.

Contents

1 Data structures

1.1 Segment tree

```
#define oper min
#define NEUT INF
struct STree { // segment tree for min over integers
  vector<int> st;int n;
  STree(int n): st(4*n+5,NEUT), n(n) {}
  void init(int k, int s, int e, int *a){
    if(s+1==e){st[k]=a[s];return;}
    int m=(s+e)/2;
    init(2*k,s,m,a);init(2*k+1,m,e,a);
    st[k]=oper(st[2*k],st[2*k+1]);
  }
  void upd(int k, int s, int e, int p, int v){
    if(s+1==e){st[k]=v;return;}
    int m=(s+e)/2;
    if(p<m)upd(2*k,s,m,p,v);
    else upd(2*k+1,m,e,p,v);
    st[k]=oper(st[2*k],st[2*k+1]);
  }
  int query(int k, int s, int e, int a, int b){
    if(s>=b||e<=a)return NEUT;
    if(s>=a&&e<=b)return st[k];
    int m=(s+e)/2;
    return oper(query(2*k,s,m,a,b),query(2*k+1,m,e,a,b));
  }
  void init(int *a){init(1,0,n,a);}
  void upd(int p, int v){upd(1,0,n,p,v);}
  int query(int a, int b){return query(1,0,n,a,b);}
}; // usage: STree rmq(n);rmq.init(x);rmq.upd(i,v);rmq.query(s,e);
```

1.2 Segment tree - Lazy propagation

```
struct STree { // example: range sum with range addition
  vector<int> st,lazy;int n;
  STree(int n): st(4*n+5,0), lazy(4*n+5,0), n(n) {}
  void init(int k, int s, int e, int *a){
    lazy[k]=0;  // lazy neutral element
    if(s+1==e){st[k]=a[s];return;}
    int m=(s+e)/2;
    init(2*k,s,m,a);init(2*k+1,m,e,a);
    st[k]=st[2*k]+st[2*k+1]; // operation
  }
  void push(int k, int s, int e){
    if(!lazy[k])return; // if neutral, nothing to do
    st[k]+=(e-s)*lazy[k]; // update st according to lazy
    if(s+1<e){ // propagate to children
      lazy[2*k]+=lazy[k];
      lazy[2*k+1]+=lazy[k];
    }
    lazy[k]=0; // clear node lazy
  }
  void upd(int k, int s, int e, int a, int b, int v){
    push(k,s,e);
    if(s>=b||e<=a)return;
    if(s>=a&&e<=b){
      lazy[k]+=v; // accumulate lazy
      push(k,s,e);return;
    }
    int m=(s+e)/2;
    upd(2*k,s,m,a,b,v);upd(2*k+1,m,e,a,b,v);
    st[k]=st[2*k]+st[2*k+1]; // operation
  }
  int query(int k, int s, int e, int a, int b){
    if(s>=b||e<=a)return 0; // operation neutral
    push(k,s,e);
    if(s>=a&&e<=b)return st[k];
    int m=(s+e)/2;
    return query(2*k,s,m,a,b)+query(2*k+1,m,e,a,b); // operation
  }
  void init(int *a){init(1,0,n,a);}
  void upd(int a, int b, int v){upd(1,0,n,a,b,v);}
  int query(int a, int b){return query(1,0,n,a,b);}
}; // usage: STree rmq(n);rmq.init(x);rmq.upd(s,e,v);rmq.query(s,e);
```

1.3 Segment tree - Persistence

```
#define oper min
#define NEUT INF
struct STree { // persistent segment tree for min over integers
  vector<int> st,l,r;int n,rt,sz;
  STree(int n): st(24*n,NEUT),l(24*n,0),r(24*n,0),n(n),rt(0),sz(1){}
  // be careful with memory! 4*n+q*log(n) . 24*n should be enough
  int init(int s, int e, int *a){ // not necessary in most cases
    int k=sz++;
    if(s+1==e){st[k]=a[s];return k;}
    int m=(s+e)/2;
    l[k]=init(s,m,a);r[k]=init(m,e,a);
    st[k]=oper(st[l[k]],st[r[k]]);
    return k;
  }
  int upd(int k, int s, int e, int p, int v){
    int nk=sz++;l[nk]=l[k];r[nk]=r[k];
    if(s+1==e){st[nk]=v;return nk;}
    int m=(s+e)/2;
    if(p<m)l[nk]=upd(l[k],s,m,p,v);
    else r[nk]=upd(r[k],m,e,p,v);
    st[nk]=oper(st[l[nk]],st[r[nk]]);
    return nk;
  }
  int query(int k, int s, int e, int a, int b){
    if(s>=b||e<=a)return NEUT;
    if(s>=a&&e<=b)return st[k];
    int m=(s+e)/2;
    return oper(query(l[k],s,m,a,b),query(r[k],m,e,a,b));
  }
  int init(int *a){return init(0,n,a);}
  int upd(int k, int p, int v){return rt=upd(k,0,n,p,v);}
  int upd(int p, int v){return upd(rt,p,v);} // update on last root
  int query(int k, int a, int b){return query(k,0,n,a,b);}
}; // usage: STree rmq(n);root=rmq.init(x);new_root=rmq.upd(root,i,v);rmq.query(root,s,e);
```

1.4 Segment tree - 2D

```
int n,m;
int a[MAXN][MAXN],st[2*MAXN][2*MAXN];
void build(){
  forn(i,n)forn(j,m)st[i+n][j+m]=a[i][j];
  forn(i,n)for(int j=m-1;j;--j)
    st[i+n][j]=op(st[i+n][j<<1],st[i+n][j<<1|1]);
  for(int i=n-1;i;--i)forn(j,2*m)
    st[i][j]=op(st[i<<1][j],st[i<<1|1][j]);
}
void upd(int x, int y, int v){
  st[x+n][y+m]=v;
  for(int j=y+m;j>1;j>>=1)st[x+n][j>>1]=op(st[x+n][j],st[x+n][j^1]);
  for(int i=x+n;i>1;i>>=1)for(int j=y+m;j;j>>=1)
    st[i>>1][j]=op(st[i][j],st[i^1][j]);
}
int query(int x0, int x1, int y0, int y1){
  int r=NEUT;
  for(int i0=x0+n,i1=x1+n;i0<i1;i0>>=1,i1>>=1){
    int t[4],q=0;
    if(i0&1)t[q++]=i0++;
    if(i1&1)t[q++]=--i1;
    forn(k,q)for(int j0=y0+m,j1=y1+m;j0<j1;j0>>=1,j1>>=1){
      if(j0&1)r=op(r,st[t[k]][j0++]);
      if(j1&1)r=op(r,st[t[k]][--j1]);
    }
  }
  return r;
}
```

1.5 Sparse table (static RMQ)

```
#define oper min
int st[K][1<<K];int n;  // K such that 2^K>n
void st_init(int *a){
  forn(i,n)st[0][i]=a[i];
  forr(k,1,K)forn(i,n-(1<<k)+1)
    st[k][i]=oper(st[k-1][i],st[k-1][i+(1<<(k-1))]);
}
int st_query(int s, int e){
  int k=31-__builtin_clz(e-s);
  return oper(st[k][s],st[k][e-(1<<k)]);
}
```

1.6 Wavelet tree

```
struct WT {
  vector<int> wt[1<<20];int n;
  void init(int k, int s, int e){
    if(s+1==e)return;
    wt[k].clear();wt[k].pb(0);
    int m=(s+e)/2;
    init(2*k,s,m);init(2*k+1,m,e);
  }
  void add(int k, int s, int e, int v){
    if(s+1==e)return;
    int m=(s+e)/2;
    if(v<m)wt[k].pb(wt[k].back()),add(2*k,s,m,v);
    else wt[k].pb(wt[k].back()+1),add(2*k+1,m,e,v);
  }
  int query0(int k, int s, int e, int a, int b, int i){
    if(s+1==e)return s;
    int m=(s+e)/2;
    int q=(b-a)-(wt[k][b]-wt[k][a]);
    if(i<q)return query0(2*k,s,m,a-wt[k][a],b-wt[k][b],i);
    else return query0(2*k+1,m,e,wt[k][a],wt[k][b],i-q);
  }
  void upd(int k, int s, int e, int i){
    if(s+1==e)return;
    int m=(s+e)/2;
    int v0=wt[k][i+1]-wt[k][i],v1=wt[k][i+2]-wt[k][i+1];
    if(!v0&&!v1)upd(2*k,s,m,i-wt[k][i]);
    else if(v0&&v1)upd(2*k+1,m,e,wt[k][i]);
    else if(v0)wt[k][i+1]--;
    else wt[k][i+1]++;
  }
  void init(int _n){n=_n;init(1,0,n);} // (values in range [0,n))
  void add(int v){add(1,0,n,v);}
  int query0(int a, int b, int i){ // ith element in range [a,b)
    return query0(1,0,n,a,b,i);    // (if it was sorted)
  }
  void upd(int i){ // swap positions i,i+1
    upd(1,0,n,i);
  }
};
```

1.7 STL extended set

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int,null_type,less<int>,rb_tree_tag,tree_order_statistics_node_update> ordered_set;
// find_by_order(i) -> iterator to ith element
// order_of_key(k) -> position (int) of lower_bound of k
```

1.8 Treap (as BST)

```
typedef struct item *pitem;
struct item {
  int key,pr,cnt;
  pitem l,r;
  item(int key):key(key),pr(rand()),cnt(1),l(0),r(0) {}
};
int cnt(pitem t){return t?t->cnt:0;}
void upd_cnt(pitem t){if(t)t->cnt=cnt(t->l)+cnt(t->r)+1;}
void split(pitem t, int key, pitem& l, pitem& r){ // l: < key, r: >= key
  if(!t)l=r=0;
  else if(key<t->key)split(t->l,key,l,t->l),r=t;
  else split(t->r,key,t->r,r),l=t;
  upd_cnt(t);
}
void insert(pitem& t, pitem it){
  if(!t)t=it;
  else if(it->pr>t->pr)split(t,it->key,it->l,it->r),t=it;
  else insert(it->key<t->key?t->l:t->r,it);
  upd_cnt(t);
}
void merge(pitem& t, pitem l, pitem r){
  if(!l||!r)t=l?l:r;
  else if(l->pr>r->pr)merge(l->r,l->r,r),t=l;
  else merge(r->l,l,r->l),t=r;
  upd_cnt(t);
}
void erase(pitem& t, int key){
  if(t->key==key)merge(t,t->l,t->r);
  else erase(key<t->key?t->l:t->r,key);
  upd_cnt(t);
}
pitem kth(pitem t, int k){
  if(!t)return 0;
  if(k==cnt(t->l))return t;
  return k<cnt(t->l)?kth(t->l,k):kth(t->r,k-cnt(t->l)-1);
}
pair<int,int> lb(pitem t, int key){ // position and value of lower_bound
  if(!t)return mp(0,1<<30); // (special value)
  if(key>t->key){
    auto w=lb(t->r,key);w.fst+=cnt(t->l)+1;return w;
  }
  auto w=lb(t->l,key);
  if(w.fst==cnt(t->l))w.snd=t->key;
  return w;
}
```

1.9 Treap (implicit key)

