

# Contents

<b>1 Basic</b>	<b>1</b>
1.1 vimrc	1
1.2 default code	1
1.3 state machine	1
<b>2 Flow</b>	<b>2</b>
2.1 Dinic	2
2.2 GomoryHu tree	2
2.3 min cost flow	3
2.4 SW mincut 全點對最小割	3
<b>3 Matching</b>	<b>3</b>
3.1 Hungarian	3
3.2 KM	4
3.3 Matching.txt	4
3.4 Maximum General Matching	4
3.5 Minimum General Weighted Matching	5
<b>4 Graph</b>	<b>5</b>
4.1 BCC edge	5
4.2 Dijkstra	6
4.3 Domination.txt	6
4.4 max clique	6
4.5 min mean cycle	6
4.6 SSSP related concepts	7
4.7 Tarjan.cpp	7
4.8 2-SAT	7
4.9 平面圖判定	8
<b>5 Math</b>	<b>10</b>
5.1 ax+by=gcd(a,b)	10
5.2 FFT	10
5.3 GaussElimination	11
5.4 inverse	11
5.5 Miller-Rabin	11
5.6 Mobius	11
5.7 pollardRho	11
5.8 SG	11
5.9 theorem	12
<b>6 Geometry</b>	<b>12</b>
6.1 2D point template	12
6.2 circumcentre	13
6.3 ConvexHull	13
6.4 half plane intersection	13
6.5 Intersection of two circle	13
6.6 Intersection of two lines	13
6.7 Smallest Circle	13
<b>7 String</b>	<b>14</b>
7.1 AC automaton	14
7.2 KMP	14
7.3 palindromic tree	14
7.4 SAM	15
7.5 smallest rotation	15
7.6 suffix array	15
7.7 Z value	15
7.8 BWT (Burrows-Wheeler Transform)	16
<b>8 Data structure</b>	<b>16</b>
8.1 2D range tree	16
8.2 ext heap	16
8.3 KD tree	16
8.4 Link-Cut tree	17
8.5 Treap Lin	18
<b>9 Other</b>	<b>19</b>
9.1 count spanning tree	19
9.2 C++11 random	20
9.3 Digit Counting	20
9.4 DP optimization	20
9.5 DP 1D/1D	20
9.6 stable marriage	20
9.7 Mo's algorithm	21
9.8 Parser	21
9.9 java cheat sheet	22
9.10 python cheat sheet	22

# 1 Basic

## 1.1 vimrc

```

=== .vimrc ===

set et nu cin ls=2 ts=4 sw=4 sts=4 ttm=100
syntax on

nn <F4> :w ! cat -n \| lpr <CR>
nn <F7> :w <bar> :!vim %<_in<left><left><left>
nn <F8> :w <bar> :!g++ % -o %< -std=c++11 -fsanitize=
    undefined -Wall -Wextra -Wshadow -DFOX && for i in
    %<_*.in; do echo go && ./%< < $i; done <CR>
nn <F9> :w <bar> :!g++ % -o %< -std=c++11 -fsanitize=
    undefined -Wall -Wextra -Wshadow -DFOX && echo '>>>'
    ' && ./%<

```

## 1.2 default code

```

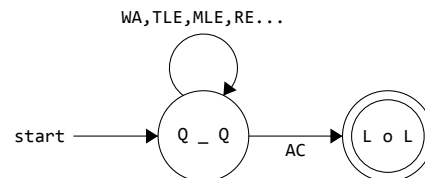
#pragma GCC optimize("Ofast")
#include <bits/stdc++.h>
#include <sys/time.h>
#include <sys/resource.h>
using namespace std;

void setstack(){
    // Set soft limit and hard limit to max
    const rlimit tmp {RLIM_INFINITY,RLIM_INFINITY};
    setrlimit(RLIMIT_STACK,&tmp);
}

int main(){
    #define name ""
    #ifndef FOX
    freopen(name".in","r",stdin);
    freopen(name".out","w",stdout);
    #endif
    static_assert(strlen(name));
    ios::sync_with_stdio(0);
    cin.tie(0), cout.tie(0);
}

```

## 1.3 state machine



## 2 Flow

### 2.1 Dinic

(a) Bounded Maxflow Construction:

1. add two node ss, tt
2. add\_edge(ss, tt, INF)
3. **for** each edge u -> v with capacity [l, r]:  
     add\_edge(u, tt, l)  
     add\_edge(ss, v, l)  
     add\_edge(u, v, r-l)
4. see (b), check **if** it is possible.
5. answer is maxflow(ss, tt) + maxflow(s, t)

(b) Bounded Possible Flow:

1. same construction method as (a)
2. run maxflow(ss, tt)
3. **for** every edge connected with ss **or** tt:  
     rule: check **if** their rest flow is exactly 0
4. answer is possible **if** every edge **do** satisfy the rule
5. otherwise, it is NOT possible.

(c) Bounded Minimum Flow:

1. same construction method as (a)
2. answer is maxflow(ss, tt)

(d) Bounded Minimum Cost Flow:

- \* the concept is somewhat like bounded possible flow.
- 1. same construction method as (a)
- 2. answer is maxflow(ss, tt) + ( $\sum 1 * \text{cost}$  **for** every edge)

(e) Minimum Cut:

1. run maxflow(s, t)
2. run cut(s)
3. ss[i] = 1: node i is at the same side with s.

```
const long long INF = 1LL<<60;
struct Dinic { //O(VVE), with minimum cut
    static const int MAXN = 5003;
    struct Edge{
        int u, v;
        long long cap, rest;
    };

    int n, m, s, t, d[MAXN], cur[MAXN];
    vector<Edge> edges;
    vector<int> G[MAXN];

    void init(){
        edges.clear();
        for ( int i = 0 ; i < MAXN ; i++ ) G[i].clear();
    }

    // min cut start
    bool side[MAXN];
    void cut(int u) {
        side[u] = 1;
        for ( int i : G[u] ) {
            if ( !side[ edges[i].v ] && edges[i].rest )
                cut(edges[i].v);
        }
    }
    // min cut end

    void add_edge(int u, int v, long long cap){
        edges.push_back( {u, v, cap, cap} );
        edges.push_back( {v, u, 0, 0LL} );
        m = edges.size();
        G[u].push_back(m-2);
        G[v].push_back(m-1);
    }
};
```

```
bool bfs(){
    memset(d, -1, sizeof(d));
    queue<int> que;
    que.push(s); d[s]=0;
    while (!que.empty()){
        int u = que.front(); que.pop();
        for (int ei : G[u]){
            Edge &e = edges[ei];
            if (d[e.v] < 0 && e.rest > 0){
                d[e.v] = d[u] + 1;
                que.push(e.v);
            }
        }
    }
    return d[t] >= 0;
}

long long dfs(int u, long long a){
    if ( u == t || a == 0 ) return a;
    long long flow = 0, f;
    for ( int &i=cur[u]; i < (int)G[u].size() ; i++ ) {
        Edge &e = edges[ G[u][i] ];
        if ( d[u] + 1 != d[e.v] ) continue;
        f = dfs(e.v, min(a, e.rest));
        if ( f > 0 ) {
            e.rest -= f;
            edges[ G[u][i]^1 ].rest += f;
            flow += f;
            a -= f;
            if ( a == 0 ) break;
        }
    }
    return flow;
}

long long maxflow(int _s, int _t){
    s = _s, t = _t;
    long long flow = 0, mf;
    while ( bfs() ){
        memset(cur, 0, sizeof(cur));
        while ( (mf = dfs(s, INF)) ) flow += mf;
    }
    return flow;
}
} dinic;
```

### 2.2 GomoryHu tree

Construct of Gomory Hu Tree

1. make sure the whole graph is clear
2. set node 0 as root, also be the parent of other nodes.
3. **for** every node i > 0, we run maxflow from i to parent[i]
4. hence we know the weight between i **and** parent[i]
5. **for** each node j > i, **if** j is at the same side with i, make the parent of j as i

```
int e[MAXN][MAXN];
int p[MAXN];

Dinic D; // original graph

void gomory_hu() {
    fill(p, p+n, 0);
    fill(e[0], e[n], INF);
    for ( int s = 1 ; s < n ; s++ ) {
        int t = p[s];
        Dinic F = D;
        int tmp = F.max_flow(s, t);

        for ( int i = 1 ; i < s ; i++ )
```

```

        e[s][i] = e[i][s] = min(tmp, e[t][i]);

    for ( int i = s+1 ; i <= n ; i++ )
        if ( p[i] == t && F.side[i] ) p[i] = s;
    }
}

```

## 2.3 min cost flow

```

// Long Long version
typedef pair<long long, long long> pll;
struct CostFlow {
    static const int MAXN = 350;
    static const long long INF = 1LL<<60;
    struct Edge {
        int to, r;
        long long rest, c;
    };
    int n, pre[MAXN], preL[MAXN]; bool inq[MAXN];
    long long dis[MAXN], fl, cost;
    vector<Edge> G[MAXN];
    void init() {
        for ( int i = 0 ; i < MAXN ; i++ ) G[i].clear();
    }
    void add_edge(int u, int v, long long rest, long long c) {
        G[u].push_back({v, (int)G[v].size(), rest, c});
        G[v].push_back({u, (int)G[u].size()-1, 0, -c});
    }
    pll flow(int s, int t) {
        fl = cost = 0;
        while (true) {
            fill(dis, dis+MAXN, INF);
            fill(inq, inq+MAXN, 0);
            dis[s] = 0;
            queue<int> que;
            que.push(s);
            while ( !que.empty() ) {
                int u = que.front(); que.pop();
                inq[u] = 0;
                for ( int i = 0 ; i < (int)G[u].size() ; i++ ) {
                    int v = G[u][i].to;
                    long long w = G[u][i].c;
                    if ( G[u][i].rest > 0 && dis[v] >
                        dis[u] + w ) {
                        pre[v] = u; preL[v] = i;
                        dis[v] = dis[u] + w;
                        if (!inq[v]) {
                            inq[v] = 1;
                            que.push(v);
                        }
                    }
                }
            }

            if (dis[t] == INF) break;
            long long tf = INF;
            for (int v = t, u, l ; v != s ; v = u ) {
                u = pre[v]; l = preL[v];
                tf = min(tf, G[u][l].rest);
            }
            for (int v = t, u, l ; v != s ; v = u ) {
                u = pre[v]; l = preL[v];
                G[u][l].rest -= tf;
                G[v][G[u][l].r].rest += tf;
            }
            cost += tf * dis[t];
            fl += tf;
        }
        return {fl, cost};
    }
} flow;

```

## 2.4 SW mincut 全點對最小割

```

// all pair min cut
// global min cut
struct SW{ // O(V^3)
    static const int MXN = 514;
    int n,vst[MXN],del[MXN];
    int edge[MXN][MXN],wei[MXN];
    void init(int _n){
        n = _n; FZ(edge); FZ(del);
    }
    void addEdge(int u, int v, int w){
        edge[u][v] += w; edge[v][u] += w;
    }
    void search(int &s, int &t){
        FZ(vst); FZ(wei);
        s = t = -1;
        while (true){
            int mx=-1, cur=0;
            for (int i=0; i<n; i++)
                if (!del[i] && !vst[i] && mx<wei[i])
                    cur = i, mx = wei[i];
            if (mx == -1) break;
            vst[cur] = 1;
            s = t; t = cur;
            for (int i=0; i<n; i++)
                if (!vst[i] && !del[i]) wei[i] += edge[cur][i];
        }
    }
    int solve(){
        int res = 2147483647;
        for (int i=0,x,y; i<n-1; i++){
            search(x,y);
            res = min(res,wei[y]);
            del[y] = 1;
            for (int j=0; j<n; j++)
                edge[x][j] = (edge[j][x] += edge[y][j]);
        }
        return res;
    }
}graph;

