JYCの算法竞赛模板

求个star(っ•ω•)っ»(っ•ω•)っ»(っ•ω•)っ»

目录

缺省源

模板正式开始

```cpp
#include<bits/stdc++.h>
using namespace std;

//#define ONLINE
#ifndef ONLINE
#define debug(...) fprintf(stderr,##__VA_ARGS__)
#else
#define debug(...) ;
#endif

using LL=long long;
using PII=pair<int,int>;

const int mod=998244353;
mt19937 rng(chrono::system_clock::now().time_since_epoch().count());

//double关键字比大小
#define eps (1e-8)
inline int sign(const double& x){
    if(fabs(x)<eps) return 0;
    return x>0.0?1:-1;
}
inline int dcmp(const double& x,const double& y){
    return sign(x-y);
}

//快读
template<typename T>
inline T READ(){
    T x=0; bool f=0; char c=getchar();
    while(!isdigit(c)) f|=(c=='-'),c=getchar();
    while(isdigit(c)) x=x*10+c-'0',c=getchar();
    return f?-x:x;
}
inline int read(){return READ<int>();}
inline LL readLL(){return READ<LL>();}
```

数据结构

ST 表

```cpp
#define N 1000010
namespace ST_C{ //Sparse Table Constant
    vector<int>Logn;
    void init(int maxn=N){
        Logn.resize(maxn);
        Logn[1]=0;
        Logn[2]=1;
        for (int i=3;i<maxn;i++) {
            Logn[i]=Logn[i/2]+1;
        }
    }
}
template<typename T>
class ST{
    using VT=vector<T>;
    using VVT=vector<VT>;
    using func_t=function<T(const T&,const T&)>; //func_t is func type
    VVT a; //a is Sparse Table
    func_t op;
public:
    //v的有效下标为 1~n
    //func是比较函数，得传个std::function<T(T,T)>或者lamdba表达式
    //auto max_int=[](const int& x,const int& y)->int{return x>y?x:y;};
    ST(){}
    ST(const vector<T> &v,int n,func_t func){
        init(v,n,func);
    }
    void init(const vector<T> &v,int n,func_t func){
        op=func;
        int mx_l=ST_C::Logn[n]+1; //max log
        a.assign(n+1,VT(mx_l,0));
        for(int i=1;i<=n;i++){
            a[i][0]=v[i];
        }
        for(int j=1;j<mx_l;j++){
            int p=(1<<(j-1));
            for(int i=1;i+p<=n;i++){
                a[i][j]=op(a[i][j-1],a[i+(1<<(j-1))][j-1]);
            }
        }
    }
    T query(int l,int r){
        int lt=r-l+1;
        int p=ST_C::Logn[lt];
        return op(a[l][p],a[r-(1<<p)+1][p]);
    }
};
```

树状数组

```
template<typename T>//T is int or LL
class FenTree{
private:
    int n;
    vector<T>c;
    inline int lowbit(const int& x){
        return x&(-x);
    }
public:
    FenTree(int n_){
        c.assign(n=n_,T(0));
    }
    void init(int n_){
        c.assign(n=n_,T(0));
    }
    void add(int i,int x){
        for(;i<n;i+=lowbit(i)) c[i]+=x;
    }
    T query(int i){
        T res=0;
        for(;i;i-=lowbit(i)) res+=c[i];
        return res;
    }
};
```

