }
42     res = calc(l);
43     cout << res.F + k*l << endl;
44 }

```

## 6 Graph

### 6.1 Bellman-Ford + SPFA

```

1 int n, m;
2
3 // Graph
4 vector<vector<pair<int, ll> > > g;
5 vector<ll> dis;
6 vector<bool> negCycle;
7
8 // SPFA
9 vector<int> rlx;
10 queue<int> q;
11 vector<bool> inq;
12 vector<int> pa;
13 void SPFA(vector<int>& src) {
14     dis.assign(n+1, LINF);
15     negCycle.assign(n+1, false);
16     rlx.assign(n+1, 0);
17     while (!q.empty()) q.pop();
18     inq.assign(n+1, false);
19     pa.assign(n+1, -1);
20
21     for (auto& s : src) {
22         dis[s] = 0;
23         q.push(s); inq[s] = true;
24     }
25
26     while (!q.empty()) {
27         int u = q.front();
28         q.pop(); inq[u] = false;
29         if (rlx[u] >= n) {
30             negCycle[u] = true;
31         }
32         else for (auto& e : g[u]) {
33             int v = e.first;
34             ll w = e.second;
35             if (dis[v] > dis[u] + w) {
36                 dis[v] = dis[u] + w;
37                 rlx[v] = rlx[u] + 1;
38                 pa[v] = u;
39                 if (!inq[v]) {
40                     q.push(v);
41                     inq[v] = true;
42                 }
43             }
44         }
45     }
46
47 // Bellman-Ford
48 queue<int> q;
49 vector<int> pa;
50 void BellmanFord(vector<int>& src) {
51     dis.assign(n+1, LINF);
52     negCycle.assign(n+1, false);
53     pa.assign(n+1, -1);
54
55     for (auto& s : src) dis[s] = 0;
56
57     for (int rlx = 1; rlx <= n; rlx++) {
58         for (int u = 1; u <= n; u++) {
59             if (dis[u] == LINF) continue; // Important
60             !!
61             for (auto& e : g[u]) {
62                 int v = e.first; ll w = e.second;
63                 if (dis[v] > dis[u] + w) {
64                     dis[v] = dis[u] + w;
65                     pa[v] = u;
66                     if (rlx == n) negCycle[v] = true;
67                 }
68             }
69         }
70     }
71
72 // Negative Cycle Detection
73 void NegCycleDetect() {
74     /* No Neg Cycle: NO
75     Exist Any Neg Cycle:
76     YES
77     v0 v1 v2 ... vk v0 */
78
79     vector<int> src;
80     for (int i = 1; i <= n; i++)
81         src.emplace_back(i);
82 }

```

```

77 SPFA(src);
78 // BellmanFord(src);
79
80 int ptr = -1;
81 for (int i = 1; i <= n; i++) if (negCycle[i])
82     { ptr = i; break; }
83
84 if (ptr == -1) { return cout << "NO" << endl, void
85     (); }
86
87 cout << "YES\n";
88 vector<int> ans;
89 vector<bool> vis(n+1, false);
90
91 while (true) {
92     ans.emplace_back(ptr);
93     if (vis[ptr]) break;
94     vis[ptr] = true;
95     ptr = pa[ptr];
96 }
97 reverse(ans.begin(), ans.end());
98
99 vis.assign(n+1, false);
100 for (auto& x : ans) {
101     cout << x << ' ';
102     if (vis[x]) break;
103     vis[x] = true;
104 }
105 cout << endl;
106 }
107
108 // Distance Calculation
109 void calcDis(int s) {
110     vector<int> src;
111     src.emplace_back(s);
112     SPFA(src);
113     // BellmanFord(src);
114
115     while (!q.empty()) q.pop();
116     for (int i = 1; i <= n; i++)
117         if (negCycle[i]) q.push(i);
118
119     while (!q.empty()) {
120         int u = q.front(); q.pop();
121         for (auto& e : g[u]) {
122             int v = e.first;
123             if (!negCycle[v]) {
124                 q.push(v);
125                 negCycle[v] = true;
126             }
127         }
128     }
129 }

```

## 6.2 BCC - AP

```

1 int n, m;
2 int low[maxn], dfn[maxn], instp;
3 vector<int> E, g[maxn];
4 bitset<maxn> isap;
5 bitset<maxn> vis;
6 stack<int> stk;
7 int bccnt;
8 vector<int> bcc[maxn];
9 inline void popout(int u) {
10     bccnt++;
11     bcc[bccnt].emplace_back(u);
12     while (!stk.empty()) {
13         int v = stk.top();
14         if (u == v) break;
15         stk.pop();
16         bcc[bccnt].emplace_back(v);
17     }
18 }
19 void dfs(int u, bool rt = 0) {
20     stk.push(u);
21     low[u] = dfn[u] = ++instp;
22     int kid = 0;
23     Each(e, g[u]) {
24         if (vis[e]) continue;
25         vis[e] = true;
26         int v = E[e]^u;
27         if (!dfn[v]) {

```

```

28 // tree edge
29     kid++; dfs(v);
30     low[u] = min(low[u], low[v]);
31     if (!rt && low[v] >= dfn[u]) {
32         // bcc found: u is ap
33         isap[u] = true;
34         popout(u);
35     }
36 } else {
37     // back edge
38     low[u] = min(low[u], dfn[v]);
39 }
40 }
41 // special case: root
42 if (rt) {
43     if (kid > 1) isap[u] = true;
44     popout(u);
45 }
46 }
47 void init() {
48     cin >> n >> m;
49     fill(low, low+maxn, INF);
50     REP(i, m) {
51         int u, v;
52         cin >> u >> v;
53         g[u].emplace_back(i);
54         g[v].emplace_back(i);
55         E.emplace_back(u^v);
56     }
57 }
58 void solve() {
59     FOR(i, 1, n+1, 1) {
60         if (!dfn[i]) dfs(i, true);
61     }
62     vector<int> ans;
63     int cnt = 0;
64     FOR(i, 1, n+1, 1) {
65         if (isap[i]) cnt++, ans.emplace_back(i);
66     }
67     cout << cnt << endl;
68     Each(i, ans) cout << i << ' ';
69     cout << endl;
70 }

```

## 6.3 BCC - Bridge

```

1 int n, m;
2 vector<int> g[maxn], E;
3 int low[maxn], dfn[maxn], instp;
4 int bccnt, bccid[maxn];
5 stack<int> stk;
6 bitset<maxn> vis, isbrg;
7 void init() {
8     cin >> n >> m;
9     REP(i, m) {
10         int u, v;
11         cin >> u >> v;
12         E.emplace_back(u^v);
13         g[u].emplace_back(i);
14         g[v].emplace_back(i);
15     }
16     fill(low, low+maxn, INF);
17 }
18 void popout(int u) {
19     bccnt++;
20     while (!stk.empty()) {
21         int v = stk.top();
22         if (v == u) break;
23         stk.pop();
24         bccid[v] = bccnt;
25     }
26 }
27 void dfs(int u) {
28     stk.push(u);
29     low[u] = dfn[u] = ++instp;
30
31     Each(e, g[u]) {
32         if (vis[e]) continue;
33         vis[e] = true;
34
35         int v = E[e]^u;

```

```

36     if (dfn[v]) {
37         // back edge
38         low[u] = min(low[u], dfn[v]);
39     } else {
40         // tree edge
41         dfs(v);
42         low[u] = min(low[u], low[v]);
43         if (low[v] == dfn[v]) {
44             isbrg[e] = true;
45             popout(u);
46         }
47     }
48 }
49 }
50 void solve() {
51     FOR(i, 1, n+1, 1) {
52         if (!dfn[i]) dfs(i);
53     }
54     vector<pii> ans;
55     vis.reset();
56     FOR(u, 1, n+1, 1) {
57         Each(e, g[u]) {
58             if (!isbrg[e] || vis[e]) continue;
59             vis[e] = true;
60             int v = E[e]^u;
61             ans.emplace_back(mp(u, v));
62         }
63     }
64     cout << (int)ans.size() << endl;
65     Each(e, ans) cout << e.F << ' ' << e.S << endl;
66 }

```

## 6.4 SCC - Tarjan

```

1 // 2-SAT
2 vector<int> E, g[maxn]; // 1~n, n+1~2n
3 int low[maxn], in[maxn], instp;
4 int sccnt, sccid[maxn];
5
6 stack<int> stk;
7 bitset<maxn> ins, vis;
8
9 int n, m;
10
11 void init() {
12     cin >> m >> n;
13     E.clear();
14     fill(g, g+maxn, vector<int>());
15     fill(low, low+maxn, INF);
16     memset(in, 0, sizeof(in));
17     instp = 1;
18     sccnt = 0;
19     memset(sccid, 0, sizeof(sccid));
20     ins.reset();
21     vis.reset();
22 }
23
24 inline int no(int u) {
25     return (u > n ? u-n : u+n);
26 }
27
28 int ecnt = 0;
29 inline void clause(int u, int v) {
30     E.pb(no(u)^v);
31     g[no(u)].pb(ecnt++);
32     E.pb(no(v)^u);
33     g[no(v)].pb(ecnt++);
34 }
35
36 void dfs(int u) {
37     in[u] = instp++;
38     low[u] = in[u];
39     stk.push(u);
40     ins[u] = true;
41
42     Each(e, g[u]) {
43         if (vis[e]) continue;
44         vis[e] = true;
45
46         int v = E[e]^u;
47         if (ins[v]) low[u] = min(low[u], in[v]);

```

```

48     else if (!in[v]) {
49         dfs(v);
50         low[u] = min(low[u], low[v]);
51     }
52 }
53
54 if (low[u] == in[u]) {
55     sccnt++;
56     while (!stk.empty()) {
57         int v = stk.top();
58         stk.pop();
59         ins[v] = false;
60         sccid[v] = sccnt;
61         if (u == v) break;
62     }
63 }
64 }
65
66 int main() {
67     WiWiHorz
68     init();
69
70     REP(i, m) {
71         char su, sv;
72         int u, v;
73         cin >> su >> u >> sv >> v;
74         if (su == '-') u = no(u);
75         if (sv == '-') v = no(v);
76         clause(u, v);
77     }
78
79     FOR(i, 1, 2*n+1, 1) {
80         if (!in[i]) dfs(i);
81     }
82
83     FOR(u, 1, n+1, 1) {
84         int du = no(u);
85         if (sccid[u] == sccid[du]) {
86             return cout << "IMPOSSIBLE\n", 0;
87         }
88     }
89
90     FOR(u, 1, n+1, 1) {
91         int du = no(u);
92         cout << (sccid[u] < sccid[du] ? '+' : '-') << '
93             ' << ' ';
94     }
95     cout << endl;
96
97     return 0;
98 }

```

## 6.5 Eulerian Path - Undir

```

1 // from 1 to n
2 #define gg return cout << "IMPOSSIBLE\n", void();
3
4 int n, m;
5 vector<int> g[maxn];
6 bitset<maxn> inodd;
7
8 void init() {
9     cin >> n >> m;
10    inodd.reset();
11    for (int i = 0; i < m; i++) {
12        int u, v; cin >> u >> v;
13        inodd[u] = inodd[u] ^ true;
14        inodd[v] = inodd[v] ^ true;
15        g[u].emplace_back(v);
16        g[v].emplace_back(u);
17    }
18    stack<int> stk;
19    void dfs(int u) {
20        while (!g[u].empty()) {
21            int v = g[u].back();
22            g[u].pop_back();
23            dfs(v);
24        }
25        stk.push(u);

```

## 6.6 Eulerian Path - Dir

```

1 // from node 1 to node n
2 #define gg return cout << "IMPOSSIBLE\n", 0
3
4 int n, m;
5 vector<int> g[maxn];
6 stack<int> stk;
7 int in[maxn], out[maxn];
8
9 void init() {
10     cin >> n >> m;
11     for (int i = 0; i < m; i++) {
12         int u, v; cin >> u >> v;
13         g[u].emplace_back(v);
14         out[u]++, in[v]++;
15     }
16     for (int i = 1; i <= n; i++) {
17         if (i == 1 && out[i]-in[i] != 1) gg;
18         if (i == n && in[i]-out[i] != 1) gg;
19         if (i != 1 && i != n && in[i] != out[i]) gg;
20     }
21     void dfs(int u) {
22         while (!g[u].empty()) {
23             int v = g[u].back();
24             g[u].pop_back();
25             dfs(v);
26         }
27         stk.push(u);
28     }
29     void solve() {
30         dfs(1)
31         for (int i = 1; i <= n; i++)
32             if ((int)g[i].size()) gg;
33         while (!stk.empty()) {
34             int u = stk.top();
35             stk.pop();
36             cout << u << ' ';
37         }
38     }
39 }

```

## 6.7 Hamilton Path