```

## 3 Matching

### 3.1 Hungarian

```

// Maximum Cardinality Bipartite Matching
// Worst case O(nm)

struct Graph{
    static const int MAXN = 5003;
    vector<int> G[MAXN];
    int n, match[MAXN], vis[MAXN];

    void init(int _n){
        n = _n;
        for (int i=0; i<n; i++) G[i].clear();
    }

    bool dfs(int u){
        for (int v:G[u]){
            if (vis[v]) continue;
            vis[v]=true;
            if (match[v]==-1 || dfs(match[v])){
                match[v] = u;
                match[u] = v;
                return true;
            }
        }
        return false;
    }

    int solve(){
        int res = 0;
    }
}

```

```

    memset(match, -1, sizeof(match));
    for (int i=0; i<n; i++){
        if (match[i]==-1){
            memset(vis, 0, sizeof(vis));
            if (dfs(i)) res++;
        }
    }
    return res;
}
} graph;

```

### 3.2 KM

```

const int MAX_N = 400 + 10;
const ll INF64 = 0x3f3f3f3f3f3f3f3fLL;
int n1, nr;
int pre[MAX_N];
ll slack[MAX_N];
ll W[MAX_N][MAX_N];
ll lx[MAX_N], ly[MAX_N];
int mx[MAX_N], my[MAX_N];
bool vx[MAX_N], vy[MAX_N];
void augment(int u) {
    if(!u) return;
    augment(mx[pre[u]]);
    mx[pre[u]] = u;
    my[u] = pre[u];
}
inline void match(int x) {
    queue<int> que;
    que.push(x);
    while(1) {
        while(!que.empty()) {
            x = que.front();
            que.pop();
            vx[x] = 1;
            REP1(y, 1, nr) {
                if(vy[y]) continue;
                ll t = lx[x] + ly[y] - W[x][y];
                if(t > 0) {
                    if(slack[y] >= t) slack[y] = t,
                        pre[y] = x;
                    continue;
                }
                pre[y] = x;
                if(!my[y]) {
                    augment(y);
                    return;
                }
                vy[y] = 1;
                que.push(my[y]);
            }
        }
        ll t = INF64;
        REP1(y, 1, nr) if(!vy[y]) t = min(t, slack[y]);
        REP1(x, 1, n1) if(vx[x]) lx[x] -= t;
        REP1(y, 1, nr) {
            if(vy[y]) ly[y] += t;
            else slack[y] -= t;
        }
        REP1(y, 1, nr) {
            if(vy[y] || slack[y]) continue;
            if(!my[y]) {
                augment(y);
                return;
            }
            vy[y] = 1;
            que.push(my[y]);
        }
    }
}
int main() {
    int m;
    RI(n1, nr, m);
    nr = max(n1, nr);

```

```

while(m--) {
    int x, y;
    ll w;
    RI(x, y, w);
    W[x][y] = w;
    lx[x] = max(lx[x], w);
}
REP1(i, 1, n1) {
    REP1(x, 1, n1) vx[x] = 0;
    REP1(y, 1, nr) vy[y] = 0, slack[y] = INF64;
    match(i);
}
ll ans = 0LL;
REP1(x, 1, n1) ans += W[x][mx[x]];
PL(ans);
REP1(x, 1, n1) printf("%d%c", W[x][mx[x]] ? mx[x] : 0, "\n"[x == n1]);
return 0;
}

```

### 3.3 Matching.txt

最大匹配 + 最小邊覆蓋 = V  
 最大獨立集 + 最小點覆蓋 = V  
 最大匹配 = 最小點覆蓋  
 最小路徑覆蓋數 = V - 最大匹配數

### 3.4 Maximum General Matching

```

// Maximum Cardinality Matching
struct Graph {
    vector<int> G[MAXN];
    int pa[MAXN], match[MAXN], st[MAXN], S[MAXN], vis[
        MAXN];
    int t, n;

    void init(int _n) {
        n = _n;
        for (int i = 1; i <= n; i++) G[i].clear();
    }

    void add_edge(int u, int v) {
        G[u].push_back(v);
        G[v].push_back(u);
    }

    int lca(int u, int v) {
        for (++t; ; swap(u, v)) {
            if (u == 0) continue;
            if (vis[u] == t) return u;
            vis[u] = t;
            u = st[pa[match[u]]];
        }
    }

    void flower(int u, int v, int l, queue<int> &q) {
        while (st[u] != 1) {
            pa[u] = v;
            if (S[v == match[u]] == 1) {
                q.push(v);
                S[v] = 0;
            }
            st[u] = st[v] = 1;
            u = pa[v];
        }
    }

    bool bfs(int u) {
        for (int i = 1; i <= n; i++) st[i] = i;
        memset(S, -1, sizeof(S));
        queue<int> q;
        q.push(u);
        S[u] = 0;
        while (!q.empty()) {
            u = q.front(); q.pop();
            for (int i = 0; i < (int)G[u].size(); i++) {

```

```

    int v = G[u][i];
    if ( S[v] == -1 ) {
        pa[v] = u;
        S[v] = 1;
        if ( !match[v] ) {
            for ( int lst ; u ; v = lst, u = pa[v] ) {
                lst = match[u];
                match[u] = v;
                match[v] = u;
            }
            return 1;
        }
        q.push(match[v]);
        S[ match[v] ] = 0;
    } else if ( !S[v] && st[v] != st[u] ) {
        int l = lca(st[v], st[u]);
        flower(v, u, l, q);
        flower(u, v, l, q);
    }
}
return 0;
}
int solve(){
    memset(pa, 0, sizeof(pa));
    memset(match, 0, sizeof(match));
    int ans = 0;
    for ( int i = 1 ; i <= n ; i++ )
        if ( !match[i] && bfs(i) ) ans++;
    return ans;
}
} graph;

```

```

        match[i] = i+1;
        match[i+1] = i;
    }
    while (true){
        int found = 0;
        for ( int i = 0 ; i < n ; i++ )
            onstk[ i ] = d[ i ] = 0;
        for ( int i = 0 ; i < n ; i++ ) {
            stk.clear();
            if ( !onstk[i] && SPFA(i) ) {
                found = 1;
                while ( stk.size() >= 2 ) {
                    int u = stk.back(); stk.
                        pop_back();
                    int v = stk.back(); stk.
                        pop_back();
                    match[u] = v;
                    match[v] = u;
                }
            }
        }
        if (!found) break;
    }
    int ret = 0;
    for ( int i = 0 ; i < n ; i++ )
        ret += e[i][match[i]];
    ret /= 2;
    return ret;
}
} graph;

```

### 3.5 Minimum General Weighted Matching

```

// Minimum Weight Perfect Matching (Perfect Match)
struct Graph {
    static const int MAXN = 105;
    int n, e[MAXN][MAXN];
    int match[MAXN], d[MAXN], onstk[MAXN];
    vector<int> stk;
    void init(int _n) {
        n = _n;
        for( int i = 0 ; i < n ; i ++ )
            for( int j = 0 ; j < n ; j ++ )
                e[i][j] = 0;
    }
    void add_edge(int u, int v, int w) {
        e[u][v] = e[v][u] = w;
    }
    bool SPFA(int u){
        if (onstk[u]) return true;
        stk.push_back(u);
        onstk[u] = 1;
        for ( int v = 0 ; v < n ; v++ ) {
            if (u != v && match[u] != v && !onstk[v] )
            {
                int m = match[v];
                if ( d[m] > d[u] - e[v][m] + e[u][v] )
                {
                    d[m] = d[u] - e[v][m] + e[u][v];
                    onstk[v] = 1;
                    stk.push_back(v);
                    if (SPFA(m)) return true;
                    stk.pop_back();
                    onstk[v] = 0;
                }
            }
        }
        onstk[u] = 0;
        stk.pop_back();
        return false;
    }
    int solve() {
        for ( int i = 0 ; i < n ; i += 2 ) {

```

## 4 Graph

- Maximum Independent Set
  - General: [NPC] maximum clique of complement of G
  - Bipartite Graph: [P] Maximum Cardinality Bipartite Matching
  - Tree: [P] dp
- Minimum Dominating Set
  - General: [NPC]
  - Bipartite Graph: [NPC]
  - Tree: [P] DP
- Minimum Vertex Cover
  - General: [NPC] (?)maximum clique of complement of G
  - Bipartite Graph: [P] Maximum Cardinality Bipartite Matching
  - Tree: [P] Greedy, from leaf to root
- Minimum Edge Cover
  - General: [P] V - Maximum Matching
  - Bipartite Graph: [P] Greedy, strategy: cover small degree node first.
  - (Min/Max)Weighted: [P]: Minimum/Minimum Weight Matching

### 4.1 BCC edge

邊雙連通

任意兩點間至少有兩條不重疊的路徑連接，找法：

1. 標記出所有的橋
2. 對全圖進行 DFS，不走橋，每一次 DFS 就是一個新的邊雙連通

// from BCW

```

struct BccEdge {
    static const int MXN = 100005;
    struct Edge { int v, eid; };
    int n, m, step, par[MXN], dfn[MXN], low[MXN];
    vector<Edge> E[MXN];

```

```

DisjointSet djs;
void init(int _n) {
    n = _n; m = 0;
    for (int i=0; i<n; i++) E[i].clear();
    djs.init(n);
}
void add_edge(int u, int v) {
    E[u].PB({v, m});
    E[v].PB({u, m});
    m++;
}
void DFS(int u, int f, int f_eid) {
    par[u] = f;
    dfn[u] = low[u] = step++;
    for (auto it:E[u]) {
        if (it.eid == f_eid) continue;
        int v = it.v;
        if (dfn[v] == -1) {
            DFS(v, u, it.eid);
            low[u] = min(low[u], low[v]);
        } else {
            low[u] = min(low[u], dfn[v]);
        }
    }
}
void solve() {
    step = 0;
    memset(dfn, -1, sizeof(int)*n);
    for (int i=0; i<n; i++) {
        if (dfn[i] == -1) DFS(i, i, -1);
    }
    djs.init(n);
    for (int i=0; i<n; i++) {
        if (low[i] < dfn[i]) djs.uni(i, par[i]);
    }
}
}graph;

```

## 4.2 Dijkstra

```

typedef struct Edge{
    int v; long long len;
    bool operator > (const Edge &b)const { return len>b
        .len; }
} State;

const long long INF = 1LL<<60;

void Dijkstra(int n, vector<Edge> G[], long long d[],
    int s, int t=-1){
    static priority_queue<State, vector<State>, greater
        <State> > pq;
    while ( pq.size() )pq.pop();
    for (int i=1; i<=n; i++)d[i]=INF;
    d[s]=0; pq.push( (State){s,d[s]} );
    while ( pq.size() ){
        auto x = pq.top(); pq.pop();
        int u = x.v;
        if (d[u]<x.len)continue;
        if (u==t)return;
        for (auto &e:G[u]){
            if (d[e.v] > d[u]+e.len){
                d[e.v] = d[u]+e.len;
                pq.push( (State) {e.v,d[e.v]} );
            }
        }
    }
}

```

## 4.3 Domination.txt

Maximum Independent Set  
General: [NPC] maximum clique of complement of G

Tree: [P] Greedy  
Bipartite Graph: [P] Maximum Cardinality Bipartite Matching  
-----  
Minimum Dominating Set  
General: [NPC]  
Tree: [P] DP  
Bipartite Graph: [NPC]  
-----  
Minimum Vertex Cover  
General: [NPC] (?)maximum clique of complement of G  
Tree: [P] Greedy, from leaf to root  
Bipartite Graph: [P] Maximum Cardinality Bipartite Matching  
-----  
Minimum Edge Cover  
General: [P] V - Maximum Matching  
Bipartite Graph: [P] Greedy, strategy: cover small degree node first.  
(Min/Max)Weighted: [P]: Minimum/Minimum Weight Matching