```
// example that supports range reverse and addition updates, and range sum query
// (commented parts are specific to this  problem)
```

```
typedef struct item *pitem;
struct item {
  int cnt,pr,val;
//  int sum; // (paramters for range query)
//  bool rev;int add; // (parameters for lazy prop)
  pitem l,r;
  item(int val): pr(rand()),cnt(1),val(val),l(0),r(0)/*,sum(val),rev(0),add(0)*/ {}
};
void push(pitem it){
  if(it){
    /*if(it->rev){
      swap(it->l,it->r);
      if(it->l)it->l->rev^=true;
      if(it->r)it->r->rev^=true;
      it->rev=false;
    }
    it->val+=it->add;it->sum+=it->cnt*it->add;
    if(it->l)it->l->add+=it->add;
    if(it->r)it->r->add+=it->add;
    it->add=0;*/
  }
}
int cnt(pitem t){return t?t->cnt:0;}
// int sum(pitem t){return t?push(t),t->sum:0;}
void upd_cnt(pitem t){
  if(t){
    t->cnt=cnt(t->l)+cnt(t->r)+1;
    // t->sum=t->val+sum(t->l)+sum(t->r);
  }
}
void merge(pitem& t, pitem l, pitem r){
  push(l);push(r);
  if(!l||!r)t=l?l:r;
  else if(l->pr>r->pr)merge(l->r,l->r,r),t=l;
  else merge(r->l,l,r->l),t=r;
  upd_cnt(t);
}
void split(pitem t, pitem& l, pitem& r, int sz){ // sz:desired size of l
  if(!t){l=r=0;return;}
  push(t);
  if(sz<=cnt(t->l))split(t->l,l,t->l,sz),r=t;
  else split(t->r,t->r,r,sz-1-cnt(t->l)),l=t;
  upd_cnt(t);
}
void output(pitem t){ // useful for debugging
  if(!t)return;
  push(t);
  output(t->l);printf("␣%d",t->val);output(t->r);
}
// use merge and split for range updates and queries
```

1.10 Convex hull trick (static)

```
typedef ll tc;
struct Line{tc m,h;};
struct CHT { // for minimum (for maximum just change the sign of lines)
  vector<Line> c;
  int pos=0;
  tc in(Line a, Line b){
    tc x=b.h-a.h,y=a.m-b.m;
    return x/y+(x%y?!((x>0)^(y>0)):0); // ==ceil(x/y)
  }
  void add(tc m, tc h){ // m's should be non increasing
    Line l=(Line){m,h};
    if(c.size()&&m==c.back().m){
      l.h=min(h,c.back().h);c.pop_back();if(pos)pos--;
    }
    while(c.size()>1&&in(c.back(),l)<=in(c[c.size()-2],c.back())){
      c.pop_back();if(pos)pos--;
    }
    c.pb(l);
  }
  inline bool fbin(tc x, int m){return in(c[m],c[m+1])>x;}
  tc eval(tc x){
    // O(log n) query:
    int s=0,e=c.size();
    while(e-s>1){int m=(s+e)/2;
      if(fbin(x,m-1))e=m;
      else s=m;
    }
    return c[s].m*x+c[s].h;
    // O(1) query (for ordered x's):
    while(pos>0&&fbin(x,pos-1))pos--;
    while(pos<c.size()-1&&!fbin(x,pos))pos++;
    return c[pos].m*x+c[pos].h;
  }
};
```

1.11 Convex hull trick (dynamic)

```
typedef ll tc;
const tc is_query=-(1LL<<62); // special value for query
struct Line {
  tc m,b;
  mutable multiset<Line>::iterator it,end;
  const Line* succ(multiset<Line>::iterator it) const {
    return (++it==end? NULL : &*it);}
  bool operator<(const Line& rhs) const {
    if(rhs.b!=is_query)return m<rhs.m;
    const Line *s=succ(it);
    if(!s)return 0;
    return b-s->b<(s->m-m)*rhs.m;
  }
};
struct HullDynamic : public multiset<Line> { // for maximum
  bool bad(iterator y){
    iterator z=next(y);
    if(y==begin()){
      if(z==end())return false;
      return y->m==z->m&&y->b<=z->b;
    }
    iterator 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);
  }
  iterator next(iterator y){return ++y;}
  iterator prev(iterator y){return --y;}
  void add(tc m, tc b){
    iterator y=insert((Line){m,b});
    y->it=y;y->end=end();
    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));
  }
  tc eval(tc x){
    Line l=*lower_bound((Line){x,is_query});
    return l.m*x+l.b;
  }
};
```

1.12 Max Queue

```
struct MaxQueue { // for min, change < with >.
  deque<int> d; queue<int> q;
  void push(int v){while(sz(d)&&d.back()<v)d.pop_back();d.pb(v);q.push(v);}
  void pop(){if(sz(d)&&d.front()==q.front())d.pop_front();q.pop();}
  int getMax(){return sz(d)?d.front():NEUT;}
};
```

1.13 Union Find

```
int uf[MAXN];
void uf_init(){memset(uf,-1,sizeof(uf));}
int uf_find(int x){return uf[x]<0?x:uf[x]=uf_find(uf[x]);}
bool uf_join(int x, int y){
  x=uf_find(x);y=uf_find(y);
  if(x==y)return false;
  if(uf[x]>uf[y])swap(x,y);
  uf[x]+=uf[y];uf[y]=x;
  return true;
}
```

2 Graphs

2.1 Bellman-Ford

```
int n;
vector<pair<int,int> > g[MAXN]; // u->[(v,cost)]
ll dist[MAXN];
void bford(int src){ // O(nm)
  fill(dist,dist+n,INF);dist[src]=0;
  forn(_,n-1)forn(x,n)if(dist[x]!=INF)for(auto t:g[x]){
    dist[t.fst]=min(dist[t.fst],dist[x]+t.snd);
  }
  forn(x,n)if(dist[x]!=INF)for(auto t:g[x]){
    if(dist[t.fst]>dist[x]+t.snd){
      // neg cycle: all nodes reachable from t.fst have -INF distance
      // to reconstruct neg cycle: save "prev" of each node, go up from t.fst until repeating a node. this node and all nodes between the two occurences form a neg cycle
    }
  }
}
```

2.2 Floyd-Warshall

```
// g[i][j]: weight of edge (i, j) or INF if there's no edge
// g[i][i]=0
ll g[MAXN][MAXN];int n;
void floyd(){ // O(n^3) . Replaces g with min distances
  forn(k,n)forn(i,n)if(g[i][k]<INF)forn(j,n)if(g[k][j]<INF)
    g[i][j]=min(g[i][j],g[i][k]+g[k][j]);
}
bool inNegCycle(int v){return g[v][v]<0;}
bool hasNegCycle(int a, int b){ // true iff there's neg cycle in between
  forn(i,n)if(g[a][i]<INF&&g[i][b]<INF&&g[i][i]<0)return true;
  return false;
}
```

2.3 Strongly connected components (+ 2-SAT)

```
// MAXN: max number of nodes or 2 * max number of variables (2SAT)
bool truth[MAXN]; // truth[cmp[i]]=value of variable i (2SAT)
int nvar;int neg(int x){return MAXN-1-x;} // (2SAT)
vector<int> g[MAXN];
int n,lw[MAXN],idx[MAXN],qidx,cmp[MAXN],qcmp;
stack<int> st;
void tjn(int u){
  lw[u]=idx[u]=++qidx;
  st.push(u);cmp[u]=-2;
  for(int v:g[u]){
    if(!idx[v]||cmp[v]==-2){
      if(!idx[v]) tjn(v);
      lw[u]=min(lw[u],lw[v]);
    }
  }
  if(lw[u]==idx[u]){
    int x;
    do{x=st.top();st.pop();cmp[x]=qcmp;}while(x!=u);
    truth[qcmp]=(cmp[neg(u)]<0); // (2SAT)
    qcmp++;
  }
}
void scc(){
  memset(idx,0,sizeof(idx));qidx=0;
  memset(cmp,-1,sizeof(cmp));qcmp=0;
  forn(i,n)if(!idx[i])tjn(i);
}
// Only for 2SAT:
void addor(int a, int b){g[neg(a)].pb(b);g[neg(b)].pb(a);}
bool satisf(int _nvar){
  nvar=_nvar;n=MAXN;scc();
  forn(i,nvar)if(cmp[i]==cmp[neg(i)])return false;
  return true;
}
```

2.4 Articulation - Bridges - Biconnected

```
vector<int> g[MAXN];int n;
struct edge {int u,v,comp;bool bridge;};
vector<edge> e;
void add_edge(int u, int v){
  g[u].pb(e.size());g[v].pb(e.size());
  e.pb((edge){u,v,-1,false});
}
int D[MAXN],B[MAXN],T;
int nbc;  // number of biconnected components
int art[MAXN];  // articulation point iff !=0
stack<int> st;  // only for biconnected
void dfs(int u,int pe){
  B[u]=D[u]=T++;
  for(int ne:g[u])if(ne!=pe){
    int v=e[ne].u^e[ne].v^u;
    if(D[v]<0){
      st.push(ne);dfs(v,ne);
      if(B[v]>D[u])e[ne].bridge = true; // bridge
      if(B[v]>=D[u]){
        art[u]++; // articulation
        int last; // start biconnected
        do {
          last=st.top();st.pop();
          e[last].comp=nbc;
        } while(last!=ne);
        nbc++;    // end biconnected
      }
      B[u]=min(B[u],B[v]);
    }
    else if(D[v]<D[u])st.push(ne),B[u]=min(B[u],D[v]);
  }
}
void doit(){
  memset(D,-1,sizeof(D));memset(art,0,sizeof(art));
  nbc=T=0;
  forn(i,n)if(D[i]<0)dfs(i,-1),art[i]--;
}
```

2.5 Chu-Liu (minimum spanning arborescence)

