

Contents

1 Basic	1	7 Data Structure	24
1.1 default code	1	7.1 Link-Cut Tree	24
1.2 .vimrc	1	7.2 Black Magic	24
1.3 Increase Stack Size (linux)	1	8 Others	25
1.4 Misc	1	8.1 SOS dp	25
1.5 check	2	8.2 Number of Occurrences of Digit	25
1.6 python-related	2	8.3 Find max tangent(x,y is increasing)	25
2 flow	2	8.4 Exact Cover Set	25
2.1 ISAP	2	8.5 Hilbert Curve	25
2.2 MinCostFlow	2	8.6 De Bruijn sequence	25
2.3 Dinic	2	1 Basic	1
2.4 Kuhn Munkres 最大完美二分匹配	3	1.1 default code	1
2.5 Directed MST	3	#ifdef LOCAL // ===== Local =====	
2.6 SW min-cut (不限 S-T 的 min-cut)	3	void dbg() { cerr << '\n'; }	
2.7 Max flow with lower/upper bound	4	template<class T, class ...U> void dbg(T a, U ...b) {	
2.8 HLPPA (稠密圖 flow)	4	cerr << a << ' ', dbg(b...); }	
2.9 Flow Method	4	template<class T> void org(T l, T r) { while (l != r)	
3 Math	5	cerr << *l++ << ' '; cerr << '\n'; }	
3.1 FFT	5	#define debug(args...) (dbg("#> (" + string(#args) + ")	
3.2 NTT	5	= (" , args, ")")	
3.3 Fast Walsh Transform	5	#define orange(args...) (cerr << "#> [" + string(#args)	
3.4 Poly operator	6	+ ") = " , org(args))	
3.5 O(1)mul	6	#else // ===== OnlineJudge =====	
3.6 Linear Recurrence	6	#pragma GCC optimize("O3,unroll-loops")	
3.7 Miller Rabin	6	#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")	
3.8 Faulhaber ($\sum_{i=1}^n i^p$)	7	#define debug(...) ((void)0)	
3.9 Chinese Remainder	7	#define orange(...) ((void)0)	
3.10 Pollard Rho	7	#endif	
3.11 Josephus Problem	7	template<class T> bool chmin(T &a, T b) { return b < a	
3.12 ax+by=gcd	7	and (a = b, true); }	
3.13 Discrete sqrt	7	template<class T> bool chmax(T &a, T b) { return b > a	
3.14 Romberg 定積分	8	and (a = b, true); }	
3.15 Prefix Inverse	8	1.2 .vimrc	
3.16 Roots of Polynomial 找多項式的根	8	set nu rnu ts=4 sw=4 bs=2 ai hls cin mouse=a	
3.17 Primes	8	color default	
3.18 Phi	8	sy on	
3.19 Result	8	inoremap {<CR> {<CR>}<C-o>0	
4 Geometry	9	inoremap jk <Esc>	
4.1 definition	9	nnoremap J 5j	
4.2 Intersection of 2 lines	9	nnoremap K 5k	
4.3 halfPlaneIntersection	9	nnoremap run :w<bar>!g++ -std=c++14 -DLOCAL -Wfatal-	
4.4 Convex Hull	9	errors -o test "%>" && echo "done." && time ./test<	
4.5 Convex Hull 3D	9	CR>	
4.6 Intersection of 2 segments	10	1.3 Increase Stack Size (linux)	
4.7 Intersection of circle and segment	10	#include <sys/resource.h>	
4.8 Intersection of polygon and circle	10	void increase_stack_size() {	
4.9 Point In Polygon	10	const rlim_t ks = 64*1024*1024;	
4.10 Intersection of 2 circles	11	struct rlimit rl;	
4.11 Circle cover	11	int res=getrlimit(RLIMIT_STACK, &rl);	
4.12 Convex Hull trick	11	if(res==0){	
4.13 Tangent line of two circles	12	if(rl.rlim_cur<ks){	
4.14 Minimum distance of two convex	12	rl.rlim_cur=ks;	
4.15 KD Tree	12	res=setrlimit(RLIMIT_STACK, &rl);	
4.16 Poly Union	12	} }	
4.17 Lower Concave Hull	12	1.4 Misc	
4.18 Min Enclosing Circle	13	編譯參數: -std=c++14 -Wall -Wshadow (-fsanitize=	
4.19 Min Enclosing Ball	13	undefined)	
4.20 Minkowski sum	14	mt19937 gen(chrono::steady_clock::now().	
4.21 Li Chao Segment Tree	14	time_since_epoch().count());	
4.22 Area of Rectangles	14	int randint(int lb, int ub)	
4.23 Min dist on Cuboid	15	{ return uniform_int_distribution<int>(lb, ub)(gen); }	
4.24 Heart of Triangle	15	#define SECS ((double)clock() / CLOCKS_PER_SEC)	
5 Graph	15	struct KeyHasher {	
5.1 DominatorTree	15	size_t operator()(const Key& k) const {	
5.2 Maximum Clique 最大團	15	return k.first + k.second * 100000;	
5.3 Maximal Clique 極大團	16	} };	
5.4 Strongly Connected Component	16	typedef unordered_map<Key,int,KeyHasher> map_t;	
5.5 Dynamic MST	16	__builtin_popcountll // 二進位有幾個1	
5.6 Maximum General graph Matching	17	__builtin_clzll // 左起第一個1之前0的個數	
5.7 Minimum General Weighted Matching	17	__builtin_parityll // 1的個數的奇偶性	
5.8 Maximum General Weighted Matching	17	__builtin_mul_overflow(a,b,&h) // a*b是否溢位	
5.9 Minimum Steiner Tree	19		
5.10 BCC based on vertex	19		
5.11 Min Mean Cycle	19		
5.12 Directed Graph Min Cost Cycle	20		
5.13 K-th Shortest Path	20		
5.14 SPFA	21		
5.15 差分約束	21		
6 String	21		
6.1 PalTree	21		
6.2 KMP	21		
6.3 SAIS	22		
6.4 SuffixAutomata	22		
6.5 Z Value	22		
6.6 BWT	23		
6.7 ZValue Palindrome	23		
6.8 Smallest Rotation	23		
6.9 Cyclic LCS	23		

1.5 check

```
for ((i=0;;i++))
do
    echo "$i"
    python3 gen.py > input
    ./ac < input > ac.out
    ./wa < input > wa.out
    diff ac.out wa.out || break
done
```

1.6 python-related

```
parser:
int(eval(num.replace("/", "///")))

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

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

format(x, '0.10f') # set precision

N = 200
def angle(cosT):
    """given cos(theta) in decimal return theta"""
    for i in range(N):
        cosT = ((cosT + 1) / two) ** itwo
        sinT = (1 - cosT * cosT) ** itwo
        return sinT * (2 ** N)
pi = angle(Decimal(-1))
```

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;
    } } }flow;
```

2.2 MinCostFlow