## 4.4 max clique

```

const int MAXN = 105;
int best;
int n;
int num[MAXN];
int path[MAXN];
int G[MAXN][MAXN];

bool dfs( int *adj, int total, int cnt ){
    int t[MAXN];
    if (total == 0){
        if( best < cnt ){
            best = cnt;
            return true;
        }
        return false;
    }
    for(int i = 0; i < total; i++){
        if( cnt+(total-i) <= best ) return false;
        if( cnt+num[adj[i]] <= best ) return false;
        int k=0;
        for(int j=i+1; j<total; j++)
            if(G[ adj[i] ][ adj[j] ])
                t[k++] = adj[j];
        if (dfs(t, k, cnt+1)) return true;
    }
    return false;
}

int MaximumClique(){
    int adj[MAXN];
    if (n <= 0) return 0;
    best = 0;
    for(int i = n-1; i >= 0; i--){
        int k=0;
        for(int j = i+1; j < n; j++)
            if (g[i][j]) adj[k++] = j;
        dfs( adj, k, 1 );
        num[i] = best;
    }
    return best;
}

```

## 4.5 min mean cycle

```

// from BCW
/* minimum mean cycle */
const int MAXE = 1805;
const int MAXN = 35;
const double inf = 1029384756;
const double eps = 1e-6;

```

```

struct Edge {
    int v,u;
    double c;
};
int n,m,prv[MAXN][MAXN], prve[MAXN][MAXN], vst[MAXN];
Edge e[MAXE];
vector<int> edgeID, cycle, rho;
double d[MAXN][MAXN];
inline void bellman_ford() {
    for(int i=0; i<n; i++) d[0][i]=0;
    for(int i=0; i<n; i++) {
        fill(d[i+1], d[i+1]+n, inf);
        for(int j=0; j<m; j++) {
            int v = e[j].v, u = e[j].u;
            if(d[i][v]<inf && d[i+1][u]>d[i][v]+e[j].c) {
                d[i+1][u] = d[i][v]+e[j].c;
                prv[i+1][u] = v;
                prve[i+1][u] = j;
            }
        }
    }
}
double karp_mmc() {
    // returns inf if no cycle, mmc otherwise
    double mmc=inf;
    int st = -1;
    bellman_ford();
    for(int i=0; i<n; i++) {
        double avg=-inf;
        for(int k=0; k<n; k++) {
            if(d[n][i]<inf-eps) avg=max(avg,(d[n][i]-d[k][i])/(n-k));
            else avg=max(avg,inf);
        }
        if (avg < mmc) tie(mmc, st) = tie(avg, i);
    }
    for(int i=0; i<n; i++) vst[i] = 0;
    edgeID.clear(); cycle.clear(); rho.clear();
    for (int i=n; !vst[st]; st=prv[i--][st]) {
        vst[st]++;
        edgeID.PB(prve[i][st]);
        rho.PB(st);
    }
    while (vst[st] != 2) {
        int v = rho.back(); rho.pop_back();
        cycle.PB(v);
        vst[v]++;
    }
    reverse(ALL(edgeID));
    edgeID.resize(SZ(cycle));
    return mmc;
}

```

## 4.6 SSSP related concepts

最短路問題分類：

三個工具 Bellman-Ford, Floyd, Dijkstra,

1. 可以把 Dijkstra Priority Queue 裡面存的東西想成「狀態」，他可以拿來統計甚至轉移。
2. 當遇到邊權會扣掉走的人的血量（或油量之類的），當不能有負值的時候，就要使用 Bellman-Ford 來做，一開始可以把起點設為最初的血量（油量），拿去做 Bellman-Ford，當做了  $n-1$  次之後，還能轉移，那就是有負環或正環（端看如何轉移 Bellman-Ford，這部分的轉移式很自由可以依照題目敘述亂改。）
3. 特別注意如果要判到某一個點的長度是不是無限小，可在做了  $n-1$  次之後，發現  $u \rightarrow v$  可以更新，那我可以去看  $v$  是否可以到另一點  $k$ ，如果是聯通的，代表  $k$  這個點的長度是無限小。

## 4.7 Tarjan.cpp

割點

點  $u$  為割點 **if and only if** 滿足 1. **or** 2.

1.  $u$  為樹根，且  $u$  有多於一個子樹。
2.  $u$  不為樹根，且滿足存在  $(u,v)$  為樹枝邊（或稱父子邊，即  $u$  為  $v$  在搜索樹中的父親），使得  $DFN(u) \leq Low(v)$ 。

橋

一條無向邊  $(u,v)$  是橋 **if and only if**  $(u,v)$  為樹枝邊，且滿足  $DFN(u) < Low(v)$ 。

// 0 base

```

struct TarjanSCC{
    static const int MAXN = 1000006;
    int n, dfn[MAXN], low[MAXN], scc[MAXN], scn, count;
    vector<int> G[MAXN];
    stack<int> stk;
    bool ins[MAXN];

    void tarjan(int u){
        dfn[u] = low[u] = ++count;
        stk.push(u);
        ins[u] = true;

        for(auto v:G[u]){
            if(!dfn[v]){
                tarjan(v);
                low[u] = min(low[u], low[v]);
            }else if(ins[v]){
                low[u] = min(low[u], dfn[v]);
            }
        }

        if(dfn[u] == low[u]){
            int v;
            do {
                v = stk.top();
                stk.pop();
                scc[v] = scn;
                ins[v] = false;
            } while(v != u);
            scn++;
        }
    }

    void getSCC(){
        memset(dfn,0,sizeof(dfn));
        memset(low,0,sizeof(low));
        memset(ins,0,sizeof(ins));
        memset(scc,0,sizeof(scc));
        count = scn = 0;
        for(int i = 0 ; i < n ; i++ ){
            if(!dfn[i]) tarjan(i);
        }
    }
}SCC;

```

## 4.8 2-SAT

const int MAXN = 2020;

```

struct TwoSAT{
    static const int MAXv = 2*MAXN;
    vector<int> GO[MAXv],BK[MAXv],stk;
    bool vis[MAXv];
    int SC[MAXv];

    void imply(int u,int v){ // u imply v
        GO[u].push_back(v);
        BK[v].push_back(u);
    }
}

```



```

    }
    int dfs(int u, vector<int>*G, int sc){
        vis[u]=1, SC[u]=sc;
        for (int v:G[u]) if (!vis[v])
            dfs(v, G, sc);
        if (G==GO) stk.push_back(u);
    }
    int scc(int n=MAXV){
        memset(vis, 0, sizeof(vis));
        for (int i=0; i<n; i++) if (!vis[i])
            dfs(i, GO, -1);
        memset(vis, 0, sizeof(vis));
        int sc=0;
        while (!stk.empty()){
            if (!vis[stk.back()])
                dfs(stk.back(), BK, sc++);
            stk.pop_back();
        }
    }
}SAT;

int main(){
    SAT.scc(2*n);
    bool ok=1;
    for (int i=0; i<n; i++){
        if (SAT.SC[2*i]==SAT.SC[2*i+1]) ok=0;
    }
    if (ok){
        for (int i=0; i<n; i++){
            if (SAT.SC[2*i]>SAT.SC[2*i+1]){
                cout << i << endl;
            }
        }
    }
    else puts("NO");
}

void warshall(){
    bitset<2003> d[2003];
    for (int k=0; k<n; k++){
        for (int i=0; i<n; i++) if (d[i][k]) {
            d[i] |= d[k];
        }
    }
}

```

## 4.9 平面圖判定

```

//skydog
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <iomanip>

#include <vector>
#include <cstring>
#include <string>
#include <queue>
#include <deque>
#include <stack>
#include <map>
#include <set>

#include <utility>
#include <list>

#include <cmath>
#include <algorithm>
#include <cassert>
#include <bitset>
#include <complex>
#include <climits>
#include <functional>
using namespace std;

typedef long long ll;

```

```

typedef pair<int, int> ii;
typedef pair<ll, ll> ll;

#define mp make_pair
#define pb push_back

#define debug(x) cerr << #x << " = " << x << " "

const int N=400+1;

struct Planar
{
    int n, m, hash[N], fa[N], deep[N], low[N], ecp[N];
    vector<int> g[N], son[N];
    set< pair<int, int> > SDlist[N], proots[N];
    int nxt[N][2], back[N], rev[N];
    deque<int> q;
    void dfs(int u)
    {
        hash[u]=1; q.pb(u);
        ecp[u]=low[u]=deep[u];
        int v;
        for (int i = 0; i < g[u].size(); ++i)
            if (!hash[v=g[u][i]])
            {
                fa[v]=u;
                deep[v]=deep[u]+1;
                dfs(v);
                low[u]=min(low[u], low[v]);
                SDlist[u].insert(mp(low[v], v));
            }
            else ecp[u]=min(ecp[u], deep[v]);
        low[u]=min(low[u], ecp[u]);
    }

    int visited[N];

    void addtree(int u, int t1, int v, int t2)
    {
        nxt[u][t1]=v; nxt[v][t2]=u;
    }

    void findnxt(int u, int v, int& u1, int& v1)
    {
        u1=nxt[u][v^1];
        if(nxt[u1][0]==u) v1=0;
        else v1=1;
    }

    void walkup(int u, int v)
    {
        back[v]=u;
        int v1=v, v2=v, u1=1, u2=0, z;
        for (;;)
        {
            if(hash[v1]==u || hash[v2]==u) break;
            hash[v1]=u; hash[v2]=u; z=max(v1, v2);
            if(z>n)
            {
                int p=fa[z-n];
                if(p!=u)
                {
                    proots[p].insert(mp(-low[z-n], z));
                    v1=p, v2=p, u1=0, u2=1;
                }
                else break;
            }
            else
            {
                findnxt(v1, u1, v1, u1);
                findnxt(v2, u2, v2, u2);
            }
        }
    }

    int topstack;
    pair<int, int> stack[N];
}

```



```

int outer(int u,int v)
{
    return ecp[v]<deep[u] || (SDlist[v].size() &&
        SDlist[v].begin()->first<deep[u]);
}

int inside(int u,int v)
{
    return proots[v].size()>0 || back[v]==u;
}

int active(int u,int v)
{
    return inside(u,v) || outer(u,v);
}

void push(int a,int b)
{
    stack[++topstack]=mp(a,b);
}

void mergestack()
{
    int v1,t1,v2,t2,s,s1;
    v1=stack[topstack].first;t1=stack[topstack].
        second;
    topstack--;
    v2=stack[topstack].first;t2=stack[topstack].
        second;
    topstack--;

    s=nxt[v1][t1^1];
    s1=(nxt[s][1]==v1);
    nxt[s][s1]=v2;
    nxt[v2][t2]=s;

    SDlist[v2].erase( make_pair(low[v1-n],v1-n) );
    proots[v2].erase( make_pair(-low[v1-n],v1) );
}

void findnxtActive(int u,int t,int& v,int& w1,int S
)
{
    findnxt(u,t,v,w1);
    while(u!=v && !active(S,v))
        findnxt(v,w1,v,w1);
}

void walkdown(int S,int u)
{
    topstack=0;
    int t1,v=S,w1,x2,y2,x1,y1,p;
    for(t1=0;t1<2;++t1)
    {
        findnxt(S,t1^1,v,w1);
        while(v!=S)
        {
            if(back[v]==u)
            {
                while(topstack>0) mergestack();
                addtree(S,t1,v,w1); back[v]=0;
            }
            if(proots[v].size())
            {
                push(v,w1);
                p=proots[v].begin()->second;
                findnxtActive(p,1,x1,y1,u);
                findnxtActive(p,0,x2,y2,u);
                if(active(u,x1) && !outer(u,x1))
                    v=x1,w1=y1;
                else if(active(u,x2) && !outer(u,x2)
                    )
                    v=x2,w1=y2;
                else if(inside(u,x1) || back[x1]==u
                    )
                    v=x1,w1=y1;
            }
        }
    }
}

