

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	SPFA	
6.2	Bellman-Ford	
6.3	BCC - AP	
6.4	BCC - Bridge	
6.5	SCC - Tarjan	
6.6	Eulerian Path - Undir	
6.7	Eulerian Path - Dir	
6.8	Hamilton Path	
6.9	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 - Instruction	
7.7	Suffix Array	
7.8	SA-IS	
7.9	Minimum Rotation	
7.10	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	
9.6	Phi	
9.7	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

```
#include <bits/stdc++.h>

using namespace std;
using ll = long long;
using pii = pair<int, int>;
using pll = pair<ll, ll>;

#define endl '\n'

#define F first
#define S second
#define ep emplace
#define pb push_back
#define eb emplace_back
#define ALL(x) x.begin(), x.end()
#define SZ(x) (int)x.size()

namespace{
const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3f;

template<typename T> using V=vector<T>;
template<typename T1,typename T2=T1> using P = pair<T1, T2>;

void _debug() {}
template<typename A,typename... B> void _debug(A a,B... b){
    cerr<<a<<' ',_debug(b...);
}

#define debug(...) cerr<<#__VA_ARGS__<<" ": ,_debug(
__VA_ARGS__),cerr<<endl;

template<typename T>
ostream& operator<<(ostream& os,const vector<T>& v){
    for(const auto& i:v)
        os<<i<<' ';
    return os;
}

}

/*-----*/

const ll MOD = 1e9 + 7;
const int maxn = 2e5 + 5;

void init() {
    ;
}

void solve() {
    ;
}
```

```

49 }
50
51 /*
52
53 */
54
55 signed main() {
56     cin.tie(0), ios::sync_with_stdio(0);
57
58     int T = 1;
59     // cin >> T;
60     while (T--) {
61         init();
62         solve();
63     }
64
65     return 0;
66 }
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 -I ~/Desktop/cpp/include -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++ pC.cpp -o gen.out
2 g++ pB.cpp -o ac.out
3 g++ pA.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
22 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 template<typename T>
2 struct node{
3     node *l,*r;

```

```

4     T v;
5     node(T x):v(x){
6         l=r=nullptr;
7     }
8 };
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     swap(a->l,a->r);
15     return a;
16 }

```

4.3 Leftist Heap

```

1 template<typename T>
2 struct node{
3     node *l,*r;
4     int d;T v;
5     node(T x):d(1),v(x){
6         l=r=nullptr;
7     }
8 };
9 static inline int d(node* x){return x?x->d:0;}
10 node* merge(node* a,node* b){
11     if(!a||!b) return a?:b;
12     // min heap
13     if(a->v>b->v) swap(a,b);
14     a->r=merge(a->r,b);
15     if(d(a->l)<d(a->r))
16         swap(a->l,a->r);
17     a->d=d(a->r)+1;
18     return a;
19 }

```

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             dp[0][i] = mp(dp[1][i-1].F + a[i] - p, dp[1][i-1].S+1);
18         } else {
19             dp[0][i] = mp(dp[0][i-1].F, min(dp[0][i-1].S, dp[1][i-1].S+1));
20         }
21         if (dp[0][i-1].F - a[i] > dp[1][i-1].F) {
22             dp[1][i] = mp(dp[0][i-1].F - a[i], dp[0][i-1].S);
23         } else if (dp[0][i-1].F - a[i] < dp[1][i-1].F) {
24             dp[1][i] = dp[1][i-1];
25         } else {
26             dp[1][i] = mp(dp[1][i-1].F, min(dp[0][i-1].S, dp[1][i-1].S));
27         }
28     }
29     return dp[0][n-1];
30 }
31 void solve() {
32     ll l = 0, r = 1e7;
33     pll res = calc(0);
34     if (res.S <= k) return cout << res.F << endl, void();
35     while (l < r) {
36         ll mid = (l+r)>>1;
37         res = calc(mid);
38         if (res.S <= k) r = mid;
39         else l = mid+1;
40     }
41     res = calc(l);

```

```

42     cout << res.F + k*1 << endl;
43 }

```

6 Graph

6.1 SPFA

```

1 typedef pair<ll, int> edge;
2
3 int n, m;
4 vector<edge> g[maxn];
5 ll dis[maxn];
6 bitset<maxn> inq;
7
8 void init() {
9     cin >> n >> m;
10    fill(g, g+maxn, vector<edge>());
11    memset(dis, 0, sizeof(dis));
12    inq.reset();
13 }
14
15 void spfa(int sr) {
16    fill(dis, dis+maxn, LLINF);
17    inq.reset();
18
19    queue<int> q;
20    dis[sr] = 0;
21    q.push(sr);
22    inq[sr] = true;
23
24    while (!q.empty()) {
25        int u = q.front();
26        q.pop();
27        inq[u] = false;
28
29        Each(e, g[u]) {
30            int v = e.S;
31            ll w = e.F;
32
33            if (dis[v] > dis[u] + w) {
34                dis[v] = dis[u] + w;
35                if (!inq[v]) {
36                    q.push(v);
37                    inq[v] = true;
38                }
39            }
40        }
41    }
42 }

```

6.2 Bellman-Ford

