Contents

1 Basic

1.1 .vimrc

```
syn on
se ai nu rnu ru cul mouse=a
se cin et ts=2 sw=2 sts=2
so $VIMRUNTIME/mswin.vim
colo desert
filet plugin indent on
no <F5> :!./a.out<CR>
no <F9> :!g++ -O2 -std=c++17 % -g -fsanitize=undefined -
    Wall -Wextra -Wshadow -Wno-unused-result<CR>
se undofile undodir=~/.vim/undodir " mkdir manually
```

1.2 hash.sh

```
#!/bin/bash
cpp -dD -P -fpreprocessed $1 | tr -d '[:space:]'| md5sum
    |cut -c-6
```

1.3 Custom Hash

```
struct custom_hash {
  static uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
  }
  size_t operator()(uint64_t x) const {
    static const uint64_t FIXED_RANDOM = chrono::
        steady_clock::now().time_since_epoch().count();
    return splitmix64(x + FIXED_RANDOM);
  }
};
```

1.4 Fast Division/Modular

```
struct FastDiv{
  ull b,m;
  FastDiv(ull _b):b(_b),m(-1ULL/_b){}
  pair<ull,ull> div(ull a){ // (a/b,a%b)
    ull q=((__uint128_t(m)*a)>>64),r=a-q*b;
    return r>=b?make_pair(q+1,r-b):make_pair(q,r);
  }
};
```

1.5 python-related

```
from fractions import Fraction
from decimal import Decimal, getcontext
getcontext().prec = 250 # set precision
itwo,two,N = Decimal(0.5),Decimal(2),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

```
#define SZ(c) ((int)(c).size())
struct Maxflow{
  static const int MAXV=50010;
  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];
  int iter[MAXV],d[MAXV],gap[MAXV],tot;
  void init(int n,int _s,int _t){
    tot=n,s=_s,t=_t;
    for(int i=0;i<=tot;i++){
      G[i].clear(); iter[i]=d[i]=gap[i]=0;
    }
  }
  void add_edge(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 flow(){
    int res=0;
    for(res=0,gap[0]=tot;d[s]<tot;res+=DFS(s,INF));
    return res;
  }
} 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 add_edge(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 };
  }
} flow;
```

2.3 Dinic

```
struct Dinic{
  static const int MXN=10000;
  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].push_back({v,f,(int)E[v].size()});
    E[v].push_back({u,0,(int)E[u].size()-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

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

```
        if(t==0){
          pa[y]=x;
          if(!my[y]){ augment(y); return; }
          vy[y]=1,q.push(my[y]);
        }else if(sy[y]>t) pa[y]=x,sy[y]=t;
      }
    }
    ll cut=INF;
    for(int y=1;y<=n;++y)
      if(!vy[y]&&cut>sy[y]) cut=sy[y];
    for(int j=1;j<=n;++j){
      if(vx[j]) lx[j]-=cut;
      if(vy[j]) ly[j]+=cut;
      else sy[j]-=cut;
    }
    for(int y=1;y<=n;++y) if(!vy[y]&&sy[y]==0){
      if(!my[y]){ augment(y); return; }
      vy[y]=1,q.push(my[y]);
  } } }
  ll solve(){
    fill(mx,mx+n+1,0);fill(my,my+n+1,0);
    fill(ly,ly+n+1,0);fill(lx,lx+n+1,-INF);
    for(int x=1;x<=n;++x) for(int y=1;y<=n;++y)
      lx[x]=max(lx[x],g[x][y]);
    for(int x=1;x<=n;++x) bfs(x);
    ll ans=0;
    for(int y=1;y<=n;++y) ans+=g[my[y]][y];
    return ans;
  }
}graph;
```

2.5 SW min-cut

```
const int INF=0x3f3f3f3f;
template<typename T>
struct stoer_wagner{// 0-base
  static const int MAXN=501;
  T g[MAXN][MAXN],dis[MAXN]; int nd[MAXN],n,s,t;
  void init(int _n){
    n=_n;
    for(int i=0;i<n;++i) for(int j=0;j<n;++j )g[i][j]=0;
  }
  void add_edge(int u,int v,T w){ g[u][v]=g[v][u]+=w; }
  T min_cut(){
    T ans=INF;
    for(int i=0;i<n;++i) nd[i]=i;
    for(int ind,tn=n;tn>1;--tn){
      for(int i=1;i<tn;++i)dis[nd[i]]=0;
      for(int i=1;i<tn;++i){
        ind=i;
        for(int j=i;j<tn;++j){
          dis[nd[j]]+=g[nd[i-1]][nd[j]];
          if(dis[nd[ind]]<dis[nd[j]])ind=j;
        }
        swap(nd[ind],nd[i]);
      }
      if(ans>dis[nd[ind]])
        ans=dis[t=nd[ind]],s=nd[ind-1];
      for(int i=0;i<tn;++i)
        g[nd[ind-1]][nd[i]]=g[nd[i]][nd[ind-1]]
                          +=g[nd[i]][nd[ind]];
    }
    return ans;
  }
};
```

2.6 Max Cost Circulation

```
struct MaxCostCirc {
  static const int MAXN=33;
  struct Edge { int v,w,c,r; };
  vector<Edge> g[MAXN];
  int dis[MAXN],prv[MAXN],prve[MAXN];
  int n,m,ans; bool vis[MAXN];
  void init(int _n,int _m) : n(_n),m(_m) {}
  void adde(int u,int v,int w,int c) {
    g[u].push_back({v,w,c,(int)g[v].size()});
    g[v].push_back({u,-w,0,(int)g[u].size()-1});
  }
  bool poscyc() {
    fill(dis,dis+n+1,0); fill(prv,prv+n+1,0);
    fill(vis,vis+n+1,0); int tmp=-1;
    for(int t=0;t<=n;t++) {
      for(int i=1;i<=n;i++) {
        for(int j=0;j<(int)g[i].size();j++) {
          Edge& e=g[i][j];
          if(e.c&&dis[e.v]<dis[i]+e.w) {
            dis[e.v]=dis[i]+e.w;
            prv[e.v]=i; prve[e.v]=j;
            if(t==n) { tmp=i; break; }
          } } } }
    if(tmp==-1) return 0;
    int cur=tmp;
    while(!vis[cur]) { vis[cur]=1; cur=prv[cur]; }
    int now=cur,cost=0,df=100000;
    do{
      Edge &e=g[prv[now]][prve[now]];
      df=min(df,e.c); cost+=e.w; now=prv[now];
    }while(now!=cur);
    ans+=df*cost; now=cur;
    do{
      Edge &e=g[prv[now]][prve[now]];
      Edge &re=g[now][e.r];
      e.c-=df; re.c+=df; now=prv[now];
    }while(now!=cur);
    return 1;
  }
} circ;
```

2.7 Gomory-Hu Tree

```
//n,Dinic::flow must be filled
//result:e[u][v]=u-v mincut;p[u]:u's parent on cut tree
int n,e[MXN][MXN],p[MXN];
void gomory_hu(){
  fill(p,p+n,0); fill(e[0],e[n],INF);
  for(int s=1;s<n;s++){
    int t=p[s]; Dinic F; F.init(n,s,t);
    copy(flow.E,flow.E+MXN,F.E); int tmp=F.flow();
    for(int i=0;i<s;i++)
      e[s][i]=e[i][s]=min(tmp,e[t][i]);
    for(int i=s+1;i<n;i++)
      if(p[i]==t&&F.level[i]!=-1) p[i]=s;
  }
}
```

2.8 Max flow with lower/upper bound

```
// Max flow with lower/upper bound on edges
// use with ISAP
int in[N],out[N],l[M],r[M],a[M],b[M];
int solve(int n, int m, int s, int t){
  flow.init(n);
  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=0;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(t,s,INF);
  if( flow.solve() != nd ) return -1; // no solution
  int ans=flow.G[s].back().c; // source to sink
  flow.G[s].back().c=flow.G[t].back().c=0;
  // take out super source and super sink
  for(size_ti=0;i<flow.G[flow.s].size();i++){
    flow.G[flow.s][i].c=0;
    Maxflow::Edge &e=flow.G[flow.s][i];
    flow.G[e.v][e.r].c=0;
  }
  for(size_ti=0;i<flow.G[flow.t].size();i++){
    flow.G[flow.t][i].c=0;
    Maxflow::Edge &e=flow.G[flow.t][i];
    flow.G[e.v][e.r].c=0;
  }
  flow.addEdge(flow.s,s,INF);
```

```
  flow.addEdge(t,flow.t,INF);
  flow.reset(); // set iter,d,gap to 0
  return ans + flow.solve();
}
```

2.9 HLPPA

```
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; T ef[MAXN]; vector<Edge> adj[MAXN];
  deque<int> lst[MAXN]; vector<int> gap[MAXN];
  int ptr[MAXN],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 add_edge(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?0:f});
  }
  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 flow() {
    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;
  }
};
```

2.10 Flow Method

```
Maximize c^T x subject to Ax ≤ b, x ≥ 0;
with the corresponding symmetric dual problem,
Minimize b^T y subject to A^T y ≥ c, y ≥ 0.

Maximize c^T x subject to Ax ≤ b;
with the corresponding asymmetric dual problem,
Minimize b^T y subject to A^T y = c, y ≥ 0.

Maximize \sum x subject to x_i + x_j ≤ Aij, x ≥ 0;
=> Maximize \sum x subject to x_i + x_j ≤ A_ij;
=> Minimize A^T y = \sum A_ij y_ij subject to for all v,
     \sum_{i=v or j=v} y_ij = 1, y_ij ≥ 0
=> possible optimal solution: y_ij = {0, 0.5, 1}
=> y'=2y: \sum_{i=v or j=v} y'_ij = 2, y'_ij = {0, 1, 2}
=> Minimum Bipartite perfect matching/2 (V1=X,V2=X,E=A)

General Graph:
|Max Ind. Set| + |Min Vertex Cover| = |V|
|Max Ind. Edge Set| + |Min Edge Cover| = |V|
Bipartite Graph:
|Max Ind. Set| = |Min Edge Cover|
|Max Ind. Edge Set| = |Min Vertex Cover|

To reconstruct the minimum vertex cover, dfs from each
unmatched vertex on the left side and with unused edges
only. Equivalently, dfs from source with unused edges
only and without visiting sink. Then, a vertex is
chosen iff. it is on the left side and without visited
or on the right side and visited through dfs.

Minimum Weighted Bipartite Edge Cover:
Construct new bipartite graph with n+m vertices on each
     side:
for each vertex u, duplicate a vertex u' on the other
     side
for each edge (u,v,w), add edges (u,v,w) and (v',u',w)
for each vertex u, add edge (u,u',2w) where w is min
     edge connects to u
then the answer is the minimum perfect matching of the
     new graph (KM)

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.

Maximum closed subgraph
1. connect source with positive weighted vertex(capacity
     =weight)
2. connect sink with negitive weighted vertex(capacity=-
     weight)
3. make capacity of the original edges = inf
4. ans = sum(positive weighted vertex weight) - (max
     flow)

Minimum Path Cover of DAG
1. For each vertex v, split it to v_in and v_out.
2. For each edge (u->v), add an edge between u_out and
     v_in
3. |Minimum Path Cover| = |V| - |Maximum Matching| of
     the new bipartite graph
```