```
typedef ll tw;const tw INF=1LL<<60;
struct edge {int src,dst;tw w;};
struct ChuLiu {
  int n,r;tw cost;bool found;
  vector<int> no,pr,mark;
  vector<vector<int> > comp,nx;
  vector<tw> mcost;
  vector<vector<edge> > h;
  ChuLiu(int n):n(n),h(n){}
  void add_edge(int x, int y, tw w){h[y].pb((edge){x,y,w});}
  void visit(int v, int s){
    if(mark[v]){
      vector<int> temp=no;found=true;
      do {
        cost+=mcost[v];v=pr[v];
        if(v!=s)while(comp[v].size()>0){
          no[comp[v].back()]=s;
          comp[s].pb(comp[v].back());
          comp[v].pop_back();
        }
      }while(v!=s);
      for(int j:comp[s])if(j!=r)for(edge& e:h[j])
        if(no[e.src]!=s)e.w-=mcost[temp[j]];
    }
    mark[v]=true;
    for(int i:nx[v])if(no[i]!=no[v]&&pr[no[i]]==v)
      if(!mark[no[i]]||i==s)
        visit(i,s);
  }
  tw doit(int _r){ // r: root (O(nm))
    r=_r;
    no.resize(n);comp.clear();comp.resize(n);
    forn(x,n)comp[x].pb(no[x]=x);
    for(cost=0;;){
      pr.clear();pr.resize(n,-1);
      mcost=vector<tw>(n,INF);
      forn(j,n)if(j!=r)for(edge e:h[j])
        if(no[e.src]!=no[j]&&e.w<mcost[no[j]])
          mcost[no[j]]=e.w,pr[no[j]]=no[e.src];
      nx.clear();nx.resize(n);
      forn(x,n)if(pr[x]>=0)nx[pr[x]].pb(x);
      bool stop=true;
      mark.clear();mark.resize(n);
      forn(x,n)if(x!=r&&!mark[x]&&!comp[x].empty()){
        found=false;visit(x,x);
        if(found)stop=false;
      }
      if(stop){
        forn(x,n)if(pr[x]>=0)cost+=mcost[x];
        return cost;
      }
    }
  }
};
```

2.6 LCA - Binary Lifting

```
vector<int> g[1<<K];int n;  // K such that 2^K>=n
int F[K][1<<K],D[1<<K];
void lca_dfs(int x){
  for(int y:g[x]){if(y==F[0][x])continue;
    F[0][y]=x;D[y]=D[x]+1;lca_dfs(y);
  }
}
void lca_init(){
  D[0]=0;F[0][0]=-1;
  lca_dfs(0);
  forr(k,1,K)forn(x,n)
    if(F[k-1][x]<0)F[k][x]=-1;
    else F[k][x]=F[k-1][F[k-1][x]];
}
int lca(int x, int y){
  if(D[x]<D[y])swap(x,y);
  for(int k=K-1;k>=0;--k)if(D[x]-(1<<k)>=D[y])x=F[k][x];
  if(x==y)return x;
  for(int k=K-1;k>=0;--k)if(F[k][x]!=F[k][y])x=F[k][x],y=F[k][y];
  return F[0][x];
}
```

2.7 Heavy-Light decomposition

```
vector<int> g[MAXN];
int wg[MAXN],dad[MAXN],dep[MAXN]; // weight,father,depth
void dfs1(int x){
  wg[x]=1;
  for(int y:g[x])if(y!=dad[x]){
    dad[y]=x;dep[y]=dep[x]+1;dfs1(y);
    wg[x]+=wg[y];
  }
}
int curpos,pos[MAXN],head[MAXN];
void hld(int x, int c){
  if(c<0)c=x;
  pos[x]=curpos++;head[x]=c;
  int mx=-1;
  for(int y:g[x])if(y!=dad[x]&&(mx<0||wg[mx]<wg[y]))mx=y;
  if(mx>=0)hld(mx,c);
  for(int y:g[x])if(y!=mx&&y!=dad[x])hld(y,-1);
}
void hld_init(){dad[0]=-1;dep[0]=0;dfs1(0);curpos=0;hld(0,-1);}
int query(int x, int y, STree& rmq){
  int r=NEUT;
  while(head[x]!=head[y]){
    if(dep[head[x]]>dep[head[y]])swap(x,y);
    r=oper(r,rmq.query(pos[head[y]],pos[y]+1));
    y=dad[head[y]];
  }
  if(dep[x]>dep[y])swap(x,y); // now x is lca
  r=oper(r,rmq.query(pos[x],pos[y]+1));
  return r;
}
// for updating: rmq.upd(pos[x],v);
```

2.8 Centroid decomposition

```
vector<int> g[MAXN];int n;
bool tk[MAXN];
int fat[MAXN]; // father in centroid decomposition
int szt[MAXN]; // size of subtree
int calcsz(int x, int f){
  szt[x]=1;
  for(auto y:g[x])if(y!=f&&!tk[y])szt[x]+=calcsz(y,x);
  return szt[x];
}
void cdfs(int x=0, int f=-1, int sz=-1){ // O(nlogn)
  if(sz<0)sz=calcsz(x,-1);
  for(auto y:g[x])if(!tk[y]&&szt[y]*2>=sz){
    szt[x]=0;cdfs(y,f,sz);return;
  }
  tk[x]=true;fat[x]=f;
  for(auto y:g[x])if(!tk[y])cdfs(y,x);
}
void centroid(){memset(tk,false,sizeof(tk));cdfs();}
```

2.9 Eulerian path

```
// Directed version (uncomment commented code for undirected)
struct edge {
  int y;
//  list<edge>::iterator rev;
  edge(int y):y(y){}
};
list<edge> g[MAXN];
void add_edge(int a, int b){
  g[a].push_front(edge(b));//auto ia=g[a].begin();
//  g[b].push_front(edge(a));auto ib=g[b].begin();
//  ia->rev=ib;ib->rev=ia;
}
vector<int> p;
void go(int x){
  while(g[x].size()){
    int y=g[x].front().y;
    //g[y].erase(g[x].front().rev);
    g[x].pop_front();
    go(y);
  }
  p.push_back(x);
}
vector<int> get_path(int x){ // get a path that begins in x
// check that a path exists from x before calling to get_path!
  p.clear();go(x);reverse(p.begin(),p.end());
  return p;
}
```

2.10 Dynamic connectivity

```
struct UnionFind {
  int n,comp;
  vector<int> uf,si,c;
  UnionFind(int n=0):n(n),comp(n),uf(n),si(n,1){
    forn(i,n)uf[i]=i;}
  int find(int x){return x==uf[x]?x:find(uf[x]);}
  bool join(int x, int y){
    if((x=find(x))==(y=find(y)))return false;
    if(si[x]<si[y])swap(x,y);
    si[x]+=si[y];uf[y]=x;comp--;c.pb(y);
    return true;
  }
  int snap(){return c.size();}
  void rollback(int snap){
    while(c.size()>snap){
      int x=c.back();c.pop_back();
      si[uf[x]]-=si[x];uf[x]=x;comp++;
    }
  }
};
enum {ADD,DEL,QUERY};
struct Query {int type,x,y;};
struct DynCon {
  vector<Query> q;
  UnionFind dsu;
  vector<int> mt;
  map<pair<int,int>,int> last;
  DynCon(int n):dsu(n){}
  void add(int x, int y){
    if(x>y)swap(x,y);
    q.pb((Query){ADD,x,y});mt.pb(-1);last[mp(x,y)]=q.size()-1;
  }
  void remove(int x, int y){ // the edge to remove must exist
    if(x>y)swap(x,y);
    q.pb((Query){DEL,x,y});
    int pr=last[mp(x,y)];mt[pr]=q.size()-1;mt.pb(pr);
  }
  void query(){q.pb((Query){QUERY,-1,-1});mt.pb(-1);}
  void process(){ // answers all queries in order
    if(!q.size())return;
    forn(i,q.size())if(q[i].type==ADD&&mt[i]<0)mt[i]=q.size();
    go(0,q.size());
  }
  void go(int s, int e){
    if(s+1==e){
      if(q[s].type==QUERY) // answer query using DSU
        printf("%d\n",dsu.comp);
      return;
    }
    int k=dsu.snap(),m=(s+e)/2;
    for(int i=e-1;i>=m;--i)if(mt[i]>=0&&mt[i]<s)dsu.join(q[i].x,q[i].y);
    go(s,m);dsu.rollback(k);
    for(int i=m-1;i>=s;--i)if(mt[i]>=e)dsu.join(q[i].x,q[i].y);
    go(m,e);dsu.rollback(k);
  }
};
```

2.11 Edmond's blossom (matching in general graphs)

```
vector<int> g[MAXN];
int n,m,mt[MAXN],qh,qt,q[MAXN],ft[MAXN],bs[MAXN];
bool inq[MAXN],inb[MAXN],inp[MAXN];
int lca(int root, int x, int y){
  memset(inp,0,sizeof(inp));
  while(1){
    inp[x=bs[x]]=true;
    if(x==root)break;
    x=ft[mt[x]];
  }
  while(1){
    if(inp[y=bs[y]])return y;
    else y=ft[mt[y]];
  }
}
void mark(int z, int x){
  while(bs[x]!=z){
    int y=mt[x];
    inb[bs[x]]=inb[bs[y]]=true;
    x=ft[y];
    if(bs[x]!=z)ft[x]=y;
  }
}
void contr(int s, int x, int y){
  int z=lca(s,x,y);
  memset(inb,0,sizeof(inb));
  mark(z,x);mark(z,y);
  if(bs[x]!=z)ft[x]=y;
  if(bs[y]!=z)ft[y]=x;
  forn(x,n)if(inb[bs[x]]){
    bs[x]=z;
    if(!inq[x])inq[q[++qt]=x]=true;
  }
}
```