```

1 // top down DP
2 // Be Aware Of Multiple Edges
3 int n, m;
4 ll dp[maxn][1<<maxn];
5 int adj[maxn][maxn];
6
7 void init() {
8     cin >> n >> m;
9     fill(dp[0], dp[maxn-1]+(1<<maxn), -1);
10 }
11
12 void DP(int i, int msk) {
13     if (dp[i][msk] != -1) return;
14     dp[i][msk] = 0;
15     REP(j, n) if (j != i && (msk & (1<<j)) && adj[j][i]) {
16         int sub = msk ^ (1<<i);
17         if (dp[j][sub] == -1) DP(j, sub);
18         dp[i][msk] += dp[j][sub] * adj[j][i];
19         if (dp[i][msk] >= MOD) dp[i][msk] %= MOD;
20     }
21 }
22
23 int main() {
24     WiwiHorz
25     init();
26
27     REP(i, m) {
28         int u, v;
29         cin >> u >> v;
30         if (u == v) continue;
31         adj[--u][--v]++;
32     }
33
34     dp[0][1] = 1;
35     FOR(i, 1, n, 1) {
36         dp[i][1] = 0;
37         dp[i][1|(1<<i)] = adj[0][i];
38     }
39 }

```

```

39     }
40     FOR(msk, 1, (1<<n), 1) {
41         if (msk == 1) continue;
42         dp[0][msk] = 0;
43     }
44
45     DP(n-1, (1<<n)-1);
46     cout << dp[n-1][(1<<n)-1] << endl;
47
48     return 0;
49 }
50 }

```

## 6.8 Kth Shortest Path

```

1 // time: O(|E| \lg |E|+|V| \lg |V|+K)
2 // memory: O(|E| \lg |E|+|V|)
3 struct KSP{ // 1-base
4     struct nd{
5         int u,v; ll d;
6         nd(int ui=0,int vi=0,ll di=INF){ u=ui; v=vi; d=di; }
7     };
8     struct heap{ nd* edge; int dep; heap* chd[4]; };
9     static int cmp(heap* a,heap* b)
10     { return a->edge->d > b->edge->d; }
11     struct node{
12         int v; ll d; heap* H; nd* E;
13         node(){ }
14         node(ll _d,int _v,nd* _E){ d=_d; v=_v; E=_E; }
15         node(heap* _H,ll _d){ H=_H; d=_d; }
16         friend bool operator<(node a,node b)
17         { return a.d>b.d; }
18     };
19     int n,k,s,t,dst[N]; nd *nxt[N];
20     vector<nd*> g[N],rg[N]; heap *nullNd,*head[N];
21     void init(int _n,int _k,int _s,int _t){
22         n=_n; k=_k; s=_s; t=_t;
23         for(int i=1;i<=n;i++){
24             g[i].clear(); rg[i].clear();
25             nxt[i]=NULL; head[i]=NULL; dst[i]=-1;
26         }
27     }
28     void addEdge(int ui,int vi,ll di){
29         nd* e=new nd(ui,vi,di);
30         g[ui].push_back(e); rg[vi].push_back(e);
31     }
32     queue<int> dfsQ;
33     void dijkstra(){
34         while(dfsQ.size()) dfsQ.pop();
35         priority_queue<node> Q; Q.push(node(0,t,NULL));
36         while (!Q.empty()){
37             node p=Q.top(); Q.pop(); if(dst[p.v]!=-1)continue;
38             dst[p.v]=p.d; nxt[p.v]=p.E; dfsQ.push(p.v);
39             for(auto e:rg[p.v]) Q.push(node(p.d+e->d,e->u,e));
40         }
41     }
42     heap* merge(heap* curNd,heap* newNd){
43         if(curNd==nullNd) return newNd;
44         heap* root=new heap;memcpy(root,curNd,sizeof(heap));
45         if(newNd->edge->d<curNd->edge->d){
46             root->edge=newNd->edge;
47             root->chd[2]=newNd->chd[2];
48             root->chd[3]=newNd->chd[3];
49             newNd->edge=curNd->edge;
50             newNd->chd[2]=curNd->chd[2];
51             newNd->chd[3]=curNd->chd[3];
52         }
53         if(root->chd[0]->dep<root->chd[1]->dep)
54             root->chd[0]=merge(root->chd[0],newNd);
55         else root->chd[1]=merge(root->chd[1],newNd);
56         root->dep=max(root->chd[0]->dep,
57                     root->chd[1]->dep)+1;
58         return root;
59     }
60     vector<heap*> V;
61     void build(){

```



```

62 nullNd=new heap; nullNd->dep=0; nullNd->edge=new nd
63 ;
64 fill(nullNd->chd,nullNd->chd+4,nullNd);
65 while(not dfsQ.empty()){
66     int u=dfsQ.front(); dfsQ.pop();
67     if(!nxt[u]) head[u]=nullNd;
68     else head[u]=head[nxt[u]->v];
69     V.clear();
70     for(auto&& e:g[u]){
71         int v=e->v;
72         if(dst[v]==-1) continue;
73         e->d+=dst[v]-dst[u];
74         if(nxt[u]!=e){
75             heap* p=new heap; fill(p->chd,p->chd+4,nullNd);
76             p->dep=1; p->edge=e; V.push_back(p);
77         }
78         if(V.empty()) continue;
79         make_heap(V.begin(),V.end(),cmp);
80 #define L(X) ((X<<1)+1)
81 #define R(X) ((X<<1)+2)
82         for(size_t i=0;i<V.size();i++){
83             if(L(i)<V.size()) V[i]->chd[2]=V[L(i)];
84             else V[i]->chd[2]=nullNd;
85             if(R(i)<V.size()) V[i]->chd[3]=V[R(i)];
86             else V[i]->chd[3]=nullNd;
87         }
88         head[u]=merge(head[u],V.front());
89     }
90 }
91 vector<ll> ans;
92 void first_K(){
93     ans.clear(); priority_queue<node> Q;
94     if(dst[s]==-1) return;
95     ans.push_back(dst[s]);
96     if(head[s]!=nullNd)
97         Q.push(node(head[s],dst[s]+head[s]->edge->d));
98     for(int _=1;_<=k and not Q.empty();_++){
99         node p=Q.top(); Q.pop(); ans.push_back(p.d);
100         if(head[p.H->edge->v]!=nullNd){
101             q.H=head[p.H->edge->v]; q.d=p.d+q.H->edge->d;
102             Q.push(q);
103         }
104         for(int i=0;i<4;i++){
105             if(p.H->chd[i]!=nullNd){
106                 q.H=p.H->chd[i];
107                 q.d=p.d-p.H->edge->d+p.H->chd[i]->edge->d;
108                 Q.push(q);
109             }
110         }
111     }
112     void solve(){ // ans[i] stores the i-th shortest path
113         dijkstra(); build();
114         first_K(); // ans.size() might less than k
115     }
116 } solver;

```

## 7 String

### 7.1 Rolling Hash

```

1 const ll C = 27;
2 inline int id(char c) {return c-'a'+1;}
3 struct RollingHash {
4     string s; int n; ll mod;
5     vector<ll> Cexp, hs;
6     RollingHash(string& _s, ll _mod):
7         s(_s), n((int)s.size()), mod(_mod)
8     {
9         Cexp.assign(n, 0);
10        hs.assign(n, 0);
11        Cexp[0] = 1;
12        for (int i = 1; i < n; i++) {
13            Cexp[i] = Cexp[i-1] * C;
14            if (Cexp[i] >= mod) Cexp[i] %= mod;
15        }
16        hs[0] = id(s[0]);
17        for (int i = 1; i < n; i++) {
18            hs[i] = hs[i-1] * C + id(s[i]);
19            if (hs[i] >= mod) hs[i] %= mod;
20        }
21    }

```

```

21 inline ll query(int l, int r) {
22     ll res = hs[r] - (l ? hs[l-1] * Cexp[r-l+1] :
23         0);
24     res = (res % mod + mod) % mod;
25     return res; }

```

### 7.2 Trie

```

1 struct node {
2     int c[26]; ll cnt;
3     node(): cnt(0) {memset(c, 0, sizeof(c));}
4     node(ll x): cnt(x) {memset(c, 0, sizeof(c));}
5 };
6 struct Trie {
7     vector<node> t;
8     void init() {
9         t.clear();
10        t.emplace_back(node());
11    }
12    void insert(string s) { int ptr = 0;
13        for (auto& i : s) {
14            if (!t[ptr].c[i-'a']) {
15                t.emplace_back(node());
16                t[ptr].c[i-'a'] = (int)t.size()-1;
17                ptr = t[ptr].c[i-'a'];
18            }
19            t[ptr].cnt++; }
20    }
21 } trie;

```

### 7.3 KMP

```

1 int n, m;
2 string s, p;
3 vector<int> f;
4 void build() {
5     f.clear(); f.resize(m, 0);
6     int ptr = 0; for (int i = 1; i < m; i++) {
7         while (ptr && p[i] != p[ptr]) ptr = f[ptr-1];
8         if (p[i] == p[ptr]) ptr++;
9         f[i] = ptr;
10    }
11    void init() {
12        cin >> s >> p;
13        n = (int)s.size();
14        m = (int)p.size();
15        build(); }
16    void solve() {
17        int ans = 0, pi = 0;
18        for (int si = 0; si < n; si++) {
19            while (pi && s[si] != p[pi]) pi = f[pi-1];
20            if (s[si] == p[pi]) pi++;
21            if (pi == m) ans++, pi = f[pi-1];
22        }
23        cout << ans << endl; }

```

### 7.4 Z Value

```

1 string is, it, s;
2 int n; vector<int> z;
3 void init() {
4     cin >> is >> it;
5     s = it+'0'+is;
6     n = (int)s.size();
7     z.resize(n, 0); }
8 void solve() {
9     int ans = 0; z[0] = n;
10    for (int i = 1, l = 0, r = 0; i < n; i++) {
11        if (i <= r) z[i] = min(z[i-l], r-i+1);
12        while (i+z[i] < n && s[z[i]] == s[i+z[i]]) z[i]++;
13        if (i+z[i]-1 > r) l = i, r = i+z[i]-1;
14        if (z[i] == (int)it.size()) ans++;
15    }
16    cout << ans << endl; }

```

### 7.5 Manacher

```

1 int n; string S, s;

```



```

2 vector<int> m;
3 void manacher() {
4     s.clear(); s.resize(2*n+1, '.');
5     for (int i = 0, j = 1; i < n; i++, j += 2) s[j] = S[i];
6     m.clear(); m.resize(2*n+1, 0);
7     // m[i] := max k such that s[i-k, i+k] is palindrome
8     int mx = 0, mxk = 0;
9     for (int i = 1; i < 2*n+1; i++) {
10         if (mx-(i-mx) >= 0) m[i] = min(m[mx-(i-mx)], mx+mxk-i
11             );
12         while (0 <= i-m[i]-1 && i+m[i]+1 < 2*n+1 &&
13             s[i-m[i]-1] == s[i+m[i]+1]) m[i]++;
14         if (i+m[i] > mx+mxk) mx = i, mxk = m[i];
15     }
16 void init() { cin >> S; n = (int)S.size(); }
17 void solve() {
18     manacher();
19     int mx = 0, ptr = 0;
20     for (int i = 0; i < 2*n+1; i++) if (mx < m[i])
21         { mx = m[i]; ptr = i; }
22     for (int i = ptr-mx; i <= ptr+mx; i++)
23         if (s[i] != '.') cout << s[i];
24 cout << endl; }

```

## 7.6 Suffix Array

```

1 #define F first
2 #define S second
3 struct SuffixArray { // don't forget s += "$";
4     int n; string s;
5     vector<int> suf, lcp, rk;
6     vector<int> cnt, pos;
7     vector<pair<pii, int> > buc[2];
8     void init(string _s) {
9         s = _s; n = (int)s.size();
10        // resize(n): suf, rk, cnt, pos, lcp, buc[0~1]
11    }
12    void radix_sort() {
13        for (int t : {0, 1}) {
14            fill(cnt.begin(), cnt.end(), 0);
15            for (auto& i : buc[t]) cnt[ (t ? i.F.F : i.
16                F.S) ]++;
17            for (int i = 0; i < n; i++)
18                pos[i] = (i ? 0 : pos[i-1] + cnt[i-1])
19                ;
20            for (auto& i : buc[t])
21                buc[t^1][pos[ (t ? i.F.F : i.F.S) ]++]
22                = i;
23        }
24    bool fill_suf() {
25        bool end = true;
26        for (int i = 0; i < n; i++) suf[i] = buc[0][i].
27            S;
28        rk[suf[0]] = 0;
29        for (int i = 1; i < n; i++) {
30            int dif = (buc[0][i].F != buc[0][i-1].F);
31            end &= dif;
32            rk[suf[i]] = rk[suf[i-1]] + dif;
33        } return end;
34    }
35    void sa() {
36        for (int i = 0; i < n; i++)
37            buc[0][i] = make_pair(make_pair(s[i], s[i]
38                ), i);
39        sort(buc[0].begin(), buc[0].end());
40        if (fill_suf()) return;
41        for (int k = 0; (1<k) < n; k++) {
42            for (int i = 0; i < n; i++)
43                buc[0][i] = make_pair(make_pair(rk[i],
44                    rk[(i + (1<k)) % n]), i);
45            radix_sort();
46            if (fill_suf()) return;
47        }
48    }
49    void LCP() { int k = 0;
50        for (int i = 0; i < n-1; i++) {
51            if (rk[i] == 0) continue;
52            int pi = rk[i];
53            int j = suf[pi-1];
54            while (i+k < n && j+k < n && s[i+k] == s[j+
55                k]) k++;
56            lcp[pi] = k;
57        }
58    }
59 }

```