线段树

封装的不是很好，具体 $Info$ 的传递还是得自己修改源码，没法直接传个 $class$ 进来

初始化时，传入一个一维 $vector\ a$ 以及它的长度 n，数据存储在下标 $1\sim n$

```
#define ls (id<<1)
#define rs (id<<1|1)
class SGT{//以线段树维护区间最大值为例，支持区间修改
private:
    struct Node{
        int l,r;
        int mx; bool tag;
        friend Node operator +(const Node& A,const Node& B){
            Node ret;
            ret.l=A.l,ret.r=B.r;

            //update the main content
            ret.mx=max(A.mx,B.mx);
            ret.tag=0;

            return ret;
        }
    };
    vector<Node>q;
    void spread(int id){
        if(q[id].l==q[id].r) return;
        //spread the lazy tag
        if(q[id].tag){
            q[ls].mx=q[id].mx;
            q[rs].mx=q[id].mx;
            q[ls].tag=1;
            q[rs].tag=1;
            q[id].tag=0;
        }
    }
    void build(const vector<int>& a,int l,int r,int id=1){
        if(l==r){
            q[id].l=l,q[id].r=r;

            //init value
            q[id].mx=a[l],q[id].tag=0;

            return;
        }
        int mid=l+r>>1;
        build(a,l,mid,ls);
        build(a,mid+1,r,rs);
        q[id]=q[ls]+q[rs];
    }
public:
    SGT(const vector<int>& a,int n){
        init(a,n);
    }
    void init(const vector<int>& a,int n){
        q.resize(n*5);
        build(a,1,n);
```

```
    }
    void modify(int l,int r,int val,int id=1){
        if(q[id].l==l && q[id].r==r){
            //modify the value
            q[id].mx=max(q[id].mx,val),q[id].tag=1;
            return;
        }
        spread(id);
        int mid=q[id].l+q[id].r>>1;
        if(r<=mid) modify(l,r,val,ls);
        else if(l>mid) modify(l,r,val,rs);
        else modify(l,mid,val,ls),modify(mid+1,r,val,rs);
        q[id]=q[ls]+q[rs];
    }
    int query(int l,int r,int id=1){
        if(q[id].l==l && q[id].r==r) return q[id].mx;
        spread(id);
        int mid=q[id].l+q[id].r>>1;
        if(r<=mid) return query(l,r,ls);
        else if(l>mid) return query(l,r,rs);
        return max(query(l,mid,ls),query(mid+1,r,rs));
    }
    int get(int pos,int id=1){
        if(q[id].l==q[id].r) return q[id].mx;
        spread(id);
        int mid=q[id].l+q[id].r>>1;
        if(pos<=mid) return get(pos,ls);
        return get(pos,rs);
    }
};
#undef ls
#undef rs
```

主席树

可持久化权值线段树

```cpp
//P3834【模板】可持久化线段树 2
//区间静态查找第 k 小
const int inf=1e9;
class pSGT{//persistent SGT
private:
    struct Node{
        int ls,rs,cnt;
    };
    vector<Node>q;
public:
    vector<int>root;
    pSGT(){
        init();
    }
    void init(){
        q.clear(),root.clear();
        q.push_back(Node{});
        root.push_back(0);
    }
    void add_root(const int& x){
        root.push_back(x);
    }
    int insert(int id,int l,int r,int val){
        int new_id=q.size();
        q.push_back(q[id]);
        q[new_id].cnt++;
        if(l!=r){
            int mid=l+r>>1;
            if(val<=mid) q[new_id].ls=insert(q[id].ls,l,mid,val);
            else q[new_id].rs=insert(q[id].rs,mid+1,r,val);
        }
        return new_id;
    }
    int find(int idL,int idR,int l,int r,int k){
        if(l==r) return l;
        int mid=l+r>>1;
        int lcnt=q[q[idR].ls].cnt-q[q[idL].ls].cnt;
        if(k<=lcnt) return find(q[idL].ls,q[idR].ls,l,mid,k);
        else return find(q[idL].rs,q[idR].rs,mid+1,r,k-lcnt);
    }
};

int main(){
    int n=read(),T=read();
    pSGT sgt;
    for(int i=1;i<=n;i++){
        sgt.add_root(sgt.insert(sgt.root[i-1],0,inf,read()));
    }
    for(int l,r;T--;){
        l=read(),r=read();
        printf("%d\n",sgt.find(sgt.root[l-1],sgt.root[r],0,inf,read()));
    }
```

```
    return 0;
}
```

可持久化普通线段树

以CF893F为模板题

题目大意：给你一颗有根树，点有权值，m 次询问，每次问你某个点的子树中距离其不超过 k 的点的权值的最小值。（边权均为 1，点权有可能重复，k 值每次询问有可能不同）