```
struct zkwflow{
    static const int maxN=10000;
    struct Edge{ int v,f,re; ll w;};
    int n,s,t,ptr[maxN]; bool vis[maxN]; ll dis[maxN];
    vector<Edge> E[maxN];
    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 addEdge(int u,int v,int f,ll w){
        E[u].push_back({v,f,(int)E[v].size(),w});
        E[v].push_back({u,0,(int)E[u].size()-1,-w});
    }
    bool SPFA(){
        fill_n(dis,n,LLONG_MAX); fill_n(vis,n,false);
        queue<int> q; q.push(s); dis[s]=0;
        while (!q.empty()){
            int u=q.front(); q.pop(); vis[u]=false;
            for(auto &it:E[u]){
                if(it.f>0&&dis[it.v]>dis[u]+it.w){
                    dis[it.v]=dis[u]+it.w;
                    if(!vis[it.v]){
                        vis[it.v]=true; q.push(it.v);
                    }
                }
            }
        }
        return dis[t]!=LLONG_MAX;
    }
    int DFS(int u,int nf){
        if(u==t) return nf;
        int res=0; vis[u]=true;
        for(int &i=ptr[u];i<(int)E[u].size();i++){
            auto &it=E[u][i];
            if(it.f>0&&dis[it.v]==dis[u]+it.w&&!vis[it.v]){
                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){ vis[u]=false; break; }
            }
        }
        return res;
    }
    pair<int,ll> flow(){
        int flow=0; ll cost=0;
        while (SPFA()){
            fill_n(ptr,n,0);
            int f=DFS(s,INT_MAX);
            flow+=f; cost+=dis[t]*f;
        }
        return{ flow,cost };
    } // reset: do nothing
} flow;
```

2.3 Dinic

```
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 最大完美二分匹配

```

struct KM{ // max weight, for min negate the weights
    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) { // 1-based
        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)
    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個點的數量, m個邊的數量, 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 )
        return -1; // no solution
    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 HLPPA (稠密圖 flow)

```

template <int MAXN, class T = int>
struct HLPP {
    const T INF = numeric_limits<T>::max();
    struct Edge {
        int to, rev; T f;
    };
    int n, s, t;
    vector<Edge> adj[MAXN];
    deque<int> lst[MAXN];
    vector<int> gap[MAXN];
    int ptr[MAXN];
    T ef[MAXN];
    int h[MAXN], cnt[MAXN], work, hst=0/*highest*/;
    void init(int _n, int _s, int _t) {
        n=_n+1; s=_s; t=_t;
    }

```

```

    for(int i=0;i<n;i++) adj[i].clear();
}
void addEdge(int u,int v,T f,bool isDir = true){
    adj[u].push_back({v,adj[v].size(),f});
    adj[v].push_back({u,adj[u].size()-1,isDir?f:0});
}
void updHeight(int v, int nh) {
    work++;
    if(h[v] != n) cnt[h[v]]--;
    h[v] = nh;
    if(nh == n) return;
    cnt[nh]++, hst = nh; gap[nh].push_back(v);
    if(ef[v]>0) lst[nh].push_back(v), ptr[nh]++;
}
void globalRelabel() {
    work = 0;
    fill(h, h+n, n);
    fill(cnt, cnt+n, 0);
    for(int i=0; i<=hst; i++)
        lst[i].clear(), gap[i].clear(), ptr[i] = 0;
    queue<int> q({t}); h[t] = 0;
    while(!q.empty()) {
        int v = q.front(); q.pop();
        for(auto &e : adj[v])
            if(h[e.to] == n && adj[e.to][e.rev].f > 0)
                q.push(e.to), updHeight(e.to, h[v] + 1);
        hst = h[v];
    }
}
void push(int v, Edge &e) {
    if(ef[e.to] == 0)
        lst[h[e.to]].push_back(e.to), ptr[h[e.to]]++;
    T df = min(ef[v], e.f);
    e.f -= df, adj[e.to][e.rev].f += df;
    ef[v] -= df, ef[e.to] += df;
}
void discharge(int v) {
    int nh = n;
    for(auto &e : adj[v]) {
        if(e.f > 0) {
            if(h[v] == h[e.to] + 1) {
                push(v, e);
                if(ef[v] <= 0) return;
            }
            else nh = min(nh, h[e.to] + 1);
        }
    }
    if(cnt[h[v]] > 1) updHeight(v, nh);
    else {
        for(int i = h[v]; i < n; i++) {
            for(auto j : gap[i]) updHeight(j, n);
            gap[i].clear(), ptr[i] = 0;
        }
    }
}
T solve() {
    fill(ef, ef+n, 0);
    ef[s] = INF, ef[t] = -INF;
    globalRelabel();
    for(auto &e : adj[s]) push(s, e);
    for(; hst >= 0; hst--) {
        while(!lst[hst].empty()) {
            int v=lst[hst].back(); lst[hst].pop_back();
            discharge(v);
            if(work > 4 * n) globalRelabel();
        }
    }
    return ef[t] + INF;
} }

```

2.9 Flow Method

Maximize $c^T x$ subject to $Ax \leq b, x \geq 0$;
with the corresponding symmetric dual problem,
Minimize $b^T y$ subject to $A^T y \geq c, y \geq 0$.

Maximize $c^T x$ subject to $Ax \leq b$;
with the corresponding asymmetric dual problem,
Minimize $b^T y$ subject to $A^T y = c, y \geq 0$.

Minimum vertex cover on bipartite graph =
Maximum matching on bipartite graph

Minimum edge cover on bipartite graph =
vertex number - Minimum vertex cover (Maximum matching)

Independent set on bipartite graph =

vertex number - Minimum vertex cover(Maximum matching)

找出最小點覆蓋，做完dinic之後，從源點dfs只走還有流量的邊，紀錄每個點有被走到，左邊被走到的點跟右邊被走到的點就是答案

Maximum density subgraph ($\sum W_e + \sum W_v$) / |V|

Binary search on answer:

For a fixed D, construct a Max flow model as follow:

Let S be Sum of all weight(or inf)

1. from source to each node with cap = S

2. For each (u,v,w) in E, (u->v,cap=w), (v->u,cap=w)

3. For each node v, from v to sink with cap = S + 2 * D - deg[v] - 2 * (W of v)

where deg[v] = \sum weight of edge associated with v

If maxflow < S * |V|, D is an answer.

Requiring subgraph: all vertex can be reached from source with edge whose cap > 0.

3 Math

3.1 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);
}
```

3.2 NTT

```
// Remember coefficient are mod P
/* p=a*2^n+1
n    2^n    p    a    root
16   65536   65537   1    3
20   1048576 7340033   7    3 */
// (must be 2^k)
template<LL P, LL root, int MAXN>
struct NTT{
    static LL bigmod(LL a, LL b) {
        LL res = 1;
        for (LL bs = a; b >= 1; bs = (bs * bs) % P)
            if(b&1) res=(res*bs)%P;
        return res;
    }
    static LL inv(LL a, LL b) {
        if(a==1)return 1;
        return (((LL)(a-inv(b*a,a))*b+1)/a)%b;
    }
    LL omega[MAXN+1];
    NTT() {
        omega[0] = 1;
        LL r = bigmod(root, (P-1)/MAXN);
        for (int i=1; i<=MAXN; i++)
            omega[i] = (omega[i-1]*r)%P;
    }
    // n must be 2^k
    void tran(int n, LL a[], bool inv_ntt=false){
        int basic = MAXN / n, theta = basic;
        for (int m = n; m >= 2; m >= 1) {
            int mh = m >> 1;
            for (int i = 0; i < mh; i++) {
                LL w = omega[i*theta%MAXN];
                for (int j = i; j < n; j += m) {
                    int k = j + mh;
                    LL x = a[j] - a[k];
                    if (x < 0) x += P;
                    a[j] += a[k];
                    if (a[j] > P) a[j] -= P;
                    a[k] = (w * x) % P;
                }
            }
            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_ntt) {
            LL ni = inv(n,P);
            reverse(a+1, a+n);
            for (i = 0; i < n; i++)
                a[i] = (a[i] * ni) % P;
        }
    }
};
const LL P=2013265921, root=31;
const int MAXN=4194304;
NTT<P, root, MAXN> ntt;
```