```

49         k = max(k-1, 0);
50     }
51 }
52 SuffixArray suffixarray;

```

## 7.7 SA-IS

```

1 const int N=300010;
2 struct SA{
3     #define REP(i,n) for(int i=0;i<int(n);i++)
4     #define REP1(i,a,b) for(int i=(a);i<=int(b);i++)
5     bool _t[N*2]; int _s[N*2],_sa[N*2];
6     int _c[N*2],x[N],_p[N],_q[N*2],hei[N],r[N];
7     int operator [](int i){ return _sa[i]; }
8     void build(int *s,int n,int m){
9         memcpy(_s,s,sizeof(int)*n);
10        sais(_s,_sa,_p,_q,_t,_c,n,m); mkhei(n);
11    }
12    void mkhei(int n){
13        REP(i,n) r[_sa[i]]=i;
14        hei[0]=0;
15        REP(i,n) if(r[i]) {
16            int ans=i>0?max(hei[r[i-1]]-1,0):0;
17            while(_s[i+ans]==_s[_sa[r[i]-1]+ans]) ans++;
18            hei[r[i]]=ans;
19        }
20    }
21    void sais(int *s,int *sa,int *p,int *q,bool *t,int *c
22        ,int n,int z){
23        bool uniq=t[n-1]=true,neq;
24        int nn=0,nmxz=-1,*nsa=sa+n,*ns=s+n,lst=-1;
25        #define MS0(x,n) memset((x),0,n*sizeof(*(x)))
26        #define MAGIC(XD) MS0(sa,n);\
27        memcpy(x,c,sizeof(int)*z); XD;\
28        memcpy(x+1,c,sizeof(int)*(z-1));\
29        REP(i,n) if(sa[i]&&!t[sa[i]-1]) sa[x[s[sa[i]-1]]++] =sa[
30            i]-1;\
31        memcpy(x,c,sizeof(int)*z);\
32        for(int i=n-1;i>=0;i--) if(sa[i]&&t[sa[i]-1]) sa[--x[s[
33            sa[i]-1]]]=sa[i]-1;
34        MS0(c,z); REP(i,n) uniq&=++c[s[i]]<2;
35        REP(i,z-1) c[i+1]+=c[i];
36        if(uniq) { REP(i,n) sa[--c[s[i]]]=i; return; }
37        for(int i=n-2;i>=0;i--)
38            t[i]=(s[i]==s[i+1]?t[i+1]:s[i]<s[i+1]);
39        MAGIC(REP1(i,1,n-1) if(t[i]&&!t[i-1]) sa[--x[s[i]
40            ]]=p[q[i]=nn++]=i);
41        REP(i,n) if(sa[i]&&t[sa[i]]&&!t[sa[i]-1]){
42            neq=lst<0||memcmp(s+sa[i],s+lst,(p[q[sa[i]]+1]-sa
43                [i])*sizeof(int));
44            ns[q[lst=sa[i]]]=nmxz+=neq;
45        }
46        sais(ns,nsa,p+nn,q+n,t+n,c+z,nn,nmxz+1);
47        MAGIC(for(int i=nn-1;i>=0;i--) sa[--x[s[p[nsa[i]
48            ]]]]=p[nsa[i]]);
49    }
50 }sa;
51 int H[N],SA[N],RA[N];
52 void suffix_array(int* ip,int len){
53     // should padding a zero in the back
54     // ip is int array, len is array length
55     // ip[0..n-1] != 0, and ip[len]=0
56     ip[len++]=0; sa.build(ip,len,128);
57     memcpy(H,sa.hei+1,len<<2); memcpy(SA,sa._sa+1,len<<2)
58     ;
59     for(int i=0;i<len;i++) RA[i]=sa.r[i]-1;
60     // resulting height, sa array \in [0,len)
61 }

```

## 7.8 Minimum Rotation

```

1 //rotate(begin(s), begin(s)+minRotation(s), end(s))
2 int minRotation(string s) {
3     int a = 0, n = s.size(); s += s;
4     for(int b = 0; b < n; b++) for(int k = 0; k < n; k++) {
5         if(a + k == b || s[a + k] < s[b + k]) {
6             b += max(0, k - 1);
7             break; }
8         if(s[a + k] > s[b + k]) {
9             a = b;
10        }
11    }
12 }

```

```

10     break;
11 } }
12 return a; }

```

## 7.9 Aho Corasick

```

1 struct ACautomata{
2     struct Node{
3         int cnt;
4         Node *go[26], *fail, *dic;
5         Node(){
6             cnt = 0; fail = 0; dic=0;
7             memset(go,0,sizeof(go));
8         }
9     }pool[1048576],*root;
10     int nMem;
11     Node* new_Node(){
12         pool[nMem] = Node();
13         return &pool[nMem++];
14     }
15     void init() { nMem = 0; root = new_Node(); }
16     void add(const string &str) { insert(root,str,0); }
17     void insert(Node *cur, const string &str, int pos){
18         for(int i=pos;i<str.size();i++){
19             if(!cur->go[str[i]-'a'])
20                 cur->go[str[i]-'a'] = new_Node();
21             cur=cur->go[str[i]-'a'];
22         }
23         cur->cnt++;
24     }
25     void make_fail(){
26         queue<Node*> que;
27         que.push(root);
28         while (!que.empty()){
29             Node* fr=que.front(); que.pop();
30             for (int i=0; i<26; i++){
31                 if (fr->go[i]){
32                     Node *ptr = fr->fail;
33                     while (ptr && !ptr->go[i]) ptr = ptr->fail;
34                     fr->go[i]->fail=ptr=(ptr?ptr->go[i]:root);
35                     fr->go[i]->dic=(ptr->cnt?ptr:ptr->dic);
36                     que.push(fr->go[i]);
37                 } } } }
38 }AC;

```

## 8 Geometry

### 8.1 Basic Operations

```

1 typedef long long T;
2 // typedef long double T;
3 const long double eps = 1e-8;
4
5 short sgn(T x) {
6     if (abs(x) < eps) return 0;
7     return x < 0 ? -1 : 1;
8 }
9
10 struct Pt {
11     T x, y;
12     Pt(T _x=0, T _y=0):x(_x), y(_y) {}
13     Pt operator+(Pt a) { return Pt(x+a.x, y+a.y); }
14     Pt operator-(Pt a) { return Pt(x-a.x, y-a.y); }
15     Pt operator*(T a) { return Pt(x*a, y*a); }
16     Pt operator/(T a) { return Pt(x/a, y/a); }
17     T operator*(Pt a) { return x*a.x + y*a.y; }
18     T operator^(Pt a) { return x*a.y - y*a.x; }
19     bool operator<(Pt a)
20     { return x < a.x || (x == a.x && y < a.y); }
21     //return sgn(x-a.x) < 0 || (sgn(x-a.x) == 0 && sgn(y-a.
22     y) < 0); }
23     bool operator==(Pt a)
24     { return sgn(x-a.x) == 0 && sgn(y-a.y) == 0; }
25 };
26
27 Pt mv(Pt a, Pt b) { return b-a; }
28 T len2(Pt a) { return a*a; }
29 T dis2(Pt a, Pt b) { return len2(b-a); }

```

```

30 short ori(Pt a, Pt b) { return ((a^b)>0) - ((a^b)<0); }
31 bool onseg(Pt p, Pt l1, Pt l2) {
32     Pt a = mv(p, l1), b = mv(p, l2);
33     return ((a^b) == 0) && ((a*b) <= 0);
34 }

```

### 8.2 InPoly

```

1 short inPoly(Pt p) {
2     // 0=Bound 1=In -1=Out
3     REP(i, n) if (onseg(p, E[i], E[(i+1)%n])) return 0;
4     int cnt = 0;
5     REP(i, n) if (banana(p, Pt(p.x+1, p.y+2e9),
6                         E[i], E[(i+1)%n])) cnt ^= 1;
7     return (cnt ? 1 : -1);
8 }

```

### 8.3 Sort by Angle

```

1 int ud(Pt a) { // up or down half plane
2     if (a.y > 0) return 0;
3     if (a.y < 0) return 1;
4     return (a.x >= 0 ? 0 : 1);
5 }
6 sort(ALL(E), [&](const Pt& a, const Pt& b){
7     if (ud(a) != ud(b)) return ud(a) < ud(b);
8     return (a^b) > 0;
9 });

```

### 8.4 Line Intersect Check

```

1 inline bool banana(Pt p1, Pt p2, Pt q1, Pt q2) {
2     if (onseg(p1, q1, q2) || onseg(p2, q1, q2) ||
3         onseg(q1, p1, p2) || onseg(q2, p1, p2)) {
4         return true;
5     }
6     Pt p = mv(p1, p2), q = mv(q1, q2);
7     return (ori(p, mv(p1, q1)) * ori(p, mv(p1, q2)) < 0 &&
8         ori(q, mv(q1, p1)) * ori(q, mv(q1, p2)) < 0);
9 }

```

### 8.5 Line Intersection

```

1 // T: Long double
2 Pt bananaPoint(Pt p1, Pt p2, Pt q1, Pt q2) {
3     if (onseg(q1, p1, p2)) return q1;
4     if (onseg(q2, p1, p2)) return q2;
5     if (onseg(p1, q1, q2)) return p1;
6     if (onseg(p2, q1, q2)) return p2;
7     double s = abs(mv(p1, p2) ^ mv(p1, q1));
8     double t = abs(mv(p1, p2) ^ mv(p1, q2));
9     return q2 * (s/(s+t)) + q1 * (t/(s+t));
10 }

```

### 8.6 Convex Hull

```

1 vector<Pt> hull;
2 void convexHull() {
3     hull.clear(); sort(ALL(E));
4     REP(t, 2) {
5         int b = SZ(hull);
6         Each(ei, E) {
7             while (SZ(hull) - b >= 2 &&
8                 ori(mv(hull[SZ(hull)-2], hull.back()),
9                     mv(hull[SZ(hull)-2], ei)) == -1) {
10                 hull.pop_back();
11             }
12             hull.pb(ei);
13         }
14         hull.pop_back();
15         reverse(ALL(E));
16     } }

```

### 8.7 Polygon Area

```

1 T dbarea(vector<Pt>& e) {
2 ll res = 0;
3 REP(i, SZ(e)) res += e[i]^e[(i+1)%SZ(e)];
4 return abs(res);
5 }

```

## 8.8 Pick's Theorem

Consider a polygon which vertices are all lattice points.

Let  $i$  = number of points inside the polygon.

Let  $b$  = number of points on the boundary of the polygon.

Then we have the following formula:

$$Area = i + \frac{b}{2} - 1$$

## 8.9 Minimum Enclosing Circle

```

1 Pt circumcenter(Pt A, Pt B, Pt C) {
2 // a1(x-A.x) + b1(y-A.y) = c1
3 // a2(x-A.x) + b2(y-A.y) = c2
4 // solve using Cramer's rule
5 T a1 = B.x-A.x, b1 = B.y-A.y, c1 = dis2(A, B)/2.0;
6 T a2 = C.x-A.x, b2 = C.y-A.y, c2 = dis2(A, C)/2.0;
7 T D = Pt(a1, b1) ^ Pt(a2, b2);
8 T Dx = Pt(c1, b1) ^ Pt(c2, b2);
9 T Dy = Pt(a1, c1) ^ Pt(a2, c2);
10 if (D == 0) return Pt(-INF, -INF);
11 return A + Pt(Dx/D, Dy/D);
12 }
13 Pt center; T r2;
14 void minEncloseCircle() {
15 mt19937 gen(chrono::steady_clock::now().
16   time_since_epoch().count());
17 shuffle(ALL(E), gen);
18 center = E[0], r2 = 0;
19 for (int i = 0; i < n; i++) {
20   if (dis2(center, E[i]) <= r2) continue;
21   center = E[i], r2 = 0;
22   for (int j = 0; j < i; j++) {
23     if (dis2(center, E[j]) <= r2) continue;
24     center = (E[i] + E[j]) / 2.0;
25     r2 = dis2(center, E[j]);
26     for (int k = 0; k < j; k++) {
27       if (dis2(center, E[k]) <= r2) continue;
28       center = circumcenter(E[i], E[j], E[k]);
29       r2 = dis2(center, E[i]);
30     }
31   }
32 } }

```

## 8.10 Closest Pair of Points

```

1 int N;
2 T ans = 9e18; // don't use LINF!!!
3 vector<Pt> p, tmp;
4
5 void init() {
6   cin >> N;
7   p.clear(); p.resize(N);
8   Each(i, p) cin >> i.x >> i.y;
9   sort(p.begin(), p.end());
10 }
11
12 void divide(int l, int r) {
13
14   int n = r-l+1;
15   if (n <= 20) {
16     for (int i = l; i <= r; i++)
17       for (int j = l; j < i; j++)
18         ans = min(ans, dis(p[i], p[j]));
19     return;
20   }
21
22   int mid = (l+r) >> 1;
23   int ml = mid, mr = mid;
24   T midx = p[mid].x;

```