做法：dfs 得到 dfn 和 dep。按照 dep 构建主席树，每次查询 $[dep[u],dep[u]+k]$ 中 $[dfn[u],dfn[u]+siz[u]-1]$ 的最小值，因为 $dep[u]$ 之前 $[dfn[u],dfn[u]+siz[u]-1]$ 不会被更新，所以直接在 $dep[u]+k$ 这个 $root$ 处查询即可

```
//CF893F
const int inf=1e9;
class pSGT{//persistent SGT
private:
    struct Node{
        int ls,rs,mn;
    };
    vector<Node>q;
public:
    vector<int>root;
    pSGT(){
        init();
    }
    void init(){
        q.clear(),root.clear();
        q.push_back(Node{0,0,inf});
        root.push_back(0);
    }
    void add_root(const int& x){
        root.push_back(x);
    }
    int insert(int id,int l,int r,int pos,int val){
        int new_id=q.size();
        q.push_back(q[id]);
        q[new_id].mn=min(q[id].mn,val);
        if(l!=r){
            int mid=l+r>>1;
            if(pos<=mid) q[new_id].ls=insert(q[id].ls,l,mid,pos,val);
            else q[new_id].rs=insert(q[id].rs,mid+1,r,pos,val);
        }
        return new_id;
    }
    //query [L,R]'s min, current Node id's range is [l,r]
    int find(int id,int L,int R,int l,int r){
        if(L==l && R==r) return q[id].mn;
        int mid=l+r>>1;
        if(R<=mid) return find(q[id].ls,L,R,l,mid);
        else if(L>mid) return find(q[id].rs,L,R,mid+1,r);
        else return min(find(q[id].ls,L,mid,l,mid),
find(q[id].rs,mid+1,R,mid+1,r));
    }
};

int main(){
    int n=read(),root=read();
    vector<int>a(n+1);
```

```
    for(int i=1;i<=n;i++) a[i]=read();
    vector<vector<int>>e(n+1);
    for(int i=1,u,v;i<n;i++){
        u=read(),v=read();
        e[u].push_back(v);
        e[v].push_back(u);
    }
    vector<int>dfn(n+1),dep(n+1),siz(n+1);
    int timStamp=0,max_dep=0;
    auto dfs=[&](auto self,int u,int fa)->void{
        dfn[u]=++timStamp,dep[u]=dep[fa]+1,siz[u]=1;
        max_dep=max(max_dep,dep[u]);
        for(int v:e[u]){
            if(v==fa) continue;
            self(self,v,u);
            siz[u]+=siz[v];
        }
    };
    dfs(dfs,root,0);
    vector<vector<int>>add(max_dep+1);
    for(int i=1;i<=n;i++){
        add[dep[i]].push_back(i);
    }
    vector<int>rt(max_dep+1);
    pSGT sgt;
    for(int i=1,tot=0;i<=max_dep;i++){
        for(int u:add[i]){
            sgt.add_root(sgt.insert(sgt.root[tot++],1,n,dfn[u],a[u]));
        }
        rt[i]=tot;
    }
    for(int T=read(),last=0,u,k;T--;){
        u=(read()+last)%n+1,k=(read()+last)%n;
        last=sgt.find(sgt.root[rt[min(max_dep,dep[u]+k)]],
dfn[u],dfn[u]+siz[u]-1,1,n);
        printf("%d\n",last);
    }
    return 0;
}
```

