西北大学 ICPC 集训队算法模板

Northwest University @nervending

2019 年 12 月 20 日

目录

1 DataStruct

1.1 基础-单调队列

```
int main() {//luogu1440
  rep(i,1,n){
    if(que.empty()) ans[i]=0;
    else ans[i]=a[que.front()];
    if(!que.empty()&&i-que.front()>=m) que.pop_front();
    while(!que.empty()&&a[i]<a[que.back()]) que.pop_back();
    que.push_back(i);
  }//min(ai-m,...,ai-1)
}
int main(){//luogu2852
  rep(i,1,n){
    while(!que.empty()&&h[i]<h[que.back()]) que.pop_back();
    que.push_back(i);
    while(!que.empty()&&i-que.front()>=m) que.pop_front();
    if(que.empty())ans[i]=0;
    else ans[i]=h[que.front()];
  }//min(ai-m+1,...,ai)
}
```

1.2 基础-单调栈

```
int main() {//hdu1506
  cin>>n;
  rep(i,1,n) cin>>a[i];
  ll ans=a[1];
  while(!stk.empty()) stk.pop();
  rep(i,1,n){
    while(!stk.empty()&&a[stk.top()]>=a[i]) stk.pop();
    if(!stk.empty()) l[i]=stk.top()+1;
    else l[i]=1;
    stk.push(i);
  }
  while(!stk.empty()) stk.pop();
  per(i,1,n){
    while(!stk.empty()&&a[stk.top()]>=a[i]) stk.pop();
    if(!stk.empty()) r[i]=stk.top()-1;
    else r[i]=n;
    stk.push(i);
  }
  rep(i,1,n) ans=max(ans,a[i]*(r[i]-l[i]+1));
  cout<<ans<<endl;
}
```

1.3 基础-拉链法哈希表

```
const int maxsz=3e5+7;//maxsz 素数表
//1e7+19,2e7+3,3e7+23,4e5+9 maxsz 最好为素数
//1e6+3,2e6+3,3e6+7,4e6+9,1e5+3,2e5+3,3e5+7
//因为是 vector 不需要限制操作次数了
//count 操作不增加新节点
template<typename key,typename val>
class hash_map{public:
  struct node{key u;val v;int next;};
  vector<node> e;
  int head[maxsz],nume,numk,id[maxsz];
  int geths(key &u){return u;}
//geths 是把 key 映射到 [0,maxsz-1] 的函数
  bool count(key &u){
    int hs=geths(u);
    for(int i=head[hs];i;i=e[i].next)
      if(e[i].u==u) return 1;
    return 0;
  }
  val& operator[](key &u){//视情况加引用, 可能引起 CE
    int hs=geths(u);
    for(int i=head[hs];i;i=e[i].next)if(e[i].u==u) return e[i].v;
    if(!head[hs])id[++numk]=hs;
    if(++nume>=e.size())e.resize(nume<<1);
    return e[nume]=(node){u,0,head[hs]},head[hs]=nume,e[nume].v;
  }
  void clear(){
```

```
    rep(i,0,numk)head[id[i]]=0;
    numk=nume=0;
  }
};
```

1.4 基础-可删堆

```
//可删堆
//保证 remove 元素被包含在 x 中，不能多删
template<typename T>class re_heap{public:
  priority_queue<T> x,y;
  void push(T a){x.push(a);}
  void remove(T a){y.push(a);}
  T top(){
    while(y.size()&&x.top()==y.top())
    x.pop(),y.pop();return x.top();
  }
  int size(){return x.size()-y.size();}
  void pop(){
    while(y.size()&&x.top()==y.top())
    x.pop(),y.pop();
    x.pop();
  }
  T sectop(){
    T a=top();pop();
    T b=top();push(a);
    return b;
  }
};
re_heap<pii> heap;
```

1.5 并查集-非递归 + 按秩合并

```
int pre[maxn],rk[maxn];
int find(int a){
  if(pre[a]==-1) return a;
  int t,rt=a;
  while(pre[rt]!=-1) rt=pre[rt];
  while(a!=rt)t=pre[a],pre[a]=rt,a=t;
  return rt;
}
#define same(a,b) (find(a)==find(b))
void unite(int a,int b){
  a=find(a),b=find(b);
  if(a==b) return;
  if(rk[a]>rk[b])pre[b]=a;
  else{
    pre[a]=b;
    if(rk[a]==rk[b])rk[b]++;
  }
}
class ufs{public:
  void init(int n){rep(i,0,n) pre[i]=i;}
  int find(int a){return pre[a]==a?a:pre[a]=find(pre[a]);}
  bool same(int a,int b) {return find(a)==find(b);}
  void unite(int a,int b){
    a=find(a),b=find(b);
    if(a==b) return;
    pre[b]=a;
  }
}dsu;
```

1.6 并查集-可删除

```
#define same(a,b) (find(a)==find(b))
int pre[maxn],id[maxn];
int numid;
int fd(int a) {
    return pre[a]==a?a:pre[a]=find(pre[a]);
}
int find(int a){return fd(id[a])};
void unite(int a,int b){
    a=find(id[a]),b=find(id[b]);
    pre[b]=a;
}
```

```
void remove(int now){
    int pos=find(id[now]);
    id[now]=++numid;
    pre[numid]=numid;
}
```

1.7 并查集-可撤销

```
class ufs{public:
  int fa[maxn],sz[maxn];
  stack<pii> stk;
  void init(int tot){
    rep(i,1,tot) fa[i]=i,sz[i]=1;
  }
  int find(int now){
    if(fa[now]==now) return now;
    return find(fa[now]);
  }
  bool unite(int a,int b){
    a=find(a),b=find(b);
    if(a==b)return 0;
    if(sz[a]<sz[b]) fa[a]=b,sz[b]+=sz[a],stk.push(make_pair(a,b));
    else fa[b]=a,sz[a]+=sz[b],stk.push(make_pair(b,a));
    return 1;
  }
  void undo(){
    if(!stk.empty()){
      auto now=stk.top();
      stk.pop();
      fa[now.fi]=now.fi;
      sz[now.se]-=sz[now.fi];
    }
  }
}dsu;
```

1.8 区间信息-数列分块

```
const int maxn=1e5+10,maxm=sqrt(maxn+1)+10;
class sqblock{public:
  int id[maxn],sz,cnt;
  struct block{int l,r;ll sum,tag;}node[maxm];
  void init(int n){
    sz=sqrt(n+0.5);
    rep(i,1,n) id[i]=(i-1)/sz+1;cnt=id[n];
    rep(i,1,cnt) node[i].l=(i-1)*sz+1;
    rep(i,1,cnt-1) node[i].r=i*sz;node[cnt].r=n;
    rep(i,1,n) node[id[i]].sum+=arr[i];
  }
  void update(int s,int t,ll x){
    int p1=id[s],p2=id[t];
    if(p1==p2)rep(i,s,t)node[p1].sum+=x,arr[i]+=x;
    else {
      rep(i,s,node[p1].r) arr[i]+=x,node[p1].sum+=x;
      rep(i,node[p2].l,t) arr[i]+=x,node[p2].sum+=x;
      rep(i,p1+1,p2-1) node[i].tag+=x;
    }
  }
  ll query(int s,int t){
    int p1=id[s],p2=id[t];ll ans=0;
    if(p1==p2) rep(i,s,t) ans+=arr[i]+node[p1].tag;
    else {
      rep(i,s,node[p1].r) ans+=arr[i]+node[p1].tag;
      rep(i,node[p2].l,t) ans+=arr[i]+node[p2].tag;
      rep(i,p1+1,p2-1) ans+=node[i].sum+node[i].tag*sz;
    }
    return ans;
  }
}square;
```

1.9 区间信息-ST 表

```
const int maxp=21;
class stable{public:
  int logn[maxn],dp[maxp][maxn],pos[maxp][maxn];
  int *a;
```

```
  void init(int n=maxn-1){
    logn[2]=1;
    rep(i,3,n) logn[i]=logn[i>>1]+1;
  }
  void cal(int *_a,int n){//init(n)
    a=_a;
    rep(i,1,n) dp[0][i]=a[i],pos[0][i]=i;
    for(int j=1;(1<<j)<=n;j++) for(int i=1;i+(1<<j)-1<=n;++i)
      dp[j][i]=min(dp[j-1][i],dp[j-1][i+(1<<(j-1))]);
    for(int j=1;(1<<j)<=n;j++) for(int i=1;i+(1<<j)-1<=n;++i)
      if(a[pos[j-1][i]]<=a[pos[j-1][i+(1<<(j-1))]]) pos[j][i]=pos[j-1][i];
      else pos[j][i]=pos[j-1][i+(1<<(j-1))];
  }
  inline int query(int l,int r){
    int lg=logn[r-l+1];
    return min(dp[lg][l],dp[lg][r-(1<<lg)+1]);
  }
  inline int query_pos(int l,int r){
    int lg=logn[r-l+1];
    if(a[pos[lg][l]]<=a[pos[lg][r-(1<<lg)+1]])return pos[lg][l];
    else return pos[lg][r-(1<<lg)+1];
  }
}st;
```

1.10 区间信息-树状数组

```
//树状数组区间求和，单点修改
template<typename T> class bit{public:
  T val[maxn];
  int lb(int x) {return x&(-x);}
  int len;
  void init(int _n){
    len=_n;
    fill_n(val+1,len+1,INF);
  }
  void update(int pos,T x){
    for(;pos>0&&pos<=n;pos+=lb(pos)) val[pos]+=x;
  }
  T psum(int pos){
    T sum=0;
    for(;pos>0;pos-=lb(pos)) sum+=val[pos];
    return sum;
  }
  T query(int l,int r){return psum(r)-psum(l-1);}
}tree;
//树状数组前缀最小值，支持单点的取 min 覆盖操作
template<typename T> class bit{public:
  T val[2*maxn];
  int lb(int x) {return x&-x;}
  int len;
  void init(int _n){
    len=_n;
    fill_n(val+1,len+1,INF);
  }
  void update(int pos,T x){
    for(;pos<=len;pos+=lb(pos)) val[pos]=min(val[pos],x);
  }
  T query(int pos){
    T res=INF;
    for(;pos;pos-=lb(pos)) res=min(val[pos],res);
    return res;
  }
};
bit<int> tree;
```

1.11 区间信息-扫描线算法

```
double dis[maxn];
struct node{
  double x,y1,y2;int tag;
  bool operator <(node &other) const{return x<other.x;}
}seg[maxn];
class segtree{public:
#define nd node[now]
```

```
#define ndl node[now<<1]
#define ndr node[now<<1|1]
  struct segnode {
    int l,r,tag;double dis;
    inline int mid(){return (r+l)>>1;}
    inline int len(){return r-l+1;}
  }node[maxn<<2|3];
  inline void update(int now){
    if(nd.tag) nd.dis=dis[nd.r+1]-dis[nd.l];
    else if(nd.len()==1) nd.dis=0;
    else nd.dis=ndl.dis+ndr.dis;
  }
  void maketree(int s,int t,int now=1){
    nd={s,t,0,0};
    if(s==t) return ;
    maketree(s,nd.mid(),now<<1);
    maketree(nd.mid()+1,t,now<<1|1);
  }
  void update(int s,int t,int x,int now=1){
    if(s<=nd.l&&t>=nd.r) {
    nd.tag+=x;update(now);
    return ;
    }
    if(s<=ndl.r) update(s,t,x,now<<1);
    if(t>ndl.r) update(s,t,x,now<<1|1);
    update(now);
  }
}tree;
int main() {IO;cout<<fixed<<setprecision(2);
  while((cin>>n)&&n){
  m=0;
  rep(i,1,n){
    double a,b,c,d;cin>>a>>b>>c>>d;
    dis[++m]=b;seg[m]={a,b,d,1};
    dis[++m]=d;seg[m]={c,b,d,-1};
  }
  sort(dis+1,dis+m+1);
  sort(seg+1,seg+m+1);
  int cnt=unique(dis+1,dis+1+m)-dis-1;
  tree.maketree(1,cnt);
  double ans=0;
  rep(i,1,m-1){
    int l=lower_bound(dis+1,dis+1+cnt,seg[i].y1)-dis;
    int r=lower_bound(dis+1,dis+1+cnt,seg[i].y2)-dis;
    r--;
    if(l<=r) tree.update(l,r,seg[i].tag);
    ans+=tree.node[1].dis*(seg[i+1].x-seg[i].x);
  }
  cout<<"Test case #"<<++casn<<endl;
  cout<<"Total explored area: "<<ans<<endl<<endl;
  }
}
```

1.12 区间信息-动态开点线段树

```
int root;
const ll rangel=0,ranger=1e9+10;
class dsegtree{public:
#define nd node[now]
#define ndl node[node[now].son[0]]
#define ndr node[node[now].son[1]]
  struct dsegnode{
    int son[2],mx,tag;
    void update(int x){mx+=x,tag+=x;}
  }node[maxn*50];
  int cnt;
  void pushup(int now){nd.mx=max(ndl.mx,ndr.mx);}
  void pushdown(int now){
    if(nd.tag){
      if(!nd.son[0]) nd.son[0]=++cnt;
      if(!nd.son[1]) nd.son[1]=++cnt;
      ndl.update(nd.tag),ndr.update(nd.tag);
      nd.tag=0;
    }
```

```
  }
  void update(ll s,ll t,int x,ll l=rangel,ll r=ranger,int &now=root){
    if(!now) {now=++cnt;nd={0,0,0,0};}
    if(s<=l&&t>=r){
      nd.update(x);
      return ;
    }
    pushdown(now);
    if(s<=((l+r)>>1)) update(s,t,x,l,(l+r)>>1,nd.son[0]);
    if(t>((l+r)>>1)) update(s,t,x,1+((l+r)>>1),r,nd.son[1]);
    pushup(now);
  }
}tree;
```

1.13 区间信息-线段树优化建图

```
int cnt;
class segtree{public:
#define nd node[now]
#define ndl node[now<<1]
#define ndr node[now<<1|1]
  int flag;//1==intree,0==outtree
  struct segnode {
    int l,r,id;
    inline int mid(){return (r+l)>>1;}
    inline int len(){return r-l+1;}
  };
  segnode node[maxn<<2|3];
  vector<int> v;
  void init(int n,int flag){
    this->flag=flag;
    maketree(1,n);
  }
  void pushup(int now){
    if(!flag){
      g.add(nd.id,ndl.id,0);
      g.add(nd.id,ndr.id,0);
    }else {
      g.add(ndl.id,nd.id,0);
      g.add(ndr.id,nd.id,0);
    }
  }
  void maketree(int s,int t,int now=1){
    nd={s,t,++cnt};
    if(s==t){
      if(flag) g.add(s,nd.id,0);
      else g.add(nd.id,s,0);
      return ;
    }
    maketree(s,nd.mid(),now<<1);
    maketree(nd.mid()+1,t,now<<1|1);
    pushup(now);
  }
  vector<int> query(int s,int t){v.clear();find(s,t);return v;}
  void find(int s,int t,int now=1){
    if(s<=nd.l&&t>=nd.r) {
      v.emplace_back(nd.id);
      return ;
    }
    if(s<=ndl.r) find(s,t,now<<1);
    if(t>ndl.r) find(s,t,now<<1|1);
  }
}intree,outree;
```

1.14 区间信息-线段树合并

```
int a[maxn],cnt,root[maxn];
vector<int> g[maxn];
ll ans[maxn];
class dsegtree{public:
#define nd node[now]
#define ndl node[node[now].son[0]]
#define ndr node[node[now].son[1]]
  struct dsegnode {
```

```
    int son[2],cnt,id,ans;
  }node[maxn*50];
  void pushup(int now){
    if(ndl.cnt>ndr.cnt){
      nd.cnt=ndl.cnt;
      nd.id=ndl.id;
      nd.ans=ndl.ans;
    }else if(ndr.cnt>ndl.cnt){
      nd.cnt=ndr.cnt;
      nd.id=ndr.id;
      nd.ans=ndr.ans;
    }else {
      nd.cnt=ndr.cnt;
      nd.id=ndr.id;
      nd.ans=ndr.ans+ndl.ans;
    }
  }
  void update(int l,int r,int pos,int &now){
    if(!now) now=++cnt;
    if(l==r){
      nd.id=nd.ans=l;
      nd.cnt+=1;
      return ;
    }
    int mid=(l+r)>>1;
    if(pos<=mid) update(l,mid,pos,nd.son[0]);
    else update(mid+1,r,pos,nd.son[1]);
    pushup(now);
  }
  int merge(int now,int b,int l,int r){
    if(!now||!b) return now^b;
    if(l==r){
      nd.id=nd.ans=l;
      nd.cnt+=node[b].cnt;
      return now;
    }
    nd.son[0]=merge(nd.son[0],node[b].son[0],l,(l+r)/2);
    nd.son[1]=merge(nd.son[1],node[b].son[1],(l+r)/2+1,r);
    pushup(now);
    return now;
  }
}tree;
void dfs(int now,int fa){
  for(int to:g[now]){
    if(to==fa) continue;
    dfs(to,now);
    tree.merge(root[now],root[to],1,1e5);
  }
  tree.update(1,1e5,a[now],root[now]);
  ans[now]=tree.node[root[now]].ans;
}
int main() {
  cin>>n;
  rep(i,1,n){
     cin>>a[i];
     root[i]=i;
  }
  cnt=n;
  rep(i,2,n){
    int a,b;cin>>a>>b;
    g[a].push_back(b);
    g[b].push_back(a);
  }
  dfs(1,0);
  rep(i,1,n) cout<<ans[i]<<' ';
}
```

1.15 区间信息-二维线段树

```
class sstree{public:
#define nd node[nowx][nowy]
    struct segnode {int val;};
    int n,m,x1,x2,y1,y2,x,nowx;int ans;
    segnode node[maxn][maxn];
```

```
    void init(int nn,int mm) {
        n=nn,m=mm;
        memset(node ,0,sizeof node);
    }
    void update(int xx1,int xx2,int yy1,int yy2){
        x1=xx1,y1=yy1,x2=xx2,y2=yy2;
        updatex(1,n);
    }
    void updatey(int l,int r,int nowy=1){
        if(y1>r||y2<l) return ;
        if(y1<=l&&y2>=r){nd.val^=1;return ;}
        updatey(l,(l+r)>>1,nowy<<1); updatey(((l+r)>>1)+1,r,nowy<<1|1);
    }
    void updatex(int l,int r,int now=1){
        if(x1>r||x2<l) return ;
        if(x1<=l&&x2>=r) {nowx=now;updatey(1,m);return ;}
        updatex(l,(l+r)>>1,now<<1); updatex(((l+r)>>1)+1,r,now<<1|1);
    }
    int query(int xx,int yy){
        x1=xx,y1=yy;ans=0;
        queryx(1,n);
        return ans;
    }
    void queryy(int l,int r,int nowy=1){
        if(y1>r||y1<l) return ;
        ans^=nd.val;
        if(l==r) return ;
        queryy(l,(l+r)>>1,nowy<<1);queryy(((l+r)>>1)+1,r,nowy<<1|1);
    }
    void queryx(int l,int r,int now=1){
        if(x1>r||x1<l) return ;
    nowx=now;queryy(1,m);
        if(l==r) return ;
        queryx(l,(l+r)>>1,now<<1);queryx(((l+r)>>1)+1,r,now<<1|1);
    }
}tree;
```

1.16 可持久化数据结构-01 字典树

```
int rt[maxn];
class ptrie{public:
  int node[maxn*40][2],top;
  void init(){rt[0]=node[0][1]=node[0][0]=top=0;}
  int add(int pre,int val,int bit=31){
    int now=++top;
    if(bit<0) return now;
    int t=val>>bit&1;
    node[now][t]=add(node[pre][t],val,bit-1);
    node[now][t^1]=node[pre][t^1];
    return now;
  }
  int query(int now,int pre,int val,int bit=31,int ans=0){
    if(bit<0) return ans;
    int t=val>>bit&1;
    if(node[now][t^1]-node[pre][t^1]>0) return query(node[now][t^1],node[pre][t^1],val,bit-1,ans|(1<<bit
        ));
    return query(node[now][t],node[pre][t],val,bit-1,ans);

  }
}tree;
```

1.17 可持久化数据结构-主席树

```
int rt[maxn];//树根
class ptree{public:
#define nd node[now]
#define ndp node[pre]
#define mid (s+t)/2
  int cnt;
  struct segnode{int l,r,sum;}node[maxn*30];
  void maketree(int s,int t,int &now=rt[0]){
    now=++cnt;nd={s,t,0};
    if(s==t) return ;
    maketree(s,mid,nd.l); maketree(mid+1,t,nd.r);
```

```
  }
  void update(int pos,int val,int s,int t,int &now,int pre){
    now=++cnt;nd=ndp;nd.sum+=val;
    if(s==t) return ;
    if(pos<=mid) update(pos,val,s,mid,nd.l,ndp.l);
    else update(pos,val,mid+1,t,nd.r,ndp.r);
  }
  ll query(int l,int r,int s,int t,int now,int pre){
    if(l<=s&&r>=t)return nd.sum-ndp.sum;
    ll sum=0;
    if(l<=mid) sum+=query(l,r,s,mid,nd.l,ndp.l);
    if(r>mid) sum+=query(l,r,mid+1,t,nd.r,ndp.r);
    return sum;
  }
#undef mid
}tree;
```

1.18 可持久化数据结构-主席树区间第 k 大

```
#include<bits/stdc++.h>
#define rep(ii,a,b) for(int ii=a;ii<=b;++ii)
#define all(x) x.begin(),x.end()
using namespace std;//head
const int maxn=2e5+10,maxm=2e6+10;
int casn,n,m,k;
int rt[maxn];//树根
class ptree{public:
#define nd node[now]
#define ndp node[pre]
#define mid (s+t)/2
  int cnt;
  struct segnode{int l,r,sum;}node[maxn*20];
  void init(){cnt=0;}
  void update(int pre,int &now,int pos,int s=1,int t=k){
    now=++cnt;nd=ndp;nd.sum++;
    if(s==t) return ;
    if(pos<=mid) update(ndp.l,nd.l,pos,s,mid);
    else update(ndp.r,nd.r,pos,mid+1,t);
  }
  int query(int pre,int now,int pos,int s=1,int t=k){
    if(s==t) return s;
    int sum=node[nd.l].sum-node[ndp.l].sum;
    if(pos<=sum) return query(ndp.l,nd.l,pos,s,mid);
    else return query(ndp.r,nd.r,pos-sum,mid+1,t);
  }
#undef mid
}tree;
int a[maxn];
vector<int>pos;
int main() {IO;
  cin>>n>>m;
  rep(i,1,n) cin>>a[i];
  rep(i,1,n) pos.push_back(a[i]);
  sort(all(pos));
  pos.erase(unique(all(pos)),pos.end());
  k=pos.size();
  rep(i,1,n){
    int id=lower_bound(all(pos),a[i])-pos.begin();
    tree.update(rt[i-1],rt[i],id+1);
  }
  while(m--){
    int a,b,c;cin>>a>>b>>c;
    cout<<pos[tree.query(rt[a-1],rt[b],c)-1]<<'\n';
  }
  return 0;
}
```

