

# Contents

|                                        |           |                                                                 |           |
|----------------------------------------|-----------|-----------------------------------------------------------------|-----------|
| <b>1 Basic</b>                         | <b>1</b>  | <b>7 Data Structure</b>                                         | <b>20</b> |
| 1.1 Vimrc                              | 1         | 7.1 Binary Index Tree                                           | 20        |
| 1.2 Default Code                       | 1         | 7.2 Segment Tree                                                | 20        |
| 1.3 Stringstream                       | 1         | 7.3 Treap                                                       | 20        |
| 1.4 Check                              | 1         | 7.4 Link-Cut Tree                                               | 21        |
| 1.5 python-related                     | 1         | 7.5 Disjoint Set                                                | 21        |
| 1.6 Compare                            | 2         | 7.6 Trie                                                        | 22        |
|                                        |           | 7.7 Persistent Segment Tree                                     | 22        |
|                                        |           | 7.8 SQRT-Decomposition                                          | 22        |
| <b>2 Flow</b>                          | <b>2</b>  | <b>8 DP</b>                                                     | <b>23</b> |
| 2.1 ISAP                               | 2         | 8.1 SOS DP                                                      | 23        |
| 2.2 MinCostFlow                        | 2         | 8.2 Subset Sum                                                  | 23        |
| 2.3 Dinic                              | 2         | 8.3 knapsack                                                    | 23        |
| 2.4 Kuhn Munkres 最大完美二分匹配              | 3         | 8.4 Knapsack Unlimited                                          | 23        |
| 2.5 Directed MST                       | 3         | 8.5 Knapsack Limited                                            | 23        |
| 2.6 SW min-cut (不限 S-T 的 min-cut)      | 3         | 8.6 DP on DAG                                                   | 23        |
| 2.7 Max flow with lower/upper bound    | 4         | 8.7 LIS                                                         | 23        |
| 2.8 Flow Conclusion                    | 4         | 8.8 Matrix Fast Power                                           | 23        |
| <b>3 Math</b>                          | <b>4</b>  | <b>9 Others</b>                                                 | <b>23</b> |
| 3.1 Fast power                         | 4         | 9.1 Mo's Algorithm                                              | 23        |
| 3.2 FFT                                | 4         | 9.2 Reverse Pair                                                | 23        |
| 3.3 Poly operator                      | 4         |                                                                 |           |
| 3.4 O(1)mul                            | 5         | <b>1 Basic</b>                                                  |           |
| 3.5 Miller Rabin                       | 5         | <b>1.1 Vimrc</b>                                                |           |
| 3.6 Faulhaber ( $\sum_{i=1}^n i^p$ )   | 5         | set nu                                                          |           |
| 3.7 Chinese Remainder                  | 6         | set cindent                                                     |           |
| 3.8 Pollard Rho                        | 6         | set tabstop=4                                                   |           |
| 3.9 Josephus Problem                   | 6         | set shiftwidth=4                                                |           |
| 3.10 Gaussian Elimination              | 6         | set softtabstop=4                                               |           |
| 3.11 ax+by=gcd                         | 6         | set expandtab                                                   |           |
| 3.12 Roots of Polynomial 找多項式的根        | 6         | set autowrite                                                   |           |
| 3.13 Primes                            | 7         | set mouse=a                                                     |           |
| 3.14 Result                            | 7         | autocmd filetype cpp noremap <F5> :w<CR> :!g++ % && ./a.out<CR> |           |
| <b>4 Geometry</b>                      | <b>7</b>  | <b>1.2 Default Code</b>                                         |           |
| 4.1 definition                         | 7         | #include <bits/stdc++.h>                                        |           |
| 4.2 Intersection of 2 lines            | 8         | using namespace std;                                            |           |
| 4.3 halfPlaneIntersection              | 8         | #define ld long double                                          |           |
| 4.4 Convex Hull                        | 8         | #define ll long long                                            |           |
| 4.5 Intersection of 2 segments         | 8         | #define pb push_back                                            |           |
| 4.6 Intersection of circle and segment | 8         | #define endl '\n'                                               |           |
| 4.7 Circle cover                       | 9         | #define all(x) x.begin(), x.end()                               |           |
| 4.8 Convex Hull trick                  | 9         | #define IO ios::sync_with_stdio(0);cin.tie(0)                   |           |
| 4.9 Tangent line of two circles        | 9         |                                                                 |           |
| 4.10 KD Tree                           | 9         | <b>1.3 Stringstream</b>                                         |           |
| 4.11 Lower Concave Hull                | 10        | //stringstream                                                  |           |
| 4.12 Min Enclosing Circle              | 10        | string in, out, tmp;                                            |           |
| 4.13 Min Enclosing Ball                | 10        | cin >> in;                                                      |           |
| 4.14 Minkowski Sum                     | 11        | cin.ignore();                                                   |           |
| 4.15 Min dist on Cuboid                | 11        | stringstream ss(in);                                            |           |
| 4.16 Heart of Triangle                 | 11        | while(getline(ss, tmp, {char})) out += tmp;                     |           |
| 4.17 Closest Pair                      | 11        | //while(ss >> tmp) out += tmp;                                  |           |
| <b>5 Graph</b>                         | <b>12</b> | <b>1.4 Check</b>                                                |           |
| 5.1 Maximum Clique 最大團                 | 12        | for ((i=0;;i++))                                                |           |
| 5.2 Maximal Clique 極大團                 | 12        | do                                                              |           |
| 5.3 Strongly Connected Component       | 12        | echo "\$i"                                                      |           |
| 5.4 Dynamic MST                        | 12        | python3 gen.py > input                                          |           |
| 5.5 Maximum General graph Matching     | 13        | ./ac < input > ac.out                                           |           |
| 5.6 Minimum General Weighted Matching  | 13        | ./wa < input > wa.out                                           |           |
| 5.7 Minimum Steiner Tree               | 14        | diff ac.out wa.out    break                                     |           |
| 5.8 BCC based on vertex                | 14        | done                                                            |           |
| 5.9 Min Mean Cycle                     | 14        | <b>1.5 python-related</b>                                       |           |
| 5.10 Directed Graph Min Cost Cycle     | 15        | parser:                                                         |           |
| 5.11 K-th Shortest Path                | 15        | int(eval(num.replace("/", "/"))) )                              |           |
| 5.12 SPFA                              | 16        |                                                                 |           |
| 5.13 Kruskal                           | 16        | from fractions import Fraction                                  |           |
| 5.14 Dijkstra                          | 16        | from decimal import Decimal, getcontext                         |           |
| 5.15 LCA                               | 17        | getcontext().prec = 250 # set precision                         |           |
| 5.16 Bellman                           | 17        | itwo = Decimal(0.5)                                             |           |
| 5.17 差分約束                              | 17        | two = Decimal(2)                                                |           |
| 5.18 Euler Path                        | 17        | format(x, '0.10f') # set precision                              |           |
| 5.19 Tree Flatten                      | 17        | N = 200                                                         |           |
| <b>6 String</b>                        | <b>17</b> |                                                                 |           |
| 6.1 PalTree                            | 17        |                                                                 |           |
| 6.2 KMP                                | 18        |                                                                 |           |
| 6.3 Suffix Array                       | 18        |                                                                 |           |
| 6.4 Suffix Automata                    | 18        |                                                                 |           |
| 6.5 Aho-Corasick                       | 19        |                                                                 |           |
| 6.6 Z Value                            | 19        |                                                                 |           |
| 6.7 Z Value Palindrome                 | 19        |                                                                 |           |
| 6.8 Smallest Rotation                  | 19        |                                                                 |           |
| 6.9 Cyclic LCS                         | 19        |                                                                 |           |
| 6.10 Rolling Hash                      | 20        |                                                                 |           |

```
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))
```

## 1.6 Compare

```
struct Example{
    int x;
    friend bool operator>(const Example &a, const Example
        &b){
        return a.x > b.x;
    }
};
set<Example> s;// map<Example, int> mp;
struct cmp{
    bool operator()(const int &a,const int &b){return a
        >b;}
};
set<int, cmp> s;//map<int, int, cmp> mp;
```

## 2 Flow

### 2.1 ISAP

```
struct Maxflow {
    static const int MAXV = 20010;
    static const int INF = 1000000;
    struct Edge {
        int v, c, r;
        Edge(int _v, int _c, int _r):
            v(_v), c(_c), r(_r) {}
    };
    int s, t;
    vector<Edge> G[MAXV*2];
    int iter[MAXV*2], d[MAXV*2], gap[MAXV*2], tot;
    void init(int x) {
        tot = x+2;
        s = x+1, t = x+2;
        for(int i = 0; i <= tot; i++) {
            G[i].clear();
            iter[i] = d[i] = gap[i] = 0;
        }
    }
    void addEdge(int u, int v, int c) {
        G[u].push_back(Edge(v, c, SZ(G[v])));
        G[v].push_back(Edge(u, 0, SZ(G[u]) - 1));
    }
    int dfs(int p, int flow) {
        if(p == t) return flow;
        for(int &i = iter[p]; i < SZ(G[p]); i++) {
            Edge &e = G[p][i];
            if(e.c > 0 && d[p] == d[e.v]+1) {
                int f = dfs(e.v, min(flow, e.c));
                if(f) {
                    e.c -= f;
                    G[e.v][e.r].c += f;
                    return f;
                }
            }
        }
        if(--gap[d[p]] == 0) d[s] = tot;
        else {
            d[p]++;
            iter[p] = 0;
            ++gap[d[p]];
        }
        return 0;
    }
    int solve() {
        int res = 0;
        gap[0] = tot;
        for(res = 0; d[s] < tot; res += dfs(s, INF));
        return res;
    }
    void reset() {
        for(int i=0;i<=tot;i++) {
            iter[i]=d[i]=gap[i]=0;
        }
    }
};
```

### 2.2 MinCostFlow

```
struct MinCostMaxFlow{
    typedef int Tcost;
    static const int MAXV = 20010;
    static const int INFf = 1000000;
    static const Tcost INFc = 1e9;
    struct Edge{
        int v, cap;
        Tcost w;
        int rev;
        Edge(){}
        Edge(int t2, int t3, Tcost t4, int t5)
            : v(t2), cap(t3), w(t4), rev(t5) {}
    };
    int V, s, t;
    vector<Edge> g[MAXV];
    void init(int n, int _s, int _t){
        V = n; s = _s; t = _t;
        for(int i = 0; i <= V; i++) g[i].clear();
    }
    void addEdge(int a, int b, int cap, Tcost w){
        g[a].push_back(Edge(b, cap, w, (int)g[b].size()));
        g[b].push_back(Edge(a, 0, -w, (int)g[a].size()-1));
    }
    Tcost d[MAXV];
    int id[MAXV], mom[MAXV];
    bool inqu[MAXV];
    queue<int> q;
    pair<int,Tcost> solve(){
        int mxf = 0; Tcost mnc = 0;
        while(1){
            fill(d, d+1+V, INFc);
            fill(inqu, inqu+1+V, 0);
            fill(mom, mom+1+V, -1);
            mom[s] = s;
            d[s] = 0;
            q.push(s); inqu[s] = 1;
            while(q.size()){
                int u = q.front(); q.pop();
                inqu[u] = 0;
                for(int i = 0; i < (int) g[u].size(); i++){
                    Edge &e = g[u][i];
                    int v = e.v;
                    if(e.cap > 0 && d[v] > d[u]+e.w){
                        d[v] = d[u]+e.w;
                        mom[v] = u;
                        id[v] = i;
                        if(!inqu[v]) q.push(v), inqu[v] = 1;
                    }
                }
                if(mom[t] == -1) break;
                int df = INFf;
                for(int u = t; u != s; u = mom[u])
                    df = min(df, g[mom[u]][id[u]].cap);
                for(int u = t; u != s; u = mom[u]){
                    Edge &e = g[mom[u]][id[u]];
                    e.cap -= df;
                    g[e.v][e.rev].cap += df;
                }
                mxf += df;
                mnc += df*d[t];
            }
            return {mxf,mnc};
        }
    }
};
```

### 2.3 Dinic

```
const int MXN = 10000;
struct Dinic{
    struct Edge{ int v,f,re; };
    int n,s,t,level[MXN];
    vector<Edge> E[MXN];
    void init(int _n, int _s, int _t){
        n = _n; s = _s; t = _t;
        for (int i=0; i<n; i++) E[i].clear();
    }
    void add_edge(int u, int v, int f){
        E[u].PB({v,f,SZ(E[v])});
        E[v].PB({u,0,SZ(E[u])-1});
    }
    bool BFS(){
        for (int i=0; i<n; i++) level[i] = -1;
        queue<int> que;
        que.push(s);
```

```

level[s] = 0;
while (!que.empty()){
    int u = que.front(); que.pop();
    for (auto it : E[u]){
        if (it.f > 0 && level[it.v] == -1){
            level[it.v] = level[u]+1;
            que.push(it.v);
        } }
    return level[t] != -1;
}
int DFS(int u, int nf){
    if (u == t) return nf;
    int res = 0;
    for (auto &it : E[u]){
        if (it.f > 0 && level[it.v] == level[u]+1){
            int tf = DFS(it.v, min(nf,it.f));
            res += tf; nf -= tf; it.f -= tf;
            E[it.v][it.re].f += tf;
            if (nf == 0) return res;
        } }
    if (!res) level[u] = -1;
    return res;
}
int flow(int res=0){
    while ( BFS() )
        res += DFS(s,2147483647);
    return res;
} }flow;

```

## 2.4 Kuhn Munkres 最大完美二分配

```

struct KM{ // max weight, for min negate the weights
    static const int MXN = 2001; // 1-based
    static const ll INF = 0x3f3f3f3f;
    int n, mx[MXN], my[MXN], pa[MXN];
    ll g[MXN][MXN], lx[MXN], ly[MXN], sy[MXN];
    bool vx[MXN], vy[MXN];
    void init(int _n) {
        n = _n;
        for(int i=1; i<=n; i++) fill(g[i], g[i]+n+1, 0);
    }
    void addEdge(int x, int y, ll w) {g[x][y] = w;}
    void augment(int y) {
        for(int x, z; y; y = z)
            x=pa[y], z=mx[x], my[y]=x, mx[x]=y;
    }
    void bfs(int st) {
        for(int i=1; i<=n; ++i) sy[i]=INF, vx[i]=vy[i]=0;
        queue<int> q; q.push(st);
        for(;;) {
            while(q.size()) {
                int x=q.front(); q.pop(); vx[x]=1;
                for(int y=1; y<=n; ++y) if(!vy[y]){
                    ll t = lx[x]+ly[y]-g[x][y];
                    if(t==0){
                        pa[y]=x;
                        if(!my[y]){augment(y);return;}
                        vy[y]=1, q.push(my[y]);
                    }else if(sy[y]>t) pa[y]=x,sy[y]=t;
                }
            }
            ll cut = INF;
            for(int y=1; y<=n; ++y)
                if(!vy[y]&&cut>sy[y]) cut=sy[y];
            for(int j=1; j<=n; ++j){
                if(vx[j]) lx[j] -= cut;
                if(vy[j]) ly[j] += cut;
                else sy[j] -= cut;
            }
            for(int y=1; y<=n; ++y) if(!vy[y]&&sy[y]==0){
                if(!my[y]){augment(y);return;}
                vy[y]=1, q.push(my[y]);
            }
        }
    }
    ll solve(){
        fill(mx, mx+n+1, 0); fill(my, my+n+1, 0);
        fill(ly, ly+n+1, 0); fill(lx, lx+n+1, -INF);
        for(int x=1; x<=n; ++x) for(int y=1; y<=n; ++y)
            lx[x] = max(lx[x], g[x][y]);
        for(int x=1; x<=n; ++x) bfs(x);
        ll ans = 0;
        for(int y=1; y<=n; ++y) ans += g[my[y]][y];
        return ans;
    } }graph;