```

25 while (l <= ml && p[ml].x == midx) ml--;
26 while (mr <= r && p[mr].x == midx) mr++;
27 divide(l, ml);
28 divide(mr, r);
29
30 tmp.clear();
31 for (int i = mid; i >= l; i--) {
32   if ((p[i].x-midx) * (p[i].x-midx) <= ans)
33     tmp.emplace_back(p[i]);
34   else break;
35 }
36 for (int i = mid+1; i <= r; i++) {
37   if ((p[i].x-midx) * (p[i].x-midx) <= ans)
38     tmp.emplace_back(p[i]);
39   else break;
40 }
41 sort(tmp.begin(), tmp.end(),
42   [&](const Pt& a, const Pt& b) {
43     return a.y < b.y;
44   });
45
46 int nt = (int)tmp.size();
47 REP(i, nt) for (int j = i+1, cnt = 0; j < nt && cnt <
48   3; j++, cnt++)
49   ans = min(ans, dis(tmp[i], tmp[j]));
50 }

```

## 8.11 PolyUnion

```

1 struct PY{
2   int n; Pt pt[5]; double area;
3   Pt& operator[](const int x){ return pt[x]; }
4   void init(){ //n,pt[0~n-1] must be filled
5     area=pt[n-1]^pt[0];
6     for(int i=0;i<n-1;i++) area+=pt[i]^pt[i+1];
7     if((area/=2)<0)reverse(pt,pt+n),area=-area;
8   }
9 };
10 PY py[500]; pair<double,int> c[5000];
11 inline double segP(Pt &p,Pt &p1,Pt &p2){
12   if(dcmp(p1.x-p2.x)==0) return (p.y-p1.y)/(p2.y-p1.y);
13   return (p.x-p1.x)/(p2.x-p1.x);
14 }
15 double polyUnion(int n){ //py[0~n-1] must be filled
16   int i,j,ii,jj,ta,tb,r,d; double z,w,s,sum=0,tc,td;
17   for(i=0;i<n;i++) py[i][py[i].n]=py[i][0];
18   for(i=0;i<n;i++){
19     for(ii=0;ii<py[i].n;ii++){
20       r=0;
21       c[r++]=make_pair(0.0,0); c[r++]=make_pair(1.0,0);
22       for(j=0;j<n;j++){
23         if(i==j) continue;
24         for(jj=0;jj<py[j].n;jj++){
25           ta=dcmp(tri(py[i][ii],py[i][ii+1],py[j][jj]))
26             ;
27           tb=dcmp(tri(py[i][ii],py[i][ii+1],py[j][jj
28             +1]));
29           if(ta==0 && tb==0){
30             if((py[j][jj+1]-py[j][jj])*(py[i][ii+1]-py[i][ii])>0&&j<i){
31               c[r++]=make_pair(segP(py[j][jj],py[i][ii],py[i][ii+1]),1);
32               c[r++]=make_pair(segP(py[j][jj+1],py[i][ii+1],py[i][ii+1]),-1);
33             }
34           }else if(ta>0 && tb<0){
35             tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
36             td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
37             c[r++]=make_pair(tc/(tc+td),1);
38           }else if(ta<0 && tb>0){
39             tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
40             td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
41             c[r++]=make_pair(tc/(tc+td),-1);
42           } } }
43   sort(c,c+r);
44   z=min(max(c[0].first,0.0),1.0); d=c[0].second; s=0;
45   for(j=1;j<r;j++){
46     w=min(max(c[j].first,0.0),1.0);
47     if(!d) s+=w-z;

```

```

46         d+=c[j].second; z=w;
47     }
48     sum+=(py[i][ii]^py[i][ii+1])*s;
49 }
50 }
51 return sum/2;
52 }

```

## 8.12 Minkowski Sum

```

1  /* convex hull Minkowski Sum */
2  #define INF 1000000000000000LL
3  int pos( const Pt& tp ){
4      if( tp.Y == 0 ) return tp.X > 0 ? 0 : 1;
5      return tp.Y > 0 ? 0 : 1;
6  }
7  #define N 300030
8  Pt pt[ N ], qt[ N ], rt[ N ];
9  LL Lx,Rx;
10 int dn,un;
11 inline bool cmp( Pt a, Pt b ){
12     int pa=pos( a ),pb=pos( b );
13     if(pa==pb) return (a^b)>0;
14     return pa<pb;
15 }
16 int minkowskiSum(int n,int m){
17     int i,j,r,p,q,fi,fj;
18     for(i=1,p=0;i<n;i++){
19         if( pt[i].Y<pt[p].Y ||
20            (pt[i].Y==pt[p].Y && pt[i].X<pt[p].X) ) p=i; }
21     for(i=1,q=0;i<m;i++){
22         if( qt[i].Y<qt[q].Y ||
23            (qt[i].Y==qt[q].Y && qt[i].X<qt[q].X) ) q=i; }
24     rt[0]=pt[p]+qt[q];
25     r=1; i=p; j=q; fi=fj=0;
26     while(1){
27         if((fj&&j==q) ||
28            (!fi||i==p) &&
29            cmp(pt[(p+1)%n]-pt[p],qt[(q+1)%m]-qt[q]) ) ){
30             rt[r]=rt[r-1]+pt[(p+1)%n]-pt[p];
31             p=(p+1)%n;
32             fi=1;
33         }else{
34             rt[r]=rt[r-1]+qt[(q+1)%m]-qt[q];
35             q=(q+1)%m;
36             fj=1;
37         }
38         if(r<=1 || ((rt[r]-rt[r-1])^(rt[r-1]-rt[r-2]))!=0)
39             r++;
40         else rt[r-1]=rt[r];
41         if(i==p && j==q) break;
42     }
43     return r-1;
44 }
45 void initInConvex(int n){
46     int i,p,q;
47     LL Ly,Ry;
48     Lx=INF; Rx=-INF;
49     for(i=0;i<n;i++){
50         if(pt[i].X<Lx) Lx=pt[i].X;
51         if(pt[i].X>Rx) Rx=pt[i].X;
52     }
53     Ly=Ry=INF;
54     for(i=0;i<n;i++){
55         if(pt[i].X==Lx && pt[i].Y<Ly){ Ly=pt[i].Y; p=i; }
56         if(pt[i].X==Rx && pt[i].Y<Ry){ Ry=pt[i].Y; q=i; }
57     }
58     for(dn=0,i=p;i!=q;i=(i+1)%n){ qt[dn++]=pt[i]; }
59     qt[dn]=pt[q]; Ly=Ry=-INF;
60     for(i=0;i<n;i++){
61         if(pt[i].X==Lx && pt[i].Y>Ly){ Ly=pt[i].Y; p=i; }
62         if(pt[i].X==Rx && pt[i].Y>Ry){ Ry=pt[i].Y; q=i; }
63     }
64     for(un=0,i=p;i!=q;i=(i+n-1)%n){ rt[un++]=pt[i]; }
65     rt[un]=pt[q];
66 }
67 inline int inConvex(Pt p){
68     int L,R,M;
69     if(p.X<Lx || p.X>Rx) return 0;
70     L=0;R=dn;
71     while(L<R-1){ M=(L+R)/2;

```

```

71     if(p.X<qt[M].X) R=M; else L=M; }
72     if(tri(qt[L],qt[R],p)<0) return 0;
73     L=0;R=un;
74     while(L<R-1){ M=(L+R)/2;
75         if(p.X<rt[M].X) R=M; else L=M; }
76     if(tri(rt[L],rt[R],p)>0) return 0;
77     return 1;
78 }
79 int main(){
80     int n,m,i;
81     Pt p;
82     scanf("%d",&n);
83     for(i=0;i<n;i++) scanf("%Lld%Lld",&pt[i].X,&pt[i].Y);
84     scanf("%d",&m);
85     for(i=0;i<m;i++) scanf("%Lld%Lld",&qt[i].X,&qt[i].Y);
86     n=minkowskiSum(n,m);
87     for(i=0;i<n;i++) pt[i]=rt[i];
88     scanf("%d",&m);
89     for(i=0;i<m;i++) scanf("%Lld%Lld",&qt[i].X,&qt[i].Y);
90     n=minkowskiSum(n,m);
91     for(i=0;i<n;i++) pt[i]=rt[i];
92     initInConvex(n);
93     scanf("%d",&m);
94     for(i=0;i<m;i++){
95         scanf("%Lld %Lld",&p.X,&p.Y);
96         p.X*=3; p.Y*=3;
97         puts(inConvex(p)? "YES": "NO");
98     }
99 }

```

## 9 Number Theory

### 9.1 Pollard's rho

```

1  from itertools import count
2  from math import gcd
3  from sys import stdin
4
5  for s in stdin:
6      number, x = int(s), 2
7      break2 = False
8      for cycle in count(1):
9          y = x
10         if break2:
11             break
12         for i in range(1 << cycle):
13             x = (x * x + 1) % number
14             factor = gcd(x - y, number)
15             if factor > 1:
16                 print(factor)
17                 break2 = True
18                 break

```

### 9.2 Miller Rabin

```

1  // n < 4,759,123,141      3 : 2, 7, 61
2  // n < 1,122,004,669,633  4 : 2, 13, 23, 1662803
3  // n < 3,474,749,660,383  6 : pirmes <= 13
4  // n < 2^64              7 :
5  // 2, 325, 9375, 28178, 450775, 9780504, 1795265022
6  bool witness(ll a,ll n,ll u,int t){
7      if(!(a%n)) return 0;
8      ll x=myspow(a,u,n);
9      for(int i=0;i<t;i++) {
10         ll nx=mul(x,x,n);
11         if(nx==1&&x!=1&&x!=n-1) return 1;
12         x=nx;
13     }
14     return x!=1;
15 }
16 bool miller_rabin(ll n,int s=100) {
17     // iterate s times of witness on n
18     // return 1 if prime, 0 otherwise
19     if(n<2) return 0;
20     if(!(n&1)) return n == 2;
21     ll u=n-1; int t=0;
22     while(!(u&1)) u>>=1, t++;
23     while(s--){
24         ll a=randll()%(n-1)+1;

```

```

25     if(witness(a,n,u,t)) return 0;
26 }
27 return 1;
28 }

```

### 9.3 Fast Power

Note:  $a^n \equiv a^{(n \bmod (p-1))} \pmod{p}$

### 9.4 Extend GCD

```

1 ll gcd;
2 pll extgcd(ll a, ll b) {
3     if (b == 0) {
4         gcd = a;
5         return pll{1, 0};
6     }
7     pll ans = extgcd(b, a % b);
8     return pll{ans.S, ans.F - a/b * ans.S};
9 }
10 pll bezout(ll a, ll b, ll c) {
11     bool negx = (a < 0), negy = (b < 0);
12     pll ans = extgcd(abs(a), abs(b));
13     if (c % gcd != 0) return pll{-LLINF, -LLINF};
14     return pll{ans.F * c/gcd * (negx ? -1 : 1),
15               ans.S * c/gcd * (negy ? -1 : 1)};
16 }
17 ll inv(ll a, ll p) {
18     if (p == 1) return -1;
19     pll ans = bezout(a % p, -p, 1);
20     if (ans == pll{-LLINF, -LLINF}) return -1;
21     return (ans.F % p + p) % p;
22 }

```

### 9.5 Mu + Phi