1.19 可持久化数据结构-bit 套主席树

```
namespace dsegtree{
#define nd node[now]
#define ndl node[node[now].son[0]]
#define ndr node[node[now].son[1]]
  struct dsegnode{int son[2],sum;}node[maxn*200];
```

```
  int cnt;
  int pos,s,t,x;
  void update(int l,int r,int &now){
    if(!now) now=++cnt;
    nd.sum+=x;
    if(l==r) return ;
    if(pos<=((l+r)>>1)) update(l,(l+r)>>1,nd.son[0]);
    else update(1+((l+r)>>1),r,nd.son[1]);
  }
  void update_1(int _pos,int _x,int &root){
    pos=_pos,x=_x;
    update(1,n,root);
  }
  int query(int l,int r,int now){
    if(!now||nd.sum==0)return 0;
    if(s<=l&&t>=r) return nd.sum;
    int mid=(l+r)>>1;
    int sum=0;
    if(nd.son[0]&&s<=mid)sum+=query(l,mid,nd.son[0]);
    if(nd.son[1]&&t>mid) sum+=query(mid+1,r,nd.son[1]);
    return sum;
  }
  int query_1(int _s,int _t,int root){
    s=_s,t=_t;
    return query(1,n,root);
  }
}
namespace bit{
  int node[maxn];
  inline void update(int pos,int y,int val){
    for(;pos<=n;pos+=pos&(-pos))
      dsegtree::update_1(y,val,node[pos]);
  }
  inline int ask(int pos,int x,int y){
    int sum=0;
    for(;pos;pos-=pos&(-pos))
      sum+=dsegtree::query_1(x,y,node[pos]);
    return sum;
  }
  inline int query(int l,int r,int x,int y){
    return ask(r,x,y)-ask(l-1,x,y);
  }
}
int aa[maxn];
int main() {
  cin>>n>>m;
  rep(i,1,n) {
    cin>>aa[i];
    if(aa[i]!=aa[i-1]) bit::update(i,aa[i],1);
  }
  register int x,y,b,c,d,e,a;
  while(m--){
    cin>>a;
    if(a==1){
      cin>>x>>y;
      if(aa[x]==y) continue;
      if(aa[x]!=aa[x-1]) bit::update(x,aa[x],-1);
      if(x+1<n&&aa[x+1]!=aa[x]) bit::update(x+1,aa[x+1],-1);
      aa[x]=y;
      if(aa[x]!=aa[x-1]) bit::update(x,aa[x],1);
      if(x+1<n&&aa[x+1]!=aa[x]) bit::update(x+1,aa[x+1],1);
    }else {
      cin>>b>>c>>d>>e;
      int ans=bit::query(b,c,d,e);
      if(aa[b]==aa[b-1]&&aa[b]>=d&&aa[b]<=e) ans++;
      cout<<ans<<endl
    }
  }
}
```

1.20 可持久化数据结构-数组

```
//可持久化数组，主席树实现
```

```
int a0[maxn];//初始数组
int rt[maxn];//树根
template<typename T>class parray{public:
#define nd node[now]
#define ndp node[pre]
#define mid (s+t)/2
  int cnt;
  static int l0,r0;//数组区间
  struct segnode{int l,r;T val;}node[maxn*20];
  void init(int s,int t,T a[]){
    l0=s,r0=t,cnt=0;
    makearray(a);
  }
  void makearray(T a[],int &now=rt[0],int s=l0,int t=r0){
    now=++cnt;
    if(s==t){nd.val=a[s];return;}
    makearray(a,nd.l,s,mid);makearray(a,nd.r,mid+1,t);
  }
  void update(int pre,int &now,int pos,T val,int s=l0,int t=r0){
    now=++cnt;nd=ndp;
    if(s==t){nd.val=val;return;}
    if(pos<=mid) update(ndp.l,nd.l,pos,val,s,mid);
    else update(ndp.r,nd.r,pos,val,mid+1,t);
  }
  T query(int now,int pos,int s=l0,int t=r0){
    if(s==t) return nd.val;
    if(pos<=mid) return query(nd.l,pos,s,mid);
    else return query(nd.r,pos,mid+1,t);
  }
#undef mid
};
parray<int>arr;
```

1.21 可持久化数据结构-并查集

```
//可持久化并查集，主席树实现
int rt[maxn];
int l0,r0;
class pdsu{public:
#define nd node[now]
#define ndp node[pre]
#define mid (s+t)/2
  int cnt;
  struct segnode{int l,r,fa,dep;}node[maxn*30];
  void init(int n){
    l0=1,r0=n,cnt=0;
    makearray();
  }
  void makearray(int &now=rt[0],int s=l0,int t=r0){
    now=++cnt;nd={0,0,s,0};
    if(s==t)return;
    makearray(nd.l,s,mid);makearray(nd.r,mid+1,t);
  }
  void merge(int pre,int &now,int pos,int fa,int s=l0,int t=r0){
    now=++cnt;nd=ndp;
    if(s==t){nd.fa=fa;return;}
    if(pos<=mid) merge(ndp.l,nd.l,pos,fa,s,mid);
    else merge(ndp.r,nd.r,pos,fa,mid+1,t);
  }
  void update(int now,int pos,int s=l0,int t=r0){
    if(s==t) {nd.dep++;return;}
    if(pos<=mid) update(nd.l,pos,s,mid);
    else update(nd.r,pos,mid+1,t);
  }
  int query(int now,int pos,int s=l0,int t=r0){
    if(s==t) return now;
    if(pos<=mid) return query(nd.l,pos,s,mid);
    else return query(nd.r,pos,mid+1,t);
  }
//找到第 ver 个版本的集合根:
  int find(int ver,int pos){
    int now=query(ver,pos);
    if(nd.fa==pos) return now;
```

```
    return find(ver,nd.fa);
  }
//在 ver1 的基础上, 合并 a,b 集合, 得到 ver2:
  void unite(int ver1,int ver2,int a,int b){
      rt[ver2]=rt[ver1];
      int fa=find(rt[ver2],a),fb=find(rt[ver2],b);
      if(node[fa].fa==node[fb].fa) return;
      if(node[fa].dep>node[fb].dep) swap(fa,fb);
      merge(rt[ver1],rt[ver2],node[fa].fa,node[fb].fa);
      if(node[fa].dep==node[fb].dep)
        update(rt[ver2],node[fb].fa);
  }
//复制 ver1 的状态到 ver2, 并查询 a,b 是否为同集合
  bool same(int ver1,int ver2,int a,int b){
    rt[ver2]=rt[ver1];
    int fa=find(rt[ver2],a),fb=find(rt[ver2],b);
    return node[fa].fa==node[fb].fa;
  }
//令 ver2 的状态回到 ver1:
  void popback(int ver1,int ver2){
    rt[ver2]=rt[ver1];
  }
#undef mid
}dsu;
```

1.22 平衡树-老司机树

```
class odt{public:
  struct segnode{
    int l,r;mutable int val;
    bool operator<(const segnode &b)const {return l<b.l;}
  };
  set<segnode> nd;
  void init(int n=maxn-5){nd.insert(1,n,0);}
  #define iter set<segnode>::iterator
  auto split(int pos){
    auto it=nd.lower_bound({pos,pos,0});
    if(it!=nd.end()&&it->l==pos) return it;
    it--;
    int l=it->l,r=it->r,val=it->val;
    nd.erase(it);nd.insert({l,pos-1,val});
    return nd.insert({pos,r,val}).fi;
  }
  void update(int l,int r,int val){
    auto itr=split(r+1);auto itl=split(l);
    nd.erase(itl,itr);nd.insert({l,r,val});
  }
  int query(int pos){
       auto it=nd.lower_bound({pos,pos,0});
        if(it!=nd.end()&&it->l==pos) return it->val;
        it--;return it->val;
    }
}tree;
```

1.23 平衡树-Treap

```
class splaytree{public:
#define nd node[now]
#define ndl node[node[now].son[0]]
#define ndr node[node[now].son[1]]
    struct splaynode{
        int son[2],fa,val,size;
        splaynode(){size=1,fa=son[0]=son[1]=0;}
    };
    int cnt,root;
    vector<splaynode> node;
    inline void pushup(int now){nd.size=ndl.size+ndr.size+1;}
    inline void pushdown(int now){}
    inline int wh(int now){return node[nd.fa].son[1]==now;}
    void rotate(int now){
        int fa=nd.fa,gf=node[fa].fa,c=wh(now);
        pushdown(fa);pushdown(now);
        if(gf) node[gf].son[wh(fa)]=now;
        nd.fa=gf;
```

```
        node[fa].son[c]=nd.son[c^1];
        node[node[fa].son[c]].fa=fa;nd.son[c^1]=fa;node[fa].fa=now;
        pushup(fa);pushup(now);
    }
    void splay(int now,int dst=0){
        for(;nd.fa!=dst;rotate(now))
            if(node[nd.fa].fa!=dst)rotate(wh(now)==wh(nd.fa)?nd.fa:now);
            if(!dst) root=now;
    }
    void insert(int pos){
        int now=root,fa=0,val=node[pos].val;
        while(now) fa=now,now=val<nd.val?nd.son[0]:nd.son[1];
        now=pos;
        node[fa].son[val>node[fa].val]=now;
        nd.fa=fa;
        splay(now);
    }
    void order(int now){
        int l=nd.son[0],r=nd.son[1];
        nd.son[0]=nd.son[1]=nd.fa=0;
        nd.size=1;
        if(l) order(l);
        insert(now);
        if(r) order(r);
    }
    void merge(int a,int b){
        if(a==b) return ;
        splay(a);splay(b);
        if(node[a].size>node[b].size) swap(a,b);
        pre[a]=b;root=b;
        order(a);
    }
    int kth(int now,int k){
        splay(now);int lsize=0;
        while(now){
            int lsum=lsize+ndl.size;
            if(k<=lsum) now=nd.son[0];
            else if(k==lsum+1) return now;
            else lsize=lsum+1,now=nd.son[1];
        }
        return -1;
    }
    splaytree(int n){
        node.resize(n+7,splaynode());
        rep(i,1,n) node[i].val=val[i];
        node[0].size=0;
        root=0,cnt=0;
    }
};
```

1.24 平衡树-Splay

```
int root;//树根
class splaytree{public:
  int fa[maxn],son[maxn][2],sz[maxn],val[maxn],cnt[maxn];
  int tot;//根权值重复次数子树大小
  inline void pushup(int now){
    sz[now]=sz[son[now][0]]+sz[son[now][1]]+cnt[now];
  }//更新当前节点信息
  inline bool getson(int now) {return now==son[fa[now]][1];}//真为右儿子
  inline void clear(int now){
    fa[now]=son[now][0]=son[now][1]=sz[now]=val[now]=cnt[now]=0;
  }//清空节点
  inline void rotate(int now){
    int f=fa[now],gf=fa[fa[now]],flag=getson(now);
    son[f][flag]=son[now][flag^1];
    fa[son[now][flag^1]]=f;
    son[now][flag^1]=f;
    fa[f]=now;fa[now]=gf;
    if(gf) son[gf][f==son[gf][1]]=now;
    pushup(now);pushup(f);
  }//旋转一层
  void splay(int now){
```

```
  for(int f=fa[now];f=fa[now],f;rotate(now))
    if(fa[f]) rotate(getson(now)==getson(f)?f:now);
  root=now;
}//旋转到根
void insert(int x,int now=root,int f=0){
  if(!now){
    now=++tot;
    val[now]=x,cnt[now]++;
    if(!root)root=now;
    pushup(now);
    if(f){
      fa[now]=f;
      son[f][val[f]<x]=now;
      pushup(f);splay(now);
    }
  }else if(val[now]==x){
    ++cnt[now];
    pushup(now);pushup(f);
    splay(now);
  }else insert(x,son[now][val[now]<x],now);
}//插入新点
int getrank(int x,int now=root,int ans=0){
  while(1){
    if(x<val[now]) now=son[now][0];
    else {
      ans+=sz[son[now][0]];
      if(x==val[now]) return splay(now),ans+1;
      ans+=cnt[now];now=son[now][1];
    }
  }
}//多少个元素小于x
int get(int k,int now=root){
  while(1){
    if(son[now][0]&&k<=sz[son[now][0]]) now=son[now][0];
    else {
      k-=cnt[now]+sz[son[now][0]];
      if(k<=0) return val[now];
      now=son[now][1];
    }
  }
}//查询元素
int pre(){
  int now=son[root][0];
  while(son[now][1]) now=son[now][1];
  return now;
}//前驱
int nxt(){
  int now=son[root][1];
  while(son[now][0]) now=son[now][0];
  return now;
}//后缀
int lower(int x){
  insert(x);
  int ans=val[pre()];
  erase(x);return ans;
}//查询前驱
int upper(int x){
  insert(x);
  int ans=val[nxt()];
  erase(x);return ans;
}//查询后缀
void erase(int x){
  getrank(x);
  if(cnt[root]>1){
    --cnt[root];pushup(root);
  }else if(!son[root][0]&&!son[root][1]){
    clear(root);root=0;
  }else if(!son[root][0]||!son[root][1]){
    int t=root;
    if(!son[root][0]) root=son[root][1];
    else root=son[root][0];
```

```
      fa[root]=0;clear(t);
    }else {
      int now=pre(),t=root;
      splay(now);
      fa[son[t][1]]=now;
      son[now][1]=son[t][1];
      clear(t);pushup(root);
    }
  }//删除元素
}tree;
int main() {
  cin>>n;
  while(n--){
    int opt,x;cin>>opt>>x;
    if(opt==1) tree.insert(x);
    if(opt==2) tree.erase(x);
    if(opt==3) cout<<tree.getrank(x)<<endl;
    if(opt==4) cout<<tree.get(x)<<endl;
    if(opt==5) cout<<tree.lower(x)<<endl;
    if(opt==6) cout<<tree.upper(x)<<endl;
  }
  return 0;
}
```

2 Graph

2.1 基础-链式前向星

```
class graph{public:
  struct node{int to,next,cost;};
  node e[maxm];int head[maxn],nume;
  void init(int n=maxn-5){nume=0;fill_n(head,n+1,0);}
  void add(int a,int b,int c){e[++nume]={b,head[a],c};head[a]=nume;}
}
```

2.2 最短路-Dijkstra 算法

```
namespace dij{
  struct road{
    int now;ll dis;
    road(int a=0,ll _dis=0):now(a),dis(_dis){}
    bool operator<(const road &x)const {return dis>x.dis};
  }
  ll dis[maxn];
  bool vis[maxn];
  priority_queue<road>que;
  void cal(int st,int n=maxn-5){
    fill_n(dis,n+1,1e18);fill_n(vis,n+1,0);
    que.emplace(st,0);dis[st]=0;
    while(!que.empty()){
      road t=que.top();que.pop();
      for(auto e:g[t.now]){
        ll cost=t.dis+e.cost;
        if(cost<dis[e.to]){
          dis[e.to]=cost;
          que.emplace(e.to,cost);
        }
      }
    }
  }
}
template<typename T> class shortpath{public:
  T dis[maxn];
  bool vis[maxn];
  #define pdi pair<T,int>
  priority_queue<pdi,vector<pdi>,greater<pdi> >que;
  void cal(int st,int n,vector<pdi> g[]){
    fill_n(dis,n+1,INF);fill_n(vis,n+1,0);
    que.emplace(0,st);dis[st]=0;
    while(!que.empty()){
      pdi t=que.top();que.pop();
      for(auto e:g[t.se]){
        T cost=t.fi+e.fi;
        if(cost<dis[e.se]){
          dis[e.se]=cost;
```

```
        que.emplace(cost,e.se);
      }
    }
  }
}
};
```

2.3 最短路-Floyd 求最小环

```
const int inf=500;
const int N=105;
int n,m,ans,num;
int dis[N][N],mapp[N][N],pre[N][N],path[N];
void floyd() {
  ans=inf;
  for(int k=1; k<=n; k++) {
    for(int i=1; i<=n; i++) {
      for(int j=1; j<=n; j++) {
        if(i==j||k==i||k==j) {
          continue;
        }
        if(dis[i][j]+mapp[i][k]+mapp[k][j]<ans) {
          ans=dis[i][j]+mapp[i][k]+mapp[k][j];
          num=0;
          path[num++]=k;
          int temp=j;
          while(temp!=i) {
            path[num++]=temp;
            temp=pre[i][temp];
          }
          path[num++]=i;
        }
      }
    }
    for(int i=1; i<=n; i++) {
      for(int j=1; j<=n; j++) {
        if(i==j||k==i||k==j) {
          continue;
        }
        if(dis[i][j]>dis[i][k]+dis[k][j]) {
          dis[i][j]=dis[i][k]+dis[k][j];
          pre[i][j]=pre[k][j];
        }
      }
    }
  }
}
int main() {
  cin>>n>>m;
  for(int i=1; i<=n; i++) {
    for(int j=1; j<=n; j++) {
      pre[i][j]=i;
      if(i==j) {
        mapp[i][j]=dis[i][j]=0;
      } else {
        mapp[i][j]=dis[i][j]=inf;
      }
    }
  }
  int u,v,w;
  for(int i=1; i<=m; i++) {
    cin>>u>>v>>w;
    mapp[u][v]=mapp[v][u]=dis[u][v]=dis[v][u]=min(w,mapp[u][v]);
  }
  floyd();
  if(ans==inf) {
    printf("No solution.\n");
  } else {
    for(int i=num-1; i>=0; i--) {
      printf("%d",path[i]);
      if(i==0) {
        printf("\n");
      } else {
```

```
      printf(" ");
    }
  }
 }
 return 0;
}
```

2.4 强连通-求桥

```
struct node{int to,nx;}e[maxm],e2[maxm];
int head[maxn],head2[maxn],nume,nume2;
bool bg[maxm];int dfn[maxn],low[maxn];
int numc,cnt,vis1[maxn],belong[maxn];
void add(int a,int b){
 e[++nume]={b,head[a]};head[a]=nume;
}
void add2(int a,int b){
 e2[++nume2]={b,head2[a]};head2[a]=nume2;
}
void tdfs(int now,int in){
 dfn[now]=low[now]=++cnt;
 for(int i=head[now];i;i=e[i].nx){
  int to=e[i].to;
  if(!dfn[to]) {
   tdfs(to,i);
   low[now]=min(low[now],low[to]);
   if(low[to]>dfn[now]) bg[i]=bg[i^1]=1;
  }else if(i!=(in^1))low[now]=min(low[now],dfn[to]);
 }
}
void dfs(int now){
 belong[now]=numc;
 for(int i=head[now];i;i=e[i].nx){
  int to=e[i].to;
  if(belong[to]||bg[i]) continue;
  dfs(to);
 }
}
pii dfs2(int now,int fa,int d=0){
 vis1[now]=1;pii x={d,now};
 for(int i=head2[now];i;i=e2[i].nx){
  int to=e2[i].to;
  if(to==fa) continue;
  pii t=dfs2(to,now,d+1);
  if(x<t) x=t;
 }
 return x;
}
int main() {
 cin>>casn;
 while(casn--){
  cin>>n>>m;
  rep(i,1,n){
   belong[i]=head[i]=head2[i]=0;
   low[i]=vis1[i]=dfn[i]=0;
  }
  rep(i,1,m*2+2)bg[i]=0;
  cnt=numc=0,nume=nume2=1;
  while(m--){
   int a,b;cin>>a>>b;
   add(a,b);add(b,a);
  }
  rep(i,1,n) if(!dfn[i]) tdfs(i,0);
  rep(i,1,n)
   if(!belong[i]) {
    numc++;dfs(i);
   }
  k=0;
  for(int i=2;i<=nume;i+=2){
   int a=e[i].to,b=e[i^1].to;
   if(belong[a]!=belong[b]){
    k++;
    add2(belong[a],belong[b]);
```

```
      add2(belong[b],belong[a]);
    }
  }
  int c=0;
  rep(i,1,numc){
    if(vis1[i]) continue;
    int t=dfs2(i,i).se;
    c=max(c,dfs2(t,t).fi);
  }
  cout<<k-c<<endl;
 }
}
```

2.5 强连通-求割顶

```
void init(int n){
 for(int i=0;i<=n;i++){if(i!=0) f[i]=i;}
 memset(cut,false,n<<2);memset(low,0,n<<2);
 memset(dfn,0,n<<2);memset(head,0,n<<2);
 nume=cnt=0;
}
void tarjan(int u,int p){
 low[u]=dfn[u]=++cnt;
 int son=0;
 for(int i=head[u];i;i=e[i].next){
  int v=e[i].to;
  if(v==p)continue;
  if(!dfn[v]){
   son++;
   tarjan(v,u);
   low[u]=min(low[u],low[v]);
   if(u!=p&&low[v]>=dfn[u]){
    cut[u]=true;
   }
  } else low[u]=min(low[u],dfn[v]);
 }
 if(u==p&&son>1) cut[u]=true;
}
```

2.6 强连通-缩点

```
int casn,n,m,k;
struct node {int to,next;}e[maxm];int head[maxn],nume;
inline void add(int a,int b){e[++nume]=(node){b,head[a]};head[a]=nume;}
namespace tarjan{
 int stk[maxn],top,cnt,dfn[maxn],low[maxn],numc,belong[maxn],vis[maxn];
 vector<int>g[maxn];
 void tdfs(int now,int fa){
  dfn[now]=low[now]=++cnt;
  stk[top++]=now,vis[now]=1;
  for(int i=head[now];i;i=e[i].next){
   int to=e[i].to;
   if(!dfn[to]){tdfs(to,now);low[now]=min(low[now],low[to]);}
   else if(vis[to]) low[now]=min(low[now],dfn[to]);
  /*
   if(to==fa) continue;
   if(!dfn[to]){tdfs(to,now);low[now]=min(low[now],low[to]);}
   else low[now]=min(low[now],dfn[to]);
  */
  }
  if(low[now]==dfn[now]){
   numc++;
   int to;
   do{to=stk[--top];
    belong[to]=numc;
    vis[to]=0;
   }while(to!=now);
  }
 }
 void makegraph(int n){
  for(int i=1;i<=n;i++) if(!dfn[i]) tdfs(i,i);
  rep(i,1,n){
   int a=belong[i];
   for(int j=head[i];j;j=e[j].next){
```

```
      int b=belong[e[j].to];
      if(a!=b){
        g[a].emplace_back(b);
        g[b].emplace_back(a);
      }
    }
  }
}
```

2.7 强连通-无向图双连通

```
int stk[maxn],top,cnt,dfn[maxn],low[maxn],numc,belong[maxn],vis[maxn],sz[maxn];
struct node {int to,next;}e[maxm];int head[maxn],nume;
inline void add(int a,int b){e[++nume]=(node){b,head[a]};head[a]=nume;}
void tdfs(int now,int pre){
  dfn[now]=low[now]=++cnt;
  stk[top++]=now,vis[now]=1;
  for(int i=head[now];i;i=e[i].next){
    int to=e[i].to;
    if(to==pre) continue;
    if(!dfn[to]){tdfs(to,now);low[now]=min(low[now],low[to]);}
    else low[now]=min(low[now],dfn[to]);
  }
  if(low[now]==dfn[now]){
    numc++;
    int to;
    do{to=stk[--top];
      belong[to]=numc;
      sz[numc]++;
    }while(to!=now);
  }
}
```

2.8 树-Prim 算法

```
//最小生成树 prim 算法
int head[30],next[200],point[200],val[200],size,dist[30];
bool vis[30];
void add (int a,int b, int v){
  int i;
  for(i=head[a];~i;i=next[i]){
    if(point[i]==b){
      if(val[i]>v)val[i]=v;
      return;
    }
  }
  point[size]=b;
  val[size]=v;
  next[size]=head[a];
  head[a]=size++;
}
struct cmp{
  bool operator()(pii a,pii b){
    return a.first>b.first;
  }
};
void prim(int s){
  int i,ans=0;
  memset(dist,-1,sizeof(dist));
  memset(vis,0,sizeof(vis));
  priority_queue<pii,vector<pii>,cmp>q;
  for (i=head[s];~i;i=next[i]){
    dist[point[i]]=val[i];
    q.push(make_pair(dist[point[i]],point[i]));
  }
  dist[s]=0;
  vis[s]=1;
  while(!q.empty()){
    pii u=q.top();
    q.pop();
    if(vis[u.second])continue;
    vis[u.second]=1;
    ans+=u.first;
```

```
    for(i=head[u.second];~i;i=next[i]){
      int j=point[i];
      if(!vis[j]&&(dist[j]>val[i]||dist[j]==-1)){
        dist[j]=val[i];
        q.push(make_pair(dist[j],j));
      }
    }
  }
  printf("%d\n",ans);
}
```