树链剖分（重链剖分）（LCA模板）

直接把P3384【模板】重链剖分/树链剖分的代码复制了一遍

因为原题有取模操作，抄模板的时候记得**删去取模，删去取模，删去取模！！！**

也可以把树链剖分作为 LCA 的模板来使用，比倍增的写法优秀很多

```
void heavy_path_decomposition(){
    int n=read(),T=read(),root=read();// mod=read();
    vector<int>a(n+1);
    for(int i=1;i<=n;i++) a[i]=read()%mod;//origin value
    vector<vector<int>>e(n+1);
    for(int i=1,u,v;i<n;i++){
        u=read(),v=read();
        e[u].push_back(v);
        e[v].push_back(u);
    }
    vector<int>dep(n+1),fa(n+1),siz(n+1),son(n+1);
    auto dfs1=[&](auto&& self,int u,int pre)->void{
        dep[u]=dep[pre]+1,fa[u]=pre,siz[u]=1;
        for(int v:e[u]){
            if(v==pre) continue;
            self(self,v,u);
            siz[u]+=siz[v];
            if(siz[v]>siz[son[u]]) son[u]=v;
        }
    };
    dfs1(dfs1,root,0);
    vector<int>id(n+1),nw(n+1),top(n+1);
    int timStamp=0;
    auto dfs2=[&](auto&& self,int u,int t)->void{
        id[u]=++timStamp,nw[timStamp]=u,top[u]=t;
        if(!son[u]) return;
        self(self,son[u],t);
        for(int v:e[u]){
            if(v==fa[u] || v==son[u]) continue;
            self(self,v,v);
        }
    };
    dfs2(dfs2,root,root);
    auto LCA=[&](int u,int v)->int{
        while(top[u]!=top[v]){
            if(dep[top[u]]<dep[top[v]]) swap(u,v);
            u=fa[top[u]];
        }
        return dep[u]<dep[v]?u:v;
    };
    vector<int>sgt_init(n+1);
    for(int i=1;i<=n;i++) sgt_init[i]=a[nw[i]];
    SGT sgt(sgt_init,n);//sgt needs to support seg add, seg query
    auto modify_path=[&](int u,int v,LL val)->void{
        while(top[u]!=top[v]){
            if(dep[top[u]]<dep[top[v]]) swap(u,v);
            sgt.modify(id[top[u]],id[u],val);
            u=fa[top[u]];
        }
```

```
        if(dep[u]<dep[v]) swap(u,v);
        sgt.modify(id[v],id[u],val);
    };
    auto query_path=[&](int u,int v)->LL{
        LL cnt=0ll;
        while(top[u]!=top[v]){
            if(dep[top[u]]<dep[top[v]]) swap(u,v);
            cnt+=sgt.query(id[top[u]],id[u]);
            u=fa[top[u]];
        }
        if(dep[u]<dep[v]) swap(u,v);
        cnt+=sgt.query(id[v],id[u]);
        return cnt%mod;
    };
    auto modify_tree=[&](int u,int val)->void{
        sgt.modify(id[u],id[u]+siz[u]-1,val);
    };
    auto query_tree=[&](int u)->LL{
        return sgt.query(id[u],id[u]+siz[u]-1);
    };
    for(int op,u,v,w;T--;){
        op=read(),u=read();
        if(op==1){
            v=read(),w=read();
            modify_path(u,v,w);
        }
        else if(op==2){
            v=read();
            printf("%lld\n",query_path(u,v));
        }
        else if(op==3){
            w=read();
            modify_tree(u,w);
        }
        else{//op==4
            printf("%lld\n",query_tree(u));
        }
    }
}
```

倍增法求 LCA

```
vector<int>dep(n+1);
const int mx_p=log2(n)+2;
vector lca(n+1,vector<int>(mx_p));
auto dfs_init=[&](auto&& self,int u,int fa)->void{
    dep[u]=dep[fa]+1,lca[u][0]=fa;
    for(int v:e[u]){
        if(v==fa) continue;
        self(self,v,u);
    }
};
dfs_init(dfs_init,root,0);
for(int p=1;p<mx_p;p++){
    for(int u=1;u<=n;u++){
        lca[u][p]=lca[lca[u][p-1]][p-1];
    }
}
auto LCA=[&](int u,int v)->int{
    if(dep[v]<dep[u]) swap(u,v);
    for(int p=mx_p-1;p>=0;p--){
        if(dep[lca[v][p]]>=dep[u]){
            v=lca[v][p];
        }
    }
    if(u==v) return u;
    for(int p=mx_p-1;p>=0;p--){
        if(lca[u][p]!=lca[v][p]){
            u=lca[u][p],
            v=lca[v][p];
        }
    }
    return lca[u][0];
};
```