```
int findp(int s){
  memset(inq,0,sizeof(inq));
  memset(ft,-1,sizeof(ft));
  forn(i,n)bs[i]=i;
  inq[q[qh=qt=0]=s]=true;
  while(qh<=qt){
    int x=q[qh++];
    for(int y:g[x])if(bs[x]!=bs[y]&&mt[x]!=y){
      if(y==s||mt[y]>=0&&ft[mt[y]]>=0)contr(s,x,y);
      else if(ft[y]<0){
        ft[y]=x;
        if(mt[y]<0)return y;
        else if(!inq[mt[y]])inq[q[++qt]=mt[y]]=true;
      }
    }
  }
  return -1;
}
int aug(int s, int t){
  int x=t,y,z;
  while(x>=0){
    y=ft[x];
    z=mt[y];
    mt[y]=x;mt[x]=y;
    x=z;
  }
  return t>=0;
}
int edmonds(){ // O(n^2 m)
  int r=0;
  memset(mt,-1,sizeof(mt));
  forn(x,n)if(mt[x]<0)r+=aug(x,findp(x));
  return r;
}
```

3 Math

3.1 Identities

$C_n = \frac{2(2n-1)}{n+1} C_{n-1}$

$C_n = \frac{1}{n+1}\binom{2n}{n}$

$C_n \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$

$\sigma(n) = O(\log(\log(n)))$ (number of divisors of n)

$F_{2n+1} = F_n^2 + F_{n+1}^2$

$F_{2n} = F_{n+1}^2 - F_{n-1}^2$

$\sum_{i=1}^{n} F_i = F_{n+2} - 1$

$F_{n+i}F_{n+j} - F_nF_{n+i+j} = (-1)^n F_iF_j$

(Möbius Inv. Formula) Let $g(n) = \sum_{d|n} f(d)$, then $f(n) = \sum d \mid ng(d)\mu\left(\frac{n}{d}\right))$.

3.2 Theorems

```
(Tutte) A graph, G = (V, E), has a perfect matching if and only if for
    every subset U of V, the subgraph induced by V - U has at most |U|
    connected components with an odd number of vertices.
Petersens Theorem. Every cubic, bridgeless graph contains a perfect
    matching.
(Dilworth) In any finite partially ordered set, the maximum number of
    elements in any antichain equals the minimum number of chains in any
    partition of the set into chains
Pick: A=I+B/2-1  (area of polygon, points inside, points on border)
```

3.3 Integer floor division

```
void floordiv(ll x, ll y, ll& q, ll& r) { // (for negative x)
  q=x/y;r=x%y;
  if((r!=0)&&((r<0)!=(y<0)))q--,r+=y;
}
```

3.4 Extended Euclid

```
ll euclid(ll a, ll b, ll& x, ll& y){ // a*(x+k*(b/d))+b*(y-k*(a/d))=d
  if(!b){x=1;y=0;return a;}          // (for any k)
  ll d=euclid(b,a%b,x,y);
  ll t=y;y=x-(a/b)*y;x=t;
  return d;
}
```

3.5 Pollard's rho

```
ll gcd(ll a, ll b){return a?gcd(b%a,a):b;}
ull mulmod(ull a, ull b, ull m){ // 0 <= a, b < m
   long double x; ull c; ll r;
   x = a; c = x * b / m;
   r = (ll)(a * b - c * m) % (ll)m;
   return r < 0 ? r + m : r;
}
ll expmod(ll b, ll e, ll m){
  if(!e)return 1;
  ll q=expmod(b,e/2,m);q=mulmod(q,q,m);
  return e&1?mulmod(b,q,m):q;
}
bool is_prime_prob(ll n, int a){
  if(n==a)return true;
  ll s=0,d=n-1;
  while(d%2==0)s++,d/=2;
  ll x=expmod(a,d,n);
  if((x==1)||(x+1==n))return true;
  forn(_,s-1){
    x=mulmod(x,x,n);
    if(x==1)return false;
    if(x+1==n)return true;
  }
  return false;
}
bool rabin(ll n){ // true iff n is prime
  if(n==1)return false;
  int ar[]={2,3,5,7,11,13,17,19,23};
  forn(i,9)if(!is_prime_prob(n,ar[i]))return false;
  return true;
}
ll rho(ll n){
    if(!(n&1))return 2;
    ll x=2,y=2,d=1;
    ll c=rand()%n+1;
    while(d==1){
        x=(mulmod(x,x,n)+c)%n;
        y=(mulmod(y,y,n)+c)%n;
        y=(mulmod(y,y,n)+c)%n;
        if(x>=y)d=gcd(x-y,n);
        else d=gcd(y-x,n);
    }
    return d==n?rho(n):d;
}
void fact(ll n, map<ll,int>& f){ //O (lg n)^3
  if(n==1)return;
  if(rabin(n)){f[n]++;return;}
  ll q=rho(n);fact(q,f);fact(n/q,f);
}
```

3.6 Simpson's rule

```
double integrate(double f(double), double a, double b, int n=10000){
  double r=0,h=(b-a)/n,fa=f(a),fb;
  forn(i,n){
    fb=f(a+h*(i+1));
    r+=fa+4*f(a+h*(i+0.5))+fb;fa=fb;
  }
  return r*h/6.;
}
```

3.7 Polynomials

```
typedef int tp; // type of polynomial
template<class T=tp>
struct poly {  // poly<> : 1 variable, poly<poly<>>: 2 variables, etc.
  vector<T> c;
  T& operator[](int k){return c[k];}
  poly(vector<T>& c):c(c){}
  poly(initializer_list<T> c):c(c){}
  poly(int k):c(k){}
  poly(){}
  poly operator+(poly<T> o){
    int m=c.size(),n=o.c.size();
    poly res(max(m,n));
    forn(i,m)res[i]=res[i]+c[i];
    forn(i,n)res[i]=res[i]+o.c[i];
    return res;
  }
  poly operator*(tp k){
    poly res(c.size());
    forn(i,c.size())res[i]=c[i]*k;
    return res;
  }
  poly operator*(poly o){
    int m=c.size(),n=o.c.size();
    poly res(m+n-1);
    forn(i,m)forn(j,n)res[i+j]=res[i+j]+c[i]*o.c[j];
    return res;
  }
  poly operator-(poly<T> o){return *this+(o*-1);}
  T operator()(tp v){
    T sum(0);
    for(int i=c.size()-1;i>=0;--i)sum=sum*v+c[i];
    return sum;
  }
};
// example: p(x,y)=2*x^2+3*x*y-y+4
// poly<poly<>> p={{4,-1},{0,3},{2}}
// printf("%d\n",p(2)(3)) // 27 (p(2,3))
set<tp> roots(poly<> p){ // only for integer polynomials
  set<tp> r;
  while(!p.c.empty()&&!p.c.back())p.c.pop_back();
  if(!p(0))r.insert(0);
  if(p.c.empty())return r;
  tp a0=0,an=abs(p[p.c.size()-1]);
  for(int k=0;!a0;a0=abs(p[k++]));
  vector<tp> ps,qs;
  forr(i,1,sqrt(a0)+1)if(a0%i==0)ps.pb(i),ps.pb(a0/i);
  forr(i,1,sqrt(an)+1)if(an%i==0)qs.pb(i),qs.pb(an/i);
  for(auto pt:ps)for(auto qt:qs)if(pt%qt==0){
    tp x=pt/qt;
    if(!p(x))r.insert(x);
    if(!p(-x))r.insert(-x);
  }
  return r;
}
pair<poly<>,tp> ruffini(poly<> p, tp r){ // returns pair (result,rem)
  int n=p.c.size()-1;
  vector<tp> b(n);
  b[n-1]=p[n];
  for(int k=n-2;k>=0;--k)b[k]=p[k+1]+r*b[k+1];
  return mp(poly<>(b),p[0]+r*b[0]);
}
// only for double polynomials
pair<poly<>,poly<> > polydiv(poly<> p, poly<> q){ // returns pair (result,
    rem)
  int n=p.c.size()-q.c.size()+1;
  vector<tp> b(n);
  for(int k=n-1;k>=0;--k){
    b[k]=p.c.back()/q.c.back();
    forn(i,q.c.size())p[i+k]-=b[k]*q[i];
    p.c.pop_back();
  }
  while(!p.c.empty()&&abs(p.c.back())<EPS)p.c.pop_back();
  return mp(poly<>(b),p);
}
// only for double polynomials
poly<> interpolate(vector<tp> x, vector<tp> y){ //TODO TEST
  poly<> q={1},S={0};
  for(tp a:x)q=poly<>({-a,1})*q;
  forn(i,x.size()){
    poly<> Li=ruffini(q,x[i]).fst;
    Li=Li*(1.0/Li(x[i])); // change for int polynomials
    S=S+Li*y[i];
  }
  return S;
}
```

3.8 Bairstow

```
double pget(poly<>& p, int k){return k<p.c.size()?p[k]:0;}
poly<> bairstow(poly<> p){ // returns polynomial of degree 2 that
  int n=p.c.size()-1;    // divides p
  assert(n>=3&&abs(p.c.back())>EPS);
  double u=p[n-1]/p[n],v=p[n-2]/p[n];
  forn(_,ITER){
    auto w=polydiv(p,{v,u,1});
    poly<> q=w.fst,r0=w.snd;
    poly<> r1=polydiv(q,{v,u,1}).snd;
    double c=pget(r0,1),d=pget(r0,0),g=pget(r1,1),h=pget(r1,0);
    double det=1/(v*g*g+h*(h-u*g)),uu=u;
    u-=det*(-h*c+g*d);v-=det*(-g*v*c+(g*uu-h)*d);

  }
  return {v,u,1};
}
void addr(vector<double>& r, poly<>& p){
  assert(p.c.size()<=3);
  if(p.c.size()<=1)return;
  if(p.c.size()==2)r.pb(-p[0]/p[1]);
  if(p.c.size()==3){
    double a=p[2],b=p[1],c=p[0];
    double d=b*b-4*a*c;
    if(d<-0.1)return; // huge epsilon because of bad precision
    d=d>0?sqrt(d):0;r.pb((-b-d)/2/a);r.pb((-b+d)/2/a);
  }
}
vector<double> roots(poly<> p){
  while(!p.c.empty()&&abs(p.c.back())<EPS)p.c.pop_back();
  forn(i,p.c.size())p[i]/=p.c.back();
  vector<double> r;int n;
  while((n=p.c.size()-1)>=3){
    poly<> q=bairstow(p);addr(r,q);
    p=polydiv(p,q).fst;
    while(p.c.size()>n-1)p.c.pop_back();
  }
  addr(r,p);
  return r;
}
```

3.9 Fast Fourier Transform