2.9 树-灭绝树算法

```
int cntin[maxn];
struct node{int to,next;};
class graph{public:
  node e[maxn];int head[maxn],nume;
  void init(int n=maxn-5){nume=0;fill_n(head,n+1,0);}
  void add(int a,int b){e[++nume]={b,head[a]};head[a]=nume;}
}inv,nxt,dom;
class domtree{public://DAG
  int deep[maxn],anc[maxn][maxp],que[maxn];
  vector<int>edge;
  void init(int n=maxn-5){
    inv.init(n),nxt.init(n),dom.init(n);
    edge.clear();
  }
  void bit(int &x,int h){
    for(int i=0;h>0;++i){
      if(h&1) x=anc[x][i];
      h>>=1;
    }
  }
  void add(int a,int b){nxt.add(a,b);inv.add(b,a);}
  int lca(int a,int b){
    if(deep[a]<deep[b]) swap(a,b);
    bit(a,deep[a]-deep[b]);
    if(a==b) return a;
    per(i,0,maxp-1) if(anc[a][i]!=anc[b][i])
      a=anc[a][i],b=anc[b][i];
    return anc[a][0];
  }
  void tpsort(int n){
    int tp=0,ed=0;
    rep(i,1,n) {
      if(!cntin[i]) {
        que[ed++]=i;
        inv.add(0,i);
        nxt.add(i,0);
        edge.push_back(i);
      }
    }
    while(ed!=tp){
      int now=que[tp++];
      forn(i,now,inv.head,inv.e){
        int to=inv.e[i].to;
        cntin[to]--;
        if(!cntin[to]) que[ed++]=to,edge.push_back(to);
      }
    }
  }
  void maketree(int n){
    tpsort(n);
    for(auto i:edge){
      int fa=-1;
      forn(j,i,nxt.head,nxt.e){
        int to=nxt.e[j].to;
        if(fa==-1) fa=to;
        else fa=lca(fa,to);
      }fa=fa==-1?0:fa;
      deep[i]=deep[fa]+1,anc[i][0]=fa;
      rep(j,1,maxp-1) anc[i][j]=anc[anc[i][j-1]][j-1];
      dom.add(fa,i);
```

```
    }
  }
  int ans[maxn];
  int cal(int now){
    ans[now]=1;
    forn(i,now,dom.head,dom.e) ans[now]+=cal(dom.e[i].to);
    return ans[now];
  }
}tree;
```

2.10 树-支配树算法

```
int root;
class domtree{public://dom 为最终的支配树,root 为根,cnt 为每个点的支配点的数量
  int dfn[maxn],rev[maxn],anc[maxn];
  int semi[maxn],idom[maxn];
  int fa[maxn],mi[maxn],clo;
  struct node{int to,next;};
  struct graph{
    node e[maxn];int head[maxn],nume;
    void init(int n=maxn-5){nume=0;fill_n(head,n+1,0);}
    void add(int a,int b){e[++nume]={b,head[a]};head[a]=nume;}
  }inv,nxt,dom;
  void init(int n=maxn-5){
    clo=0;
    rep(i,1,n)fa[i]=mi[i]=semi[i]=i,rev[i]=dfn[i]=anc[i]=idom[i]=0;
    nxt.init(n),inv.init(n),dom.init(n);
  }
  void add(int a,int b){inv.add(b,a),nxt.add(a,b);}
  int find(int now){
    if(fa[now]==now) return now;
    int fx=fa[now],y=find(fa[now]);
    if(dfn[semi[mi[fx]]]<dfn[semi[mi[now]]])
      mi[now]=mi[fx];
    return fa[now]=y;
  }
  void tdfs(int now){
    dfn[now]=++clo;rev[clo]=now;
    forn(i,now,nxt.head,nxt.e)if(!dfn[nxt.e[i].to])
      anc[nxt.e[i].to]=now,tdfs(nxt.e[i].to);
  }
  void maketree(int root,int n=maxn-5){
    tdfs(root);
    per(i,2,n){
      int now=rev[i],tmp=n;
      forn(i,now,inv.head,inv.e){
        int to=inv.e[i].to;if(!dfn[to]) continue;
        if(dfn[to]<dfn[now]) tmp=min(tmp,dfn[to]);
        else find(to),tmp=min(tmp,dfn[semi[mi[to]]]);
      }
      semi[now]=rev[tmp];fa[now]=anc[now];
      dom.add(semi[now],now);
      now=rev[i-1];
      forn(i,now,dom.head,dom.e){
        int to=dom.e[i].to;find(to);
        if(semi[mi[to]]==now) idom[to]=now;
        else idom[to]=mi[to];
      }
    }
    rep(i,2,n){
      int to=rev[i];
      if(idom[to]!=semi[to]) idom[to]=idom[idom[to]];
    }
    dom.init(n);
    rep(i,1,n) if(i!=root)dom.add(idom[i],i);
  }
}tree;
```

2.11 树-最小树形图

```
//定根最小树形图
struct node{int a,b,c;}e[maxm];
int in[maxn],pre[maxn],vis[maxn],id[maxn];
ll mdst(){
```

```
  ll ans=0;int cnt=0,a,b,laz;
  while(1){
    rep(i,1,n) in[i]=INF,id[i]=vis[i]=0;
    rep(i,1,m) if(e[i].a^e[i].b&&e[i].c<in[e[i].b])
      pre[e[i].b]=e[i].a,in[e[i].b]=e[i].c;
    in[k]=0;
    rep(i,1,n){
      if(in[i]==INF) return -1;
      ans+=in[i];
      for(a=i;a^k&&vis[a]^i&&!id[a];a=pre[a])vis[a]=i;
      if(a^k&&!id[a]){
        id[a]=++cnt;
        for(b=pre[a];a^b;b=pre[b])id[b]=cnt;
      }
    }
    if(!cnt) return ans;
    rep(i,1,n) if(!id[i]) id[i]=++cnt;
    rep(i,1,m) {
      laz=in[e[i].b];
      if((e[i].a=id[e[i].a])^(e[i].b=id[e[i].b]))
        e[i].c-=laz;
    }
    n=cnt;k=id[k],cnt=0;
  }
}
int main() {IO;
  cin>>n>>m>>k;
  rep(i,1,m)cin>>e[i].a>>e[i].b>>e[i].c;
  cout<<mdst()<<endl;
}
```

2.12 树-RMQ 求 LCA+ 树上链交

```
//rmq 求 lca+ 快速求树上链交
const int maxp=18;
class graph{public:
  struct node{int to,next;}e[maxn<<1];
  int head[maxn],nume,dfn[maxn],deep[maxn];
  int logn[maxn],pos[maxp][maxn],cnt;
  inline void add(int a,int b){e[++nume]={b,head[a]};head[a]=nume;}
  void init(int n){rep(i,1,n) head[i]=0;cnt=0,nume=1;}
  void cal(int n){
    logn[2]=1;
    rep(i,3,n) logn[i]=logn[i>>1]+1;
    for(int j=1;(1<<j)<=n;j++) for(int i=1;i+(1<<j)-1<=n;++i){
      int r=i+(1<<(j-1));
      if(deep[pos[j-1][i]]<deep[pos[j-1][r]]) pos[j][i]=pos[j-1][i];
      else pos[j][i]=pos[j-1][r];
    }
  }
  void dfs(int now=root,int fa=root,int d=1){
    dfn[now]=++cnt;deep[now]=d;pos[0][cnt]=now;
    forn(i,now){
      if(e[i].to==fa) continue;
      dfs(e[i].to,now,d+1);pos[0][++cnt]=now;
    }
  }
  inline int lca(int l,int r){
    l=dfn[l],r=dfn[r];if(l>r) swap(l,r);
    int lg=logn[r-l+1];
    if(deep[pos[lg][l]]<deep[pos[lg][r-(1<<lg)+1]])return pos[lg][l];
    else return pos[lg][r-(1<<lg)+1];
  }
  inline int getdis(int a,int b){return deep[a]+deep[b]-2*deep[lca(a,b)];}
  void getlca(){dfs();cal(cnt);}
  inline bool check(int a,int b){return lca(a,b)==a;}
  int getans(int a1,int a2,int b1,int b2){
    int ra=lca(a1,a2);
    bool f1=check(ra,b1),f2=check(ra,b2);
    if(!f1&&!f2) return 0;
    if(f1&&f2){
      int rb=lca(b1,b2);
      if(!( check(rb,a1)||check(rb,a2)))return 0;
```

```
    int r1=lca(a1,b1),r2=lca(a1,b2);
    int r3=lca(a2,b1),r4=lca(a2,b2);
    if(r1==r3&&r2==r4) return 1;
    return getdis(r1==ra?r3:r1,r2==ra?r4:r2)+1;
  }
  if(!f1)swap(b1,b2);
  int r1=lca(a1,b1),r3=lca(a2,b1);
  return getdis(r1==ra?r3:r1,ra)+1;
 }
}g;
```

2.13 树-点分治

```
namespace graph{
 vector<int>g[maxn];
 int all,sz[maxn],root,maxt;
 bool vis[maxn];
 int dfs_root(int now,int fa){
  int cnt=1;
  for(auto to:g[now])if(to!=fa&&!vis[to])
   cnt+=dfs_root(to,now);
  int tmp=max(cnt-1,all-cnt);
  if(maxt>tmp) maxt=tmp,root=now;
  return sz[now]=cnt;
 }//基础部分
 int ans[maxn];
 void dfs_col(int now,int fa,int c){
  ans[now]=c;
  for(auto to:g[now])if(to!=fa&&!vis[to])
   dfs_col(to,now,c);
 }
 void dfs_dv(int now,int d=0){
  vis[now]=1;dfs_col(now,now,d);
  for(auto to:g[now]){
   if(vis[to]) continue;
   maxt=root=n+1;all=sz[to];
   dfs_root(to,now);dfs_dv(root,d+1);
  }
 }
 void solve(int n){
  all=maxt=root=n+1;
  dfs_root(1,1);
  all-=maxt;
  dfs_dv(root);
 }
}
```

2.14 树-轻重链剖分

```
class graph{//按边
 struct node{int from,to,cost,next;}e[maxn<<1];
 int head[maxn],nume,cnt2;
 inline void add(int a,int b,int c){
  e[++nume]={a,b,c,head[a]};head[a]=nume;
 }
 int fa[maxn],sz[maxn],top[maxn],remp[maxn],ans[maxn];
 int son[maxn],in[maxn],cnt,deep[maxn];
 void dfs1(int now,int pre,int d){
  deep[now]=d;sz[now]=1;fa[now]=pre;
  for(int i=head[now];i;i=e[i].next){
   if(e[i].to==pre) continue;
   dfs1(e[i].to,now,d+1);
   sz[now]+=sz[e[i].to];
   if(sz[son[now]]<sz[e[i].to]) son[now]=e[i].to;
  }
 }
 void dfs2(int now,int pre,int st){
  top[now]=st;in[now]=++cnt;remp[cnt]=now;
  if(son[now]) dfs2(son[now],now,st);
  for(int i=head[now];i;i=e[i].next)
   if(e[i].to!=pre&&e[i].to!=son[now])
    dfs2(e[i].to,now,e[i].to);
 }
 int query(int a,int b){
```

```
    int sum=0;
    while(top[a]!=top[b]){
      if(deep[top[a]]<deep[top[b]]) swap(a,b);
      sum+=tree.query(in[top[a]],in[a]);
      a=fa[top[a]];
    }
    if(a==b)return sum;
    if(deep[a]>deep[b]) swap(a,b);
    sum+=tree.query(in[son[a]],in[b]);
    return sum;
  }
  void getchain(){dfs1(1,1,0);dfs2(1,1,1);}
}g;
int root=1;
class graph{public://按点
  struct node{int to,next;}e[maxn<<1];
  int head[maxn],nume,mp[maxn];
  inline void add(int a,int b){
    e[++nume]={b,head[a]};
    head[a]=nume;
  }
  int ltop[maxn],fa[maxn],deep[maxn];
  int sz[maxn],remp[maxn];
  int son[maxn],cnt;
  void init(int n){rep(i,1,n) head[i]=0;cnt=0,nume=1;}
  void dfs1(int now=root,int pre=root,int d=0){
    deep[now]=d,fa[now]=pre,sz[now]=1,son[now]=0;
    forn(i,now){
      int to=e[i].to;
      if(to!=pre) {
        dfs1(to,now,d+1);
        sz[now]+=sz[to];
        if(sz[to]>sz[son[now]]) son[now]=to;
      }
    }
  }
  void dfs2(int now=root,int pre=root,int sp=root){
    ltop[now]=sp;mp[now]=++cnt;remp[cnt]=now;
      if(son[now]) dfs2(son[now],now,sp);
      forn(i,now){
        int to=e[i].to;
        if(to!=son[now]&&to!=pre) dfs2(to,now,to);
      }
  }
  void getchain(){dfs1();dfs2();}
  int lca(int x,int y){
    for(;ltop[x]!=ltop[y];deep[ltop[x]]>deep[ltop[y]]?x=fa[ltop[x]]:y=fa[ltop[y]]);
    return deep[x]<deep[y]?x:y;
  }
  inline int getdis(int a,int b){return deep[a]+deep[b]-2*deep[lca(a,b)];}
  inline bool check(int a,int b){return dfn[a]<=dfn[b]&&dfn[a]+sz[a]-1>=dfn[b]+sz[b]-1;}
  //基础部分
  void update(int a,int b,int val){
    while(ltop[a]!=ltop[b]){
        if(deep[ltop[a]]<deep[ltop[b]])swap(a,b);
        tree.update(mp[ltop[a]],mp[a],val);
        a=fa[ltop[a]];
    }
      if(deep[a]>deep[b])swap(a,b);
      tree.update(mp[a],mp[b],val);
  }
  int query(int a,int b,int k){
    int sum=0;
    while(ltop[a]!=ltop[b]){
        if(deep[ltop[a]]<deep[ltop[b]])swap(a,b);
        sum+=tree.query(mp[ltop[a]],mp[a],k);
        a=fa[ltop[a]];
    }
      if(deep[a]>deep[b])swap(a,b);
      sum+=tree.query(mp[a],mp[b],k);
      return sum;
  }
```

```
};
```

2.15 树-动态点分治

```
//动态点分治 (点分树)
template<const int N>class Graph{public:
  vector<int>g[N+10];
  bool vis[N+10];
  int all,sz[N+10],root,maxt,father[N+10];
  int _deep[N+10],_dfn[N+10],_cnt;
  int son[N+10];
  int _lca[N+10][int(log(N+10)/log(2))+1];
  int logn[N+10];
  void add(int a,int b){
    g[a].emplace_back(b);g[b].emplace_back(a);
  }
  int dfs_root(int now,int fa){
    int cnt=1;
    son[now]=0;
    for(auto to:g[now])if(to!=fa&&!vis[to]){
      int ch=dfs_root(to,now);
      son[now]=max(son[now],ch);
      cnt+=ch;
    }
    son[now]=max(son[now],all-cnt);
    if(son[now]<son[root]) root=now;
    return sz[now]=cnt;
  }
  void dfs_lca(int now,int fa){
    _dfn[now]=++_cnt;
    _lca[_cnt][0]=_deep[now]=_deep[fa]+1;
    for(auto to:g[now]) {
      if(to==fa) continue;
      dfs_lca(to,now);
      _lca[++_cnt][0]=_deep[now];
    }
  }
  void cal_st(){
    logn[0]=-1;
    rep(i,1,2e5+10) logn[i]=logn[i>>1]+1;
    rep(j,1,logn[_cnt])rep(i,1,_cnt-(1<<j)+1)
      _lca[i][j]=min(_lca[i][j-1],_lca[i+(1<<(j-1))][j-1]);
  }
  int _dis(int a,int b){
    a=_dfn[a],b=_dfn[b];
    if(a>b) swap(a,b);
    int len=logn[b-a+1];
    return min(_lca[a][len],_lca[b-(1<<len)+1][len]);
  }
  int dis(int a,int b){//两点距离
    int res=_deep[a]+_deep[b]-2*_dis(a,b);
    return res;
  }
  void dfs_cal(int now,int fa){
    for(auto to:g[now]){
      if(to==fa||vis[to]) continue;
      dfs_cal(to,now);
    }
  }
  void dfs_dv(int now,int fa){
    father[now]=fa;vis[now]=1;
    dfs_cal(now,0);
    for(auto to:g[now]){
      if(vis[to]) continue;
      all=sz[to],root=0;
      dfs_root(to,0);
      int tmp=root;
      dfs_dv(root,now);
    }
  }
  void init(int n){//初始化
    _cnt=0;
    dfs_lca(1,0);
```

```
    cal_st();
    son[0]=n;
    all=n;root=0;
    dfs_root(1,0);
    dfs_dv(root,0);
  }
  void update(int pos){
    for(int now=pos;father[now];now=father[now]){
      //操作
    }
  }
};
Graph<1e6> tree;
```

2.16 最大流-ISAP 算法

```
template<typename T>class mxf{public:
  struct node{int to,next;T cap;}e[maxm<<1];
  int cur[maxn],head[maxn],dis[maxn],gap[maxn];
  int nume=1,s,t,tot;
  void init(int n){
    rep(i,0,n) head[i]=gap[i]=dis[i]=0;
    nume=1;
  }
  void add(int a,int b,T c){
    e[++nume]={b,head[a],c};head[a]=nume;
    e[++nume]={a,head[b],0};head[b]=nume;
  }
  T dfs(int now,T flow=INF){
    if (now==t||!flow) return flow;
    T use=0,tmp;
    int d=dis[now]-1,to;
    for (int &i=cur[now];i;i=e[i].next) {
      if(dis[to=e[i].to]==d&&(tmp=e[i].cap)){
        e[i].cap-=(tmp=dfs(to,min(flow-use,tmp)));
        e[i^1].cap+=tmp;
        if((use+=tmp)==flow) return use;
      }
    }
    if (!--gap[dis[now]]) dis[s]=tot+1;
    ++gap[++dis[now]];
    cur[now]=head[now];
    return use;
  }
  T getflow(int ss,int tt,int n,T ans=0){
    tot=n;s=ss;t=tt;gap[0]=tot;
    memcpy(cur,head,(tot+1)<<2);
    while(dis[s]<=tot) ans+=dfs(s);
    return ans;
  }
};
mxf<int> net;
```

2.17 最大流-Dinic 算法

```
template<typename T>class mxf{public:
  struct node{int to,next;T cap;}e[maxm<<1];
  int cur[maxn],head[maxn],que[maxn],dis[maxn];
  int nume=1,s,t,tot,tp,ed;
  inline void adde(int a,int b,T c){e[++nume]={b,head[a],c};head[a]=nume;}
  inline void add(int a,int b,T c){adde(a,b,c);adde(b,a,0);}
  void init(int n=maxn-1){memset(head,0,(n+1)<<2);nume=1;}
  bool bfs(){
    rep(i,0,ed) dis[que[i]]=0;
    dis[t]=1,que[0]=t;
    tp=0,ed=1;
    cur[t]=head[t];
    int now,to;
    while(tp!=ed) for(int i=head[now=que[tp++]];i;i=e[i].next)
      if(dis[to=e[i].to]==0&&e[i^1].cap>0){
        cur[to]=head[to];
        dis[to]=dis[now]+1;
        if((que[ed++]=to)==s) return true;
      }
```

```
    return false;
  }
  T dfs(int now,T flow=INF){
    if(now==t||flow==0) return flow;
    int to,d=dis[now]-1;
    T use=0,tmp;
    for(int &i=cur[now];i;i=e[i].next){
      if(dis[to=e[i].to]!=d||!(tmp=e[i].cap))continue;
      e[i].cap-=(tmp=dfs(to,min(tmp,flow-use)));
      e[i^1].cap+=tmp,use+=tmp;
      if(use==flow) return use;
    }
    if(use==0) dis[now]=-1;
    return use;
  }
  T getflow(int ss,int tt,int n,T ans=0){
    s=ss,t=tt,tot=n;
    while(bfs())ans+=dfs(s);
    return ans;
  }
};
mxf<int> net;
```

2.18 网络流-上下界可行流-Dinic

```
//上下界可行流
template<typename T>class mxf{public:
  struct node{int id,to,rev;T cap;}w[maxm<<1],e[maxm<<1];
  int cur[maxn],head[maxn],dis[maxn],que[maxn];
  T up[maxm<<1],in[maxn],sum,ans[maxm<<1];
  int num[maxn],numv,nume,s,t,tot,tp,ed,fr[maxm<<1];
  bool bfs(){
    rep(i,0,ed) dis[que[i]]=0;
    dis[t]=1,que[0]=t;
    tp=0,ed=1;
    cur[t]=head[t];
    int now,to;
    while(tp!=ed) for(int i=head[now=que[tp++]];i<=num[now];++i)
      if(dis[to=e[i].to]==0&&e[e[i].rev].cap>0){
        cur[to]=head[to];
        dis[to]=dis[now]+1;
        if((que[ed++]=to)==s) return true;
      }
    return false;
  }
  T dfs(int now,T flow=0x3f3f3f3f){
    if(now==t||flow==0) return flow;
    int to,d=dis[now]-1;
    T use=0,tmp;
    for(int &i=cur[now];i<=num[now];++i){
      if(dis[to=e[i].to]!=d||!(tmp=e[i].cap))continue;
      e[i].cap-=(tmp=dfs(to,min(tmp,flow-use)));
      e[e[i].rev].cap+=tmp,use+=tmp;
      if(use==flow) return use;
    }
    if(use==0) dis[now]=-1;
    return use;
  }
  T getflow(int ss,int tt,int n,T ans=0){
    s=ss,t=tt,tot=n;
    while(bfs())ans+=dfs(s);
    return ans;
  }
  void init(int n){
    rep(i,0,n) num[i]=in[i]=head[i]=dis[i]=0;
    nume=0;tot=n;sum=0;
  }
  void add(int a,int b,T c,int id){
    w[++nume]=(node){id,b,0,c};++num[a],fr[nume]=a;
    w[++nume]=(node){0,a,0,0}; ++num[b],fr[nume]=b;
  }
  void addbound(int a,int b,T c,T d,int id){
    add(a,b,d-c,id);
```

```
    up[id]=d,in[b]+=c,in[a]-=c;
  }
  bool fesbflow(int n){
    s=n+1,t=n+2;numv=n;tot=t;
    rep(i,1,numv){
      if(in[i]>0) add(s,i,in[i],0),sum+=in[i];
      if(in[i]<0) add(i,t,-in[i],0);
    }
    head[1]=1;
    rep(i,2,tot) head[i]=head[i-1]+num[i-1];
    rep(i,1,tot-1) num[i]=head[i+1]-1;
    num[tot]=nume;
    rep(i,1,nume){
      e[head[fr[i]]+cur[fr[i]]++]=w[i];
      if(!(i%2)){
        e[head[fr[i]]+cur[fr[i]]-1].rev=head[w[i].to]+cur[w[i].to]-1;
        e[head[w[i].to]+cur[w[i].to]-1].rev=head[fr[i]]+cur[fr[i]]-1;
      }
    }
    T flow=getflow(s,t,t);
    if(flow<sum) return 0;
    rep(i,1,nume){
      node &x=e[i];
      if(x.id) ans[x.id]=up[x.id]-x.cap;
    }
    return 1;
  }
};
mxf<int> net;
```

2.19 网络流-上下界网络流-ISAP

```
template<typename T>class mxf{public:
  struct node{int to,rev;T cap;}w[maxm<<1],e[maxm<<1];
  int cur[maxn],head[maxn],dis[maxn],gap[maxn];
  int num[maxn],numv,nume,s,t,tot,last,fr[maxm<<1];
  T in[maxn],sum;
  T dfs(int now,T flow=INF){
    if (now==t||!flow) return flow;
    T use=0,tmp;
    int d=dis[now]-1,to;
    for (int &i=cur[now];i<=num[now];++i) {
      if(dis[to=e[i].to]==d&&(tmp=e[i].cap)){
        e[i].cap-=(tmp=dfs(to,min(flow-use,tmp)));
        e[e[i].rev].cap+=tmp;
        if((use+=tmp)==flow) return use;
      }
    }
    if (!--gap[dis[now]]) dis[s]=tot+1;
    ++gap[++dis[now]];
    cur[now]=head[now];
    return use;
  }
  T getflow(int ss,int tt,int n,T ans=0){
    rep(i,0,n)dis[i]=gap[i]=0;
    tot=n;s=ss;t=tt;gap[0]=tot;
    memcpy(cur,head,(tot+1)<<2);
    while(dis[s]<=tot) ans+=dfs(s);
    return ans;
  }
  void init(int n){
    rep(i,0,n) num[i]=in[i]=head[i]=dis[i]=0;
    nume=0;tot=n;sum=0;
  }
  void add(int a,int b,T c){
    w[++nume]=(node){b,0,c};++num[a],fr[nume]=a;
    w[++nume]=(node){a,0,0}; ++num[b],fr[nume]=b;
  }
  void addbound(int a,int b,T c,T d){
    add(a,b,d-c);
    in[b]+=c,in[a]-=c;
  }
  void makeflow(int n){
```

```
    s=n+1,t=n+2;numv=n;tot=t;
    rep(i,1,numv){
      if(in[i]>0) add(s,i,in[i]),sum+=in[i];
      if(in[i]<0) add(i,t,-in[i]);
    }
    head[1]=1;
    rep(i,2,tot) head[i]=head[i-1]+num[i-1];
    rep(i,1,tot-1) num[i]=head[i+1]-1;
    num[tot]=nume;
    rep(i,1,nume){
      e[head[fr[i]]+cur[fr[i]]++]=w[i];
      if(!(i%2)){
        e[head[fr[i]]+cur[fr[i]]-1].rev=head[w[i].to]+cur[w[i].to]-1;
        e[head[w[i].to]+cur[w[i].to]-1].rev=head[fr[i]]+cur[fr[i]]-1;
      }
    }
  }
  T fesbflow(int n){
    makeflow(n);
    T flow=getflow(s,t,t);
    if(flow!=sum) return -1;
    return flow;
  }
  T fesbflow(int ss,int tt,int n){
    add(tt,ss,INF);
    makeflow(n);
    rep(i,head[tt],num[tt])
      if(e[i].to==ss&&e[i].cap==INF) {
        last=i;
        break;
      }
    T flow=getflow(s,t,t);
    if(flow!=sum) return -1;
    return flow;
  }
  T maxflow(int ss,int tt,int n){
    if(fesbflow(ss,tt,n)==-1) return -1;
    return getflow(ss,tt,n+2);
  }
  T minflow(int ss,int tt,int n){
    if(fesbflow(ss,tt,n)==-1) return -1;
    node &x=e[last];
    T ans=INF-x.cap;
    x.cap=e[x.rev].cap=0;
    return ans-getflow(tt,ss,n+2);
  }
};
mxf<int> net;
```