```

1 const int maxn = 1e6 + 5;
2 ll f[maxn];
3 vector<int> lpf, prime;
4 void build() {
5     lpf.clear(); lpf.resize(maxn, 1);
6     prime.clear();
7     f[1] = ...; /* mu[1] = 1, phi[1] = 1 */
8     for (int i = 2; i < maxn; i++) {
9         if (lpf[i] == 1) {
10             lpf[i] = i; prime.emplace_back(i);
11             f[i] = ...; /* mu[i] = 1, phi[i] = i-1 */
12         }
13         for (auto& j : prime) {
14             if (i*j >= maxn) break;
15             lpf[i*j] = j;
16             if (i % j == 0) f[i*j] = ...; /* 0, phi[i]*j */
17             else f[i*j] = ...; /* -mu[i], phi[i]*phi[j] */
18             if (j >= lpf[i]) break;
19         }
20     }
21 }

```

### 9.6 Other Formulas

- Inversion:  
 $aa^{-1} \equiv 1 \pmod{m}$ .  $a^{-1}$  exists iff  $\gcd(a, m) = 1$ .
- Linear inversion:  
 $a^{-1} \equiv (m - \lfloor \frac{m}{a} \rfloor) \times (m \bmod a)^{-1} \pmod{m}$
- Fermat's little theorem:  
 $a^p \equiv a \pmod{p}$  if  $p$  is prime.
- Euler function:  
 $\phi(n) = n \prod_{p|n} \frac{p-1}{p}$
- Euler theorem:  
 $a^{\phi(n)} \equiv 1 \pmod{n}$  if  $\gcd(a, n) = 1$ .
- Extended Euclidean algorithm:  
 $ax + by = \gcd(a, b) = \gcd(b, a \bmod b) = \gcd(b, a - \lfloor \frac{a}{b} \rfloor b) = bx_1 + (a - \lfloor \frac{a}{b} \rfloor b)y_1 = ay_1 + b(x_1 - \lfloor \frac{a}{b} \rfloor y_1)$

- Divisor function:

$$\sigma_x(n) = \sum_{d|n} d^x. \quad n = \prod_{i=1}^r p_i^{a_i}.$$

$$\sigma_x(n) = \prod_{i=1}^r \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1} \text{ if } x \neq 0. \quad \sigma_0(n) = \prod_{i=1}^r (a_i + 1).$$

- Chinese remainder theorem (Coprime Moduli):

$$x \equiv a_i \pmod{m_i}.$$

$$M = \prod m_i. \quad M_i = M/m_i. \quad t_i = M_i^{-1}.$$

$$x = kM + \sum a_i t_i M_i, \quad k \in \mathbb{Z}.$$

- Chinese remainder theorem:

$$x \equiv a_1 \pmod{m_1}, x \equiv a_2 \pmod{m_2} \Rightarrow x = m_1 p + a_1 = m_2 q + a_2 \Rightarrow m_1 p - m_2 q = a_2 - a_1$$

Solve for  $(p, q)$  using ExtGCD.

$$x \equiv m_1 p + a_1 \equiv m_2 q + a_2 \pmod{\text{lcm}(m_1, m_2)}$$

- Avoiding Overflow:  $ca \bmod cb = c(a \bmod b)$

- Dirichlet Convolution:  $(f * g)(n) = \sum_{d|n} f(n)g(n/d)$

- Important Multiplicative Functions + Properties:

- $\epsilon(n) = [n = 1]$
- $1(n) = 1$
- $id(n) = n$
- $\mu(n) = 0$  if  $n$  has squared prime factor
- $\mu(n) = (-1)^k$  if  $n = p_1 p_2 \cdots p_k$
- $\epsilon = \mu * 1$
- $\phi = \mu * id$
- $[n = 1] = \sum_{d|n} \mu(d)$
- $[gcd = 1] = \sum_{d|gcd} \mu(d)$

- Möbius inversion:  $f = g * 1 \Leftrightarrow g = f * \mu$

## 10 Linear Algebra

### 10.1 Gaussian-Jordan Elimination

```

1 int n; vector<vector<ll>> v;
2 void gauss(vector<vector<ll>>& v) {
3     int r = 0;
4     for (int i = 0; i < n; i++) {
5         bool ok = false;
6         for (int j = r; j < n; j++) {
7             if (v[j][i] != 0) continue;
8             swap(v[j], v[r]);
9             ok = true; break;
10        }
11        if (!ok) continue;
12        ll div = inv(v[r][i]);
13        for (int j = 0; j < n+1; j++) {
14            v[r][j] *= div;
15            if (v[r][j] >= MOD) v[r][j] %= MOD;
16        }
17        for (int j = 0; j < n; j++) {
18            if (j == r) continue;
19            ll t = v[j][i];
20            for (int k = 0; k < n+1; k++) {
21                v[j][k] -= v[r][k] * t % MOD;
22                if (v[j][k] < 0) v[j][k] += MOD;
23            }
24            r++;
25        }
26    }
27 }

```

### 10.2 Determinant

- Use GJ Elimination, if there's any row consists of only 0, then det = 0, otherwise det = product of diagonal elements.
- Properties of det:
  - Transpose: Unchanged

- Row Operation 1 - Swap 2 rows:  $-det$
- Row Operation 2 -  $k\vec{r}_i$ :  $k \times det$
- Row Operation 3 -  $k\vec{r}_i$  add to  $\vec{r}_j$ : Unchanged

## 11 Flow / Matching

## 11.1 Dinic

```

1 struct Dinic {
2     struct Edge {
3         int t, c, r;
4         Edge() {}
5         Edge(int _t, int _c, int _r):
6             t(_t), c(_c), r(_r) {}
7     };
8     vector<vector<Edge>> G;
9     vector<int> dis, iter;
10    int s, t;
11    void init(int n) {
12        G.resize(n), dis.resize(n), iter.resize(n);
13        for(int i = 0; i < n; ++i)
14            G[i].clear();
15    }
16    void add(int a, int b, int c) {
17        G[a].eb(b, c, G[b].size());
18        G[b].eb(a, 0, G[a].size() - 1);
19    }
20    bool bfs() {
21        fill(ALL(dis), -1);
22        dis[s] = 0;
23        queue<int> que;
24        que.push(s);
25        while(!que.empty()) {
26            int u = que.front(); que.pop();
27            for(auto& e : G[u]) {
28                if(e.c > 0 && dis[e.t] == -1) {
29                    dis[e.t] = dis[u] + 1;
30                    que.push(e.t);
31                }
32            }
33        }
34        return dis[t] != -1;
35    }
36    int dfs(int u, int cur) {
37        if(u == t) return cur;
38        for(int &i = iter[u]; i < (int)G[u].size(); ++i) {
39            auto& e = G[u][i];
40            if(e.c > 0 && dis[u] + 1 == dis[e.t]) {
41                int ans = dfs(e.t, min(cur, e.c));
42                if(ans > 0) {
43                    G[e.t][e.r].c += ans;
44                    e.c -= ans;
45                    return ans;
46                }
47            }
48        }
49        return 0;
50    }
51
52    int flow(int a, int b) {
53        s = a, t = b;
54        int ans = 0;
55        while(bfs()) {
56            fill(ALL(iter), 0);
57            int tmp;
58            while((tmp = dfs(s, INF)) > 0)
59                ans += tmp;
60        }
61        return ans;
62    }
63 };

```

## 11.2 ISAP

```
1 #define SZ(c) ((int)(c).size())
2 struct Maxflow{
3     static const int MAXV=50010;
```

```

4 static const int INF = 1000000;
5 struct Edge{
6     int v,c,r;
7     Edge(int _v,int _c,int _r):v(_v),c(_c),r(_r){}
8 };
9 int s,t; vector<Edge> G[MAXV];
10 int iter[MAXV],d[MAXV],gap[MAXV],tot;
11 void init(int n,int _s,int _t){
12     tot=n,s=_s,t=_t;
13     for(int i=0;i<=tot;i++){
14         G[i].clear(); iter[i]=d[i]=gap[i]=0;
15     }
16 }
17 void addEdge(int u,int v,int c){
18     G[u].push_back(Edge(v,c,SZ(G[v])));
19     G[v].push_back(Edge(u,0,SZ(G[u])-1));
20 }
21 int DFS(int p,int flow){
22     if(p==t) return flow;
23     for(int &i=iter[p];i<SZ(G[p]);i++){
24         Edge &e=G[p][i];
25         if(e.c>0&&d[p]==d[e.v]+1){
26             int f=DFS(e.v,min(flow,e.c));
27             if(f){ e.c-=f; G[e.v][e.r].c+=f; return f; }
28         }
29     }
30     if((--gap[d[p]]==0) d[s]=tot;
31     else{ d[p]++; iter[p]=0; ++gap[d[p]]; }
32     return 0;
33 }
34 int flow(){
35     int res=0;
36     for(res=0,gap[0]=tot;d[s]<tot;res+=DFS(s,INF));
37     return res;
38 } // reset: set iter,d,gap to 0
39 } flow;

```

### 11.3 MCMP

```

1 struct MCMF {
2     struct Edge {
3         int to, cap, rev;
4         ll cost;
5         Edge() {}
6         Edge(int _to, int _cap, int _rev, ll _cost) :
7             to(_to), cap(_cap), rev(_rev), cost(_cost)
8         {}
9     };
10    static const int N = 2000;
11    vector<Edge> G[N];
12    int n, s, t;
13    void init(int _n, int _s, int _t) {
14        n = _n, s = _s, t = _t;
15        for(int i = 0; i <= n; ++i)
16            G[i].clear();
17    }
18    void add_edge(int from, int to, int cap, ll cost) {
19        G[from].eb(to, cap, (int)G[to].size(), cost);
20        G[to].eb(from, 0, (int)G[from].size() - 1, -
21            cost);
22    }
23
24    bool vis[N];
25    int iter[N];
26    ll dis[N];
27    bool SPFA() {
28        for(int i = 0; i <= n; ++i)
29            vis[i] = 0, dis[i] = LINF;
30
31        dis[s] = 0; vis[s] = 1;
32        queue<int> que; que.push(s);
33        while(!que.empty()) {
34            int u = que.front(); que.pop();
35            vis[u] = 0;
36            for(auto& e : G[u]) if(e.cap > 0 && dis[e.
37                to] > dis[u] + e.cost) {
38                dis[e.to] = dis[u] + e.cost;
39                if(!vis[e.to]) {
40                    que.push(e.to);
41                    vis[e.to] = 1;
42                }
43            }
44        }
45    }
46
47    ll minCost() {
48        SPFA();
49        ll res = 0;
50        for(int i = 0; i < n; ++i)
51            res += dis[i] * iter[i];
52        return res;
53    }
54
55    ll maxFlow() {
56        SPFA();
57        ll res = 0;
58        while(dis[t] < LINF) {
59            int u = t;
60            while(u != s) {
61                int v = G[u].front().to;
62                iter[v]++;
63                u = v;
64            }
65            res += min(dis[t], iter[t]);
66            dis[t] = LINF;
67        }
68        return res;
69    }
70
71    ll minCostMaxFlow() {
72        ll res = 0;
73        while(maxFlow())
74            res += minCost();
75        return res;
76    }
77
78    ll minCostMaxFlow(int f) {
79        ll res = 0;
80        while(f) {
81            ll flow = min(f, maxFlow());
82            res += minCost();
83            f -= flow;
84        }
85        return res;
86    }
87
88    ll minCostMaxFlow(int f, int c) {
89        ll res = 0;
90        while(f) {
91            ll flow = min(f, maxFlow());
92            res += minCost();
93            f -= flow;
94        }
95        return res;
96    }
97
98    ll minCostMaxFlow(int f, int c, int s, int t) {
99        ll res = 0;
100        while(f) {
101            ll flow = min(f, maxFlow(s, t));
102            res += minCost(s, t);
103            f -= flow;
104        }
105        return res;
106    }
107
108    ll minCostMaxFlow(int f, int c, int s, int t, int cap) {
109        ll res = 0;
110        while(f) {
111            ll flow = min(f, maxFlow(s, t, cap));
112            res += minCost(s, t, cap);
113            f -= flow;
114        }
115        return res;
116    }
117
118    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev) {
119        ll res = 0;
120        while(f) {
121            ll flow = min(f, maxFlow(s, t, cap, rev));
122            res += minCost(s, t, cap, rev);
123            f -= flow;
124        }
125        return res;
126    }
127
128    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int s2, int t2) {
129        ll res = 0;
130        while(f) {
131            ll flow = min(f, maxFlow(s, t, cap, rev, s2, t2));
132            res += minCost(s, t, cap, rev, s2, t2);
133            f -= flow;
134        }
135        return res;
136    }
137
138    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int s2, int t2, int cap2, int rev2) {
139        ll res = 0;
140        while(f) {
141            ll flow = min(f, maxFlow(s, t, cap, rev, s2, t2, cap2, rev2));
142            res += minCost(s, t, cap, rev, s2, t2, cap2, rev2);
143            f -= flow;
144        }
145        return res;
146    }
147
148    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int s2, int t2, int cap2, int rev2, int s3, int t3) {
149        ll res = 0;
150        while(f) {
151            ll flow = min(f, maxFlow(s, t, cap, rev, s2, t2, cap2, rev2, s3, t3));
152            res += minCost(s, t, cap, rev, s2, t2, cap2, rev2, s3, t3);
153            f -= flow;
154        }
155        return res;
156    }
157
158    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int s2, int t2, int cap2, int rev2, int s3, int t3, int cap3, int rev3) {
159        ll res = 0;
160        while(f) {
161            ll flow = min(f, maxFlow(s, t, cap, rev, s2, t2, cap2, rev2, s3, t3, cap3, rev3));
162            res += minCost(s, t, cap, rev, s2, t2, cap2, rev2, s3, t3, cap3, rev3);
163            f -= flow;
164        }
165        return res;
166    }
167
168    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int s2, int t2, int cap2, int rev2, int s3, int t3, int cap3, int rev3, int s4, int t4) {
169        ll res = 0;
170        while(f) {
171            ll flow = min(f, maxFlow(s, t, cap, rev, s2, t2, cap2, rev2, s3, t3, cap3, rev3, s4, t4));
172            res += minCost(s, t, cap, rev, s2, t2, cap2, rev2, s3, t3, cap3, rev3, s4, t4);
173            f -= flow;
174        }
175        return res;
176    }
177
178    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int s2, int t2, int cap2, int rev2, int s3, int t3, int cap3, int rev3, int s4, int t4, int cap4, int rev4) {
179        ll res = 0;
180        while(f) {
181            ll flow = min(f, maxFlow(s, t, cap, rev, s2, t2, cap2, rev2, s3, t3, cap3, rev3, s4, t4, cap4, rev4));
182            res += minCost(s, t, cap, rev, s2, t2, cap2, rev2, s3, t3, cap3, rev3, s4, t4, cap4, rev4);
183            f -= flow;
184        }
185        return res;
186    }
187
188    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int s2, int t2, int cap2, int rev2, int s3, int t3, int cap3, int rev3, int s4, int t4, int cap4, int rev4, int s5, int t5) {
189        ll res = 0;
190        while(f) {
191            ll flow = min(f, maxFlow(s, t, cap, rev, s2, t2, cap2, rev2, s3, t3, cap3, rev3, s4, t4, cap4, rev4, s5, t5));
192            res += minCost(s, t, cap, rev, s2, t2, cap2, rev2, s3, t3, cap3, rev3, s4, t4, cap4, rev4, s5, t5);
193            f -= flow;
194        }
195        return res;
196    }
197
198    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int s2, int t2, int cap2, int rev2, int s3, int t3, int cap3, int rev3, int s4, int t4, int cap4, int rev4, int s5, int t5, int cap5, int rev5) {
199        ll res = 0;
200        while(f) {
201            ll flow = min(f, maxFlow(s, t, cap, rev, s2, t2, cap2, rev2, s3, t3, cap3, rev3, s4, t4, cap4, rev4, s5, t5, cap5, rev5));
202            res += minCost(s, t, cap, rev, s2, t2, cap2, rev2, s3, t3, cap3, rev3, s4, t4, cap4, rev4, s5, t5, cap5, rev5);
203            f -= flow;
204        }
205        return res;
206    }
207
208    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int s2, int t2, int cap2, int rev2, int s3, int t3, int cap3, int rev3, int s4, int t4, int cap4, int rev4, int s5, int t5, int cap5, int rev5, int s6, int t6) {
209        ll res = 0;
210        while(f) {
211            ll flow = min(f, maxFlow(s, t, cap, rev, s2, t2, cap2, rev2, s3, t3, cap3, rev3, s4, t4, cap4, rev4, s5, t5, cap5, rev5, s6, t6));
212            res += minCost(s, t, cap, rev, s2, t2, cap2, rev2, s3, t3, cap3, rev3, s4, t4, cap4, rev4, s5, t5, cap5, rev5, s6, t6);
213            f -= flow;
214        }
215        return res;
216    }
217
218    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int s2, int t2, int cap2, int rev2, int s3, int t3, int cap3, int rev3, int s4, int t4, int cap4, int rev4, int s5, int t5, int cap5, int rev5, int s6, int t6, int cap6, int rev6) {
219        ll res = 0;
220        while(f) {
221            ll flow = min(f, maxFlow(s, t, cap, rev, s2, t2, cap2, rev2, s3, t3, cap3, rev3, s4, t4, cap4, rev4, s5, t5, cap5, rev5, s6, t6, cap6, rev6));
222            res += minCost(s, t, cap, rev, s2, t2, cap2, rev2, s3, t3, cap3, rev3, s4, t4, cap4, rev4, s5, t5, cap5, rev5, s6, t6, cap6, rev6);
223            f -= flow;
224        }
225        return res;
226    }
227
228    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int s2, int t2, int cap2, int rev2, int s3, int t3, int cap3, int rev3, int s4, int t4, int cap4, int rev4, int s5, int t5, int cap5, int rev5, int s6, int t6, int cap6, int rev6, int s7, int t7) {
229        ll res = 0;
230        while(f) {
231            ll flow = min(f, maxFlow(s, t, cap, rev, s2, t2, cap2, rev2, s3, t3, cap3, rev3, s4, t4, cap4, rev4, s5, t5, cap5, rev5, s6, t6, cap6, rev6, s7, t7));
232            res += minCost(s, t, cap, rev, s2, t2, cap2, rev2, s3, t3, cap3, rev3, s4, t4, cap4, rev4, s5, t5, cap5, rev5, s6, t6, cap6, rev6, s7, t7);
233            f -= flow;
234        }
235        return res;
236    }
237
238    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int s2, int t2, int cap2, int rev2, int s3, int t3, int cap3, int rev3, int s4, int t4, int cap4, int rev4, int s5, int t5, int cap5, int rev5, int s6, int t6, int cap6, int rev6, int s7, int t7, int cap7, int rev7) {
239        ll res = 0;
240        while(f) {
241
```