```

## 2.5 Directed MST

```

/* Edmond's algoirthm for Directed MST
 * runs in O(VE) */
const int MAXV = 10010;
const int MAXE = 10010;
const int INF = 2147483647;
struct Edge{
    int u, v, c;
    Edge(int x=0, int y=0, int z=0) : u(x), v(y), c(z){}
};
int V, E, root;
Edge edges[MAXE];
inline int newV(){ return ++ V; }
inline void addEdge(int u, int v, int c)
{ edges[++E] = Edge(u, v, c); }
bool con[MAXV];
int mnInW[MAXV], prv[MAXV], cyc[MAXV], vis[MAXV];
inline int DMST(){
    fill(con, con+V+1, 0);
    int r1 = 0, r2 = 0;
    while(1){
        fill(mnInW, mnInW+V+1, INF);
        fill(prv, prv+V+1, -1);
        REP(i, 1, E){
            int u=edges[i].u, v=edges[i].v, c=edges[i].c;
            if(u != v && v != root && c < mnInW[v])
                mnInW[v] = c, prv[v] = u;
        }
        fill(vis, vis+V+1, -1);
        fill(cyc, cyc+V+1, -1);
        r1 = 0;
        bool jf = 0;
        REP(i, 1, V){
            if(con[i]) continue;
            if(prv[i] == -1 && i != root) return -1;
            if(prv[i] > 0) r1 += mnInW[i];
            int s;
            for(s = i; s != -1 && vis[s] == -1; s = prv[s])
                vis[s] = i;
            if(s > 0 && vis[s] == i){
                // get a cycle
                jf = 1; int v = s;
                do{
                    cyc[v] = s, con[v] = 1;
                    r2 += mnInW[v]; v = prv[v];
                }while(v != s);
                con[s] = 0;
            }
        }
        if(!jf) break;
        REP(i, 1, E){
            int &u = edges[i].u;
            int &v = edges[i].v;
            if(cyc[v] > 0) edges[i].c -= mnInW[edges[i].v];
            if(cyc[u] > 0) edges[i].u = cyc[edges[i].u];
            if(cyc[v] > 0) edges[i].v = cyc[edges[i].v];
            if(u == v) edges[i--] = edges[E--];
        }
        return r1+r2;
    }
}

```

## 2.6 SW min-cut (不限 S-T 的 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;

```

## 2.7 Max flow with lower/upper bound

```

// flow use ISAP
// Max flow with lower/upper bound on edges
// source = 1, sink = n
int in[ N ], out[ N ];
int l[ M ], r[ M ], a[ M ], b[ M ]; // 0-base, a 下界, b 上界
int solve(){
    flow.init( n ); // n為點的數量, m為邊的數量, 點是1-base
    for( int i = 0 ; i < m ; i ++ ){
        in[ r[ i ] ] += a[ i ];
        out[ l[ i ] ] += a[ i ];
        flow.addEdge( l[ i ], r[ i ], b[ i ] - a[ i ] );
        // flow from l[i] to r[i] must in [a[i], b[i]]
    }
    int nd = 0;
    for( int i = 1 ; i <= n ; i ++ ){
        if( in[ i ] < out[ i ] ){
            flow.addEdge( i, flow.t, out[ i ] - in[ i ] );
            nd += out[ i ] - in[ i ];
        }
        if( out[ i ] < in[ i ] )
            flow.addEdge( flow.s, i, in[ i ] - out[ i ] );
    }
    // original sink to source
    flow.addEdge( n, 1, INF );
    if( flow.maxflow() != nd )
        // no solution
        return -1;
    int ans = flow.G[ 1 ].back().c; // source to sink
    flow.G[ 1 ].back().c = flow.G[ n ].back().c = 0;
    // take out super source and super sink
    for( size_t i = 0 ; i < flow.G[ flow.s ].size() ; i ++ ){
        flow.G[ flow.s ][ i ].c = 0;
        Edge &e = flow.G[ flow.s ][ i ];
        flow.G[ e.v ][ e.r ].c = 0;
    }
    for( size_t i = 0 ; i < flow.G[ flow.t ].size() ; i ++ ){
        flow.G[ flow.t ][ i ].c = 0;
        Edge &e = flow.G[ flow.t ][ i ];
        flow.G[ e.v ][ e.r ].c = 0;
    }
    flow.addEdge( flow.s, 1, INF );
    flow.addEdge( n, flow.t, INF );
    flow.reset();
    return ans + flow.maxflow();
}

```

## 2.8 Flow Conclusion

二分圖的最小點覆蓋數=該二分圖的最大匹配數  
 二分圖的最小邊覆蓋數=圖中的頂點數- (最小點覆蓋數) 該二分圖的最大匹配數  
 DAG圖的最短路徑覆蓋可以轉化為二分圖的人後求解  
 二分圖的最大點獨立數=點的個數-最小點覆蓋數 (最大匹配)  
 最大匹配: 超級原點連一團, 超級終點連另一團, 流量即是最大匹配

## 3 Math

### 3.1 Fast power

```

int power(int a, int b){
    int ans=1, base=a;
    while(b!=0){
        if(b&1){ // 如果是奇數
            ans*=base; // 向結果賦值
        }
        base*=base; // 基礎相乘
        b=b/2;
    }
    return ans;
}

```

### 3.2 FFT

```

// const int MAXN = 262144;
// (must be 2^k)
// before any usage, run pre_fft() first
typedef long double ld;
typedef complex<ld> cplx; // real(), imag()
const ld PI = acos(-1);
const cplx I(0, 1);
cplx omega[MAXN+1];
void pre_fft(){
    for(int i=0; i<=MAXN; i++)
        omega[i] = exp(i * 2 * PI / MAXN * I);
}
// n must be 2^k
void fft(int n, cplx a[], bool inv=false){
    int basic = MAXN / n;
    int theta = basic;
    for (int m = n; m >= 2; m >= 1) {
        int mh = m >> 1;
        for (int i = 0; i < mh; i++) {
            cplx w = omega[inv ? MAXN - (i*theta%MAXN) : i*theta%MAXN];
            for (int j = i; j < n; j += m) {
                int k = j + mh;
                cplx x = a[j] - a[k];
                a[j] += a[k];
                a[k] = w * x;
            }
            theta = (theta * 2) % MAXN;
        }
        int i = 0;
        for (int j = 1; j < n - 1; j++) {
            for (int k = n >> 1; k > (i ^= k); k >>= 1);
            if (j < i) swap(a[i], a[j]);
        }
        if(inv) for (i = 0; i < n; i++) a[i] /= n;
    }
    cplx arr[MAXN+1];
    inline void mul(int _n, ll a[], int _m, ll b[], ll ans[])
    {
        int n=1, sum=_n+_m-1;
        while(n<sum)
            n<<=1;
        for(int i=0; i<n; i++){
            double x=(i<_n?a[i]:0), y=(i<_m?b[i]:0);
            arr[i]=complex<double>(x+y, x-y);
        }
        fft(n, arr);
        for(int i=0; i<n; i++)
            arr[i]=arr[i]*arr[i];
        fft(n, arr, true);
        for(int i=0; i<sum; i++)
            ans[i]=(long long int)(arr[i].real()/4+0.5);
    }
}

```

### 3.3 Poly operator

```

struct PolyOp {
#define FOR(i, c) for (int i = 0; i < (c); ++i)
    NTT<P, root, MAXN> ntt;
    static int nxt2k(int x) {
        int i = 1; for (; i < x; i <= 1); return i;
    }
}

```

```

// c[i]=sum{j=0~i}a[j]*b[i-j] -> c[i+j]+=a[i]*b[j](加
分卷積)
// if c[i-j]+=a[i]*b[j] (減法卷積)
// (轉換成加法捲積) -> reverse(a); c=mul(a,b);
reverse(c);
void Mul(int n, LL a[], int m, LL b[], LL c[]) {
    static LL aa[MAXN], bb[MAXN];
    int N = nxt2k(n+m);
    copy(a, a+n, aa); fill(aa+n, aa+N, 0);
    copy(b, b+m, bb); fill(bb+m, bb+N, 0);
    ntt.tran(N, aa); ntt.tran(N, bb);
    FOR(i, N) c[i] = aa[i] * bb[i] % P;
    ntt.tran(N, c, 1);
}
void Inv(int n, LL a[], LL b[]) {
    // ab = aa^-1 = 1 mod x^(n/2)
    // (b - a^-1)^2 = 0 mod x^n
    // bb - a^-2 + 2 ba^-1 = 0
    // bba - a^-1 + 2b = 0
    // bba + 2b = a^-1
    static LL tmp[MAXN];
    if (n == 1) {b[0] = ntt.inv(a[0], P); return;}
    Inv((n+1)/2, a, b);
    int N = nxt2k(n*2);
    copy(a, a+n, tmp);
    fill(tmp+n, tmp+N, 0);
    fill(b+n, b+N, 0);
    ntt.tran(N, tmp); ntt.tran(N, b);
    FOR(i, N) {
        LL t1 = (2 - b[i] * tmp[i]) % P;
        if (t1 < 0) t1 += P;
        b[i] = b[i] * t1 % P;
    }
    ntt.tran(N, b, 1);
    fill(b+n, b+N, 0);
}
void Div(int n, LL a[], int m, LL b[], LL d[], LL r
[]) {
    // Ra = Rb * Rd mod x^(n-m+1)
    // Rd = Ra * Rb^-1 mod
    static LL aa[MAXN], bb[MAXN], ta[MAXN], tb[MAXN];
    if (n < m) {copy(a, a+n, r); fill(r+n, r+m, 0);
        return;}
    // d: n-1 - (m-1) = n-m (n-m+1 terms)
    copy(a, a+n, aa); copy(b, b+m, bb);
    reverse(aa, aa+n); reverse(bb, bb+m);
    Inv(n-m+1, bb, tb);
    Mul(n-m+1, ta, n-m+1, tb, d);
    fill(d+n-m+1, d+n, 0); reverse(d, d+n-m+1);
    // r: m-1 - 1 = m-2 (m-1 terms)
    Mul(m, b, n-m+1, d, ta);
    FOR(i, n) { r[i] = a[i] - ta[i]; if (r[i] < 0) r[i]
        += P; }
}
void dx(int n, LL a[], LL b[]) { REP(i, 1, n-1) b[i
-1] = i * a[i] % P; }
void Sx(int n, LL a[], LL b[]) {
    b[0] = 0;
    FOR(i, n) b[i+1] = a[i] * ntt.inv(i+1, P) % P;
}
void Ln(int n, LL a[], LL b[]) {
    // Integral a' a^-1 dx
    static LL a1[MAXN], a2[MAXN], b1[MAXN];
    int N = nxt2k(n*2);
    dx(n, a, a1); Inv(n, a, a2);
    Mul(n-1, a1, n, a2, b1);
    Sx(n+n-1-1, b1, b);
    fill(b+n, b+N, 0);
}
void Exp(int n, LL a[], LL b[]) {
    // Newton method to solve g(a(x)) = Ln b(x) - a(x)
    // = 0
    // b' = b - g(b(x)) / g'(b(x))
    // b' = b (1 - lnb + a)
    static LL lnb[MAXN], c[MAXN], tmp[MAXN];
    assert(a[0] == 0); // dont know exp(a[0]) mod P
    if (n == 1) {b[0] = 1; return;}
    Exp((n+1)/2, a, b);
    fill(b+(n+1)/2, b+n, 0);
    Ln(n, b, lnb);
    fill(c, c+n, 0); c[0] = 1;
}

```

```

FOR(i, n) {
    c[i] += a[i] - lnb[i];
    if (c[i] < 0) c[i] += P;
    if (c[i] >= P) c[i] -= P;
}
Mul(n, b, n, c, tmp);
copy(tmp, tmp+n, b);
}
} polyop;

```

### 3.4 O(1)mul

```

LL mul(LL x, LL y, LL mod){
    LL ret=x*y-(LL)((long double)x/mod*y)*mod;
    // LL ret=x*y-(LL)((long double)x*y/mod+0.5)*mod;
    return ret<0?ret+mod:ret;
}

```

### 3.5 Miller Rabin

```

// n < 4,759,123,141      3 : 2, 7, 61
// n < 1,122,004,669,633  4 : 2, 13, 23, 1662803
// n < 3,474,749,660,383  6 : pirmses <= 13
// n < 2^64              7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
// Make sure testing integer is in range [2, n-2] if
// you want to use magic.
LL magic[]={}
bool witness(LL a, LL n, LL u, int t){
    if(!a) return 0;
    LL x=myspow(a,u,n);
    for(int i=0;i<t;i++){
        LL nx=mul(x,x,n);
        if(nx==1&&x!=1&&x!=n-1) return 1;
        x=nx;
    }
    return x!=1;
}
bool miller_rabin(LL n) {
    int s=(magic number size)
    // iterate s times of witness on n
    if(n<2) return 0;
    if(!(n&1)) return n == 2;
    ll u=n-1; int t=0;
    // n-1 = u*2^t
    while(!(u&1)) u>>=1, t++;
    while(s--){
        LL a=magic[s]%n;
        if(witness(a,n,u,t)) return 0;
    }
    return 1;
}

```

### 3.6 Faulhaber ( $\sum_{i=1}^n i^p$ )

```

/* faulhaber' s formula -
* cal power sum formula of all p=1~k in O(k^2) */
#define MAXK 2500
const int mod = 1000000007;
int b[MAXK]; // bernoulli number
int inv[MAXK+1]; // inverse
int cm[MAXK+1][MAXK+1]; // combinactories
int co[MAXK][MAXK+2]; // coeeficient of x^j when p=i
inline int getinv(int x) {
    int a=x, b=mod, a0=1, a1=0, b0=0, b1=1;
    while(b) {
        int q,t;
        q=a/b; t=b; b=a-b*q; a=t;
        t=b0; b0=a0-b0*q; a0=t;
        t=b1; b1=a1-b1*q; a1=t;
    }
    return a0<0?a0+mod:a0;
}
inline void pre() {
    /* combinational */
    for(int i=0;i<=MAXK;i++){
        cm[i][0]=cm[i][i]=1;
        for(int j=1;j<i;j++){
            cm[i][j]=add(cm[i-1][j-1],cm[i-1][j]);
        }
    }
    /* inverse */
}

```

```

for(int i=1;i<=MAXK;i++) inv[i]=getinv(i);
/* bernoulli */
b[0]=1; b[1]=getinv(2); // with b[1] = 1/2
for(int i=2;i<MAXK;i++) {
    if(i&1) { b[i]=0; continue; }
    b[i]=1;
    for(int j=0;j<i;j++)
        b[i]=sub(b[i],
            mul(cm[i][j],mul(b[j], inv[i-j+1])));
}
/* faulhaber */
// sigma_x=1~n {x^p} =
// 1/(p+1) * sigma_j=0~p {C(p+1,j)*Bj*n^(p-j+1)}
for(int i=1;i<MAXK;i++) {
    co[i][0]=0;
    for(int j=0;j<=i;j++)
        co[i][i-j+1]=mul(inv[i+1], mul(cm[i+1][j], b[j]));
}
}
/* sample usage: return f(n,p) = sigma_x=1~n (x^p) */
inline int solve(int n,int p) {
    int sol=0,m=n;
    for(int i=1;i<=p+1;i++) {
        sol=add(sol,mul(co[p][i],m));
        m = mul(m, n);
    }
    return sol;
}

```

### 3.7 Chinese Remainder

```

LL x[N],m[N];
LL CRT(LL x1, LL m1, LL x2, LL m2) {
    LL g = __gcd(m1, m2);
    if((x2 - x1) % g) return -1; // no sol
    m1 /= g; m2 /= g;
    pair<LL,LL> p = gcd(m1, m2);
    LL lcm = m1 * m2 * g;
    LL res = p.first * (x2 - x1) * m1 + x1;
    return (res % lcm + lcm) % lcm;
}
LL solve(int n){ // n>=2, be careful with no solution
    LL res=CRT(x[0],m[0],x[1],m[1]),p=m[0]/__gcd(m[0],m[1])*m[1];
    for(int i=2;i<n;i++){
        res=CRT(res,p,x[i],m[i]);
        p=p/__gcd(p,m[i])*m[i];
    }
    return res;
}

```

### 3.8 Pollard Rho

```

// does not work when n is prime
LL f(LL x, LL mod){ return add(mul(x,x,mod),1,mod); }
LL pollard_rho(LL n) {
    if(!(n&1)) return 2;
    while(true){
        LL 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;
    }
}

```

### 3.9 Josephus Problem

```

int josephus(int n, int m){ //n人每m次
    int ans = 0;
    for (int i=1; i<=n; ++i)
        ans = (ans + m) % i;
    return ans;
}

```

### 3.10 Gaussian Elimination

```
const int GAUSS_MOD = 100000007LL;
```

```

struct GAUSS{
    int n;
    vector<vector<int>> v;
    int ppow(int a, int k){
        if(k == 0) return 1;
        if(k % 2 == 0) return ppow(a * a % GAUSS_MOD, k >> 1);
        if(k % 2 == 1) return ppow(a * a % GAUSS_MOD, k >> 1) * a % GAUSS_MOD;
    }
    vector<int> solve(){
        vector<int> ans(n);
        REP(now, 0, n){
            REP(i, now, n) if(v[now][now] == 0 && v[i][now] != 0)
                swap(v[i], v[now]); // det = -det;
            if(v[now][now] == 0) return ans;
            int inv = ppow(v[now][now], GAUSS_MOD - 2);
            REP(i, 0, n) if(i != now){
                int tmp = v[i][now] * inv % GAUSS_MOD;
                REP(j, now, n + 1) (v[i][j] += GAUSS_MOD - tmp * v[now][j] % GAUSS_MOD) %= GAUSS_MOD;
            }
            REP(i, 0, n) ans[i] = v[i][n + 1] * ppow(v[i][i], GAUSS_MOD - 2) % GAUSS_MOD;
            return ans;
        }
        // gs.v.clear(), gs.v.resize(n, vector<int>(n + 1, 0));
    } gs;
}

```

### 3.11 ax+by=gcd