3.3 Fast Walsh Transform

```
/* xor convolution:
* x = (x0,x1), y = (y0,y1)
* z = ( x0y0 + x1y1, x0y1 + x1y0 )
* =>
* x' = ( x0+x1, x0-x1 ), y' = ( y0+y1, y0-y1 )
* z' = ( ( x0+x1 )( y0+y1 ), ( x0-x1 )( y0-y1 ) )
* z = (1/2) * z'
* or convolution:
* x = (x0, x0+x1), inv = (x0, x1-x0) w/o final div
* and convolution:
* x = (x0+x1, x1), inv = (x0-x1, x1) w/o final div */
const int MAXN = (1<<20)+10;
inline LL inv( LL x ) {
    return mypow( x, MOD-2 );
}
inline void fwt( LL x[ MAXN ], int N, bool inv=0 ) {
    for( int d = 1; d < N; d <= 1 ) {
        int d2 = d<<1;
```



```

for( int s = 0 ; s < N ; s += d2 )
    for( int i = s , j = s+d ; i < s+d ; i++ , j++ ){
        LL ta = x[ i ] , tb = x[ j ];
        x[ i ] = ta+tb;
        x[ j ] = ta-tb;
        if( x[ i ] >= MOD ) x[ i ] -= MOD;
        if( x[ j ] < 0 ) x[ j ] += MOD;
    }
if( inv )
    for( int i = 0 ; i < N ; i++ ) {
        x[ i ] *= inv( N );
        x[ i ] %= MOD;
    }
}

```

3.4 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] (國法卷積)
    // (轉國成加法國積) -> 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;

```

3.5 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;
}

```

3.6 Linear Recurrence

```

// Usage: linearRec({0, 1}, {1, 1}, k) //k'th fib
typedef vector<ll> Poly;
//S: 前i項的值, tr: 遞迴係數, k: 求第k項
ll linearRec(Poly& S, Poly& tr, ll k) {
    int n = tr.size();
    auto combine = [&](Poly& a, Poly& b) {
        Poly res(n * 2 + 1);
        rep(i, 0, n+1) rep(j, 0, n+1)
            res[i+j] = (res[i+j] + a[i]*b[j])%mod;
        for(int i = 2*n; i > n; --i) rep(j, 0, n)
            res[i-1-j] = (res[i-1-j] + res[i]*tr[j])%mod;
        res.resize(n + 1);
        return res;
    };
    Poly pol(n + 1, e(pol));
    pol[0] = e[1] = 1;
    for (++k; k; k /= 2) {
        if (k % 2) pol = combine(pol, e);
        e = combine(e, e);
    }
    ll res = 0;
    rep(i, 0, n) res = (res + pol[i+1]*S[i])%mod;
    return res;
}

```

3.7 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 : pimes <= 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 = mypow(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;
}

```

3.8 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]; // combinatorics
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;
}

```

3.9 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;
}

```

3.10 Pollard Rho

```

// does not work when n is prime O(n^(1/4))
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;
    }
}

```

3.11 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;
}

```

3.12 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)};
}

```

3.13 Discrete sqrt

```

void calch(LL &t, LL &h, const LL p) {
    LL tmp=p-1; for(t=0;(tmp&1)==0;tmp/=2) t++; h=tmp;
}
// solve equation x^2 mod p = a
bool solve(LL a, LL p, LL &x, LL &y) {
    if(p == 2) { x = y = 1; return true; }
    int p2 = p / 2, tmp = mypow(a, p2, p);
    if (tmp == p - 1) return false;
    if ((p + 1) % 4 == 0) {
        x=mypow(a,(p+1)/4,p); y=p-x; return true;
    } else {
        LL t, h, b, pb; calch(t, h, p);
        if (t >= 2) {
            do {b = rand() % (p - 2) + 2;
                while (mypow(b, p / 2, p) != p - 1);
                pb = mypow(b, h, p);
            } int s = mypow(a, h / 2, p);
            for (int step = 2; step <= t; step++) {
                int ss = (((LL)(s * s) % p) * a) % p;
                for(int i=0;i<t-step;i++) ss=mul(ss,ss,p);
                if (ss + 1 == p) s = (s * pb) % p;
                pb = ((LL)pb * pb) % p;
            } x = ((LL)s * a) % p; y = p - x;
        } return true;
    }
}

```

3.14 Romberg 定積分

```
// Estimates the definite integral of
// \int_a^b f(x) dx
template<class T>
double romberg( T& f, double a, double b, double eps=1e-8){
    vector<double>t; double h=b-a,last,curr; int k=1,i=1;
    t.push_back(h*(f(a)+f(b))/2);
    do{ last=t.back(); curr=0; double x=a+h/2;
        for(int j=0;j<k;j++) curr+=f(x), x+=h;
        curr=(t[0] + h*curr)/2; double k1=4.0/3.0,k2
            =1.0/3.0;
        for(int j=0;j<i;j++){ double temp=k1*curr-k2*t[j];
            t[j]=curr; curr=temp; k2/=4*k1-k2; k1=k2+1;
        } t.push_back(curr); k*=2; h/=2; i++;
    }while( fabs(last-curr) > eps);
    return t.back();
}
```

3.15 Prefix Inverse

```
void solve( int m ){
    inv[ 1 ] = 1;
    for( int i = 2 ; i < m ; i ++ )
        inv[ i ] = ((LL)(m - m / i) * inv[m % i]) % m;
}
```

3.16 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]
```

3.17 Primes

```
/* 12721, 13331, 14341, 75577, 123457, 222557, 556679
* 999983, 1097774749, 1076767633, 100102021, 999997771
* 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;
}
```

3.18 Phi

```
ll phi(ll n){ // 計算小於n的數中與n互質的有幾個
    ll res = n, a=n; // 0(sqrtN)
    for(ll i=2;i*i<=a;i++){
        if(a%i==0){
            res = res/i*(i-1);
            while(a%i==0) a/=i;
        }
    }
    if(a>1) res = res/a*(a-1);
    return res;
}
```

3.19 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):
 $S(n, k) = \text{coefficient of } x^k \text{ in } \Pi_{i=0}^{n-1} (x+i)$
- 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 \square 部格點數目 i 、邊上格點數目 b 的關 \square
- 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 \square 方法數, m \square 總數):
$$\left(\sum_{i=1}^m c^{g^{cd(i,m)}} \right) / m$$
- Burnside lemma:
$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$$