```

```

        else v=x2,w1=y2;
        push(p,v==x2);
    }
    else if(v>n || ( ecp[v]>=deep[u] && !
        outer(u,v) ))
        findnxt(v,w1,v,w1);
    else if(v<=n && outer(u,v) && !topstack
        )
    {
        addtree(S,t1,v,w1); break;
    }
    else break;
}
}

int work(int u)
{
    int v;
    for (int i = 0; i < g[u].size(); ++i)
        if(fa[v=g[u][i]]==u)
        {
            son[u].push_back(n+v);
            proots[n+v].clear();
            addtree(n+v,1,v,0);
            addtree(n+v,0,v,1);
        }
    for (int i = 0; i < g[u].size(); ++i)
        if(deep[v=g[u][i]]>deep[u]+1)
            walkup(u,v);
    topstack=0;
    for (int i = 0; i < son[u].size(); ++i)
        walkdown(son[u][i], u);
    for (int i = 0; i < g[u].size(); ++i)
        if(deep[v=g[u][i]]>deep[u]+1 && back[v])
            return 0;
    return 1;
}

void init(int _n)
{
    n = _n;
    m = 0;
    for(int i=1;i<=2*n;++i)
    {
        g[i].clear();
        SDlist[i].clear();
        son[i].clear();
        proots[i].clear();
        nxt[i][0]=nxt[i][1]=0;
        fa[i]=0;
        hash[i]=0;low[i]=ecp[i]=deep[i]=back[i]=0;
        q.clear();
    }
}

void add(int u, int v)
{
    ++m;
    g[u].pb(v); g[v].pb(u);
}

bool check_planar()
{
    if(m>3*n-5)
        return false;
    // memset(hash,0,sizeof hash);
    for(int i=1;i<=n;++i)
        if(!hash[i])
        {
            deep[i]=1;
            dfs(i);
        }
    memset(hash,0,sizeof hash);
    //memset(hash, 0, (2*n+1)*sizeof(hash[0]));
    // originally only looks at last n element
    assert(q.size() == n);
    while (!q.empty())
    {

```

```

        if (!work(q.back()))
            return false;
        q.pop_back();
    }
    return true;
}
} base, _new;
vector<ii> edges;
int n, m;
inline void build(int n, Planar &_new)
{
    _new.init(n);
    for (auto e : edges)
        _new.add(e.first, e.second);
}
void end()
{
    puts("-1");
    exit(0);
}
bool vis[N];
const int maxp = 5;
int path[maxp], tp=0;
void dfs(int cur)
{
    vis[cur] = true;
    path[tp++] = cur;
    if (tp == maxp)
    {
        auto it = lower_bound(base.g[cur].begin(), base.g[
            cur].end(), path[0]);
        if (it != base.g[cur].end() && *it == path[0])
        {
            //a cycle
            int x = n+1;
            for (int i = 0; i < 5; ++i) edges.pb(mp(x,
                path[i]));
            build(x, _new);
            if (_new.check_planar())
            {
                for (int i = 0; i < maxp; ++i) printf(
                    "%d%c", path[i], i==maxp-1?'\\n':' ');
                exit(0);
            }
            for (int i = 0; i < 5; ++i) edges.pop_back
                ();
        }
        else
        {
            for (auto e : base.g[cur]) if (!vis[e]) dfs(e);
        }
    }
    vis[cur] = false;
    --tp;
}
int main()
{
    scanf("%d %d", &n, &m);
    if (n <= 4)
    {
        assert(false);
        puts("0"); return 0;
    }
    for (int i = 0; i < m; ++i)
    {
        int u, v; scanf("%d %d", &u, &v);
        edges.pb(mp(u, v));
    }
    build(n, base);
    if (!base.check_planar()) end();
    for (int i = 1; i <= n; ++i)
        sort(base.g[i].begin(), base.g[i].end());
    for (int i = 1; i <= n; ++i)
        dfs(i);
    end();
}

```

## 5 Math

- Stirling number of second kind  
 $S(n, m)$  :  $n$  個相異球, 放到  $m$  個相同的箱子, 每個箱子至少 1  
 $= m \times S(n-1, m) + S(n-1, m-1)$   
 $= \frac{1}{m!} \sum_{j=0}^m \binom{m}{j} (m-j)^n (-1)^j$
- Stirling number of first kind  
 $s(n, m)$  :  $n$  個相異球, 分配到  $m$  個有向環, 每個環至少 1  
 $s(n+1, m) = n \times s(n, m) + s(n, m-1)$   
 $s(n, m) \equiv \binom{\lfloor n/2 \rfloor}{m - \lfloor n/2 \rfloor} \pmod{2}$
- Pick's Theorem (Bangkok regional 2016 pD)  
 多邊形頂點都在整數點上  
 多邊形面積 = 內部整數點個數 + 邊上格子點個數/2 - 1  
 $A = i + b/2 - 1$

### 5.1 $ax+by=\gcd(a,b)$

```

pair<int,int> extgcd(int a, int b){
    if (b==0) return {1,0};
    int k = a/b;
    pair<int,int> p = extgcd(b,a-k*b);
    return { p.second, p.first - k*p.second };
}

```

### 5.2 FFT

```

// use llround() to avoid EPS
typedef double Double;
const Double PI = acos(-1);

// STL complex may TLE
typedef complex<Double> Complex;
#define x real()
#define y imag()

template<typename Iter> // Complex*
void BitReverse(Iter a, int n){
    for (int i=1, j=0; i<n; i++){
        for (int k = n>>1; k>(j^=k); k>>=1);
        if (i<j) swap(a[i],a[j]);
    }
}

template<typename Iter> // Complex*
void FFT(Iter a, int n, int rev=1){ // rev = 1 or -1
    assert( (n&(-n)) == n ); // n is power of 2
    BitReverse(a,n);
    Iter A = a;

    for (int s=1; (1<<s)<=n; s++){
        int m = (1<<s);

        Complex wm( cos(2*PI*rev/m), sin(2*PI*rev/m) );
        for (int k=0; k<n; k+=m){
            Complex w(1,0);
            for (int j=0; j<(m>>1); j++){
                Complex t = w * A[k+j+(m>>1)];
                Complex u = A[k+j];
                A[k+j] = u+t;
                A[k+j+(m>>1)] = u-t;
                w = w*wm;
            }
        }
    }

    if (rev==-1){
        for (int i=0; i<n; i++){
            A[i] /= n;
        }
    }
}

```

```
    }
}
```

### 5.3 GaussElimination

// by bcw\_codebook

```
const int MAXN = 300;
const double EPS = 1e-8;

int n;
double A[MAXN][MAXN];

void Gauss() {
    for(int i = 0; i < n; i++) {
        bool ok = 0;
        for(int j = i; j < n; j++) {
            if(fabs(A[j][i]) > EPS) {
                swap(A[j], A[i]);
                ok = 1;
                break;
            }
        }
        if(!ok) continue;

        double fs = A[i][i];
        for(int j = i+1; j < n; j++) {
            double r = A[j][i] / fs;
            for(int k = i; k < n; k++) {
                A[j][k] -= A[i][k] * r;
            }
        }
    }
}
```

### 5.4 inverse

```
const int MAXN = 1000006;
int inv[MAXN];
void invTable(int bound, int p){
    inv[1] = 1;
    for (int i=2; i<bound; i++){
        inv[i] = (long long)inv[p%i] * (p-p/i) %p;
    }
}

int inv(int b, int p){
    if (b==1) return 1;
    return (long long)inv(p%b,p) * (p-p/b) %p;
}
```

### 5.5 Miller-Rabin

```
typedef long long LL;

inline LL bin_mul(LL a, LL n, const LL& MOD){
    LL re=0;
    while (n>0){
        if (n&1) re += a;
        a += a; if (a>=MOD) a-=MOD;
        n>>=1;
    }
    return re%MOD;
}

inline LL bin_pow(LL a, LL n, const LL& MOD){
    LL re=1;
    while (n>0){
        if (n&1) re = bin_mul(re,a,MOD);
        a = bin_mul(a,a,MOD);
        n>>=1;
    }
}
```

```
    }
    return re;
}

bool is_prime(LL n){
    //static LL sprp[3] = { 2LL, 7LL, 61LL};
    static LL sprp[7] = { 2LL, 325LL, 9375LL,
        28178LL, 450775LL, 9780504LL,
        1795265022LL };
    if (n==1 || (n&1)==0 ) return n==2;
    int u=n-1, t=0;
    while ( (u&1)==0 ) u>>=1, t++;
    for (int i=0; i<3; i++){
        LL x = bin_pow( sprp[i]%n, u, n);
        if (x==0 || x==1 || x==n-1)continue;

        for (int j=1; j<t; j++){
            x=x*x%n;
            if (x==1 || x==n-1)break;
        }
        if (x==n-1)continue;
        return 0;
    }
    return 1;
}
```

### 5.6 Mobius

```
void mobius() {
    fill(isPrime, isPrime + MAXN, 1);
    mu[1] = 1, num = 0;
    for (int i = 2; i < MAXN; ++i) {
        if (isPrime[i]) primes[num++] = i, mu[i] = -1;
        static int d;
        for (int j = 0; j < num && (d = i * primes[j])
            < MAXN; ++j) {
            isPrime[d] = false;
            if (i % primes[j] == 0) {
                mu[d] = 0; break;
            } else mu[d] = -mu[i];
        }
    }
}
```

### 5.7 pollardRho

```
// from PEC
// does not work when n is prime
Int f(Int x, Int mod){
    return add(mul(x, x, mod), 1, mod);
}

Int pollard_rho(Int n) {
    if ( !(n & 1) ) return 2;
    while (true) {
        Int y = 2, x = rand()%(n-1) + 1, res = 1;
        for ( int sz = 2 ; res == 1 ; sz *= 2 ) {
            for ( int i = 0 ; i < sz && res <= 1 ; i++ ) {
                x = f(x, n);
                res = __gcd(abs(x-y), n);
            }
            y = x;
        }
        if ( res != 0 && res != n ) return res;
    }
}
```

### 5.8 SG

Anti Nim (取走最後一個石子者敗)

先手必勝 if and only if

1. 「所有」堆的石子數都為 1 且遊戲的 SG 值為 0。
2. 「有些」堆的石子數大於 1 且遊戲的 SG 值不為 0。

Anti-SG (決策集合為空的遊戲者贏)

定義 SG 值為 0 時，遊戲結束，  
則先手必勝 **if and only if**

1. 遊戲中沒有單一遊戲的 SG 函數大於 1 且遊戲的 SG 函數為 0。
2. 遊戲中某個單一遊戲的 SG 函數大於 1 且遊戲的 SG 函數不為 0。

Sprague-Grundy

1. 雙人、回合制
2. 資訊完全公開
3. 無隨機因素
4. 可在有限步內結束
5. 沒有和局
6. 雙方可採取的行動相同

SG(S) 的值為 0：後手(P)必勝  
不為 0：先手(N)必勝

```
int mex(set S) {
    // find the min number >= 0 that not in the S
    // e.g. S = {0, 1, 3, 4} mex(S) = 2
}
```

```
state = []
int SG(A) {
    if (A not in state) {
        S = sub_states(A)
        if( len(S) > 1 ) state[A] = reduce(operator.xor, [
            SG(B) for B in S])
        else state[A] = mex(set(SG(B) for B in next_states(
            A)))
    }
    return state[A]
}
```

## 5.9 theorem

```
/*
Lucas's Theorem
For non-negative integer n,m and prime P,
C(m,n) mod P = C(m/M,n/M) * C(m%M,n%M) mod P
= mult_i ( C(m_i,n_i) )
where m_i is the i-th digit of m in base P.
```

```
Kirchhoff's theorem
A_{ii} = deg(i), A_{ij} = (i,j) \in E ? -1 : 0
Deleting any one row, one column, and cal the det(A)
```

```
Nth Catalan recursive function:
C_0 = 1, C_{n+1} = C_n * 2(2n + 1)/(n+2)
```

```
Mobius Formula
u(n) = 1, if n = 1
      (-1)^m, 若 n 無平方數因數，且 n = p1*p2*p3
          *...*pk
      0, 若 n 有大於 1 的平方數因數
```

- Property
1. (積性函數)  $u(a)u(b) = u(ab)$
  2.  $\sum_{d|n} u(d) = [n == 1]$

```
Mobius Inversion Formula
if f(n) = \sum_{d|n} g(d)
then g(n) = \sum_{d|n} u(n/d)f(d)
          = \sum_{d|n} u(d)f(n/d)
- Application
```

```
the number/power of gcd(i, j) = k
- Trick
分塊, O(sqrt(n))
```