2.20 网络流-SW 全局最小割算法

```
//全局最小割
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#include <queue>
#include <numeric>
typedef long long LL;
const int MAXV = 3010;
const int MAXE = 100010 * 2;
const int INF = 0x3f3f3f3f;
int head[MAXV], val[MAXV], ecnt;
int to[MAXE], next[MAXE], weight[MAXE];
bool vis[MAXV];
int fa[MAXV], link[MAXV];
int n, m;
void init() {
  memset(head + 1, -1, sizeof(int) * n);
  memset(link + 1, -1, sizeof(int) * n);
  for (int i = 1; i <= n; ++i)
    fa[i] = i;
  ecnt = 0;
```

```
}
void add_edge(int u, int v, int w) {
  to[ecnt] = v; weight[ecnt] = w; next[ecnt] = head[u]; head[u] = ecnt++;
  to[ecnt] = u; weight[ecnt] = w; next[ecnt] = head[v]; head[v] = ecnt++;
}
int findset(int u) {
  return u == fa[u] ? u : fa[u] = findset(fa[u]);
}
void merge(int u, int v) {
  int p = u;
  while (~link[p]) p = link[p];
  link[p] = v;
  fa[v] = u;
}
int MinimumCutPhase(int cnt, int &s, int &t) {
  memset(val + 1, 0, sizeof(int) * n);
  memset(vis + 1, 0, sizeof(bool) * n);
  std::priority_queue<std::pair<int, int>> que;
  t = 1;
  while (--cnt) {
    vis[s = t] = true;
    for (int u = s; ~u; u = link[u]) {
      for (int p = head[u]; ~p; p = next[p]) {
        int v = findset(to[p]);
        if (!vis[v])
          que.push(std::make_pair(val[v] += weight[p], v));
      }
    }
    t = 0;
    while (!t) {
      if (que.empty()) return 0;
      auto pa = que.top(); que.pop();
      if (val[pa.second] == pa.first) t = pa.second;
    }
  }
  return val[t];
}
int StoerWagner() {
  int res = INF;
  for (int i = n, s, t; i > 1; --i) {
    res = std::min(res, MinimumCutPhase(i, s, t));
    if (res == 0)
      break;
    merge(s, t);
  }
  return res;
}
int main() {
  while (scanf("%d%d", &n, &m) != EOF) {
    init();
    for (int i = 0, u, v, w; i < m; ++i) {
      scanf("%d%d%d", &u, &v, &w);
      add_edge(u, v, w);
    }
    printf("%d\n", StoerWagner());
  }
}
```

2.21 网络流-最大密度子图

```
const double eps=1e-8;
template<typename T>class mxf{public:
  struct node{int to,next;T cap;}e[maxm<<1];
  int cur[maxn],head[maxn],que[maxn],dis[maxn],nume=1,s,t;
  inline void adde(int a,int b,T c){
    e[++nume]={b,head[a],c};head[a]=nume;
  }
  inline void add(int a,int b,T c){adde(a,b,c);adde(b,a,0);}
  void init(int n=maxn-1){memset(head,0,(n+1)<<2);nume=1;}
  bool bfs(){
    memset(dis,-1,(t+1)<<2);
    dis[t]=0,que[0]=t;
    int tp=0,ed=1;
```

```
    while(tp!=ed){
      int now=que[tp++];if(tp==maxn) tp=0;
      for(int i=head[now];i;i=e[i].next){
        int to=e[i].to;
        if(dis[to]==-1&&e[i^1].cap>0){
          dis[to]=dis[now]+1;
          if(to==s) return true;
          que[ed++]=to;
          if(ed==maxn) ed=0;
        }
      }
    }
    return false;
  }
  T dfs(int now,T flow=1e9){
    if(now==t||flow==0) return flow;
    T use=0;
    for(int &i=head[now];i&&use!=flow;i=e[i].next){
      int to=e[i].to;
      if(dis[to]+1!=dis[now])continue;
      T tmp=dfs(to,min(e[i].cap,flow-use));
      e[i].cap-=tmp,e[i^1].cap+=tmp,use+=tmp;
    }
    if(use==0) dis[now]=-1;
    return use;
  }
  T getflow(int ss,int tt){
    s=ss,t=tt;T ans=0;
    memcpy(cur,head,(t+1)<<2);
    while(bfs()){
      ans+=dfs(s);
      memcpy(head,cur,(t+1)<<2);
    }
    return ans;
  }
};
mxf<double> net;
const int maxn2=500;
int mt[maxn2][maxn2];
double d[maxn2];
int val[maxn2],tag[maxn2];
void init(int n){rep(i,1,n)rep(j,i+1,n) mt[i][j]=mt[j][i]=0;}
void adde(int a,int b,int v){mt[a][b]=mt[b][a]=v;}
const double all=400*2200;//点权和 + 边权和 *2
bool check(double mid){
  int s=n+1,t=n+2;
  net.init(n+3);
  double f=0;
  rep(i,1,n){
    d[i]=0.0;
    rep(j,1,n){
      if(i==j||!mt[i][j])continue;
      d[i]+=mt[i][j];
//如果公式计算出来，边权跟 mid 有关，就要加上相应的 mid
      net.add(i,j,mt[i][j]);
    }
  }
  rep(i,1,n){
    if(tag[i]){
      f+=all+2*mid-d[i];
      net.add(s,i,INF);
    }else {
      net.add(s,i,all);
      net.add(i,t,all+2*mid-d[i]);
//有点权的话，这个 2*mid 还要再乘那个点权
    }
  }
  double x=net.getflow(s,t);
  double ans=(all*n-f-x)*0.5;
  return ans>eps;
}
int main(){IO;
```

```
  cin>>casn;
  cout<<fixed<<setprecision(10);
  rep(kase,1,casn){
    cin>>n;
    init(n);
    rep(i,1,n)cin>>val[i];
    rep(i,1,n)rep(j,i+1,n)
      if(val[j]<val[i]) adde(i,j,1);
//    rep(i,1,n) tag[i]=0;//是否必须用
    double l=0,r=n,mid;
    while(r-l>=eps){
      mid=(l+r)*0.5;
      if(check(mid)) l=mid;
      else r=mid;
    }
    cout<<"Case #"<<kase<<": "<<(l+r)*0.5<<'\n';
  }
  return 0;
}
```

2.22 费用流-Dijkstra

```
//原始对偶算法,dijkstra 寻找增广路, 单路增广
//无优化空间
template<typename T1,typename T2,const int N>class mcf{public:
#define pdi pair<T2,int>
  priority_queue<pdi,vector<pdi>,greater<pdi>>que;
  struct node{int to;T1 cap;T2 cost;int rev;};
  int prev[N+10],pree[N+10],numv;
  T2 dis[N+10],h[N+10];
  vector<node> g[N+10];
  void init(int n){
    numv=n;
    rep(i,0,n) g[i].clear();
  }
  inline void add(int from,int to,T1 cap,T2 cost){
    g[from].push_back({to,cap,cost,(int)g[to].size()});
    g[to].push_back({from,0,-cost,(int)g[from].size()-1});
  }
  pair<T1,T2>getcost(int s,int t,int n){
    numv=n;
    T1 flow=0; T2 cost=0;
    fill_n(h,numv+1,0);
    while(1){
      fill_n(dis,numv+1,INF);
      dis[s]=0;que.push(make_pair(0,s));
      while(!que.empty()){
        auto now=que.top();que.pop();
        if(dis[now.second]<now.first)continue;
        int x=now.second;
        int cnt=0;
        for(auto &i:g[x])
          if(i.cap>0&&dis[i.to]>dis[x]+h[x]-h[i.to]+i.cost){
            dis[i.to]=dis[x]+i.cost+h[x]-h[i.to];
            prev[i.to]=x;
            pree[i.to]=cnt++;
            que.push(make_pair(dis[i.to],i.to));
          }else cnt++;
      }
      if(dis[t]==INF)break;
      rep(i,0,numv) h[i]+=dis[i];
      T1 d=INF;
      for(int now=t;now!=s;now=prev[now])
        d=min(d,g[prev[now]][pree[now]].cap);
      if(d==INF)break;
      flow+=d;cost+=d*h[t];
      for(int now=t;now!=s;now=prev[now]){
        node &e=g[prev[now]][pree[now]];
        e.cap-=d,g[now][e.rev].cap+=d;
      }
    }
    return make_pair(flow,cost);
  }
```

```
};
mcf<int,int,(int)1e4>net;
```

2.23 费用流-SPFA

```
//zkw 费用流, 单路增广
//可将单路增广改为多路增广 + 当前弧优化, 但提升不大
template<typename T1,typename T2>class mcf{public:
  int nume=1,s,t,numv,head[maxn],pre[maxn];
  bool vis[maxn];
  queue<int>q;
  T1 flow[maxn],mflow;
  T2 dis[maxn],mcost;
  struct node{int to,next;T1 cap;T2 cost;}e[maxm<<1];
  void init(int n=maxn-10){
    numv=n;
    fill(head,head+n+2,0);nume=1,mflow=mcost=0;
  }
  inline void add(int from,int to,int cap,T2 cost){
    e[++nume]={to,head[from],cap,cost};head[from]=nume;
    e[++nume]={from,head[to],0,-cost};head[to]=nume;
  }
  bool spfa(){
    fill(dis,dis+2+numv,INF);
    fill(vis,vis+2+numv,false);
    dis[s]=0;flow[s]=INF;q.push(s);
    while (!q.empty()){
      int now=q.front();q.pop();
      vis[now]=false;
      for (int i=head[now];i;i=e[i].next){
        int to=e[i].to;
        T2 cost=e[i].cost;
        if (e[i].cap&&dis[now]+cost<dis[to]){
          dis[to]=dis[now]+cost;
          flow[to]=min(flow[now],e[i].cap);
          pre[to]=i;
          if (!vis[to]){
            vis[to]=true;
            q.push(to);
          }
        }
      }
    }
    return dis[t]<INF;
  }
  void dfs(){
    int x=t;
    while (x!=s){
      int i=pre[x];
      e[i].cap-=flow[t];
      e[i^1].cap+=flow[t];
      x=e[i^1].to;
    }
    mflow+=flow[t];
    mcost+=(T2)flow[t]*dis[t];
  }
  pair<T1,T2> getcost(int ss,int tt){
    s=ss,t=tt;
    while (spfa())dfs();
    return make_pair(mflow,mcost);
  }
};
mcf<int,int>net;
```

2.24 杂项-欧拉路径

```
int vis[maxn];
int cnt;
struct node {
  int to,flag,id,next;
}e[maxm];
int head[maxn],nume,deg[maxn];
inline void _add(int a,int b,int c){
  e[++nume]=(node){b,1,c,head[a]};
```

```
  head[a]=nume;
}
inline void add(int a,int b,int c){
  _add(a,b,c);_add(b,a,-c);
}
vector<int> ans[maxn];
void dfs(int now){
  vis[now]=1;
  for(int i=head[now];i;i=e[i].next){
    if(!e[i].flag) continue;
    e[i].flag=e[i^1].flag=0;
    dfs(e[i].to);
    ans[cnt].push_backe(-e[i].id);
  }
}
void solve(){
  rep(i,1,n){
    if(!vis[i]&&deg[i]&1) {
      cnt++;
      dfs(i);
    }
  }
  rep(i,1,n){
    if(!vis[i]&&deg[i]){
      cnt++;
      dfs(i);
    }
  }
}
```

2.25 杂项-三元环计数

```
vector<pii>g[maxn];
int deg[maxn],a[maxn],b[maxn],cnt[maxn],pos[maxn],v[maxn];
int main() {
  while(cin>>n>>m){
    rep(i,1,n){
      g[i].clear();
      v[i]=deg[i]=pos[i]=0;
    }
    rep(i,1,m){
      cin>>a[i]>>b[i];
      deg[a[i]]++,deg[b[i]]++;
    }
    rep(i,1,m){
      cnt[i]=0;
      if(deg[a[i]]<deg[b[i]])g[a[i]].emplace_back(b[i],i);
      else if(deg[a[i]]>deg[b[i]])g[b[i]].emplace_back(a[i],i);
      else {
        if(a[i]<b[i]) g[a[i]].emplace_back(b[i],i);
        else g[b[i]].emplace_back(a[i],i);
      }
    }
    rep(i,1,m){
      int u=a[i],to=b[i];
      for(auto j:g[u]) pos[j.fi]=j.se,v[j.fi]=i+1;
      for(auto j:g[to]){
        int t=j.fi;
        if(v[t]==i+1){
          cnt[i]++;
          cnt[pos[t]]++;
          cnt[j.se]++;
        }
      }
    }
    ll ans=0;
    rep(i,1,m) ans+=1ll*cnt[i]*(cnt[i]-1)/2;
    cout<<ans<<endl;
  }
}
```

3 Geometry

3.1 几何类-点类与基础

```cpp
//点类与基础
#define db double
#define pb push_back
const db eps = 1e-7;
const db pi = acos(-1);
const db inf = 1e9;
int sign(db x){ if(fabs(x) < eps) return 0; return x > 0 ? 1 : -1; }
int cmp(db k1, db k2){ return sign(k1-k2); }
//k1 在 k2、k3 之间:
bool inmid(db k1, db k2, db k3){ return sign(k2-k1)*sign(k3-k1) <= 0; }
//区间相交判定, 区间 1 在区间 2 前:
bool intersect(db l1,db r1,db l2,db r2){
  if(l1>r1) swap(l1,r1); if(l2>r2) swap(l2,r2);
  return cmp(r1,l2)!=-1 && cmp(r2,l1)!=-1;
}
struct point{
  db x, y;
  point(){}
  point(db k1, db k2){ x = k1, y = k2; }
  //向量加法、点 + 向量 = 点:*/
  point operator + (const point &k1) const { return point(x+k1.x, y+k1.y); }
  //向量减法、点-点 = 向量:*/
  point operator - (const point &k1) const { return point(x-k1.x, y-k1.y); }
  //向量数乘:*/
  point operator * (db k1) const { return (point){x*k1, y*k1}; }
  //向量数除:*/
  point operator / (db k1) const { return (point){x/k1, y/k1}; }
  //比较两个点 (向量) 是否相同:*/
  bool operator == (const point &k1) const {
    return cmp(x,k1.x)==0 && cmp(y,k1.y)==0;
  }
  //逆时针旋转:*/
  point turn(db k1){
    return (point){x*cos(k1)-y*sin(k1), x*sin(k1)+y*cos(k1)};
  }
  //逆时针旋转 90 度:*/
  point turn90(){return (point){-y, x};}
  //比较两个点 (向量) 的大小:
  //x 越小则点越小, 若 x 相等, 则 y 越小点越小. 可以实现按点的坐标排序*/
  bool operator < (const point k1) const{
    int a = cmp(x, k1.x);
    if(a == -1) return 1;
    else if(a == 1) return 0;
    else return cmp(y,k1.y)==-1;
  }
  //向量模长:
  db len(){ return sqrt(x*x+y*y); }
  //向量模长的平方:
  db len2(){ return x*x+y*y; }
  //单位向量:
  point unit(){ return (*this)/(*this).len(); }
  //向量的极角:
  db angle() { return atan2(y, x); }
  //将点放入第一象限:
  //当横坐标为负时, 或横坐标为 0 纵坐标为负时, 将点按原点做对称角度是 [−/2, /2]
  point getdel(){
    if (sign(x)==-1||(sign(x)==0&&sign(y)==-1)) return (*this)*(-1);
    else return (*this);
  }
  //判断点是否在 1 2 象限, 或者在 x 的负半轴上角度是 (0, ]
  bool getp() const {return sign(y)==1 || (sign(y)==0&&sign(x)==-1); }
  void scan(){cin>>x>>y;}
  void print(){cout<<x<<' '<<y<<'\n'; }
};
//判断 k1 在 [k2,k3] 内:
bool inmid(point k1, point k2, point k3){
  return inmid(k1.x,k2.x,k3.x) && inmid(k1.y,k2.y,k3.y);
}
//得到两点中点:
point midpo(point k1, point k2){ return (k1+k2)/2; }
```

```
//两点距离的平方
db dis2(point k1, point k2){
  return (k1.x-k2.x)*(k1.x-k2.x) + (k1.y-k2.y)*(k1.y-k2.y);
}
db dis(point k1, point k2){ return sqrt(dis2(k1, k2)); }
//叉乘:
db cross(point k1, point k2){ return k1.x*k2.y - k1.y*k2.x; }
//点乘:
db dot(point k1, point k2){ return k1.x*k2.x + k1.y*k2.y; }
//向量夹角:
db rad(point k1, point k2){
  return acos(dot(k1,k2)/k1.len()/k2.len());
//return atan2(cross(k1,k2), dot(k1,k2));
}
//极角排序,[- , ]:
bool compareangle (point k1,point k2){
  return k1.getp()<k2.getp() ||
       (k1.getp()==k2.getp() && sign(cross(k1,k2))>0);
}
//k1 k2 k3 逆时针 1 顺时针-1 否则 0:
int clockwise(point k1,point k2,point k3){return sign(cross(k2-k1,k3-k1));}
```

3.2 几何类-直线类与线段类

```
//直线与线段
//直线类
struct line{
//方向为 p[0]->p[1]
  point p[2];
  line(){}
  line(db x1,db y1,db x2,db y2){p[0]=point(x1,y1),p[1]=point(x2,y2);}
  line(point k1,point k2){p[0]=k1; p[1]=k2;}
  point& operator [] (int k){return p[k];}
//点在直线左侧的判定:
//沿着 p0->p1 的左侧为 1, 右侧为 0
  bool include(point k){
    return sign(cross(p[0]-k,p[1]-k))>0;
  }
//方向向量:
  point dir(){return p[1]-p[0];}
//向外 (左) 平移 eps
  line push(){
    point delta=(p[1]-p[0]).turn90().unit()*eps;
    return {p[0]-delta, p[1]-delta};
  }
};
//线段类:
struct segment{
  point p[2];
  segment(){}
  segment(db x1,db y1,db x2,db y2){p[0]=point(x1,y1),p[1]=point(x2,y2);}
  segment(point a, point b){ p[0]= a, p[1] = b; }
  point dir(){return p[1]-p[0];}
  point& operator [] (int k){ return p[k]; }
};
//q 到直线 k1,k2 的投影:
point proj(point q, point k1, point k2){
  point k=k2-k1;
  return k1+k*(dot(q-k1,k)/k.len2());
}
//q 关于直线 k1, k2 的对称点:
point reflect(point q, point k1, point k2){
  return proj(q,k1,k2)*2-q;
}
//点在线段上的判定:
bool checkons(point q,point k1,point k2){
  return inmid(q,k1,k2) && sign(cross(k1-q, k2-k1))==0;
}
//点在直线上的判定:
bool checkonl(point q,point k1,point k2){
  return sign(cross(k1-q, k2-k1))==0;
```

```
}
//点在射线 k1->k2 上的判定:
bool checkonr(point q, point k1, point k2){
  return sign(cross(q-k1, k2-k1)) == 0 && sign(dot(q-k1, k2-k1)) >= 0;
}
//直线平行判定, 可以重合:
bool parallel(line k1,line k2){ return sign(cross(k1.dir(),k2.dir()))==0; }
//直线同向判定:
bool samedir(line k1,line k2){
  return parallel(k1,k2)&&sign(dot(k1.dir(),k2.dir()))==1;
}
//直线的比较, 极角排序, 范围是 [- , ]:
bool operator < (line k1,line k2){
  if (samedir(k1,k2)) return k2.include(k1[0]);
  return compareangle(k1.dir(),k2.dir());
}
//直线相交判定:
//叉积计算面积, 两直线不平行必相交 (除去重合的情况), 平行时, 三角形面积相等:
bool checkll(point k1,point k2,point k3,point k4){
  return cmp(cross(k3-k1,k4-k1),cross(k3-k2,k4-k2))!=0;
}
//直线相交判定:
bool checkll(line k1,line k2){
  return checkll(k1[0],k1[1],k2[0],k2[1]);
}
//直线交点:
point getll(point k1,point k2,point k3,point k4){
  db w1=cross(k1-k3,k4-k3),w2=cross(k4-k3,k2-k3);
  return (k1*w2+k2*w1)/(w1+w2);
}
//直线交点:
point getll(line k1,line k2){
  return getll(k1[0],k1[1],k2[0],k2[1]);
}
//直线与线段相交判定:
//线段的两端点在直线的两侧
bool checkls(point k1, point k2, point k3, point k4){
  return sign(cross(k1-k3, k2-k3)) * sign(cross(k1-k4, k2-k4)) <= 0;
}
// 线段相交判定:
bool checkss(point k1,point k2,point k3,point k4){
  return intersect(k1.x,k2.x,k3.x,k4.x)&&intersect(k1.y,k2.y,k3.y,k4.y) &&
        sign(cross(k3-k1,k4-k1))*sign(cross(k3-k2,k4-k2))<=0 &&
        sign(cross(k1-k3,k2-k3))*sign(cross(k1-k4,k2-k4))<=0;
}
// 线段相交判定:
bool checkss(segment k1, segment k2){
  return checkss(k1[0], k1[1], k2[0], k2[1]);
}
//线段规范相交判定:
//端点相交不算
bool strictcheckss(point k1, point k2, point k3, point k4){
  return sign(cross(k3-k1,k4-k1))*sign(cross(k3-k2,k4-k2))<0 &&
        sign(cross(k1-k3,k2-k3))*sign(cross(k1-k4,k2-k4))<0;
}
// 线段规范相交判定:
bool strictcheckss(segment k1, segment k2){
  return strictcheckss(k1[0], k1[1], k2[0], k2[1]);
}
//点到直线的距离:
db displ(point q, point k1, point k2){
  if(k1 == k2) return dis(q, k1);
  return fabs(cross(k2-k1, q-k1)) / (k2-k1).len();
}
//点到直线的距离:
db displ(point q, line l){
  return displ(q, l[0], l[1]);
}
//点到线段的距离:
db disps(point q,point k1,point k2){
```

```
  point k3 = proj(q,k1,k2);
  if (inmid(k3,k1,k2)) return dis(q, k3);
  else return min(dis(q, k1),dis(q, k2));
}
//点到线段的距离:
db disps(point q, segment k1){
  return disps(q, k1[0], k1[0]);
}
//线段到线段间的距离:
db disss(point k1,point k2,point k3,point k4){
  if (checkss(k1,k2,k3,k4)) return 0;
  else return min(min(disps(k3,k1,k2),disps(k4,k1,k2)),
                  min(disps(k1,k3,k4),disps(k2,k3,k4)));
}
//线段到线段间的距离:
db disss(segment k1, segment k2){
  return disss(k1[0], k1[1], k2[0], k2[1]);
}
```