```
struct CD {  // or typedef complex<double> CD; (but 4x slower)
  double r,i;
  CD(double r=0, double i=0):r(r),i(i){}
  void operator/=(const int c){r/=c, i/=c;}
};
CD operator*(const CD& a, const CD& b){
  return CD(a.r*b.r-a.i*b.i,a.r*b.i+a.i*b.r);}
CD operator+(const CD& a, const CD& b){return CD(a.r+b.r,a.i+b.i);}
CD operator-(const CD& a, const CD& b){return CD(a.r-b.r,a.i-b.i);}
const double pi=acos(-1.0);
CD cp1[MAXN+9],cp2[MAXN+9],w[MAXN+9];  // MAXN must be power of 2 !!
int R[MAXN+9];
void dft(CD* a, int n, bool inv){
  forn(i,n)if(R[i]<i)swap(a[R[i]],a[i]);
  for(int m=2;m<=n;m*=2){
    double z=2*pi/m*(inv?-1:1);
    CD wi=CD(cos(z),sin(z));
    for(int j=0;j<n;j+=m){
      w[0]=1;
      for(int k=j,k2=j+m/2,t=1;k2<j+m;k++,k2++,t++){
        CD u=a[k];CD v=a[k2]*w[t-1];a[k]=u+v;a[k2]=u-v;
        w[t]=t%2?wi*w[t-1]:w[t/2]*w[t/2];
      }
    }
  }
  if(inv)forn(i,n)a[i]/=n;
}
vector<int> multiply(vector<int>& p1, vector<int>& p2){
  int n=p1.size()+p2.size()+1;
  int m=1,cnt=0;
  while(m<=n)m+=m,cnt++;
  forn(i,m){R[i]=0;forn(j,cnt)R[i]=(R[i]<<1)|((i>>j)&1);}
  forn(i,m)cp1[i]=0,cp2[i]=0;
  forn(i,p1.size())cp1[i]=p1[i];
  forn(i,p2.size())cp2[i]=p2[i];
  dft(cp1,m,false);dft(cp2,m,false);
  forn(i,m)cp1[i]=cp1[i]*cp2[i];
  dft(cp1,m,true);
  vector<int> res;
  n-=2;
  forn(i,n)res.pb((ll)floor(cp1[i].r+0.5));
  return res;
}
```

3.10 Fast Hadamard Transform

```
ll c1[MAXN+9],c2[MAXN+9];  // MAXN must be power of 2 !!
void fht(ll* p, int n, bool inv){
  for(int l=1;2*l<=n;l*=2){
    for(int i=0;i<n;i+=2*l){
      forn(j,l){
        ll u=p[i+j],v=p[i+l+j];
        // XOR
        if(!inv)p[i+j]=u+v,p[i+l+j]=u-v;
        else p[i+j]=(u+v)/2,p[i+l+j]=(u-v)/2;
        // AND
        //if(!inv)p[i+j]=v,p[i+l+j]=u+v;
        //else p[i+j]=-u+v,p[i+l+j]=u;
        // OR
        //if(!inv)p[i+j]=u+v,p[i+l+j]=u;
        //else p[i+j]=v,p[i+l+j]=u-v;
      }
    }
  }
}
// like polynomial multiplication, but XORing exponents
// instead of adding them (also ANDing, ORing)
vector<ll> multiply(vector<ll>& p1, vector<ll>& p2){
  int n=1<<(32-__builtin_clz(max(sz(p1),sz(p2))-1));
  forn(i,n)c1[i]=0,c2[i]=0;
  forn(i,sz(p1))c1[i]=p1[i];
  forn(i,sz(p2))c2[i]=p2[i];
  fht(c1,n,false);fht(c2,n,false);
  forn(i,n)c1[i]*=c2[i];
  fht(c1,n,true);
  return vector<ll>(c1,c1+n);
}
```

3.11 Karatsuba

```
typedef ll tp;
```

```
#define add(n,s,d,k) forn(i,n)(d)[i]+=(s)[i]*k
tp* ini(int n){tp *r=new tp[n];fill(r,r+n,0);return r;}
void karatsura(int n, tp* p, tp* q, tp* r){
  if(n<=0)return;
  if(n<35)forn(i,n)forn(j,n)r[i+j]+=p[i]*q[j];
  else {
    int nac=n/2,nbd=n-n/2;
    tp *a=p,*b=p+nac,*c=q,*d=q+nac;
    tp *ab=ini(nbd+1),*cd=ini(nbd+1),*ac=ini(nac*2),*bd=ini(nbd*2);
    add(nac,a,ab,1);add(nbd,b,ab,1);
    add(nac,c,cd,1);add(nbd,d,cd,1);
    karatsura(nac,a,c,ac);karatsura(nbd,b,d,bd);
    add(nac*2,ac,r+nac,-1);
    add(nbd*2,bd,r+nac,-1);
    add(nac*2,ac,r,1);
    add(nbd*2,bd,r+nac*2,1);
    karatsura(nbd+1,ab,cd,r+nac);
    free(ab);free(cd);free(ac);free(bd);
  }
}
vector<tp> multiply(vector<tp> p0, vector<tp> p1){
  int n=max(p0.size(),p1.size());
  tp *p=ini(n),*q=ini(n),*r=ini(2*n);
  forn(i,p0.size())p[i]=p0[i];
  forn(i,p1.size())q[i]=p1[i];
  karatsura(n,p,q,r);
  vector<tp> rr(r,r+p0.size()+p1.size()-1);
  free(p);free(q);free(r);
  return rr;
}
```

3.12 Modular inverse

```
inv[1]=1; //O(MAXN), i*inv[i] = 1 mod p, MAXN <= p
forr(i, 2, MAXN) inv[i]=p-((ll)(p/i)*inv[p%i])%p;
```

3.13 Chinese remainder theorem (Euge)

```
#define mod(a, m) (((a)%m + m)%m)
struct Meq { // requires euclid, inv, mulmod (from pollard rho)
    ll a, b, m; // a*x = b (mod m)
    Meq(ll a = 0, ll b = 0, ll m = 0): a(a), b(b), m(m){}
    bool norm(){ // returns false if equation is not consistent
        a = mod(a, m); b = mod(b, m);
        ll g = __gcd(a, m); if(b%g) return false;
        a/=g; b/=g; m/=g; b = b*inv(a, m)%m; a = 1;
        return true;
    }
};
Meq Euge(Meq S, Meq T){ // Requires S, T to be normalized first
    ll x, y, g = euclid(S.m, -T.m, x, y);
    if(g < 0) x *= -1, y *= -1, g *= -1;
    if((S.b - T.b)%g) return Meq(); // returns m = 0 if not consistent
    ll M = S.m * (T.m/g), r = (T.b - S.b)/g;
    x = mulmod(x, r, M);
    ll A = mod(mulmod(S.m, x, M) + S.b, M);
    return Meq(1, A, M);
}
```

3.14 Mobius

```
short mu[MAXN] = {0,1};
void mobius(){
  forr(i,1,MAXN)if(mu[i])for(int j=i+i;j<MAXN;j+=i)mu[j]-=mu[i];
}
```

3.15 Linear Recurrence

```
struct LRec{
    int n; vector<int> In, T; vector<vector<int>> B;
    vector<int> add(vector<int> &a, vector<int> &b){
        vector<int> ans(2*n+1, 0);
        forn(i, n+1)forn(j, n+1)
            ans[i+j] = (ans[i+j] + (ll)a[i]*b[j]%MOD + MOD)%MOD;
        for(int i = 2*n; i > n; i--)forn(j, n)
            ans[i-1-j] = (ans[i-1-j] + (ll)ans[i]*T[j]%MOD + MOD)%MOD;
        ans.resize(n+1); return ans; }
    LRec(vector<int> V, vector<int> T): In(V), T(T){
        n = sz(V);
        vector<int> a(n+1, 0);
        a[1] = 1; B.pb(a);
          forr(i, 1, LOG) B.pb(add(B[i-1], B[i-1])); }
    int calc(ll k){
        vector<int> a(n+1, 0); a[0] = 1;
        forn(i, LOG)if(k>>i&1)a = add(a, B[i]);
        int ret = 0;
        forn(i, n)ret = (ret + (ll)a[i+1]*In[i]%MOD + MOD)%MOD;
        return ret; }
};
```

3.16 Gaussian Elimination

```
double reduce(vector<vector<double> >& x){ // returns determinant
  int n=x.size(),m=x[0].size();
  int i=0,j=0;double r=1.;
  while(i<n&&j<m){
    int l=i;
    forr(k,i+1,n)if(abs(x[k][j])>abs(x[l][j]))l=k;
    if(abs(x[l][j])<EPS){j++;r=0.;continue;}
    if(l!=i){r=-r;swap(x[i],x[l]);}
    r*=x[i][j];
    for(int k=m-1;k>=j;k--)x[i][k]/=x[i][j];
    forn(k,n){
      if(k==i)continue;
      for(int l=m-1;l>=j;l--)x[k][l]-=x[k][j]*x[i][l];
    }
    i++;j++;
  }
  return r;
}
```

3.17 Simplex

```
vector<int> X,Y;
vector<vector<double> > A;
vector<double> b,c;
double z;
int n,m;
void pivot(int x,int y){
  swap(X[y],Y[x]);
  b[x]/=A[x][y];
  forn(i,m)if(i!=y)A[x][i]/=A[x][y];
  A[x][y]=1/A[x][y];
  forn(i,n)if(i!=x&&abs(A[i][y])>EPS){
    b[i]-=A[i][y]*b[x];
    forn(j,m)if(j!=y)A[i][j]-=A[i][y]*A[x][j];
    A[i][y]=-A[i][y]*A[x][y];
  }
  z+=c[y]*b[x];
  forn(i,m)if(i!=y)c[i]-=c[y]*A[x][i];
  c[y]=-c[y]*A[x][y];
}
pair<double,vector<double> > simplex( // maximize c^T x s.t. Ax<=b, x>=0
    vector<vector<double> > _A, vector<double> _b, vector<double> _c){
  // returns pair (maximum value, solution vector)
  A=_A;b=_b;c=_c;
  n=b.size();m=c.size();z=0.;
  X=vector<int>(m);Y=vector<int>(n);
  forn(i,m)X[i]=i;
  forn(i,n)Y[i]=i+m;
  while(1){
    int x=-1,y=-1;
    double mn=-EPS;
    forn(i,n)if(b[i]<mn)mn=b[i],x=i;
    if(x<0)break;
    forn(i,m)if(A[x][i]<-EPS){y=i;break;}
    assert(y>=0); // no solution to Ax<=b
    pivot(x,y);
  }
  while(1){
    double mx=EPS;
    int x=-1,y=-1;
    forn(i,m)if(c[i]>mx)mx=c[i],y=i;
    if(y<0)break;
    double mn=1e200;
    forn(i,n)if(A[i][y]>EPS&&b[i]/A[i][y]<mn)mn=b[i]/A[i][y],x=i;
    assert(x>=0); // c^T x is unbounded
    pivot(x,y);
  }
  vector<double> r(m);
  forn(i,n)if(Y[i]<m)r[Y[i]]=b[i];
  return mp(z,r);
}
```

4 Geometry

4.1 Point