Chinese Remainder Theorem ( $m_i$  兩兩互質)

```
x = a_1 (mod m_1)
x = a_2 (mod m_2)
....
x = a_i (mod m_i)
```

construct a solution:

```
Let M = m_1 * m_2 * m_3 * ... * m_n
Let M_i = M / m_i
```

```
t_i = 1 / M_i
t_i * M_i = 1 (mod m_i)
```

```
solution x = a_1 * t_1 * M_1 + a_2 * t_2 * M_2 + ...
            + a_n * t_n * M_n + k * M
            = k*M + \sum a_i * t_i * M_i, k is positive integer.
```

```
under mod M, there is one solution x = \sum a_i * t_i *
M_i
```

Burnside's Lemma

$|G| * |X/G| = \sum (|X^g|) \text{ where } g \text{ in } G$

總方法數：每一種旋轉下不動點的個數總和 除以 旋轉的方法數

```
*/
```

## 6 Geometry

### 6.1 2D point template

```
typedef double Double;
struct Point {
    Double x,y;

    bool operator < (const Point &b)const{
        //return tie(x,y) < tie(b.x,b.y);
        //return atan2(y,x) < atan2(b.y,b.x);
        assert(0 && "choose compare");
    }
    Point operator + (const Point &b)const{
        return (Point){x+b.x,y+b.y};
    }
    Point operator - (const Point &b)const{
        return (Point){x-b.x,y-b.y};
    }
    Point operator * (const Double &d)const{
        return Point(d*x,d*y);
    }
    Double operator * (const Point &b)const{
        return x*b.x + y*b.y;
    }
    Double operator % (const Point &b)const{
        return x*b.y - y*b.x;
    }
    friend Double abs2(const Point &p){
        return p.x*p.x + p.y*p.y;
    }
    friend Double abs(const Point &p){
        return sqrt( abs2(p) );
    }
};
typedef Point Vector;

struct Line{
    Point P; Vector v;
    bool operator < (const Line &b)const{
        return atan2(v.y,v.x) < atan2(b.v.y,b.v.x);
    }
}
```

```
};
```

## 6.2 circumcentre

```
#include "2Dpoint.cpp"
```

```
Point circumcentre(Point &p0, Point &p1, Point &p2){
    Point a = p1-p0;
    Point b = p2-p0;
    Double c1 = abs2(a)*0.5;
    Double c2 = abs2(b)*0.5;
    Double d = a % b;
    Double x = p0.x + ( c1*b.y - c2*a.y ) / d;
    Double y = p0.y + ( c2*a.x - c1*b.x ) / d;
    return {x,y};
}
```

## 6.3 ConvexHull

```
#include "2Dpoint.cpp"
```

```
// return H, 第一個點會在 H 出現兩次
void ConvexHull(vector<Point> &P, vector<Point> &H){
    int n = P.size(), m=0;
    sort(P.begin(),P.end());
    H.clear();

    for (int i=0; i<n; i++){
        while (m>=2 && (P[i]-H[m-2]) % (H[m-1]-H[m-2])
            <0)H.pop_back(), m--;
        H.push_back(P[i]), m++;
    }

    for (int i=n-2; i>=0; i--){
        while (m>=2 && (P[i]-H[m-2]) % (H[m-1]-H[m-2])
            <0)H.pop_back(), m--;
        H.push_back(P[i]), m++;
    }
}
```

## 6.4 half plane intersection

```
bool OnLeft(const Line& L, const Point& p){
    return Cross(L.v, p-L.P)>0;
}
Point GetIntersection(Line a, Line b){
    Vector u = a.P-b.P;
    Double t = Cross(b.v, u)/Cross(a.v, b.v);
    return a.P + a.v*t;
}
int HalfplaneIntersection(Line* L, int n, Point* poly){
    sort(L, L+n);

    int first, last;
    Point *p = new Point[n];
    Line *q = new Line[n];
    q[first=last=0] = L[0];
    for(int i=1; i<n; i++){
        while(first < last && !OnLeft(L[i], p[last-1])) last--;
        while(first < last && !OnLeft(L[i], p[first])) first++;
        q[++last]=L[i];
        if(fabs(Cross(q[last].v, q[last-1].v))<EPS){
            last--;
            if(OnLeft(q[last], L[i].P)) q[last]=L[i];
        }
        if(first < last) p[last-1]=GetIntersection(q[last-1], q[last]);
    }
}
```

```
while(first<last && !OnLeft(q[first], p[last-1])) last--;
if(last-first<=1) return 0;
p[last]=GetIntersection(q[last], q[first]);

int m=0;
for(int i=first; i<=last; i++) poly[m++]=p[i];
return m;
}
```

## 6.5 Intersection of two circle

```
vector<Double> interCircle(Double o1, Double r1, Double
    o2, Double r2) {
    Double d2 = abs2(o1 - o2);
    Double d = sqrt(d2);
    if (d < fabs(r1-r2) || r1+r2 < d) return {};
    Double u = 0.5*(o1+o2) + ((r2*r2-r1*r1)/(2.0*d2))*(o1
        -o2);
    Double A = sqrt((r1+r2+d) * (r1-r2+d) * (r1+r2-d) *
        (-r1+r2+d));
    Double v = A / (2.0*d2) * Double(o1.S-o2.S, -o1.F+o2.
        F);
    return {u+v, u-v};
}
```

## 6.6 Intersection of two lines

```
Point interPnt(Point p1, Point p2, Point q1, Point q2,
    bool &res){
    Double f1 = cross(p2, q1, p1);
    Double f2 = -cross(p2, q2, p1);
    Double f = (f1 + f2);

    if(fabs(f) < EPS) {
        res = false;
        return {};
    }

    res = true;
    return (f2 / f) * q1 + (f1 / f) * q2;
}
```

## 6.7 Smallest Circle

```
#include "circumcentre.cpp"
pair<Point, Double> SmallestCircle(int n, Point _p[]){
    Point *p = new Point[n];
    memcpy(p, _p, sizeof(Point)*n);
    random_shuffle(p, p+n);

    Double r2=0;
    Point cen;
    for (int i=0; i<n; i++){
        if ( abs2(cen-p[i]) <= r2)continue;
        cen = p[i], r2=0;
        for (int j=0; j<i; j++){
            if ( abs2(cen-p[j]) <= r2)continue;
            cen = (p[i]+p[j])*0.5;
            r2 = abs2(cen-p[i]);
            for (int k=0; k<j; k++){
                if ( abs2(cen-p[k]) <= r2)continue;
                cen = circumcentre(p[i], p[j], p[k]);
                r2 = abs2(cen-p[k]);
            }
        }
    }

    delete[] p;
    return {cen, r2};
}
// auto res = SmallestCircle(,);
```

## 7 String

### 7.1 AC automaton

```
// remember make_fail() !!!
// notice MLE

const int sigma = 62;
const int MAXC = 200005;

inline int idx(char c){
    if ('A' <= c && c <= 'Z') return c - 'A';
    if ('a' <= c && c <= 'z') return c - 'a' + 26;
    if ('0' <= c && c <= '9') return c - '0' + 52;
}

struct ACautomaton{
    struct Node{
        Node *next[sigma], *fail;
        int cnt; // dp
        Node(){
            memset(next, 0, sizeof(next));
            fail = 0;
            cnt = 0;
        }
    } buf[MAXC], *bufp, *ori, *root;

    void init(){
        bufp = buf;
        ori = new (bufp++) Node();
        root = new (bufp++) Node();
    }

    void insert(int n, char *s){
        Node *ptr = root;
        for (int i=0; s[i]; i++){
            int c = idx(s[i]);
            if (ptr->next[c] == NULL)
                ptr->next[c] = new (bufp++) Node();
            ptr = ptr->next[c];
        }
        ptr->cnt++;
    }

    Node* trans(Node *o, int c){
        while (o->next[c] == NULL) o = o->fail;
        return o->next[c];
    }

    void make_fail(){
        static queue<Node*> que;

        for (int i=0; i<sigma; i++)
            ori->next[i] = root;
        root->fail = ori;

        que.push(root);
        while (que.size()){
            Node *u = que.front(); que.pop();
            for (int i=0; i<sigma; i++){
                if (u->next[i] == NULL) continue;
                u->next[i]->fail = trans(u->fail, i);
                que.push(u->next[i]);
            }
            u->cnt += u->fail->cnt;
        }
    }
} ac;
```

### 7.2 KMP

```
template<typename T>
void build_KMP(int n, T *s, int *f){ // 1 base
    f[0] = -1, f[1] = 0;
```

```
    for (int i=2; i<=n; i++){
        int w = f[i-1];
        while (w>0 && s[w+1] != s[i]) w = f[w];
        f[i] = w+1;
    }
}

template<typename T>
int KMP(int n, T *a, int m, T *b){
    build_KMP(m, b, f);
    int ans = 0;

    for (int i=1, w=0; i<=n; i++){
        while (w>0 && b[w+1] != a[i]) w = f[w];
        w++;
        if (w==m){
            ans++;
            w = f[w];
        }
    }
    return ans;
}
```

### 7.3 palindromic tree

```
// remember init() !!!
// remember make_fail() !!!
// insert s need 1 base !!!
// notice MLE

const int sigma = 62;
const int MAXC = 1000006;
inline int idx(char c){
    if ('a' <= c && c <= 'z') return c - 'a';
    if ('A' <= c && c <= 'Z') return c - 'A' + 26;
    if ('0' <= c && c <= '9') return c - '0' + 52;
}

struct PalindromicTree{
    struct Node{
        Node *next[sigma], *fail;
        int len, cnt; // for dp
        Node(){
            memset(next, 0, sizeof(next));
            fail = 0;
            len = cnt = 0;
        }
    } buf[MAXC], *bufp, *even, *odd;

    void init(){
        bufp = buf;
        even = new (bufp++) Node();
        odd = new (bufp++) Node();
        even->fail = odd;
        odd->len = -1;
    }

    void insert(char *s){
        Node* ptr = even;
        for (int i=1; s[i]; i++){
            ptr = extend(ptr, s[i]);
        }
    }

    Node* extend(Node *o, char *ptr){
        int c = idx(*ptr);
        while (*ptr != *(ptr-1-o->len)) o = o->fail;
        Node *&np = o->next[c];
        if (!np){
            np = new (bufp++) Node();
            np->len = o->len+2;
            Node *f = o->fail;
            if (f){
                while (*ptr != *(ptr-1-f->len)) f = f->fail;
                np->fail = f->next[c];
            }
            else {
```

```

        np->fail = even;
    }
    np->cnt = np->fail->cnt;
}
np->cnt++;
return np;
}
} PAM;

```

## 7.4 SAM

```

// par : fail link
// val : a topological order ( useful for DP )
// go[x] : automata edge ( x is integer in [0,26) )

struct SAM{
    struct State{
        int par, go[26], val;
        State () : par(0), val(0){ FZ(go); }
        State (int _val) : par(0), val(_val){ FZ(go); }
    };
    vector<State> vec;
    int root, tail;

    void init(int arr[], int len){
        vec.resize(2);
        vec[0] = vec[1] = State(0);
        root = tail = 1;
        for (int i=0; i<len; i++)
            extend(arr[i]);
    }
    void extend(int w){
        int p = tail, np = vec.size();
        vec.PB(State(vec[p].val+1));
        for ( ; p && vec[p].go[w]==0; p=vec[p].par)
            vec[p].go[w] = np;
        if (p == 0){
            vec[np].par = root;
        } else {
            if (vec[vec[p].go[w]].val == vec[p].val+1){
                vec[np].par = vec[p].go[w];
            } else {
                int q = vec[p].go[w], r = vec.size();
                vec.PB(vec[q]);
                vec[r].val = vec[p].val+1;
                vec[q].par = vec[np].par = r;
                for ( ; p && vec[p].go[w] == q; p=vec[p].par)
                    vec[p].go[w] = r;
            }
        }
        tail = np;
    }
};