```

40     }
41 }
42 return dis[t] != LINF;
43 }
44
45 int dfs(int u, int cur) {
46     if(u == t) return cur;
47     int ret = 0; vis[u] = 1;
48     for(int &i = iter[u]; i < (int)G[u].size(); ++i) {
49         auto &e = G[u][i];
50         if(e.cap > 0 && dis[e.to] == dis[u] + e.
51             cost && !vis[e.to]) {
52             int tmp = dfs(e.to, min(cur, e.cap));
53             e.cap -= tmp;
54             G[e.to][e.rev].cap += tmp;
55             cur -= tmp;
56             ret += tmp;
57             if(cur == 0) {
58                 vis[u] = 0;
59                 return ret;
60             }
61         }
62         vis[u] = 0;
63         return ret;
64     }
65     pair<int, ll> flow() {
66         int flow = 0; ll cost = 0;
67         while(SPFA()) {
68             memset(iter, 0, sizeof(iter));
69             int tmp = dfs(s, INF);
70             flow += tmp, cost += tmp * dis[t];
71         }
72         return {flow, cost};
73     }
74 };

```

## 11.4 Hopcroft-Karp

```

1 struct HopcroftKarp {
2     // id: X = [1, nx], Y = [nx+1, nx+ny]
3     int n, nx, ny, m, MXCNT;
4     vector<vector<int>> > g;
5     vector<int> mx, my, dis, vis;
6     void init(int nnx, int nny, int mm) {
7         nx = nnx, ny = nny, m = mm;
8         n = nx + ny + 1;
9         g.clear(); g.resize(n);
10    }
11    void add(int x, int y) {
12        g[x].emplace_back(y);
13        g[y].emplace_back(x);
14    }
15    bool dfs(int x) {
16        vis[x] = true;
17        Each(y, g[x]) {
18            int px = my[y];
19            if (px == -1 ||
20                (dis[px] == dis[x]+1 &&
21                 !vis[px] && dfs(px))) {
22                mx[x] = y;
23                my[y] = x;
24                return true;
25            }
26        }
27        return false;
28    }
29    void get() {
30        mx.clear(); mx.resize(n, -1);
31        my.clear(); my.resize(n, -1);
32
33        while (true) {
34            queue<int> q;
35            dis.clear(); dis.resize(n, -1);
36            for (int x = 1; x <= nx; x++){
37                if (mx[x] == -1) {
38                    dis[x] = 0;
39                    q.push(x);
40                }
41            }

```

```

42        while (!q.empty()) {
43            int x = q.front(); q.pop();
44            Each(y, g[x]) {
45                if (my[y] != -1 && dis[my[y]] ==
46                    -1) {
47                    dis[my[y]] = dis[x] + 1;
48                    q.push(my[y]);
49                }
50            }
51        }
52        bool brk = true;
53        vis.clear(); vis.resize(n, 0);
54        for (int x = 1; x <= nx; x++)
55            if (mx[x] == -1 && dfs(x))
56                brk = false;
57
58        if (brk) break;
59    }
60    MXCNT = 0;
61    for (int x = 1; x <= nx; x++) if (mx[x] != -1)
62        MXCNT++;
63 } hk;

```

## 11.5 Cover / Independent Set