- 錯排公式: (n 個人中, 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:

$$A^B \pmod{p} = \text{pow}(A, \text{pow}(B, C, p - 1)) \pmod{p}$$

- 歐拉函數降冪公式:

$$A^B \pmod{C} = A^{B \pmod{\phi(C)} + \phi(C)} \pmod{C}$$

- 6 的倍數:

$$(a - 1)^3 + (a + 1)^3 + (-a)^3 + (-a)^3 = 6a$$

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;
} // segment should add Counterclockwise
// 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 Convex Hull 3D

```
struct Pt{
    Pt cross(const Pt &p) const
    { return Pt(y * p.z - z * p.y, z * p.x - x * p.z, x *
        p.y - y * p.x); }
} info[N];
int mark[N][N], n, cnt;
```

```

double mix(const Pt &a, const Pt &b, const Pt &c)
{ return a * (b ^ c); }
double area(int a, int b, int c)
{ return norm((info[b] - info[a]) ^ (info[c] - info[a])) / 2; }
double volume(int a, int b, int c, int d)
{ return mix(info[b] - info[a], info[c] - info[a], info[d] - info[a]); }
struct Face{
    int a, b, c; Face(){}
    Face(int a, int b, int c): a(a), b(b), c(c) {}
    int &operator [](int k)
    { if (k == 0) return a; if (k == 1) return b; return c; }
};
vector<Face> face;
void insert(int a, int b, int c)
{ face.push_back(Face(a, b, c)); }
void add(int v) {
    vector<Face> tmp; int a, b, c; cnt++;
    for (int i = 0; i < SIZE(face); i++) {
        a = face[i][0]; b = face[i][1]; c = face[i][2];
        if (Sign(volume(v, a, b, c)) < 0)
            mark[a][b] = mark[b][a] = mark[b][c] = mark[c][b] =
                mark[c][a] = mark[a][c] = cnt;
        else tmp.push_back(face[i]);
    } face = tmp;
    for (int i = 0; i < SIZE(tmp); i++) {
        a = face[i][0]; b = face[i][1]; c = face[i][2];
        if (mark[a][b] == cnt) insert(b, a, v);
        if (mark[b][c] == cnt) insert(c, b, v);
        if (mark[c][a] == cnt) insert(a, c, v);
    }
}
int Find(){
    for (int i = 2; i < n; i++) {
        Pt ndir = (info[0] - info[i]) ^ (info[1] - info[i]);
        if (ndir == Pt()) continue; swap(info[i], info[2]);
        for (int j = i + 1; j < n; j++) if (Sign(volume(0, 1, 2, j)) != 0) {
            swap(info[j], info[3]); insert(0, 1, 2); insert(0, 2, 1); return 1;
        }
    } return 0; }
int main() {
    for (; scanf("%d", &n) == 1; ) {
        for (int i = 0; i < n; i++) info[i].Input();
        sort(info, info + n); n = unique(info, info + n) - info;
        face.clear(); random_shuffle(info, info + n);
        if (Find()) { memset(mark, 0, sizeof(mark)); cnt = 0;
            for (int i = 3; i < n; i++) add(i); vector<Pt> Ndir;
            for (int i = 0; i < SIZE(face); i++) {
                Pt p = (info[face[i][0]] - info[face[i][1]]) ^
                    (info[face[i][2]] - info[face[i][1]]);
                p = p / norm(p); Ndir.push_back(p);
            } sort(Ndir.begin(), Ndir.end());
            int ans = unique(Ndir.begin(), Ndir.end()) - Ndir.begin();
            printf("%d\n", ans);
        } else printf("1\n");
    }
}
double calcDist(const Pt &p, int a, int b, int c)
{ return fabs(mix(info[a] - p, info[b] - p, info[c] - p)) / area(a, b, c); }
//compute the minimal distance of center of any faces
double findDist() { //compute center of mass
    double totalWeight = 0; Pt center(.0, .0, .0);
    Pt first = info[face[0][0]];
    for (int i = 0; i < SIZE(face); i++) {
        Pt p = (info[face[i][0]] + info[face[i][1]] + info[face[i][2]] + first) * .25;
        double weight = mix(info[face[i][0]] - first, info[face[i][1]] - first, info[face[i][2]] - first);
        totalWeight += weight; center = center + p * weight;
    }
    center = center / totalWeight;
    double res = 1e100; //compute distance
    for (int i = 0; i < SIZE(face); i++)

```

```

        res = min(res, calcDist(center, face[i][0], face[i][1], face[i][2]));
    }
    return res; }

```

4.6 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) > 0) 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);
}

```

4.7 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 || bb4ac < 0);
}

```

4.8 Intersection of polygon and circle

```

ld PCIntersect(vector<Pt> v, Circle cir) {
    for (int i = 0; i < (int)v.size(); i++) v[i] = v[i] - cir.o;
    ld ans = 0, r = cir.r;
    int n = v.size();
    for (int i = 0; i < n; i++) {
        Pt pa = v[i], pb = v[(i+1)%n];
        if (norm(pa) < norm(pb)) swap(pa, pb);
        if (dcmp(norm(pb)) == 0) continue;
        ld s, h, theta;
        ld a = norm(pb), b = norm(pa), c = norm(pb-pa);
        ld cosB = (pb*(pb-pa))/a/c, B = acos(cosB);
        if (cosB > 1) B = 0;
        else if (cosB < -1) B = PI;
        ld cosC = (pa*pb)/a/b, C = acos(cosC);
        if (cosC > 1) C = 0;
        else if (cosC < -1) C = PI;
        if (a > r) {
            s = (C/2)*r*r;
            h = a*b*sin(C)/c;
            if (h < r && B < PI/2) s -= (acos(h/r)*r*r - h*sqrt(r*r-h*h));
        }
        else if (b > r) {
            theta = PI - B - asin(sin(B)/r*a);
            s = 0.5*a*r*sin(theta) + (C-theta)/2*r*r;
        }
        else s = 0.5*sin(C)*a*b;
        ans += abs(s)*dcmp(v[i]^v[(i+1)%n]);
    }
    return abs(ans);
}

```

4.9 Point In Polygon

```

int ptInPoly(vector<Pt> ps, Pt p) {
    int c=0;
    for (int i=0; i<ps.size(); i++) {
        int a=i, b=(i+1)%ps.size(); Line l(ps[a], ps[b]);
        Pt q=l.s+l.v*((l.v*(p-l.s))/norm2(l.v)); // project
        if (norm(p-q)<eps&&onseg(q,l)) return 1; // boundary
        if (dcmp(ps[a].y-ps[b].y)==0&&dcmp(ps[a].y-p.y)==0) continue;
        if (ps[a].y>ps[b].y) swap(a,b);
    }
}

```

```

    if(ps[a].y<=p.y&&ps[a].y<ps[b].y&&p.x<=ps[a].x+(ps[b].x
      -ps[a].x)/(ps[b].y-ps[a].y)*(p.y-ps[a].y)) ++c;
  }
  return (c&1)*2; // 0: outside, 1: boundary, 2: inside
} // check whether a point is in a polygon

```

4.10 Intersection of 2 circles

4.11 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 Cinter( 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 false;
    if( norm( o1 - o2 ) < max(r1, r2) - min(r1, r2) )
      return true;
    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 disjunct( 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] ||
          disjunct(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;
          Cinter(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;
      }
    }
  }
}

```

4.12 Convex Hull trick

```

/* Given a convexhull, 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;
}
};
```

4.13 Tangent line of two circles

```
vector<Line> go( const Cir& c1 , const Cir& c2 , int
    sign1 ){
    // sign1 = 1 for outer tang, -1 for inter 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;
}
```

4.14 Minimum distance of two convex

```
double TwoConvexHullMinDis(Pt P[], Pt Q[], int n, int m){
    int mn=0, mx=0; double tmp, ans=1e9;
    for(int i=0; i<n; ++i) if(P[i].y<P[mn].y) mn=i;
    for(int i=0; i<m; ++i) if(Q[i].y>Q[mx].y) mx=i;
    P[n]=P[0]; Q[m]=Q[0];
    for (int i=0; i<n; ++i) {
        while(tmp=((Q[mx+1]-P[mn+1])^(P[mn]-P[mn+1]))>((Q[
            mx]-P[mn+1])^(P[mn]-P[mn+1]))) mx=(mx+1)%m;
        if(tmp<0) // pt to segment distance
            ans=min(ans, dis(Line(P[mn], P[mn+1]), Q[mx]));
        else // segment to segment distance
            ans=min(ans, dis(Line(P[mn], P[mn+1]), Line(Q[mx], Q[
                mx+1])));
        mn=(mn+1)%n;
    }
    return ans;
}
```