```

PII gcd(int a, int b){
    if(b == 0) return {1, 0};
    PII q = gcd(b, a % b);
    return {q.second, q.first - q.second * (a / b)};
}

```

### 3.12 Roots of Polynomial 找多項式的根

```

const double eps = 1e-12;
const double inf = 1e+12;
double a[ 10 ], x[ 10 ]; // a[0..n](coef) must be filled
int n; // degree of polynomial must be filled
int sign( double x ){return (x < -eps)?(-1):(x>eps);}
double f(double a[], int n, double x){
    double tmp=1,sum=0;
    for(int i=0;i<=n;i++){
        sum=sum+a[i]*tmp; tmp=tmp*x; }
    return sum;
}
double binary(double l,double r,double a[],int n){
    int sl=sign(f(a,n,l)),sr=sign(f(a,n,r));
    if(sl==0) return l; if(sr==0) return r;
    if(sl*sr>0) return inf;
    while(r-l>eps){
        double mid=(l+r)/2;
        int ss=sign(f(a,n,mid));
        if(ss==0) return mid;
        if(ss*sl>0) l=mid; else r=mid;
    }
    return l;
}
void solve(int n,double a[],double x[],int &nx){
    if(n==1){ x[1]=-a[0]/a[1]; nx=1; return; }
    double da[10], dx[10]; int ndx;
    for(int i=n;i>=1;i--) da[i-1]=a[i]*i;
    solve(n-1,da,dx,ndx);
    nx=0;
    if(ndx==0){
        double tmp=binary(-inf,inf,a,n);
        if (tmp<inf) x[++nx]=tmp;
        return;
    }
    double tmp;
    tmp=binary(-inf,dx[1],a,n);
    if(tmp<inf) x[++nx]=tmp;
}

```



```

for(int i=1;i<=ndx-1;i++){
    tmp=binary(dx[i],dx[i+1],a,n);
    if(tmp<inf) x[+nx]=tmp;
}
tmp=binary(dx[ndx],inf,a,n);
if(tmp<inf) x[+nx]=tmp;
} // roots are stored in x[1..nx]

```

### 3.13 Primes

```

/* 12721, 13331, 14341, 75577, 123457, 222557, 556679
 * 999983, 1097774749, 1076767633, 100102021, 99997771
 * 1001010013, 1000512343, 987654361, 999991231
 * 999888733, 98789101, 987777733, 999991921, 1010101333
 * 1010102101, 1000000000039, 100000000000037
 * 2305843009213693951, 4611686018427387847
 * 9223372036854775783, 18446744073709551557 */
int mu[ N ], p_tbl[ N ];
vector<int> primes;
void sieve() {
    mu[ 1 ] = p_tbl[ 1 ] = 1;
    for( int i = 2 ; i < N ; i ++ ){
        if( !p_tbl[ i ] ){
            p_tbl[ i ] = i;
            primes.push_back( i );
            mu[ i ] = -1;
        }
        for( int p : primes ){
            int x = i * p;
            if( x >= M ) break;
            p_tbl[ x ] = p;
            mu[ x ] = -mu[ i ];
            if( i % p == 0 ){
                mu[ x ] = 0;
                break;
            }
        }
    }
}
vector<int> factor( int x ){
    vector<int> fac{ 1 };
    while( x > 1 ){
        int fn = SZ(fac), p = p_tbl[ x ], pos = 0;
        while( x % p == 0 ){
            x /= p;
            for( int i = 0 ; i < fn ; i ++ )
                fac.PB( fac[ pos ++ ] * p );
        }
    }
    return fac;
}

```

### 3.14 Result

- Lucas' Theorem :  
For  $n, m \in \mathbb{Z}^*$  and prime  $P$ ,  $C(m, n) \bmod P = \prod C(m_i, n_i)$  where  $m_i$  is the  $i$ -th digit of  $m$  in base  $P$ .
- Stirling approximation :  
$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n e^{\frac{1}{12n}}$$
- Stirling Numbers(permutation  $|P| = n$  with  $k$  cycles):  
 $S(n, k) = \text{coefficient of } x^k \text{ in } \prod_{i=0}^{n-1} (x + i)$
- Stirling Numbers(Partition  $n$  elements into  $k$  non-empty set):  
$$S(n, k) = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n$$
- Pick' s Theorem :  $A = i + b/2 - 1$   
其面積  $A$  和內部格點數目  $i$ 、邊上格點數目  $b$  的關係
- Catalan number :  $C_n = \binom{2n}{n} / (n+1)$   
$$C_n^{n+m} - C_{n+1}^{n+m} = (m+n)! \frac{n-m+1}{n+1} \quad \text{for } n \geq m$$
  
$$C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$$
  
$$C_0 = 1 \quad \text{and} \quad C_{n+1} = 2 \binom{2n+1}{n+2} C_n$$
  
$$C_0 = 1 \quad \text{and} \quad C_{n+1} = \sum_{i=0}^n C_i C_{n-i} \quad \text{for } n \geq 0$$
- Euler Characteristic:  
planar graph:  $V - E + F - C = 1$   
convex polyhedron:  $V - E + F = 2$   
 $V, E, F, C$ : number of vertices, edges, faces(regions), and components
- Kirchhoff's theorem :  
 $A_{ii} = \deg(i), A_{ij} = (i, j) \in E ? -1 : 0$ , Deleting any one row, one column, and cal the  $\det(A)$
- Polya' theorem ( $c$  為方法數  $\cdot m$  為總數):  
$$\left( \sum_{i=1}^m c^{gcd(i,m)} \right) / m$$

- 錯排公式: ( $n$  個人中  $\cdot$  每個人皆不再原來位置的組合數):  
$$dp[0] = 1; dp[1] = 0;$$
  
$$dp[i] = (i-1) * (dp[i-1] + dp[i-2]);$$
- Bell 數 (有  $n$  個人, 把他們拆組的方法總數) :  
$$B_0 = 1$$
  
$$B_n = \sum_{k=0}^n s(n, k) \quad (\text{second-stirling})$$
  
$$B_{n+1} = \sum_{k=0}^n \binom{n}{k} B_k$$
- Wilson's theorem :  
$$(p-1)! \equiv -1 \pmod{p}$$
- Fermat's little theorem :  
$$a^p \equiv a \pmod{p}$$
- Euler's totient function:  
$$A^{B^C} \bmod p = \text{pow}(A, \text{pow}(B, C, p-1)) \bmod p$$

## 4 Geometry

### 4.1 definition

```

typedef long double ld;
const ld eps = 1e-8;
int dcmp(ld x) {
    if(abs(x) < eps) return 0;
    else return x < 0 ? -1 : 1;
}
struct Pt {
    ld x, y;
    Pt(ld _x=0, ld _y=0):x(_x), y(_y) {}

    Pt operator+(const Pt &a) const {
        return Pt(x+a.x, y+a.y);
    }
    Pt operator-(const Pt &a) const {
        return Pt(x-a.x, y-a.y);
    }
    Pt operator*(const ld &a) const {
        return Pt(x*a, y*a);
    }
    Pt operator/(const ld &a) const {
        return Pt(x/a, y/a);
    }
    ld operator*(const Pt &a) const {
        return x*a.x + y*a.y;
    }
    ld operator^(const Pt &a) const {
        return x*a.y - y*a.x;
    }
    bool operator<(const Pt &a) const {
        return x < a.x || (x == a.x && y < a.y);
        //return dcmp(x-a.x) < 0 || (dcmp(x-a.x) == 0 &&
            dcmp(y-a.y) < 0);
    }
    bool operator==(const Pt &a) const {
        return dcmp(x-a.x) == 0 && dcmp(y-a.y) == 0;
    }
};

ld norm2(const Pt &a) {
    return a*a;
}
ld norm(const Pt &a) {
    return sqrt(norm2(a));
}
Pt perp(const Pt &a) {
    return Pt(-a.y, a.x);
}
Pt rotate(const Pt &a, ld ang) {
    return Pt(a.x*cos(ang)-a.y*sin(ang), a.x*sin(ang)+a.y*cos(ang));
}

struct Line {
    Pt s, e, v; // start, end, end-start
    ld ang;
    Line(Pt _s=Pt(0, 0), Pt _e=Pt(0, 0)):s(_s), e(_e) { v = e-s; ang = atan2(v.y, v.x); }

    bool operator<(const Line &L) const {
        return ang < L.ang;
    }
};

struct Circle {
    Pt o; ld r;

```

```
Circle(Pt _o=Pt(0, 0), ld _r=0):o(_o), r(_r) {}
};
```

## 4.2 Intersection of 2 lines

```
Pt LLIntersect(Line a, Line b) {
    Pt p1 = a.s, p2 = a.e, q1 = b.s, q2 = b.e;
    ld f1 = (p2-p1)^(q1-p1), f2 = (p2-p1)^(p1-q2), f;
    if(dcmp(f=f1+f2) == 0)
        return dcmp(f1)?Pt(NAN,NAN):Pt(INFINITY,INFINITY);
    return q1*(f2/f) + q2*(f1/f);
}
```

## 4.3 halfPlaneIntersection

```
// for point or line solution, change > to >=
bool onleft(Line L, Pt p) {
    return dcmp(L.v^(p-L.s)) > 0;
}
// assume that Lines intersect
vector<Pt> HPI(vector<Line>& L) {
    sort(L.begin(), L.end()); // sort by angle
    int n = L.size(), fir, las;
    Pt *p = new Pt[n];
    Line *q = new Line[n];
    q[fir=las=0] = L[0];
    for(int i = 1; i < n; i++) {
        while(fir < las && !onleft(L[i], p[las-1])) las--;
        while(fir < las && !onleft(L[i], p[fir])) fir++;
        q[++las] = L[i];
        if(dcmp(q[las].v^q[las-1].v) == 0) {
            las--;
            if(onleft(q[las], L[i].s)) q[las] = L[i];
        }
        if(fir < las) p[las-1] = LLIntersect(q[las-1], q[las]);
    }
    while(fir < las && !onleft(q[fir], p[las-1])) las--;
    if(las-fir <= 1) return {};
    p[las] = LLIntersect(q[las], q[fir]);
    int m = 0;
    vector<Pt> ans(las-fir+1);
    for(int i = fir; i <= las; i++) ans[m++] = p[i];
    return ans;
}
```

## 4.4 Convex Hull

```
double cross(Pt o, Pt a, Pt b){
    return (a-o) ^ (b-o);
}
vector<Pt> convex_hull(vector<Pt> pt){
    sort(pt.begin(), pt.end());
    int top=0;
    vector<Pt> stk(2*pt.size());
    for (int i=0; i<(int)pt.size(); i++){
        while (top >= 2 && cross(stk[top-2], stk[top-1], pt[i]) <= 0)
            top--;
        stk[top++] = pt[i];
    }
    for (int i=pt.size()-2, t=top+1; i>=0; i--){
        while (top >= t && cross(stk[top-2], stk[top-1], pt[i]) <= 0)
            top--;
        stk[top++] = pt[i];
    }
    stk.resize(top-1);
    return stk;
}
```

## 4.5 Intersection of 2 segments

```
int ori( const Pt& o, const Pt& a, const Pt& b ){
    LL ret = (a - o) ^ (b - o);
    return (ret > 0) - (ret < 0);
}
// p1 == p2 || q1 == q2 need to be handled
bool banana( const Pt& p1, const Pt& p2,
              const Pt& q1, const Pt& q2 ){
    if( ( ( p2 - p1 ) ^ ( q2 - q1 ) ) == 0 ){ // parallel
        if( ori( p1, p2, q1 ) ) return false;
```

```
        return ( ( p1 - q1 ) * ( p2 - q1 ) ) <= 0 ||
               ( ( p1 - q2 ) * ( p2 - q2 ) ) <= 0 ||
               ( ( q1 - p1 ) * ( q2 - p1 ) ) <= 0 ||
               ( ( q1 - p2 ) * ( q2 - p2 ) ) <= 0;
    }
    return (ori( p1, p2, q1 ) * ori( p1, p2, q2 ) <= 0) &&
           (ori( q1, q2, p1 ) * ori( q1, q2, p2 ) <= 0);
}
```

## 4.6 Intersection of circle and segment

```
bool Inter( const Pt& p1, const Pt& p2, Circle& cc ){
    Pt dp = p2 - p1;
    double a = dp * dp;
    double b = 2 * ( dp * ( p1 - cc.O ) );
    double c = cc.O * cc.O + p1 * p1 - 2 * ( cc.O * p1 )
              - cc.R * cc.R;
    double bb4ac = b * b - 4 * a * c;
    return !( fabs( a ) < eps or bb4ac < 0 );
}
```

## 4.7 Circle cover

```
#define N 1021
#define D long double
struct CircleCover{
    int C; Circ c[ N ]; //填入C(圓數量),c(圓陣列)
    bool g[ N ][ N ], overlap[ N ][ N ];
    // Area[i] : area covered by at Least i circles
    D Area[ N ];
    void init( int _C ){ C = _C; }
    bool CCinter( Circ& a, Circ& b, Pt& p1, Pt& p2 ){
        Pt o1 = a.O, o2 = b.O;
        D r1 = a.R, r2 = b.R;
        if( norm( o1 - o2 ) > r1 + r2 ) return {};
        if( norm( o1 - o2 ) < max(r1, r2) - min(r1, r2) )
            return {};
        D d2 = ( o1 - o2 ) * ( o1 - o2 );
        D d = sqrt(d2);
        if( d > r1 + r2 ) return false;
        Pt u=(o1+o2)*0.5 + (o1-o2)*((r2*r2-r1*r1)/(2*d2));
        D A=sqrt((r1+r2+d)*(r1-r2+d)*(r1+r2-d)*(-r1+r2+d));
        Pt v=Pt( o1.Y-o2.Y, -o1.X + o2.X ) * A / (2*d2);
        p1 = u + v; p2 = u - v;
        return true;
    }
    struct Teve {
        Pt p; D ang; int add;
        Teve() {}
        Teve(Pt _a, D _b, int _c):p(_a), ang(_b), add(_c){}
        bool operator<(const Teve &a)const {
            return ang < a.ang;
        }
    }eve[ N * 2 ];
    // strict: x = 0, otherwise x = -1
    bool disjuct( Circ& a, Circ &b, int x )
    {return sign( norm( a.O - b.O ) - a.R - b.R ) > x;}
    bool contain( Circ& a, Circ &b, int x )
    {return sign( a.R - b.R - norm( a.O - b.O ) ) > x;}
    bool contain(int i, int j){
        /* c[j] is non-strictly in c[i]. */
        return (sign(c[i].R - c[j].R) > 0 ||
                (sign(c[i].R - c[j].R) == 0 && i < j) ) &&
                contain(c[i], c[j], -1);
    }
    void solve(){
        for( int i = 0; i <= C + 1; i ++ )
            Area[ i ] = 0;
        for( int i = 0; i < C; i ++ )
            for( int j = 0; j < C; j ++ )
                overlap[i][j] = contain(i, j);
        for( int i = 0; i < C; i ++ )
            for( int j = 0; j < C; j ++ )
                g[i][j] = !(overlap[i][j] || overlap[j][i] ||
                           disjuct(c[i], c[j], -1));
        for( int i = 0; i < C; i ++ ){
            int E = 0, cnt = 1;
            for( int j = 0; j < C; j ++ )
                if( j != i && overlap[j][i] )
                    cnt ++;
            for( int j = 0; j < C; j ++ )
                if( i != j && g[i][j] ){
```



```
Pt aa, bb;
CCinter(c[i], c[j], aa, bb);
D A=atan2(aa.Y - c[i].O.Y, aa.X - c[i].O.X);
D B=atan2(bb.Y - c[i].O.Y, bb.X - c[i].O.X);
eve[E ++] = Teve(bb, B, 1);
eve[E ++] = Teve(aa, A, -1);
if(B > A) cnt ++;
}
if( E == 0 ) Area[ cnt ] += pi * c[i].R * c[i].R;
else{
    sort( eve , eve + E );
    eve[E] = eve[0];
    for( int j = 0 ; j < E ; j ++ ){
        cnt += eve[j].add;
        Area[cnt] += (eve[j].p ^ eve[j + 1].p) * 0.5;
        D theta = eve[j + 1].ang - eve[j].ang;
        if (theta < 0) theta += 2.0 * pi;
        Area[cnt] +=
            (theta - sin(theta)) * c[i].R*c[i].R * 0.5;
    }
}
```

## 4.8 Convex Hull trick