```

## 7.5 smallest rotation

```

string mcp(string s){
    int n = s.length();
    s += s;
    int i=0, j=1;
    while (i<n && j<n){
        int k = 0;
        while (k < n && s[i+k] == s[j+k]) k++;
        if (s[i+k] <= s[j+k]) j += k+1;
        else i += k+1;
        if (i == j) j++;
    }
    int ans = i < n ? i : j;
    return s.substr(ans, n);
}

```

Contact GitHub API Training Shop Blog About

## 7.6 suffix array

*/\*he[i]保存了在后缀数组中相邻两个后缀的最长公共前缀长度  
\*sa[i]表示的是字典序排名为i的后缀是谁 (字典序越小的排名越靠前)  
\*rk[i]表示的是后缀我所对应的排名是多少 \*/*

```

const int MAX = 1020304;
int ct[MAX], he[MAX], rk[MAX];
int sa[MAX], tsa[MAX], tp[MAX][2];
void suffix_array(char *ip){
    int len = strlen(ip);
    int alp = 256;
    memset(ct, 0, sizeof(ct));
    for(int i=0; i<len; i++) ct[ip[i]+1]++;
    for(int i=1; i<alp; i++) ct[i]+=ct[i-1];
    for(int i=0; i<len; i++) rk[i]=ct[ip[i]];
    for(int i=1; i<len; i*=2){
        for(int j=0; j<len; j++){
            if(j+i>len) tp[j][1]=0;
            else tp[j][1]=rk[j+i]+1;
            tp[j][0]=rk[j];
        }
        memset(ct, 0, sizeof(ct));
        for(int j=0; j<len; j++) ct[tp[j][1]+1]++;
        for(int j=1; j<len+2; j++) ct[j]+=ct[j-1];
        for(int j=0; j<len; j++) tsa[ct[tp[j][1]]+j]=j;
        memset(ct, 0, sizeof(ct));
        for(int j=0; j<len; j++) ct[tp[j][0]+1]++;
        for(int j=1; j<len+1; j++) ct[j]+=ct[j-1];
        for(int j=0; j<len; j++)
            sa[ct[tp[tsa[j]][0]]+j]=tsa[j];
        rk[sa[0]]=0;
        for(int j=1; j<len; j++){
            if (tp[sa[j]][0] == tp[sa[j-1]][0] &&
                tp[sa[j]][1] == tp[sa[j-1]][1])
                rk[sa[j]] = rk[sa[j-1]];
            else
                rk[sa[j]] = j;
        }
    }
    for(int i=0, h=0; i<len; i++){
        if(rk[i]==0) h=0;
        else{
            int j=sa[rk[i]-1];
            h=max(0, h-1);
            for(; ip[i+h]==ip[j+h]; h++);
        }
        he[rk[i]]=h;
    }
}

```

## 7.7 Z value

```

z[0] = 0;
for ( int bst = 0, i = 1; i < len ; i++ ) {
    if ( z[bst] + bst <= i ) z[i] = 0;
    else z[i] = min(z[i - bst], z[bst] + bst - i);
    while ( str[i + z[i]] == str[z[i]] ) z[i]++;
    if ( i + z[i] > bst + z[bst] ) bst = i;
}

```

*// 回文版*

```

void Zpal(const char *s, int len, int *z) {
    // Only odd palindrome len is considered
    // z[i] means that the longest odd palindrom
    // centered at
    // i is [i-z[i] .. i+z[i]]
    z[0] = 0;
    for (int b=0, i=1; i<len; i++) {
        if (z[b] + b >= i) z[i] = min(z[2*b-i], b+z[b]-i);
    }
}

```



```

    else z[i] = 0;
    while (i+z[i]+1 < len and i-z[i]-1 >= 0 and
           s[i+z[i]+1] == s[i-z[i]-1]) z[i] ++;
    if (z[i] + i > z[b] + b) b = i;
}
}

```

## 7.8 BWT (Burrows-Wheeler Transform)

```

string BWT(string); // by suffix array

string iBWT(string &s, int start=0){
    int n = (int) s.size();
    string ret(n, ' ');
    vector<int> next(n,0), box[256];

    for (int i=0; i<n; i++) // bucket sort
        box[ (int)s[i] ].push_back(i);

    for (int i=0, j=0; i<256; i++)
        for (int x:box[i])
            next[j++] = x;

    for (int i=0, p=start; i<n; i++)
        ret[i] = s[ p=next[p] ];

    return ret;
}

```

## 8 Data structure

### 8.1 2D range tree

```

// remember sort x !!!!!
typedef int T;
const int LGN = 20;
const int MAXN = 100005;

struct Point{
    T x, y;
    friend bool operator < (Point a, Point b){
        return tie(a.x,a.y) < tie(b.x,b.y);
    }
};

struct TREE{
    Point pt;
    int toleft;
}tree[LGN][MAXN];

struct SEG{
    T mx, Mx;
    int sz;
    TREE *st;
}seg[MAXN*4];

vector<Point> P;

void build(int l, int r, int o, int deep){
    seg[o].mx = P[l].x;
    seg[o].Mx = P[r].x;
    seg[o].sz = r-l+1;

    if(l == r){
        tree[deep][r].pt = P[r];
        tree[deep][r].toleft = 0;
        seg[o].st = &tree[deep][r];
        return;
    }
    int mid = (l+r)>>1;
    build(l,mid,o+o,deep+1);
    build(mid+1,r,o+o+1,deep+1);

    TREE *ptr = &tree[deep][l];

```

```

    TREE *pl = &tree[deep+1][l], *nl = &tree[deep+1][
        mid+1];
    TREE *pr = &tree[deep+1][mid+1], *nr = &tree[deep
        +1][r+1];

    int cnt = 0;
    while(pl != nl && pr != nr) {
        *(ptr) = pl->pt.y <= pr->pt.y ? cnt++, *(pl++):
            *(pr++);
        ptr -> toleft = cnt; ptr++;
    }
    while(pl != nl) *(ptr) = *(pl++), ptr -> toleft =
        ++cnt, ptr++;
    while(pr != nr) *(ptr) = *(pr++), ptr -> toleft =
        cnt, ptr++;
}

int main(){
    int n; cin >> n;
    for(int i = 0 ; i < n; i++){
        T x,y; cin >> x >> y;
        P.push_back((Point){x,y});
    }
    sort(P.begin(),P.end());
    build(0,n-1,1,0);
}

```

### 8.2 ext heap

```

#include <bits/extc++.h>
typedef __gnu_pbds::priority_queue<int> heap_t;
heap_t a,b;

int main() {
    a.clear();
    b.clear();
    a.push(1);
    a.push(3);
    b.push(2);
    b.push(4);
    assert(a.top() == 3);
    assert(b.top() == 4);
    // merge two heap
    a.join(b);
    assert(a.top() == 4);
    assert(b.empty());

    return 0;
}

```

### 8.3 KD tree

```

// from BCW

const int MXN = 100005;

struct KDTree {
    struct Node {
        int x,y,x1,y1,x2,y2;
        int id,f;
        Node *L, *R;
    }tree[MXN];
    int n;
    Node *root;

    long long dis2(int x1, int y1, int x2, int y2) {
        long long dx = x1-x2;
        long long dy = y1-y2;
        return dx*dx+dy*dy;
    }

    static bool cmpx(Node& a, Node& b){ return a.x<b.x; }
    static bool cmpy(Node& a, Node& b){ return a.y<b.y; }
    void init(vector<pair<int,int>> ip) {

```

```

n = ip.size();
for (int i=0; i<n; i++) {
    tree[i].id = i;
    tree[i].x = ip[i].first;
    tree[i].y = ip[i].second;
}
root = build_tree(0, n-1, 0);
}
Node* build_tree(int L, int R, int dep) {
    if (L>R) return nullptr;
    int M = (L+R)/2;
    tree[M].f = dep%2;
    nth_element(tree+L, tree+M, tree+R+1, tree[M].f ?
        cmpx : cmpy);
    tree[M].x1 = tree[M].x2 = tree[M].x;
    tree[M].y1 = tree[M].y2 = tree[M].y;

    tree[M].L = build_tree(L, M-1, dep+1);
    if (tree[M].L) {
        tree[M].x1 = min(tree[M].x1, tree[M].L->x1);
        tree[M].x2 = max(tree[M].x2, tree[M].L->x2);
        tree[M].y1 = min(tree[M].y1, tree[M].L->y1);
        tree[M].y2 = max(tree[M].y2, tree[M].L->y2);
    }

    tree[M].R = build_tree(M+1, R, dep+1);
    if (tree[M].R) {
        tree[M].x1 = min(tree[M].x1, tree[M].R->x1);
        tree[M].x2 = max(tree[M].x2, tree[M].R->x2);
        tree[M].y1 = min(tree[M].y1, tree[M].R->y1);
        tree[M].y2 = max(tree[M].y2, tree[M].R->y2);
    }

    return tree+M;
}
int touch(Node* r, int x, int y, long long d2){
    long long dis = sqrt(d2)+1;
    if (x<r->x1-dis || x>r->x2+dis || y<r->y1-dis || y>
        r->y2+dis)
        return 0;
    return 1;
}
void nearest(Node* r, int x, int y, int &mID, long
    long &md2) {
    if (!r || !touch(r, x, y, md2)) return;
    long long d2 = dis2(r->x, r->y, x, y);
    if (d2 < md2 || (d2 == md2 && mID < r->id)) {
        mID = r->id;
        md2 = d2;
    }
    // search order depends on split dim
    if ((r->f == 0 && x < r->x) ||
        (r->f == 1 && y < r->y)) {
        nearest(r->L, x, y, mID, md2);
        nearest(r->R, x, y, mID, md2);
    } else {
        nearest(r->R, x, y, mID, md2);
        nearest(r->L, x, y, mID, md2);
    }
}
int query(int x, int y) {
    int id = 1029384756;
    long long d2 = 102938475612345678LL;
    nearest(root, x, y, id, d2);
    return id;
}
}tree;

```

## 8.4 Link-Cut tree

// from bcw codebook

```

const int MXN = 100005;
const int MEM = 100005;
struct Splay {

```

```

    static Splay nil, mem[MEM], *pmem;
    Splay *ch[2], *f;
    int val, rev, size;
    Splay () : val(-1), rev(0), size(0) {
        f = ch[0] = ch[1] = &nil;
    }
    Splay (int _val) : val(_val), rev(0), size(1) {
        f = ch[0] = ch[1] = &nil;
    }
    bool isr() {
        return f->ch[0] != this && f->ch[1] != this;
    }
    int dir() {
        return f->ch[0] == this ? 0 : 1;
    }
    void setCh(Splay *c, int d) {
        ch[d] = c;
        if (c != &nil) c->f = this;
        pull();
    }
    void push() {
        if (rev) {
            swap(ch[0], ch[1]);
            if (ch[0] != &nil) ch[0]->rev ^= 1;
            if (ch[1] != &nil) ch[1]->rev ^= 1;
            rev=0;
        }
    }
    void pull() {
        size = ch[0]->size + ch[1]->size + 1;
        if (ch[0] != &nil) ch[0]->f = this;
        if (ch[1] != &nil) ch[1]->f = this;
    }
} Splay::nil, Splay::mem[MEM], *Splay::pmem = Splay::
    mem;
Splay *nil = &Splay::nil;

void rotate(Splay *x) {
    Splay *p = x->f;
    int d = x->dir();
    if (!p->isr()) p->f->setCh(x, p->dir());
    else x->f = p->f;
    p->setCh(x->ch[!d], d);
    x->setCh(p, !d);
    p->pull(); x->pull();
}

vector<Splay*> splayVec;
void splay(Splay *x) {
    splayVec.clear();
    for (Splay *q=x;; q=q->f) {
        splayVec.push_back(q);
        if (q->isr()) break;
    }
    reverse(begin(splayVec), end(splayVec));
    for (auto it : splayVec) it->push();
    while (!x->isr()) {
        if (x->f->isr()) rotate(x);
        else if (x->dir()==x->f->dir()) rotate(x->f), rotate
            (x);
        else rotate(x), rotate(x);
    }
}

Splay* access(Splay *x) {
    Splay *q = nil;
    for (;x!=nil;x=x->f) {
        splay(x);
        x->setCh(q, 1);
        q = x;
    }
    return q;
}

void evert(Splay *x) {
    access(x);
    splay(x);
    x->rev ^= 1;