```

1 typedef pair<pii, ll> edge;
2
3 int n, m, cycle;
4 bool ans;
5 vector<edge> E;
6 ll dis[maxn]; int p[maxn];
7
8 void init() {
9     cin >> n >> m;
10    cycle = -1;
11    ans = false;
12    REP(i, m) {
13        int u, v; ll w;
14        cin >> u >> v >> w;
15        E.pb(mp(mp(u, v), w));
16    }
17 }
18
19 void bellmanford() {
20    fill(dis, dis+maxn, LLINF);
21    dis[1] = 0;
22    FOR(upd, 1, n+1, 1) {
23        Each(e, E) {
24            int u = e.F.F, v = e.F.S; ll w = e.S;
25            if (dis[v] > dis[u] + w) {
26                dis[v] = dis[u] + w;
27                p[v] = u;
28                if (upd == n) cycle = v;

```

```

28     }
29 }
30 }
31 }
32 void solve() {
33     stack<int> output;
34     bellmanford();
35     if (cycle == -1) return cout << "NO\n", void();
36     cout << "YES\n";
37     for (int i = 0; i < n; i++) cycle = p[cycle]; //
38     //VIP!!
39     for (int cur = cycle; ; cur = p[cur]) {
40         output.push(cur);
41         if (cur == cycle && (int)output.size() > 1)
42             break;
43     }
44     while (!output.empty()) {
45         cout << output.top() << ' ';
46         output.pop();
47     }
48     cout << endl;
49 }

```

6.3 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 }

```

6.4 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 | }

```

6.5 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)].eb(ecnt++);
32     E.pb(no(v)^u);
33     g[no(v)].eb(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
67 int main() {
68     WiWiHorz
69     init();
70
71     REP(i, m) {
72         char su, sv;
73         int u, v;
74         cin >> su >> u >> sv >> v;
75         if (su == '-') u = no(u);
76         if (sv == '-') v = no(v);

```

```

77         clause(u, v);
78     }
79
80     FOR(i, 1, 2*n+1, 1) {
81         if (!in[i]) dfs(i);
82     }
83
84     FOR(u, 1, n+1, 1) {
85         int du = no(u);
86         if (sccid[u] == sccid[du]) {
87             return cout << "IMPOSSIBLE\n", 0;
88         }
89     }
90
91     FOR(u, 1, n+1, 1) {
92         int du = no(u);
93         cout << (sccid[u] < sccid[du] ? '+' : '-') << '
94         ';
95     }
96     cout << endl;
97     return 0;
98 }

```

6.6 Eulerian Path - Undir

```

1  int n, m;
2  vector<int> g[maxn];
3  bitset<maxn> inodd;
4
5  void init() {
6     cin >> n >> m;
7     inodd.reset();
8 }
9
10 stack<int> stk;
11 void dfs(int u) {
12     while (!g[u].empty()) {
13         int v = g[u].back();
14         g[u].pop_back();
15         dfs(v);
16     }
17     stk.push(u);
18 }
19
20
21 int main() {
22     WiWiHorz
23     init();
24
25     REP(i, m) {
26         int u, v;
27         cin >> u >> v;
28         inodd[u] = inodd[u] ^ true;
29         inodd[v] = inodd[v] ^ true;
30         g[u].emplace_back(v);
31         g[v].emplace_back(u);
32     }
33
34     FOR(i, 1, n+1, 1) {
35         if (inodd[i]) return cout << "IMPOSSIBLE\n", 0;
36     }
37
38     dfs(1);
39
40     FOR(i, 1, n+1, 1) {
41         if ((int)g[i].size()) return cout << "
42         IMPOSSIBLE\n", 0;
43     }
44
45     while (!stk.empty()) {
46         int u = stk.top();
47         stk.pop();
48         cout << u << ' ';
49     }
50     cout << endl;
51     return 0;
52 }