```

/* Given a convex hull, answer queries in  $O(\lg N)$ 
CH should not contain identical points, the area should
be  $> 0$ , min pair(x, y) should be listed first */
double det( const Pt& p1 , const Pt& p2 )
{ return p1.X * p2.Y - p1.Y * p2.X; }
struct Conv{
    int n;
    vector<Pt> a;
    vector<Pt> upper, lower;
    Conv(vector<Pt> _a) : a(_a){
        n = a.size();
        int ptr = 0;
        for(int i=1; i<n; ++i) if (a[ptr] < a[i]) ptr = i;
        for(int i=0; i<=ptr; ++i) lower.push_back(a[i]);
        for(int i=ptr; i<n; ++i) upper.push_back(a[i]);
        upper.push_back(a[0]);
    }
    int sign( LL x ){ // fixed when changed to double
        return x < 0 ? -1 : x > 0; }
    pair<LL,int> get_tang(vector<Pt> &conv, Pt vec){
        int l = 0, r = (int)conv.size() - 2;
        for( ; l + 1 < r; ){
            int mid = (l + r) / 2;
            if(sign(det(conv[mid+1]-conv[mid],vec))>0)r=mid;
            else l = mid;
        }
        return max(make_pair(det(vec, conv[r]), r),
            make_pair(det(vec, conv[0]), 0));
    }
    void upd_tang(const Pt &p, int id, int &i0, int &i1){
        if(det(a[i0] - p, a[id] - p) > 0) i0 = id;
        if(det(a[i1] - p, a[id] - p) < 0) i1 = id;
    }
    void bi_search(int l, int r, Pt p, int &i0, int &i1){
        if(l == r) return;
        upd_tang(p, l % n, i0, i1);
        int sl=sign(det(a[l % n] - p, a[(l + 1) % n] - p));
        for( ; l + 1 < r; ) {
            int mid = (l + r) / 2;
            int smid=sign(det(a[mid%n]-p, a[(mid+1)%n]-p));
            if (smid == sl) l = mid;
            else r = mid;
        }
        upd_tang(p, r % n, i0, i1);
    }
    int bi_search(Pt u, Pt v, int l, int r) {
        int sl = sign(det(v - u, a[l % n] - u));
        for( ; l + 1 < r; ) {
            int mid = (l + r) / 2;
            int smid = sign(det(v - u, a[mid % n] - u));
            if (smid == sl) l = mid;
            else r = mid;
        }
        return l % n;
    }
}
// 1. whether a given point is inside the CH
bool contain(Pt p) {
    if (p.X < lower[0].X || p.X > lower.back().X)
        return 0;

```

```

int id = lower_bound(lower.begin(), lower.end(), Pt
    (p.X, -INF)) - lower.begin();
if (lower[id].X == p.X) {
    if (lower[id].Y > p.Y) return 0;
} else if (det(lower[id-1]-p, lower[id]-p) < 0) return 0;
id = lower_bound(upper.begin(), upper.end(), Pt(p.X
    , INF), greater<Pt>()) - upper.begin();
if (upper[id].X == p.X) {
    if (upper[id].Y < p.Y) return 0;
} else if (det(upper[id-1]-p, upper[id]-p) < 0) return 0;
return 1;
}

// 2. Find 2 tang pts on CH of a given outside point
// return true with i0, i1 as index of tangent points
// return false if inside CH
bool get_tang(Pt p, int &i0, int &i1) {
    if (contain(p)) return false;
    i0 = i1 = 0;
    int id = lower_bound(lower.begin(), lower.end(), p)
        - lower.begin();
    bi_search(0, id, p, i0, i1);
    bi_search(id, (int)lower.size(), p, i0, i1);
    id = lower_bound(upper.begin(), upper.end(), p,
        greater<Pt>()) - upper.begin();
    bi_search((int)lower.size() - 1, (int)lower.size()
        - 1 + id, p, i0, i1);
    bi_search((int)lower.size() - 1 + id, (int)lower.
        size() - 1 + (int)upper.size(), p, i0, i1);
    return true;
}

// 3. Find tangent points of a given vector
// ret the idx of vertex has max cross value with vec
int get_tang(Pt vec){
    pair<LL, int> ret = get_tang(upper, vec);
    ret.second = (ret.second + (int)lower.size() - 1) % n;
    ret = max(ret, get_tang(lower, vec));
    return ret.second;
}

// 4. Find intersection point of a given line
// return 1 and intersection is on edge (i, next(i))
// return 0 if no strictly intersection
bool get_intersection(Pt u, Pt v, int &i0, int &i1){
    int p0 = get_tang(u - v), p1 = get_tang(v - u);
    if (sign(det(v-u, a[p0]-u)) * sign(det(v-u, a[p1]-u)) < 0){
        if (p0 > p1) swap(p0, p1);
        i0 = bi_search(u, v, p0, p1);
        i1 = bi_search(u, v, p1, p0 + n);
        return 1;
    }
    return 0;
}
};

```

#### 4.9 Tangent line of two circles

```
vector<Line> go( const Cir& c1 , const Cir& c2 , int
    sign1 ){
    // sign1 = 1 for outer tang, -1 for inner tang
    vector<Line> ret;
    double d_sq = norm2( c1.0 - c2.0 );
    if( d_sq < eps ) return ret;
    double d = sqrt( d_sq );
    Pt v = ( c2.0 - c1.0 ) / d;
    double c = ( c1.R - sign1 * c2.R ) / d;
    if( c * c > 1 ) return ret;
    double h = sqrt( max( 0.0 , 1.0 - c * c ) );
    for( int sign2 = 1 ; sign2 >= -1 ; sign2 -= 2 ){
        Pt n = { v.X * c - sign2 * h * v.Y ,
                v.Y * c + sign2 * h * v.X };
        Pt p1 = c1.0 + n * c1.R;
        Pt p2 = c2.0 + n * ( c2.R * sign1 );
        if( fabs( p1.X - p2.X ) < eps and
            fabs( p1.Y - p2.Y ) < eps )
            p2 = p1 + perp( c2.0 - c1.0 );
        ret.push_back( { p1 , p2 } );
    }
    return ret;
}
```

## 4.10 KD Tree

```
const int MXN=100005;
const int MXK=10;
```

```

struct KDTree{
    struct Nd{
        LL x[MXK],mn[MXK],mx[MXK];
        int id,f;
        Nd *l,*r;
    }tree[MXN],*root;
    int n,k;
    LL dis(LL a,LL b){return (a-b)*(a-b);}
    LL dis(LL a[MXK],LL b[MXK]){
        LL ret=0;
        for(int i=0;i<k;i++) ret+=dis(a[i],b[i]);
        return ret;
    }
    void init(vector<vector<LL>> &ip,int _n,int _k){
        n=_n,k=_k;
        for(int i=0;i<n;i++){
            tree[i].id=i;
            copy(ip[i].begin(),ip[i].end(),tree[i].x);
        }
        root=build(0,n-1,0);
    }
    Nd* build(int l,int r,int d){
        if(l>r) return NULL;
        if(d==k) d=0;
        int m=(l+r)>>1;
        nth_element(tree+l,tree+m,tree+r+1,[&](const Nd &a,
            const Nd &b){return a.x[d]<b.x[d];});
        tree[m].f=d;
        copy(tree[m].x,tree[m].x+k,tree[m].mn);
        copy(tree[m].x,tree[m].x+k,tree[m].mx);
        tree[m].l=build(l,m-1,d+1);
        if(tree[m].l){
            for(int i=0;i<k;i++){
                tree[m].mn[i]=min(tree[m].mn[i],tree[m].l->mn[i]);
                tree[m].mx[i]=max(tree[m].mx[i],tree[m].l->mx[i]);
            }
        }
        tree[m].r=build(m+1,r,d+1);
        if(tree[m].r){
            for(int i=0;i<k;i++){
                tree[m].mn[i]=min(tree[m].mn[i],tree[m].r->mn[i]);
                tree[m].mx[i]=max(tree[m].mx[i],tree[m].r->mx[i]);
            }
        }
        return tree+m;
    }
    LL pt[MXK],md;
    int mID;
    bool touch(Nd *r){
        LL d=0;
        for(int i=0;i<k;i++){
            if(pt[i]<=r->mn[i]) d+=dis(pt[i],r->mn[i]);
            else if(pt[i]>=r->mx[i]) d+=dis(pt[i],r->mx[i]);
        }
        return d<md;
    }
    void nearest(Nd *r){
        if(!r||!touch(r)) return;
        LL td=dis(r->x,pt);
        if(td<md) md=td,mID=r->id;
        nearest(pt[r->f]<r->x[r->f]?r->l:r->r);
        nearest(pt[r->f]>r->x[r->f]?r->r:r->l);
    }
    pair<LL,int> query(vector<LL> &_pt,LL _md=1LL<<57){
        mID=-1,md=_md;
        copy(_pt.begin(),_pt.end(),pt);
        nearest(root);
        return {md,mID};
    }
}tree;

```

#### 4.11 Lower Concave Hull

```

struct Line {
    mutable ll m, b, p;
    bool operator<(const Line& o) const { return m < o.m; }
}

```

```

bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b); }
    bool isect(iterator x, iterator y) {
        if (y == end()) { x->p = inf; return false; }
        if (x->m == y->m) x->p = x->b > y->b ? inf : -inf;
        else x->p = div(y->b - x->b, x->m - y->m);
        return x->p >= y->p;
    }
    void insert_line(ll m, ll b) {
        auto z = insert({m, b, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll eval(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.m * x + l.b;
    }
};

```

#### 4.12 Min Enclosing Circle

```

struct Mec{
    // return pair of center and r
    static const int N = 101010;
    int n;
    Pt p[ N ], cen;
    double r2;
    void init( int _n , Pt _p[] ){
        n = _n;
        memcpy( p , _p , sizeof(Pt) * n );
    }
    double sqr(double a){ return a*a; }
    Pt center(Pt p0, Pt p1, Pt p2) {
        Pt a = p1-p0;
        Pt b = p2-p0;
        double c1=norm2( a ) * 0.5;
        double c2=norm2( b ) * 0.5;
        double d = a ^ b;
        double x = p0.X + (c1 * b.Y - c2 * a.Y) / d;
        double y = p0.Y + (a.X * c2 - b.X * c1) / d;
        return Pt(x,y);
    }
    pair<Pt,double> solve(){
        random_shuffle(p,p+n);
        r2=0;
        for (int i=0; i<n; i++){
            if (norm2(cen-p[i]) <= r2) continue;
            cen = p[i];
            r2 = 0;
            for (int j=0; j<i; j++){
                if (norm2(cen-p[j]) <= r2) continue;
                cen=Pt((p[i].X+p[j].X)/2,(p[i].Y+p[j].Y)/2);
                r2 = norm2(cen-p[j]);
                for (int k=0; k<j; k++){
                    if (norm2(cen-p[k]) <= r2) continue;
                    cen = center(p[i],p[j],p[k]);
                    r2 = norm2(cen-p[k]);
                }
            }
        }
        return {cen,sqrt(r2)};
    }
} mec;

```

#### 4.13 Min Enclosing Ball

```

// Pt : { x , y , z }
#define N 202020
int n, nouter; Pt pt[ N ], outer[4], res;
double radius,tmp;
void ball() {
    Pt q[3]; double m[3][3], sol[3], L[3], det;

```

```

int i,j; res.x = res.y = res.z = radius = 0;
switch ( nouter ) {
case 1: res=outer[0]; break;
case 2: res=(outer[0]+outer[1])/2; radius=norm2(res, outer[0]); break;
case 3:
for (i=0; i<2; ++i) q[i]=outer[i+1]-outer[0];
for (i=0; i<2; ++i) for(j=0; j<2; ++j) m[i][j]=(q[i] * q[j])*2;
for (i=0; i<2; ++i) sol[i]=(q[i] * q[i]);
if (fabs(det=m[0][0]*m[1][1]-m[0][1]*m[1][0])<eps) return;
L[0]=(sol[0]*m[1][1]-sol[1]*m[0][1])/det;
L[1]=(sol[1]*m[0][0]-sol[0]*m[1][0])/det;
res=outer[0]+q[0]*L[0]+q[1]*L[1];
radius=norm2(res, outer[0]);
break;
case 4:
for (i=0; i<3; ++i) q[i]=outer[i+1]-outer[0], sol[i]=(q[i] * q[i]);
for (i=0; i<3; ++i) for(j=0; j<3; ++j) m[i][j]=(q[i] * q[j])*2;
det= m[0][0]*m[1][1]*m[2][2]
+ m[0][1]*m[1][2]*m[2][0]
+ m[0][2]*m[1][0]*m[2][1]
- m[0][2]*m[1][1]*m[2][0]
- m[0][1]*m[1][0]*m[2][2]
- m[0][0]*m[1][2]*m[2][1];
if ( fabs(det)<eps ) return;
for (j=0; j<3; ++j) {
for (i=0; i<3; ++i) m[i][j]=sol[i];
L[j]=( m[0][0]*m[1][1]*m[2][2]
+ m[0][1]*m[1][2]*m[2][0]
+ m[0][2]*m[1][0]*m[2][1]
- m[0][2]*m[1][1]*m[2][0]
- m[0][1]*m[1][0]*m[2][2]
- m[0][0]*m[1][2]*m[2][1]
) / det;
for (i=0; i<3; ++i) m[i][j]=(q[i] * q[j])*2;
} res=outer[0];
for (i=0; i<3; ++i) res = res + q[i] * L[i];
radius=norm2(res, outer[0]);
}}
void minball(int n){ ball();
if( nouter < 4 ) for( int i = 0 ; i < n ; i ++ )
if( norm2(res, pt[i]) - radius > eps ){
outer[ nouter ++ ] = pt[ i ]; minball(i); --
nouter;
if(i>0){ Pt Tt = pt[i];
memmove(&pt[1], &pt[0], sizeof(Pt)*i); pt[0]=Tt;
};
}}
double solve(){
// n points in pt
random_shuffle(pt, pt+n); radius=-1;
for(int i=0;i<n;i++) if(norm2(res,pt[i])-radius>eps)
nouter=1, outer[0]=pt[i], minball(i);
return sqrt(radius);
}

```

#### 4.14 Minkowski Sum

```

vector<Pt> minkowski(vector<Pt> p, vector<Pt> q){
int n = p.size(), m = q.size();
Pt c = Pt(0, 0);
for( int i = 0; i < m; i ++ ) c = c + q[i];
c = c / m;
for( int i = 0; i < m; i ++ ) q[i] = q[i] - c;
int cur = -1;
for( int i = 0; i < m; i ++ )
if( (q[i] ^ (p[0] - p[n-1])) > -eps)
if( cur == -1 || (q[i] ^ (p[0] - p[n-1])) >
(q[cur] ^ (p[0] - p[n-1])) )
cur = i;
vector<Pt> h;
p.push_back(p[0]);
for( int i = 0; i < n; i ++ )
while( true ){
h.push_back(p[i] + q[cur]);
int nxt = (cur + 1 == m ? 0 : cur + 1);
if((q[cur] ^ (p[i+1] - p[i])) < -eps) cur = nxt;
else if( (q[nxt] ^ (p[i+1] - p[i])) >

```

```

(q[cur] ^ (p[i+1] - p[i])) ) cur = nxt;
else break;
}
for(auto &&i : h) i = i + c;
return convex_hull(h);
}

```

#### 4.15 Min dist on Cuboid

```

typedef LL T;
T r;
void turn(T i, T j, T x, T y, T z,
T x0, T y0, T L, T W, T H) {
if (z==0) { T R = x*x+y*y; if (R<r) r=R; return; }
if(i>0 && i< 2) turn(i+1, j, x0+L+z, y, x0+L-x,
x0+L, y0, H, W, L);
if(j>0 && j< 2) turn(i, j+1, x, y0+W+z, y0+W-y,
x0, y0+W, L, H, W);
if(i<=0 && i>-2) turn(i-1, j, x0-z, y, x-x0,
x0-H, y0, H, W, L);
if(j<=0 && j>-2) turn(i, j-1, x, y0-z, y-y0,
x0, y0-H, L, H, W);
}
T solve(T L, T W, T H,
T x1, T y1, T z1, T x2, T y2, T z2){
if( z1!=0 && z1!=H ){
if( y1==0 || y1==W )
swap(y1,z1), swap(y2,z2), swap(W,H);
else swap(x1,z1), swap(x2,z2), swap(L,H);
}
if (z1==H) z1=0, z2=H-z2;
r=INF; turn(0,0,x2-x1,y2-y1,z2,-x1,-y1,L,W,H);
return r;
}

```

#### 4.16 Heart of Triangle