3.3 几何类-圆类

```
//圆类
struct circle{
  point o; db r;
  circle(){}
  circle(point _o, db _r){ o = _o, r = _r; }
//点在圆内判定:
  bool include(point k){ return cmp(dis(o, k), r) <= 0; }
};
//求直线与圆的交点沿着 k2->k3 方向给出, 相切给出两个:
vector<point> getcl(circle k1,point k2,point k3){
  point k=proj(k1.o,k2,k3);
  db d=k1.r*k1.r-(k-k1.o).len2();
  if (sign(d)==-1) return {};
  point del=(k3-k2).unit()*sqrt(max((db)0.0, d));
  return {k-del, k+del};
}
// 返回两个圆的公切线数量:
int checkposcc(circle k1,circle k2){
  if (cmp(k1.r,k2.r)==-1) swap(k1,k2);
  db d=dis(k1.o,k2.o); int w1=cmp(d,k1.r+k2.r),w2=cmp(d,k1.r-k2.r);
  if (w1>0) return 4; //相离:
  else if (w1==0) return 3; //相切:
  else if (w2>0) return 2; //相交:
  else if (w2==0) return 1; //内切:
  else return 0; //内含:
}
//求两圆交点沿圆 k1 逆时针给出, 相切给出两个:
vector<point> getcc(circle k1,circle k2){
  int pd=checkposcc(k1,k2);
  if(pd==0||pd==4) return {};
  db a=(k2.o-k1.o).len2();
  db cosA=(k1.r*k1.r+a-k2.r*k2.r)/(2*k1.r*sqrt(max(a,(db)0.0)));
  db b=k1.r*cosA;
  db c=sqrt(max((db)0.0,k1.r*k1.r-b*b));
  point k=(k2.o-k1.o).unit(), m=k1.o+k*b, del=k.turn90()*c;
  return {m-del, m+del};
}
//过圆外一点作圆的切线的切点:
//沿圆 k1 逆时针给出
vector<point> tangentcp(circle k1,point k2){
  db a=(k2-k1.o).len(),b=k1.r*k1.r/a,c=sqrt(max((db)0.0,k1.r*k1.r-b*b));
  point k=(k2-k1.o).unit(),m=k1.o+k*b,del=k.turn90()*c;
  return {m-del, m+del};
}
//求两圆的外切线:
vector<line> tangentoutcc(circle k1,circle k2){
  int pd=checkposcc(k1,k2); if (pd==0) return {};
//内含, 返回一条切线
  if (pd==1){
    point p1=getcc(k1,k2)[0]; point p2=p1+((p1-k1.o).turn90()/(p1-k1.o).len());
```

```
    return {(line){p1,p2}};
  }
  if (cmp(k1.r,k2.r)==0){
    point del=(k2.o-k1.o).unit().turn90().getdel();
    return {(line){k1.o-del*k1.r,k2.o-del*k2.r},(line){k1.o+del*k1.r,k2.o+del*k2.r}};
  } else {
    point p=(k2.o*k1.r-k1.o*k2.r)/(k1.r-k2.r);
    vector<point>A=tangentcp(k1,p),B=tangentcp(k2,p);
    vector<line>ans; for (int i=0;i<A.size();i++) ans.push_back((line){A[i],B[i]});
    return ans;
  }
}
//求两圆的内切线:
vector<line> tangentincc(circle k1,circle k2){
  int pd=checkposcc(k1,k2); if (pd<=2) return {};
  if (pd==3){
    point p1=getcc(k1,k2)[0]; point p2=p1+((p1-k1.o).turn90()/(p1-k1.o).len());
    return {(line){p1,p2}};
  }
  point p=(k2.o*k1.r+k1.o*k2.r)/(k1.r+k2.r);
  vector<point>A=tangentcp(k1,p),B=tangentcp(k2,p);
  vector<line>ans; for (int i=0;i<A.size();i++) ans.push_back((line){A[i],B[i]});
  return ans;
}
//求两圆所有切线:
vector<line> tangentcc(circle k1,circle k2){
  int flag=0; if (k1.r<k2.r) swap(k1,k2),flag=1;
  vector<line>A=tangentoutcc(k1,k2),B=tangentincc(k1,k2);
  for (line k:B) A.push_back(k);
  if (flag) for (line &k:A) swap(k[0],k[1]);
  return A;
}
// 圆 k1 与三角形 k2 k3 k1.o 的有向面积交:
db circleinsarea(circle k1,point k2,point k3){
  point k=k1.o; k1.o=k1.o-k; k2=k2-k; k3=k3-k;
  int pd1=k1.include(k2),pd2=k1.include(k3);
  vector<point>A=getcl(k1,k2,k3);
//有一个点在圆内或圆上:
  if (pd1>=0){
//三角形整个落在圆内, 返回三角形的有向面积
    if (pd2>=0) return cross(k2,k3)/2;
//三角形的一个点落在圆内, 一个点落在圆外
    else return k1.r*k1.r*rad(A[1],k3)/2+cross(k2,A[1])/2;
//三角形的一个点落在圆内, 一个点落在圆外
  } else if (pd2>=0){
    return k1.r*k1.r*rad(k2,A[0])/2+cross(A[0],k3)/2;
//否则, 三角形的两个点都落在圆外:
  }else {
    int pd=cmp(k1.r,disps(k1.o,k2,k3));
//返回一个扇形面积:
    if (pd<=0) return k1.r*k1.r*rad(k2,k3)/2;
//返回两个扇形加一个三角形:
    else return cross(A[0],A[1])/2+k1.r*k1.r*(rad(k2,A[0])+rad(A[1],k3))/2;
  }
}
//以 k1k2 为直径的圆:
circle getexcir2(point k1, point k2){
  point c = midpo(k1, k2);
  db r = dis(c, k1);
  return circle(c, r);
}
//三角形的外接圆:
circle getexcir3(point k1, point k2, point k3){
  point c = getll( midpo(k1, k2),
                   (k1-k2).turn90()+midpo(k1, k2),
                   midpo(k1, k3),
                   (k1-k3).turn90()+midpo(k1, k3));
  db r = dis(c, k1);
  return circle(c, r);
}
//最小圆覆盖:
```

```
circle mincircover(vector<point>A){
  random_shuffle(A.begin(), A.end()); int n = A.size();
  circle now = circle(A[0], 0);
  for(int i = 0; i < n; i++)if(!now.include(A[i])){
    now = circle(A[i], 0);
    for(int j = 0; j < i; j++)if(!now.include(A[j])){
      now = getexcir2(A[i], A[j]);
      for(int k = 0; k < j; k++) if(!now.include(A[k]))
        now = getexcir3(A[i], A[j], A[k]);
    }
  }
  return now;
}
```

3.4 几何函数-多边形与半平面

```
//多边形函数
//三角形面积:
db tarea(point a,point b,point c){
  return fabs((b.x-a.x)*(c.y-a.y)-(b.y-a.y)*(c.x-a.x))/2;
}
//多边形面积:
//多边形用 vector<point> 表示, 逆时针
db polyarea(vector<point>A){
  db ans = 0;
  sort(all(A),compareangle);
  for(int i=0;i<A.size();i++) ans += cross(A[i],A[(i+1)%A.size()]);
  return fabs(ans/2);
}
//多边形周长:
db polyperimeter(vector<point>&A){
  db ans = 0;
  for(int i = 0; i < A.size(); i++) ans += dis(A[i], A[(i+1)%A.size()]);
  return ans;
}
//多边形重心:
point polyfocus(vector<point>&A){
  int n = A.size();
  db sumx= 0, sumy = 0, sumarea = 0, area;
  for(int i = 1; i+1 < n; i++){
    area = cross(A[i]-A[0], A[i+1]-A[0])/2.0;
    sumarea += area;
    sumx += (A[0].x+A[i].x+A[i+1].x)*area;
    sumy += (A[0].y+A[i].y+A[i+1].y)*area;
  }
  return point(sumx/sumarea/3.0, sumy/sumarea/3.0);
}
//点与多边形的位置关系:
// 2 内部 1 边界 0 外部
int contain(vector<point>&A, point q){
  int pd=0; A.push_back(A[0]);
  for (int i=1;i<A.size();i++){
    point u=A[i-1], v=A[i];
    if (checkons(q,u,v)) return 1;
    if (cmp(u.y,v.y)>0) swap(u,v);
    if (cmp(u.y,q.y)>=0||cmp(v.y,q.y)<0) continue;
    if (sign(cross(u-v,q-v))<0) pd^=1;
  }
  return pd<<1;
  int wn = 0;
  int n = A.size();
  for(int i = 0; i < n; i++){
    if(checkons(q, A[i], A[(i+1)%n])); return -1;//onside
    int k = sign(cross(A[(i+1)%n]-A[i], q-A[i]));
    int d1 = sign(A[i].y-q.y);
    int d2 = sign(A[(i+1)%n].y-q.y);
    if(k > 0 && d1 <= 0 && d2 > 0) wn++;
    if(k < 0 && d2 <= 0 && d1 > 0) wn--;
  }
  if(wn != 0) return 1;//inside
  return 0;//outside
}
```

```
//逆时针凸包判定:
int checkconvex(vector<point>&A){
  int n=A.size(); A.pb(A[0]); A.pb(A[1]);
  for (int i=0;i<n;i++) if(sign(cross(A[i+1]-A[i],A[i+2]-A[i]))==-1) return 0;
  return 1;
}
//求凸包:
//flag=0 不严格 flag=1 严格
vector<point> convexhull(vector<point>A, int flag=1){
  int n=A.size(); vector<point>ans(n*2);
  sort(A.begin(), A.end());
  int now=-1;
//下凸壳
  for(int i=0;i<n;i++){
    while(now>0 && sign(cross(ans[now]-ans[now-1], A[i]-ans[now-1])) < flag) now--;
    ans[++now]=A[i];
  }
  int pre=now;
//上凸壳
  for(int i=n-2;i>=0;i--){
    while(now>pre && sign(cross(ans[now]-ans[now-1], A[i]-ans[now-1])) < flag) now--;
    ans[++now]=A[i];
  }
//因为 A[0] 会被算两次, 所以舍弃最后一次的 A[0]
  ans.resize(now);
  return ans;
}
//切割凸包:
//保留直线左边的所有点
vector<point> convexcut(vector<point>A,point k1,point k2){
  int n=A.size(); A.push_back(A[0]); vector<point>ans;
  for(int i=0;i<n;i++){
    int w1=clockwise(k1,k2,A[i]), w2=clockwise(k1,k2,A[i+1]);
    if (w1>=0) ans.push_back(A[i]);
    if (w1*w2<0) ans.push_back(getll(k1,k2,A[i],A[i+1]));
  }
  return ans;
}
//凸包最近点对:
//先要按照 x 坐标排序
bool _cmp(point k1,point k2){return k1.y<k2.y;}
db closestpoint(vector<point>&A,int l,int r){
  if (r-l<=5){
//当点数小于等于 5 时, 暴力计算:
    db ans=1e20;
    for (int i=l;i<=r;i++) for (int j=i+1;j<=r;j++) ans=min(ans,dis(A[i],A[j]));
    return ans;
  }
  int mid=l+r>>1; db ans=min(closestpoint(A,l,mid),closestpoint(A,mid+1,r));
  vector<point>B; for (int i=l;i<=r;i++) if (abs(A[i].x-A[mid].x)<=ans) B.push_back(A[i]);
  sort(B.begin(),B.end(),_cmp);
  for (int i=0;i<B.size();i++) for (int j=i+1;j<B.size()&&B[j].y-B[i].y<ans;j++)
    ans=min(ans,dis(B[i],B[j]));
  return ans;
}
//凸包的直径 (最远点对):
//旋转卡壳, 得到的答案为最远距离的平方
db convexdiameter(vector<point>&A){
  int n = A.size();
  int now = 1;
  db res = 0;
  for(int i = 0; i < n; i++){
    while(1){
      db x=cross(A[i]-A[(i+1)%n],A[i]-A[(now+1)%n]);
      db y=cross(A[i]-A[(i+1)%n],A[i]-A[now]);
      if(x<y) break;
      now=(now+1)%n;
    }
    res = max(res, dis2(A[now], A[i]));
  }
  return res;
```

```
}
//点集中的最大三角形:
db maxtriangle(vector<point>&A){
  int m = A.size();
  int a = 1, b = 2;
  db res = 0;
  for(int i = 0; i < m; i++){
    while(cross(A[a]-A[i], A[(b+1)%m]-A[i]) > cross(A[a]-A[i], A[b]-A[i]))
      b = (b + 1) % m;
    res = max(res, cross(A[a]-A[i], A[b]-A[i]) / 2.0);
    while(cross(A[(a+1)%m]-A[i], A[b]-A[i]) > cross(A[a]-A[i], A[b]-A[i]))
      a = (a + 1) % m;
    res = max(res, cross(A[a]-A[i], A[b]-A[i]) / 2.0);
  }
  return res;
}
//凸包间的最小距离:
db mindisbetconvex(vector<point>&A, vector<point>&B){
  int n = A.size(), m = B.size();
  if(n < 3 && m < 3){
    if(n == 1){
      if(m == 1) return dis(A[0], B[0]);
      else return disps(A[0], B[0], B[1]);
    }
    else{
      if(m == 1) return disps(B[0], A[0], A[1]);
      else return disss(A[0], A[1], B[0], B[1]);
    }
  }
  int ai = 0, bi = 0;
  for(int i = 0; i < n; i++) if(A[i].y < A[ai].y){ ai = i; }
  for(int i = 0; i < m; i++) if(B[i].y > A[bi].y){ bi = i; }
  db ans = 1e18;
  for(int i = 0; i < n; i++){
    db ck;
    while(ck = sign(cross(B[(bi+1)%m]-B[bi], A[(ai+1)%n]-A[ai])) < 0) bi = (bi+1)%m;
    if(ck == 0) ans = min(ans, disss(A[(ai+1)%n], A[ai], B[(bi+1)%m], B[bi]));
    else ans = min(ans, disps(B[bi], A[(ai+1)%n], A[ai]));
    ai = (ai+1)%n;
  }
  return ans;
}
//最小正方形覆盖:
db minsquarecover(vector<point>&A, db rad){
  db minx = inf, maxx = -inf, miny = inf, maxy = -inf;
  for(int i = 0; i < A.size(); i++){
    point p = A[i].turn(rad);
    minx = min(minx, p.x);
    miny = min(miny, p.y);
    maxx = max(maxx, p.x);
    maxy = max(maxy, p.y);
  }
  return max(maxx-minx, maxy-miny);
}
//三分--最小正方形覆盖:
db t_divide(vector<point>&A, db l, db r){
  db m,rm,eps=1e-8;
  while(r-l>=eps){
    m=l+(r-l)/3;
    rm=r-(r-l)/3;
    if(minsquarecover(A,m)>minsquarecover(A,rm)) l=m;
    else r=rm;
  }
  return minsquarecover(A, (m+rm)/2);
}
//求半平面交:
//半平面是逆时针方向, 输出按照逆时针
vector<point> gethalf(vector<line> L){
  int n = L.size();
  sort(L.begin(), L.end());
  int first = 0, last = 0;
//双端队列指针
```

```
  line *q = new line[n];
//双端队列
  point *p = new point[n];
//p[i] 为 l[i] 和 l[i+1] 的交点
  q[last] = L[0];
//初始化为一个半平面
  for(int i = 0; i < n; i++){
    while(first < last && !L[i].include(p[last-1])) last--;
    while(first < last && !L[i].include(p[first])) first++;
    q[++last] = L[i];
    if(samedir(q[last], q[last-1])) last--;
    if(first < last) p[last-1] = getll(q[last], q[last-1]);
  }
  while(first < last && !q[first].include(p[last-1])) last--;
  vector<point>ans;
  if(last - first <= 1) return ans;
  p[last] = getll(q[last], q[first]);
  for(int i = first; i <= last; i++) ans.pb(p[i]);
  return ans;
}
int checkpos(line k1,line k2,line k3){return k3.include(getll(k1,k2));}
//求半平面交:
//半平面是逆时针方向, 输出按照逆时针
vector<line> gethl(vector<line> L){
  sort(L.begin(),L.end()); deque<line> q;
  for (int i=0;i<(int)L.size();i++){
    if (i&&samedir(L[i],L[i-1])) continue;
    while (q.size()>1&&!checkpos(q[q.size()-2],q[q.size()-1],L[i])) q.pop_back();
    while (q.size()>1&&!checkpos(q[1],q[0],L[i])) q.pop_front();
    q.push_back(L[i]);
  }
  while (q.size()>2&&!checkpos(q[q.size()-2],q[q.size()-1],q[0])) q.pop_back();
  while (q.size()>2&&!checkpos(q[1],q[0],q[q.size()-1])) q.pop_front();
  vector<line>ans; for (int i=0;i<q.size();i++) ans.push_back(q[i]);
  return ans;
}
```

3.5 几何函数-圆的反演

```
/*
一、反演的概念
设在平面内给定一点 O 和常数 k(k 不等于零), 对于平面内任意一点 A,
确定 A', 使 A' 为直线 OA 上一点, 并且有向线段 OA' 与 OA 满足 OA'·OA = k, 我们称这种变换是以 O 为反演中心,
以 k 为反演幂的反演变换, 简称反演. 称 A' 为 A 关于 O(r) 的互为反演点.
二、作已知点的反演点的方法
给出反演极 O 和反演幂 k>0, 作点 A 的反演点 A'.
令 k = r^2, 作出反演基圆 O(r),
1) 若点 A 在 O(r) 外, 则过点 A 作圆的切线 (两条), 两个切点相连与 OA 连线交点就是点 A'.
2) 若点 A 在 O(r) 内, 则把上述过程逆过来:
 连结 OA, 过点 A 作直线垂直于 OA, 直线与 O(r) 的交点处的切线的交点就是点 A'.
3) 若点 A 在 O(r) 上, 反演点 A' 就是点 A 自身.
4)O 没有反演点
三、圆的反演变换
圆在不同情形下的反演成像:
1. 当圆不经过反演中心, 它的反演图形仍旧是个不过反演中心的圆, 并且反演中心为这两个互为反形的圆的位似中心:
2. 当圆与反演圆相交, 交点是保持不变的;
3. 当圆在反演圆的外面的时候, 反演成像位于圆的内部; 反之, 当圆位于反演圆的内部, 反演成像位于圆的外部.
4. 当圆经过反演中心, 它的反演图形是一条直线.
   反之, 任意一条不过反演中心的直线, 其反演成像是一个经过反演中心的圆.
5. 相切两圆反向任相切, 且切点不变, 若切点是反演中心,
   则其反象是两条平行直线; 两圆相切, 若反演中心在某圆上, 则为反形为相切的直线与圆;
*/
//c1 关于 c0 的反演圆:
circle getinvertcir(circle c1, circle c0){
  circle c2;
  db x0 = c0.o.x, y0 = c0.o.y, r0 = c0.r,
  db x1 = c1.o.x, y1 = c1.o.y, r1 = c1.r;
  db d01 = dis(c0.o, c1.o);
  c2.r = 0.5*((1/(d01-r1))-(1/(d01+r1)))*r0*r0;
  db d02 = r0*r0/(d01+r1)+c2.r;
```

```
  //db _d02 = r0*r0/(d01-r1)-c2.r;
  c2.o.x = x0 + d02/d01*(x1-x0);
  c2.o.y = y0 + d02/d01*(y1-y0);
  return c2;
}
//直线 k1 关于 c0 的反演圆:
circle getinvertcir(line k1, circle c0){
  point a = proj(c0.o, k1[0], k1[1]);
  db oa = dis(c0.o, a);
  db ob = c0.r*c0.r/oa;
  point v = a-c0.o; v = v/v.len();
  point b = c0.o+v*ob;
  circle res;
  res.o = midpo(c0.o, b);
  res.r = ob/2;
  return res;
}
//c1 关于 c0 的反演直线:
line getinvertline(circle c1, circle c0){
  point v = c1.o - c0.o;
  v = v / v.len();
  db d = c0.r*c0.r / (2*c1.r);
  point k1 = v * d + c0.o;
  v = v.turn90();
  point k2 = k1 + v;
  return line(k1, k2);
}
//p 关于 c 的反演点:
point getinvertpoint(point p, circle c){
  point v = (p-c.o).unit();
  db len = c.r*c.r/dis(c.o, p);
  return c.o+v*len;
}
```

3.6 几何函数-圆上整点

```
struct point{
  ll x,y;//两圆上整数点对
  point(ll _x=0,ll _y=0){x=_x,y=_y;}
  void print2(){printf("%lld %lld\n",x,y);}
  void print1(){printf("%lld %lld ",x,y);}
  bool operator==(const point&other)const{
    return x==other.x&&y==other.y;
  }
  bool operator<(const point&other)const{
    if(x==other.x) return y<other.y;
    return x<other.x;
  }
};
ll dis(point a, point b){
  return (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y);
}
bool check(ll n){ll x=sqrt(n);return x*x==n;}
void solve(ll rr,ll r,ll d,vector<point>&A){
  for (ll i=1;i*i<=rr;++i){
    ll t=rr-i*i;
    if(!check(t))continue;
    ll j=sqrt(t);
    if(i>=j)break;
    if(__gcd(i*i,t)==1){
      ll x=i*j*d;
      ll y=sqrt(r*r-x*x);
      A.pb(point(x,y));
      A.pb(point(-x,y));
      A.pb(point(x,-y));
      A.pb(point(-x,-y));
    }
  }
}
void deal(vector<point>&A,ll r){
  A.pb(point(0,r));
  A.pb(point(0,-r));
  A.pb(point(-r,0));
```

```
  A.pb(point(r,0));
  r<<=1;
  for (ll d=1;d*d<=r;d++){
    if(r%d!=0) continue;
    solve(r/d,r/2,d,A);
    if(d*d==r) break;
    solve(d,r/2,r/d,A);
  }
}
vector<point>veca,vecb;
struct node{
  point A,B;
  node(point _A,point_B){A=_A;B=_B;}
  bool operator<(const node&other)const{
    if(A==other.A)return B<other.B;
    else return A<other.A;
  }
};
vector<node>ans;
int main(){
  int T;ll a,b,c;
  scanf("%d",&T);
  while(T--){
    veca.clear();vecb.clear();ans.clear();
    scanf("%lld%lld%lld",&a,&b,&c);
    deal(veca,a);deal(vecb,b);
    for(int i=0;i<veca.size();i++)
      for(int j=0;j<vecb.size();j++)
        if(dis(veca[i],vecb[j])==c*c)
          ans.pb(node(veca[i],vecb[j]));
    sort(ans.begin(),ans.end());
    printf("%d\n",ans.size());
    for(int i=0;i<ans.size();++i){
      ans[i].A.print1();ans[i].B.print2();
    }
  }
}
```

3.7 几何函数-多边形面积并

```
/*
 * 多边形的交，多边形的边一定是要按逆时针方向给出
 * 还要判断是凸包还是凹包，调用相应的函数
 * 面积并，只要和面积减去交即可
 */
#include <bits/stdc++.h>
using namespace std;
const int maxn = 300;
const double eps = 1e-8;
int dcmp(double x){
    if(x > eps) return 1;
    return x < -eps ? -1 : 0;
}
struct Point{
    double x, y;
};
double cross(Point a,Point b,Point c) ///叉积
{
    return (a.x-c.x)*(b.y-c.y)-(b.x-c.x)*(a.y-c.y);
}
Point intersection(Point a,Point b,Point c,Point d){
    Point p = a;
    double t =((a.x-c.x)*(c.y-d.y)-(a.y-c.y)*(c.x-d.x))/((a.x-b.x)*(c.y-d.y)-(a.y-b.y)*(c.x-d.x));
    p.x +=(b.x-a.x)*t;
    p.y +=(b.y-a.y)*t;
    return p;
}
//计算多边形面积
double PolygonArea(Point p[], int n){
    if(n < 3) return 0.0;
    double s = p[0].y * (p[n - 1].x - p[1].x);
    p[n] = p[0];
    for(int i = 1; i < n; ++ i)
```

```
        s += p[i].y * (p[i - 1].x - p[i + 1].x);
    return fabs(s * 0.5);
}
double CPIA(Point a[], Point b[], int na, int nb)//ConvexPolygonIntersectArea
{
    Point p[20], tmp[20];
    int tn, sflag, eflag;
    a[na] = a[0], b[nb] = b[0];
    memcpy(p,b,sizeof(Point)*(nb + 1));
    for(int i = 0; i < na && nb > 2; i++){
        sflag = dcmp(cross(a[i + 1], p[0],a[i]));
        for(int j = tn = 0; j < nb; j++, sflag = eflag){
            if(sflag>=0) tmp[tn++] = p[j];
            eflag = dcmp(cross(a[i + 1], p[j + 1],a[i]));
            if((sflag ^ eflag) == -2)
                tmp[tn++] = intersection(a[i], a[i + 1], p[j], p[j + 1]); //求交点
        }
        memcpy(p, tmp, sizeof(Point) * tn);
        nb = tn, p[nb] = p[0];
    }
    if(nb < 3) return 0.0;
    return PolygonArea(p, nb);
}
double SPIA(Point a[], Point b[], int na, int nb)//SimplePolygonIntersectArea 调用此函数
{
    int i, j;
    Point t1[4], t2[4];
    double res = 0, num1, num2;
    a[na] = t1[0] = a[0], b[nb] = t2[0] = b[0];
    for(i = 2; i < na; i++){
        t1[1] = a[i-1], t1[2] = a[i];
        num1 = dcmp(cross(t1[1], t1[2],t1[0]));
        if(num1 < 0) swap(t1[1], t1[2]);
        for(j = 2; j < nb; j++)
        {
            t2[1] = b[j - 1], t2[2] = b[j];
            num2 = dcmp(cross(t2[1], t2[2],t2[0]));
            if(num2 < 0) swap(t2[1], t2[2]);
            res += CPIA(t1, t2, 3, 3) * num1 * num2;
        }
    }
    return res;
}
Point p1[maxn], p2[maxn];
int n1, n2;
int main(){
    freopen("in.txt", "r", stdin);
    while(cin>>n1>>n2){
        for(int i = 0; i < n1; i++) scanf("%lf%lf", &p1[i].x, &p1[i].y);
        for(int i = 0; i < n2; i++) scanf("%lf%lf", &p2[i].x, &p2[i].y);
        double Area = SPIA(p1, p2, n1, n2);
        cout << Area << endl;
    }
    return 0;
}
```