```

6.7 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 }
12
13 void dfs(int u) {
14     while (!g[u].empty()) {
15         int v = g[u].back();
16         g[u].pop_back();
17         dfs(v);
18     }
19     stk.push(u);
20 }
21
22 int main() {
23     WiwiHorz
24     init();
25
26     REP(i, m) {
27         int u, v;
28         cin >> u >> v;
29         g[u].emplace_back(v);
30         out[u]++, in[v]++;
31     }
32
33     FOR(i, 1, n+1, 1) {
34         if (i == 1 && out[i]-in[i] != 1) gg;
35         if (i == n && in[i]-out[i] != 1) gg;
36         if (i != 1 && i != n && in[i] != out[i]) gg;
37     }
38
39     dfs(1);
40
41     FOR(i, 1, n+1, 1) {
42         if ((int)g[i].size()) gg;
43     }
44
45     while (!stk.empty()) {
46         int u = stk.top();
47         stk.pop();
48         cout << u << ' ';
49     }
50
51     return 0;
52 }
53

```

6.8 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
24 int main() {
25     WiwiHorz
26     init();
27
28     REP(i, m) {
29         int u, v;
30         cin >> u >> v;
31         if (u == v) continue;
32         adj[--u][--v]++;
33     }
34
35     dp[0][1] = 1;
36     FOR(i, 1, n, 1) {
37         dp[i][1] = 0;
38         dp[i][1|(1<<i)] = adj[0][i];
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

```

6.9 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         }
48     }
49 }
50

```



```

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                 ;
77                 p->dep=1; p->edge=e; V.push_back(p);
78             }
79             if(V.empty()) continue;
80             make_heap(V.begin(),V.end(),cmp);
81 #define L(X) ((X<<1)+1)
82 #define R(X) ((X<<1)+2)
83             for(size_t i=0;i<V.size();i++){
84                 if(L(i)<V.size()) V[i]->chd[2]=V[L(i)];
85                 else V[i]->chd[2]=nullNd;
86                 if(R(i)<V.size()) V[i]->chd[3]=V[R(i)];
87                 else V[i]->chd[3]=nullNd;
88             }
89             head[u]=merge(head[u],V.front());
90         }
91     }
92     vector<ll> ans;
93     void first_K(){
94         ans.clear(); priority_queue<node> Q;
95         if(dst[s]==-1) return;
96         ans.push_back(dst[s]);
97         if(head[s]!=nullNd)
98             Q.push(node(head[s],dst[s]+head[s]->edge->d));
99         for(int _=1;_<k and not Q.empty();_++){
100             node p=Q.top(),q; Q.pop(); ans.push_back(p.d);
101             if(head[p.H->edge->v]!=nullNd){
102                 q.H=head[p.H->edge->v]; q.d=p.d+q.H->edge->d;
103                 Q.push(q);
104             }
105             for(int i=0;i<4;i++){
106                 if(p.H->chd[i]!=nullNd){
107                     q.H=p.H->chd[i];
108                     q.d=p.d-p.H->edge->d+p.H->chd[i]->edge->d;
109                     Q.push(q);
110                 }
111             }
112         }
113     }
114 } solver;

```

7 String

7.1 Rolling Hash

```

1 // count how many times t occurs in s
2 string s, t;
3 int ns, nt;
4 const ll C = 26;
5 const ll MOD = 1e9 + 7;

```

```

6 ll Cexp[maxn], ht[maxn], hs;
7
8 void build_Cexp() {
9     Cexp[0] = 1;
10    FOR(i, 1, nt, 1) {
11        Cexp[i] = Cexp[i-1] * C;
12        if (Cexp[i] >= MOD) Cexp[i] %= MOD;
13    }
14 }
15
16 void build_hash() {
17     REP(i, ns) {
18         hs += Cexp[ns-1-i] * (s[i] - 'a');
19         if (hs >= MOD) hs %= MOD;
20     }
21     ht[0] = (t[0] - 'a');
22     FOR(i, 1, nt, 1) {
23         ht[i] = ht[i-1] * C + (t[i] - 'a');
24         if (ht[i] >= MOD) ht[i] %= MOD;
25     }
26 }
27
28 inline ll ht_query(int l, int r) {
29     ll res = ht[r] - (l ? ht[l-1] * Cexp[len(l, r)] :
30     0);
31     res = (res%MOD + MOD) % MOD;
32     return res;
33 }

```

7.2 Trie

```

1 struct node {
2     int c[26];
3     ll cnt;
4     node() {
5         memset(c, 0, sizeof(c));
6         cnt = 0;
7     }
8     node(ll x) {
9         memset(c, 0, sizeof(c));
10        cnt = x;
11    }
12 };
13 struct Trie {
14     vector<node> t;
15     void init() {
16         t.clear();
17         t.emplace_back(node());
18     }
19     void insert(string s) {
20         int ptr = 0;
21         Each(i, s) {
22             if (!t[ptr].c[i-'a']) {
23                 t.emplace_back(node());
24                 t[ptr].c[i-'a'] = (int)t.size()-1;
25             }
26             ptr = t[ptr].c[i-'a'];
27         }
28         t[ptr].cnt++;
29     }
30 };
31 Trie 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;
7     for (int i = 1; i < m; i++) {
8         while (ptr && p[i] != p[ptr]) ptr = f[ptr-1];
9         if (p[i] == p[ptr]) ptr++;
10        f[i] = ptr;
11    }
12 }
13 void init() {
14     cin >> s >> p;
15     n = (int)s.size();

```