平衡树

普通平衡树

非常朴素的 $fhqTreap$，用 $vector$ 实现了一下动态开点

```
template<typename T>
class fhqTreap{
private:
    struct Node{
        int l,r,siz; LL rnd;
        T val; LL sum;
        Node(){
            l=r=siz=0; rnd=0ll;
            val=T(0); sum=0ll;
        }
        Node(int l_,int r_,int siz_,LL rnd_,T val_,LL sum_){
            l=l_,r=r_,siz=siz_; rnd=rnd_;
            val=val_; sum=sum_;
        }
    };
    vector<Node>q;
    int root,rootX,rootY,rootZ;
    int New(T val){
        Node new_node=Node(0,0,1,rng(),val,val);
        q.push_back(new_node);
        return q.size()-1;
    }
    void Update(int id){
        q[id].siz=q[q[id].l].siz+q[q[id].r].siz+1;
        q[id].sum=1ll*q[q[id].l].val+q[q[id].r].val+q[id].val;
    }
    void Split(int id,T key,int& idX,int& idY){
        if(id==0){
            idX=idY=0;
            return;
        }
        if(q[id].val<=key){
            idX=id;
            Split(q[id].r,key,q[id].r,idY);
        }
        else{
            idY=id;
            Split(q[id].l,key,idX,q[id].l);
        }
        Update(id);
    }
    int Merge(int l,int r){
        if(l==0 || r==0) return l+r;
        if(q[l].rnd<=q[r].rnd){
            q[r].l=Merge(l,q[r].l);
            Update(r);
            return r;
        }
        else{
            q[l].r=Merge(q[l].r,r);
```

```
            Update(l);
            return l;
        }
    }
public:
    fhqTreap(){
        init();
    }
    void init(){
        root=0; q.clear();
        Node empty_node=Node();
        q.push_back(empty_node);
    }
    void insert(T val){
        Split(root,val,rootX,rootY);
        root=Merge(Merge(rootX,New(val)),rootY);
    }
    void erase(T val){//actually, 'extract' may be more precise
        Split(root,val,rootX,rootZ);
        Split(rootX,val-1,rootX,rootY);
        rootY=Merge(q[rootY].l,q[rootY].r);
        root=Merge(Merge(rootX,rootY),rootZ);
    }
    T prev(T val){
        Split(root,val-1,rootX,rootY);
        int tmp=rootX;
        while(q[tmp].r) tmp=q[tmp].r;
        root=Merge(rootX,rootY);
        return q[tmp].val;
    }
    T next(T val){
        Split(root,val,rootX,rootY);
        int tmp=rootY;
        while(q[tmp].l) tmp=q[tmp].l;
        root=Merge(rootX,rootY);
        return q[tmp].val;
    }
    int rank(T val){//val's rank
        Split(root,val-1,rootX,rootY);
        int ans=q[rootX].siz+1;
        root=Merge(rootX,rootY);
        return ans;
    }
    T get(int rank){
        int id=root;
        while(1){
//          printf("id:%d val:%d l:%d lsiz:%d r:%d
rsiz:%d|rank:%d\n",id,q[id].val,q[id].l,q[q[id].l].siz,q[id].r,q[q[id].r].s
iz,rank);
            if(q[q[id].l].siz>=rank) id=q[id].l;
            else if(q[q[id].l].siz+1==rank) return q[id].val;
            else{
                rank-=(q[q[id].l].siz+1);
                id=q[id].r;
            }
        }
    }
//  void output(){
```

```
107 //      auto dfs=[&](auto self,int u)->void{
108 //          if(!u) return;
109 //          self(self,q[u].l);
110 //          debug("%d|val=%d ls=%d rs=%d\n",u,q[u].val,q[u].l,q[u].r);
111 //          self(self,q[u].r);
112 //      };
113 //      debug("output fhqTreap\n");
114 //      dfs(dfs,root);
115 //      debug("\n");
116 //  }
117 };
```