```

```

    x->push(); x->pull();
}
void link(Splay *x, Splay *y) {
    // evert(x);
    access(x);
    splay(x);
    evert(y);
    x->setCh(y, 1);
}
void cut(Splay *x, Splay *y) {
    // evert(x);
    access(y);
    splay(y);
    y->push();
    y->ch[0] = y->ch[0]->f = nil;
}

int N, Q;
Splay *vt[MXN];

int ask(Splay *x, Splay *y) {
    access(x);
    access(y);
    splay(x);
    int res = x->f->val;
    if (res == -1) res=x->val;
    return res;
}

int main(int argc, char** argv) {
    scanf("%d%d", &N, &Q);
    for (int i=1; i<=N; i++)
        vt[i] = new (Splay::pmem++) Splay(i);
    while (Q--) {
        char cmd[105];
        int u, v;
        scanf("%s", cmd);
        if (cmd[1] == 'i') {
            scanf("%d%d", &u, &v);
            link(vt[v], vt[u]);
        } else if (cmd[0] == 'c') {
            scanf("%d", &v);
            cut(vt[1], vt[v]);
        } else {
            scanf("%d%d", &u, &v);
            int res=ask(vt[u], vt[v]);
            printf("%d\n", res);
        }
    }
    return 0;
}

```

## 8.5 Treap Lin

```

#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <string.h>
using namespace std;
const int INF = 999999999;
int ran(){
    static unsigned x = 20170928;
    return x = 0xdefaced*x+1;
}
struct Treap{
    Treap *l,*r;
    int num,m,sz,tag,ra,ad;
    Treap(int a){
        l=r=NULL;
        num=m=a;
        sz=1;
        tag=ad=0;
        ra = ran();
    }
}*head,*tp;

```

```

int size(Treap *a){
    return a ? a->sz : 0;
}
int min(Treap *a){
    return a ? a->m+a->ad : INF;
}
int add(Treap *a){
    return a ? a->ad : 0;
}
void push(Treap *a){
    if(!a) return;
    if(a->tag){
        swap(a->l,a->r);
        if(a->l)a->l->tag ^= 1;
        if(a->r)a->r->tag ^= 1;
        a->tag=0;
    }
    if(a->l)a->l->ad += a->ad;
    if(a->r)a->r->ad += a->ad;
    a->num += a->ad;
    a->m += a->ad;
    a->ad = 0;
}
void pull(Treap *a){
    if(!a) return;
    a->sz=1+size(a->l)+size(a->r);
    a->m = min( a->num, min( min(a->l), min(a->r) ) );
}

Treap* merge(Treap *a, Treap *b){
    if(!a || !b) return a ? a : b;
    if(a->ra > b->ra){
        push(a);
        a->r = merge(a->r,b);
        pull(a);
        return a;
    }else{
        push(b);
        b->l = merge(a,b->l);
        pull(b);
        return b;
    }
}

void split (Treap *o, Treap *&a, Treap *&b,int k){
    if(!k) a=NULL, b=o;
    else if(size(o)==k) a=o, b=NULL;
    else{
        push(o);
        if(k <= size(o->l)){
            b = o;
            split(o->l, a, b->l,k);
            pull(b);
        }else{
            a = o;
            split(o->r, a->r, b, k-size(o->l)-1);
            pull(a);
        }
    }
}

int main(){
    int n,tmp;
    scanf("%d",&n);
    for(int i = 0 ; i < n ; i++){
        scanf("%d",&tmp);
        tp = new Treap(tmp);
        head = merge(head,tp);
    }
    int Q;
    scanf("%d\n",&Q);
    char ss[50];
    int a, b, c;
    Treap *ta, *tb, *tc, *td;
    while(Q--){
        scanf("%s",ss);
        if(strcmp(ss,"ADD")==0){
            scanf("%d %d %d",&a,&b,&c);
            split(head,tb,tc,b);

```

```

        split(tb,ta,tb,a-1);
        tb -> ad += c;
        head = merge(ta, merge(tb, tc));
    }else if(strcmp(ss,"REVERSE")==0){
        scanf("%d %d",&a,&b);
        split(head,tb,tc,b);
        split(tb,ta,tb,a-1);
        tb -> tag ^= 1;
        head = merge(ta, merge(tb, tc));
    }else if(strcmp(ss,"REVOLVE")==0){
        scanf("%d %d %d",&a,&b,&c);
        split(head,tb,tc,b);
        split(tb,ta,tb,a-1);
        int szz = size(tb);
        c %= szz;
        split(tb,tb,td,szz-c);
        tb=merge(td,tb);
        head = merge(ta, merge(tb, tc));
    }else if(strcmp(ss,"INSERT")==0){
        scanf("%d %d",&a,&b);
        split(head,ta,tc,a);
        tb = new Treap(b);
        head = merge(ta, merge(tb, tc));
    }else if(strcmp(ss,"DELETE")==0){
        scanf("%d",&a);
        split(head,ta,tc,a-1);
        split(tc,tb,tc,1);
        delete tb;
        head = merge(ta,tc);
    }else if(strcmp(ss,"MIN")==0){
        scanf("%d %d",&a,&b);
        split(head,tb,tc,b);
        split(tb,ta,tb,a-1);
        printf("%d\n",min(tb));
        head = merge(ta, merge(tb, tc));
    }
}
}
}

```

## 9 Other

### 9.1 count spanning tree

新的方法介绍

下面我们介绍一种新的方法——Matrix-Tree定理(Kirchhoff矩阵-树定理)。

Matrix-Tree定理是解决生成树计数问题最有力的武器之一。它首先于1847年被Kirchhoff证明。在介绍定理之前，我们首先明确几个概念：

1、G的度数矩阵D[G]是一个n\*n的矩阵，并且满足：当i≠j时，dij=0；当i=j时，dij等于vi的度数。

2、G的邻接矩阵A[G]也是一个n\*n的矩阵，并且满足：如果vi、vj之间有边直接相连，则aij=1，否则为0。

我们定义G的Kirchhoff矩阵(也称为拉普拉斯算子)C[G]为C[G]=D[G]-A[G]，

则Matrix-Tree定理可以描述为：G的所有不同的生成树的个数等于其Kirchhoff矩阵C[G]任何一个n-1阶主子式的行列式的绝对值。

所谓n-1阶主子式，就是对于r(1≤r≤n)，将C[G]的第r行、第r列同时去掉后得到的新矩阵，用Cr[G]表示。

生成树计数

算法步骤：

1、构建拉普拉斯矩阵

Matrix[i][j] =  
degree(i), i=j  
-1, i-j有边  
0, 其他情况

2、去掉第r行，第r列 (r任意)

3、计算矩阵的行列式

```

/* *****
MYID   : Chen Fan
LANG   : G++
PROG   : Count_Spaning_Tree_From_Kuangbin
***** */
#include <stdio.h>
#include <string.h>
#include <algorithm>
#include <iostream>
#include <math.h>
using namespace std;
const double eps = 1e-8;
const int MAXN = 110;
int sgn(double x)
{
    if(fabs(x) < eps)return 0;
    if(x < 0)return -1;
    else return 1;
}
double b[MAXN][MAXN];
double det(double a[][MAXN],int n)
{
    int i, j, k, sign = 0;
    double ret = 1;
    for(i = 0; i < n; i++)
        for(j = 0; j < n; j++) b[i][j] = a[i][j];
    for(i = 0; i < n; i++)
    {
        if(sgn(b[i][i]) == 0)
        {
            for(j = i + 1; j < n; j++)
                if(sgn(b[j][i]) != 0) break;
            if(j == n)return 0;
            for(k = i; k < n; k++) swap(b[i][k],b[j][k]);
            sign++;
        }
        ret *= b[i][i];
        for(k = i + 1; k < n; k++) b[i][k]/=b[i][i];
        for(j = i+1; j < n; j++)
            for(k = i+1; k < n; k++) b[j][k] -= b[j][i]*b[i][k];
    }
    if(sign & 1)ret = -ret;
    return ret;
}
double a[MAXN][MAXN];
int g[MAXN][MAXN];
int main()
{
    int T;
    int n,m;
    int u,v;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d%d",&n,&m);
        memset(g,0,sizeof(g));
        while(m--)
        {
            scanf("%d%d",&u,&v);
            u--;v--;
            g[u][v] = g[v][u] = 1;
        }
        memset(a,0,sizeof(a));
        for(int i = 0; i < n; i++)
            for(int j = 0; j < n; j++)
                if(i != j && g[i][j])
                {
                    a[i][i]++;
                    a[i][j] = -1;
                }
        double ans = det(a,n-1);
        printf("%.0Lf\n",ans);
    }
    return 0;
}

```

## 9.2 C++11 random

```
void init(){
    std::random_device rd;
    std::default_random_engine gen( rd() );
    std::uniform_int_distribution<unsigned long long>
        dis(0,ULLONG_MAX);

    for (int i=0; i<MAXN; i++){
        h[i] = dis(gen);
    }
}
```

## 9.3 Digit Counting

```
int dfs(int pos, int state1, int state2 ....., bool
    limit, bool zero) {
    if ( pos == -1 ) return 是否符合條件;
    int &ret = dp[pos][state1][state2][....];
    if ( ret != -1 && !limit ) return ret;
    int ans = 0;
    int upper = limit ? digit[pos] : 9;
    for ( int i = 0 ; i <= upper ; i++ ) {
        ans += dfs(pos - 1, new_state1, new_state2,
            limit & ( i == upper), ( i == 0 ) && zero);
    }
    if ( !limit ) ret = ans;
    return ans;
}

int solve(int n) {
    int it = 0;
    for ( ; n ; n /= 10 ) digit[it++] = n % 10;
    return dfs(it - 1, 0, 0, 1, 1);
}
```

## 9.4 DP optimization

Monotonicity & 1D/1D DP & 2D/1D DP

Definition xD/yD  
 1D/1D  $DP[j] = \min(0 \leq i < j) \{ DP[i] + w(i, j) \}$ ;  $DP[0] = k$   
 2D/1D  $DP[i][j] = \min(i < k \leq j) \{ DP[i][k - 1] + DP[k][j] \}$   
 $+ w(i, j)$ ;  $DP[i][i] = 0$

Monotonicity

	c	d
a	w(a, c)	w(a, d)
b	w(b, c)	w(b, d)

Monge Condition

Concave (凹四邊形不等式):  $w(a, c) + w(b, d) \geq w(a, d) + w(b, c)$

Convex (凸四邊形不等式):  $w(a, c) + w(b, d) \leq w(a, d) + w(b, c)$

Totally Monotone

Concave (凹單調):  $w(a, c) \leq w(b, d) \rightarrow w(a, d) \leq w(b, c)$

Convex (凸單調):  $w(a, c) \geq w(b, d) \rightarrow w(a, d) \geq w(b, c)$

1D/1D DP  $O(n^2) \rightarrow O(n \lg n)$

\*\*CONSIDER THE TRANSITION POINT\*\*

Solve 1D/1D Concave by Stack

Solve 1D/1D Convex by Deque

2D/1D Convex DP (Totally Monotone)  $O(n^3) \rightarrow O(n^2)$

$h(i, j - 1) \leq h(i, j) \leq h(i + 1, j)$

## 9.5 DP 1D/1D

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

int t, n, L;
int p;
char s[MAXN][35];
ll sum[MAXN] = {0};
long double dp[MAXN] = {0};
int prevd[MAXN] = {0};

long double pw(long double a, int n) {
    if ( n == 1 ) return a;
    long double b = pw(a, n/2);
    if ( n & 1 ) return b*b*a;
    else return b*b;
}

long double f(int i, int j) {
    // cout << (sum[i] - sum[j]+i-j-1-L) << endl;
    return pw(abs(sum[i] - sum[j]+i-j-1-L), p) + dp[j];
}

struct INV {
    int L, R, pos;
};
INV stk[MAXN*10];
int top = 1, bot = 1;
void update(int i) {
    while ( top > bot && i < stk[top].L && f(stk[top].L,
        i) < f(stk[top].L, stk[top].pos) ) {
        stk[top - 1].R = stk[top].R;
        top--;
    }
    int lo = stk[top].L, hi = stk[top].R, mid, pos =
        stk[top].pos;
    //if ( i >= lo ) lo = i + 1;
    while ( lo != hi ) {
        mid = lo + (hi - lo) / 2;
        if ( f(mid, i) < f(mid, pos) ) hi = mid;
        else lo = mid + 1;
    }
    if ( hi < stk[top].R ) {
        stk[top + 1] = (INV) { hi, stk[top].R, i };
        stk[top++].R = hi;
    }
}

int main() {
    cin >> t;
    while ( t-- ) {
        cin >> n >> L >> p;
        dp[0] = sum[0] = 0;
        for ( int i = 1 ; i <= n ; i++ ) {
            cin >> s[i];
            sum[i] = sum[i-1] + strlen(s[i]);
            dp[i] = numeric_limits<long double>::max();
        }
        stk[top] = (INV) {1, n + 1, 0};
        for ( int i = 1 ; i <= n ; i++ ) {
            if ( i >= stk[bot].R ) bot++;
            dp[i] = f(i, stk[bot].pos);
            update(i);
            // cout << (ll) f(i, stk[bot].pos) << endl;
        }
        if ( dp[n] > 1e18 ) {
            cout << "Too hard to arrange" << endl;
        } else {
            vector<PI> as;
            cout << (ll)dp[n] << endl;
        }
    }
    return 0;
}
```

## 9.6 stable marriage