```
}
}
```

4.15 KD Tree

```
struct KDTree{ // O(sqrtN + K)
    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;
```

4.16 Poly Union

```
struct PY{
    int n; Pt pt[5]; double area;
    Pt& operator[](const int x){ return pt[x]; }
```

```

void init(){ //n,pt[0~n-1] must be filled
    area=pt[n-1]^pt[0];
    for(int i=0;i<n-1;i++) area+=pt[i]^pt[i+1];
    if((area/=2)<0)reverse(pt,pt+n),area=-area;
} };
PY py[500]; pair<double,int> c[5000];
inline double segP(Pt &p,Pt &p1,Pt &p2){
    if(dcmp(p1.x-p2.x)==0) return (p.y-p1.y)/(p2.y-p1.y);
    return (p.x-p1.x)/(p2.x-p1.x);
}
double polyUnion(int n){ //py[0~n-1] must be filled
    int i,j,ii,jj,ta,tb,r,d; double z,w,s,sum=0,tc,td;
    for(i=0;i<n;i++) py[i][py[i].n]=py[i][0];
    for(i=0;i<n;i++){
        for(ii=0;ii<py[i].n;ii++){
            r=0;
            c[r++]=make_pair(0.0,0); c[r++]=make_pair(1.0,0);
            for(j=0;j<n;j++){
                if(i==j) continue;
                for(jj=0;jj<py[j].n;jj++){
                    ta=dcmp(tri(py[i][ii],py[i][ii+1],py[j][jj])));
                    ;
                    tb=dcmp(tri(py[i][ii],py[i][ii+1],py[j][jj+1]));
                    if(ta==0 && tb==0){
                        if((py[j][jj+1]-py[j][jj])*(py[i][ii+1]-py[i][ii])>0&&j<i){
                            c[r++]=make_pair(segP(py[j][jj],py[i][ii],py[i][ii+1]),1);
                            c[r++]=make_pair(segP(py[j][jj+1],py[i][ii],py[i][ii+1]),-1);
                        }
                    } else if(ta>0 && tb<0){
                        tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
                        td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
                        c[r++]=make_pair(tc/(tc-td),1);
                    } else if(ta<0 && tb>0){
                        tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
                        td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
                        c[r++]=make_pair(tc/(tc-td),-1);
                    }
                }
            }
            sort(c,c+r);
            z=min(max(c[0].first,0.0),1.0); d=c[0].second; s=0;
            for(j=1;j<r;j++){
                w=min(max(c[j].first,0.0),1.0);
                if(!d) s+=w-z;
                d+=c[j].second; z=w;
            }
            sum+=(py[i][ii]^py[i][ii+1])*s;
        }
    }
    return sum/2;
}

```

4.17 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;
}
}

```

4.18 Min Enclosing Circle

```

struct Mec{ // return pair of center and r
    int n;
    Pt p[ MXN ], 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;

```

4.19 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);
}
}

```

4.20 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);
}

```

4.21 Li Chao Segment Tree

```

struct LiChao_min{
    struct line{
        ll m,c;
        line(ll _m=0,ll _c=0){ m=_m; c=_c; }
        ll eval(ll x){ return m*x+c; } // overflow
    };
    struct node{
        node *l,*r; line f;
        node(line v){ f=v; l=r=NULL; }
    };
    typedef node* pnode;
    pnode root; ll sz,ql,qr;
#define mid ((l+r)>>1)
    void insert(line v,ll l,ll r,pnode &nd){
        /* if(ql<=l&&r<=qr){
            if(!nd) nd=new node(line(0,INF));
            if(ql<=mid) insert(v,l,mid,nd->l);
            if(qr>mid) insert(v,mid+1,r,nd->r);
        }
        */

```

```

        return;
    } used for adding segment */
    if(!nd){ nd=new node(v); return; }
    ll trl=nd->f.eval(l),trr=nd->f.eval(r);
    ll vl=v.eval(l),vr=v.eval(r);
    if(trl<=vl&&trr<=vr) return;
    if(trl>vl&&trr>vr) { nd->f=v; return; }
    if(trl>vl) swap(nd->f,v);
    if(nd->f.eval(mid)<v.eval(mid))
        insert(v,mid+1,r,nd->r);
    else swap(nd->f,v),insert(v,l,mid,nd->l);
}
ll query(ll x,ll l,ll r,pnode &nd){
    if(!nd) return INF;
    if(l==r) return nd->f.eval(x);
    if(mid>=x)
        return min(nd->f.eval(x),query(x,l,mid,nd->l));
    return min(nd->f.eval(x),query(x,mid+1,r,nd->r));
}
/* -sz<=ll query_x<=sz */
void init(ll _sz){ sz=_sz+1; root=NULL; }
void add_line(ll m,ll c,ll l=-INF,ll r=INF){
    line v(m,c); ql=l; qr=r; insert(v,-sz,sz,root);
}
ll query(ll x) { return query(x,-sz,sz,root); }
};

```

4.22 Area of Rectangles

```

struct AreaofRectangles{
#define cl(x) (x<<1)
#define cr(x) (x<<1|1)
    ll n, id, sid;
    pair<ll,ll> tree[MXN<<3]; // count, area
    vector<ll> ind;
    tuple<ll,ll,ll,ll> scan[MXN<<1];
    void pull(int i, int l, int r){
        if(tree[i].first) tree[i].second = ind[r+1] -
            ind[l];
        else if(l != r){
            int mid = (l+r)>>1;
            tree[i].second = tree[cl(i)].second + tree[
                cr(i)].second;
        }
        else tree[i].second = 0;
    }
    void upd(int i, int l, int r, int ql, int qr, int v)
    ){
        if(ql <= l && r <= qr){
            tree[i].first += v;
            pull(i, l, r); return;
        }
        int mid = (l+r) >> 1;
        if(ql <= mid) upd(cl(i), l, mid, ql, qr, v);
        if(qr > mid) upd(cr(i), mid+1, r, ql, qr, v);
        pull(i, l, r);
    }
    void init(int _n){
        n = _n; id = sid = 0;
        ind.clear(); ind.resize(n<<1);
        fill(tree, tree+(n<<2), make_pair(0, 0));
    }
    void addRectangle(int lx, int ly, int rx, int ry){
        ind[id++] = lx; ind[id++] = rx;
        scan[sid++] = make_tuple(ly, 1, lx, rx);
        scan[sid++] = make_tuple(ry, -1, lx, rx);
    }
    ll solve(){
        sort(ind.begin(), ind.end());
        ind.resize(unique(ind.begin(), ind.end()) - ind
            .begin());
        sort(scan, scan + sid);
        ll area = 0, pre = get<0>(scan[0]);
        for(int i = 0; i < sid; i++){
            auto [x, v, l, r] = scan[i];
            area += tree[1].second * (x-pre);
            upd(1, 0, ind.size()-1, lower_bound(ind.
                begin(), ind.end(), l)-ind.begin(),
                lower_bound(ind.begin(), ind.end(), r)-
                ind.begin()-1, v);
            pre = x;
        }
    }
}

```

```
    return area;
} }rect;
```

4.23 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;
}
```

4.24 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 othroCenter( 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*ba.X-ba.X*ca.Y*c.X) / A,
           y0 = -ba.X * (x0 - c.X) / ba.Y + ca.Y;
    return Pt(x0, y0);
}
```