3.8 几何函数-辛普森积分

```
/*自适应辛普森积分*/
db f(db x){
    /*积分表达式，或积分微元*/
}
db simpson(db l,db r){return (r-l)*(f(l)+f(r)+4*f((l+r)/2))/6;}/*辛普森积分公式*/
db asr(db l,db r,db eps,db s){
    double mid=(l+r)/2, ls=simpson(l,mid),rs=simpson(mid,r);
    if(fabs(ls+rs-s) <= eps*15){
        return ls+rs+(ls+rs-s)/15.0;
    }
    return asr(l, mid, eps/2, ls) + ars(mid, r, eps/2, rs);
}
int main(){
    db ans = asr();
}
```

3.9 几何函数-最小圆覆盖

```cpp
#include <algorithm>
#include <iostream>
#include <cstring>
#include <cstdio>
#include <cmath>

using namespace std;

struct vec
{
    double x, y;
    vec (const double& x0 = 0, const double& y0 = 0) : x(x0), y(y0) {}
    vec operator + (const vec& t) const {return vec(x+t.x, y+t.y);}
    vec operator - (const vec& t) const {return vec(x-t.x, y-t.y);}
    vec operator * (const double& t) const {return vec(x*t, y*t);}
    vec operator / (const double& t) const {return vec(x/t, y/t);}
    const double len2 () const {return x*x + y*y;}
    const double len () const {return sqrt(len2());}
    vec norm() const {return *this/len();}
    vec rotate_90_c () {return vec(y, -x);}
};

double dot(const vec& a, const vec& b) {return a.x*b.x + a.y*b.y;}
double crs(const vec& a, const vec& b) {return a.x*b.y - a.y*b.x;}

vec lin_lin_int(const vec& p0, const vec& v0, const vec& p1, const vec& v1)
{
    double t = crs(p1-p0, v1) / crs(v0, v1);
    return p0 + v0 * t;
}

vec circle(const vec& a, const vec& b, const vec& c)
{
    return lin_lin_int((a+b)/2, (b-a).rotate_90_c(), (a+c)/2, (c-a).rotate_90_c());
}

int n;
vec pot[100005];

int main()
{
    scanf("%d", &n);
    for(int i=1; i<=n; i++) scanf("%lf%lf", &pot[i].x, &pot[i].y);
    random_shuffle(pot+1, pot+n+1);
    vec o;
    double r2 = 0;
    for(int i=1; i<=n; i++)
    {
        if((pot[i]-o).len2() > r2)
        {
            o = pot[i], r2 = 0;
            for(int j=1; j<i; j++)
            {
                if((pot[j]-o).len2() > r2)
                {
                    o = (pot[i]+pot[j])/2, r2 = (pot[j]-o).len2();
                    for(int k=1; k<j; k++)
                    {
                        if((pot[k]-o).len2() > r2)
                        {
                            o = circle(pot[i], pot[j], pot[k]), r2 = (pot[k]-o).len2();
                        }
                    }
                }
            }
        }
    }
    printf("%.10lf\n%.10lf %.10lf\n", sqrt(r2), o.x, o.y);
    return 0;
}
```

3.10 几何函数-最小球覆盖

```
double cx=0,cy=0,cz=0;
double x[maxn],y[maxn],z[maxn];
double dis(int now){
  return sqrt((x[now]-cx)*(x[now]-cx)+
       (y[now]-cy)*(y[now]-cy)+
       (z[now]-cz)*(z[now]-cz));
}
int main() {
  cin>>n;
  rep(i,1,n) cin>>x[i]>>y[i]>>z[i];
  int pos=1;
  double dmax=1e4,ans=1e18;
  while(dmax>1e-7){
    rep(i,1,n) if(dis(i)>dis(pos)) pos=i;
    double d=dis(pos);
    ans=min(ans,d);
    cx+=(x[pos]-cx)/d*dmax;
    cy+=(y[pos]-cy)/d*dmax;
    cz+=(z[pos]-cz)/d*dmax;
    dmax*=0.98;
  }
  printf("%.6f\n",ans);
  return 0;
}
```

4 Game

4.1 SG 函数

```
int sg[maxn][maxn];
int getmex(bool *vis,int n){
  int t=0;
  while(vis[t]) ++t;
  return t;
}
int getsg(int n,int m){
  if(~sg[n][m]) return sg[n][m];
  if(n<=m) return sg[n][m]=1;
  bool vis[n+1]={0};
  rep(i,1,m)vis[getsg(n-i,m)]=1;
  return sg[n][m]=getmex(vis,n);
}
int main() {
  memset(sg,-1,sizeof sg);
  cin>>casn;
  while(casn--){
    cin>>n>>m;
    if(getsg(n,m)) cout<<"first\n";
    else cout<<"second\n";
  }
  return 0;
}
```

4.2 威佐夫游戏

```
int main(){
  double k=(1+sqrt(5.0))/2;
  while(scanf("%d%d",&n,&m)!=EOF) {
    if (n>m) swap(n,m);
    int t=m-n;
    if (n==(int)((double)t*k)) printf("0\n");
    else printf("1\n");
  }
}
```

4.3 K 倍取石子博弈

```

#include <iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int maxn=2000000;
int a[maxn],b[maxn];
```

```
int main()
{
    int t,n,k;
    cin>>t;
    for(int cas=1;cas<=t;cas++)
    {
        cin>>n>>k; //n 个石子每次拿前一次的最多 k 倍
        printf("Case %d: ",cas);
        if(n<=k+1)
        {
            printf("lose\n");
            continue;
        }
        a[0]=b[0]=1;
        int i=0,j=0;
        while(a[i]<n)
        {
            i++;
            a[i]=b[i-1]+1;
            while(a[j+1]*k<a[i]) j++;
            if(a[j]*k<a[i]) b[i]=b[j]+a[i];
            else b[i]=a[i];
        }

        if(a[i]==n)printf("lose\n");
        else
        {
            int ans=0;
            while(n)
            {
                if(n>=a[i])
                {
                    n-=a[i];
                    ans=a[i];
                }
                i--;
            }
            cout<<ans<<endl;
        }
    }
}
```

5 Math

5.1 数论-基础函数

```
ll p=1e9+7;
ll gcd(ll a,ll b) {return b?gcd(b,a%b):a;}
ll lcm(ll a,ll b) {return a*gcd(a,b)/b;}
ll exgcd(ll a,ll b,ll &x,ll &y) {
  if(b==0) return (x=1,y=0,a);
  if(a==0) return (x=0,y=1,b);
  ll r=exgcd(b,a%b,y,x);
  y-=(a/b)*x;
  return r;
}
ll lcm_mod(ll a,ll b,ll c=p) {
  return (a/gcd(a,b)*b)%c;
}
ll pow_mod(ll a,ll b,ll c=p,ll ans=1) {
  while(b) {
    if(b&1) ans=(a*ans)%c;
    a=(a*a)%c,b>>=1;
  }
  return ans;
}
//long double 1e9 以下表现良好, 不会出现误差
ll mul_mod_2(ll a,ll b,ll m){
 ll c=a*b-(ll)((long double)a*b/m+0.5)*m;
 return c<0?c+m:c;
}
//不丢失精度的快速乘
ll mul_mod(ll a,ll b,ll c){return (__int128)a*b%c;}
ll pow_mul_mod(ll a,ll b,ll c=p,ll ans=1){
```

```
  while(b){
    if(b&1)res=mul_mod(res,a,c);
    a=mul_mod(a,a,c),b>>=1;
  }
  return res;
}
ll inv_gcd(ll a,ll c=p){
  a%=c;
  if(a<0)a+=c;
  ll b=c,u=0,v=1;
  while(a) {
    ll t=b/a;b-=t*a;
    swap(a,b);
    u-=t*v;
    swap(u,v);
  }
  if(u<0)u+=c;
  return u;
}
```

5.2 数论-欧拉函数线性筛

```
// #x that x<=n && gcd(x,n)==1
int euler_phi(int n) {
  int m = (int)sqrt(n+0.5);
  int ans = n;
  for (int i = 2; i <= m; ++ i) if (n % i == 0) {
    ans = ans / i * (i-1);
    while (n%i == 0) n /= i;
  }
  if (n > 1) ans = ans / n * (n-1);
  return ans;
}
int phi[maxn];
void phi_table(int n) {
  for (int i = 2; i <= n; ++ i) phi[i] = 0;
  phi[1] = 1;
  for (int i = 2; i <= n; ++ i) {
    if (!phi[i]) {
      for (int j = i; j <= n; j += i) {
        if (!phi[j]) phi[j] = j;
        phi[j] = phi[j] / i * (i-1);
      }
    }
    phi[i] += phi[i-1];
  }
}
```

5.3 数论-莫比乌斯函数线性筛

```
int mu[maxn],prime[maxn],sum[maxn],nump;
bool isp[maxn];
void getmu(){
  mu[1]=1,nump=0;
  int n=maxn-10;
  rep(i,2,n){
    if(!isp[i]) prime[++nump]=i,mu[i]=-1;
    for(int j=1;j<=nump&&prime[j]*i<=n;j++){
      isp[i*prime[j]]=1;
      if(i%prime[j]==0) mu[i*prime[j]]=0,j=nump+10;
      else mu[i*prime[j]]=-mu[i];
    }
  }
  rep(i,1,n){
    sum[i]=sum[i-1]+mu[i];
  }
}
```

5.4 数论-Miller Rabin 素数判定

```
//两种米勒罗宾素数筛实现
//第一种速度稍快，但实现麻烦，使用前需要先初始化
//单次复杂度约为 1-maxpe 中的素数个数 *logn,maxpe 不要小于 50
const int maxpe=100;
auto randint=bind(uniform_int_distribution<int>(1,1e9),mt19937(rand()));
```

```
ll mul_mod(ll a,ll b,ll c){return (__int128)a*(__int128)b%(__int128)c;}
int mul_mod(int a,int b,int c){return (ll)a*(ll)b%c;}
template<typename T> int pow_mod(int a,int b,int c){
  int res=1;
  while(b){
    if(b&1)res=mul_mod(res,a,c);
    a=mul_mod(a,a,c);
    b>>=1;
  }
  return res;
}
template<typename T> class miller_rabin{public:
  T prime[maxpe],cntp;
  void init(int n=maxpe-5){
    cntp=0;
    bool vis[n+1];
    rep(i,2,n)vis[i]=1;
    rep(j,2,n)if(vis[j]==1)
      for(int m=2;j*m<=n;++m)vis[j*m]=0;
    rep(i,2,n)if(vis[i]==1) prime[cntp++]=i;
  }
  bool _test(T n,T a,T d) {
    if(n==2||n==a) return true;
    if((n&1)==0) return false;
    while(!(d&1))d>>=1;
    T t=pow_mod(a,d,n);
    while(d!=n-1&&t!=1&&t!=n-1){
      t=mul_mod(t,t,n);
      d<<=1;
    }
    return (t==n-1||(d&1)==1);
  }
  bool test(T n) {
    if(n<2||n%2==0) return false;
    rep(i,0,cntp-1) if(!_test(n,prime[i],n-1)) return false;
    return true;
  }
};
miller_rabin<int> miller;
//第二种实现
//速度稍慢，复杂度约为 logn*time,time 不要低于 10
const int test_time=20;
template<typename T> bool miller_rabin(T n) {
  if(n<3)return n==2;
  T a=n-1,b=0;
  while(a%2==0) a/=2,++b;
  for(int i=1,j; i<=test_time;++i){
    T x=randint()%(n-2)+2,v=pow_mod(x,a,n);
    if(v==1||v==n-1) continue;
    for(j=0;j<b;++j){
      v=mul_mod(v,v,n);
      if(v==n-1) break;
    }
    if(j>=b)return 0;
  }
  return 1;
}
```

5.5 数论-Pollard Rho 因子分解

```
//得到 n 的一个随机因子，包括自身和 1
//复杂度 O(n^{1/4})
//要保证 randint 的随机范围大于等于测试数字
ll mul_mod(ll a,ll b,ll c){return (__int128)a*b%c;}
int mul_mod(int a,int b,int c){return (ll)a*(ll)b%c;}
auto randint=bind(uniform_int_distribution<ll>(1e9,1e18),mt19937(rand()));
template<typename T> T pollard_rho(T n,T c=randint()) {
  T i=1,k=2,x=randint()%(n-1)+1,y=x,d;
  while(1){
    i++;
    x=(mul_mod(x,x,n)+c)%n;
    d=__gcd(n,y-x);
    if(d>1&&d<n)return d;
```

```
    if(y==x)return n;
    if(i==k){
      k<<=1;
      y=x;
    }
  }
}
//分解因子,map 中即为素因子从小到大,first 为因子,second 为次幂
//分解 1000 个 1e18 的数字约为 600ms,10000 个 1e9 的数字约为 200ms
map<ll,int> factor;
template<typename T>void get_factor(T n,T c=randint()) {
  if(n==1) return;
  if(miller_rabin(n)) {
    factor[n]++;
    return;
  }
  T p=n;
  while(p>=n) p=pollard_rho(p,c--);
  get_factor(p,c);
  get_factor(n/p,c);
}

//新版实现常数降低
template<typename T>class Pollard_rho{public:
  ull s1,s2,s3,s4;
  Pollard_rho(){
    srand(time(0)+rand());
    s1=(ull)rand()*rand()*rand()*rand();
    s2=(ull)rand()*rand()*rand()*rand();
    s3=(ull)rand()*rand()*rand()*rand();
    s4=(ull)rand()*rand()*rand()*rand();
  }
  ull getinteger() {
    ull t=s1^(s1<<11);
    s1=s2;s2=s3;s3=s4;
    return s4=s4^(s4>>19)^t^(t>>8);
  }
  T randinteger(T l,T r){return T(getinteger()%((ull)r-l+1)+l);}
  T get_factor(T n,T c) {
  T i=1,k=2,x=randinteger(1,1e18)%(n-1)+1,y=x,d;
  while(1){
    i++;
    x=(mul_mod(x,x,n)+c)%n;
    d=__gcd(n,y-x);
    if(d>1&&d<n)return d;
    if(y==x)return n;
    if(i==k){
      k<<=1;
      y=x;
    }
  }
}
};
Pollard_rho<ll> rho;
```

5.6 多项式-拉格朗日插值

```
class polysum {public:
  ll a[maxn],f[maxn],g[maxn],p[maxn],p1[maxn],p2[maxn],b[maxn],h[maxn][2],C[maxn];
  ll calcn(int d,ll *a,ll n) {//len=d get(an)
    if (n<=d) return a[n];
    p1[0]=p2[0]=1;
    rep(i,0,d) {
      ll t=(n-i+mod)%mod;
      p1[i+1]=p1[i]*t%mod;
    }
    rep(i,0,d) {
      ll t=(n-d+i+mod)%mod;
      p2[i+1]=p2[i]*t%mod;
    }
    ll ans=0;
    rep(i,0,d) {
      ll t=g[i]*g[d-i]%mod*p1[i]%mod*p2[d-i]%mod*a[i]%mod;
```

```
    if ((d-i)&1) ans=(ans-t+mod)%mod;
    else ans=(ans+t)%mod;
  }
  return ans;
}
void init(int maxm) {//init
  f[0]=f[1]=g[0]=g[1]=1;
  rep(i,2,maxm+4) f[i]=f[i-1]*i%mod;
  g[maxm+4]=pow_mod(f[maxm+4],mod-2);
  per(i,1,maxm+3) g[i]=g[i+1]*(i+1)%mod;
}
ll polysum(ll n,ll *a,ll m){//a[i]会被修改
  // 初始化预处理阶乘和逆元 (取模乘法)a[0].. a[m] \sum_{i=0}^{n-1} a[i]
  // len=m,psum_n
  a[m+1]=calcn(m,a,m+1);
  rep(i,1,m+1) a[i]=(a[i-1]+a[i])%mod;
  return calcn(m+1,a,n-1);
}
ll qpolysum(ll R,ll n,ll *a,ll m) { // a[0].. a[m] \sum_{i=0}^{n-1} a[i]*R^i
  if (R==1) return polysum(n,a,m);
  a[m+1]=calcn(m,a,m+1);
  ll r=pow_mod(R,mod-2),p3=0,p4=0,c,ans;
  h[0][0]=0;
  h[0][1]=1;
  rep(i,1,m+1) {
    h[i][0]=(h[i-1][0]+a[i-1])*r%mod;
    h[i][1]=h[i-1][1]*r%mod;
  }
  rep(i,0,m+1) {
    ll t=g[i]*g[m+1-i]%mod;
    if (i&1) p3=((p3-h[i][0]*t)%mod+mod)%mod,p4=((p4-h[i][1]*t)%mod+mod)%mod;
    else p3=(p3+h[i][0]*t)%mod,p4=(p4+h[i][1]*t)%mod;
  }
  c=pow_mod(p4,mod-2)*(mod-p3)%mod;
  rep(i,0,m+1) h[i][0]=(h[i][0]+h[i][1]*c)%mod;
  rep(i,0,m+1) C[i]=h[i][0];
  ans=(calcn(m,C,n)*pow_mod(R,n)-c)%mod;
  if (ans<0) ans+=mod;
  return ans;
}
}
```

5.7 多项式-快速傅立叶变换

```
const double pi=acos(-1.0);
struct cp{double x,y;};
cp operator*(cp a,cp b){return {a.x*b.x-a.y*b.y,a.x*b.y+a.y*b.x};}
cp operator+(cp a,cp b){return {a.x+b.x,a.y+b.y};}
cp operator-(cp a,cp b){return {a.x-b.x,a.y-b.y};}
#define carr vector<cp>
const int maxl=6e4+10;//卷积单个数组的最大长度
class fourier{public:
  int rev[maxl<<2],len,pw;
  vector<cp> wt;
  void init_0(int ml=maxl){
    for(int mid=1;mid<2*ml;mid<<=1){
      wt.resize(mid*2+1);
      cp wn={cos(pi/mid),sin(pi/mid)};
      wt[mid]=(cp){1,0};
      rep(j,1,mid-1) wt[mid+j]=wt[mid+j-1]*wn;
    }
  }
  void init(int n){
    len=1,pw=0;
    while(len<=n) len<<=1,++pw;--pw;
    rep(i,0,len-1) rev[i]=rev[i>>1]>>1|(i&1)<<pw;
  }
  void transform(carr &a,int flag){
    if(a.size()!=len) a.resize(len,(cp){0,0});
    rep(i,0,len-1) if(i<rev[i]) swap(a[i],a[rev[i]]);
    for(int mid=1;mid<len;mid<<=1){
      for(int r=mid<<1,j=0;j<len;j+=r){
        for(int k=0;k<mid;++k){
```

```
        cp wn=wt[mid+k];
        if(flag==-1) wn.y=-wn.y;
        cp y=wn*a[mid+j+k];
        a[j+k+mid]=a[j+k]-y,a[j+k]=a[j+k]+y;
      }
    }
  }
  if(flag==-1) rep(i,0,len-1) a[i].x/=len;
 }//会破坏掉 a,b 数组, 视情况可以去掉引用
 carr mul(carr &a,carr &b){
  int la=a.size(),lb=b.size();
  if((ll)la*lb<=1000){
   carr c(la+lb,(cp){0,0});
   rep(i,0,la-1) rep(j,0,lb-1) c[i+j]=c[i+j]+a[i]*b[j];
   return c;
  }
  init(la+lb);
  carr c(len,(cp){0,0});
  transform(a,1);transform(b,1);
  rep(i,0,len-1)c[i]=a[i]*b[i];
  transform(c,-1);
  return c;
 }
}fft;
```

5.8 多项式-快速傅立叶变换-数组实现

```
const double pi=acos(-1.0);
struct cp{double x,y;};
cp operator*(cp a,cp b){return {a.x*b.x-a.y*b.y,a.x*b.y+a.y*b.x};}
cp operator+(cp a,cp b){return {a.x+b.x,a.y+b.y};}
cp operator-(cp a,cp b){return {a.x-b.x,a.y-b.y};}
const int maxl=2e6+10;
class fourier{public:
  int rev[maxl<<2],len,pw;
  void init(int n){
    len=1,pw=0;
    while(len<=n) len<<=1,++pw;--pw;
    rep(i,0,len-1) rev[i]=rev[i>>1]>>1|(i&1)<<pw;
  }
  cp c1[maxl<<2],c2[maxl<<2];
  vector<cp> wt;
  void init_0(){
    for(int mid=1;mid<2*maxl;mid<<=1){
      wt.resize(mid*2+1);
      cp wn{cos(pi/mid),sin(pi/mid)};
      wt[mid]=(cp){1,0};
      rep(j,1,mid-1) wt[mid+j]=wt[mid+j-1]*wn;
    }
  }
  void transform(cp*a,int flag){
    rep(i,0,len-1) if(i<rev[i]) swap(a[i],a[rev[i]]);
    for(int mid=1;mid<len;mid<<=1){
      for(int r=mid<<1,j=0;j<len;j+=r){
        for(int k=0;k<mid;++k){
          cp wn=wt[mid+k];
          if(flag==-1) wn.y=-wn.y;
          cp y=wn*a[mid+j+k];
          a[j+k+mid]=a[j+k]-y,a[j+k]=a[j+k]+y;
        }
      }
    }
    if(flag==-1) rep(i,0,len-1) a[i].x/=len;
  }
  void fix(vector<int>&a){
    while(!a.empty()&&!a.back())a.pop_back();
  }//传入整数, 进行卷积
  void mul(vector<int> &a,vector<int> &b){
    int la=a.size(),lb=b.size();
    init(la+lb);
    rep(i,0,la-1) c1[i]=(cp){a[i],0};
    rep(i,la,len-1) c1[i]=(cp){0,0};
    rep(i,0,lb-1) c2[i]=(cp){b[i],0};
```

```
    rep(i,lb,len-1) c2[i]=(cp){0,0};
    transform(c1,1);transform(c2,1);
    rep(i,0,len-1) c1[i]=c1[i]*c2[i];
    transform(c1,-1);
    a.resize(len);
    rep(i,0,len-1) a[i]=int(c1[i].x+0.5)?1:0;
    fix(a);
  }
  void sqr(vector<int> &a){
    int la=a.size(); init(2*la);
    rep(i,0,la-1)c1[i]=(cp){a[i],0};
    rep(i,la,len-1)c1[i]=(cp){0,0};
    transform(c1,1);
    rep(i,0,len-1) c1[i]=c1[i]*c1[i];
    transform(c1,-1);
    a.resize(len);
    rep(i,0,len-1) a[i]=int(c1[i].x+0.5)?1:0;
    fix(a);
  }
  vector<int> pow(vector<int>a,int k){
    fix(a);
    vector<int>ret;
    while(k){
      if(k&1){
        if(ret.empty()) ret=a;
        else mul(ret,a);
      }
      sqr(a);
      k/=2;
    }
    return ret;
  }
}
```