3 Math

3.1 FFT

```
const int MXN=1048576;// (must be 2^k)
// before any usage,run pre_fft() first
typedef long double ld;
typedef complex<ld> cplx;
```

```
const ld PI=acosl(-1); const cplx I(0,1);
cplx omega[MXN+1];
void pre_fft(){
  for(int i=0;i<=MXN;i++)
    omega[i]=exp(i*2*PI/MXN*I);
}
// n must be 2^k; fft(cplx(a+b,a-b)); Re(ifft(c))/4+0.5;
void fft(int n,cplx a[],bool inv=false){
  int basic=MXN/n,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?MXN-(i*theta%MXN):i*theta%MXN];
      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)%MXN;
  }
  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;
}
```

3.2 NTT

```
/* p=a*2^k+1
   p                    a      k      root
   998244353            119    23     3
   1107296257           33     25     10
   2013265921           15     27     31
   2061584302081        15     37     7
   2748779069441        5      39     3
   1945555039024054273  27     56     5 */
template<ll P,ll root,int MAXK,int MAXN>
struct NTT{
  static ll powi(ll a,ll b){
    ll ret=1;
    for(;b;b>>=1,a=mul(a, a, P)){
      if(b&1) ret=mul(ret, a, P);
    }
    return ret;
  }
  static ll inv(ll a,ll b){
    if(a==1) return 1;
    return (((a-inv(b%a,a))*b+1)/a)%b; // overflow
  }
  ll omega[MAXK+1],inv_omega[MAXK+1];
  NTT(){
    omega[MAXK]=powi(root,(P-1)>>MAXK);
    for(int i=MAXK-1;i>=0;i--)
      omega[i]=mul(omega[i+1], omega[i+1], P);
    for(int i=0;i<=MAXK;i++)
      inv_omega[i]=inv(omega[i],P);
  }
  void tran(int n,ll a[],bool inv_ntt=false){//n=2^i
    for(int i=1,j=0;i<n;i++){
      for(int k=n>>1;!((j^=k)&k);k>>=1);
      if(i<j) swap(a[i],a[j]);
    }
    ll *G=(inv_ntt?inv_omega:omega);
    for(int k=2,t=1;k<=n;k<<=1){
      int k2=k>>1;ll dw=G[t++];
      for(int j=0;j<n;j+=k){
        ll w=1;
        for(int i=j;i<j+k2;i++){
          ll x=a[i],y=mul(a[i+k2], w, P);
          a[i]=x+y; if(a[i]>=P) a[i]-=P;
          a[i+k2]=x-y; if(a[i+k2]<0) a[i+k2]+=P;
          w=mul(w, dw, P);
    } } }
    if(inv_ntt){
      ll inv_n=inv(n,P);
      for(int i=0;i<n;i++) a[i]=mul(a[i], inv_n, P);
    }
  }
};
const int MAXN=4194304,MAXK=22; //MAXN=2^k
const ll P=2013265921,root=31;
NTT<P,root,MAXK,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
*ternery xor convolution:
*x=(x0+x1+x2,x0+x1w+x2w^2,x0+x1w^2+x2w)
*inv=(1/3)*(x0+x1+x2,x0+x1w^2+x2w,x0+x1w+x2w^2)
*where w^3=1 and w^2=-w-1 */
typedef long long ll;
const int MAXN=(1<<20)+10; const ll MOD=1e9+7;
inline ll pw(ll x,ll k) {
  ll res=1;
  for(ll bs=x;k;k>>=1,bs=(bs*bs)%MOD)
    if(k&1) res=(res*bs) % MOD;
  return res;
}
inline ll invf(ll x) { return pw(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;
      }
  }
  ll invN=invf(N);
  if(inv)
    for(int i=0;i<N;i++) { x[i] *= invN; x[i] %= MOD; }
}
```

3.4 FFT Mod

```
void fftmod(ll a[],int n,ll b[],int m,ll c[],ll mod){
  int B=32-__builtin_clz(n+m-1),N=1<<B,cut=sqrt(mod);
  vector<cplx> L(N),R(N),outs(N),outl(N);
  for(int i=0;i<n;i++) L[i]=cplx(a[i]/cut,a[i]%cut);
  for(int i=0;i<m;i++) R[i]=cplx(b[i]/cut,b[i]%cut);
  fft(N,L.data()); fft(N,R.data());
  for(int i=0;i<N;i++){
    int j=-i&(N-1);
    outl[j]=(L[i]+conj(L[j]))*R[i]/(2.0L*N);
    outs[j]=(L[i]-conj(L[j]))*R[i]/(2.0L*N)/1il;
  }
  fft(N,outl.data()); fft(N,outs.data());
  for(int i=0;i<n+m-1;i++){
    ll av=real(outl[i])+.5,cv=imag(outs[i])+.5;
    ll bv=(ll)(imag(outl[i])+.5)+(ll)(real(outs[i])+.5);
    c[i]=((av%mod*cut+bv)%mod*cut+cv)%mod;
  }
} // NlogN*mod < 8.6e14 (maybe >=1e16 in practice)
```

3.5 Poly operator

```
struct PolyOp {
#define FOR(i,c) for (int i=0; i<(c); ++i)
  NTT<P,root,MAXK,MAXN> ntt;
  static int nxt2k(int x) {
    int i=1; for (; i<x; i <<= 1); return i;
  }
  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
  // a^-1=2b-bba
  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[]){
  for(int i=1;i<=n-1;i++) 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);
}
bool Sqrt(int n,ll a[],ll b[]){
  // Square root of a : b*b=a ( mod x^n )
  // bb=a  mod x^(n/2)
  // ( bb-a )^2=0 mod x^n
  // ( bb+a )^2=4 bba
  // ( ( bb+a ) / 2b )^2=a
  // sqrt(a)=b / 2+a / 2b
  static ll c[MAXN]; int ind=0,x,y,p=1;
  while(a[ind]==0) ind++;
  for(int i=0;i<n;i++) a[i]=a[i+ind];
  if((ind&1)||!dsqrt(a[0],mod,x,y)) // discrete sqrt
    return 0;
  b[0]=min(x,y);
  while(p<n) p<<=1;
  for(int t=2;t<=p;t<<=1){
    Inv(t,b,c); Mul(t,a,t,c,c);
    for(int i=0;i<t;i++)
      b[i]=(b[i]+c[i])*inv(2)%mod;
  }
  if(ind){
    for(int i=p-1;i>=ind/2;i--) b[i]=b[i-ind/2];
    for(int i=0;i<ind/2;i++) b[i]=0;
    for(int i=p-1;i>=ind;i--) a[i]=a[i-ind];
    for(int i=0;i<ind;i++) a[i]=0;
  }
}
} polyop;
```

3.6 Poly Interpolation

```
typedef vector<double> poly;
poly interpolate(poly x,poly y,int n){
  poly res(n),temp(n);
  for(int k=0;k<n-1;k++) for(int i=k+1;i<n;i++)
    y[i]=(y[i]-y[k])/(x[i]-x[k]);
  double last=0; temp[0]=1;
  for(int k=0;k<n;k++) for(int i=0;i<n;i++){
    res[i]+=y[k]*temp[i];
    swap(last,temp[i]); temp[i]-=last*x[k];
  }
  return res;
}
```

3.7 Linear Recurrence

```
// Usage: linearRec({0, 1}, {1, 1}, k) //k'th fib
typedef vector<ll> Poly;
ll linearRec(Poly&& S, Poly&& tr, ll k) {
  int n=tr.size();
  auto combine=[&](Poly& a, Poly& b) {
    Poly res(n*2+1);
    for(int i=0;i<=n;i++) for(int j=0;j<=n;j++)
      res[i+j]=(res[i+j]+a[i]*b[j])%mod;
    for(int i=2*n;i>n;--i) for(int j=0;j<n;j++)
      res[i-1-j]=(res[i-1-j]+res[i]*tr[j])%mod;
    res.resize(n+1);
    return res;
  }; // combine: a * b mod (x^n-tr)
  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;
  for(int i=0;i<n;i++) res=(res+pol[i+1]*S[i])%mod;
  return res;
}
```

3.8 BerlekampMassey

```
// find shortest linear recurrence relation O(n^2)
// example: BM({1,1,2,3,5,8,13,21})
// 2*len terms for uniqueness
inline vector<ll> BM(const vector<ll> &x) {
  vector<ll> ls, cur; int lf; ll ld;
  for(int i=0;i<x.size();++i) {
    ll t=0;
    for(int j=0;j<cur.size();++j)
      t=(t+x[i-j-1]*cur[j])%mod;
    if((t-x[i])%mod==0) continue;
    if(!cur.size()) {
      cur.resize(i+1); lf=i; ld=(t-x[i])%mod; continue;
    }
    ll k=-(x[i]-t)*inv(ld, mod)%mod;
    vector<ll> c(i-lf-1); c.push_back(k);
    for(auto j:ls) c.push_back(-j*k%mod);
    if(c.size()<cur.size()) c.resize(cur.size());
    for(int j=0;j<cur.size();++j)c[j]=(c[j]+cur[j])%mod;
    if(i-lf+(int)ls.size()>=(int)cur.size())
      ls=cur,lf=i,ld=(t-x[i])%mod;
    cur=move(c);
  }
  for(auto& xx:cur) xx=(xx%mod+mod)%mod;
  return cur;
}
```

3.9 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 :  pirmes <= 13
// n < 2^64                       7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
bool witness(ll a,ll n,ll u,int t){
```

```
  if(!(a%=n)) 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=100) {
  // iterate s times of witness on n
  // return 1 if prime, 0 otherwise
  if(n<2) return 0;
  if(!(n&1)) return n == 2;
  ll u=n-1; int t=0;
  while(!(u&1)) u>>=1, t++;
  while(s--){
    ll a=randll()%(n-1)+1;
    if(witness(a,n,u,t)) return 0;
  }
  return 1;
}
```

3.10 Simplex