```

Pt inCenter( Pt &A, Pt &B, Pt &C) { // 內心
double a = norm(B-C), b = norm(C-A), c = norm(A-B);
return (A * a + B * b + C * c) / (a + b + c);
}
Pt circumCenter( Pt &a, Pt &b, Pt &c) { // 外心
Pt bb = b - a, cc = c - a;
double db=norm2(bb), dc=norm2(cc), d=2*(bb ^ cc);
return a-Pt(bb.Y*dc-cc.Y*db, cc.X*db-bb.X*dc) / d;
}
Pt orthoCenter( Pt &a, Pt &b, Pt &c) { // 垂心
Pt ba = b - a, ca = c - a, bc = b - c;
double Y = ba.Y * ca.Y * bc.Y,
A = ca.X * ba.Y - ba.X * ca.Y,
x0= (Y+ca.X*ba.Y*b.X-ba.X*ca.Y*c.X) / A,
y0= -ba.X * (x0 - c.X) / ba.Y + ca.Y;
return Pt(x0, y0);
}

```

#### 4.17 Closest Pair

```

#define x first
#define y second
vector<pair<int, int> > p;
set<pair<int, int> > s;
ld dis(pair<int, int> &a, pair<int, int> &b){
ld x = a.x-b.x, y = a.y-b.y;
return sqrt(x*x + y*y);
}
int main(){
int n;
while(cin>>n){
p.assign(n,{0,0});
for(int i = 0;i < n;i++)cin>>p[i].x>>p[i].y;
sort(p.begin(),p.end());
s.clear();
s.insert({p[0].y,p[0].x});
int l = 0;ld ans = 5e18;
for(int i = 1;i < n;i++){
int d = ceil(ans);
while(l < i && p[l].x < p[i].x - d){
s.erase({p[l].y,p[l].x});
l++;
}
auto it_l = s.lower_bound({p[i].y - d,0});
auto it_r = s.upper_bound({p[i].y + d,0});
}
}

```

```

        for(auto it = it_l; it != it_r; it++){
            ans = min(ans, dis({it->y, it->x}, p[i]));
        }
        s.insert({p[i].y, p[i].x});
    }
    cout<<ans<<endl;
}
return 0;
}

```

## 5 Graph

### 5.1 MaximumClique 最大團

```

#define N 111
struct MaxClique{ // 0-base
    typedef bitset<N> Int;
    Int linkto[N], v[N];
    int n;
    void init(int _n){
        n = _n;
        for(int i = 0; i < n; i++){
            linkto[i].reset(); v[i].reset();
        }
    }
    void addEdge(int a, int b)
    { v[a][b] = v[b][a] = 1; }
    int popcount(const Int& val)
    { return val.count(); }
    int lowbit(const Int& val)
    { return val._Find_first(); }
    int ans, stk[N];
    int id[N], di[N], deg[N];
    Int cans;
    void maxclique(int elem_num, Int candi){
        if(elem_num > ans){
            ans = elem_num; cans.reset();
            for(int i = 0; i < elem_num; i++){
                cans[id[stk[i]]] = 1;
            }
            int potential = elem_num + popcount(candi);
            if(potential <= ans) return;
            int pivot = lowbit(candi);
            Int smaller_candi = candi & (~linkto[pivot]);
            while(smaller_candi.count() && potential > ans){
                int next = lowbit(smaller_candi);
                candi[next] = !candi[next];
                smaller_candi[next] = !smaller_candi[next];
                potential--;
                if(next == pivot || (smaller_candi & linkto[next]).count()){
                    stk[elem_num] = next;
                    maxclique(elem_num + 1, candi & linkto[next]);
                }
            }
        }
        int solve(){
            for(int i = 0; i < n; i++){
                id[i] = i; deg[i] = v[i].count();
            }
            sort(id, id + n, [&](int id1, int id2){
                return deg[id1] > deg[id2]; });
            for(int i = 0; i < n; i++) di[id[i]] = i;
            for(int i = 0; i < n; i++){
                for(int j = 0; j < n; j++){
                    if(v[i][j]) linkto[di[i]][di[j]] = 1;
                }
            }
            Int cand; cand.reset();
            for(int i = 0; i < n; i++) cand[i] = 1;
            ans = 1;
            cans.reset(); cans[0] = 1;
            maxclique(0, cand);
            return ans;
        }
    } solver;
}

```

### 5.2 MaximalClique 極大團

```

#define N 80
struct MaxClique{ // 0-base
    typedef bitset<N> Int;
    Int lnk[N], v[N];
    int n;

```

```

    void init(int _n){
        n = _n;
        for(int i = 0; i < n; i++){
            lnk[i].reset(); v[i].reset();
        }
    }
    void addEdge(int a, int b)
    { v[a][b] = v[b][a] = 1; }
    int ans, stk[N], id[N], di[N], deg[N];
    Int cans;
    void dfs(int elem_num, Int candi, Int ex){
        if(candi.none() && ex.none()){
            cans.reset();
            for(int i = 0; i < elem_num; i++){
                cans[id[stk[i]]] = 1;
            }
            ans = elem_num; // cans is a maximal clique
            return;
        }
        int pivot = (candi|ex)._Find_first();
        Int smaller_candi = candi & (~lnk[pivot]);
        while(smaller_candi.count()){
            int nxt = smaller_candi._Find_first();
            candi[nxt] = smaller_candi[nxt] = 0;
            ex[nxt] = 1;
            stk[elem_num] = nxt;
            dfs(elem_num+1, candi&lnk[nxt], ex&lnk[nxt]);
        }
    }
    int solve(){
        for(int i = 0; i < n; i++){
            id[i] = i; deg[i] = v[i].count();
        }
        sort(id, id + n, [&](int id1, int id2){
            return deg[id1] > deg[id2]; });
        for(int i = 0; i < n; i++) di[id[i]] = i;
        for(int i = 0; i < n; i++){
            for(int j = 0; j < n; j++){
                if(v[i][j]) lnk[di[i]][di[j]] = 1;
            }
        }
        ans = 1; cans.reset(); cans[0] = 1;
        dfs(0, Int(string(n, '1')), 0);
        return ans;
    }
} solver;

```

### 5.3 Strongly Connected Component

```

struct Scc{
    int n, nScc, vst[MXN], bln[MXN], num;
    int dfn[MXN], low[MXN];
    vector<int> E[MXN];
    stack<int> stk;
    void init(int _n){
        n = _n;
        num = nScc = 0;
        for(int i = 0; i <= n; ++i) E[i].clear();
        FZ(vst); FZ(dfn); FZ(low);
        while(!stk.empty()) stk.pop();
    }
    void addEdge(int u, int v){E[u].pb(v);}
    void dfnlow(int u){
        int v;
        dfn[u] = low[u] = ++num;
        stk.push(u); vst[u] = 1;
        for(int i = 0; i < E[u].size(); ++i){
            v = E[u][i];
            if(!dfn[v]){dfnlow(v); low[u] = min(low[u], low[v]);}
            else if(vst[v]) low[u] = min(low[u], dfn[v]);
        }
        if(dfn[u] == low[u]){
            int t;
            ++nScc;
            do{
                t = stk.top(); stk.pop();
                vst[t] = 0; bln[t] = nScc;
            }while (t != u);
        }
    }
    void solve() {
        for(int i = 1; i <= n; ++i) if(!dfn[i]) dfnlow(i);
    }
}

```

### 5.4 Dynamic MST

```

/* Dynamic MST  $O(Q \lg^2 Q)$ 
(qx[i], qy[i]) -> chg weight of edge No.qx[i] to qy[i]
delete an edge: (i, \infty)
add an edge: change from \infty to specific value */
const int SZ=M+3*MXQ;
int a[N],*tz;
int find(int xx){
    int root=xx; while(a[root]) root=a[root];
    int next; while((next=a[xx])){a[xx]=root; xx=next; }
    return root;
}
bool cmp(int aa,int bb){ return tz[aa]<tz[bb]; }
int kx[N],ky[N],kt, vd[N],id[M], app[M];
bool extra[M];
void solve(int *qx,int *qy,int Q,int n,int *x,int *y,
    int *z,int m1,long long ans){
    if(Q==1){
        for(int i=1;i<=n;i++) a[i]=0;
        z[ qx[0] ]=qy[0]; tz = z;
        for(int i=0;i<m1;i++) id[i]=i;
        sort(id,id+m1,cmp); int ri,rj;
        for(int i=0;i<m1;i++){
            ri=find(x[id[i]]); rj=find(y[id[i]]);
            if(ri!=rj){ ans+=z[id[i]]; a[ri]=rj; }
        }
        printf("%lld\n",ans);
        return;
    }
    int ri,rj;
    //contract
    kt=0;
    for(int i=1;i<=n;i++) a[i]=0;
    for(int i=0;i<Q;i++){
        ri=find(x[qx[i]]); rj=find(y[qx[i]]); if(ri!=rj) a[ri]=rj;
    }
    int tm=0;
    for(int i=0;i<m1;i++) extra[i]=true;
    for(int i=0;i<Q;i++) extra[ qx[i] ]=false;
    for(int i=0;i<m1;i++) if(extra[i]) id[tm++]=i;
    tz=z; sort(id,id+tm,cmp);
    for(int i=0;i<tm;i++){
        ri=find(x[id[i]]); rj=find(y[id[i]]);
        if(ri!=rj){
            a[ri]=rj; ans += z[id[i]];
            kx[kt]=x[id[i]]; ky[kt]=y[id[i]]; kt++;
        }
    }
    for(int i=1;i<=n;i++) a[i]=0;
    for(int i=0;i<kt;i++) a[ find(kx[i]) ]=find(ky[i]);
    int n2=0;
    for(int i=1;i<=n;i++) if(a[i]==0)
        vd[i]=++n2;
    for(int i=1;i<=n;i++) if(a[i])
        vd[i]=vd[find(i)];
    int m2=0, *Nx=x+m1, *Ny=y+m1, *Nz=z+m1;
    for(int i=0;i<m1;i++) app[i]=-1;
    for(int i=0;i<Q;i++) if(app[qx[i]]==-1){
        Nx[m2]=vd[ x[ qx[i] ] ]; Ny[m2]=vd[ y[ qx[i] ] ];
        Nz[m2]=z[ qx[i] ];
        app[qx[i]]=m2; m2++;
    }
    for(int i=0;i<Q;i++){ z[ qx[i] ]=qy[i]; qx[i]=app[qx[i]]; }
    for(int i=1;i<=n2;i++) a[i]=0;
    for(int i=0;i<tm;i++){
        ri=find(vd[ x[id[i]] ]); rj=find(vd[ y[id[i]] ]);
        if(ri!=rj){
            a[ri]=rj; Nx[m2]=vd[ x[id[i]] ];
            Ny[m2]=vd[ y[id[i]] ]; Nz[m2]=z[id[i]]; m2++;
        }
    }
    int mid=Q/2;
    solve(qx,qy,mid,n2,Nx,Ny,Nz,m2,ans);
    solve(qx+mid,qy+mid,Q-mid,n2,Nx,Ny,Nz,m2,ans);
}
int x[SZ],y[SZ],z[SZ],qx[MXQ],qy[MXQ],n,m,Q;
void init(){
    scanf("%d%d",&n,&m);
    for(int i=0;i<m;i++) scanf("%d%d%d",x+i,y+i,z+i);
    scanf("%d",&Q);
    for(int i=0;i<Q;i++){ scanf("%d%d",qx+i,qy+i); qx[i]
        ]--; }
}

```

```
void work(){ if(Q) solve(qx,qy,Q,n,x,y,z,m,0); }
```

## 5.5 Maximum General graph Matching

```

const int N = 514, E = (2e5) * 2;
struct Graph{
    int to[E],bro[E],head[N],e;
    int lnk[N],vis[N],stp,n;
    void init( int _n ){
        stp = 0; e = 1; n = _n;
        for( int i = 1 ; i <= n ; i ++ )
            lnk[i] = vis[i] = 0;
    }
    void add_edge(int u,int v){
        to[e]=v,bro[e]=head[u],head[u]=e++;
        to[e]=u,bro[e]=head[v],head[v]=e++;
    }
    bool dfs(int x){
        vis[x]=stp;
        for(int i=head[x];i;i=bro[i]){
            int v=to[i];
            if(!lnk[v]){
                lnk[x]=v,lnk[v]=x;
                return true;
            }else if(vis[lnk[v]]<stp){
                int w=lnk[v];
                lnk[x]=v,lnk[v]=x,lnk[w]=0;
                if(dfs(w)){
                    return true;
                }
                lnk[w]=v,lnk[v]=w,lnk[x]=0;
            }
        }
        return false;
    }
    int solve(){
        int ans = 0;
        for(int i=1;i<=n;i++)
            if(!lnk[i]){
                stp++; ans += dfs(i);
            }
        return ans;
    }
} graph;

```

## 5.6 Minimum General Weighted Matching

```

struct Graph {
    // Minimum General Weighted Matching (Perfect Match)
    static const int MXN = 105;
    int n, edge[MXN][MXN];
    int match[MXN],dis[MXN],onstk[MXN];
    vector<int> stk;
    void init(int _n) {
        n = _n;
        for( int i = 0 ; i < n ; i ++ )
            for( int j = 0 ; j < n ; j ++ )
                edge[ i ][ j ] = 0;
    }
    void add_edge(int u, int v, int w)
    { edge[u][v] = edge[v][u] = w; }
    bool SPFA(int u){
        if (onstk[u]) return true;
        stk.PB(u);
        onstk[u] = 1;
        for (int v=0; v<n; v++){
            if (u != v && match[u] != v && !onstk[v]){
                int m = match[v];
                if (dis[m] > dis[u] - edge[v][m] + edge[u][v]){
                    dis[m] = dis[u] - edge[v][m] + edge[u][v];
                    onstk[v] = 1;
                    stk.PB(v);
                    if (SPFA(m)) return true;
                    stk.pop_back();
                    onstk[v] = 0;
                }
            }
        }
        onstk[u] = 0;
        stk.pop_back();
        return false;
    }
    int solve() {
        // find a match
    }
}

```

```

for (int i=0; i<n; i+=2){
    match[i] = i+1;
    match[i+1] = i;
}
while (true){
    int found = 0;
    for( int i = 0 ; i < n ; i ++ )
        onstk[ i ] = dis[ i ] = 0;
    for (int i=0; i<n; i++){
        stk.clear();
        if (!onstk[i] && SPFA(i)){
            found = 1;
            while (SZ(stk)>=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 += edge[i][match[i]];
    ret /= 2;
    return ret;
}
}graph;

```

## 5.7 Minimum Steiner Tree

```

// Minimum Steiner Tree 重要點的mst
// O(V 3^T + V^2 2^T)
struct SteinerTree{
#define V 33
#define T 8
#define INF 1023456789
    int n , dst[V][V] , dp[1 << T][V] , tdst[V];
    void init( int _n ){
        n = _n;
        for( int i = 0 ; i < n ; i ++ ){
            for( int j = 0 ; j < n ; j ++ )
                dst[ i ][ j ] = INF;
            dst[ i ][ i ] = 0;
        }
        void add_edge( int ui , int vi , int wi ){
            dst[ ui ][ vi ] = min( dst[ ui ][ vi ] , wi );
            dst[ vi ][ ui ] = min( dst[ vi ][ ui ] , wi );
        }
        void shortest_path(){
            for( int k = 0 ; k < n ; k ++ )
                for( int i = 0 ; i < n ; i ++ )
                    for( int j = 0 ; j < n ; j ++ )
                        dst[ i ][ j ] = min( dst[ i ][ j ] ,
                            dst[ i ][ k ] + dst[ k ][ j ] );
        }
        int solve( const vector<int>& ter ){
            int t = (int)ter.size();
            for( int i = 0 ; i < ( 1 << t ) ; i ++ )
                for( int j = 0 ; j < n ; j ++ )
                    dp[ i ][ j ] = INF;
            for( int i = 0 ; i < n ; i ++ )
                dp[ 0 ][ i ] = 0;
            for( int msk = 1 ; msk < ( 1 << t ) ; msk ++ ){
                if( msk == ( msk & (-msk) ) ){
                    int who = __lg( msk );
                    for( int i = 0 ; i < n ; i ++ )
                        dp[ msk ][ i ] = dst[ ter[ who ] ][ i ];
                    continue;
                }
                for( int i = 0 ; i < n ; i ++ )
                    for( int submsk = ( msk - 1 ) & msk ; submsk ;
                        submsk = ( submsk - 1 ) & msk )
                        dp[ msk ][ i ] = min( dp[ msk ][ i ] ,
                            dp[ submsk ][ i ] +
                            dp[ msk ^ submsk ][ i ] );
                for( int i = 0 ; i < n ; i ++ ){
                    tdst[ i ] = INF;
                    for( int j = 0 ; j < n ; j ++ )
                        tdst[ i ] = min( tdst[ i ] ,
                            dp[ msk ][ j ] + dst[ j ][ i ] );
                }
            }
            for( int i = 0 ; i < n ; i ++ )

```