5.9 多项式-杜教 BM

```
namespace bm{
  const int maxl=1e4+10;
  ll res[maxl],base[maxl],_c[maxl],_md[maxl];
  vector<ll> md;
  ll inv(ll a,ll c=mod) {
    a%=c;if(a<0)a+=c;
    ll b=c,u=0,v=1;
    while(a) {
      ll t=b/a;b-=t*a;
      swap(a,b);u-=t*v;
      swap(u,v);
    }
    if(u<0)u+=c;
    return u;
  }
  void mul(ll *a,ll *b,int k) {
    for(int i=0;i<k+k;i++) _c[i]=0;
    for(int i=0;i<k;i++) if (a[i])
      for(int j=0;j<k;j++) _c[i+j]=(_c[i+j]+a[i]*b[j])%mod;
    for (ll i=k+k-1;i>=k;i--) if (_c[i])
      for(int j=0;j<md.size();j++)
        _c[i-k+md[j]]=(_c[i-k+md[j]]-_c[i]*_md[md[j]])%mod;
    for(int i=0;i<k;i++) a[i]=_c[i];
  }
  int solve(ll n,vector<ll> a,vector<ll> b) {
  //a 系数 b 初值 b[n+1]=a[0]*b[n]+...
  //求出的是第 n+1 项
    ll ans=0,pnt=0;
    ll k=a.size();
    for(int i=0;i<k;i++) _md[k-1-i]=-a[i];_md[k]=1;
    md.clear();
    for(int i=0;i<k;i++) if (_md[i]!=0) md.push_back(i);
    for(int i=0;i<k;i++) res[i]=base[i]=0;
    res[0]=1;
    while ((1ll<<pnt)<=n) pnt++;
    for (ll p=pnt;p>=0;p--) {
      mul(res,res,k);
```

```
    if ((n>>p)&1) {
      for (ll i=k-1;i>=0;i--) res[i+1]=res[i];res[0]=0;
      for(int j=0;j<md.size();j++)
        res[md[j]]=(res[md[j]]-res[k]*_md[md[j]])%mod;
    }
  }
  for(int i=0;i<k;i++) ans=(ans+res[i]*b[i])%mod;
  if (ans<0) ans+=mod;
  return ans;
 }
 vector<ll> init(vector<ll> s) {
  vector<ll> coe(1,1),base(1,1);
  int len=0,m=1,b=1;
  for(int n=0;n<s.size();n++) {
    ll d=0;
    for(int i=0;i<len+1;i++) d=(d+(ll)coe[i]*s[n-i])%mod;
    if (d==0) ++m;
    else if (2*len<=n) {
      vector<ll> tmp=coe;
      ll c=mod-d*inv(b)%mod;
      while (coe.size()<base.size()+m) coe.push_back(0);
      for(int i=0;i<base.size();i++) coe[i+m]=(coe[i+m]+c*base[i])%mod;
      len=n+1-len; base=tmp; b=d; m=1;
    } else {
      ll c=mod-d*inv(b)%mod;
      while (coe.size()<base.size()+m) coe.push_back(0);
      for(int i=0;i<base.size();i++) coe[i+m]=(coe[i+m]+c*base[i])%mod;
      ++m;
    }
  }
  return coe;
 }
 vector<ll> c,a;
 void inita(vector<ll> _a){
  a=_a;
  c=init(a);c.erase(c.begin());
  for(auto &i:c) i=(mod-i)%mod;
 }
 int get(ll n) {
  return solve(n,c,vector<ll>(a.begin(),a.begin()+c.size()));
 }
 int get(vector<ll> a,ll n) {
  vector<ll> c=init(a);
  c.erase(c.begin());
  for(int i=0;i<c.size();i++) c[i]=(mod-c[i])%mod;
  return solve(n,c,vector<ll>(a.begin(),a.begin()+c.size()));
 }
};
```

5.10 多项式-快速数论变换

```
const ll mod=998244353,modg=3,modi=332748118;
int a[maxn],b[maxn];
int pow_mod(int a,int b){
  int ans=1;
  while(b){
    if(b&1) ans=(ll)ans*a%mod;
    a=(ll)a*a%mod,b>>=1;
  }
  return ans;
}
int add(int a,int b){
  a+=b;if(a>mod) return a-mod;
  return a;
}
int sub(int a,int b){
  a-=b;if(a<0) return a+mod;
  return a;
}
#define arr vector<int>
const int maxl=2e4+10;//卷积单个数组的最大长度
arr operator*(arr&,arr&b){
  int len=a.size();
```

```
  arr c(len));
  rep(i,0,len-1) c[i]=a[i]*b[i];
  return c;
}
class nubmer{public:
  int rev[maxl<<2],len,pw;
  int wt[2][maxl<<2];
  void init_0(){
    int len=1;//最开始的初始化, 整个程序一次就够
    while(len<=maxl) len<<=1;
    for(int mid=1;mid<len;mid<<=1){
      ll wn1=pow_mod(modg,(mod-1)/(mid<<1));
      ll wn2=pow_mod(modi,(mod-1)/(mid<<1));
      wt[0][mid]=wt[1][mid]=1;
      ll wt1=wn1,wt2=wn2;
      rep(j,1,mid){
        wt[0][mid+j]=wt1;wt1=wt1*wn1%mod;
        wt[1][mid+j]=wt2;wt2=wt2*wn2%mod;
      }
    }
  }
  void init(int n){
    len=1,pw=0;
    while(len<=n) len<<=1,++pw;--pw;
    rep(i,0,len-1) rev[i]=rev[i>>1]>>1|(i&1)<<pw;
  }
  void transform(arr &a,int flag){
    ll* f=flag==1?wt[0]:wt[1];
    if(a.size()!=len) a.resize(len);
    rep(i,0,len-1) if(i<rev[i]) swap(a[i],a[rev[i]]);
    for(int mid=1;mid<len;mid<<=1){
      for(int r=mid<<1,j=0;j<len;j+=r){
        ll *p=f+mid;
        for(int k=0;k<mid;++k,++p){
          int x=a[j+k],y=(*p)*a[j+k+mid]%mod;
          a[j+k+mid]=sub(a[j+k],y),a[j+k]+=y;
          if(a[j+k]>mod) a[j+k]-=mod;
        }
      }
    }
    if(flag==-1) {
      ll inv=pow_mod(len,mod-2);
      rep(i,0,len-1){
        a[i]=a[i]*inv%mod;
        if(a[i]<0)a[i]+=mod;
      }
    }
  }
  void fix(arr &a){//分治 ntt 优化
    while(!a.empty()&&!a.back()) a.pop_back();
  }//会破坏掉 a,b 数组, 视情况可以去掉引用
  arr mul(arr &a,arr &b){
    int la=a.size(),lb=b.size();
    if(la*lb<=1000){
      arr c(la+lb,0);
      rep(i,0,la-1) rep(j,0,lb-1)
        c[i+j]=(c[i+j]+(ll)a[i]*b[j])%mod;
      return c;
    }
    int n=la+lb;
    init(n);
    transform(a,1);transform(b,1);
    arr c=a*b;
    rep(i,0,len-1)c[i]=(ll)a[i]*b[i]%mod;
    transform(c,-1);
    return c;
  }

}ntt;
```

5.11 多项式-快速沃尔什变换

```
#define add(a,b) ((a+=b)>=mod?a-=mod:a)
```

```
class walsh{public:
  void transform_or(ll *a,int len,int flag){
    for(int i=1;i<len;i<<=1)
      for(int p=i<<1,j=0;j<len;j+=p)
        for(int k=0;k<i;++k)
          add(a[i+j+k],flag==1?a[j+k]:mod-a[j+k]);
  }
  void transform_and(ll *a,int len,int flag){
    for(int i=1;i<len;i<<=1)
      for(int p=i<<1,j=0;j<len;j+=p)
        for(int k=0;k<i;++k)
          add(a[j+k],flag==1?a[i+j+k]:mod-a[i+j+k]);
  }
  void transform_xor(ll *a,int len,int flag){
    for(int i=1;i<len;i<<=1)
      for(int p=i<<1,j=0;j<len;j+=p)
        for(int k=0;k<i;++k){
          int x=a[j+k],y=a[i+j+k];
          a[j+k]=(x+y)%mod,a[i+j+k]=(x+mod-y)%mod;
          if(flag==-1)
            a[j+k]=a[j+k]*inv2%mod,a[i+j+k]=1ll*a[i+j+k]*inv2%mod;
        }
  }
}fwt;
int main(){
  cin>>n;n=(1<<n);
  rep(i,0,n-1) cin>>a[i];
  rep(i,0,n-1) cin>>b[i];
  fwt.transform_or(a,n,1),fwt.transform_or(b,n,1);
  rep(i,0,n-1) ans[i]=a[i]*b[i]%mod;
  fwt.transform_or(a,n,-1),fwt.transform_or(b,n,-1);
  fwt.transform_or(ans,n,-1);
  rep(i,0,n-1) cout<<ans[i]<<' ';cout<<endl;
  fwt.transform_and(a,n,1),fwt.transform_and(b,n,1);
  rep(i,0,n-1) ans[i]=a[i]*b[i]%mod;
  fwt.transform_and(a,n,-1),fwt.transform_and(b,n,-1);
  fwt.transform_and(ans,n,-1);
  rep(i,0,n-1) cout<<ans[i]<<' ';cout<<endl;
  fwt.transform_xor(a,n,1),fwt.transform_xor(b,n,1);
  rep(i,0,n-1) ans[i]=a[i]*b[i]%mod;
  fwt.transform_xor(a,n,-1),fwt.transform_xor(b,n,-1);
  fwt.transform_xor(ans,n,-1);
  rep(i,0,n-1) cout<<ans[i]<<' ';cout<<endl;
}
```

5.12 线性代数-异或线性基

```
template<typename T,const int len> class lbass{public:
  T d[len+1];int cnt;bool flag;
  void init(){flag=0;memset(d,0,sizeof d);}
  bool insert(T x){
    for(int i=len;x&&i>=0;--i)
      if((T)1<<i&x){
        if(!d[i]) {d[i]=x;return true;}
        else x^=d[i];
      }
    flag=1;
    return false;
  }
  //线性基和 x 异或的最值
  T querymax(T x=0){
    per(i,0,len)x=max(x,x^d[i]);
    return x;
  }
  T querymin(T x=0){
    per(i,0,len)x=min(x,x^d[i]);
    return x;
  }
  //求所有异或值去重后的第 k 大, 需要预处理
  T p[len+1];
  void makekth(){
    cnt=0;
    memset(p,0,sizeof p);
```

```
    T ans=0;
    per(i,0,len)per(j,0,i-1)
      if((T)1<<j&d[i]) d[i]^=d[j];
    rep(i,0,len) if(d[i]) p[cnt++]=d[i];
  }
  T querykth(T k){
    if(flag)--k;//包含零
    if(!k) return 0;
    T res=0;
    if(k>=(T)1<<cnt) return -1;
    per(i,0,len) if((T)1<<i&k)res^=p[i];
    return res;
  }
};
template<typename T,const int len> lbass<T,len> merge(const lbass<T,len> &a,const lbass<T,len> &b){
  lbass<T,len>res=a;
  rep(i,0,len) res.insert(b.d[i]);
  return res;
}
```

5.13 线性代数-线段树维护区间线性基

```
class segtree{public:
#define nd node[now]
#define ndl node[now<<1]
#define ndr node[now<<1|1]
  struct segnode{
    int l,r,flag,val;
    int d[32];
    inline void init(){val=flag=0;memset(d,0,sizeof d);}
    inline void insert(ll x){
      for(register int i=30;x&&i>=0;--i)
          if(x&(1ll<<i)){
            if(!d[i]) {d[i]=x;return;}
            else x^=d[i];
          }
    }
    int count(){int ans=0;per(i,0,30) if(d[i])ans++; return ans;}
    void update(int x){val^=x;flag^=x;}
  }node[maxn<<2|3];
  inline segnode marge(segnode &a,segnode b)const {
    segnode ans;ans.init();
    per(i,0,30) ans.insert(a.d[i]),ans.insert(b.d[i]);
    ans.insert(a.val^b.val);
    ans.val=a.val;
    ans.l=a.l,ans.r=b.r;
    return ans;
  }
  inline void down(int now){
    if(nd.flag){
      ndl.update(nd.flag);ndr.update(nd.flag);
      nd.flag=0;
    }
  }
  void maketree(int s,int t,int now=1){
    nd.l=s,nd.r=t;nd.init();
    if(s==t) {cin>>nd.val;return ;}
    maketree(s,(s+t)/2,now<<1);
    maketree((s+t)/2+1,t,now<<1|1);
    nd=marge(ndl,ndr);
  }
  void update(int s,int t,int x,int now=1){
    if(s<=nd.l&&t>=nd.r) {nd.update(x);return;}
    down(now);
    if(s<=ndl.r) update(s,t,x,now<<1);
    if(t>ndl.r) update(s,t,x,now<<1|1);
    nd=marge(ndl,ndr);
  }
  segnode query(int s,int t,int now=1){
    if(s<=nd.l&&t>=nd.r) {
      if(s==nd.l) {
        segnode x;x.init();
        return marge(x,nd);
```

```
    }else return nd;
  }
  down(now);
  segnode ans;ans.init();
  if(s<=ndl.r) ans=marge(ans,query(s,t,now<<1));
  if(t>ndl.r) ans=marge(ans,query(s,t,now<<1|1));
  nd=marge(ndl,ndr);
  return ans;
 }
}tree;
int main() {
  cin>>n>>m;
  register int a,b,c,d;
  tree.maketree(1,n);
  while(m--){
    cin>>a>>b>>c;
    if(a==1)cin>>d;tree.update(b,c,d);
    else cout<<(1<<tree.query(b,c).count())<<endl;
  }
}
```

5.14 线性代数-矩阵运算

```
class matrix{public://mod
 int a,b;
 vector<vector<ll> > x;
 matrix(int a=1,int b=1){
  this->a=a,this->b=b;x.resize(a);
  for(auto &i:x){i.resize(b);std::fill(all(i),0);}
 }
 void e(int n){a=b=n;x=matrix(n,n).x;for(int i=0;i<n;i++)x[i][i]=1;}
 void fill(ll xx=0){for(int i=0;i<a;i++)for(int j=0;j<b;j++)x[i][j]=xx;}
 void fill(vector<vector<ll>> &y){x=y;a=y.size();b=y[0].size();}
 matrix operator *(matrix &m){
  matrix ans(a,m.b);
  for(int i=0;i<a;i++)for(int j=0;j<m.b;j++)for(int k=0;k<b;k++)if(x[i][k]&&m.x[k][j])
   ans.x[i][j]=(mod+ans.x[i][j]+(x[i][k]*m.x[k][j]+mod)%mod)%mod;
  return ans;
 }
 matrix operator +(matrix &m){
  matrix ans(a,m.b);
  for(int i=0;i<a;i++)for(int j=0;j<b;j++)ans.x[i][j]=(x[i][j]+m.x[i][j]+mod)%mod;
  return ans;
 }
 matrix operator -(matrix &m){
  matrix ans(a,m.b);
  for(int i=0;i<a;i++)for(int j=0;j<b;j++)ans.x[i][j]=(x[i][j]-m.x[i][j]+mod)%mod;
  return ans;
 }
 matrix pow(ll p){
  matrix ans;ans.e(a);matrix t;t.fill(x);
  while(p){if(p&1) ans=t*ans;t=t*t;p>>=1;}return ans;
 }
};
```

5.15 杂项-高精度整数类

```
namespace bignumbers{
const int maxd=9999,dlen=4;
class bignumber {public:
 int len,a[maxn];
 bignumber(){len=1;memset(a,0,sizeof(a));}
 bignumber(const ll);
 bignumber(const char*);
 bignumber(const bignumber &);
 bignumber &operator=(const bignumber &);
 friend istream& operator>>(istream&,bignumber&);
 bignumber operator+(const bignumber &)const;
 bignumber operator-(const bignumber &)const;
 bignumber operator*(const bignumber &)const;
 bignumber operator/(const int &)const;
 bignumber operator^(const int &)const;
 ll operator%(const ll &)const;
 bool operator>(const bignumber &T)const;
```

```
  bool operator>(const int &t)const;
  void print();
};
bignumber::bignumber(const ll b) {
  int c,d=b;len=0;
  memset(a,0,sizeof(a));
  while(d>maxd){
    c=d-(d/(maxd+1))*(maxd+1);
    d=d/(maxd+1);
    a[len++]=c;
  }
  a[len++]=d;
}
bignumber::bignumber(const char *s) {
  int t,k,index,L;
  memset(a,0,sizeof(a));
  L=strlen(s);
  len=L/dlen;
  if(L%dlen)len++;
  index=0;
  for(int i=L-1;i>=0;i-=dlen) {
    t=0,k=i-dlen+1;
    if(k<0)k=0;
    rep(j,k,i)t=t*10+s[j]-'0';
    a[index++]=t;
  }
}
bignumber::bignumber(const bignumber &T):len(T.len) {
  memset(a,0,sizeof(a));
  rep(i,0,len-1)a[i]=T.a[i];
}
bignumber &bignumber::operator=(const bignumber &n) {
  memset(a,0,sizeof(a));
  rep(i,0,n.len-1)a[i]=n.a[i];
  len=n.len;
  return *this;
}
char ch[maxn*dlen];
istream& operator>>(istream &in,bignumber &b) {
  in>>ch;
  int L=strlen(ch);
  int count=0,sum=0;
  for(int i=L-1;i>=0;) {
    int t=1;sum=0;
    for(int j=0;j<4&&i>=0;j++,i--,t*=10)
      sum+=(ch[i] - '0')*t;
    b.a[count++]=sum;
  }
  b.len=count++;
  return in;
}
bignumber bignumber::operator+(const bignumber &T)const {
  bignumber t(*this);
  int big=T.len>len?T.len:len;
  rep(i,0,big-1){
    t.a[i]+=T.a[i];
    if(t.a[i]>maxd)t.a[i+1]++,t.a[i]-=maxd+1;
  }
  if(t.a[big]!=0) t.len=big+1;
  else t.len=big;
  return t;
}
bignumber bignumber::operator-(const bignumber &T)const {
  bool flag;
  bignumber t1,t2;
  if(*this>T) {
    t1=*this;t2=T;flag=0;
  } else {
    t1=T;t2=*this;flag=1;
  }
  int j,big=t1.len;
  rep(i,0,big-1){
    if(t1.a[i]<t2.a[i]) {
```

```
      j=i+1;
      while(t1.a[j]==0) j++;
      t1.a[j--]--;
      while(j>i) t1.a[j--]+=maxd;
      t1.a[i]+=maxd+1-t2.a[i];
    } else t1.a[i]-=t2.a[i];
  }
  t1.len=big;
  while(t1.a[t1.len-1]==0&&t1.len>1)
    t1.len--,big--;
  if(flag) t1.a[big-1]=0-t1.a[big-1];
  return t1;
}
bignumber bignumber::operator*(const bignumber &T)const {
  bignumber ret;
  int up,temp,temp1;
  rep(i,0,len-1){
    up=0;
    rep(j,0,T.len-1){
      temp=a[i]*T.a[j]+ret.a[i+j]+up;
      if(temp>maxd) {
        temp1=temp-temp/(maxd+1)*(maxd+1);
        up=temp/(maxd+1),ret.a[i+j]=temp1;
      } else up=0,ret.a[i+j]=temp;
    }
    if(up!=0)ret.a[i+T.len]=up;
  }
  ret.len=len+T.len;
  while(ret.a[ret.len-1]==0&&ret.len>1)ret.len--;
  return ret;
}
bignumber bignumber::operator/(const int &b)const {
  bignumber ret;
  int down=0;
  per(i,0,len-1) {
    ret.a[i]=(a[i]+down*(maxd+1))/b;
    down=a[i]+down*(maxd+1) - ret.a[i]*b;
  }
  ret.len=len;
  while(ret.a[ret.len-1]==0&&ret.len>1) ret.len--;
  return ret;
}
ll bignumber::operator%(const ll &b)const {
  int d=0;
  per(i,0,len-1)
    d=((d*(maxd+1))%b+a[i])%b;
  return d;
}
bignumber bignumber::operator^(const int &n)const {
  if(n==0)return 1;
  if(n==1)return *this;
  bignumber t,ret(1);
  int m=n,i;
  while(m>1) {
    t=*this;
    for(i=1;(i<<1)<=m;i<<=1)t=t*t;
    m-=i,ret=ret*t;
    if(m==1)ret=ret*(*this);
  }
  return ret;
}
bool bignumber::operator>(const bignumber &T)const {
  if(len>T.len)return true;
  else if(len==T.len) {
    int ln=len-1;
    while(a[ln]==T.a[ln]&&ln>=0)ln--;
    if(ln>=0&&a[ln]>T.a[ln]) return true;
    else return false;
  }else return false;
}
bool bignumber::operator>(const int &t)const {
  bignumber b(t);
  return *this>b;
```

```
}
void bignumber::print() {
  printf("%d",a[len-1]);
  per(i,0,len-2)printf("%04d",a[i]);
  printf("\n");
}
}using bignumbers::bignumber;
```

5.16 杂项-分数类

```
//num 是分子,den 是分母, 分母始终保持为正
template<typename T>class farction{public:
  T num,den;
  farction(T num=0,T den=1) {
    if (den<0) {
      num=-num;
      den=-den;
    }
    T g=__gcd(abs(num),den);
    this->num=num/g;
    this->den=den/g;
  }
  farction operator +(const farction &o) const {
    return farction(num*o.den+den*o.num,den*o.den);
  }
  farction operator -(const farction &o) const {
    return farction(num*o.den-den*o.num,den*o.den);
  }
  farction operator *(const farction &o) const {
    return farction(num*o.num,den*o.den);
  }
  farction operator /(const farction &o) const {
    return farction(num*o.den,den*o.num);
  }
  bool operator <(const farction &o) const {
    return num*o.den<den*o.num;
  }
  bool operator >(const farction &o) const {
    return num*o.den>den*o.num;
  }
  bool operator ==(const farction &o) const {
    return num*o.den==den*o.num;
  }
};
```

5.17 杂项-N 进制快速幂优化

```
//底数不变, 求值次数 1e5 以上的情况, 可以加速 10-100 倍
//1e9: maxp 1e3,maxv 1e9,100 倍
//1e18: maxp 32000,maxv 1e18,14 倍左右
const ll maxp=32000,maxv=1e18;
const ll maxw=log(maxv)/log(maxp)+1;
class basepow{public:
  ll pw[maxw][maxp];
  void init(ll base){
    base%=mod;
    rep(i,0,maxw-1)pw[i][0]=1;
    rep(i,1,maxp-1) pw[0][i]=(pw[0][i-1]*base)%mod;
    rep(i,1,maxw-1){
      pw[i][1]=pw[i-1][maxp-1]*pw[i-1][1]%mod;
      rep(j,2,maxp-1) pw[i][j]=pw[i][j-1]*pw[i][1]%mod;
    }
  }
  inline ll getpow(ll b,ll res=1,int cnt=0){
    while(b){
      res=res*pw[cnt++][b%maxp]%mod;
      b/=maxp;
    }
    return res;
  }
}p;
```

6 String

6.1 KMP 算法

```
//输入的数组从 0 开始,next 函数为 p, 从 1 开始
template<typename T>class prefix{public:
  int p[maxn],lens;
  T *s;
  void init(T *_s,int _lens){
    s=_s,lens=_lens;
    rep(i,0,lens-1) p[i]=0;
    p[0]=-1;
    int now=0,pos=-1;
    while(now<lens)
      if(pos==-1||s[now]==s[pos]) p[++now]=++pos;
      else pos=p[pos];
  }
  vector<int> find(T *t,int lent){
    int now,pos=0;
    vector<int> ans;
    while(now<lent) {
      if(pos==-1||t[now]==s[pos]) pos++,now++;
      else pos=p[pos];
      if(pos==lens) pos=p[pos],ans.push_back(now-lens);
    }
    return ans;
  }
};
prefix<char> kmp;
```