```

1 V(E) Cover: choose some V(E) to cover all E(V)
2 V(E) Independ: set of V(E) not adj to each other
3
4 M = Max Matching
5 Cv = Min V Cover
6 Ce = Min E Cover
7 Iv = Max V Ind
8 Ie = Max E Ind (equiv to M)
9
10 M = Cv (Konig Theorem)
11 Iv = V \ Cv
12 Ce = V - M
13
14 Construct Cv:
15 1. Run Dinic
16 2. Find s-t min cut
17 3. Cv = {X in T} + {Y in S}

```

## 11.6 KM

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4 const int inf = 1e9;
5
6 struct KuhnMunkres {
7     int n;
8     vector<vector<int>> > g;
9     vector<int> lx, ly, slack;
10    vector<int> match, visx, visy;
11    KuhnMunkres(int n) : n(n), g(n, vector<int>(n)),
12        lx(n), ly(n), slack(n), match(n), visx(n), visy
13        (n) {}
14    vector<int> & operator[](int i) { return g[i]; }
15    bool dfs(int i, bool aug) { // aug = true 表示要更新 match
16        if(visx[i]) return false;
17        visx[i] = true;
18        for(int j = 0; j < n; j++) {
19            if(visy[j]) continue;
20            // 一邊擴增交錯樹、尋找增廣路徑
21            // 一邊更新 slack: 樹上的點跟樹外的點所造成的最小權重
22            int d = lx[i] + ly[j] - g[i][j];
23            if(d == 0) {
24                visy[j] = true;
25                if(match[j] == -1 || dfs(match[j], aug)) {
26                    if(aug)
27                        match[j] = i;
28                    return true;
29                }
30            } else {
31                slack[j] = d;
32            }
33        }
34        return false;
35    }
36    int solve() {
37        for(int i = 0; i < n; i++) {
38            lx[i] = *max_element(g[i].begin(), g[i].end());
39            ly[j] = 0;
40            match[j] = -1;
41            slack[j] = inf;
42        }
43        for(int i = 0; i < n; i++) {
44            while(true) {
45                visx.clear(); visx.resize(n, false);
46                visy.clear(); visy.resize(n, false);
47                if(dfs(i, false)) continue;
48                int d = inf;
49                for(int j = 0; j < n; j++) {
50                    if(visx[j]) continue;
51                    if(slack[j] < d) d = slack[j];
52                }
53                for(int i = 0; i < n; i++) {
54                    if(visx[i]) lx[i] += d;
55                    for(int j = 0; j < n; j++) {
56                        if(visy[j]) ly[j] -= d;
57                        slack[j] -= d;
58                    }
59                }
60            }
61        }
62        for(int i = 0; i < n; i++) {
63            while(true) {
64                visx.clear(); visx.resize(n, false);
65                visy.clear(); visy.resize(n, false);
66                if(dfs(i, true)) continue;
67                int d = inf;
68                for(int j = 0; j < n; j++) {
69                    if(visx[j]) continue;
70                    if(slack[j] < d) d = slack[j];
71                }
72                for(int i = 0; i < n; i++) {
73                    if(visx[i]) lx[i] += d;
74                    for(int j = 0; j < n; j++) {
75                        if(visy[j]) ly[j] -= d;
76                        slack[j] -= d;
77                    }
78                }
79            }
80        }
81        for(int i = 0; i < n; i++) {
82            while(true) {
83                visx.clear(); visx.resize(n, false);
84                visy.clear(); visy.resize(n, false);
85                if(dfs(i, true)) continue;
86                int d = inf;
87                for(int j = 0; j < n; j++) {
88                    if(visx[j]) continue;
89                    if(slack[j] < d) d = slack[j];
90                }
91                for(int i = 0; i < n; i++) {
92                    if(visx[i]) lx[i] += d;
93                    for(int j = 0; j < n; j++) {
94                        if(visy[j]) ly[j] -= d;
95                        slack[j] -= d;
96                    }
97                }
98            }
99        }
100    }
101    int get_match() {
102        for(int i = 0; i < n; i++) {
103            while(true) {
104                visx.clear(); visx.resize(n, false);
105                visy.clear(); visy.resize(n, false);
106                if(dfs(i, true)) continue;
107                int d = inf;
108                for(int j = 0; j < n; j++) {
109                    if(visx[j]) continue;
110                    if(slack[j] < d) d = slack[j];
111                }
112                for(int i = 0; i < n; i++) {
113                    if(visx[i]) lx[i] += d;
114                    for(int j = 0; j < n; j++) {
115                        if(visy[j]) ly[j] -= d;
116                        slack[j] -= d;
117                    }
118                }
119            }
120        }
121        for(int i = 0; i < n; i++) {
122            while(true) {
123                visx.clear(); visx.resize(n, false);
124                visy.clear(); visy.resize(n, false);
125                if(dfs(i, true)) continue;
126                int d = inf;
127                for(int j = 0; j < n; j++) {
128                    if(visx[j]) continue;
129                    if(slack[j] < d) d = slack[j];
130                }
131                for(int i = 0; i < n; i++) {
132                    if(visx[i]) lx[i] += d;
133                    for(int j = 0; j < n; j++) {
134                        if(visy[j]) ly[j] -= d;
135                        slack[j] -= d;
136                    }
137                }
138            }
139        }
140        for(int i = 0; i < n; i++) {
141            while(true) {
142                visx.clear(); visx.resize(n, false);
143                visy.clear(); visy.resize(n, false);
144                if(dfs(i, true)) continue;
145                int d = inf;
146                for(int j = 0; j < n; j++) {
147                    if(visx[j]) continue;
148                    if(slack[j] < d) d = slack[j];
149                }
150                for(int i = 0; i < n; i++) {
151                    if(visx[i]) lx[i] += d;
152                    for(int j = 0; j < n; j++) {
153                        if(visy[j]) ly[j] -= d;
154                        slack[j] -= d;
155                    }
156                }
157            }
158        }
159        for(int i = 0; i < n; i++) {
160            while(true) {
161                visx.clear(); visx.resize(n, false);
162                visy.clear(); visy.resize(n, false);
163                if(dfs(i, true)) continue;
164                int d = inf;
165                for(int j = 0; j < n; j++) {
166                    if(visx[j]) continue;
167                    if(slack[j] < d) d = slack[j];
168                }
169                for(int i = 0; i < n; i++) {
170                    if(visx[i]) lx[i] += d;
171                    for(int j = 0; j < n; j++) {
172                        if(visy[j]) ly[j] -= d;
173                        slack[j] -= d;
174                    }
175                }
176            }
177        }
178        for(int i = 0; i < n; i++) {
179            while(true) {
180                visx.clear(); visx.resize(n, false);
181                visy.clear(); visy.resize(n, false);
182                if(dfs(i, true)) continue;
183                int d = inf;
184                for(int j = 0; j < n; j++) {
185                    if(visx[j]) continue;
186                    if(slack[j] < d) d = slack[j];
187                }
188                for(int i = 0; i < n; i++) {
189                    if(visx[i]) lx[i] += d;
190                    for(int j = 0; j < n; j++) {
191                        if(visy[j]) ly[j] -= d;
192                        slack[j] -= d;
193                    }
194                }
195            }
196        }
197        for(int i = 0; i < n; i++) {
198            while(true) {
199                visx.clear(); visx.resize(n, false);
200                visy.clear(); visy.resize(n, false);
201                if(dfs(i, true)) continue;
202                int d = inf;
203                for(int j = 0; j < n; j++) {
204                    if(visx[j]) continue;
205                    if(slack[j] < d) d = slack[j];
206                }
207                for(int i = 0; i < n; i++) {
208                    if(visx[i]) lx[i] += d;
209                    for(int j = 0; j < n; j++) {
210                        if(visy[j]) ly[j] -= d;
211                        slack[j] -= d;
212                    }
213                }
214            }
215        }
216        for(int i = 0; i < n; i++) {
217            while(true) {
218                visx.clear(); visx.resize(n, false);
219                visy.clear(); visy.resize(n, false);
220                if(dfs(i, true)) continue;
221                int d = inf;
222                for(int j = 0; j < n; j++) {
223                    if(visx[j]) continue;
224                    if(slack[j] < d) d = slack[j];
225                }
226                for(int i = 0; i < n; i++) {
227                    if(visx[i]) lx[i] += d;
228                    for(int j = 0; j < n; j++) {
229                        if(visy[j]) ly[j] -= d;
230                        slack[j] -= d;
231                    }
232                }
233            }
234        }
235        for(int i = 0; i < n; i++) {
236            while(true) {
237                visx.clear(); visx.resize(n, false);
238                visy.clear(); visy.resize(n, false);
239                if(dfs(i, true)) continue;
240                int d = inf;
241                for(int j = 0; j < n; j++) {
242                    if(visx[j]) continue;
243                    if(slack[j] < d) d = slack[j];
244                }
245                for(int i = 0; i < n; i++) {
246                    if(visx[i]) lx[i] += d;
247                    for(int j = 0; j < n; j++) {
248                        if(visy[j]) ly[j] -= d;
249                        slack[j] -= d;
250                    }
251                }
252            }
253        }
254        for(int i = 0; i < n; i++) {
255            while(true) {
256                visx.clear(); visx.resize(n, false);
257                visy.clear(); visy.resize(n, false);
258                if(dfs(i, true)) continue;
259                int d = inf;
260                for(int j = 0; j < n; j++) {
261                    if(visx[j]) continue;
262                    if(slack[j] < d) d = slack[j];
263                }
264                for(int i = 0; i < n; i++) {
265                    if(visx[i]) lx[i] += d;
266                    for(int j = 0; j < n; j++) {
267                        if(visy[j]) ly[j] -= d;
268                        slack[j] -= d;
269                    }
270                }
271            }
272        }
273        for(int i = 0; i < n; i++) {
274            while(true) {
275                visx.clear(); visx.resize(n, false);
276                visy.clear(); visy.resize(n, false);
277                if(dfs(i, true)) continue;
278                int d = inf;
279                for(int j = 0; j < n; j++) {
280                    if(visx[j]) continue;
281                    if(slack[j] < d) d = slack[j];
282                }
283                for(int i = 0; i < n; i++) {
284                    if(visx[i]) lx[i] += d;
285                    for(int j = 0; j < n; j++) {
286                        if(visy[j]) ly[j] -= d;
287                        slack[j] -= d;
288                    }
289                }
290            }
291        }
292        for(int i = 0; i < n; i++) {
293            while(true) {
294                visx.clear(); visx.resize(n, false);
295                visy.clear(); visy.resize(n, false);
296                if(dfs(i, true)) continue;
297                int d = inf;
298                for(int j = 0; j < n; j++) {
299                    if(visx[j]) continue;
300                    if(slack[j] < d) d = slack[j];
301                }
302                for(int i = 0; i < n; i++) {
303                    if(visx[i]) lx[i] += d;
304                    for(int j = 0; j < n; j++) {
305                        if(visy[j]) ly[j] -= d;
306                        slack[j] -= d;
307                    }
308                }
309            }
310        }
311        for(int i = 0; i < n; i++) {
312            while(true) {
313                visx.clear(); visx.resize(n, false);
314                visy.clear(); visy.resize(n, false);
315                if(dfs(i, true)) continue;
316                int d = inf;
317                for(int j = 0; j < n; j++) {
318                    if(visx[j]) continue;
319                    if(slack[j] < d) d = slack[j];
320                }
321                for(int i = 0; i < n; i++) {
322                    if(visx[i]) lx[i] += d;
323                    for(int j = 0; j < n; j++) {
324                        if(visy[j]) ly[j] -= d;
325                        slack[j] -= d;
326                    }
327                }
328            }
329        }
330        for(int i = 0; i < n; i++) {
331            while(true) {
332                visx.clear(); visx.resize(n, false);
333                visy.clear(); visy.resize(n, false);
334                if(dfs(i, true)) continue;
335                int d = inf;
336                for(int j = 0; j < n; j++) {
337                    if(visx[j]) continue;
338                    if(slack[j] < d) d = slack[j];
339                }
340                for(int i = 0; i < n; i++) {
341                    if(visx[i]) lx[i] += d;
342                    for(int j = 0; j < n; j++) {
343                        if(visy[j]) ly[j] -= d;
344                        slack[j] -= d;
345                    }
346                }
347            }
348        }
349        for(int i = 0; i < n; i++) {
350            while(true) {
351                visx.clear(); visx.resize(n, false);
352                visy.clear(); visy.resize(n, false);
353                if(dfs(i, true)) continue;
354                int d = inf;
355                for(int j = 0; j < n; j++) {
356                    if(visx[j]) continue;
357                    if(slack[j] < d) d = slack[j];
358                }
359                for(int i = 0; i < n; i++) {
360                    if(visx[i]) lx[i] += d;
361                    for(int j = 0; j < n; j++) {
362                        if(visy[j]) ly[j] -= d;
363                        slack[j] -= d;
364                    }
365                }
366            }
367        }
368        for(int i = 0; i < n; i++) {
369            while(true) {
370                visx.clear(); visx.resize(n, false);
371                visy.clear(); visy.resize(n, false);
372                if(dfs(i, true)) continue;
373                int d = inf;
374                for(int j = 0; j < n; j++) {
375                    if(visx[j]) continue;
376                    if(slack[j] < d) d = slack[j];
377                }
378                for(int i = 0; i < n; i++) {
379                    if(visx[i]) lx[i] += d;
380                    for(int j = 0; j < n; j++) {
381                        if(visy[j]) ly[j] -= d;
382                        slack[j] -= d;
383                    }
384                }
385            }
386        }
387        for(int i = 0; i < n; i++) {
388            while(true) {
389                visx.clear(); visx.resize(n, false);
390                visy.clear(); visy.resize(n, false);
391                if(dfs(i, true)) continue;
392                int d = inf;
393                for(int j = 0; j < n; j++) {
394                    if(visx[j]) continue;
395                    if(slack[j] < d) d = slack[j];
396                }
397                for(int i = 0; i < n; i++) {
398                    if(visx[i]) lx[i] += d;
399                    for(int j = 0; j < n; j++) {
400                        if(visy[j]) ly[j] -= d;
401                        slack[j] -= d;
402                    }
403                }
404            }
405        }
406        for(int i = 0; i < n; i++) {
407            while(true) {
408                visx.clear(); visx.resize(n, false);
409                visy.clear(); visy.resize(n, false);
410                if(dfs(i, true)) continue;
411                int d = inf;
412                for(int j = 0; j < n; j++) {
413                    if(visx[j]) continue;
414                    if(slack[j] < d) d = slack[j];
415                }
416                for(int i = 0; i < n; i++) {
417                    if(visx[i]) lx[i] += d;
418                    for(int j = 0; j < n; j++) {
419                        if(visy[j]) ly[j] -= d;
420                        slack[j] -= d;
421                    }
422                }
423            }
424        }
425        for(int i = 0; i < n; i++) {
426            while(true) {
427                visx.clear(); visx.resize(n, false);
428                visy.clear(); visy.resize(n, false);
429                if(dfs(i, true)) continue;
430                int d = inf;
431                for(int j = 0; j < n; j++) {
432                    if(visx[j]) continue;
433                    if(slack[j] < d) d = slack[j];
434                }
435                for(int i = 0; i < n; i++) {
436                    if(visx[i]) lx[i] += d;
437                    for(int j = 0; j < n; j++) {
438                        if(visy[j]) ly[j] -= d;
439                        slack[j] -= d;
440                    }
441                }
442            }
443        }
444        for(int i = 0; i < n; i++) {
445            while(true) {
446                visx.clear(); visx.resize(n, false);
447                visy.clear(); visy.resize(n, false);
448                if(dfs(i, true)) continue;
449                int d = inf;
450                for(int j = 0; j < n; j++) {
451                    if(visx[j]) continue;
452                    if(slack[j] < d) d = slack[j];
453                }
454                for(int i = 0; i < n; i++) {
455                    if(visx[i]) lx[i] += d;
456                    for(int j = 0; j < n; j++) {
457                        if(visy[j]) ly[j] -= d;
458                        slack[j] -= d;
459                    }
460                }
461            }
462        }
463        for(int i = 0; i < n; i++) {
464            while(true) {
465                visx.clear(); visx.resize(n, false);
466                visy.clear(); visy.resize(n, false);
467                if(dfs(i, true)) continue;
468                int d = inf;
469                for(int j = 0; j < n; j++) {
470                    if(visx[j]) continue;
471                    if(slack[j] < d) d = slack[j];
472                }
473                for(int i = 0; i < n; i++) {
474                    if(visx[i]) lx[i] += d;
475                    for(int j = 0; j < n; j++) {
476                        if(visy[j]) ly[j] -= d;
477                        slack[j] -= d;
478                    }
479                }
480            }
481        }
482        for(int i = 0; i < n; i++) {
483            while(true) {
484                visx.clear(); visx.resize(n, false);
485                visy.clear(); visy.resize(n, false);
486                if(dfs(i, true)) continue;
487                int d = inf;
488                for(int j = 0; j < n; j++) {
489                    if(visx[j]) continue;
490                    if(slack[j] < d) d = slack[j];
491                }
492                for(int i = 0; i < n; i++) {
493                    if(visx[i]) lx[i] += d;
494                    for(int j = 0; j < n; j++) {
495                        if(visy[j]) ly[j] -= d;
496                        slack[j] -= d;
497                    }
498                }
499            }
500        }
501        for(int i = 0; i < n; i++) {
502            while(true) {
503                visx.clear(); visx.resize(n, false);
504                visy.clear(); visy.resize(n, false);
505                if(dfs(i, true)) continue;
506                int d = inf;
507                for(int j = 0; j < n; j++) {
508                    if(visx[j]) continue;
509                    if(slack[j] < d) d = slack[j];
510                }
511                for(int i = 0; i < n; i++) {
512                    if(visx[i]) lx[i] += d;
513                    for(int j = 0; j < n; j++) {
514                        if(visy[j]) ly[j] -= d;
515                        slack[j] -= d;
516                    }
517                }
518            }
519        }
520        for(int i = 0; i < n; i++) {
521            while(true) {
522                visx.clear(); visx.resize(n, false);
523                visy.clear(); visy.resize(n, false);
524                if(dfs(i, true)) continue;
525                int d = inf;
526                for(int j = 0; j < n; j++) {
527                    if(visx[j]) continue;
528                    if(slack[j] < d) d = slack[j];
529                }
530                for(int i = 0; i < n; i++) {
531                    if(visx[i]) lx[i] += d;
532                    for(int j = 0; j < n; j++) {
533                        if(visy[j]) ly[j] -= d;
534                        slack[j] -= d;
535                    }
536                }
537            }
538        }
539        for(int i = 0; i < n; i++) {
540            while(true) {
541                visx.clear(); visx.resize(n, false);
542                visy.clear(); visy.resize(n, false);
543                if(dfs(i, true)) continue;
544                int d = inf;
545                for(int j = 0; j < n; j++) {
546                    if(visx[j]) continue;
547                    if(slack[j] < d) d = slack[j];
548                }
549                for(int i = 0; i < n; i++) {
550                    if(visx[i]) lx[i] += d;
551                    for(int j = 0; j < n; j++) {
552                        if(visy[j]) ly[j] -= d;
553                        slack[j] -= d;
554                    }
555                }
556            }
557        }
558        for(int i = 0; i < n; i++) {
559            while(true) {
560                visx.clear(); visx.resize(n, false);
561                visy.clear(); visy.resize(n, false);
562                if(dfs(i, true)) continue;
563                int d = inf;
564                for(int j = 0; j < n; j++) {
565                    if(visx[j]) continue;
566                    if(slack[j] < d) d = slack[j];
567                }
568                for(int i = 0; i < n; i++) {
569                    if(visx[i]) lx[i] += d;
570                    for(int j = 0; j < n; j++) {
571                        if(visy[j]) ly[j] -= d;
572                        slack[j] -= d;
573                    }
574                }
575            }
576        }
577        for(int i = 0; i < n; i++) {
578            while(true) {
579                visx.clear(); visx.resize(n, false);
580                visy.clear(); visy.resize(n, false);
581                if(dfs(i, true)) continue;
582                int d = inf;
583                for(int j = 0; j < n; j++) {
584                    if(visx[j]) continue;
585                    if(slack[j] < d) d = slack[j];
586                }
587                for(int i = 0; i < n; i++) {
588                    if(visx[i]) lx[i] += d;
589                    for(int j = 0; j < n; j++) {
590                        if(visy[j]) ly[j] -= d;
591                        slack[j] -= d;
592                    }
593                }
594            }
595        }
596        for(int i = 0; i < n; i++) {
597            while(true) {
598                visx.clear(); visx.resize(n, false);
599                visy.clear(); visy.resize(n, false);
600                if(dfs(i, true)) continue;
601                int d = inf;
602                for(int j = 0; j < n; j++) {
603                    if(visx[j]) continue;
604                    if(slack[j] < d) d = slack[j];
605                }
606                for(int i = 0; i < n; i++) {
607                    if(visx[i]) lx[i] += d;
608                    for(int j = 0; j < n; j++) {
609                        if(visy[j]) ly[j] -= d;
610                        slack[j] -= d;
611                    }
612                }
613            }
614        }
615        for(int i = 0; i < n; i++) {
616            while(true) {
617                visx.clear(); visx.resize(n, false);
618                visy.clear(); visy.resize(n, false);
619                if(dfs(i, true)) continue;
620                int d = inf;
621                for(int j = 0; j < n; j++) {
622                    if(visx[j]) continue;
623                    if(slack[j] < d) d = slack[j];
624                }
625                for(int i = 0; i < n; i++) {
626                    if(visx[i]) lx[i] += d;
627                    for(int j = 0; j < n; j++) {
628                        if(visy[j]) ly[j] -= d;
629                        slack[j] -= d;
630                    }
631                }
632            }
633        }
634        for(int i = 0; i < n; i++) {
635            while(true) {
636                visx.clear(); visx.resize(n, false);
637                visy.clear(); visy.resize(n, false);
638                if(dfs(i, true)) continue;
639                int d = inf;
640                for(int j = 0; j < n; j++) {
641                    if(visx[j]) continue;
642                    if(slack[j] < d) d = slack[j];
643                }
644                for(int i = 0; i < n; i++) {
645                    if(visx[i]) lx[i] += d;
646                    for(int j = 0; j < n; j++) {
647                        if(visy[j]) ly[j] -= d;
648                        slack[j] -= d;
649                    }
650                }
651            }
652        }
653        for(int i = 0; i < n; i++) {
654            while(true) {
655                visx.clear(); visx.resize(n, false);
656                visy.clear(); visy.resize(n, false);
657                if(dfs(i, true)) continue;
658                int d = inf;
659                for(int j = 0; j < n; j++) {
660                    if(visx[j]) continue;
661                    if(slack[j] < d) d = slack[j];
662                }
663                for(int i = 0; i < n; i++) {
664                    if(visx[i]) lx[i] += d;
665                    for(int j = 0; j < n; j++) {
666                        if(visy[j]) ly[j] -= d;
667                        slack[j] -= d;
668                    }
669                }
670            }
671        }
672        for(int i = 0; i < n; i++) {
673            while(true) {
674                visx.clear(); visx.resize(n, false);
675                visy.clear(); visy.resize(n, false);
676                if(dfs(i, true)) continue;
677                int d = inf;
678                for(int j = 0; j < n; j++) {
679                    if(visx[j]) continue;
680                    if(slack[j] < d) d = slack[j];
681                }
682                for(int i = 0; i < n; i++) {
683                    if(visx[i]) lx[i] += d;
684                    for(int j = 0; j < n; j++) {
685                        if(visy[j]) ly[j] -= d;
686                        slack[j] -= d;
687                    }
688                }
689            }
690        }
691        for(int i = 0; i < n; i++) {
692            while(true) {
693                visx.clear(); visx.resize(n, false);
694                visy.clear(); visy.resize(n, false);
695                if(dfs(i, true)) continue;
696                int d = inf;
697                for(int j = 0; j < n; j++) {
698                    if(visx[j]) continue;
699                    if(slack[j] < d) d = slack[j];
700                }
701                for(int i = 0; i < n; i++) {
702                    if(visx[i]) lx[i] += d;
703                    for(int j = 0; j < n; j++) {
704                        if(visy[j]) ly[j] -= d;
705                        slack[j] -= d;
706                    }
707                }
708            }
709        }
710        for(int i = 0; i < n; i++) {
711            while(true) {
712                visx.clear(); visx.resize(n, false);
713                visy.clear(); visy.resize(n, false);
714                if(dfs(i, true)) continue;
715                int d = inf;
716                for(int j = 0; j < n; j++) {
717                    if(visx[j]) continue;
718                    if(slack[j] < d) d = slack[j];
719                }
720                for(int i = 0; i < n; i++) {
721                    if(visx[i]) lx[i] += d;
722                    for(int j = 0; j < n; j++) {
723                        if(visy[j]) ly[j] -= d;
724                        slack[j] -= d;
725                    }
726                }
727            }
728        }
729        for(int i = 0; i < n; i++) {
730            while(true) {
731                visx.clear(); visx.resize(n, false);
732                visy.clear(); visy.resize(n, false);
733                if(dfs(i, true)) continue;
734                int d = inf;
735                for(int j = 0; j < n; j++) {
736                    if(visx[j]) continue;
737                    if(slack[j] < d) d = slack[j];
738                }
739                for(int i = 0; i < n; i++) {
740                    if(visx[i]) lx[i] += d;
741                    for(int j = 0; j < n; j++) {
742                        if(visy[j]) ly[j] -= d;
743                        slack[j] -= d;
744                    }
745                }
746            }
747        }
748        for(int i = 0; i < n; i++) {
749            while(true) {
750                visx.clear(); visx.resize(n, false);
751                visy.clear(); visy.resize(n, false);
752                if(dfs(i, true)) continue;
753                int d = inf;
754                for(int j = 0; j < n; j++) {
755                    if(visx[j]) continue;
756                    if(slack[j] < d) d = slack[j];
757                }
758                for(int i = 0; i < n; i++) {
759                    if(visx[i]) lx[i] += d;
760                    for(int j = 0; j < n; j++) {
761                        if(visy[j]) ly[j] -= d;
762                        slack[j] -= d;
763                    }
764                }
765            }
766        }
767        for(int i = 0; i < n; i++) {
768            while(true) {
769                visx.clear(); visx.resize(n, false);
770                visy.clear(); visy.resize(n, false);
771                if(dfs(i, true)) continue;
772                int d = inf;
773                for(int j = 0; j < n; j++) {
774                    if(visx[j]) continue;
775                    if(slack[j] < d) d = slack[j];
776                }
777                for(int i = 0; i < n; i++) {
778                    if(visx[i]) lx[i] += d;
779                    for(int j = 0; j < n; j++) {
780                        if(visy[j]) ly[j] -= d;
781                        slack[j] -= d;
782                    }
783                }
784            }
785        }
786        for(int i = 0; i < n; i++) {
787            while(true) {
788                visx.clear(); visx.resize(n, false);
789                visy.clear(); visy.resize(n, false);
790                if(dfs(i, true)) continue;
791                int d = inf;
792                for(int j = 0; j < n; j++) {
793                    if(visx[j]) continue;
794                    if(slack[j] < d) d = slack[j];
795                }
796                for(int i = 0; i < n; i++) {
797                    if(visx[i]) lx[i] += d;
798                    for(int j = 0; j < n; j++) {
799                        if(visy[j]) ly[j] -= d;
800                        slack[j] -= d;
801                    }
802                }
803            }
804        }
805        for(int i = 0; i < n; i++) {
806            while(true) {
807                visx.clear(); visx.resize(n, false);
808                visy.clear(); visy.resize(n, false);
809                if(dfs(i, true)) continue;
810                int d = inf;
```



```

30         slack[j] = min(slack[j], d);
31     }
32 }
33 return false;
34 }
35 bool augment() { // 回傳是否有增廣路
36     for(int j = 0; j < n; j++) if(!visy[j] && slack
37         [j] == 0) {
38         visy[j] = true;
39         if(match[j] == -1 || dfs(match[j], false))
40             return true;
41     }
42     return false;
43 }
44 void relabel() {
45     int delta = inf;
46     for(int j = 0; j < n; j++) if(!visy[j]) delta =
47         min(delta, slack[j]);
48     for(int i = 0; i < n; i++) if(visx[i]) lx[i] -=
49         delta;
50     for(int j = 0; j < n; j++) {
51         if(visy[j]) ly[j] += delta;
52         else slack[j] -= delta;
53     }
54 }
55 int solve() {
56     for(int i = 0; i < n; i++) {
57         lx[i] = 0;
58         for(int j = 0; j < n; j++) lx[i] = max(lx[i]
59             , g[i][j]);
60     }
61     fill(ly.begin(), ly.end(), 0);
62     fill(match.begin(), match.end(), -1);
63     for(int i = 0; i < n; i++) {
64         // slack 在每一輪都要初始化
65         fill(slack.begin(), slack.end(), inf);
66         fill(visx.begin(), visx.end(), false);
67         fill(visy.begin(), visy.end(), false);
68         if(dfs(i, true)) continue;
69         // 重複調整頂標直到找到增廣路徑
70         while(!augment()) relabel();
71         fill(visx.begin(), visx.end(), false);
72         fill(visy.begin(), visy.end(), false);
73         dfs(i, true);
74     }
75     int ans = 0;
76     for(int j = 0; j < n; j++) if(match[j] != -1)
77         ans += g[match[j]][j];
78     return ans;
79 }
80 };
81 signed main() {
82     ios_base::sync_with_stdio(0), cin.tie(0);
83     int n;
84     while(cin >> n && n) {
85         KuhnMunkres KM(n);
86         for(int i = 0; i < n; i++) {
87             for(int j = 0; j < n; j++) {
88                 int c;
89                 cin >> c;
90                 if(c > 0)
91                     KM[i][j] = c;
92             }
93         }
94         cout << KM.solve() << '\n';
95     }
96 }