```
/*target:
  max \sum_{j=1}^n A_{0,j}*x_j
condition:
  \sum_{j=1}^n A_{i,j}*x_j<=A_{i,0} |i=1~m
  x_j >= 0 |j=1~n
VDB=vector<double>*/
template<class VDB>
VDB simplex(int m,int n,vector<VDB> a){
  vector<int> left(m+1),up(n+1);
  iota(left.begin(),left.end(),n);
  iota(up.begin(),up.end(),0);
  auto pivot=[&](int x,int y){
    swap(left[x],up[y]);
    auto k=a[x][y];a[x][y]=1; vector<int> pos;
    for(int j=0;j<=n;++j){
      a[x][j]/=k;
      if(a[x][j]!=0) pos.push_back(j);
    }
    for(int i=0;i<=m;++i){
      if(a[i][y]==0||i==x) continue;
      k=a[i][y],a[i][y]=0;
      for(int j:pos) a[i][j] -= k*a[x][j];
    }
  };
  for(int x,y;;){
    for(int i=x=1;i<=m;++i) if(a[i][0]<a[x][0]) x=i;
    if(a[x][0]>=0) break;
    for(int j=y=1;j<=n;++j) if(a[x][j]<a[x][y]) y=j;
    if(a[x][y]>=0) return VDB(); // infeasible
    pivot(x,y);
  }
  for(int x,y;;){
    for(int j=y=1;j<=n;++j) if(a[0][j]>a[0][y]) y=j;
    if(a[0][y]<=0) break;
    x=-1;
    for(int i=1;i<=m;++i) if(a[i][y]>0)
      if(x==-1||a[i][0]/a[i][y]<a[x][0]/a[x][y]) x=i;
    if(x==-1) return VDB(); // unbounded
    pivot(x,y);
  }
  VDB ans(n + 1);
  for(int i=1;i<=m;++i)
    if(left[i]<=n) ans[left[i]]=a[i][0];
  ans[0]=-a[0][0];
  return ans;
}
```

3.11 Faulhaber

```
/* 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],inv[MAXK+1]; // bernoulli number,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() {
  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]);
  }
  for(int i=1;i<=MAXK;i++) inv[i]=getinv(i);
  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;
}
```

3.12 Chinese Remainder

```
ll crt(ll x1, ll m1, ll x2, ll m2) {
  ll g = __gcd(m1, m2); // or std::gcd
  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)%lcm*m1%lcm+x1; // overflow m^3
  return (res % lcm + lcm) % lcm;
}
```

3.13 Pollard Rho

```
// does not work when n is prime
ll f(ll x,ll mod){ return add(mul(x,x,mod),1,mod); }
ll pollard_rho(ll n) {
  if(!(n&1)) return 2;
  while(true){
    ll y=2,x=rand()%(n-1)+1,res=1,tmp=1;
    for(int sz=2;res==1;sz*=2){
      for(int i=0,t=0;i<sz&&res<=1;i++,t++){
        x=f(x,n); tmp=mul(tmp,abs(x-y),n);
        if(!(t&31)||i+1==sz) res=__gcd(tmp,n),tmp=1;
      }
      y=x;
    }
    if(res!=0&&res!=n) return res;
  }
}
```

3.14 ax+by=gcd

```
pair<ll,ll> gcd(ll a, ll b){
  if(b == 0) return {1, 0};
  pair<ll,ll> q = gcd(b, a % b);
  return {q.second, q.first - q.second * (a / b)};
}
```

3.15 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 where p is a prime
bool dsqrt(ll a,ll p,ll &x,ll &y){
  a%=p; if(p==2){ x=y=a; return true; }
  ll 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=0; 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);
    }
    ll s=mypow(a,h/2,p);
    for(int step=2; step<=t; step++){
      ll ss=mul(mul(s,s,p),a,p);
      for(int i=0;i<t-step;i++) ss=mul(ss,ss,p);
      if(ss+1==p) s=mul(s,pb,p);
      pb=mul(pb,pb,p);
    }
    x=mul(s,a,p); y=p-x;
  }
  return true;
}
```

3.16 Discrete logarithm

```
ll dlog(ll x,ll y,ll m){
  if(y==1||m==1) return 0;
  ll s=max((int)sqrt(m),1)+2,g=1;
  unordered_map<ll,ll> mp;
  for(ll i=0;i<s;i++,g=g*x%m){
    if(g==y) return i;
    mp[g*y%m]=i;
  }
  if(gcd(g,m)!=gcd(y,m)) return -1; // std::gcd
  for(ll i=1,h=g;i<s;i++,h=h*g%m){
    if(mp.count(h)) return i*s-mp[h];
  }
  return -1;
}
```

3.17 Romberg

```
// 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();
}
```

3.18 Simpson

```
template<class F>
ld quad(ld a,ld b,F f,const int n=1000) {
    ld h=(b-a)/2/n,v=f(a)+f(b);
    for(int i=1;i<n*2;++i) v+=f(a+i*h)*(i&1?4:2);
    return v*h/3;
}
```

3.19 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;
}
```

3.20 Roots of Polynomial

```
const double eps=1e-12,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=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]
```

3.21 Sum of Division/Modular

```
ull sumsq(ull n){ return n/2*((n-1)|1); }
// sum i=0~n-1 floor((ki+c)/m)
ull divsum(ull n,ull k,ull c,ull m){
  ull res=k/m*sumsq(n)+c/m*n; k%=m; c%=m;
  if(!k) return res;
  ull n2=(n*k+c)/m;
  return res+(n-1)*n2-divsum(n2,m,m-1-c,k);
}
// sum i=0~n-1 (ki+c)%m
ll modsum(ull n, ll k, ll c, ll m){
  c=(c%m+m)%m; k=(k%m+m)%m;
  return n*c+k*sumsq(n)-m*divsum(n,k,c,m);
}
```

3.22 Fraction Binary Search

```
//find smallest p/q in [0,1] s.t. f(p/q)=1&&p,q<=N
struct Frac{ll p,q;};
Frac fracBS(function<bool(Frac)> f,ll N) {
  bool dir=1,A=1,B=1;
  Frac lo{0,1},hi{1,1}; // set hi to 1/0 to search (0,N]
  if(f(lo)) return lo;
  assert(f(hi));
  while(A or B){
    ll adv=0,step=1; // move hi if dir, else lo
    for(int si=0;step;(step*=2)>>=si){
      adv+=step; Frac m{lo.p*adv+hi.p,lo.q*adv+hi.q};
      if(abs(m.p)>N or m.q>N or dir==!f(m))
        adv-=step,si=2;
    }
    hi.p+=lo.p*adv; hi.q+=lo.q*adv;
    dir=!dir; swap(lo,hi); A=B; B=!!adv;
  }
  return dir?hi:lo;
}
```

3.23 Closest Fraction

```
// x>=0, find closest p/q with p,q <= N. |p/q-x| <= 1/qN
pair<ll,ll> approximate(ld x,ll N) {
  ll LP=0,LQ=1,P=1,Q=0,inf=LLONG_MAX; ld y=x;
  for(;;){
    ll lim=min(P?(N-LP)/P:inf, Q?(N-LQ)/Q:inf),
```

```
    a=(ll)floor(y),b=min(a,lim),NP=b*P+LP,NQ=b*Q+LQ;
    if(a>b)
      return (abs(x-(ld)NP/(ld)NQ)<abs(x-(ld)P/(ld)Q))?
        make_pair(NP,NQ):make_pair(P,Q);
    if(abs(y=1/(y-(ld)a))>3*N) return {NP,NQ};
    LP=P; P=NP; LQ=Q; Q=NQ;
  }
}
```

3.24 Primes and μ function

```
/* 12721, 13331, 14341, 75577, 123457, 222557, 556679
* 999983, 1097774749, 1076767633, 100102021, 999997771
* 1001010013, 1000512343, 987654361, 999991231
* 999888733, 98789101, 987777733, 999991921, 1010101333
* 1010102101, 1000000000039, 1000000000000037
* 2305843009213693951, 4611686018427387847
* 9223372036854775783, 18446744073709551557 */
int mu[N],p_tbl[N]; // mobius, min prime factor
vector<int> primes;
void sieve() { // calculate multiplicative function f
  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; // f(i)=... where i is prime
    }
    for(int p:primes){
      int x=i*p;
      if(x>=N) break;
      p_tbl[x]=p; mu[x]=-mu[i];
      if(i%p==0){ // f(x)=f(i)/f(p^(k-1))*f(p^k)
        mu[x]=0; break;
      } // else f(x)=f(i)*f(p) where gcd(i,p)=1
} } }
vector<int> factor(int x){
  vector<int> fac{ 1 };
  while(x > 1){
    int fn=fac.size(),p=p_tbl[x],pos=0;
    while(x%p==0){
      x/=p;
      for(int i=0;i<fn;i++) fac.push_back(fac[pos++]*p);
    }
  }
  return fac;
}
```

3.25 Subset Convolution