```

16 m = (int)p.size();
17 build();
18 }
19 void solve() {
20     int ans = 0, pi = 0;
21     for (int si = 0; si < n; si++) {
22         while (pi && s[si] != p[pi]) pi = f[pi-1];
23         if (s[si] == p[pi]) pi++;
24         if (pi == m) ans++, pi = f[pi-1];
25     }
26     cout << ans << endl;
27 }

```

7.4 Z Value

```

1 string is, it, s;
2 int n;
3 vector<int> z;
4 void init() {
5     cin >> is >> it;
6     s = it+'0'+is;
7     n = (int)s.size();
8     z.resize(n, 0);
9 }
10 void solve() {
11     int ans = 0;
12     z[0] = n;
13     for (int i = 1, l = 0, r = 0; i < n; i++) {
14         if (i <= r) z[i] = min(z[i-l], r-i+1);
15         while (i+z[i] < n && s[z[i]] == s[i+z[i]]) z[i]++;
16         if (i+z[i]-1 > r) l = i, r = i+z[i]-1;
17         if (z[i] == (int)it.size()) ans++;
18     }
19     cout << ans << endl;
20 }

```

7.5 Manacher

```

1 int n;
2 string S, s;
3 vector<int> m;
4
5 void manacher() {
6     s.clear(); s.resize(2*n+1, '.');
7     for (int i = 0, j = 1; i < n; i++, j += 2) s[j] = S[i];
8
9     m.clear(); m.resize(2*n+1, 0);
10    // m[i] := max k such that s[i-k, i+k] is palindrome
11
12    int mx = 0, mxk = 0;
13    FOR(i, 1, 2*n+1, 1) {
14        if (mx-(i-mx) >= 0) m[i] = min(m[mx-(i-mx)], mx+mxk-i);
15        while (0 <= i-m[i]-1 && i+m[i]+1 < 2*n+1 &&
16            s[i-m[i]-1] == s[i+m[i]+1]) m[i]++;
17        if (i+m[i] > mx+mxk) mx = i, mxk = m[i];
18    }
19
20    void init() {
21        cin >> S;
22        n = (int)S.size();
23    }
24
25    void solve() {
26        manacher();
27        int mx = 0, ptr = 0;
28        REP(i, 2*n+1) {
29            if (mx < m[i]) {
30                mx = m[i];
31                ptr = i;
32            }
33        }
34        for (int i = ptr-mx; i <= ptr+mx; i++)
35            if (s[i] != '.') cout << s[i];
36        cout << endl;
37    }

```

7.6 Suffix Array - Instruction

```

1 /* Steps to build suffix array
2  * 1. Base Case: One Letter
3  *   Do AnySort() -> store in buc[0]
4  *   Fill SA and Rank
5  *
6  * 2. Repeat O(log(n)) times
7  *   Fill buc[0] with last result
8  *   Do RadixSort()
9  *   Fill SA and Rank
10 *
11 * Conditions for ending in advance:
12 *   if every element is distinct (Rank[i] all
13   diff)
14 *   // just end process
15 *
16 * Tip: Radix Sort
17 *   Repeat twice
18 *   Count
19 *   Reset bucket (build pos array)
20 *   Fill element into new bucket
21 */

```

7.7 Suffix Array

```

1 // For Building Suffix Array and LCP Array
2 int n;
3 string s;
4 vector<int> suf, lcp, rk;
5
6 // For Radix Sort
7 vector<int> cnt, pos;
8 vector<pair<pii, int>> buc[2]; // 0: result, 1: temp
9
10 void init() {
11     n = (int)s.size();
12     suf.resize(n);
13     rk.resize(n);
14     cnt.resize(n);
15     pos.resize(n);
16     Each(i, buc) i.resize(n);
17 }
18
19 void radix_sort() {
20     REP(t, 2) {
21         fill(iter(cnt), 0);
22         Each(i, buc[t]) cnt[ (t ? i.F.F : i.F.S) ]++;
23         REP(i, n) {
24             pos[i] = (!i ? 0 : pos[i-1] + cnt[i-1]);
25         }
26         Each(i, buc[t]) {
27             buc[t^1][pos[ (t ? i.F.F : i.F.S) ]++] = i;
28         }
29     }
30 }
31
32 bool fill_suf() {
33     bool end = true;
34     REP(i, n) suf[i] = buc[0][i].S;
35     rk[suf[0]] = 0;
36     FOR(i, 1, n, 1) {
37         int dif = (buc[0][i].F != buc[0][i-1].F);
38         end &= dif;
39         rk[suf[i]] = rk[suf[i-1]] + dif;
40     }
41     return end;
42 }
43
44 void sa() {
45     s += (char)30;
46     init();
47
48     REP(i, n) buc[0][i] = mp(mp(s[i], s[i]), i);
49     sort(iter(buc[0]));
50     if (fill_suf()) return;
51
52     for (int k = 0; (1<<k) < n; k++) {
53         REP(i, n) {
54             buc[0][i] = mp(mp(rk[i], rk[(i + (1<<k)) %
55                 n]), i);
56         }
57     }

```