```

0	1	1	2	5
4	14	42	132	429
8	1430	4862	16796	58786
12	208012	742900	2674440	9694845

## 12.2 Burnside's Lemma

Let  $X$  be the original set.

Let  $G$  be the group of operations acting on  $X$ .

Let  $X^g$  be the set of  $x$  not affected by  $g$ .

Let  $X/G$  be the set of orbits.

Then the following equation holds:

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$$

## 13 Special Numbers

### 13.1 Fibonacci Series

1	1	1	2	3
5	5	8	13	21
9	34	55	89	144
13	233	377	610	987
17	1597	2584	4181	6765
21	10946	17711	28657	46368
25	75025	121393	196418	317811
29	514229	832040	1346269	2178309
33	3524578	5702887	9227465	14930352

$$f(45) \approx 10^9, f(88) \approx 10^{18}$$

### 13.2 Prime Numbers

- First 50 prime numbers:

1	2	3	5	7	11
6	13	17	19	23	29
11	31	37	41	43	47
16	53	59	61	67	71
21	73	79	83	89	97
26	101	103	107	109	113
31	127	131	137	139	149
36	151	157	163	167	173
41	179	181	191	193	197
46	199	211	223	227	229

- Very large prime numbers:

1000001333	1000500889	2500001909
2000000659	900004151	850001359

- $\pi(n) \equiv$  Number of primes  $\leq n \approx n/((\ln n) - 1)$   
 $\pi(100) = 25, \pi(200) = 46$   
 $\pi(500) = 95, \pi(1000) = 168$   
 $\pi(2000) = 303, \pi(4000) = 550$   
 $\pi(10^4) = 1229, \pi(10^5) = 9592$   
 $\pi(10^6) = 78498, \pi(10^7) = 664579$

## 12 Combinatorics

### 12.1 Catalan Number

$$C_0 = 1, C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}, C_n = C_n^{2n} - C_{n-1}^{2n}$$