```
bool left(pt p, pt q){ // is it to the left of directed line pq?
  return (q-p)%(*this-p)>EPS;}
pt rot(pt r){return pt(*this%r,*this*r);}
pt rot(double a){return rot(pt(sin(a),cos(a)));}
pt ccw90(1,0); pt cw90(-1,0);
```

4.2 Line

```
int sgn2(double x){return x<0?-1:1;}
struct ln {
  pt p,pq;
  ln(pt p, pt q):p(p),pq(q-p){}
  ln(){}
  bool has(pt r){return dist(r)<EPS;}
  bool seghas(pt r){return has(r)&&(r-p)*(r-(p+pq))-EPS<0;}
//  bool operator /(ln l){return (pq.unit()^l.pq.unit()).norm()<EPS;} // 3D
  bool operator/(ln l){return abs(pq.unit()%l.pq.unit())<EPS;} // 2D
  bool operator==(ln l){return *this/l&&has(l.p);}
  pt operator^(ln l){ // intersection
    if(*this/l)return pt(DINF,DINF);
    pt r=l.p+l.pq*((p-l.p)%pq/(l.pq%pq));
//    if(!has(r)){return pt(NAN,NAN,NAN);} // check only for 3D
    return r;
  }
  double angle(ln l){return pq.angle(l.pq);}
  int side(pt r){return has(r)?0:sgn2(pq%(r-p));} // 2D
  pt proj(pt r){return p+pq*((r-p)*pq/pq.norm2());}
  pt ref(pt r){return proj(r)*2-r;}
  double dist(pt r){return (r-proj(r)).norm();}
//  double dist(ln l){ // only 3D
//    if(*this/l)return dist(l.p);
//    return abs((l.p-p)*(pq^l.pq))/(pq^l.pq).norm();
//  }
  ln rot(auto a){return ln(p,p+pq.rot(a));} // 2D
};
ln bisector(ln l, ln m){ // angle bisector
  pt p=l^m;
  return ln(p,p+l.pq.unit()+m.pq.unit());
}
ln bisector(pt p, pt q){ // segment bisector (2D)
  return ln((p+q)*.5,p).rot(ccw90);
}
```

4.3 Circle

```
struct circle {
  pt o;double r;
  circle(pt o, double r):o(o),r(r){}
  circle(pt x, pt y, pt z){o=bisector(x,y)^bisector(x,z);r=(o-x).norm();}
  vector<pt> operator^(circle c){ // ccw
    vector<pt> s;
    double d=(o-c.o).norm();
    if(d>r+c.r+EPS||d+min(r,c.r)+EPS<max(r,c.r))return s;
    double x=(d*d-c.r*c.r+r*r)/(2*d);
    double y=sqrt(r*r-x*x);
    pt v=(c.o-o)/d;
    s.pb(o+v*x-v.rot(ccw90)*y);
    if(y>EPS)s.pb(o+v*x+v.rot(ccw90)*y);
    return s;
  }
  vector<pt> operator^(ln l){
    vector<pt> s;
    pt p=l.proj(o);
    double d=(p-o).norm();
    if(d-EPS>r)return s;
    if(abs(d-r)<EPS){s.pb(p);return s;}
    d=sqrt(r*r-d*d);
    s.pb(p+l.pq.unit()*d);
    s.pb(p-l.pq.unit()*d);
    return s;
  }
  vector<pt> tang(pt p){
    double d=sqrt((p-o).norm2()-r*r);
    return *this^circle(p,d);
  }
  double intertriangle(pt a, pt b){ // area of intersection with oab
    if(abs((o-a)%(o-b))<EPS)return 0.;
    vector<pt> q={a},w=*this^ln(a,b);
    if(w.size()==2)for(auto p:w)if((a-p)*(b-p)<-EPS)q.pb(p);
    q.pb(b);
    if(q.size()==4&&(q[0]-q[1])*(q[2]-q[1])>EPS)swap(q[1],q[2]);
    double s=0;
    forn(i,q.size()-1){
      if(!has(q[i])||!has(q[i+1]))s+=r*r*(q[i]-o).angle(q[i+1]-o)/2;
      else s+=abs((q[i]-o)%(q[i+1]-o)/2);
    }
    return s;
  }
};
vector<double> intercircles(vector<circle> c){
  vector<double> r(sz(c)+1); // r[k]: area covered by at least k circles
  forn(i,sz(c)){           // O(n^2 log n) (high constant)
    int k=1;Cmp s(c[i].o);
    vector<pair<pt,int> > p={
      mp(c[i].o+pt(1,0)*c[i].r,0),
      mp(c[i].o-pt(1,0)*c[i].r,0)};
    forn(j,sz(c))if(j!=i){
      bool b0=c[i].in(c[j]),b1=c[j].in(c[i]);
      if(b0&&(!b1||i<j))k++;
      else if(!b0&&!b1){
        auto v=c[i]^c[j];
        if(sz(v)==2){
          p.pb(mp(v[0],1));p.pb(mp(v[1],-1));
          if(s(v[1],v[0]))k++;
        }
      }
    }
    sort(p.begin(),p.end(),
      [&](pair<pt,int> a, pair<pt,int> b){return s(a.fst,b.fst);});
    forn(j,sz(p)){
      pt p0=p[j?j-1:sz(p)-1].fst,p1=p[j].fst;
      double a=(p0-c[i].o).angle(p1-c[i].o);
      r[k]+=(p0.x-p1.x)*(p0.y+p1.y)/2+c[i].r*c[i].r*(a-sin(a))/2;
      k+=p[j].snd;
    }
  }
  return r;
}
```

4.4 Polygon

```
int sgn(double x){return x<-EPS?-1:x>EPS;}
struct pol {
  int n;vector<pt> p;
  pol(){}
  pol(vector<pt> _p){p=_p;n=p.size();}
  bool has(pt q){ // O(n)
    forn(i,n)if(ln(p[i],p[(i+1)%n]).seghas(q))return true;
    int cnt=0;
    forn(i,n){
      int j=(i+1)%n;
      int k=sgn((q-p[j])%(p[i]-p[j]));
      int u=sgn(p[i].y-q.y),v=sgn(p[j].y-q.y);
      if(k>0&&u<0&&v>=0)cnt++;
      if(k<0&&v<0&&u>=0)cnt--;
    }
    return cnt!=0;
  }
  void normalize(){ // (call before haslog, remove collinear first)
    if(p[2].left(p[0],p[1]))reverse(p.begin(),p.end());
    int pi=min_element(p.begin(),p.end())-p.begin();
    vector<pt> s(n);
    forn(i,n)s[i]=p[(pi+i)%n];
    p.swap(s);
  }
  bool haslog(pt q){ // O(log(n)) only CONVEX. Call normalize first
    if(q.left(p[0],p[1])||q.left(p.back(),p[0]))return false;
    int a=1,b=p.size()-1;  // returns true if point on boundary
    while(b-a>1){          // (change sign of EPS in left
      int c=(a+b)/2;       //  to return false in such case)
      if(!q.left(p[0],p[c]))a=c;
      else b=c;
    }
    return !q.left(p[a],p[a+1]);
  }
  pt farthest(pt v){ // O(log(n)) only CONVEX
    if(n<10){
      int k=0;
      forr(i,1,n)if(v*(p[i]-p[k])>EPS)k=i;
      return p[k];
    }
    if(n==sz(p))p.pb(p[0]);
    pt a=p[1]-p[0];
    int s=0,e=n,ua=v*a>EPS;
    if(!ua&&v*(p[n-1]-p[0])<=EPS)return p[0];
    while(1){
      int m=(s+e)/2;pt c=p[m+1]-p[m];
      int uc=v*c>EPS;
      if(!uc&&v*(p[m-1]-p[m])<=EPS)return p[m];
      if(ua&&(!uc||v*(p[s]-p[m])>EPS))e=m;
      else if(ua||uc||v*(p[s]-p[m])>=-EPS)s=m,a=c,ua=uc;
      else e=m;
      assert(e>s+1);
    }
  }
  pol cut(ln l){   // cut CONVEX polygon by line l
    vector<pt> q;  // returns part at left of l.pq
    forn(i,n){
      int d0=sgn(l.pq%(p[i]-l.p)),d1=sgn(l.pq%(p[(i+1)%n]-l.p));
      if(d0>=0)q.pb(p[i]);
      ln m(p[i],p[(i+1)%n]);
      if(d0*d1<0&&!(l/m))q.pb(l^m);
    }
    return pol(q);
  }
  double intercircle(circle c){ // area of intersection with circle
    double r=0.;
    forn(i,n){
      int j=(i+1)%n;double w=c.intertriangle(p[i],p[j]);
      if((p[j]-c.o)%(p[i]-c.o)>0)r+=w;
      else r-=w;
    }
    return abs(r);
  }
  double callipers(){ // square distance of most distant points
    double r=0;     // prereq: convex, ccw, NO COLLINEAR POINTS
    for(int i=0,j=n<2?0:1;i<j;++i){
      for(;;j=(j+1)%n){
        r=max(r,(p[i]-p[j]).norm2());
        if((p[(i+1)%n]-p[i])%(p[(j+1)%n]-p[j])<=EPS)break;
      }
    }
    return r;
  }
};
// Dynamic convex hull trick
vector<pol> w;
void add(pt q){ // add(q), O(log^2(n))
  vector<pt> p={q};
  while(!w.empty()&&sz(w.back().p)<2*sz(p)){
    for(pt v:w.back().p)p.pb(v);
    w.pop_back();
  }
  w.pb(pol(chull(p)));
}
ll query(pt v){ // max(q*v:q in w), O(log^2(n))
  ll r=-INF;
  for(auto& p:w)r=max(r,p.farthest(v)*v);
  return r;
}
```

4.5 Plane