```
// h(s)=\sum_{s' \subseteq s} f(s')g(s\backslash s')
vector<int> SubsetConv(int n,const vector<int> &f,const
    vector<int> &g){
  const int m=1<<n;
  vector<vector<int>> a(n+1,vector<int>(m)),b=a;
  for(int i=0;i<m;++i){
    a[__builtin_popcount(i)][i]=f[i];
    b[__builtin_popcount(i)][i]=g[i];
  }
  for(int i=0;i<=n;++i){
    for(int j=0;j<n;++j){
      for(int s=0;s<m;++s){
        if(s>>j&1){
          a[i][s]+=a[i][s^(1<<j)];
          b[i][s]+=b[i][s^(1<<j)];
  } } } }
  vector<vector<int>> c(n+1,vector<int>(m));
  for(int s=0;s<m;++s){
    for(int i=0;i<=n;++i){
      for(intj=0;j<=i;++j) c[i][s]+=a[j][s]*b[i-j][s];
  } }
  for(int i=0;i<=n;++i){
    for(int j=0;j<n;++j){
      for(int s=0;s<m;++s){
        if(s>>j&1) c[i][s]-=c[i][s^(1<<j)];
  } } }
  vector<int> res(m);
  for(int i=0;i<m;++i)
    res[i]=c[__builtin_popcount(i)][i];
  return res;
}
```

3.26 Result

- Lucas' Theorem :
  For $n,m\in\mathbb{Z}^*$ and prime P, $C(m,n)$ mod $P = \Pi(C(m_i,n_i))$ where m_i is the i-th digit of m in base P.
- 1st Stirling Numbers(permutation $|P|=n$ with k cycles):
  $S(n,k)=$ coefficient of x^k in $\Pi_{i=0}^{n-1}(x+i)$
  $S(n+1,k)=nS(n,k)+S(n,k-1)$
- 2nd Stirling Numbers(Partition n elements into k non-empty set):
  $S(n,k)=\frac{1}{k!}\sum\limits_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$
  $S(n+1,k)=kS(n,k)+S(n,k-1)$
- Calculate $f(x+n)$ where $f(x)=\sum\limits_{i=0}^{n-1}a_ix^i$:
  $f(x+n)=\sum\limits_{i=0}^{n-1}a_i(x+n)^i=\sum\limits_{i=0}^{n-1}x^i\cdot\frac{1}{i!}\sum\limits_{j=i}^{n-1}\frac{a_j}{j!}\cdot\frac{n^{j-i}}{(j-i)!}$
- Calculate $c[i-j]+=a[i]\times b[j]$ for $a[n],b[m]$
  1. a=reverse(a); c=mul(a,b); c=reverse(c[:n]);
  2. b=reverse(b); c=mul(a,b); c=rshift(c,m-1);
- Eulerian number(permutation $1\sim n$ with m $a[i]>a[i-1]$):
  $A(n,m)=\sum\limits_{i=0}^{m}(-1)^i\binom{n+1}{i}(m+1-i)^n$
  $A(n,m)=(n-m)A(n-1,m-1)+(m+1)A(n-1,m)$
- Derangement:
  $D(n)=(n-1)(D(n-1)+D(n-2))=nD(n-1)+(-1)^n$
- Pick's Theorem : $A=i+b/2-1$
- 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 :
  - number of spanning tree of undirected graph:
  degree matrix $D_{ii}=deg(i)$, $D_{ij}=0$
  adjacency matrix $G_{ij}=\#\,of\,(i,j)\in E$, $G_{ii}=0$,
  let $A=D-G$, delete any one row, one column, and cal $det(A')$
  - number of spanning tree of directed graph:
  in-degree matrix $D^{in}_{ii}=indeg(i)$, $D^{in}_{ij}=0$
  out-degree matrix $D^{out}_{ii}=outdeg(i)$, $D^{out}_{ij}=0$
  let $L^{in}=D^{in}-G$, $L^{out}=D^{out}-G$, delete the i-th row and column
  $det(L^{in}_i)$ and $det(L^{out}_i)$ is the number of spanning tree from/to root i
- Tutte Matrix:
  A graph $G=(V,E)$ has a perfect matching iff $det(A)\neq 0$ where $A_{ij}=((i,j)\in E?(i<j?x_{ij}:-x_{ji}):0)$ and x_{ij} are random numbers.
- Erdős–Gallai theorem:
  There exists a simple graph with degree sequence $d_1\geq\cdots\geq d_n$ iff $\sum\limits_{i=1}^{n}d_i$ is even and $\sum\limits_{i=1}^{k}d_i\leq k(k-1)+\sum\limits_{i=k+1}^{n}min(d_i,k),\forall 1\leq k\leq n$
- Burnside Lemma: $|X/G|=\frac{1}{|G|}\sum\limits_{g\in G}|X^g|$
- Polya theorem: $|Y^x/G|=\frac{1}{|G|}\sum\limits_{g\in G}m^{c(g)}$
  $m=|Y|$: num of colors, c(g) : num of cycle
- Prufer Sequence:
  There is a bijection between the set of labeled trees with n vertices and the set of sequences of length $n-2$ containing $1\sim n$.
  Property: Each vertex i exists d_i-1 times in the sequence.
  Tree to sequence: iterate $n-2$ times to remove a leaf with smallest id and append its adjacent vertex's id to the end of the sequence.
  Sequence to tree: iterate through $i=1\sim n-2$ and connect a_i with the smallest id that doesn't exist in $a_{i+1},\ldots,a_{n-2}$ and haven't been used yet. Also connect the remaining two unused vertices at last.
- Anti SG (the person who has no strategy wins) :
  first player wins iff either
  1. SG value of ALL subgame ≤ 1 and SG value of the game $=0$
  2. SG value of some subgame >1 and SG value of the game $\neq 0$
- Möbius inversion formula :
  $g(n)=\sum\limits_{d|n}f(d)$ for every integer $n\geq 1$, then
  $f(n)=\sum\limits_{d|n}\mu(d)g(\frac{n}{d})=\sum\limits_{d|n}\mu(\frac{n}{d})g(d)$ for every integer $n\geq 1$
  Dirichlet convolution : $f*g=g*f=\sum\limits_{d|n}f(d)g(\frac{n}{d})=\sum\limits_{d|n}f(\frac{n}{d})g(d)$
  $g=f*1\Leftrightarrow f=g*\mu$, $\epsilon=\mu*1$, $Id=\phi*1$, $d=1*1$, $\sigma=Id*1=\phi*d$,
  $\sigma_k=Id_k*1$ where $\epsilon(n)=[n=1]$, $1(n)=1$, $Id(n)=n$, $Id_k(n)=n^k$,
  $d(n)=\#(divisor)$, $\sigma(n)=\sum divisor$, $\sigma_k(n)=\sum divisor^k$
- Find a Primitive Root of n:
  n has primitive roots iff $n=2,4,p^k,2p^k$ where p is an odd prime.
  1. Find $\phi(n)$ and all prime factors of $\phi(n)$, says $P=\{p_1,...,p_m\}$
  2. $\forall g\in[2,n)$, if $g^{\frac{\phi(n)}{p_i}}\neq 1,\forall p_i\in P$, then g is a primitive root.
  3. Since the smallest one isn't too big, the algorithm runs fast.
  4. n has exactly $\phi(\phi(n))$ primitive roots.

- Sum of Two Squares Thm (Legendre):
  For a given positive integer N, let
  $D1 = $ (# of $d \in N$ dividing N that $d = 1 \pmod 4$)
  $D3 = $ (# of $d \in N$ dividing N that $d = 3 \pmod 4$)
  then N can be written as a sum of two squares in exactly $R(N) = 4(D1 - D3)$ ways.

- Difference of $D1 - D3$ Thm:
  let $N = 2^t \times [p_1^{e1} \times ... \times p_r^{er}] \times [q_1^{f1} \times ... \times q_s^{fs}]$
  where $p_i \in mod\ 4 = 1\ prime$, $q_i \in mod\ 4 = 3\ prime$
  then $D1 - D3 = \begin{cases} (e1+1)(e2+1)...(er+1) & if\ f_i\ all\ even \\ 0 & if\ any\ f_i\ is\ odd \end{cases}$

- Sherman-Morrison formula:
  suppose $A \in \mathbb{R}^{n \times n}$ is invertible and $u, v \in \mathbb{R}^n$
  $A + uv^T$ is invertible if and only if $1 + v^T A^{-1} u \neq 0$
  $(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1+v^TA^{-1}u}$

- Pohlig-Hellman algorithm (discrete log):
  Given an **order** n group, generator g, element h, find x s.t. $g^x = h$.
  If $n = p^e$:
  * let $x_0 = 0$, $\gamma = g^{p^{e-1}}$ where γ has order p.
  * for $k = 0 \sim e-1$:
  1. let $h_k = (g^{-x_k}h)^{p^{e-1-k}}$ whose order divide $p \implies h_k \in \langle\gamma\rangle$.
  2. find d_k s.t. $\gamma^{d_k} = h_k$ with baby-step giant-step in $O(\sqrt{p})$.
  3. set $x_{k+1} = x_k + p^k d_k$
  * return x_e in total time complexity $O(e\sqrt{p})$
  If $n = \Pi_{i=1}^r p_i^{e_i}$:
  * for each $i = 1 \sim r$:
  1. let $g_i = g^{n/p_i^{e_i}}$ having order $p_i^{e_i}$, $h_i = h^{n/p_i^{e_i}}$ where $h_i \in \langle g_i \rangle$.
  2. find x_i s.t. $g_i^{x_i} = h_i$ using above algorithm.
  * return $x = CRT(\{x_i \mod p_i^{e_i}\})$

4 Geometry

4.1 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);
}
```

4.2 halfPlaneIntersection

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

4.3 Intersection of 2 segments

```
bool onseg(Pt p, Line L) {
  Pt x = L.s-p, y = L.e-p;
  return dcmp(x^y) == 0 && dcmp(x*y) <= 0;
} // inseg: dcmp(x^y)==0&&dcmp(x*y)<0
// assume a.s != a.e != b.s != b.e
Pt SSIntersect(Line a, Line b) {
  Pt p = LLIntersect(a, b);
  if(isinf(p.x)&&(onseg(a.s,b)||onseg(a.e,b)||onseg(b.s,
      a)||onseg(b.e,a))) return p; // overlap
  if(isfinite(p.x)&&onseg(p,a)&&onseg(p,b)) return p;
  return {NAN,NAN}; // non-intersect
}
```

4.4 Banana

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

4.5 Intersection of circle and line

```
vector<Pt> CLInter(const Line &a,const Circle &c){
    Pt p=a.s+(c.o-a.s)*a.v/norm2(a.v)*a.v;
    ld d=c.r*c.r-norm2(c.o-p);
    if(d<-eps) return {};
    if(d<eps) return {p};
    Pt v=a.v/norm(a.v)*sqrt(d);
    return {p+v,p-v};
}
```

4.6 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,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);
}
```

4.7 Intersection of 2 circles

```
vector<Pt> CCinter(Circle& a, Circle& b){
  Pt o1=a.o,o2=b.o; ld r1=a.r,r2=b.r;
  if(norm(o1-o2)>r1+r2) return {};
  if(norm(o1-o2)<max(r1,r2)-min(r1,r2)) return {};
  ld d2=(o1-o2)*(o1-o2),d=sqrt(d2);
  if(d>r1+r2) return {};
  Pt u=(o1+o2)*0.5+(o1-o2)*((r2*r2-r1*r1)/(2*d2));
  ld 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);
  return {u+v,u-v};
}
```

4.8 Circle cover

```
#define N 1021
struct CircleCover{             // overlap
  int C; Circle c[N]; bool g[N][N],over[N][N];
```

```
  // Area[i] : area covered by at least i circles
  ld Area[N];
  void init(int _C){ C=_C;}
  struct Teve {
    Pt p;ld ang;int add;
    Teve() {}
    Teve(Pt _a,ld _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(Circle& a,Circle &b,int x)
  { return sign(norm(a.o-b.o)-a.r-b.r)>x; }
  bool contain(Circle& a,Circle &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++)
        over[i][j]=contain(i,j);
    for(int i=0;i<C;i++) for(int j=0;j<C;j++)
        g[i][j]=!(over[i][j]||over[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&&over[j][i]) cnt++;
      for(int j=0;j<C;j++)
        if(i!=j && g[i][j]){
          vector<Pt> v=CCinter(c[i],c[j]);
          ld A=atan2(v[0].y-c[i].o.y,v[0].x-c[i].o.x);
          ld B=atan2(v[1].y-c[i].o.y,v[1].x-c[i].o.x);
          eve[E++]=Teve(v[1],B,1);
          eve[E++]=Teve(v[0],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)*.5;
          ld theta=eve[j+1].ang-eve[j].ang;
          if(theta<0) theta+=2.*pi;
          Area[cnt]+=(theta-sin(theta))*c[i].r*c[i].r/2;
} } } } };
```

4.9 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); }
};
```

4.10 Convex Hull trick

```
/* Given a convexhull,answer querys 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,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]);
  } // sign: modify when changing to double
  int sign(ll x){ return x<0?-1:x>0; }
  pair<ll,int> get_tang(vector<Pt> &conv,Pt vec){
    int l=0,r=(int)conv.size()-2;
    while(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));
    while(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));
    while(l+1<r){
      int mid=(l+r)/2,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;
  }
};
```