```
// normal stable marriage problem
// input:
//3
//Albert Laura Nancy Marcy
//Brad Marcy Nancy Laura
//Chuck Laura Marcy Nancy
//Laura Chuck Albert Brad
//Marcy Albert Chuck Brad
//Nancy Brad Albert Chuck

#include<bits/stdc++.h>
using namespace std;
const int MAXN = 505;

int n;
int favor[MAXN][MAXN]; // favor[boy_id][rank] = girl_id
;
int order[MAXN][MAXN]; // order[girl_id][boy_id] = rank
;
int current[MAXN]; // current[boy_id] = rank; boy_id
will pursue current[boy_id] girl.
int girl_current[MAXN]; // girl[girl_id] = boy_id;

void initialize() {
    for ( int i = 0 ; i < n ; i++ ) {
        current[i] = 0;
        girl_current[i] = n;
        order[i][n] = n;
    }
}

map<string, int> male, female;
string bname[MAXN], gname[MAXN];
int fit = 0;

void stable_marriage() {
    queue<int> que;
    for ( int i = 0 ; i < n ; i++ ) que.push(i);
    while ( !que.empty() ) {
        int boy_id = que.front();
        que.pop();

        int girl_id = favor[boy_id][current[boy_id]];
        current[boy_id] ++;

        if ( order[girl_id][boy_id] < order[girl_id][
            girl_current[girl_id]] ) {
            if ( girl_current[girl_id] < n ) que.push(
                girl_current[girl_id]); // if not the first
                time
            girl_current[girl_id] = boy_id;
        } else {
            que.push(boy_id);
        }
    }
}

int main() {
    cin >> n;

    for ( int i = 0 ; i < n ; i++ ) {
        string p, t;
        cin >> p;
        male[p] = i;
        bname[i] = p;
        for ( int j = 0 ; j < n ; j++ ) {
            cin >> t;
            if ( !female.count(t) ) {
                gname[fit] = t;
                female[t] = fit++;
            }
            favor[i][j] = female[t];
        }
    }
}
```

```
for ( int i = 0 ; i < n ; i++ ) {
    string p, t;
    cin >> p;
    for ( int j = 0 ; j < n ; j++ ) {
        cin >> t;
        order[female[p]][male[t]] = j;
    }
}

initialize();
stable_marriage();

for ( int i = 0 ; i < n ; i++ ) {
    cout << bname[i] << " " << gname[favor[i][current[i]
        ] - 1] << endl;
}
}
```

## 9.7 Mo's algorithm

```
int l = 0, r = 0, nowAns = 0, BLOCK_SIZE, n, m;
int ans[];
struct QUE{
    int l, r, id;
    friend bool operator < (QUE a, QUE b){
        if(a.l / BLOCK_SIZE != b.l / BLOCK_SIZE)
            return a.l / BLOCK_SIZE < b.l / BLOCK_SIZE;
        return a.r < b.r;
    }
}quers[];

inline void move(int pos, int sign) {
    // update nowAns
}

void solve() {
    BLOCK_SIZE = int(ceil(pow(n, 0.5)));
    sort(quers, quers + m);
    for (int i = 0; i < m; ++i) {
        const QUE &q = quers[i];
        while (l > q.l) move(--l, 1);
        while (r < q.r) move(r++, 1);
        while (l < q.l) move(l++, -1);
        while (r > q.r) move(--r, -1);
        ans[q.id] = nowAns;
    }
}
```

## 9.8 Parser

```
using LL = long long;
const int MAXLEVEL = 2;
// binary operators
const vector<char> Ops[MAXLEVEL] = {
    {'+', '-'}, // Level 0
    {'*', '/'} // Level 1
};
// unary operators
const vector<pair<char, int>> Op1s = {
    {'-', 0} // operator negative works on Level 0
};
struct Node{
    ~Node(){ delete L; delete R; }
    enum { op, op1, num } type;
    LL val;
    Node *L, *R;
} *root;
char getOp1(int LEVEL, istream& is){
    is >> ws;
    for (auto& x : Op1s){
        auto& op = x.first;
```

```

        auto& lev = x.second;
        if (LEVEL == lev && is.peek() == op)
            return is.get();
    }
    return 0;
}
template <int LEVEL> void parse(Node*& x, istream& is){
    char op1 = getOp1(LEVEL, is);
    parse<LEVEL+1>(x, is);
    if (op1) x = new Node{Node::op1, op1, x, nullptr};
    auto& ops = Ops[LEVEL];
    while (is>>ws && count(ops.begin(), ops.end(), is.
        peek())){
        x = new Node{Node::op, is.get(), x, nullptr};
        parse<LEVEL+1>(x->R, is);
    }
}
template <> void parse<MAXLEVEL>(Node*& x, istream& is)
{
    char op1 = getOp1(MAXLEVEL, is);
    is>>ws;
    if (is.peek()>='0' && is.peek()<='9'){
        LL t; is>>t;
        x = new Node{Node::num, t, nullptr, nullptr};
    } else if (is.peek() == '('){
        is.get();
        parse<0>(x, is);
        is>>ws;
        if (is.get() != ')') throw 0;
    } else throw 0;
    if (op1) x = new Node{Node::op1, op1, x, nullptr};
}
// throw when error occur !!!!!
void build(istream& is){
    parse<0>(root, is);
    if ((is>>ws).peek() != EOF) throw 0;
}

```

## 9.9 java cheat sheet

```

import java.util.*;
import java.math.*;
import java.io.*;

public class java{
    static class Comp implements Comparator<Integer>{
        public int compare(Integer lhs, Integer rhs){
            return lhs - rhs;
        }
    }
    static class Yee implements Comparable<Yee>{
        public int compareTo(Yee y){
            return 0;
        }
    }
    static class Reader{
        private BufferedReader br;
        private StringTokenizer st;
        public Reader(){
            br = new BufferedReader(new
                InputStreamReader(System.in));
        }
        boolean hasNext() throws IOException{
            String s;
            while (st == null || !st.hasMoreElements())
            {
                if ((s = br.readLine()) == null) return
                    false;
                st = new StringTokenizer(s);
            }
            return true;
        }
        String next() throws IOException{
            while (st == null || !st.hasMoreElements())
                st = new StringTokenizer(br.readLine());
        }
    }
}

```

```

        return st.nextToken();
    }
    int nextInt() throws IOException{
        return Integer.parseInt(next());
    }
    // Long.parseLong, Double.parseDouble, br.
    // readLine
}
public static void main(String args[])throws
    IOException{
    Reader cin = new Reader();
    //Scanner cin = new Scanner(System.in);
    PrintWriter cout = new PrintWriter(System.out);
    //Scanner cin = new Scanner(new File("t.in"));
    //PrintWriter cout = new PrintWriter(new File("
    t.out"));
    // ***** cout.close() or cout.flush() is needed
    // *****

    // 2D array: int[][] a = new int[10][10];
    // input, EOF, Graph
    int n = cin.nextInt();
    // nextFloat, nextLine, next
    ArrayList<ArrayList<Integer>> G = new ArrayList
    <>();
    for (int i=0; i<n; i++) G.add(new ArrayList<>()
    );
    while (cin.hasNext()){ // EOF
        int u = cin.nextInt(), v = cin.nextInt();
        G.get(u).add(v);
    }
    // Math: E, PI, min, max, random(double 0~1),
    // sin...
    // Collections(List a): swap(a,i,j), sort(a[,
    // comp]), min(a), binarySearch(a,val[,comp])

    // set
    Set<Integer> set = new TreeSet<>();
    set.add(87); set.remove(87);
    if (!set.contains(87)) cout.println("no 87");

    // map
    Map<String, Integer> map = new HashMap<>();
    map.put("0", 1); map.put("2", 3);
    for ( Map.Entry<String,Integer> i : map.
        entrySet() )
        cout.println(i.getKey() + " " + i.getValue
        () + " wry");
    cout.println( map.get("1") );

    // Big Number: TEN ONE ZERO, modInverse
    // isProbablePrime modInverse modPow
    // add subtract multiply divide remainder, and
    // or xor not shiftLeft shiftRight

    // queue: add, peek(==null), poll
    PriorityQueue<Integer> pq = new PriorityQueue<
    Integer>(Collections.reverseOrder());
    Queue<Integer> q = new ArrayDeque<Integer>();

    // stack: push, empty, pop
    Stack<Integer> s = new Stack<Integer>();

    cout.close();
}
}

```

## 9.10 python cheat sheet

```

#!/usr/bin/env python3

# import
import math
from math import *
import math as M
from math import sqrt

```



```

# input
n = int( input() )
a = [ int(x) for x in input().split() ]

# EOF
while True:
    try:
        solve()
    except:
        break;

# output
print( x, sep=' ' )
print( ''.join( str(x)+' ' for x in a ) )
print( '{:5d}'.format(x) )

# sort
a.sort()
sorted(a)

# list
a = [ x for x in range(n) ]
a.append(x)

# Basic operator
a, b = 10, 20
a/b # 0.5
a//b # 0
a%b # 10
a**b # 10^20

# if, else if, else
if a==0:
    print('zero')
elif a>0:
    print('positive')
else:
    print('negative')

# loop
while a==b and b==c:
    for i in LIST:

# stack # C++
stack = [3,4,5]
stack.append(6) # push()
stack.pop() # pop()
stack[-1] # top()
len(stack) # size() 0(1)

# queue # C++
from collections import deque
queue = deque([3,4,5])
queue.append(6) # push()
queue.popleft() # pop()
queue[0] # front()
len(queue) # size() 0(1)

# random
from random import *
randrange(L,R,step) # [L,R) L+k*step
randint(L,R) # int from [L,R]
choice(list) # pick 1 item from list
choices(list,k) # pick k item
shuffle(list)
Uniform(L,R) # float from [L,R]

# Decimal
from fractions import Fraction
from decimal import Decimal, getcontext
getcontext().prec = 250 # set precision

itwo = Decimal(0.5)
two = Decimal(2)

N = 200
def angle(cosT):

```

```

"""given cos(theta) in decimal return theta"""
for i in range(N):
    cosT = ((cosT + 1) / two) ** itwo
    sinT = (1 - cosT * cosT) ** itwo
    return sinT * (2 ** N)
pi = angle(Decimal(-1))

# file IO
r = open("filename.in")
a = r.read() # read whole content into one string

w = open("filename.out", "w")
w.write('123\n')

# IO redirection
import sys
sys.stdin = open('filename.in')
sys.stdout = open('filename.out', 'w')

```