```

57     radix_sort();
58     if (fill_suf()) return;
59 }
60 }
61
62 // lcp[i] = lcp(rank_i, rank_(i-1))
63 // lcp[0] = 0
64 void LCP() {
65     int k = 0;
66     REP(i, n-1) {
67         int pi = rk[i];
68         int j = suf[pi-1];
69         while (s[i+k] == s[j+k]) k++;
70         lcp[pi] = k;
71         k = max(k-1, 0);
72     }
73 }
74
75 int main() {
76     elpsycongroo
77
78     cin >> s;
79
80     sa();
81
82     REP(i, n) cout << suf[i] << ' ';
83     cout << '\n';
84     REP(i, n) cout << lcp[i] << ' ';
85     cout << '\n';
86
87     return 0;
88 }

```

7.8 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[i]-1;\
30         memcpy(x,c,sizeof(int)*z);\
31         for(int i=n-1;i>=0;i--) if(sa[i]&&t[sa[i]-1]) sa[--x[s[sa[i]-1]]]=sa[i]-1;
32         MS0(c,z); REP(i,n) uniq&=++c[s[i]]<2;
33         REP(i,z-1) c[i+1]+=c[i];
34         if(uniq) { REP(i,n) sa[--c[s[i]]]=i; return; }
35         for(int i=n-2;i>=0;i--)
36             t[i]=(s[i]==s[i+1]?t[i+1]:s[i]<s[i+1]);
37         MAGIC(REP1(i,1,n-1) if(t[i]&&!t[i-1]) sa[--x[s[i]]]=p[q[i]=nn++]=i);
38         REP(i,n) if(sa[i]&&t[sa[i]]&&!t[sa[i]-1]){
39             neq=lst<0||memcmp(s+sa[i],s+lst,(p[q[sa[i]]+1]-sa[i])*sizeof(int));
40             ns[q[lst=sa[i]]]=nmxz+neq;
41         }
42         sais(ns,nsa,p+nn,q+n,t+n,c+z,nn,nmxz+1);

```

```

42     MAGIC(for(int i=nn-1;i>=0;i--) sa[--x[s[p[nsa[i]]]]]=p[nsa[i]]);
43 }
44 }sa;
45 int H[N],SA[N],RA[N];
46 void suffix_array(int* ip,int len){
47     // should padding a zero in the back
48     // ip is int array, len is array length
49     // ip[0..n-1] != 0, and ip[len]=0
50     ip[len++]=0; sa.build(ip,len,128);
51     memcpy(H,sa.hei+1,len<<2); memcpy(SA,sa._sa+1,len<<2);
52     ;
53     for(int i=0;i<len;i++) RA[i]=sa.r[i]-1;
54     // resulting height, sa array \in [0,len)
55 }

```

7.9 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          }
9          if(s[a + k] > s[b + k]) {
10             a = b;
11             break;
12         }
13     }
14     return a;
15 }

```

7.10 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
31 short ori(Pt a, Pt b) { return ((a^b)>0) - ((a^b)<0); }
32 bool onseg(Pt p, Pt l1, Pt l2) {
33     Pt a = mv(p, l1), b = mv(p, l2);
34     return ((a^b) == 0) && ((a*b) <= 0);
35 }

```

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, p2) || 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[i]);
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 = 1; i <= r; i++)
17             for (int j = 1; 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 < 3; j++, cnt++)
48         ans = min(ans, dis(tmp[i], tmp[j]));
49
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[
31             i][ii])>0&&j<i){
32                 c[r++]=make_pair(segP(py[j][jj],py[i][ii
33                 ],py[i][ii+1]),1);
34                 c[r++]=make_pair(segP(py[j][jj+1],py[i][
35                 ii],py[i][ii+1]),-1);
36             }
37         }else if(ta>0 && tb<0){
38             tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
39             td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
40             c[r++]=make_pair(tc/(tc-td),1);
41         }else if(ta<0 && tb>0){
42             tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
43             td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
44             c[r++]=make_pair(tc/(tc-td),-1);
45         } } }
46     sort(c,c+r);
47     z=min(max(c[0].first,0.0),1.0); d=c[0].second; s
48     =0;
49     for(j=1;j<r;j++){
50         w=min(max(c[j].first,0.0),1.0);
51         if(!d) s+=w-z;
52         d+=c[j].second; z=w;
53     }
54     sum+=(py[i][ii]^py[i][ii+1])*s;
55 }
56 return sum/2;
57 }

```

8.12 Minkowski Sum