```
struct plane {
  pt a,n; // n: normal unit vector
  plane(pt a, pt b, pt c):a(a),n(((b-a)^(c-a)).unit()){}
  plane(){}
  bool has(pt p){return abs((p-a)*n)<EPS;}
  double angle(plane w){return acos(n*w.n);}
  double dist(pt p){return abs((p-a)*n);}
  pt proj(pt p){inter(ln(p,p+n),p);return p;}
  bool inter(ln l, pt& r){
    double x=n*(l.p+l.pq-a),y=n*(l.p-a);
    if(abs(x-y)<EPS)return false;
    r=(l.p*x-(l.p+l.pq)*y)/(x-y);
    return true;
  }
  bool inter(plane w, ln& r){
    pt nn=n^w.n;
    pt v=n^nn;
    double d=w.n*v;
    if(abs(d)<EPS)return false;
    pt p=a+v*(w.n*(w.a-a)/d);
    r=ln(p,p+nn);
    return true;
  }
};
```

4.6 Convex hull

```
// CCW order
// Includes collinear points (change sign of EPS in left to exclude)
vector<pt> chull(vector<pt> p){
  vector<pt> r;
  sort(p.begin(),p.end()); // first x, then y
  forn(i,p.size()){ // lower hull
    while(r.size()>=2&&r.back().left(r[r.size()-2],p[i]))r.pop_back();
    r.pb(p[i]);
  }
  r.pop_back();
  int k=r.size();
  for(int i=p.size()-1;i>=0;--i){ // upper hull
    while(r.size()>=k+2&&r.back().left(r[r.size()-2],p[i]))r.pop_back();
    r.pb(p[i]);
  }
  r.pop_back();
  return r;
}
```

5 Strings

5.1 KMP

```
vector<int> kmppre(string& t){ // r[i]: longest border of t[0,i)
```

```
  vector<int> r(t.size()+1);r[0]=-1;
  int j=-1;
  forn(i,t.size()){
    while(j>=0&&t[i]!=t[j])j=r[j];
    r[i+1]=++j;
  }
  return r;
}
void kmp(string& s, string& t){ // find t in s
  int j=0;vector<int> b=kmppre(t);
  forn(i,s.size()){
    while(j>=0&&s[i]!=t[j])j=b[j];
    if(++j==sz(t))printf("Match␣at␣%d\n",i-j+1),j=b[j];
  }
}
```

5.2 Z function

```
vector<int> z_function(string& s){
  int a=0,b=0,n=sz(s);
  vector<int> z(n,0); // z[i] = max k: s[0,k) == s[i,i+k)
  forr(i,1,n){
    if(i<=b)z[i]=min(b-i+1,z[i-a]);
    while(i+z[i]<n&&s[z[i]]==s[i+z[i]])z[i]++;
    if(i+z[i]-1>b)a=i,b=i+z[i]-1;
  }
  return z;
}
```

5.3 Manacher

```
int d1[MAXN];//d1[i] = max odd palindrome centered on i
int d2[MAXN];//d2[i] = max even palindrome centered on i
//s  aabbaacaabbaa
//d1 1111117111111
//d2 0103010010301
void manacher(string& s){
  int l=0,r=-1,n=s.size();
  forn(i,n){
    int k=i>r?1:min(d1[l+r-i],r-i);
    while(i+k<n&&i-k>=0&&s[i+k]==s[i-k])k++;
    d1[i]=k--;
    if(i+k>r)l=i-k,r=i+k;
  }
  l=0;r=-1;
  forn(i,n){
    int k=i>r?0:min(d2[l+r-i+1],r-i+1);k++;
    while(i+k<=n&&i-k>=0&&s[i+k-1]==s[i-k])k++;
    d2[i]=--k;
    if(i+k-1>r)l=i-k,r=i+k-1;
  }
}
```

5.4 Aho-Corasick

```
struct vertex {
  map<char,int> next,go;
  int p,link;
  char pch;
  vector<int> leaf;
  vertex(int p=-1, char pch=-1):p(p),pch(pch),link(-1){}
};
vector<vertex> t;
void aho_init(){ //do not forget!!
  t.clear();t.pb(vertex());
}
void add_string(string s, int id){
  int v=0;
  for(char c:s){
    if(!t[v].next.count(c)){
      t[v].next[c]=t.size();
      t.pb(vertex(v,c));
    }
    v=t[v].next[c];
  }
  t[v].leaf.pb(id);
}
int go(int v, char c);
int get_link(int v){
  if(t[v].link<0)
    if(!v||!t[v].p)t[v].link=0;
    else t[v].link=go(get_link(t[v].p),t[v].pch);
  return t[v].link;
}
int go(int v, char c){
  if(!t[v].go.count(c))
    if(t[v].next.count(c))t[v].go[c]=t[v].next[c];
    else t[v].go[c]=v==0?0:go(get_link(v),c);
  return t[v].go[c];
}
```

5.5 Suffix automaton

```
struct state {int len,link;map<char,int> next;}; //clear next!!
state st[100005]; // should be >= 2*sz(s)
int sz,last;
void sa_init(){
  last=st[0].len=0;sz=1;
  st[0].link=-1;
}
void sa_extend(char c){
  int k=sz++,p;
  st[k].len=st[last].len+1;
  for(p=last;p!=-1&&!st[p].next.count(c);p=st[p].link)st[p].next[c]=k;
  if(p==-1)st[k].link=0;
  else {
    int q=st[p].next[c];
    if(st[p].len+1==st[q].len)st[k].link=q;
    else {
      int w=sz++;
      st[w].len=st[p].len+1;
      st[w].next=st[q].next;st[w].link=st[q].link;
      for(;p!=-1&&st[p].next[c]==q;p=st[p].link)st[p].next[c]=w;
      st[q].link=st[k].link=w;
    }
  }
  last=k;
}
```

5.6 Suffix array

```
#define RB(x) (x<n?r[x]:0)
void csort(vector<int>& sa, vector<int>& r, int k){
  int n=sa.size();
  vector<int> f(max(255,n),0),t(n);
  forn(i,n)f[RB(i+k)]++;
  int sum=0;
  forn(i,max(255,n))f[i]=(sum+=f[i])-f[i];
  forn(i,n)t[f[RB(sa[i]+k)]++]=sa[i];
  sa=t;
}
vector<int> constructSA(string& s){ // O(n logn)
  int n=s.size(),rank;
  vector<int> sa(n),r(n),t(n);
  forn(i,n)sa[i]=i,r[i]=s[i];
  for(int k=1;k<n;k*=2){
    csort(sa,r,k);csort(sa,r,0);
    t[sa[0]]=rank=0;
    forr(i,1,n){
      if(r[sa[i]]!=r[sa[i-1]]||RB(sa[i]+k)!=RB(sa[i-1]+k))rank++;
      t[sa[i]]=rank;
    }
    r=t;
    if(r[sa[n-1]]==n-1)break;
  }
  return sa;
}
```

5.7 LCP (Longest Common Prefix)

```
vector<int> computeLCP(string& s, vector<int>& sa){
  int n=s.size(),L=0;
  vector<int> lcp(n),plcp(n),phi(n);
  phi[sa[0]]=-1;
  forr(i,1,n)phi[sa[i]]=sa[i-1];
  forn(i,n){
    if(phi[i]<0){plcp[i]=0;continue;}
    while(s[i+L]==s[phi[i]+L])L++;
    plcp[i]=L;
    L=max(L-1,0);
  }
  forn(i,n)lcp[i]=plcp[sa[i]];
  return lcp; // lcp[i]=LCP(sa[i-1],sa[i])
}
```

5.8 Suffix Tree (Ukkonen's algorithm)

```
struct SuffixTree {
  char s[MAXN];
  map<int,int> to[MAXN];
  int len[MAXN]={INF},fpos[MAXN],link[MAXN];
  int node,pos,sz=1,n=0;
  int make_node(int p, int l){
    fpos[sz]=p;len[sz]=l;return sz++;}
  void go_edge(){
    while(pos>len[to[node][s[n-pos]]]){
      node=to[node][s[n-pos]];
      pos-=len[node];
    }
  }
  void add(int c){
    s[n++]=c;pos++;
    int last=0;
    while(pos>0){
      go_edge();
      int edge=s[n-pos];
      int& v=to[node][edge];
      int t=s[fpos[v]+pos-1];
      if(v==0){
        v=make_node(n-pos,INF);
        link[last]=node;last=0;
      }
      else if(t==c){link[last]=node;return;}
      else {
        int u=make_node(fpos[v],pos-1);
        to[u][c]=make_node(n-1,INF);
        to[u][t]=v;
        fpos[v]+=pos-1;len[v]-=pos-1;
        v=u;link[last]=u;last=u;
      }
      if(node==0)pos--;
      else node=link[node];
    }
  }
};
```

5.9 Hashing

```
struct Hash {
  int P=1777771,MOD[2],PI[2];
  vector<int> h[2],pi[2];
  Hash(const string& s){
    MOD[0]=999727999;MOD[1]=1070777777;
    PI[0]=325255434;PI[1]=10018302;
    forn(k,2)h[k].resize(sz(s)+1),pi[k].resize(sz(s)+1);
    forn(k,2){
      h[k][0]=0;pi[k][0]=1;
      ll p=1;
      forr(i,1,sz(s)+1){
        h[k][i]=(h[k][i-1]+p*s[i-1])%MOD[k];
        pi[k][i]=(1LL*pi[k][i-1]*PI[k])%MOD[k];
        p=(p*P)%MOD[k];
      }
    }
  }
  ll get(int s, int e){
    ll r[2]; forn(k, 2){
      r[k]=(h[k][e]-h[k][s]+MOD[k])%MOD[k];
      r[k]=(1LL*r[k]*pi[k][s])%MOD[k];
    }
    return (r[0]<<32)|r[1];
  }
};
```

6 Flow

6.1 Matching (slower)

```
vector<int> g[MAXN]; // [0,n)->[0,m)
int n,m;
int mat[MAXM];bool vis[MAXN];
int match(int x){
  if(vis[x])return 0;
  vis[x]=true;
  for(int y:g[x])if(mat[y]<0||match(mat[y])){mat[y]=x;return 1;}
  return 0;
}
vector<pair<int,int> > max_matching(){
  vector<pair<int,int> > r;
  memset(mat,-1,sizeof(mat));
  forn(i,n)memset(vis,false,sizeof(vis)),match(i);
  forn(i,m)if(mat[i]>=0)r.pb(mp(mat[i],i));
  return r;
}
```

6.2 Matching (Hopcroft-Karp)