```

        dp[ msk ][ i ] = tdst[ i ];
    }
    int ans = INF;
    for( int i = 0 ; i < n ; i ++ )
        ans = min( ans , dp[ ( 1 << t ) - 1 ][ i ] );
    return ans;
} }solver;

```

## 5.8 BCC based on vertex

```

struct BccVertex {
    int n,nScc,step,dfn[MXN],low[MXN];
    vector<int> E[MXN],sccv[MXN];
    int top,stk[MXN];
    void init(int _n) {
        n = _n; nScc = step = 0;
        for (int i=0; i<n; i++) E[i].clear();
    }
    void addEdge(int u, int v)
    { E[u].PB(v); E[v].PB(u); }
    void DFS(int u, int f) {
        dfn[u] = low[u] = step++;
        stk[top++] = u;
        for (auto v:E[u]) {
            if (v == f) continue;
            if (dfn[v] == -1) {
                DFS(v,u);
                low[u] = min(low[u], low[v]);
                if (low[v] >= dfn[u]) {
                    int z;
                    sccv[nScc].clear();
                    do {
                        z = stk[--top];
                        sccv[nScc].PB(z);
                    } while (z != v);
                    sccv[nScc++].PB(u);
                }
            } else
                low[u] = min(low[u], dfn[v]);
        }
    }
    vector<vector<int>> solve() {
        vector<vector<int>> res;
        for (int i=0; i<n; i++)
            dfn[i] = low[i] = -1;
        for (int i=0; i<n; i++)
            if (dfn[i] == -1) {
                top = 0;
                DFS(i,i);
            }
        REP(i,nScc) res.PB(sccv[i]);
        return res;
    }
}graph;

```

## 5.9 Min Mean Cycle

```

/* minimum mean cycle O(VE) */
struct MMC{
#define E 101010
#define V 1021
#define inf 1e9
#define eps 1e-6
    struct Edge { int v,u; double c; };
    int n, m, prv[V][V], prve[V][V], vst[V];
    Edge e[E];
    vector<int> edgeID, cycle, rho;
    double d[V][V];
    void init( int _n )
    { n = _n; m = 0; }
    // WARNING: TYPE matters
    void addEdge( int vi , int ui , double ci )
    { e[ m ++ ] = { vi , ui , ci }; }
    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 solve(){
    // 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);
    }
    fill(vst,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) {
        if(rho.empty()) return inf;
        int v = rho.back(); rho.pop_back();
        cycle.PB(v);
        vst[v]++;
    }
    reverse(ALL(edgeID));
    edgeID.resize(SZ(cycle));
    return mmc;
} }mmc;

```

## 5.10 Directed Graph Min Cost Cycle

```

// works in O(N M)
#define INF 10000000000000LL
#define N 5010
#define M 200010
struct edge{
    int to; LL w;
    edge(int a=0, LL b=0): to(a), w(b){}
};
struct node{
    LL d; int u, next;
    node(LL a=0, int b=0, int c=0): d(a), u(b), next(c){}
}b[M];
struct DirectedGraphMinCycle{
    vector<edge> g[N], grev[N];
    LL dp[N][N], p[N], d[N], mu;
    bool inq[N];
    int n, bn, bsz, hd[N];
    void b_insert(LL d, int u){
        int i = d/mu;
        if(i >= bn) return;
        b[++bsz] = node(d, u, hd[i]);
        hd[i] = bsz;
    }
    void init( int _n ){
        n = _n;
        for( int i = 1 ; i <= n ; i ++ )
            g[ i ].clear();
    }
    void addEdge( int ai , int bi , LL ci )
    { g[ai].push_back(edge(bi,ci)); }
    LL solve(){
        fill(dp[0], dp[0]+n+1, 0);
        for(int i=1; i<=n; i++){
            fill(dp[i]+1, dp[i]+n+1, INF);
            for(int j=1; j<=n; j++) if(dp[i-1][j] < INF){
                for(int k=0; k<(int)g[j].size(); k++){
                    dp[i][g[j][k].to] =min(dp[i][g[j][k].to],
                                            dp[i-1][j]+g[j][k].w);
                }
            }
        }
        mu=INF; LL bunbo=1;
        for(int i=1; i<=n; i++) if(dp[n][i] < INF){
            LL a=-INF, b=1;
            for(int j=0; j<=n-1; j++) if(dp[j][i] < INF){
                if(a*(n-j) < b*(dp[n][i]-dp[j][i])){
                    a = dp[n][i]-dp[j][i];
                    b = n-j;
                }
            }
        }
    }
}

```

```

if(mu*b > bunbo*a)
    mu = a, bunbo = b;
}
if(mu < 0) return -1; // negative cycle
if(mu == INF) return INF; // no cycle
if(mu == 0) return 0;
for(int i=1; i<=n; i++){
    for(int j=0; j<(int)g[i].size(); j++){
        g[i][j].w *= bunbo;
        memset(p, 0, sizeof(p));
        queue<int> q;
        for(int i=1; i<=n; i++){
            q.push(i);
            inq[i] = true;
        }
        while(!q.empty()){
            int i=q.front(); q.pop(); inq[i]=false;
            for(int j=0; j<(int)g[i].size(); j++){
                if(p[g[i][j].to] > p[i]+g[i][j].w-mu){
                    p[g[i][j].to] = p[i]+g[i][j].w-mu;
                    if(!inq[g[i][j].to]){
                        q.push(g[i][j].to);
                        inq[g[i][j].to] = true;
                    }
                }
            }
        }
        for(int i=1; i<=n; i++) grev[i].clear();
        for(int i=1; i<=n; i++){
            for(int j=0; j<(int)g[i].size(); j++){
                g[i][j].w += p[i]-p[g[i][j].to];
                grev[g[i][j].to].push_back(edge(i, g[i][j].w));
            }
        }
        LL mldc = n*mu;
        for(int i=1; i<=n; i++){
            bn=mldc/mu, bsz=0;
            memset(hd, 0, sizeof(hd));
            fill(d+i+1, d+n+1, INF);
            b_insert(d[i]=0, i);
            for(int j=0; j<=bn-1; j++) for(int k=hd[j]; k; k=
                b[k].next){
                int u = b[k].u;
                LL du = b[k].d;
                if(du > d[u]) continue;
                for(int l=0; l<(int)g[u].size(); l++) if(g[u][l]
                    ].to > i){
                    if(d[g[u][l].to] > du + g[u][l].w){
                        d[g[u][l].to] = du + g[u][l].w;
                        b_insert(d[g[u][l].to], g[u][l].to);
                    }
                }
            }
            for(int j=0; j<(int)grev[i].size(); j++) if(grev[
                i][j].to > i)
                mldc=min(mldc,d[grev[i][j].to] + grev[i][j].w);
        }
        return mldc / bunbo;
    }
} }graph;

```

## 5.11 K-th Shortest Path

```

// time: O(|E| \lg |E| + |V| \lg |V| + K)
// memory: O(|E| \lg |E| + |V|)
struct KSP{ // 1-base
    struct nd{
        int u, v; ll d;
        nd(int ui = 0, int vi = 0, ll di = INF)
        { u = ui; v = vi; d = di; }
    };
    struct heap{
        nd* edge; int dep; heap* chd[4];
    };
    static int cmp(heap* a, heap* b)
    { return a->edge->d > b->edge->d; }
    struct node{
        int v; ll d; heap* H; nd* E;
        node(){
            node(ll _d, int _v, nd* _E)
            { d = _d; v = _v; E = _E; }
            node(heap* _H, ll _d)
            { H = _H; d = _d; }
            friend bool operator<(node a, node b)
            { return a.d > b.d; }
        };
        int n, k, s, t;
        ll dst[ N ];
        nd *nxt[ N ];
    };
}

```

```

vector<nd*> g[ N ], rg[ N ];
heap *nullNd, *head[ N ];
void init( int _n , int _k , int _s , int _t ){
    n = _n; k = _k; s = _s; t = _t;
    for( int i = 1 ; i <= n ; i ++ ){
        g[ i ].clear(); rg[ i ].clear();
        nxt[ i ] = NULL; head[ i ] = NULL;
        dst[ i ] = -1;
    }
}
void addEdge( int ui , int vi , ll di ){
    nd* e = new nd(ui, vi, di);
    g[ ui ].push_back( e );
    rg[ vi ].push_back( e );
}
queue<int> dfsQ;
void dijkstra(){
    while(dfsQ.size()) dfsQ.pop();
    priority_queue<node> Q;
    Q.push(node(0, t, NULL));
    while (!Q.empty()){
        node p = Q.top(); Q.pop();
        if(dst[p.v] != -1) continue;
        dst[ p.v ] = p.d;
        nxt[ p.v ] = p.E;
        dfsQ.push( p.v );
        for(auto e: rg[ p.v ])
            Q.push(node(p.d + e->d, e->u, e));
    }
}
heap* merge(heap* curNd, heap* newNd){
    if(curNd == nullNd) return newNd;
    heap* root = new heap;
    memcpy(root, curNd, sizeof(heap));
    if(newNd->edge->d < curNd->edge->d){
        root->edge = newNd->edge;
        root->chd[2] = newNd->chd[2];
        root->chd[3] = newNd->chd[3];
        newNd->edge = curNd->edge;
        newNd->chd[2] = curNd->chd[2];
        newNd->chd[3] = curNd->chd[3];
    }
    if(root->chd[0]->dep < root->chd[1]->dep)
        root->chd[0] = merge(root->chd[0], newNd);
    else
        root->chd[1] = merge(root->chd[1], newNd);
    root->dep = max(root->chd[0]->dep, root->chd[1]->
        dep) + 1;
    return root;
}
vector<heap*> V;
void build(){
    nullNd = new heap;
    nullNd->dep = 0;
    nullNd->edge = new nd;
    fill(nullNd->chd, nullNd->chd+4, nullNd);
    while(not dfsQ.empty()){
        int u = dfsQ.front(); dfsQ.pop();
        if(!nxt[ u ]) head[ u ] = nullNd;
        else head[ u ] = head[nxt[ u ]->v];
        V.clear();
        for( auto&& e : g[ u ] ){
            int v = e->v;
            if( dst[ v ] == -1 ) continue;
            e->d += dst[ v ] - dst[ u ];
            if( nxt[ u ] != e ){
                heap* p = new heap;
                fill(p->chd, p->chd+4, nullNd);
                p->dep = 1;
                p->edge = e;
                V.push_back(p);
            }
            if(V.empty()) continue;
            make_heap(V.begin(), V.end(), cmp);
#define L(X) ((X<<1)+1)
#define R(X) ((X<<1)+2)
            for( size_t i = 0 ; i < V.size() ; i ++ ){
                if(L(i) < V.size()) V[i]->chd[2] = V[L(i)];
                else V[i]->chd[2]=nullNd;
                if(R(i) < V.size()) V[i]->chd[3] = V[R(i)];
                else V[i]->chd[3]=nullNd;
            }
            head[ u ] = merge(head[ u ], V.front());
        }
    }
}

```

```

vector<ll> ans;
void first_K(){
    ans.clear();
    priority_queue<node> Q;
    if( dst[ s ] == -1 ) return;
    ans.push_back( dst[ s ] );
    if( head[ s ] != nullNd )
        Q.push(node(head[ s ], dst[s]+head[s]->edge->d));
    for( int _ = 1 ; _ < k and not Q.empty() ; _ ++ ){
        node p = Q.top(), q; Q.pop();
        ans.push_back( p.d );
        if(head[ p.H->edge->v ] != nullNd){
            q.H = head[ p.H->edge->v ];
            q.d = p.d + q.H->edge->d;
            Q.push(q);
        }
    }
    for( int i = 0 ; i < 4 ; i ++ )
        if( p.H->chd[ i ] != nullNd ){
            q.H = p.H->chd[ i ];
            q.d = p.d - p.H->edge->d + p.H->chd[ i ]->
                edge->d;
            Q.push( q );
        }
}
void solve(){ // ans[i] stores the i-th shortest path
    dijkstra();
    build();
    first_K(); // ans.size() might less than k
}
} solver;

```

## 5.12 SPFA

```

bool spfa(){
    deque<int> dq;
    dis[0]=0;
    dq.push_back(0);
    inq[0]=1;
    while(!dq.empty()){
        int u=dq.front();
        dq.pop_front();
        inq[u]=0;
        for(auto i:edge[u]){
            if(dis[i.first]>i.second+dis[u]){
                dis[i.first]=i.second+dis[u];
                len[i.first]=len[u]+1;
                if(len[i.first]>n) return 1;
                if(inq[i.first]) continue;
                if(!dq.empty()&&dis[dq.front()]>dis[i.
                    first])
                    dq.push_front(i.first);
                else
                    dq.push_back(i.first);
                inq[i.first]=1;
            }
        }
    }
    return 0;
}

```

## 5.13 Kruskal

```

struct Edge{
    int u,v,w;
    friend bool operator<(const Edge& lhs,const Edge&
        rhs){
        return lhs.w<rhs.w;
    }
};
vector<Edge> graph;
void kruskal(){
    int sum=0;
    sort(graph.begin(),graph.end());
    for(auto i:graph){
        if(Find(i.u)!=Find(i.v)){
            Union(find(i.u),find(i.v));
            sum+=i.w;
        }
    }
    cout<<sum<<endl;
}

```

## 5.14 Dijkstra

```

void dijkstra(int startPoint,int endPoint){

```

```

priority_queue<pair<ll,int>,vector<pair<ll,int>>,
    greater<pair<ll,int>>> pq;
v.clear();v.resize(n);
dis.clear();dis.resize(n,INF);
dis[startPoint]=0;
pq.push({dis[startPoint],startPoint});
//將起點加進去pq裡面
while(!pq.empty()){
    //當pq還有東西時繼續做
    auto u=pq.top();pq.pop();
    //每次取出離起點最近的點
    if(v[u.second]) continue;
    //如果之前走過代表已經有更短的路經過不用在重新走
    v[u.second]=1;
    //設成走過
    for(auto i:edge[u.second]){
        if(dis[i.first]>u.first+i.second){
            //判斷是否可以鬆弛
            dis[i.first]=u.first+i.second;
            pq.push({dis[i.first],i.first});
            //將可鬆弛的路連出去
        }
    }
}
cout<<dis[endPoint]<<endl;
}

```

## 5.15 LCA

```

#define MXN 100005
#define LOG 20
vector<int> edge[MXN];
int anc[MXN][LOG];
int tin[MXN],tout[MXN],ti=0;
void build(int x,int f){
    for(int i=0;i<LOG;i++){
        anc[x][i] = f;
        f = anc[f][i];
    }
}
void dfs(int x,int f){
    tin[x] = ti++;
    build(x,f);
    for(auto i:edge[x]){
        if(i == f) continue;
        dfs(i,x);
    }
    tout[x] = ti++;
}
bool isAnc(int x,int y){
    return tin[x] <= tin[y] && tout[x] >= tout[y];
}
int query(int x,int y){
    if(isAnc(x,y)) return x;
    if(isAnc(y,x)) return y;
    for(int i=LOG-1;i>=0;i--){
        if(!isAnc(anc[x][i],y)) x = anc[x][i];
    }
    return anc[x][0];
}

```

## 5.16 Bellman

```

void bellman(int startPoint){
    dis.clear();dis.resize(n,INF);
    dis[startPoint]=0;
    for(int i=1;i<n;i++){
        for(auto x:edge){
            int a=x.u,b=x.v,c=x.w;
            if(dis[a]+c<dis[b]){
                dis[b]=dis[a]+c;
            }
        }
    }
}

```

## 5.17 差分約束

約束條件  $V_j - V_i \leq W$  建邊  $V_i - V_j > V_j$  權重為  $W \rightarrow$  bellman-ford or spfa

## 5.18 Euler Path