```

1 /* convex hull Minkowski Sum*/
2 #define INF 10000000000000LL
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 }

```

```

44 void initInConvex(int n){
45     int i,p,q;
46     LL Ly,Ry;
47     Lx=INF; Rx=-INF;
48     for(i=0;i<n;i++){
49         if(pt[i].X<Lx) Lx=pt[i].X;
50         if(pt[i].X>Rx) Rx=pt[i].X;
51     }
52     Ly=Ry=INF;
53     for(i=0;i<n;i++){
54         if(pt[i].X==Lx && pt[i].Y<Ly){ Ly=pt[i].Y; p=i; }
55         if(pt[i].X==Rx && pt[i].Y>Ry){ Ry=pt[i].Y; q=i; }
56     }
57     for(dn=0,i=p;i<q;i=(i+1)%n){ qt[dn++]=pt[i]; }
58     qt[dn]=pt[q]; Ly=Ry=-INF;
59     for(i=0;i<n;i++){
60         if(pt[i].X==Lx && pt[i].Y>Ly){ Ly=pt[i].Y; p=i; }
61         if(pt[i].X==Rx && pt[i].Y>Ry){ Ry=pt[i].Y; q=i; }
62     }
63     for(un=0,i=p;i!=q;i=(i+n-1)%n){ rt[un++]=pt[i]; }
64     rt[un]=pt[q];
65 }
66 inline int inConvex(Pt p){
67     int L,R,M;
68     if(p.X<Lx || p.X>Rx) return 0;
69     L=0;R=dn;
70     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 : pimes <= 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=mypow(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
11 pll bezout(ll a, ll b, ll c) {
12     bool negx = (a < 0), negy = (b < 0);
13     pll ans = extgcd(abs(a), abs(b));
14     if (c % GCD != 0) return pll{-LLINF, -LLINF};
15     return pll{ans.F * c/GCD * (negx ? -1 : 1),
16                ans.S * c/GCD * (negy ? -1 : 1)};
17 }
18
19 ll inv(ll a, ll p) {
20     if (p == 1) return -1;
21     pll ans = bezout(a % p, -p, 1);
22     if (ans == pll{-LLINF, -LLINF}) return -1;
23     return (ans.F % p + p) % p;
24 }

```

9.5 Mu

```

1 const int maxn = 1e6 + 5;
2 ll mu[maxn];
3 vector<int> lpf, prime;
4 void buildMu() {
5     lpf.clear(); lpf.resize(maxn, 1);
6     prime.clear();
7     mu[1] = 1;
8     for (int i = 2; i < maxn; i++) {
9         if (lpf[i] == 1) {
10             lpf[i] = i; prime.emplace_back(i);
11             mu[i] = -1;
12         }
13         Each(j, prime) {
14             if (i*j >= maxn) break;
15             lpf[i*j] = j;
16             if (i % j == 0) mu[i*j] = 0;

```

```

17     else mu[i*j] = -mu[i];
18     if (j >= lpf[i]) break;
19 }
20 }
21 }

```

9.6 Phi