文艺平衡树

依然坚定的使用无旋 $Treap$

```
template<typename T>
class fhqTreap{
private:
    struct Node{
        int l,r,siz; LL rnd;
        T val; bool lazy;
        Node(){
            l=r=siz=lazy=0;
            rnd=0ll; val=T(0);
        }
        Node(int l_,int r_,int siz_,LL rnd_,T val_){
            l=l_,r=r_,siz=siz_,lazy=0;
            rnd=rnd_; val=val_;
        }
    };
    vector<Node>q;
    int root,rootX,rootY,rootZ;
    int New(T val){
        Node new_node=Node(0,0,1,rng(),val);
        q.push_back(new_node);
        return q.size()-1;
    }
    void Update(int id){
        q[id].siz=q[q[id].l].siz+q[q[id].r].siz+1;
    }
    void Spread(int id){
        if(!id) return;
        if(q[id].lazy){
            q[q[id].l].lazy^=1;
            q[q[id].r].lazy^=1;
            swap(q[id].l,q[id].r);
            q[id].lazy=0;
        }
    }
    void Split(int id,T key,int& idX,int& idY){
        if(id==0){
            idX=idY=0;
            return;
        }
        Spread(id);
        if(q[q[id].l].siz+1<=key){
            idX=id;
            Split(q[id].r,key-q[q[id].l].siz-1,q[id].r,idY);
        }
        else{
            idY=id;
            Split(q[id].l,key,idX,q[id].l);
        }
        Update(id);
    }
    int Merge(int l,int r){
        if(l==0 || r==0) return l+r;
        if(q[l].rnd<=q[r].rnd){
```

```
            Spread(r);
            q[r].l=Merge(l,q[r].l);
            Update(r);
            return r;
        }
        else{
            Spread(l);
            q[l].r=Merge(q[l].r,r);
            Update(l);
            return l;
        }
    }
    void Print(vector<int>&a,int u){
        if(!u) return;
        Spread(u);
        Print(a,q[u].l);
        a.push_back(q[u].val);
        Print(a,q[u].r);
    }
public:
    fhqTreap(){
        init();
    }
    void init(){
        root=0; q.clear();
        Node empty_node=Node();
        q.push_back(empty_node);
    }
    void insert(int x){
        root=Merge(root,New(x));
    }
    void reverse(int l,int r){
        Split(root,l-1,rootX,rootZ);
        Split(rootZ,r-l+1,rootY,rootZ);
        q[rootY].lazy^=1;
        root=Merge(Merge(rootX,rootY),rootZ);
    }
    void to_ary(vector<int>&a){
        a.clear();
        Print(a,root);
    }
};
```