```
vector<int> g[MAXN]; // [0,n)->[0,m)
int n,m;
int mt[MAXN],mt2[MAXN],ds[MAXN];
bool bfs(){
  queue<int> q;
  memset(ds,-1,sizeof(ds));
  forn(i,n)if(mt2[i]<0)ds[i]=0,q.push(i);
  bool r=false;
  while(!q.empty()){
    int x=q.front();q.pop();
    for(int y:g[x]){
      if(mt[y]>=0&&ds[mt[y]]<0)ds[mt[y]]=ds[x]+1,q.push(mt[y]);
      else if(mt[y]<0)r=true;
    }
  }
  return r;
}
bool dfs(int x){
  for(int y:g[x])if(mt[y]<0||ds[mt[y]]==ds[x]+1&&dfs(mt[y])){
    mt[y]=x;mt2[x]=y;
    return true;
  }
  ds[x]=1<<30;
  return false;
}
int mm(){
  int r=0;
  memset(mt,-1,sizeof(mt));memset(mt2,-1,sizeof(mt2));
  while(bfs()){
    forn(i,n)if(mt2[i]<0)r+=dfs(i);
  }
  return r;
}
```

6.3 Hungarian

```
typedef double th;
const th INF=1e18;  // to maximize: set INF to 1, use negative values
struct Hungarian {
  int n,m;  // important: n must be <=m
  vector<vector<th> > a;
  vector<th> u,v;vector<int> p,way;  // p: assignment
  Hungarian(int n, int m):
  n(n),m(m),a(n+1,vector<th>(m+1,INF-1)),u(n+1),v(m+1),p(m+1),way(m+1){}
  void set(int x, int y, th v){a[x+1][y+1]=v;}
  th assign(){
    forr(i,1,n+1){
      int j0=0;p[0]=i;
      vector<th> minv(m+1,INF);
      vector<char> used(m+1,false);
      do {
        used[j0]=true;
        int i0=p[j0],j1;th delta=INF;
        forr(j,1,m+1)if(!used[j]){
          th cur=a[i0][j]-u[i0]-v[j];
          if(cur<minv[j])minv[j]=cur,way[j]=j0;
          if(minv[j]<delta)delta=minv[j],j1=j;
        }
        forn(j,m+1)
          if(used[j])u[p[j]]+=delta,v[j]-=delta;
          else minv[j]-=delta;
        j0=j1;
      } while(p[j0]);
      do {
        int j1=way[j0];p[j0]=p[j1];j0=j1;
      } while(j0);
    }
    return -v[0];  // cost
  }
};
```

6.4 Dinic

```
// Min cut: nodes with dist>=0 vs nodes with dist<0
// MVC (bipartite): left nodes with dist<0 + right nodes with dist>0
int nodes,src,dst; // remember to init nodes
int dist[MAXN],q[MAXN],work[MAXN];
// ll M[MAXN]; (MIN CAP)
struct edge {int to,rev;ll f,cap;};
vector<edge> g[MAXN];
void add_edge(int s, int t, ll cap/*, ll lcap = 0 (MIN CAP)*/){
  // if(lcap) M[s] -= lcap, M[t] += lcap, cap -= lcap; (MIN CAP)
  g[s].pb((edge){t,sz(g[t]),0,cap});
  g[t].pb((edge){s,sz(g[s])-1,0,0});
}
bool dinic_bfs(){
  fill(dist,dist+nodes,-1);dist[src]=0;
  int qt=0;q[qt++]=src;
  forn(qh,qt){
    int u=q[qh];
    forn(i,sz(g[u])){
      edge &e=g[u][i];int v=g[u][i].to;
      if(dist[v]<0&&e.f<e.cap)dist[v]=dist[u]+1,q[qt++]=v;
    }
  }
  return dist[dst]>=0;
}
ll dinic_dfs(int u, ll f){
  if(u==dst)return f;
  for(int &i=work[u];i<sz(g[u]);i++){
    edge &e=g[u][i];
    if(e.cap<=e.f)continue;
    int v=e.to;
    if(dist[v]==dist[u]+1){
      ll df=dinic_dfs(v,min(f,e.cap-e.f));
      if(df>0){e.f+=df;g[v][e.rev].f-=df;return df;}
    }
  }
  return 0;
}
ll max_flow(int _src, int _dst){ // O(m n^2)
  src=_src;dst=_dst; // if unit weights, O(m min(sqrt(m), n^{2/3}))
  ll result=0;       // if bipartite matching, O(m sqrt(n))
  while(dinic_bfs()){
    fill(work, work+nodes, 0);
    while(ll delta=dinic_dfs(src,INF))result+=delta;
  }
  return result;
}
//Checks if a strongly connected flow network has a feasible flow distribution
bool feasible(int n){ // n = number of nodes in the network
    src = n, dst = n+1, nodes = n+2;
    forn(i, n){
        if(M[i] > 0)add_edge(src, i, M[i]);
        if(M[i] < 0)add_edge(i, dst, -M[i]);
    }
    max_flow(src, dst);
    for(edge e : g[src]) if(e.f < e.cap) return false;
    return true;
}
```

6.5 Min cost max flow

```
typedef ll tf;const tf INFFLUJO=1e14;
typedef ll tc;const tc INFCOSTO=1e14;
struct edge {
  int u,v;tf cap,flow;tc cost;
  tf rem(){return cap-flow;}
};
int nodes; // remember to init nodes
vector<int> g[MAXN];
vector<edge> e;
void add_edge(int u, int v, tf cap, tc cost) {
  g[u].pb(e.size());e.pb((edge){u,v,cap,0,cost});
  g[v].pb(e.size());e.pb((edge){v,u,0,0,-cost});
}
tc dist[MAXN],mncost;
int pre[MAXN];
tf cap[MAXN],mxflow;
bool in_queue[MAXN];
void flow(int s, int t){
  memset(in_queue,0,sizeof(in_queue));
  mxflow=mncost=0;
  while(1){
    fill(dist,dist+nodes,INFCOSTO);dist[s]=0;
    memset(pre,-1,sizeof(pre));pre[s]=0;
    memset(cap,0,sizeof(cap));cap[s]=INFFLUJO;
    queue<int> q;q.push(s);in_queue[s]=1;
    while(q.size()){
      int u=q.front();q.pop();in_queue[u]=0;
      forn(_,g[u].size()){
        int i=g[u][_];
        edge &E=e[i];
        if(E.rem()&&dist[E.v]>dist[u]+E.cost+1e-9){
          dist[E.v]=dist[u]+E.cost;
          pre[E.v]=i;
          cap[E.v]=min(cap[u],E.rem());
          if(!in_queue[E.v])q.push(E.v),in_queue[E.v]=1;
        }
      }
    }
    if(pre[t]<0)break;
    mxflow+=cap[t];mncost+=cap[t]*dist[t];
    for(int v=t;v!=s;v=e[pre[v]].u){
      e[pre[v]].flow+=cap[t];e[pre[v]^1].flow-=cap[t];
    }
  }
}
```

7 Other

7.1 Mo's algorithm

```
int n,sq,nq; // array size, sqrt(array size), #queries
struct qu{int l,r,id;}; // O((n+nq)*sqrt(n)*update)
qu qs[MAXN];
ll ans[MAXN]; // ans[i] = answer to ith query
bool qcomp(const qu &a, const qu &b){
    if(a.l/sq!=b.l/sq) return a.l<b.l;
    return (a.l/sq)&1?a.r<b.r:a.r>b.r;
}
void mos(){
    forn(i,nq)qs[i].id=i;
    sq=sqrt(n)+.5;
    sort(qs,qs+nq,qcomp);
    int l=0,r=0;
    init();
    forn(i,nq){
        qu q=qs[i];
        while(l>q.l)add(--l);
        while(r<q.r)add(r++);
        while(l<q.l)remove(l++);
        while(r>q.r)remove(--r);
        ans[q.id]=get_ans();
    }
}
```

7.2 Divide and conquer DP optimization

```
// O(knlogn). For 2D dps, when the position of optimal choice is non-decreasing as the second variable increases
int k,n,f[MAXN],f2[MAXN];
void doit(int s, int e, int s0, int e0, int i){
  // [s,e): range of calculation, [s0,e0): range of optimal choice
  if(s==e)return;
  int m=(s+e)/2,r=INF,rp;
  forr(j,s0,min(e0,m)){
    int r0=f[j]+something(j,m-1); // calculate cost of taking [j,m-1]
    if(r0<r)r=r0,rp=j; // position of optimal choice
  }
  f2[m]=r;
  doit(s,m,s0,rp+1,i);doit(m+1,e,rp,e0,i);
}
int doall(){
  init_base_cases();
  forr(i,1,k+1)doit(1,n+1,0,n,i),memcpy(f,f2,sizeof(f));
  return f[n];
}
```

7.3 Dates

```
int dateToInt(int y, int m, int d){ // 1-indexado (mes 2 = febrero)
  return 1461*(y+4800+(m-14)/12)/4+367*(m-2-(m-14)/12*12)/12-
    3*((y+4900+(m-14)/12)/100)/4+d-32075;
}
void intToDate(int jd, int& y, int& m, int& d){
  int x,n,i,j;x=jd+68569;
  n=4*x/146097;x-=(146097*n+3)/4;
  i=(4000*(x+1))/1461001;x-=1461*i/4-31;
  j=80*x/2447;d=x-2447*j/80;
  x=j/11;m=j+2-12*x;y=100*(n-49)+i+x;
}
```

7.4 C++ stuff

```
const double DINF=numeric_limits<double>::infinity(); // double inf
// Custom comparator for set/map
struct comp {
  bool operator()(const double& a, const double& b) const {
    return a+EPS<b;}
};
set<double,comp> w; // or map<double,int,comp>
// Iterate over non empty subsets of bitmask
for(int s=m;s;s=(s-1)&m) // Decreasing order
for (int s=0;s=s-m&m;)   // Increasing order
// Returns the number of trailing 0-bits in x. x=0 is undefined.
int __builtin_ctz (unsigned int x)
// Returns the number of leading 0-bits in x. x=0 is undefined.
int __builtin_clz (unsigned int x)
// Use corresponding versions for long long appending ll at the end.
v=(x&(-x)) // Get the value of the least significant bit that is one.
```

7.5 Max number of divisors up to 10^n

```
(0,1) (1,4) (2,12) (3,32) (4,64) (5,128) (6,240) (7,448) (8,768) (9,1344)
    (10,2304) (11,4032) (12,6720) (13,10752) (14,17280) (15,26880)
    (16,41472) (17,64512) (18,103680)
```