5 Graph

5.1 DominatorTree

```
struct DominatorTree{ // O(N)
#define REP(i,s,e) for(int i=(s);i<=(e);i++)
#define REPD(i,s,e) for(int i=(s);i>=(e);i--)
    int n, m, s;
    vector< int > g[ MAXN ], pred[ MAXN ];
    vector< int > cov[ MAXN ];
    int dfn[ MAXN ], nfd[ MAXN ], ts;
    int par[ MAXN ]; //idom[u] s到u的最後一個必經點
    int sdom[ MAXN ], idom[ MAXN ];
    int mom[ MAXN ], mn[ MAXN ];
    inline bool cmp( int u, int v )
    { return dfn[ u ] < dfn[ v ]; }
    int eval( int u ){
        if( mom[ u ] == u ) return u;
        int res = eval( mom[ u ] );
        if(cmp( sdom[ mn[ mom[ u ] ] ], sdom[ mn[ u ] ] ))
            mn[ u ] = mn[ mom[ u ] ];
        return mom[ u ] = res;
    }
    void init( int _n, int _m, int _s ){
        ts = 0; n = _n; m = _m; s = _s;
        REP( i, 1, n ) g[ i ].clear(), pred[ i ].clear();
    }
    void addEdge( int u, int v ){
        g[ u ].push_back( v );
        pred[ v ].push_back( u );
    }
}
```

```
void dfs( int u ){
    ts++;
    dfn[ u ] = ts;
    nfd[ ts ] = u;
    for( int v : g[ u ] ) if( dfn[ v ] == 0 ){
        par[ v ] = u;
        dfs( v );
    }
}
void build(){
    REP( i, 1, n ){
        dfn[ i ] = nfd[ i ] = 0;
        cov[ i ].clear();
        mom[ i ] = mn[ i ] = sdom[ i ] = i;
    }
    dfs( s );
    REPD( i, n, 2 ){
        int u = nfd[ i ];
        if( u == 0 ) continue;
        for( int v : pred[ u ] ) if( dfn[ v ] ){
            eval( v );
            if( cmp( sdom[ mn[ v ] ], sdom[ u ] ) )
                sdom[ u ] = sdom[ mn[ v ] ];
        }
        cov[ sdom[ u ] ].push_back( u );
        mom[ u ] = par[ u ];
        for( int w : cov[ par[ u ] ] ){
            eval( w );
            if( cmp( sdom[ mn[ w ] ], par[ u ] ) )
                idom[ w ] = mn[ w ];
            else idom[ w ] = par[ u ];
        }
        cov[ par[ u ] ].clear();
    }
    REP( i, 2, n ){
        int u = nfd[ i ];
        if( u == 0 ) continue;
        if( idom[ u ] != sdom[ u ] )
            idom[ u ] = idom[ idom[ u ] ];
    }
} domT;
```

5.2 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;

```

5.3 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];
    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 = (candilex)._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]);
        }
    }
    void 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;

```

5.4 Strongly Connected Component

```

struct Scc{
    int n, nScc, vst[MXN], bln[MXN];
    vector<int> E[MXN], rE[MXN], vec;
    void init(int _n){
        n = _n;
        for(int i=0; i<MXN; i++){
            E[i].clear(), rE[i].clear();
        }
    }
    void addEdge(int u, int v){
        E[u].PB(v); rE[v].PB(u);
    }
    void DFS(int u){
        vst[u]=1;
        for(auto v : E[u]) if(!vst[v]) DFS(v);
        vec.PB(u);
    }
}

```

```

}
void rDFS(int u){
    vst[u] = 1; bln[u] = nScc;
    for(auto v : rE[u]) if(!vst[v]) rDFS(v);
}
void solve(){
    nScc = 0;
    vec.clear();
    FZ(vst);
    for(int i=0; i<n; i++){
        if(!vst[i]) DFS(i);
    }
    reverse(vec.begin(), vec.end());
    FZ(vst);
    for(auto v : vec){
        if(!vst[v]){
            rDFS(v); nScc++;
        }
    }
}
};

```

5.5 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);
        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]
    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); }

```

5.6 Maximum General graph Matching

```

// should shuffle vertices and edges
const int N=100005,E=(2e5)*2+40;
struct Graph{ // 1-based; match: i <-> lnk[i]
    int to[E],bro[E],head[N],e,lnk[N],vis[N],stp,n;
    void init(int _n){
        stp=0; e=1; n=_n;
        for(int i=1;i<=n;i++) head[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; }
        }
        for(int i=head[x];i;i=bro[i]){
            int v=to[i];
            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;

```

5.7 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;

```

5.8 Maximum General Weighted Matching

```

struct WeightGraph {
    static const int INF = INT_MAX;
    static const int N = 514;
    struct edge{
        int u,v,w; edge(){}}
    edge(int ui,int vi,int wi)
        :u(ui),v(vi),w(wi){}
};
int n,n_x;
edge g[N*2][N*2];
int lab[N*2];
int match[N*2],slack[N*2],st[N*2],pa[N*2];
int flo_from[N*2][N+1],S[N*2],vis[N*2];
vector<int> flo[N*2];
queue<int> q;
int e_delta(const edge &e){
    return lab[e.u]+lab[e.v]-g[e.u][e.v].w*2;
}
void update_slack(int u,int x){
    if(!slack[x]||e_delta(g[u][x])<e_delta(g[slack[x]]
        [x]))slack[x]=u;
}
void set_slack(int x){
    slack[x]=0;
    for(int u=1;u<=n;u++)
        if(g[u][x].w>0&&st[u]!=x&&S[st[u]]==0)
            update_slack(u,x);
}
void q_push(int x){
    if(x<=n)q.push(x);
    else for(size_t i=0;i<flo[x].size();i++)
        q_push(flo[x][i]);
}
void set_st(int x,int b){
    st[x]=b;

```

```

    if(x>n)for(size_t i=0;i<flo[x].size();++i)
        set_st(flo[x][i],b);
}
int get_pr(int b,int xr){
    int pr=find(flo[b].begin(),flo[b].end(),xr)-flo[b].begin();
    if(pr%2==1){
        reverse(flo[b].begin()+1,flo[b].end());
        return (int)flo[b].size()-pr;
    }else return pr;
}
void set_match(int u,int v){
    match[u]=g[u][v].v;
    if(u<n) return;
    edge e=g[u][v];
    int xr=flo_from[u][e.u],pr=get_pr(u,xr);
    for(int i=0;i<pr;++i)set_match(flo[u][i],flo[u][i^1]);
    set_match(xr,v);
    rotate(flo[u].begin(),flo[u].begin()+pr,flo[u].end());
}
void augment(int u,int v){
    for(;;){
        int xnv=st[match[u]];
        set_match(u,v);
        if(!xnv)return;
        set_match(xnv,st[pa[xnv]]);
        u=st[pa[xnv]],v=xnv;
    }
}
int get_lca(int u,int v){
    static int t=0;
    for(++t;u!=v;swap(u,v)){
        if(u==0)continue;
        if(vis[u]==t)return u;
        vis[u]=t;
        u=st[match[u]];
        if(u)u=st[pa[u]];
    }
    return 0;
}
void add_blossom(int u,int lca,int v){
    int b=n+1;
    while(b<=n_x&&st[b])++b;
    if(b>n_x)++n_x;
    lab[b]=0,S[b]=0;
    match[b]=match[lca];
    flo[b].clear();
    flo[b].push_back(lca);
    for(int x=u,y; x!=lca; x=st[pa[y]])
        flo[b].push_back(x),flo[b].push_back(y=st[match[x]]),q_push(y);
    reverse(flo[b].begin()+1,flo[b].end());
    for(int x=v,y; x!=lca; x=st[pa[y]])
        flo[b].push_back(x),flo[b].push_back(y=st[match[x]]),q_push(y);
    set_st(b,b);
    for(int x=1;x<=n_x;++x)g[b][x].w=g[x][b].w=0;
    for(int x=1;x<=n;++x)flo_from[b][x]=0;
    for(size_t i=0;i<flo[b].size();++i){
        int xs=flo[b][i];
        for(int x=1;x<=n_x;++x)
            if(g[b][x].w==0||e_delta(g[xs][x])<e_delta(g[b][x]))
                g[b][x]=g[xs][x],g[x][b]=g[x][xs];
        for(int x=1;x<=n;++x)
            if(flo_from[xs][x])flo_from[b][x]=xs;
    }
    set_slack(b);
}
void expand_blossom(int b){
    for(size_t i=0;i<flo[b].size();++i)
        set_st(flo[b][i],flo[b][i]);
    int xr=flo_from[b][g[b][pa[b]].u],pr=get_pr(b,xr);
    for(int i=0;i<pr;i+=2){
        int xs=flo[b][i],xns=flo[b][i+1];
        pa[xs]=g[xns][xs].u;
        S[xs]=1,S[xns]=0;
        slack[xs]=0,set_slack(xns);
        q_push(xns);
    }
    S[xr]=1,pa[xr]=pa[b];