```

#define FOR(i,a,b) for(int i=a;i<=b;i++)

```

```

int dfs_st[1000050],dfn=0;
int ans[1000050],cnt=0,num=0;
vector<int>G[1000050];
int cur[1000050];
int ind[1000050],out[1000050];
void dfs(int x){
    FOR(i,1,n)sort(G[i].begin(),G[i].end());
    dfs_st[++dfn]=x;
    memset(cur,-1,sizeof(cur));
    while(dfn>0){
        int u=dfs_st[dfn];
        int complete=1;
        for(int i=cur[u]+1;i<G[u].size();i++){
            int v=G[u][i];
            num++;
            dfs_st[++dfn]=v;
            cur[u]=i;
            complete=0;
            break;
        }
        if(complete)ans[++cnt]=u,dfn--;
    }
}
bool check(int &start){
    int l=0,r=0,mid=0;
    FOR(i,1,n){
        if(ind[i]==out[i]+1)l++;
        if(out[i]==ind[i]+1)r++,start=i;
        if(ind[i]==out[i])mid++;
    }
    if(l==1&&r==1&mid==n-2)return true;
    l=1;
    FOR(i,1,n)if(ind[i]!=out[i])l=0;
    if(l){
        FOR(i,1,n)if(out[i]>0){
            start=i;
            break;
        }
        return true;
    }
    return false;
}
int main(){
    cin>>n>>m;
    FOR(i,1,m){
        int x,y;scanf("%d%d",&x,&y);
        G[x].push_back(y);
        ind[y]++,out[x]++;
    }
    int start=-1,ok=true;
    if(check(start)){
        dfs(start);
        if(num!=m){
            puts("What a shame!");
            return 0;
        }
        for(int i=cnt;i>=1;i--){
            printf("%d ",ans[i]);
            puts("");
        }
        else puts("What a shame!");
    }
}

```

## 5.19 Tree Flatten

```

int ti=0;
vector<pair<int,int> > dfsTime(n);
int dfs(int x,int f){
    dfsTime[x].first=dfsTime[x].second=ti++;
    for(auto i:edge[x]){
        if(i==f) continue;
        dfsTime[x].second=max(dfsTime[x].second,
            dfs(i,x));//紀錄最右界
    }
    return dfsTime[x].second;
}

```

## 6 String

### 6.1 PalTree

```

// Len[s]是對應的回文長度

```

```
// num[s]是有幾個回文後綴
// cnt[s]是這個回文子字串在整個字串中的出現次數
// fail[s]是他長度次長的回文後綴 · aba的fail是a
const int MXN = 1000010;
struct PalT{
    int nxt[MXN][26], fail[MXN], len[MXN];
    int tot, lst, n, state[MXN], cnt[MXN], num[MXN];
    int diff[MXN], sfail[MXN], fac[MXN], dp[MXN];
    char s[MXN] = {-1};
    int newNode(int l, int f){
        len[tot] = l, fail[tot] = f, cnt[tot] = num[tot] = 0;
        memset(nxt[tot], 0, sizeof(nxt[tot]));
        diff[tot] = (l > 0 ? 1 - len[f] : 0);
        sfail[tot] = (l > 0 && diff[tot] == diff[f] ? sfail[f] : f);
        return tot++;
    }
    int getfail(int x){
        while(s[n - len[x] - 1] != s[n]) x = fail[x];
        return x;
    }
    int getmin(int v){
        dp[v] = fac[n - len[sfail[v]] - diff[v]];
        if(diff[v] == diff[fail[v]])
            dp[v] = min(dp[v], dp[fail[v]]);
        return dp[v] + 1;
    }
    int push(){
        int c = s[n] - 'a', np = getfail(lst);
        if(!lst || np[c] == 0){
            lst = newNode(len[np] + 2, np[getfail(fail[np])][c]);
            nxt[np][c] = lst; num[lst] = num[fail[lst]] + 1;
        }
        fac[n] = n;
        for(int v = lst; len[v] > 0; v = sfail[v])
            fac[n] = min(fac[n], getmin(v));
        return ++cnt[lst], lst;
    }
    void init(const char *_s){
        tot = lst = n = 0;
        newNode(0, 1), newNode(-1, 1);
        for(; _s[n];) s[n + 1] = _s[n], ++n, state[n - 1] = push();
        for(int i = tot - 1; i > 1; i--) cnt[fail[i]] += cnt[i];
    }
}palt;
```

## 6.2 KMP

/\*  
*len-failure[k]:*  
 在k結尾的情況下 · 這個子字串可以由開頭  
 長度為(*len-failure[k]*)的部分重複出現來表達

*failure[k]:*  
*failure[k]*為次長相同前綴後綴  
 如果我們不只想求最多 · 而且以0-base做為考量  
 · 那可能的長度由大到小會是  
*failuer[k]、failure[failuer[k]-1]*  
*、failure[failure[failuer[k]-1]-1]..*  
 直到有值為0為止

```
*/
int failure[MXN];
void KMP(string& t, string& p){
    if (p.size() > t.size()) return;
    for (int i = 1, j = failure[0] = -1; i < p.size(); ++i){
        while (j >= 0 && p[j + 1] != p[i]) j = failure[j];
        if (p[j + 1] == p[i]) j++;
        failure[i] = j;
    }
    for (int i = 0, j = -1; i < t.size(); ++i) {
        while (j >= 0 && p[j + 1] != t[i]) j = failure[j];
        if (p[j + 1] == t[i]) j++;
        if (j == p.size() - 1) {
            cout << i - p.size() + 1 << " ";
            j = failure[j];
        }
    }
}
```

## 6.3 Suffix Array

```
const int N = 300010;
struct SA{
```

```
#define REP(i,n) for ( int i=0; i<int(n); i++ )
#define REP1(i,a,b) for ( int i=(a); i<=int(b); i++ )
bool _t[N*2];
int _s[N*2], _sa[N*2], _c[N*2], x[N], _p[N], _q[N*2],
    hei[N], r[N];
int operator [] (int i){ return _sa[i]; }
void build(int *s, int n, int m){
    memcpy(_s, s, sizeof(int) * n);
    sais(_s, _sa, _p, _q, _t, _c, n, m);
    mkhei(n);
}
void mkhei(int n){
    REP(i,n) r[_sa[i]] = i;
    hei[0] = 0;
    REP(i,n) if(r[i]) {
        int ans = i > 0 ? max(hei[r[i-1]] - 1, 0) : 0;
        while(_s[i+ans] == _s[_sa[r[i]-1]+ans]) ans++;
        hei[r[i]] = ans;
    }
}
void sais(int *s, int *sa, int *p, int *q, bool *t,
    int *c, int n, int z){
    bool uniq = t[n-1] = true, neq;
    int nn = 0, nmzx = -1, *nsa = sa + n, *ns = s + n,
        lst = -1;
#define MS0(x,n) memset((x),0,n*sizeof(*(x)))
#define MAGIC(XD) MS0(sa, n); \
    memcpy(x, c, sizeof(int) * z); \
    XD; \
    memcpy(x + 1, c, sizeof(int) * (z - 1)); \
    REP(i,n) if(sa[i] && !t[sa[i]-1]) sa[x[s[sa[i]
        ]-1]++] = sa[i]-1; \
    memcpy(x, c, sizeof(int) * z); \
    for(int i = n - 1; i >= 0; i--) if(sa[i] && t[sa[i]
        ]-1) sa[--x[s[sa[i]-1]]] = sa[i]-1;
    MS0(c, z);
    REP(i,n) uniq &= ++c[s[i]] < 2;
    REP(i,z-1) c[i+1] += c[i];
    if (uniq) { REP(i,n) sa[--c[s[i]]] = i; return; }
    for(int i = n - 2; i >= 0; i--) t[i] = (s[i] == s[i
        + 1] ? t[i+1] : s[i] < s[i+1]);
    MAGIC(REP1(i,1,n-1) if(t[i] && !t[i-1]) sa[--x[s[i]
        ]]] = p[q[i]=nn++] = i);
    REP(i, n) if (sa[i] && t[sa[i]] && !t[sa[i]-1]) {
        neq = lst < 0 || memcmp(s+sa[i], s+lst, (p[q[sa[i]]+1]-sa
            [i])*sizeof(int));
        ns[q[lst=sa[i]]] = nmzx += neq;
    }
    sais(ns, nsa, p + nn, q + n, t + n, c + z, nn, nmzx
        + 1);
    MAGIC(for(int i = nn - 1; i >= 0; i--) sa[--x[p[
        nsa[i]]]] = p[nsa[i]]);
}
}sa;
int H[ N ], SA[ N ];
void suffix_array(int* ip, int len) {
    // should padding a zero in the back
    // ip is int array, len is array length
    // ip[0..n-1] != 0, and ip[len] = 0
    ip[len++] = 0;
    sa.build(ip, len, 128);
    for (int i = 0; i < len; i++) {
        H[i] = sa.hei[i + 1];
        SA[i] = sa._sa[i + 1];
    }
    // resulting height, sa array \in [0,len)
}
```

## 6.4 SuffixAutomata

```
// any path start from root forms a substring of S
// occurrence of P : iff SAM can run on input word P
// number of different substring : ds[1]-1
// total length of all different substring : dsl[1]
// max/min length of state i : mx[i]/mx[mom[i]]+1
// assume a run on input word P end at state i:
// number of occurrences of P : cnt[i]
// first occurrence position of P : fp[i]-|P|+1
// all position of P : fp of "dfs from i through rmom"
const int MXM = 1000010;
struct SAM{
    int tot, root, lst, mom[MXM], mx[MXM]; //ind[MXM]
```

```

int nxt[MXM][33]; //cnt[MXM],ds[MXM],dsl[MXM],fp[MXM]
// bool v[MXM]
int newNode(){
    int res = ++tot;
    fill(nxt[res], nxt[res]+33, 0);
    mom[res] = mx[res] = 0; //cnt=ds=dsl=fp=v=0
    return res;
}
void init(){
    tot = 0;
    root = newNode();
    lst = root;
}
void push(int c){
    int p = lst;
    int np = newNode(); //cnt[np]=1
    mx[np] = mx[p]+1; //fp[np]=mx[np]-1
    for(; p && nxt[p][c] == 0; p = mom[p])
        nxt[p][c] = np;
    if(p == 0) mom[np] = root;
    else{
        int q = nxt[p][c];
        if(mx[p]+1 == mx[q]) mom[np] = q;
        else{
            int nq = newNode(); //fp[nq]=fp[q]
            mx[nq] = mx[p]+1;
            for(int i = 0; i < 33; i++){
                nxt[nq][i] = nxt[q][i];
            }
            mom[nq] = mom[q];
            mom[q] = nq;
            mom[np] = nq;
            for(; p && nxt[p][c] == q; p = mom[p])
                nxt[p][c] = nq;
        }
    }
    lst = np;
}
void calc(){
    calc(root);
    iota(ind,ind+tot,1);
    sort(ind,ind+tot,[&](int i,int j){return mx[i]<mx[j];});
    for(int i=tot-1;i>=0;i--){
        cnt[mom[ind[i]]]+=cnt[ind[i]];
    }
}
void calc(int x){
    v[x]=ds[x]=1;dsl[x]=0; //rmom[mom[x]].push_back(x);
    for(int i=1;i<=26;i++){
        if(nxt[x][i]){
            if(!v[nxt[x][i]]) calc(nxt[x][i]);
            ds[x]+=ds[nxt[x][i]];
            dsl[x]+=ds[nxt[x][i]]+dsl[nxt[x][i]];
        }
    }
}
void push(const string& str){
    for(int i = 0; i < str.size(); i++){
        push(str[i]-'a'+1);
    }
}
} sam;

```

## 6.5 Aho-Corasick

```

struct AAutomata{
    struct Node{
        int cnt,i;
        Node *go[26], *fail, *dic;
        Node(){
            cnt = 0; fail = 0; dic=0;
            memset(go,0,sizeof(go));
        }
    }pool[1048576],*root;
    int nMem,n_pattern;
    Node* new_Node(){
        pool[nMem] = Node();
        return &pool[nMem++];
    }
    void init() {nMem=0;root=new_Node();n_pattern=0;}
    void add(const string &str) {insert(root,str,0); }
    void insert(Node *cur, const string &str, int pos){
        for(int i=pos;i<str.size();i++){
            if(!cur->go[str[i]-'a'])
                cur->go[str[i]-'a'] = new_Node();
            cur=cur->go[str[i]-'a'];
        }
    }
}

```

```

cur->cnt++; cur->i=n_pattern++;
}
void make_fail(){
    queue<Node*> que;
    que.push(root);
    while (!que.empty()){
        Node* fr=que.front(); que.pop();
        for (int i=0; i<26; i++){
            if (fr->go[i]){
                Node *ptr = fr->fail;
                while (ptr && !ptr->go[i]) ptr = ptr->fail;
                fr->go[i]->fail=ptr=(ptr?ptr->go[i]:root);
                fr->go[i]->dic=(ptr->cnt?ptr:ptr->dic);
                que.push(fr->go[i]);
            }
        }
    }
}
void query(string s){
    Node *cur=root;
    for(int i=0;i<(int)s.size();i++){
        while (cur&&!cur->go[s[i]-'a']) cur=cur->fail;
        cur=(cur?cur->go[s[i]-'a']:root);
        if(cur->i>=0) ans[cur->i]++;
        for(Node *tmp=cur->dic;tmp;tmp=tmp->dic)
            ans[tmp->i]++;
    } // ans[i] : number of occurrence of pattern i
} AC;

```

## 6.6 Z Value

```

char s[MAXN];
int len,z[MAXN];
void Z_value() { //z[i] = lcp(s[1...],s[i...])
    int i,j,left,right;
    left=right=0; z[0]=len;
    for(i=1;i<len;i++){
        j=max(min(z[i-left],right-i),0);
        for(;i+j<len&&s[i+j]==s[j];j++);
        z[i]=j;
        if(i+z[i]>right) {
            right=i+z[i];
            left=i;
        }
    }
}

```

## 6.7 ZValue Palindrome

```

void z_value_pal(char *s,int len,int *z){
    len=(len<<1)+1;
    for(int i=len-1;i>=0;i--){
        s[i]=i&1?s[i>1]:'@';
        z[0]=1;
        for(int i=1,l=0,r=0;i<len;i++){
            z[i]=i<r?min(z[l+1-i],r-i):1;
            while(i-z[i]>=0&&i+z[i]<len&&s[i-z[i]]==s[i+z[i]])
                ++z[i];
            if(i+z[i]>r) l=i,r=i+z[i];
        }
    }
}

```

## 6.8 Smallest Rotation

```

//rotate(begin(s),begin(s)+minRotation(s),end(s))
int minRotation(string s) {
    int a = 0, N = s.size(); s += s;
    rep(b,0,N) rep(k,0,N) {
        if(a+k == b || s[a+k] < s[b+k])
            {b += max(0, k-1); break;}
        if(s[a+k] > s[b+k]) {a = b; break;}
    } return a;
}

```

## 6.9 Cyclic LCS

```

#define L 0
#define LU 1
#define U 2
const int mov[3][2]={0,-1, -1,-1, -1,0};
int al,bl;
char a[MAXL*2],b[MAXL*2]; // 0-indexed
int dp[MAXL*2][MAXL];
char pred[MAXL*2][MAXL];
inline int lcs_length(int r) {
    int i=r+al,j=bl,l=0;
    while(i>r) {
        char dir=pred[i][j];
    }
}

```

```

    if(dir==LU) l++;
    i+=mov[dir][0];
    j+=mov[dir][1];
}
return l;
}
inline void reroot(int r) { // r = new base row
    int i=r,j=1;
    while(j<=bl&&pred[i][j]!=LU) j++;
    if(j>bl) return;
    pred[i][j]=L;
    while(i<2*al&&j<=bl) {
        if(pred[i+1][j]==U) {
            i++;
            pred[i][j]=L;
        } else if(j<bl&&pred[i+1][j+1]==LU) {
            i++;
            j++;
            pred[i][j]=L;
        } else {
            j++;
        }
    }
}
int cyclic_lcs() {
    // a, b, al, bl should be properly filled
    // note: a WILL be altered in process
    // -- concatenated after itself
    char tmp[MAXL];
    if(al>bl) {
        swap(al,bl);
        strcpy(tmp,a);
        strcpy(a,b);
        strcpy(b,tmp);
    }
    strcpy(tmp,a);
    strcat(a,tmp);
    // basic lcs
    for(int i=0;i<=2*al;i++) {
        dp[i][0]=0;
        pred[i][0]=U;
    }
    for(int j=0;j<=bl;j++) {
        dp[0][j]=0;
        pred[0][j]=L;
    }
    for(int i=1;i<=2*al;i++) {
        for(int j=1;j<=bl;j++) {
            if(a[i-1]==b[j-1]) dp[i][j]=dp[i-1][j-1]+1;
            else dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
            if(dp[i][j-1]==dp[i][j]) pred[i][j]=L;
            else if(a[i-1]==b[j-1]) pred[i][j]=LU;
            else pred[i][j]=U;
        }
    }
    // do cyclic lcs
    int clcs=0;
    for(int i=0;i<al;i++) {
        clcs=max(clcs,lcs_length(i));
        reroot(i+1);
    }
    // recover a
    a[al]='\0';
    return clcs;
}

```

## 6.10 Rolling Hash