4.11 Rotating Sweep Line

```
void rotatingSweepLine(vector<Pt> &ps){
  int n=int(ps.size()); vector<int> id(n),pos(n);
  vector<pair<int,int>> line(n*(n-1)/2); int m=0;
  for(int i=0;i<n;++i)
    for(int j=i+1;j<n;++j) line[m++]=make_pair(i,j);
  sort(line.begin(),line.end(),[&](const pair<int,int> &
      a,const pair<int,int> &b)->bool{
    if(ps[a.first].x==ps[a.second].y) return 0;
    if(ps[b.first].x==ps[b.second].y) return 1;
    return (double)(ps[a.first].y-ps[a.second].y)/(ps[a.
        first].x-ps[a.second].x)<(double)(ps[b.first].y-
        ps[b.second].y)/(ps[b.first].x-ps[b.second].x);
  }); // change to use multiply for better precision
  for(int i=0;i<n;++i) id[i]=i;
  sort(id.begin(),id.end(),[&](const int &a,const int &b
      ){ return ps[a]<ps[b]; }); // tie(x,y)
  for(int i=0;i<n;++i) pos[id[i]]=i;
  for(int i=0;i<m;++i){ pair<int,int> l=line[i];
    // do something: Line(ps[l.first],ps[l.second]);
    // id: sorted id of ps w.r.t dis(l,p) (dis<0 for
        points in -y direction); pos[id[i]]=i;
    tie(pos[l.first],pos[l.second],id[pos[l.first]],id[
        pos[l.second]])=make_tuple(pos[l.second],pos[l.
        first],l.second,l.first);
  }
}
```

4.12 Tangent line of two circles

```
vector<Line> go(const Circle& c1,const Circle& c2,int
    sign1){
  // sign1=1 for outer tang,-1 for inter tang
  vector<Line> ret;
  double d_sq=norm2(c1.o-c2.o);
  if(d_sq<eps) return ret;
  double d=sqrt(d_sq); Pt v=(c2.o-c1.o)/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.o+n*c1.r,p2=c2.o+n*(c2.r*sign1);
    if(fabs(p1.x-p2.x)<eps and fabs(p1.y-p2.y)<eps)
      p2=p1+perp(c2.o-c1.o);
    ret.push_back({p1,p2});
  }
  return ret;
}
```

4.13 Tangent line of point and circle

```
vector<Line> PCTangent(const Circle& C,const Pt& P){
  vector<Line> ans; Pt u=C.o-P; double dist=norm(u);
  if(dist<C.r) return ans;
  else if(abs(dist)<eps){
    ans.push_back({P,P+rotate(u,M_PI/2)});
    return ans;
  }
  else{
    double ang=asin(C.r/dist);
    ans.push_back({P,P+rotate(u,-ang)});
    ans.push_back({P,P+rotate(u,+ang)});
    return ans;
  }
}
```

4.14 Min 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 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;
}
```

4.16 Lower Concave Hull

```
const ll is_query=-(1LL<<62);
struct Line {
  ll m, b;
  mutable function<const Line*()> succ;
  bool operator<(const Line& rhs) const {
    if (rhs.b!=is_query) return m<rhs.m;
    const Line* s=succ();
    return s?b-s->b<(s->m-m)*rhs.m:0;
  }
}; // maintain upper hull for maximum
struct HullDynamic:public multiset<Line> {
  bool bad(iterator y) {
    auto z=next(y);
    if (y==begin()) {
      if (z==end()) return 0;
      return y->m==z->m&&y->b<=z->b;
    }
    auto x=prev(y);
    if(z==end()) return y->m==x->m&&y->b<=x->b;
    return
      (x->b-y->b)*(z->m-y->m)>=(y->b-z->b)*(y->m-x->m);
  }
  void insert_line(ll m, ll b) {
    auto y=insert({m, b});
    y->succ=[=]{ return next(y)==end()?0:&*next(y); };
    if(bad(y)) { erase(y); return; }
    while(next(y)!=end()&&bad(next(y))) erase(next(y));
    while(y!=begin()&&bad(prev(y))) erase(prev(y));
  }
  ll eval(ll x) {
    auto l=*lower_bound((Line) { x, is_query });
    return l.m*x + l.b;
  }
};
```

4.17 Delaunay Triangulation

```
/* Delaunay Triangulation:
Given a sets of points on 2D plane, find a
triangulation such that no points will strictly
inside circumcircle of any triangle.

find: return a triangle contain given point
add_point: add a point into triangulation

A Triangle is in triangulation iff. its has_chd is 0.
Region of triangle u: iterate each u.edge[i].tri,
each points are u.p[(i+1)%3], u.p[(i+2)%3]

Voronoi diagram: for each triangle in triangulation,
the bisector of all its edges will split the region.
nearest point will belong to the triangle containing it
*/
typedef double T;      // T is integer: eps=0
const int N=100000+5; const T inf=1e9, eps=1e-8;
T sqr(T x) { return x*x; }
// return p4 is in circumcircle of tri(p1,p2,p3)
bool in_cc(const Pt& p1, const Pt& p2, const Pt& p3,
    const Pt& p4){
  T u11=p1.x-p4.x; T u21=p2.x-p4.x; T u31=p3.x-p4.x;
  T u12=p1.y-p4.y; T u22=p2.y-p4.y; T u32=p3.y-p4.y;
  T u13=sqr(p1.x)-sqr(p4.x)+sqr(p1.y)-sqr(p4.y);
  T u23=sqr(p2.x)-sqr(p4.x)+sqr(p2.y)-sqr(p4.y);
  T u33=sqr(p3.x)-sqr(p4.x)+sqr(p3.y)-sqr(p4.y);
  T det=-u13*u22*u31+u12*u23*u31+u13*u21*u32
      -u11*u23*u32-u12*u21*u33+u11*u22*u33;
  return det > eps;
}
T side(const Pt& a, const Pt& b, const Pt& p)
{ return (b-a)^(p-a); }
typedef int SdRef; struct Tri; typedef Tri* TriRef;
struct Edge {
  TriRef tri; SdRef side;
  Edge():tri(0), side(0){}
  Edge(TriRef _tri, SdRef _side):tri(_tri), side(_side)
      {}
};
struct Tri {
  Pt p[3]; Edge edge[3]; TriRef chd[3];
  Tri() {}
  Tri(const Pt& p0, const Pt& p1, const Pt& p2) {
    p[0]=p0; p[1]=p1; p[2]=p2; chd[0]=chd[1]=chd[2]=0;
  }
  bool has_chd() const { return chd[0]!=0; }
  int num_chd() const {
    return chd[0]==0?0:(chd[1]==0?1:chd[2]==0?2:3);
  }
  bool contains(Pt const& q) const {
    for(int i=0;i<3;i++)
      if(side(p[i],p[(i+1)%3],q)<-eps) return false;
    return true;
  }
} pool[N*10], *tris;
void edge(Edge a, Edge b){
  if(a.tri) a.tri->edge[a.side]=b;
  if(b.tri) b.tri->edge[b.side]=a;
}
struct Trig { // Triangulation
  void init(){ // Tri should at least contain all points
    the_root=new(tris++)Tri(Pt(-inf,-inf),Pt(+inf+inf,-
        inf),Pt(-inf,+inf+inf));
  }
  TriRef find(Pt p)const{ return find(the_root,p); }
  void add_point(const Pt& p)
  { add_point(find(the_root,p),p); }
  TriRef the_root;
  static TriRef find(TriRef root, const Pt& p) {
    while(true){
      if(!root->has_chd()) return root;
      for(int i=0;i<3&&root->chd[i];++i)
        if (root->chd[i]->contains(p)) {
          root=root->chd[i]; break;
        }
    }
    assert(false); // "point not found"
  }
  void add_point(TriRef root, Pt const& p) {
    TriRef tab,tbc,tca; // split it into three triangles
    tab=new(tris++) Tri(root->p[0],root->p[1],p);
    tbc=new(tris++) Tri(root->p[1],root->p[2],p);
    tca=new(tris++) Tri(root->p[2],root->p[0],p);
    edge(Edge(tab,0), Edge(tbc,1));
    edge(Edge(tbc,0), Edge(tca,1));
    edge(Edge(tca,0), Edge(tab,1));
    edge(Edge(tab,2), root->edge[2]);
    edge(Edge(tbc,2), root->edge[0]);
    edge(Edge(tca,2), root->edge[1]);
    root->chd[0]=tab;root->chd[1]=tbc;root->chd[2]=tca;
    flip(tab,2); flip(tbc,2); flip(tca,2);
  }
  void flip(TriRef tri, SdRef pi) {
    TriRef trj=tri->edge[pi].tri; if (!trj) return;
    int pj=tri->edge[pi].side;
    if (!in_cc(tri->p[0],tri->p[1],tri->p[2],trj->p[pj])
        ) return;
    /* flip edge between tri,trj */
    TriRef trk=new(tris++) Tri(tri->p[(pi+1)%3],trj->p[
        pj],tri->p[pi]);
```

```
    TriRef trl=new(tris++) Tri(trj->p[(pj+1)%3],tri->p[
        pi],trj->p[pj]);
    edge(Edge(trk,0),Edge(trl,0));
    edge(Edge(trk,1),tri->edge[(pi+2)%3]);
    edge(Edge(trk,2),trj->edge[(pj+1)%3]);
    edge(Edge(trl,1),trj->edge[(pj+2)%3]);
    edge(Edge(trl,2),tri->edge[(pi+1)%3]);
    tri->chd[0]=trk; tri->chd[1]=trl; tri->chd[2]=0;
    trj->chd[0]=trk; trj->chd[1]=trl; trj->chd[2]=0;
    flip(trk,1); flip(trk,2); flip(trl,1); flip(trl,2);
  }
}tri; // the triangulation structure
vector<TriRef> triang; // vector of all triangle
set<TriRef> vst;
void go(TriRef now){ // store all tri into triang
  if(vst.find(now)!=vst.end()) return;
  vst.insert(now);
  if(!now->has_chd()){
    triang.push_back(now); return;
  }
  for(int i=0;i<now->num_chd();i++) go(now->chd[i]);
}
void build(int n,Pt* ps){ // build triangulation
  tris=pool; triang.clear(); vst.clear();
  random_shuffle(ps,ps+n); tri.init();
  for(int i=0;i<n;++i) tri.add_point(ps[i]);
  go(tri.the_root);
}
```

4.18 Min Enclosing Circle