```

```

    for(size_t i=pr+1;i<flo[b].size();++i){
        int xs=flo[b][i];
        S[xs]=-1,set_slack(xs);
    }
    st[b]=0;
}
bool on_found_edge(const edge &e){
    int u=st[e.u],v=st[e.v];
    if(S[v]==-1){
        pa[v]=e.u,S[v]=1;
        int nu=st[match[v]];
        slack[v]=slack[nu]=0;
        S[nu]=0,q_push(nu);
    }else if(S[v]==0){
        int lca=get_lca(u,v);
        if(!lca)return augment(u,v),augment(v,u),true;
        else add_blossom(u,lca,v);
    }
    return false;
}
bool matching(){
    memset(S+1,-1,sizeof(int)*n_x);
    memset(slack+1,0,sizeof(int)*n_x);
    q=queue<int>();
    for(int x=1;x<=n_x;++x)
        if(st[x]==x&&!match[x])pa[x]=0,S[x]=0,q_push(x);
    if(q.empty())return false;
    for(;;){
        while(q.size()){
            int u=q.front();q.pop();
            if(S[st[u]]==1)continue;
            for(int v=1;v<=n;++v)
                if(g[u][v].w>0&&st[u]!=st[v]){
                    if(e_delta(g[u][v])==0){
                        if(on_found_edge(g[u][v]))return true;
                    }else update_slack(u,st[v]);
                }
        }
        int d=INF;
        for(int b=n+1;b<=n_x;++b)
            if(st[b]==b&&S[b]==1)d=min(d,lab[b]/2);
        for(int x=1;x<=n_x;++x)
            if(st[x]==x&&slack[x]){
                if(S[x]==-1)d=min(d,e_delta(g[slack[x]][x]));
                else if(S[x]==0)d=min(d,e_delta(g[slack[x]][x]/2));
            }
        for(int u=1;u<=n;++u){
            if(S[st[u]]==0){
                if(lab[u]<=d)return 0;
                lab[u]-=d;
            }else if(S[st[u]]==1)lab[u]+=d;
        }
        for(int b=n+1;b<=n_x;++b)
            if(st[b]==b){
                if(S[st[b]]==0)lab[b]+=d*2;
                else if(S[st[b]]==1)lab[b]-=d*2;
            }
        q=queue<int>();
        for(int x=1;x<=n_x;++x)
            if(st[x]==x&&slack[x]&&st[slack[x]]!=x&&e_delta(g[slack[x]][x])==0)
                if(on_found_edge(g[slack[x]][x]))return true;
        for(int b=n+1;b<=n_x;++b)
            if(st[b]==b&&S[b]==1&&lab[b]==0)expand_blossom(b);
    }
    return false;
}
pair<long long,int> solve(){
    memset(match+1,0,sizeof(int)*n);
    n_x=n;
    int n_matches=0;
    long long tot_weight=0;
    for(int u=0;u<=n;++u)st[u]=u,flo[u].clear();
    int w_max=0;
    for(int u=1;u<=n;++u)
        for(int v=1;v<=n;++v){
            flo_from[u][v]=(u==v?u:0);
            w_max=max(w_max,g[u][v].w);
        }
    for(int u=1;u<=n;++u)lab[u]=w_max;
    while(matching())++n_matches;

```



```

    for(int u=1;u<=n;++u)
        if(match[u]&&match[u]<u)
            tot_weight+=g[u][match[u]].w;
    return make_pair(tot_weight,n_matches);
}
void add_edge( int ui , int vi , int wi ){
    g[ui][vi].w = g[vi][ui].w = wi;
}
void init( int _n ){
    n = _n;
    for(int u=1;u<=n;++u)
        for(int v=1;v<=n;++v)
            g[u][v]=edge(u,v,0);
}
} graph;

```

5.9 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(){ // using spfa may faster
            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 ] );
        }
        // call shorest_path before solve
        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;
}

```

5.10 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;

```

5.11 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;

```

5.12 Directed Graph Min Cost Cycle

```

// works in O(N M)
#define INF 1000000000000000LL
#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;

```

5.13 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(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
}
}

```

5.14 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;
}

```

5.15 差分約束

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

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?l-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=nxt[np][c])){
            lst=newNode(len[np]+2, nxt[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 $\textcircled{\text{E}}$ 下，這個子字串可以由開頭
長度 $\textcircled{\text{E}}$ (len-failure[k])的部分重 $\textcircled{\text{E}}$ 出現來表達

failure[k] $\textcircled{\text{E}}$ 次長相同前綴後綴

如果我們不只想求最多，而且以0-base做 $\textcircled{\text{E}}$ 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 $\textcircled{\text{E}}$ 0 $\textcircled{\text{E}}$ 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 SAIS

```

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]-lpl+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 Z Value

```

int z[MAXN];
void Z_value(const string& s) { //z[i] = lcp(s[1...], s[i...])
    int i, j, left, right, len = s.size();
    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;
        }
    }
}

```

6.6 BWT

```

struct BurrowsWheeler{
#define SIGMA 26
#define BASE 'a'
    vector<int> v[ SIGMA ];
    void BWT(char* ori, char* res){
        // make ori -> ori + ori
        // then build suffix array
    }
    void iBWT(char* ori, char* res){
        for( int i = 0 ; i < SIGMA ; i ++ )
            v[ i ].clear();
        int len = strlen( ori );
        for( int i = 0 ; i < len ; i ++ )
            v[ ori[i] - BASE ].push_back( i );
        vector<int> a;
        for( int i = 0 , ptr = 0 ; i < SIGMA ; i ++ )
            for( auto j : v[ i ] ){
                a.push_back( j );
                ori[ ptr ++ ] = BASE + i;
            }
        for( int i = 0 , ptr = 0 ; i < len ; i ++ ){
            res[ i ] = ori[ a[ ptr ] ];
            ptr = a[ ptr ];
        }
        res[ len ] = 0;
    }
}
bwt;

```

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]:'0';
        z[0]=1;
    }
}

```