```

const ll P1 = 75577;
const ll P2 = 12721; // 多一個質數 p2
const ll MOD = 998244353;
pair<ll,ll> h1[100005];
void build(const string& s){
    pair<ll,ll> val = make_pair(0,0);
    h1[0] = {0, 0};
    for(int i=0; i<s.size(); i++){
        val.first = (val.first * P1 + s[i]) % MOD;
        val.second = (val.second * P2 + s[i]) % MOD;
        h1[i + 1] = val;
    }
}

```

## 7 Data Structure

### 7.1 Binary Index Tree

```

#define lowbit(x) (x & -x)
int bit[MXN + 1], n;
void update(int x,int v){
    while(x<=n){
        bit[x] += v;
        x += lowbit(x);
    }
}
int query(int x){
    int ret=0;
    while(x){ // 當不為 0 時
        ret += bit[x]; // 回傳值加上 BIT[x]
        x -= lowbit(x); // 每次減掉自己的 Lowbit
    }
    return ret;
}

```

### 7.2 Segment Tree

```

struct seg_tree{
    ll a[MXN],val[MXN*4],tag[MXN*4],NO_TAG=0;
    void push(int i,int l,int r){
        if(tag[i]!=NO_TAG){
            val[i]+=tag[i]; // update by tag
            if(l==r){
                tag[cl(i)]+=tag[i]; // push
                tag[cr(i)]+=tag[i]; // push
            }
            tag[i]=NO_TAG;
        }
    }
    void pull(int i,int l,int r){
        int mid=(l+r)>>1;
        push(cl(i),l,mid);push(cr(i),mid+1,r);
        val[i]=max(val[cl(i)],val[cr(i)]); // pull
    }
    void build(int i,int l,int r){
        if(l==r){
            val[i]=a[l]; // set value
            return;
        }
        int mid=(l+r)>>1;
        build(cl(i),l,mid);build(cr(i),mid+1,r);
        pull(i,l,r);
    }
    void update(int i,int l,int r,int ql,int qr,int v){
        push(i,l,r);
        if(ql<=l&&r<=qr){
            tag[i]+=v; // update tag
            return;
        }
        int mid=(l+r)>>1;
        if(ql<=mid) update(cl(i),l,mid,ql,qr,v);
        if(qr>mid) update(cr(i),mid+1,r,ql,qr,v);
        pull(i,l,r);
    }
    ll query(int i,int l,int r,int ql,int qr){
        push(i,l,r);
        if(ql<=l&&r<=qr)
            return val[i]; // update answer
        int mid=(l+r)>>1,ret=0;
        if(ql<=mid) ret=max(ret,query(cl(i),l,mid,ql,qr));
        if(qr>mid) ret=max(ret,query(cr(i),mid+1,r,ql,qr));
        return ret;
    }
}tree;

```

### 7.3 Treap

```

struct Treap{
    int sz, val, pri, tag;
    Treap *l, *r;
    Treap(int _val){
        val = _val; sz = 1;
        pri = rand(); l = r = NULL; tag = 0;
    }
};
void push(Treap *a){
    if(a->tag){
        Treap *swp = a->l; a->l = a->r; a->r = swp;
        int swp2;
        if(a->l) a->l->tag ^= 1;
        if(a->r) a->r->tag ^= 1;
        a->tag = 0;
    }
}

```



```

} }
inline int Size( Treap * a ){ return a ? a->sz : 0; }
void pull( Treap * a ){
    a->sz = Size( a->l ) + Size( a->r ) + 1;
}
Treap* merge( Treap *a , Treap *b ){
    if( !a || !b ) return a ? a : b;
    if( a->pri > b->pri ){
        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_kth( Treap *t , int k, Treap*&a, Treap*&b ){
    if( !t ){ a = b = NULL; return; }
    push( t );
    if( Size( t->l ) + 1 <= k ){
        a = t;
        split_kth( t->r , k - Size( t->l ) - 1 , a->r , b );
        pull( a );
    }else{
        b = t;
        split_kth( t->l , k , a , b->l );
        pull( b );
    }
}
void split_key( Treap *t, int k, Treap*&a, Treap*&b ){
    if(!t){ a = b = NULL; return; }
    push(t);
    if(k<=t->val){
        b = t;
        split_key(t->l,k,a,b->l);
        pull(b);
    }
    else{
        a = t;
        split_key(t->r,k,a->r,b);
        pull(a);
    }
}
} }

```

## 7.4 Link-Cut Tree

```

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 (int _val=-1) : 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 ) return;
        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);
    }
}
int id(Splay *x) { return x - Splay::mem + 1; }
Splay* access(Splay *x){
    Splay *q = nil;
    for (;x!=nil;x=x->f){
        splay(x);
        x->setCh(q, 1);
        q = x;
    }
    return q;
}
void chroot(Splay *x){
    access(x);
    splay(x);
    x->rev ^= 1;
    x->push(); x->pull();
}
void link(Splay *x, Splay *y){
    access(x);
    splay(x);
    chroot(y);
    x->setCh(y, 1);
}
void cut_p(Splay *y) {
    access(y);
    splay(y);
    y->push();
    y->ch[0] = y->ch[0]->f = nil;
}
void cut(Splay *x, Splay *y){
    chroot(x);
    cut_p(y);
}
Splay* get_root(Splay *x) {
    access(x);
    splay(x);
    for(; x->ch[0] != nil; x = x->ch[0])
        x->push();
    splay(x);
    return x;
}
bool conn(Splay *x, Splay *y) {
    x = get_root(x);
    y = get_root(y);
    return x == y;
}
Splay* lca(Splay *x, Splay *y) {
    access(x);
    access(y);
    splay(x);
    if (x->f == nil) return x;
    else return x->f;
}

```

## 7.5 Disjoint Set

```

struct DisjointSet {
    int fa[MXN], h[MXN], top;
    struct Node {
        int x, y, fa, h;

```

```

Node(int _x = 0, int _y = 0, int _fa = 0, int _h = 0)
: x(_x), y(_y), fa(_fa), h(_h) {}
} stk[MXN];
void init(int n) {
    top = 0;
    for (int i = 1; i <= n; i++) fa[i] = i, h[i] = 0;
}
int find(int x) { return x == fa[x] ? x : find(fa[x]); }
void merge(int u, int v) {
    int x = find(u), y = find(v);
    if (h[x] > h[y]) swap(x, y);
    stk[top++] = Node(x, y, fa[x], h[y]);
    if (h[x] == h[y]) h[y]++;
    fa[x] = y;
}
void undo(int k=1) { //undo k times
    for (int i = 0; i < k; i++) {
        Node &it = stk[--top];
        fa[it.x] = it.fa;
        h[it.y] = it.h;
    }
}
} } djs;

```

## 7.6 Trie

```

struct trie{
    trie *nxt[26];
    int cnt,sz;
    trie():cnt(0),sz(0){
        memset(nxt,0,sizeof(nxt));
    }
};
trie *root = new trie();
void insert(string& s){
    trie *now = root; // 每次從根結點出發
    for(auto i:s){
        now->sz++;
        if(now->nxt[i-'a'] == NULL){
            now->nxt[i-'a'] = new trie();
        }
        now = now->nxt[i-'a']; //走到下一個字母
    }
    now->cnt++;
    now->sz++;
}
int query_prefix(string& s){ //查詢有多少前綴為 s
    trie *now = root; // 每次從根結點出發
    for(auto i:s){
        if(now->nxt[i-'a'] == NULL) return 0;
        now = now->nxt[i-'a'];
    }
    return now->sz;
}
int query_count(string& s){ //查詢字串 s 出現次數
    trie *now = root; // 每次從根結點出發
    for(auto i:s){
        if(now->nxt[i-'a'] == NULL) return 0;
        now = now->nxt[i-'a'];
    }
    return now->cnt;
}

```

## 7.7 Presistent Segment Tree

```

struct Node{
    Node *l, *r;
    int val;
    Node(int _val):val(_val){l = r = nullptr;}
};
void pull(Node *x){x->val = x->r->val + x->l->val;}
void build(Node *x, int l, int r){
    x = new Node(0);
    if(l == r){
        x->val = arr[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(x->l, l, mid);
    build(x->r, mid + 1, r);
    pull(x);
}

```

```

Node *update(Node *&pre, int l, int r, char &c, int &p,
int &v){
    Node *x = new Node(0);
    if(l == r){
        if(c == '+') x->val = pre->val + v;
        else x->val = pre->val - v;
        return x;
    }
    int mid = (l + r) >> 1;
    if(p <= mid){
        x->l = update(pre->l, l, mid, c, p, v);
        x->r = pre->r;
    }
    else{
        x->l = pre->l;
        x->r = update(pre->r, mid + 1, r, c, p, v);
    }
    pull(x);
    return x;
}
11 query(Node *x, int l, int r, int ql, int qr){
    if(ql <= l && r <= qr){
        return x->val;
    }
    int mid = (l + r) >> 1, ret = 0;
    if(ql <= mid) ret += query(x->l, l, mid, ql, qr);
    if(mid < qr) ret += query(x->r, mid + 1, r, ql, qr);
    return ret;
}
vector<Node *> vsn;

```

## 7.8 SQRT-Decomposition

```

struct blk{
    vector<int> local;
    int global, tag;
    blk(){
        local.clear();
        tag = global = 0;
    }
};
int len;
blk arr[MXN] = {};
void build(int &n){
    len = sqrt(n);
    int x;
    for(int i = 0; i < n; ++i){
        cin >> x;
        arr[i/len].local.pb(x);
        arr[i/len].global += x;
    }
}
void update(int ql, int qr, int v){
    int l = ql/len, r = qr/len;
    if(l == r){
        for(int i = ql; i <= qr; ++i){
            arr[l].local[i % len] += v;
            arr[l].global += v;
        }
        return;
    }
    for(int i = ql; i < (l + 1) * len; ++i){
        arr[l].local[i % len] += v;
        arr[l].global += v;
    }
    for(int i = l + 1; i < r; ++i){
        arr[i].global += (v * len);
        arr[i].tag += v;
    }
    for(int i = r * len; i <= qr; ++i){
        arr[r].local[i % len] += v;
        arr[r].global += v;
    }
}
int query(int ql, int qr){
    int l = ql/len, r = qr/len, ret = 0;
    if(l == r){
        for(int i = ql; i <= qr; ++i)
            ret += arr[l].local[i % len] + arr[l].tag;
        return ret;
    }
}

```

```

for(int i = ql; i < (l + 1) * len; ++i)
    ret += arr[l].local[i % len] + arr[l].tag;
for(int i = l + 1; i < r; ++i)
    ret += arr[i].global;
for(int i = r * len; i <= qr; ++i)
    ret += arr[r].local[i % len] + arr[r].tag;
return ret;
}

```

## 8 DP

### 8.1 SOS DP

```

for(int i = 0; i < (1 << N); ++i) F[i] = A[i];
for(int i = 0; i < N; ++i) for(int mask = 0; mask < (1 << N); ++mask){
    if(mask & (1 << i))
        F[mask] += F[mask ^ (1 << i)];
}

```

### 8.2 Subset Sum

```

bitset<MXN> dp;
int n, x;
for(int i = 0; i < MXN; ++i){
    cin >> x;
    dp <= dp | x;
}

```

### 8.3 knapsack

```

struct Data{int cost, w;};
int dp[MXN] = {};
Data arr[MXN] = {};
int n, c, w, sum = 0;
cin >> n;
for(int i = 0; i < n; ++i){
    cin >> arr[i].cost >> arr[i].w;
    sum += arr[i].w;
}
dp[0] = 0;
for(int i = 0; i < n; ++i){
    for(int j = sum; j >= arr[i].w; --j)
        dp[j] = max(dp[j], dp[j - arr[i].w]);
}

```

### 8.4 Knapsack Unlimited

```

int cost[N], weight[N];
int c[W + 1];
void knapsack(int n, int w){
    for (int i=0; i<n; ++i)
        for (int j = weight[i]; j <= w; ++j)
            c[j] = max(c[j], c[j - weight[i]] + cost[i]);
}

```

### 8.5 Knapsack Limited

```

dp[0] = 1;
for(int i = 1; i <= MXN; ++i){
    int tmp = i * w[i], now = i;
    while(tmp){
        tmp -= now;
        for(int j = now; j <= sum; ++j)
            dp[j] |= dp[j - now];
        now <= 1;
        if(now > tmp) now = tmp;
    }
}

```

### 8.6 DP on DAG

```

int dp[MXN];
bool v[MXN];
void dfs(int x){
    if(v[x]) return dp[x];
    v[x] = 1; dp[x] = 0;
    for(auto i:edge[x])
        dp[x] = max(dp[x], dfs(i) + 1);
    return dp[x];
}

```

```

int main(){
    for(int i=1;i<=n;i++)
        if(!v[i]) dfs(i);
}

```

### 8.7 LIS

```

int LIS(vector<int>& s){
    if (s.size() == 0) return 0;
    vector<int> v;
    v.push_back(s[0]);
    for (int i=1; i<s.size(); ++i){
        int n = s[i];
        if (n > v.back())v.push_back(n);
        else *lower_bound(v.begin(), v.end(), n) = n;
    }
    return v.size();
}

```

### 8.8 Matrix Fast Power

```

#define MOD 1'000'000'007
#define ll long long
vector<vector<ll>> operator*(const vector<vector<ll>>&
    lhs,const vector<vector<ll>>& rhs){
    vector<vector<ll>> ret(lhs.size(),vector<ll>(rhs
        [0].size(),0));
    for(int i=0;i<lhs.size();i++){
        for(int j=0;j<rhs[0].size();j++){
            for(int k=0;k<rhs.size();k++){
                ret[i][j] += lhs[i][k] * rhs[k][j] %
                    MOD;
                ret[i][j] %= MOD;
            } } }
    return ret;
}
vector<vector<ll>> init_value={{1},{0}}; //第0,1項
vector<vector<ll>> base={{1,1},{1,0}}; //費式數列轉移式
vector<vector<ll>> matrix={{1,0},{0,1}}; //單位矩陣
while(y){ // x^y
    if(y&1) matrix = matrix * base;
    base = base * base;
    y >>= 1;
}
matrix = matrix * init_value;
cout<< matrix[0][0] << endl;

```

## 9 Others

### 9.1 Mo's Algorithm

```

int len = sqrt(n);
struct query{
    int l,r,id; //詢問的左界右界 以及 第幾筆詢問
    friend bool operator<(const query& lhs,const query&
        rhs){
        return ((lhs.l / len) == (rhs.l / len)) ? lhs.r
            < rhs.r : lhs.l < rhs.l;
    } //先判斷是不是在同一塊 不同塊的話就比較塊的順序 ·
        否則比較右界r
};
int ans[200005] = {}, t = 0;
vector<query> q;
void add(int idx){...}
void sub(int idx){...}
void mos(){
    sort(all(q));
    for(int i = 0, l = -2, r = -1; i < q.size(); ++i){
        while(l > q[i].l) add(--l);
        while(r < q[i].r) add(++r); //先做新增元素的
        while(l < q[i].l) sub(l++); //再做移除元素的
        while(r > q[i].r) sub(r--);
        ans[q[i].id] = t; //移到區間後儲存答案
    }
}

```

### 9.2 Reverse Pair

```

int ans;
vector<int> arr;
vector<int> temparr;

```

```
void msort(int s,int t) {
    if(s==t) return ;
    int mid=(s+t)>>1;
    msort(s,mid),msort(mid+1,t);
    int i=s,j=mid+1,k=s;
    while(i<=mid && j<=t) {
        if(arr[i]<=arr[j]) temparr[k]=arr[i],k++,i++;
        else temparr[k]=arr[j],k++,j++,ans+=mid-i+1;
    }
    while(i<=mid) temparr[k]=arr[i],k++,i++;
    while(j<=t) temparr[k]=arr[j],k++,j++;
    for(int i=s;i<=t;i++) arr[i]=temparr[i];
    return ;
}
```