```

1 const int maxn = 1e6 + 5;
2 ll phi[maxn];
3 vector<int> lpf, prime;
4 void buildPhi() {
5     lpf.clear(); lpf.resize(maxn, 1);
6     prime.clear();
7     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             phi[i] = i-1;
12         }
13         Each(j, prime) {
14             if (i*j >= maxn) break;
15             lpf[i*j] = j;
16             if (i % j == 0) phi[i*j] = phi[i]*j;
17             else phi[i*j] = phi[i]*phi[j];
18             if (j >= lpf[i]) break;
19         }
20     }
21 }

```

9.7 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:
 $ax + by = \gcd(a, b) = \gcd(b, a \bmod b) = \gcd(b, a - \lfloor \frac{a}{b} \rfloor b)$
 $\lfloor \frac{a}{b} \rfloor b = bx_1 + (a - \lfloor \frac{a}{b} \rfloor b)y_1 = ay_1 + b(x_1 - \lfloor \frac{a}{b} \rfloor y_1)$

- Divisor function:
 $\sigma_x(n) = \sum_{d|n} d^x \cdot n = \prod_{i=1}^r p_i^{a_i}$
 $\sigma_x(n) = \prod_{i=1}^r \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1}$ if $x \neq 0$. $\sigma_0(n) = \prod_{i=1}^r (a_i + 1)$.

- Chinese remainder theorem (Coprime Moduli):
 $x \equiv a_i \pmod{m_i}$.
 $M = \prod m_i$. $M_i = M/m_i$. $t_i = M_i^{-1}$.
 $x = kM + \sum a_i t_i M_i$, $k \in \mathbb{Z}$.

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

- 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;
2 ll mod;
3 vector<ll> inv;
4 vector<vector<ll>> v;
5 void build() {
6     inv.clear(); inv.resize(mod, 0);
7     inv[1] = 1;
8     FOR(i, 2, mod, 1) {
9         inv[i] = (mod - mod/i) * inv[mod%i] % mod;
10    }
11 }
12 void init() {
13     cin >> n >> mod;
14     build();
15     v.resize(n, vector<ll>(n+1, 0LL));
16     REP(i, n) cin >> v[i][n];
17     REP(i, n) REP(j, n) cin >> v[j][i];
18 }
19 void gauss(vector<vector<ll>>& v) {
20     int r = 0;
21     REP(i, n) {
22         bool ok = false;
23         FOR(j, r, n, 1) {
24             if (v[j][i] != 0) continue;
25             swap(v[j], v[r]);
26             ok = true;
27             break;
28         }
29         if (!ok) continue;
30         ll div = inv[v[r][i]];
31         REP(j, n+1) {
32             v[r][j] *= div;
33             if (v[r][j] >= mod) v[r][j] %= mod;
34         }
35         REP(j, n) {
36             if (j == r) continue;
37             ll t = v[j][i];
38             REP(k, n+1) {
39                 v[j][k] -= v[r][k] * t % mod;
40                 if (v[j][k] < 0) v[j][k] += mod;
41             }
42         }
43         r++;
44     }
45 }
46 void solve() {
47     gauss(v);
48     REP(i, n) {
49         cout << v[i][n] << ' ';
50     }
51     cout << endl;
52 }

```

10.2 Determinant

- Use GJ Elimination, if there's any row consists of only 0, then $\det = 0$, otherwise $\det = \text{product of diagonal elements}$.