```

for(int i=1, l=0, r=0; i<len; i++){
    z[i]=i<r?min(z[l+l-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;
}

```

7 Data Structure

7.1 Link-Cut Tree

```

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;
}
}

```

7.2 Black Magic

```

#include <bits/extc++.h>
using namespace __gnu_pbds;
typedef tree<int,null_type,less<int>,rb_tree_tag,
    tree_order_statistics_node_update> set_t;
#include <ext/pb_ds/assoc_container.hpp>
typedef cc_hash_table<int,int> umap_t;
typedef priority_queue<int> heap;
#include<ext/rope>
using namespace __gnu_cxx;
int main(){
    // Insert some entries into s.
    set_t s; s.insert(12); s.insert(505);
    // The order of the keys should be: 12, 505.
    assert(*s.find_by_order(0) == 12);
    assert(*s.find_by_order(3) == 505);
    // The order of the keys should be: 12, 505.
    assert(s.order_of_key(12) == 0);
    assert(s.order_of_key(505) == 1);
    // Erase an entry.
    s.erase(12);
    // The order of the keys should be: 505.
    assert(*s.find_by_order(0) == 505);
    // The order of the keys should be: 505.
    assert(s.order_of_key(505) == 0);

    heap h1 , h2; h1.join( h2 );

    rope<char> r[ 2 ];
    r[ 1 ] = r[ 0 ]; // persistenet
    string t = "abc";
    r[ 1 ].insert( 0 , t.c_str() );
    r[ 1 ].erase( 1 , 1 );
    cout << r[ 1 ].substr( 0 , 2 );
}

```

8 Others

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 Number of Occurrences of Digit

```
int dp[MAXN][MAXN], a[MAXN];
int dfs(int pos, bool leadZero, bool bound, int sum, int digit) {
    if (!pos) return sum;
    if (!leadZero && !bound && dp[pos][sum] != -1)
        return dp[pos][sum];
    int top = bound ? a[pos] : 9, ans = 0;
    for (int i = 0; i <= top; ++i)
        ans += dfs(pos - 1, (i || !leadZero), bound && i == a[pos], sum + ((i == digit) && (i || !leadZero)), digit);
    if (!leadZero && !bound) dp[pos][sum] = ans;
    return ans;
}
int pre(int r, int digit) { //return num of digit in [1, r]
    int cnt = 0;
    memset(dp, -1, sizeof dp);
    while (r != 0)
        a[++cnt] = r % 10, r /= 10;
    return dfs(cnt, 1, 1, 0, digit);
}
```

8.3 Find max tangent(x,y is increasing)

```
const int MAXN = 100010;
Pt sum[MAXN], pnt[MAXN], ans, calc;
inline bool cross(Pt a, Pt b, Pt c){
    return (c.y-a.y)*(c.x-b.x) > (c.x-a.x)*(c.y-b.y);
} //pt[0]=(0,0); pt[i]=(i, pt[i-1].y+dy[i-1]), i=1~n; dx>=1
double find_max_tan(int n, int l, LL dy[]){
    int np, st, ed, now;
    sum[0].x = sum[0].y = np = st = ed = 0;
    for (int i = 1, v; i <= n; i++)
        sum[i].x = i, sum[i].y = sum[i-1].y + dy[i-1];
    ans.x = now = 1, ans.y = -1;
    for (int i = 0; i <= n - 1; i++){
        while(np>1 && cross(pnt[np-2], pnt[np-1], sum[i]))
            np--;
        if (np < now && np != 0) now = np;
        pnt[np++] = sum[i];
        while(now<np && !cross(pnt[now-1], pnt[now], sum[i+1]))
            now++;
        calc = sum[i + 1] - pnt[now - 1];
        if (ans.y * calc.x < ans.x * calc.y)
            ans = calc, st = pnt[now - 1].x, ed = i + 1;
    }
    return (double)(sum[ed].y - sum[st].y) / (sum[ed].x - sum[st].x);
}
```

8.4 Exact Cover Set

```
// given n*m 0-1 matrix
// find a set of rows s.t.
// for each column, there's exactly one 1
#define N 1024 //row
#define M 1024 //column
#define NM ((N+2)*(M+2))
char A[N][M]; //n*m 0-1 matrix
int used[N]; //answer: the row used
int id[N][M];
int L[NM], R[NM], D[NM], U[NM], C[NM], S[NM], ROW[NM];
void remove(int c){
    L[R[c]] = L[c]; R[L[c]] = R[c];
    for( int i=D[c]; i!=c; i=D[i] )
        for( int j=R[i]; j!=i; j=R[j] ){
            U[D[j]] = U[j]; D[U[j]] = D[j]; S[C[j]]--;

```

```

        }
    }
    void resume(int c){
        for( int i=D[c]; i!=c; i=D[i] )
            for( int j=L[i]; j!=i; j=L[j] ){
                U[D[j]] = D[U[j]] = j; S[C[j]]++;
            }
        L[R[c]] = R[L[c]] = c;
    }
    int dfs(){
        if(R[0]==0) return 1;
        int md=100000000, c;
        for( int i=R[0]; i!=0; i=R[i] )
            if(S[i]<md){ md=S[i]; c=i; }
        if(md==0) return 0;
        remove(c);
        for( int i=D[c]; i!=c; i=D[i] ){
            used[ROW[i]] = 1;
            for( int j=R[i]; j!=i; j=R[j] ) remove(C[j]);
            if(dfs()) return 1;
            for( int j=L[i]; j!=i; j=L[j] ) resume(C[j]);
            used[ROW[i]] = 0;
        }
        resume(c);
        return 0;
    }
    int exact_cover(int n, int m){
        for( int i=0; i<=m; i++ ){
            R[i] = i+1; L[i] = i-1; U[i] = D[i] = i;
            S[i] = 0; C[i] = i;
        }
        R[m] = 0; L[0] = m;
        int t = m+1;
        for( int i=0; i<n; i++ ){
            int k = -1;
            for( int j=0; j<m; j++ ){
                if(!A[i][j]) continue;
                if(k == -1) L[t] = R[t] = t;
                else{ L[t] = k; R[t] = R[k]; }
                k = t; D[t] = j+1; U[t] = U[j+1];
                L[R[t]] = R[L[t]] = U[D[t]] = D[U[t]] = t;
                C[t] = j+1; S[C[t]]++; ROW[t] = i; id[i][j] = t++;
            }
        }
        for( int i=0; i<n; i++ ) used[i] = 0;
        return dfs();
    }
}
```

8.5 Hilbert Curve

```
long long hilbert(int n, int x, int y){
    long long res = 0;
    for(int s=n/2; s>=1; s--){
        int rx=(x&s)>0, ry=(y&s)>0; res+=s*1ll*s*((3*rx)^ry);
        if(ry==0){ if(rx==1) x=s-1-x, y=s-1-y; swap(x,y); }
    }
    return res;
}
```

8.6 De Bruijn sequence

```
// return cyclic array of length k^n such that every
// array of length n using 0~k-1 appears as a subarray.
vector<int> DeBruijn(int k, int n){
    if(k==1) return {0};
    vector<int> aux(k*n), res;
    function<void(int,int)> f = [&](int t, int p) -> void{
        if(t>n){ if(n%p==0)
            for(int i=1; i<=p; ++i) res.push_back(aux[i]);
        }
        else{
            aux[t] = aux[t-p]; f(t+1, p);
            for(aux[t] = aux[t-p]+1; aux[t]<k; ++aux[t]) f(t+1, t);
        }
    };
    f(1, 1); return res;
}
```