```
struct Mec{ // return pair of center and r
  static const int N=101010;
  int n; Pt p[N], cen; double r2;
  void init(int _n,Pt _p[]){
    n=_n; memcpy(p,_p,sizeof(Pt)*n);
  }
  double sqr(double a){ return a*a; }
  Pt center(Pt p0, Pt p1, Pt p2){
    Pt a=p1-p0,b=p2-p0;
    double c1=norm2(a)*0.5,c2=norm2(b)*0.5,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(){ // expected O(n)
    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;
double det(double m[3][3]){
  return m[0][0]*m[1][1]*m[2][2]+m[0][1]*m[1][2]*m[2][0]
    +m[0][2]*m[2][1]*m[1][0]-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];
}
void ball(){
  Pt q[3]; double m[3][3],sol[3],L[3],d;
  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(d=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])/d;
      L[1]=(sol[1]*m[0][0]-sol[0]*m[1][0])/d;
      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;
      d=det(m); if(fabs(d)<eps) return;
      for(j=0;j<3;++j){
        for(i=0;i<3;++i) m[i][j]=sol[i];
        L[j]=det(m)/d;
        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; int cur=-1;
  for(int i=0;i<m;i++) q[i]=q[i]-c;
  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 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;
}
```

4.22 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*b.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

```
const int MAXN=100010;
struct DominatorTree{ // 1-based
#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,s; vector<int> g[MAXN],pred[MAXN],cov[MAXN];
  int dfn[MAXN],nfd[MAXN],ts,par[MAXN];
  int sdom[MAXN],idom[MAXN],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 _s){
    ts=0; n=_n; 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); }
  } // x dominates y <=> path s to y must go through x
  void build(){ // <=> x is an ancestor of y in the tree
    REP(i,1,n){ // result tree edges: idom[i] -> i
      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 Directed MST(ElogE)

```
struct RollbackUF {
  vi e; vector<pii> st;
  RollbackUF(int n):e(n,-1) {}
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x]<0?x:find(e[x]); }
  int time() { return st.size(); }
  void rollback(int t) {
    for(int i=time();i-->t;)e[st[i].first]=st[i].second;
    st.resize(t);
  }
  bool join(int a,int b) {
    a=find(a),b=find(b);
    if(a==b) return false;
    if(e[a]>e[b]) swap(a,b);
    st.push_back({a,e[a]}); st.push_back({b,e[b]});
    e[a]+=e[b]; e[b]=a;
    return true;
  }
};
struct Edge {int a,b; ll w;};
struct Node { // lazy skew heap node
  Edge key; Node *l,*r; ll d;
  void prop() {
    key.w+=d; if(l) l->d+=d; if(r) r->d+=d; d=0;
  }
  Node(Edge e):key(e),l(0),r(0),d(0){}
  Edge top() { prop(); return key; }
};
Node *merge(Node *a,Node *b) {
  if(!a||!b) return a?a:b;
  a->prop(); b->prop();
  if(a->key.w>b->key.w) swap(a,b);
  swap(a->l,(a->r=merge(b,a->r)));
  return a;
}
void pop(Node*& a){ a->prop(); a=merge(a->l,a->r); }
pair<ll,vi> dmst(int n,int r,vector<Edge>& g){
  RollbackUF uf(n); vector<Node*> pq(n);
  for(Edge e:g) pq[e.b]=merge(pq[e.b],new Node(e));
  ll res=0; vi seen(n,-1),path(n),par(n); seen[r]=r;
  vector<Edge> Q(n),in(n,{-1,-1,0});
  deque<tuple<int,int,vector<Edge>>> cycs;
  for(int s=0;s<n;s++){
    int u=s,qi=0,w;
    while(seen[u]<0){
      if(!pq[u]) return {-1,{}};
      Edge e=pq[u]->top(); pq[u]->d-=e.w,pop(pq[u]);
      Q[qi]=e,path[qi++]=u,seen[u]=s;
      res+=e.w,u=uf.find(e.a);
      if(seen[u]==s) { // found cycle,contract
        Node* cyc=0; int end=qi,t=uf.time();
        do cyc=merge(cyc,pq[w=path[--qi]]);
        while(uf.join(u,w));
        u=uf.find(u),pq[u]=cyc,seen[u]=-1;
        cycs.push_front({u,t,{&Q[qi],&Q[end]}});
      }
    }
    for(int i=0;i<qi;i++) in[uf.find(Q[i].b)]=Q[i];
  }
  for(auto& [u,t,comp]:cycs) { // restore sol
    uf.rollback(t); Edge inEdge=in[u];
    for(auto& e:comp) in[uf.find(e.b)]=e;
    in[uf.find(inEdge.b)]=inEdge;
  }
  for(int i=0;i<n;i++) par[i]=in[i].a;
  return {res,par};
}
```

5.3 MaximalClique

```
#define N 80
struct MaxClique{ // 0-base
  typedef bitset<N> Int;
  Int lnk[N],v[N]; int n;
  void init(int _n){
    n=_n;
    for(int i=0;i<n;i++){
      lnk[i].reset(); v[i].reset();
    }
  }
```

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

5.4 MaxCliqueDyn

```
#define N 150
struct MaxClique{ // Maximum Clique
  bitset<N> a[N],cs[N]; int ans,sol[N],q,cur[N],d[N],n;
  void init(int _n){
    n=_n; for(int i=0;i<n;i++) a[i].reset();
  }
  void addEdge(int u,int v){ a[u][v]=a[v][u]=1; }
  void csort(vector<int> &r,vector<int> &c){
    int mx=1,km=max(ans-q+1,1),t=0,m=r.size();
    cs[1].reset(); cs[2].reset();
    for(int i=0;i<m;i++){
      int p=r[i],k=1;
      while((cs[k]&a[p]).count()) k++;
      if(k>mx){ mx++; cs[mx+1].reset(); }
      cs[k][p]=1; if(k<km) r[t++]=p;
    }
    c.resize(m); if(t) c[t-1]=0;
    for(int k=km;k<=mx;k++){
      for(int p=cs[k]._Find_first();p<N;p=cs[k].
          _Find_next(p)){
        r[t]=p; c[t]=k; t++;
  } } }
  void dfs(vector<int> &r,vector<int> &c,int l,bitset<N>
      mask){
    while(!r.empty()){
      int p=r.back(); r.pop_back(); mask[p]=0;
      if(q+c.back()<=ans) return;
      cur[q++]=p;
      vector<int> nr,nc; bitset<N> nmask=mask&a[p];
      for(int i:r) if(a[p][i]) nr.push_back(i);
      if(!nr.empty()){
        if(l<4){
          for(int i:nr) d[i]=(a[i]&nmask).count();
          sort(nr.begin(),nr.end(),
              [&](int x,int y){return d[x]>d[y];});
        }
        csort(nr,nc); dfs(nr,nc,l+1,nmask);
      }
      else if(q>ans){ ans=q; copy(cur,cur+q,sol); }
      c.pop_back(); q--;
    }
  }                // vertex mask
  int solve(bitset<N> mask=bitset<N>(string(N,'1'))){
    vector<int> r,c; ans=q=0;
    for(int i=0;i<n;i++) if(mask[i]) r.push_back(i);
    for(int i=0;i<n;i++) d[i]=(a[i]&mask).count();
    sort(r.begin(),r.end(),
        [&](int i,int j){return d[i]>d[j];});
    csort(r,c); dfs(r,c,1,mask);
    return ans; // vertices set: sol[0 ~ ans-1]
  }
}graph;
```

5.5 Strongly Connected Component

```
void dfs(int i){
  V[i]=low[i]=++ts,stk[top++]=i,instk[i]=1;
  for(auto x:E[i]){
    if(!V[x])dfs(x),low[i]=min(low[i],low[x]);
    else if(instk[x])low[i]=min(low[i],V[x]);
  }
  if(V[i]==low[i]){
    int j;
    do{j=stk[--top],instk[j]=0,scc[j]=i;
    }while(j!=i);
  }
}
```

5.6 Dynamic MST

```
/* Dynamic MST O( Q lg^2 Q )
 n nodes, m edges, Q query
 (u[i], v[i], w[i])->edge
 (qid[i], qw[i])->chg weight of edge No.qid[i] to qw[i]
 delete an edge: (i, \infty)
 add an edge: change from \infty to specific value */
const int M=1e5,MXQ=1e5,SZ=M+3*MXQ; int a[N],*tz;
int find(int x){ return x==a[x]?x:a[x]=find(a[x]); }
bool cmp(int aa,int bb){ return tz[aa]<tz[bb]; }
int kx[N],ky[N],kt,vd[N],id[M],app[M],cur;
long long answer[MXQ]; // answer after ith query
bool extra[M];
void solve(int *qx,int *qy,int Q,int n,int *x,int *y,int
     *z,int m1,long long ans){
  if(Q==1){
    for(int i=1;i<=n;i++) a[i]=0;
    z[qx[0]]=qy[0]; tz=z;
    for(int i=0;i<m1;i++) id[i]=i;
    sort(id,id+m1,cmp); int ri,rj;
    for(int i=0;i<m1;i++){
      ri=find(x[id[i]]); rj=find(y[id[i]]);
      if(ri!=rj){ ans+=z[id[i]]; a[ri]=rj; }
    }
    answer[cur++]=ans; return;
  }
  int ri,rj,tm=0,n2=0; kt=0;
  //contract
  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;
  }
  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]);
  for(int i=1;i<=n;i++) if(a[i]==0) vd[i]=++n2;
  for(int i=1;i<=n;i++) if(a[i]) vd[i]=vd[find(i)];
  int m2=0,*Nx=x+m1,*Ny=y+m1,*Nz=z+m1;
  for(int i=0;i<m1;i++) app[i]=-1;
  for(int i=0;i<Q;i++) if(app[qx[i]]==-1){
    Nx[m2]=vd[x[qx[i]]]; Ny[m2]=vd[y[qx[i]]];
    Nz[m2]=z[qx[i]]; app[qx[i]]=m2; m2++;
  }
  for(int i=0;i<Q;i++){z[qx[i]]=qy[i];qx[i]=app[qx[i]];}
  for(int i=1;i<=n2;i++) a[i]=0;
  for(int i=0;i<tm;i++){
    ri=find(vd[x[id[i]]]); rj=find(vd[y[id[i]]]);
    if(ri!=rj){
      a[ri]=rj; Nx[m2]=vd[x[id[i]]];
      Ny[m2]=vd[y[id[i]]]; Nz[m2]=z[id[i]]; m2++;
    }
```

```
  }
  int mid=Q/2;
  solve(qx,qy,mid,n2,Nx,Ny,Nz,m2,ans);
  solve(qx+mid,qy+mid,Q-mid,n2,Nx,Ny,Nz,m2,ans);
} // fill these variables and call work()
int u[SZ],v[SZ],w[SZ],qid[MXQ],qw[MXQ],n,m,Q;
void work(){if(Q) cur=0,solve(qid,qw,Q,n,u,v,w,m,0);}
```

5.7 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;
```

5.8 Minimum General Weighted Matching