- 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] * G[i].s;
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].p;
62                res += G[v].c * G[v].s;
63                u = v;
64            }
65            int f = 1;
66            u = t;
67            while(u != s) {
68                int v = G[u].p;
69                f = min(f, G[v].c);
70                u = v;
71            }
72            u = t;
73            while(u != s) {
74                int v = G[u].p;
75                G[v].c -= f;
76                G[v].s += f;
77                u = v;
78            }
79        }
80        return res;
81    }
82
83    ll minCostMaxFlow() {
84        ll res = 0;
85        while(dis[t] < LINF) {
86            res += maxFlow();
87        }
88        return res;
89    }
90
91    ll minCostMaxFlow(int f) {
92        ll res = 0;
93        while(f > 0 && dis[t] < LINF) {
94            res += maxFlow(f);
95            f -= maxFlow(f);
96        }
97        return res;
98    }
99
100    ll minCostMaxFlow(int f, int c) {
101        ll res = 0;
102        while(f > 0 && dis[t] < LINF) {
103            res += maxFlow(f, c);
104            f -= maxFlow(f, c);
105        }
106        return res;
107    }
108
109    ll minCostMaxFlow(int f, int c, int s, int t) {
110        ll res = 0;
111        while(f > 0 && dis[t] < LINF) {
112            res += maxFlow(f, c, s, t);
113            f -= maxFlow(f, c, s, t);
114        }
115        return res;
116    }
117
118    ll minCostMaxFlow(int f, int c, int s, int t, int cap) {
119        ll res = 0;
120        while(f > 0 && dis[t] < LINF) {
121            res += maxFlow(f, c, s, t, cap);
122            f -= maxFlow(f, c, s, t, cap);
123        }
124        return res;
125    }
126
127    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev) {
128        ll res = 0;
129        while(f > 0 && dis[t] < LINF) {
130            res += maxFlow(f, c, s, t, cap, rev);
131            f -= maxFlow(f, c, s, t, cap, rev);
132        }
133        return res;
134    }
135
136    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int cost) {
137        ll res = 0;
138        while(f > 0 && dis[t] < LINF) {
139            res += maxFlow(f, c, s, t, cap, rev, cost);
140            f -= maxFlow(f, c, s, t, cap, rev, cost);
141        }
142        return res;
143    }
144
145    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int cost, int cap2) {
146        ll res = 0;
147        while(f > 0 && dis[t] < LINF) {
148            res += maxFlow(f, c, s, t, cap, rev, cost, cap2);
149            f -= maxFlow(f, c, s, t, cap, rev, cost, cap2);
150        }
151        return res;
152    }
153
154    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int cost, int cap2, int rev2) {
155        ll res = 0;
156        while(f > 0 && dis[t] < LINF) {
157            res += maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2);
158            f -= maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2);
159        }
160        return res;
161    }
162
163    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int cost, int cap2, int rev2, int cost2) {
164        ll res = 0;
165        while(f > 0 && dis[t] < LINF) {
166            res += maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2, cost2);
167            f -= maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2, cost2);
168        }
169        return res;
170    }
171
172    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int cost, int cap2, int rev2, int cost2, int cap3) {
173        ll res = 0;
174        while(f > 0 && dis[t] < LINF) {
175            res += maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2, cost2, cap3);
176            f -= maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2, cost2, cap3);
177        }
178        return res;
179    }
180
181    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int cost, int cap2, int rev2, int cost2, int cap3, int rev3) {
182        ll res = 0;
183        while(f > 0 && dis[t] < LINF) {
184            res += maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2, cost2, cap3, rev3);
185            f -= maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2, cost2, cap3, rev3);
186        }
187        return res;
188    }
189
190    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int cost, int cap2, int rev2, int cost2, int cap3, int rev3, int cost3) {
191        ll res = 0;
192        while(f > 0 && dis[t] < LINF) {
193            res += maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2, cost2, cap3, rev3, cost3);
194            f -= maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2, cost2, cap3, rev3, cost3);
195        }
196        return res;
197    }
198
199    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int cost, int cap2, int rev2, int cost2, int cap3, int rev3, int cost3, int cap4) {
200        ll res = 0;
201        while(f > 0 && dis[t] < LINF) {
202            res += maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2, cost2, cap3, rev3, cost3, cap4);
203            f -= maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2, cost2, cap3, rev3, cost3, cap4);
204        }
205        return res;
206    }
207
208    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int cost, int cap2, int rev2, int cost2, int cap3, int rev3, int cost3, int cap4, int rev4) {
209        ll res = 0;
210        while(f > 0 && dis[t] < LINF) {
211            res += maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2, cost2, cap3, rev3, cost3, cap4, rev4);
212            f -= maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2, cost2, cap3, rev3, cost3, cap4, rev4);
213        }
214        return res;
215    }
216
217    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int cost, int cap2, int rev2, int cost2, int cap3, int rev3, int cost3, int cap4, int rev4, int cost4) {
218        ll res = 0;
219        while(f > 0 && dis[t] < LINF) {
220            res += maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2, cost2, cap3, rev3, cost3, cap4, rev4, cost4);
221            f -= maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2, cost2, cap3, rev3, cost3, cap4, rev4, cost4);
222        }
223        return res;
224    }
225
226    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int cost, int cap2, int rev2, int cost2, int cap3, int rev3, int cost3, int cap4, int rev4, int cost4, int cap5) {
227        ll res = 0;
228        while(f > 0 && dis[t] < LINF) {
229            res += maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2, cost2, cap3, rev3, cost3, cap4, rev4, cost4, cap5);
230            f -= maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2, cost2, cap3, rev3, cost3, cap4, rev4, cost4, cap5);
231        }
232        return res;
233    }
234
235    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int cost, int cap2, int rev2, int cost2, int cap3, int rev3, int cost3, int cap4, int rev4, int cost4, int cap5, int rev5) {
236        ll res = 0;
237        while(f > 0 && dis[t] < LINF) {
238            res += maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2, cost2, cap3, rev3, cost3, cap4, rev4, cost4, cap5, rev5);
239            f -= maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2, cost2, cap3, rev3, cost3, cap4, rev4, cost4, cap5, rev5);
240        }
241        return res;
242    }
243
244    ll minCostMaxFlow(int f, int c, int s, int t, int cap, int rev, int cost, int cap2, int rev2, int cost2, int cap3, int rev3, int cost3, int cap4, int rev4, int cost4, int cap5, int rev5, int cost5) {
245        ll res = 0;
246        while(f > 0 && dis[t] < LINF) {
247            res += maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2, cost2, cap3, rev3, cost3, cap4, rev4, cost4, cap5, rev5, cost5);
248            f -= maxFlow(f, c, s, t, cap, rev, cost, cap2, rev2, cost2, cap3, rev3, cost3, cap4, rev4, cost4, cap5, rev5, cost5);
249        }
250        return res;

```



```

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        int res = 0;
38        for(int i = 0; i < n; i++) {
39            if(match[i] == -1) continue;
40            res += g[i][match[i]];
41        }
42        return res;
43    }
44 };

```

```

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

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