6.2 manacher 算法

```
class manacher{
  char ma[maxn<<1];int lenp[maxn<<1];
  void getp(char *s,int len){
    int p=0;
    ma[p++]='$',ma[p++]='#';
    forn(i,len) ma[p++]=s[i],ma[p++]='#';
    int r=0,mid=0;
    forn(i,(len+1)<<1){
      lenp[i]=r>i?min(lenp[(mid<<1)-i],r-i):1;
      while(ma[i+lenp[i]]==ma[i-lenp[i]]) lenp[i]++;
      if(i+lenp[i]>r)r=i+lenp[i],mid=i;
    }
  }
}
```

6.3 后缀数组-倍增

```
namespace suffix{
  int tr[maxn],rank[maxn],sa[maxn],h[maxn],has[maxn],n;
  int cmp(int x,int y,int k){
    if(x+k>n||y+k>n)return 0;
    return rank[x]==rank[y]&&rank[x+k]==rank[y+k];
  }
  void getsa(char *s,int _n,int m=233){
    int i,cnt;n=_n;
    for(i=1;i<=n;i++)has[s[i]]=0;
    for(i=0;i<=m;i++)has[i]=0;
    for(i=1;i<=n;i++)has[s[i]]++;
    for(i=1,cnt=0;i<=m;i++)if(has[i])tr[i]=++cnt;
    for(i=1;i<=m;i++)has[i]+=has[i-1];
    for(i=1;i<=n;i++)rank[i]=tr[s[i]],sa[has[s[i]]--]=i;
    for(int k=1;cnt!=n;k<<=1){
      for(i=1;i<=n;i++)has[i]=0;
      for(i=1;i<=n;i++)has[rank[i]]++;
      for(i=1;i<=n;i++)has[i]+=has[i-1];
      for(i=n;i>=1;i--)if(sa[i]>k)tr[sa[i]-k]=has[rank[sa[i]-k]]--;
      for(i=1;i<=k;i++)tr[n-i+1]=has[rank[n-i+1]]--;
      for(i=1;i<=n;i++)sa[tr[i]]=i;
      for(i=1,cnt=0;i<=n;i++)tr[sa[i]]=cmp(sa[i],sa[i-1],k) ? cnt:++cnt;
      for(i=1;i<=n;i++)rank[i]=tr[i];
    }
    fill_n(h,n+2,0);
    for(int i=1;i<=n;i++){
```

```
    if(rank[i]==1)continue;
    for(int j=max(1,h[rank[i-1]]-1);;j++){
      if(s[i+j-1]==s[sa[rank[i]-1]+j-1])h[rank[i]]=j;
      else break;
    }
  }
 }
}
```

6.4 最小表示算法

```
int minrep(int *s,int n){
  int i=0,j=1,k=0,t;
  while(i<n&&j<n&&k<n){
    t=s[i+k>=n?i+k-n:i+k]-s[j+k>=n?j+k-n:j+k];
    if(!t) ++k;
    else {
      if(t>0) i+=k+1;
      else j+=k+1;
      if(i==j) ++j;
      k=0;
    }
  }
  return min(i,j);
}
```

6.5 字符串哈希

```
const ll decm=13331,modh=(ll)1e17+13;
inline ll mulh(ll a,ll b){
  ll tmp=(a*b-(ll)((long double)a/modh*b+1.0e-8)*modh);
  return tmp<0?tmp+modh:tmp;
}
inline ll addh(ll a,ll b){ll x=a+b;if(x<0)return a+b+modh;else if(x>=modh)x-modh;else return x;}
ll pw[maxn];
void init(int n=maxn-5){pw[0]=1;rep(i,1,n) pw[i]=mulh(pw[i-1],decm);
class strhash{public:
  ll hs[maxn],len;
  void calhs(string &s,int n){len=n;rep(i,0,len-1)hs[i+1]=addh(mulh(hs[i],decm),s[i]);}
  inline ll geths(int a,int b){return addh(hs[b],-mulh(hs[a-1],pw[b-a+1]));}
}table;
```

6.6 自动机-tire 树

```
const int csize=26,minc='a';
class trie{public:
#define nd node[now]
  struct tnode{
    int cnt,son[csize];
    tnode(){
      cnt=0;
      memset(son,0,sizeof son);
    }
  }node[maxn];
  int sz=0;
  void insert(char *s,int len){
    int now=0;
    rep(i,0,len-1){
      int ch=s[i]-minc;
      if(!nd.son[ch]) nd.son[ch]=++sz;
      now=nd.son[ch];
    }
    node[now].cnt++;
  }
  int find(char *s,int len){
    int ch,now=0;
    rep(i,0,len-1){
      ch=s[i]-minc;
      if(!nd.son[ch]) return -1;
      now=nd.son[ch];
    }
    if(!nd.cnt) return -1;
    return 1;
  }
}tree;
```

6.7 自动机-tire 图

```cpp
#include<iostream>
#include<cstdio>
#include<cstring>
#include <vector>
#include <queue>
using namespace std;

const int N = 1000000 + 10, INF = 0x3f3f3f3f;

struct node {
  node *next[26];
  node *suff; //指向后缀节点
  bool flag;
} trie[N], *root;
int tot;
char ori[N];
node* node_init() {
  trie[tot].flag = false;
  trie[tot].suff = NULL;
  memset(trie[tot].next, 0, sizeof trie[tot].next);
  return trie + tot++;
}
void trie_insert(char *s) {
  node *p = root;
  for(int i = 0; s[i]; i++) {
    int j = s[i] - 'a';
    if(p->next[j] == NULL)
      p->next[j] = node_init();
    p = p->next[j];
  }
  p->flag = true;
}
void trie_graph() {
  //trie[0] 为虚拟节点，root 为 trie[1]，trie[0] 的所有边均指向 root，方便以后操作
  for(int i = 0; i < 26; i++)
    trie[0].next[i] = root;
  root->suff = trie + 0;
  trie[0].suff = NULL;
  queue<node*> que;
  que.push(root);
  while(! que.empty()) {
    node *p = que.front();
    que.pop();
    for(int i = 0; i < 26; i++)
      if(p->next[i]) {
        //查看父亲节点的后缀节点是否存在编号为 i 的边，若没有，就一直往上找，直到根节点
        node *ptr = p->suff;
        while(ptr && ! ptr->next[i])
          ptr = ptr->suff;
        //要么找到了，要么循环到了虚拟节点后，后缀节点被置为 root
        p->next[i]->suff = ptr->next[i];
        if(ptr->next[i]->flag)
          p->next[i]->flag = true;
        que.push(p->next[i]);
      }
  }
}
bool trie_query(char *s) {
  node *p = root;
  for(int i = 0; s[i]; i++) {
    int j = s[i] - 'a';
    while(true) {
      if(p->next[j] != NULL) {
        p = p->next[j];
        if(p->flag == true)
          return true;
        break;
      } else
        p = p->suff;
    }
  }
```

```
  return false;
}
int main() {
  int n;
  scanf("%d", &n);
  tot = 2;//tot+0 是虚拟节点, tot+1 是根节点, 故从 2 开始
  root = trie + 1;
  memset(trie[root].next, 0, sizeof trie[root].next);
  for(int i = 1; i <= n; i++) {
    scanf("%s", ori);
    trie_insert(ori);
  }
  trie_graph();//建 trie 图
  scanf("%s", ori);
  puts(trie_query(ori) ? "YES" : "NO");
  return 0;
}
```

6.8 自动机-AC 自动机

```
const int csize=128,minc=0;
class autom{public:
#define nd node[now]
  struct acnode{int id,son[csize],fail;}node[maxn];
  int sz=0;
  queue<int> que;
  void clear(int n=maxn-5){
    memset(node,0,sizeof(acnode)*n);
    sz=0;
  }
  void insert(int id,char *s,int len=0){
    if(!len)len=strlen(s);
    int now=0;
    rep(i,0,len-1){
      int ch=s[i]-minc;
      if(!nd.son[ch]) nd.son[ch]=++sz;
      now=nd.son[ch];
    }
    nd.id=id;
  }
  void init(){
    int now=0;
    rep(i,0,csize-1) if(nd.son[i])
      que.push(nd.son[i]);
    while(!que.empty()){
      now=que.front();que.pop();
      rep(i,0,csize-1){
        if(nd.son[i]) {
          node[nd.son[i]].fail=node[nd.fail].son[i];
          que.push(nd.son[i]);
        }else nd.son[i]=node[nd.fail].son[i];
      }
    }
  }
  vector<int> query(char *t,int len=0){
    if(!len)len=strlen(t);
    int now=0;
    vector<int> ans;
    rep(i,0,len-1) {
      now=nd.son[t[i]-minc];
      for(int j=now;j;j=node[j].fail)
        if(node[j].id)ans.push_back(node[j].id);
    }
    return ans;
  }
}acam;
```

6.9 自动机-后缀自动机

```
const int maxn=1e5+10,maxsz=26+1,ch0='a';
class saffixam {public:
  int fa[maxn<<1],son[maxn<<1][maxsz];
  int last,cnt,len[maxn<<1];
  void insert(int ch) {
```

```
    int pre=last,now=++cnt;
    last=now,len[now]=len[pre]+1;
    for(; pre&&!son[pre][ch]; pre=fa[pre])son[pre][ch]=now;
    if(!pre)fa[now]=1;
    else {
      int q=son[pre][ch];
      if(len[pre]+1==len[q])fa[now]=q;
      else {
        int nq=++cnt;
        memcpy(son[nq],son[q],sizeof(son[0]));
        fa[nq]=fa[q];
        len[nq]=len[pre]+1;
        fa[q]=fa[now]=nq;
        for(; son[pre][ch]==q; pre=fa[pre])son[pre][ch]=nq;
      }
    }
  }
  void init() {
    rep(i,0,cnt) {
      memset(son[i],0,sizeof son[0]);
      fa[i]=len[i]=0;
    }
    last=cnt=1;
  }
  void insert(string &s ) {for(auto i:s)insert(i-ch0);}
  bool find(string &s) {
    int now=1;
    for(auto i:s) if(!(now=son[now][i-ch0]))return 0;
    return 1;
  }
} sam;
```

6.10 自动机-回文自动机

```
struct PAT {
  struct node {
    int len,num,fail,son[26];
  } t[maxn];
  int last,n,tot,s[maxn];
  void init() {
    memset(t,0,sizeof(t));
    tot=last=1;
    n=0;
    t[0].len=0;
    t[1].len=-1;
    t[0].fail=t[1].fail=1;
    s[0]=-1;
  }
  int add(int c) {
    int p=last;
    s[++n]=c;
    while(s[n]!=s[n-1-t[p].len])
      p=t[p].fail;
    if(!t[p].son[c]) {
      int v=++tot,k=t[p].fail;
      while(s[n]!=s[n-t[k].len-1])
        k=t[k].fail;
      t[v].fail=t[k].son[c];
      t[v].len=t[p].len+2;
      t[v].num=t[t[v].fail].num+1;
      t[p].son[c]=v;
    }
    last=t[p].son[c];
    return t[last].num;
  }
} T;
```

7 Others

7.1 常用头文件

```
#include<bits/stdc++.h>
#define ll long long
#define rep(ii,a,b) for(int ii=a;ii<=b;++ii)
#define per(ii,a,b) for(int ii=b;ii>=a;--ii)
```

```
#define forn(i,x,g,e) for(int i=g[x];i;i=e[i].next)
#define IO ios::sync_with_stdio(false);cin.tie(0);cout.tie(0)
#define ull unsigned long long
#define fi first
#define se second
#define mp make_pair
#define pii pair<ll,ll>
#define all(x) x.begin(),x.end()
#define show(x) cout<<#x<<"="<<x<<endl
#define showa(a,b) cout<<#a<<'['<<b<<"]="<<a[b]<<endl
#define show2(x,y) cout<<#x<<"="<<x<<" "<<#y<<"="<<y<<endl
#define show3(x,y,z) cout<<#x<<"="<<x<<" "<<#y<<"="<<y<<" "<<#z<<"="<<z<<endl
#define show4(w,x,y,z) cout<<#w<<"="<<w<<" "<<#x<<"="<<x<<" "<<#y<<"="<<y<<" "<<#z<<"="<<z<<endl
#define show5(v,w,x,y,z) cout<<#v<<"="<<v<<" "<<#w<<"="<<w<<" "<<#x<<"="<<x<<" "<<#y<<"="<<y<<" "<<#z<<"="<<z<<endl
#define showa2(x,a,b) cout<<#x<<": ";rep(i,a,b) cout<<x[i]<<' ';cout<<endl
using namespace std;//head
const int maxn=1e6+10,maxm=2e6+10;
const ll INF=0x3f3f3f3f,mod=1e9+7;
int casn,n,m,k;
int main() {



}
//struct node{
//  int x,y;
//  friend bool operator<(const node&a,const node&b){
//    return a.x==b.x?a.y<b.y:a.x<b.x;
//  }
//}a[maxn];
//  auto _start=chrono::high_resolution_clock::now();
//  auto _end=chrono::high_resolution_clock::now();
//  cerr<<"elapsed time: "<<chrono::duration<double,milli>(_end-_start).count()<<" ms\n";
//int size=(64)<<20; //64MB
//  __asm__("movq %0, %%rsp\n"::"r"((char*)malloc(size)+size));
```

7.2 快速读写

```
namespace io{
  const int L=(1<<21)+1;
  char ibuf[L],*iS,*iT,obuf[L],*oS=obuf,*oT=obuf+L-1,c,st[55];int f,tp;
  #define gc() (iS==iT?(iT=(iS=ibuf)+fread(ibuf,1,L,stdin),(iS==iT?EOF:*iS++)):*iS++)
  inline void flush(){fwrite(obuf,1,oS-obuf,stdout);oS=obuf;}
  inline void putc(char x){*oS++=x;if(oS==oT)flush();}
  template<class I>
  inline void gi(I&x){
    for(f=1,c=gc();c<'0'||c>'9';c=gc())if(c=='-')f=-1;
    for(x=0;c<='9'&&c>='0';c=gc())x=x*10+(c&15);x*=f;
  }
  template<class I>
  inline void print(I x){
    if(!x)putc('0');if(x<0)putc('-'),x=-x;
    while(x)st[++tp]=x%10+'0',x/=10;
    while(tp)putc(st[tp--]);
  }
  inline void gs(char*s,int&l){
    for(c=gc();c!='_'&&(c<'a'||c>'z');c=gc());
    for(l=0;c=='_'||c<='z'&&c>='a';c=gc())s[l++]=c;
  }
};
```

7.3 快速读写加强版

```
namespace fastio{//支持读取整数, 字符串, 输出整数
bool isdigit(char c){return c>=48&&c<=57;}
const int maxsz=1e7;
class fast_iostream{public:
  char ch=get_char();
  bool endf=1,flag;
  char get_char(){
    static char buffer[maxsz],*a=buffer,*b=buffer;
    return b==a&&(b=(a=buffer)+fread(buffer,1,maxsz, stdin),b==a)?EOF:*a++;
  }
```

```
  template<typename type>bool get_int(type& tmp){
    flag=tmp=0;
    while(!isdigit(ch)&&ch!=EOF){flag=ch=='-';ch=get_char();};
    if(ch==EOF)return endf=0;
    do{tmp=ch-48+tmp*10;}while(isdigit(ch=get_char()));
    if(flag)tmp=-tmp;
    return 1;
  }
  int get_str(char* str){
    char* tmp=str;
    while(ch=='\r'||ch=='\n'||ch==' ')ch=get_char();
    if(ch==EOF)return(endf=0),*tmp=0;
    do{*(tmp++)=ch;ch=get_char();}while(ch!='\r'&&ch!='\n'&&ch!=' '&&ch!=EOF);
    *(tmp++)=0;
    return(int)(tmp-str-1);
  }
  fast_iostream& operator>>(char* tmp){get_str(tmp);return *this;}
  template<typename type>fast_iostream& operator>>(type& tmp){get_int(tmp);return *this;}
  operator bool() const {return endf;}
};
template<typename type>void put(type tmp){
  if (tmp==0){putchar(48);return;}
  static int top,stk[21];
  if (tmp<0){tmp=-tmp;putchar('-');}
  while(tmp)stk[++top]=tmp%10,tmp/=10;
  while(top)putchar(stk[top--]+48);
}
}fastio::fast_iostream io;
```

7.4 LIS

```
int num[maxn],dp[maxn],len;
int lwb(int now){
  int l=0,r=len,ans=0;
  while(l<r){
    int mid=(l+r)>>1;
    if(dp[mid]>=now) l=mid+1;
    else r=mid;
  }
  return l;
}
int lwb2(int now){
  int l=0,r=len;
  while(l<r){
    int mid=(l+r)>>1;
    if(dp[mid]>=now) r=mid;
    else l=mid+1;
  }
  return l;
}
int main(){
  cin>>n;
  for(int i=1;i<=n;i++) cin>>num[i];
  for(int i=1;i<=n;i++){
    if(dp[len]>=num[i]) dp[++len]=num[i];
    else dp[lwb(num[i])]=num[i];
  }
  cout<<len<<endl;
  len=0;
  memset(dp,0,sizeof dp);
  for(int i=1;i<=n;i++){
    if(dp[len]<num[i]) dp[++len]=num[i];
    else dp[lwb2(num[i])]=num[i];
  }
  cout<<len<<endl;
}
```

7.5 莫队算法

```
ll a[maxn],ans[maxn],sz,sum;
class block{public:
  ll cnt[maxn];
  struct node{ll l,r,id;
    bool operator <(const node &b)const {
      if(l/sz!=b.l/sz) return l<b.l;
```

```
      if((l/sz)&1) return r<b.r;
      return r>b.r;
    }
  };
  void update(int pos,ll flag=1){
    sum+=flag*a[pos]*(cnt[a[pos]]*2ll+flag);
    cnt[a[pos]]+=flag;
  }
}ask;
vector<block::node> tab;
int main() {
  cin>>n>>m;sz=sqrt(n);
  rep(i,1,n) cin>>a[i];
  rep(i,1,m) {
    int a,b;cin>>a>>b;
    tab.push_back({a,b,i});
  }
  sort(all(tab));
  int l=tab[0].l,r=tab[0].l-1;
  for(auto now:tab){
    while(l>now.l) ask.update(--l,1);
    while(l<now.l) ask.update(l++,-1);
    while(r<now.r) ask.update(++r,1);
    while(r>now.r) ask.update(r--,-1);
    ans[now.id]=sum;
  }
}
```

7.6 大随机数

```
template<typename T>class random_xor{public:
  T s1,s2,s3,s4;
//   random_xor(){
//    srand(time(0)+rand());
//    s1=(T)rand()*rand()*rand()*rand();
//    s2=(T)rand()*rand()*rand()*rand();
// }
// T getint(){
//    T s3=s1,s4=s2;
//    s1=s4;s3^=s3<<23;
//    s2=s3^s4^(s3>>17)^(s4>>26);
//    return s2+s4;
// }
  random_xor(){
    srand(time(0)+rand());
    s1=(T)rand()*rand()*rand()*rand();
    s2=(T)rand()*rand()*rand()*rand();
    s3=(T)rand()*rand()*rand()*rand();
    s4=(T)rand()*rand()*rand()*rand();
  }
  T getint() {
    T t=s1^(s1<<11);
    s1=s2;s2=s3;s3=s4;
    return s4=s4^(s4>>19)^t^(t>>8);
  }
  T get(T l,T r){return getint()%(r-l+1)+l;}
  T operator ()(T l,T r){return getint()%(r-l+1)+l;}
  T operator ()(){return getint();}
};
random_xor<unsigned __int128> a;
srand(time(0));
//mt19937::result_type seed=time(0);
auto randint=bind(uniform_int_distribution<int>(l,r),mt19937(time(0)));
auto randfloat=bind(uniform_real_distribution<double>(l,r),mt19937(time(0)));
```

7.7 大质数

```
1e9+7,1e9+9,1e9+21,1e9+33,
1e9+87,1e9+93,1e9+97,1e9+103,
1e9+123,1e9+181,1e9+207,1e9+223,
1e9+241,1e9+271,1e9+289,1e9+297,
1e9+321,1e9+349,1e9+363,1e9+403,
```

7.8 编译器位操作

```
#pragma GCC target ("popcnt")
__builtin_parity(n);//1 的个数的奇偶性
__builtin_popcount(n);//1 个数
__builtin_ctz(n);//末尾 0 个数,n!=0
__builtin_clz(n);//前导 0 的个数,n!=0
__builtin_ffs(n);//最后 1 的位置, 从 1 开始
__builtin_parity(n)//x 中 1 的奇偶性
uint32_t __builtin_bswap32(uint32_t x)//按字节翻转
```

7.9 最大子矩阵算法

```
/*
把二维看成一维
先枚举行的起点和终点
再把起点行和终点行间每一列的数值压缩到每一个点上
然后求一个最长连续子段和
复杂度 O(n^3)
*/
#include <string.h>
#include <stdio.h>
#include <iostream>
#include <algorithm>
using namespace std;
const int maxn=1e3+10;
const int maxm=1e6+10;
const int INF=0x3f3f3f3f;
#define ll long long
int casn,n,m,k;
int smax(int a[],int len){
  int mx=0,sub=0;
  for(int i=1;i<=len;i++){
    sub=max(a[i],sub+a[i]);
    mx=max(sub,mx);
  }
  return mx;
}
int arr[maxn];
int dp[maxn][maxn];
int main(){
#define test
#ifdef test
  freopen("in.txt","r",stdin);freopen("out.txt","w",stdout);
#endif

  while(~scanf("%d",&n)){
    for(int i=1;i<=n;i++){
      for(int j=1;j<=n;j++){
        scanf("%d",&dp[i][j]);
      }
    }
    int ans=-INF;
    for(int i=1;i<=n;i++){
      memset(arr,0,sizeof arr);
      for(int j=i;j<=n;j++){
        for(int k=1;k<=n;k++){
          arr[k]+=dp[j][k];
        }
        ans=max(ans,smax(arr,n));
      }
    }
    printf("%d\n",ans);
  }
#ifdef test
  fclose(stdin);fclose(stdout);system("out.txt");
#endif
  return 0;
}
```

7.10 约瑟夫环算法

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
```

```
ll ysf(ll n,ll k,ll num) //n 为人数 k 为每次点名的数第 num 个出列的人的序号 0- n-1
{
    if(k==1)
        return num-1;
    ll ans=0;
    if(num<k)
    {
        ans=(k-1)%(n-num+1);
        ll p=n-num+1;
        for(ll i=2; i<=num; i++)
        {
            ans=(ans+k)%(++p);
        }
    }
    else                                //num 很大时用乘法加速
    {
        ans=-1;
        for(ll i=n-num+1; i<=n; i++)
        {
            ll j=min(n,(i-1)+((i-1)-ans+(k-2))/(k-1));
            ans=(ans+k*(j-i+1))%j,i=j;
        }
    }

    return ans;
}
int main()
{
    ll T,T2;
    cin>>T;
    T2=T;
    while (T--)
    {
        ll n, k,num;
        cin>>n>>num>>k;
        cout<<"Case #"<<T2-T<<": "<<ysf(n,k,num)+1<<endl;
    }


}
```

7.11 java 高精度

```
package acm;
import java.math.BigInteger;
import java.util.Scanner;
public class Main{
  public static final int maxn=1000,maxm=200000;
  public static BigInteger gcd(BigInteger a,BigInteger b) {
    if(b.compareTo(BigInteger.ZERO)==0)return a;
    else return gcd(b, a.remainder(b));
  }
  public static void main(String[] argc){
    int[]a=new int[maxn];
    for(int i=2;i<maxn;i++) a[i]=i;
    for(int i=2;i<maxn;i++) if(a[i]!=0)
      for(int j=i*2;j<maxn;j=j+i) a[j]=0;
    Scanner cin=new Scanner(System.in);
    int casn=cin.nextInt();
    int[] prime=new int[10000];
    for(int ii=1;ii<=casn;++ii) {
      BigInteger x=cin.nextBigInteger();
      BigInteger res=BigInteger.ONE;
      BigInteger now=BigInteger.ONE;
      int i=1;
      while(true) {
        i++;
        if(a[i]==0) continue;
        if(res.multiply(BigInteger.valueOf(a[i])).compareTo(x)<=0) {
          res=res.multiply(BigInteger.valueOf(a[i]));
          now=now.multiply(BigInteger.valueOf(a[i]+1));
```

```
29          }else break;
30        }
31        BigInteger g=gcd(res,now);
32        res=res.divide(g);
33        now=now.divide(g);
34        System.out.println(res+"/"+now);
35      }
36    }
37  }
```