```
struct Graph {
  // Minimum General Weighted Matching (Perfect Match)
  static const int MXN=105;
  int n,edge[MXN][MXN],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.push_back(u); onstk[u]=1;
    for(int v=0;v<n;v++){
      if(u!=v&&match[u]!=v&&!onstk[v]){
        int m=match[v];
        if(dis[m]>dis[u]-edge[v][m]+edge[u][v]){
          dis[m]=dis[u]-edge[v][m]+edge[u][v];
          onstk[v]=1; stk.push_back(v);
          if(SPFA(m)) return true;
          stk.pop_back(); onstk[v]=0;
    } } }
    onstk[u]=0; stk.pop_back();
    return false;
  }
  int solve() { // 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((int)stk.size()>=2){
            int u=stk.back();stk.pop_back();
            int v=stk.back();stk.pop_back();
            match[u]=v;match[v]=u;
      } } }
      if(!found) break;
    }
    int ret=0;
    for(int i=0;i<n;i++) ret+=edge[i][match[i]];
    return ret/2;
  }
}graph;
```

5.9 Maximum General Weighted Matching

```
struct WeightGraph {
  static const int INF=INT_MAX,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,lab[N*2],match[N*2],slack[N*2],st[N*2];
  int pa[N*2],flo_from[N*2][N+1],S[N*2],vis[N*2];
  edge g[N*2][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,w_max=0; long long tot_weight=0;
    for(int u=0;u<=n;++u) st[u]=u,flo[u].clear();
    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;
```

5.10 Minimum Steiner Tree

```
// Minimum Steiner Tree O(V 3^T+V^2 2^T)
// shortest_path() should be called before solve()
// w:vertex weight, default 0
const int V=66,T=10; const ll INF=1023456789;
struct SteinerTree{
  int n,dst[V][V],dp[1<<T][V],tdst[V],w[V];
  void init(int _n){
    n=_n; fill(w,w+n,0);
    for(int i=0;i<n;i++){
      for(int j=0;j<n;j++) dst[i][j]=INF;
      dst[i][i]=0;
    }
  }
  void add_edge(int ui,int vi,int wi){
    dst[ui][vi]=min(dst[ui][vi],wi);
    dst[vi][ui]=min(dst[vi][ui],wi);
  }
  void shortest_path(){
    for(int i=0;i<n;i++) for(int j=0;j<n;j++)
        if(i!=j&&dst[i][j]!=INF) dst[i][j]+=w[i];
    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]);
    for(int i=0;i<n;i++) for(int j=0;j<n;j++)
        if(dst[i][j]!=INF) dst[i][j]+=w[j];
  }
  int solve(const vector<int>& ter){
    int t=(int)ter.size();
    for(int i=0;i<(1<<t);i++)
      for(int j=0;j<n;j++) dp[i][j]=INF;
    for(int i=0;i<n;i++) dp[0][i]=0;
    for(int msk=1;msk<(1<<t);msk ++){
      if(msk==(msk&(-msk))){
        int who=__lg(msk);
        for(int i=0;i<n;i++)dp[msk][i]=dst[ter[who]][i];
        continue;
      }
      for(int i=0;i<n;i++)
        for(int submsk=(msk-1)&msk;submsk;submsk=(submsk
            -1)&msk)
          dp[msk][i]=min(dp[msk][i],dp[submsk][i]+dp[msk
              ^submsk][i]-w[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]-w[j]);
      }
      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;
```

5.11 BCC based on vertex

```
struct BccVertex{
  int n,nBcc,step,dfn[MXN],low[MXN],top,stk[MXN];
  vector<int> E[MXN],bccv[MXN];
  // vector<pair<int,int>> bcce[MXN];
  // pair<int,int> estk[MXM];// max edge number
  // int etop,id[MXN],pos[MXN];
  void init(int _n){
    n=_n;nBcc=step=0; for(int i=0;i<n;i++) E[i].clear();
  }
  void addEdge(int u,int v)
  { E[u].push_back(v);E[v].push_back(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){
        // estk[etop++]={u,v};
        DFS(v,u); low[u]=min(low[u],low[v]);
        if(low[v]>=dfn[u]){
          int z;bccv[nBcc].clear();//bcce[nBcc].clear();
          // pair<int,int> ez;
          // do{
          //   ez=estk[--etop];bcce[nBcc].push_back(ez);
          // }while(ez.first!=u);
          do{
            z=stk[--top]; bccv[nBcc].push_back(z);
            // id[z]=nBcc;pos[z]=bccv[nBcc].size();
          }while(z!=v);
          bccv[nBcc++].push_back(u);
        }
      }else{
        low[u]=min(low[u],dfn[v]);
        // if(dfn[v]<dfn[u]) estk[etop++]={u,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); // etop=0;
      }
    for(int i=0;i<nBcc;i++) res.push_back(bccv[i]);
    return res;
  }
  /* bccv[.first][{.second.first,.second.second}]=={u,v}
  pair<int,pair<int,int>> getpos(int u,int v){
    if(dfn[u]>dfn[v]) swap(u,v);
    int cid=id[v];
    if(id[u]==cid) return{cid,{pos[v],pos[u]}};
    else return{cid,{pos[v],bccv[cid].size()-1}};
  } */
}graph;
```

5.12 Min Mean Cycle

```
/* minimum mean cycle O(VE) */
const int E=101010,V=1021;
const double inf=1e9,eps=1e-8;
struct MMC{
  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);
    }
    if(st==-1) return inf;
    FZ(vst); edgeID.clear(); cycle.clear(); rho.clear();
    for(int i=n;!vst[st];st=prv[i--][st]){
      vst[st]++; edgeID.push_back(prve[i][st]);
      rho.push_back(st);
    }
    while(vst[st]!=2){
      int v=rho.back(); rho.pop_back();
      cycle.push_back(v); vst[v]++;
    }
    reverse(ALL(edgeID));
    edgeID.resize((int)cycle.size());
    return mmc;
  }
}mmc;
```

5.13 Directed Graph Min Cost Cycle

```
const int N=5010,M=200010; const ll INF=(1ll<<55);
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{ // works in O(NM)
  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;
```

5.14 K-th Shortest Path

```
// time: O(|E| \lg |E|+|V| \lg |V|+K)
// memory: O(|E| \lg |E|+|V|)
struct KSP{ // 1-base
  struct nd{
    int u,v; ll d;
    nd(int ui=0,int vi=0,ll di=INF){ u=ui; v=vi; d=di; }
  };
  struct heap{ nd* edge; int dep; heap* chd[4]; };
  static int cmp(heap* a,heap* b)
  { return a->edge->d > b->edge->d; }
  struct node{
    int v; ll d; heap* H; nd* E;
    node(){}
    node(ll _d,int _v,nd* _E){ d =_d; v=_v; E=_E; }
    node(heap* _H,ll _d){ H=_H; d=_d; }
    friend bool operator<(node a,node b)
    { return a.d>b.d; }
  };
  int n,k,s,t,dst[N]; nd *nxt[N];
  vector<nd*> g[N],rg[N]; heap *nullNd,*head[N];
  void init(int _n,int _k,int _s,int _t){
    n=_n; k=_k; s=_s; t=_t;
    for(int i=1;i<=n;i++){
      g[i].clear(); rg[i].clear();
      nxt[i]=NULL; head[i]=NULL; dst[i]=-1;
    }
  }
  void addEdge(int ui,int vi,ll di){
    nd* e=new nd(ui,vi,di);
    g[ui].push_back(e); rg[vi].push_back(e);
  }
  queue<int> dfsQ;
  void dijkstra(){
    while(dfsQ.size()) dfsQ.pop();
    priority_queue<node> Q; Q.push(node(0,t,NULL));
    while (!Q.empty()){
      node p=Q.top(); Q.pop(); if(dst[p.v]!=-1)continue;
      dst[p.v]=p.d; nxt[p.v]=p.E; dfsQ.push(p.v);
      for(auto e:rg[p.v]) Q.push(node(p.d+e->d,e->u,e));
    }
  }
  heap* merge(heap* curNd,heap* newNd){
    if(curNd==nullNd) return newNd;
    heap* root=new heap;memcpy(root,curNd,sizeof(heap));
    if(newNd->edge->d<curNd->edge->d){
      root->edge=newNd->edge;
      root->chd[2]=newNd->chd[2];
      root->chd[3]=newNd->chd[3];
      newNd->edge=curNd->edge;
      newNd->chd[2]=curNd->chd[2];
      newNd->chd[3]=curNd->chd[3];
    }
    if(root->chd[0]->dep<root->chd[1]->dep)
      root->chd[0]=merge(root->chd[0],newNd);
    else root->chd[1]=merge(root->chd[1],newNd);
    root->dep=max(root->chd[0]->dep,
              root->chd[1]->dep)+1;
    return root;
  }
  vector<heap*> V;
  void build(){
    nullNd=new heap; nullNd->dep=0; nullNd->edge=new nd;
    fill(nullNd->chd,nullNd->chd+4,nullNd);
    while(not dfsQ.empty()){
      int u=dfsQ.front(); dfsQ.pop();
      if(!nxt[u]) head[u]=nullNd;
      else head[u]=head[nxt[u]->v];
      V.clear();
      for(auto&& e:g[u]){
        int v=e->v;
        if(dst[v]==-1) continue;
        e->d+=dst[v]-dst[u];
        if(nxt[u]!=e){
          heap* p=new heap;fill(p->chd,p->chd+4,nullNd);
          p->dep=1; p->edge=e; V.push_back(p);
        }
      }
      if(V.empty()) continue;
      make_heap(V.begin(),V.end(),cmp);
#define L(X) ((X<<1)+1)
#define R(X) ((X<<1)+2)
      for(size_t i=0;i<V.size();i++){
        if(L(i)<V.size()) V[i]->chd[2]=V[L(i)];
        else V[i]->chd[2]=nullNd;
        if(R(i)<V.size()) V[i]->chd[3]=V[R(i)];
        else V[i]->chd[3]=nullNd;
      }
      head[u]=merge(head[u],V.front());
    }
  }
  vector<ll> ans;
  void first_K(){
    ans.clear(); priority_queue<node> Q;
    if(dst[s]==-1) return;
    ans.push_back(dst[s]);
    if(head[s]!=nullNd)
      Q.push(node(head[s],dst[s]+head[s]->edge->d));
    for(int _=1;_<k and not Q.empty();_++){
      node p=Q.top(),q; Q.pop(); ans.push_back(p.d);
      if(head[p.H->edge->v]!=nullNd){
        q.H=head[p.H->edge->v]; q.d=p.d+q.H->edge->d;
        Q.push(q);
      }
      for(int i=0;i<4;i++)
        if(p.H->chd[i]!=nullNd){
          q.H=p.H->chd[i];
          q.d=p.d-p.H->edge->d+p.H->chd[i]->edge->d;
          Q.push(q);
  } } }
  void solve(){ // ans[i] stores the i-th shortest path
    dijkstra(); build();
    first_K(); // ans.size() might less than k
  }
} solver;
```

5.15 Chordal Graph

```
struct Chordal{
  static const int MXN=100010;
  vector<int> E[MXN],V[MXN];
  int n,f[MXN],rk[MXN],order[MXN],stk[MXN],nsz[MXN];
  bool vis[MXN],isMaximalClique[MXN];
  void init(int _n){
    n=_n;
    for(int i=0;i<=n;++i){
      E[i].clear(),V[i].clear();
      f[i]=rk[i]=order[i]=vis[i]=0;
    }
```

```
  }
  void addEdge(int x,int y){
    E[x].push_back(y),E[y].push_back(x);
  }
  void mcs(){
    for(int i=1;i<=n;++i) V[0].push_back(i);
    for(int i=n,M=0;i>=1;--i){
      for(;;){
        while(V[M].size()&&vis[V[M].back()])
          V[M].pop_back();
        if(V[M].size()) break; else M--;
      }
      auto x=V[M].back();order[i]=x;rk[x]=i;vis[x]=1;
      for(auto y:E[x]) if(!vis[y])
        f[y]++,V[f[y]].push_back(y),M=max(M,f[y]);
    }
  }
  bool isChordal(){
    for(int i=0;i<=n;++i) vis[i]=stk[i]=0;
    for(int i=n;i>=1;--i){
      int top=0,cnt=0,m=n+1;
      for(auto x:E[order[i]]) if(rk[x] > i)
        stk[top++]=x,vis[x]=1,m=min(m,rk[x]);
      if(m==n+1) continue;
      for(auto x:E[order[m]]) if(vis[x]) ++cnt;
      for(int j=0;j<top;++j) vis[stk[j]]=0;
      if(cnt+1!=top) return 0;
    }
    return 1;
  }
  void getMaximalClique(){
    for(int i=n;i>=1;--i){
      int M=n+1,w=order[i],v=0;
      nsz[w]=0;isMaximalClique[w]=1;
      for(auto x:E[w]) if(rk[x]>i){
        nsz[w]++; if(rk[x]<M) M=rk[x],v=x;
      }
      if(v) isMaximalClique[v]&=nsz[v]+1>nsz[w];
    }
  }
  int getMaximumClique(){
    int res=0;
    for(int i=1;i<=n;++i) res=max(res,f[i]+1);
    return res;
  }
  int getMaximumIndependentSet(){
    for(int i=0;i<=n;++i) vis[i]=0;
    int res=0;
    for(int i=1;i<=n;++i) if(!vis[order[i]]){
      res++,vis[order[i]]=1;
      for(auto x:E[order[i]]) vis[x]=1;
    }
    return res;
  }
};
```

5.16 Matroid Intersection

```
/* Matroid Definition:
 * 1. Empty set is ind. 2. Subset of ind. set is ind.
 * 3. If set A, B are ind. and |A| < |B|,
 *    there exists x in B\A s.t. A U {x} is ind.
 * Max Weighted Matroid Intersection: (memorize testInd)
 * Let vertex weight l(x) = (x is chosen ? w(x) : -w(x))
 * Find shortest aug. path with SPFA, based on minimize
 * tie(sum of l(x),number of edges) on the path. */
struct MatroidIntersection {
  int n;          // Elem: bool chosen, int p, info...
  vector<Elem> GS; // Ground Set.
  vector<int> indSet; // Current chosen ind. set
  bool testInd1(int add){} // indSet U {a}
  bool testInd1(int add,int removed){} // ind\{r}U{a}
  bool testInd2(int add){}
  bool testInd2(int add,int removed){}
  bool augment(){ // prepareInd1(), prepareInd2();
    for(auto &x:GS) x.p=-2; // init l,dis,len,inque
    int ep=-3;queue<int> q;
    for(int i=0;i<n;++i) if(!GS[i].chosen&&testInd1(i))
      GS[i].p=-1,q.push(i);
    while(!q.empty()){ // bfs -> SPFA
      int cur=q.front(); q.pop();
      if(GS[cur].chosen){ // SPFA dont check .p != -2
        for(int nxt=0;nxt<n;++nxt){
          if(GS[nxt].chosen or GS[nxt].p!=-2) continue;
          if(!testInd1(nxt,cur)) continue;
          GS[nxt].p=cur; q.push(nxt);
        }
      }else{ // SPFA record nearest ep, dont break
        if(testInd2(cur)){ ep=cur; break; }
        for(auto nxt:indSet){
          if(GS[nxt].p!=-2 or !testInd2(cur,nxt))
            continue;
          GS[nxt].p=cur;q.push(nxt);
    } } }
    if(ep==-3) return false;
    do{ GS[ep].chosen^=1; ep=GS[ep].p; } while(ep!=-1);
    indSet.clear();
    for(int i=0;i<n;i++) if(GS[i].chosen)
      indSet.push_back(i);
    return true;
  }
  void solve(){ n=GS.size(); while(augment()); }
}MI;
```

5.17 DeBrujin 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;
}
```

5.18 Graph Hash

$$F_t(i) = (F_{t-1}(i)\times A + \sum_{i\to j} F_{t-1}(j)\times B + \sum_{j\to i} F_{t-1}(j)\times C + D\times(i=a)) \ mod\ P$$

for each node i, iterate t times. t, A, B, C, D, P are hash parameter

5.19 Graph Method

```
Manhattan MST
For each point, consider the points that surround it(8
    octants). Then, connect it with the closest point.
    For example, consider 45~90. For each point p, the
    closest point is min{x+y | x-y >= p.x-p.y, x >= p.x
    }. Finally, the answer is this new graphs(E=4N) MST.
```

6 String

6.1 PalTree

```
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 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];
  int _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,nmxz=-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]]]=nmxz+=neq;
    }
    sais(ns,nsa,p+nn,q+n,t+n,c+z,nn,nmxz+1);
    MAGIC(for(int i=nn-1;i>=0;i--) sa[--x[s[p[nsa[i
        ]]]]]=p[nsa[i]]);
  }
}sa;
int H[N],SA[N],RA[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);
  memcpy(H,sa.hei+1,len<<2); memcpy(SA,sa._sa+1,len<<2);
  for(int i=0;i<len;i++) RA[i]=sa.r[i]-1;
  // resulting height, sa array \in [0,len)
}
```

6.3 SuffixAutomata

```
// any path start from root forms a substring of S
// occurrence of P: iff SAM can run on input word P
// number of different substring: ds[1]-1
// total length of all different substring: dsl[1]
// max/min length of state i: mx[i]/mx[mom[i]]+1
// assume a run on input word P end at state i:
// number of occurrences of P: cnt[i]
// first occurrence position of P: fp[i]-|P|+1
// all position: !clone nodes in 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],clone[MXN]
  int newNode(){
    int res=++tot; fill(nxt[res],nxt[res]+33,0);
    mom[res]=mx[res]=0; // cnt=ds=dsl=fp=v=clone=0
    return res;
  }
  void init(){ tot=0;root=newNode();lst=root; }
  void push(int c){
    int p=lst,np=newNode(); // cnt[np]=1,clone[np]=0
    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],clone[nq]=1
        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=0;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(char *str){
    for(int i=0;str[i];i++) push(str[i]-'a');
  }
} sam;
```

6.4 Z Value

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

6.5 BWT

```
const int SIGMA=26; const char BASE='a';
struct BurrowsWheeler{
  vector<int> v[SIGMA];
  void BWT(char* ori,char* res){
    // make ori -> ori+ori and then build suffix array
  }
  void iBWT(char* ori,char* res){
    for(int i=0;i<SIGMA;i++) v[i].clear();
    int len=strlen(ori); vector<int> a;
```

```
    for(int i=0;i<len;i++) v[ori[i]-BASE].push_back(i);
    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;
```

6.6 ZValue Palindrome

```
void z_value_pal(char *s,int len,int *z){
  len=(len<<1)+1; z[0]=1;
  for(int i=len-1;i>=0;i--) s[i]=i&1?s[i>>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];
  }
}
```

6.7 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;
}
```

6.8 Cyclic LCS

```
const int L=0,LU=1,U=2,mov[3][2]={0,-1,-1,-1,-1,0};
int al,bl,dp[MAXL*2][MAXL];
char a[MAXL*2],b[MAXL*2],pred[MAXL*2][MAXL]; // 0-based
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;
  }
  }
  int clcs=0; // do cyclic lcs
  for(int i=0;i<al;i++){
    clcs=max(clcs,lcs_length(i)); reroot(i+1);
  }
  a[al]='\0'; // recover a
  return clcs;
}
```

7 Data Structure

7.1 Link-Cut Tree

```
const int MEM=100005;
struct Splay {
  static Splay nil,mem[MEM],*pmem; Splay *ch[2],*f;
  int val,rev,size; // int sum,vir,tot;
  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; }
  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;
    // sum={ch[0]->sum,ch[1]->sum,val}; tot={sum,vir};
    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; vector<Splay*> splayVec;
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);
}
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->vir+={x->ch[0]->tot}-{q->tot};
    x->setCh(q,1); q=x;
  }
  return q;
}
void chroot(Splay *x){ access(x); splay(x); x->rev^=1; }
void link(Splay *x,Splay *y){
  chroot(y); access(x); splay(x); y->f=x;
  // x->vir+={y->tot};
}
void cut_p(Splay *y){
  access(y);splay(y); y->ch[0]=y->ch[0]->f=nil;
}
void cut(Splay *x,Splay *y){ chroot(x); cut_p(y); }
Splay* get_root(Splay *x) {
  x=access(x);
  for(;x->ch[0]!=nil;x=x->ch[0]) x->push();
  splay(x); return x;
}
bool conn(Splay *x,Splay *y){
  return get_root(x)==get_root(y);
```

```
}
Splay* lca(Splay *x,Splay *y){
  access(x); return access(y);
}
/* query(Splay *x,Splay *y){ // path
  setroot(y),x=access(x); return x->size; // x->sum;
} */
/* query(Splay *x,Splay *y){ // path
  Splay *p=lca(x,y);
  return 1+p->ch[1]->size+(x!=p?x->size:0);
  // {p->val,p->ch[1]->sum,x!=p?x->sum:0};
} */
/* query(Splay *x){ // subtree
  access(x); return {x->val,x->vir};
} */
```

8 Others

8.1 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>=l
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-l;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+l]))
      now++;
    calc=sum[i+l]-pnt[now-1];
    if(ans.y*calc.x<ans.x*calc.y)
      ans=calc,st=pnt[now-1].x,ed=i+l;
  }
  return (double)(sum[ed].y-sum[st].y)/(sum[ed].x-sum[st
      ].x);
}
```

8.2 Exact Cover Set

```
// given n*m 0-1 matrix, find a set of rows s.t.
// for each column, there's exactly one 1
const int N=1024,M=1024,NM=((N+2)*(M+2)) // row,col
bool A[N][M]; // n*m 0-1 matrix
bool 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;
}
bool 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;
}
bool 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();
}
```

8.3 Binary Next Permutation

```
ull next_perm(ull v){
  ull t=v|(v-1);
  return (t+1)|(((~t&-~t)-1)>>(__builtin_ctzll(v)+1));
}
```

8.4 Hilbert Curve

```
long long hilbert(int n,int x,int y){
  long long res=0;
  for(int s=n/2;s;s>>=1){
    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;
}
```