吉司机线段树

把 wiki 里的代码改了改

```
const int inf=INT_MAX;

#define ls (id<<1)
#define rs (id<<1|1)
#define SPREAD_ID spread(id,q[id].l,q[id].r)

#define MODIFY(FUNC) \
SPREAD_ID; \
int mid=q[id].l+q[id].r>>1; \
if(r<=mid) FUNC(l,r,val,ls); \
else if(l>mid) FUNC(l,r,val,rs); \
else FUNC(l,mid,val,ls),FUNC(mid+1,r,val,rs); \
q[id]=q[ls]+q[rs]

#define ADD(A,B) A+B
#define QUERY(FUNC,QRY_FUNC) \
SPREAD_ID;\
int mid=q[id].l+q[id].r>>1; \
if(r<=mid) return FUNC(l,r,ls); \
else if(l>mid) return FUNC(l,r,rs); \
else return QRY_FUNC(FUNC(l,mid,ls),FUNC(mid+1,r,rs))

class JiSGT{
private:
    struct Node{
        int l,r;
        //count_max, count_min, tag_max, tag_min, tag_add
        int mx,mx2,mn,mn2,cmx,cmn,tmx,tmn,tad;
        LL sum;
        friend Node operator +(const Node& A,const Node& B){
            Node ret;
            ret.l=A.l,ret.r=B.r;
            ret.sum = A.sum + B.sum;
            if (A.mx == B.mx) {
                ret.mx = A.mx, ret.cmx = A.cmx + B.cmx;
                ret.mx2 = max(A.mx2, B.mx2);
            } else if (A.mx > B.mx) {
                ret.mx = A.mx, ret.cmx = A.cmx;
                ret.mx2 = max(A.mx2, B.mx);
            } else {
                ret.mx = B.mx, ret.cmx = B.cmx;
                ret.mx2 = max(A.mx, B.mx2);
            }
            if (A.mn == B.mn) {
                ret.mn = A.mn, ret.cmn = A.cmn + B.cmn;
                ret.mn2 = min(A.mn2, B.mn2);
            } else if (A.mn < B.mn) {
                ret.mn = A.mn, ret.cmn = A.cmn;
                ret.mn2 = min(A.mn2, B.mn);
            } else {
                ret.mn = B.mn, ret.cmn = B.cmn;
                ret.mn2 = min(A.mn, B.mn2);
```

```
            }
            ret.tmx=-inf, ret.tmn=inf, ret.tad=0;
            return ret;
        }
    };
    vector<Node>q;
    void spread_add(int id, int l, int r, int v) {
        // 更新加法标记的同时，更新 $\min$ 和 $\max$ 标记
        q[id].sum += (r - l + 1ll) * v;
        q[id].mx += v, q[id].mn += v;
        if (q[id].mx2 != -inf) q[id].mx2 += v;
        if (q[id].mn2 != inf) q[id].mn2 += v;
        if (q[id].tmx != -inf) q[id].tmx += v;
        if (q[id].tmn != inf) q[id].tmn += v;
        q[id].tad += v;
    }
    void spread_min(int id, int tg) {
        // 注意比较 $\max$ 标记
        if (q[id].mx <= tg) return;
        q[id].sum += (tg * 1ll - q[id].mx) * q[id].cmx;
        if (q[id].mn2 == q[id].mx) q[id].mn2 = tg;  // !!!
        if (q[id].mn == q[id].mx) q[id].mn = tg;    // !!!!!
        if (q[id].tmx > tg) q[id].tmx = tg;         // 更新取 $\max$ 标记
        q[id].mx = tg, q[id].tmn = tg;
    }
    void spread_max(int id, int tg) {
        if (q[id].mn > tg) return;
        q[id].sum += (tg * 1ll - q[id].mn) * q[id].cmn;
        if (q[id].mx2 == q[id].mn) q[id].mx2 = tg;
        if (q[id].mx == q[id].mn) q[id].mx = tg;
        if (q[id].tmn < tg) q[id].tmn = tg;
        q[id].mn = tg, q[id].tmx = tg;
    }
    void spread(int id,int l,int r){
        if(q[id].l==q[id].r) return;
        int mid=l+r>>1;
        if (q[id].tad){
            spread_add(ls, l, mid, q[id].tad);
            spread_add(rs, mid + 1, r, q[id].tad);
        }
        if (q[id].tmx != -inf){
            spread_max(ls, q[id].tmx);
            spread_max(rs, q[id].tmx);
        }
        if (q[id].tmn != inf){
            spread_min(ls, q[id].tmn);
            spread_min(rs, q[id].tmn);
        }
        q[id].tmx = -inf, q[id].tmn = inf, q[id].tad = 0;
    }
    void build(const vector<int>& a,int l,int r,int id=1){
        if(l==r){
            q[id].l=l,q[id].r=r;
            q[id].sum=q[id].mx = q[id].mn = a[l];
            q[id].mx2=-inf, q[id].mn2 = inf;
            q[id].cmx=q[id].cmn = 1;
            q[id].tmx=-inf, q[id].tmn = inf, q[id].tad=0;
            return;
```

```
        }
        int mid=l+r>>1;
        build(a,l,mid,ls);
        build(a,mid+1,r,rs);
        q[id]=q[ls]+q[rs];
    }
public:
    JiSGT(const vector<int>& a,int n){
        init(a,n);
    }
    void init(const vector<int>& a,int n){
        q.resize(n*4);
        build(a,1,n);
    }
    void add(int l,int r,int val,int id=1){
        if(q[id].l==l && q[id].r==r){
            return spread_add(id,l,r,val);
        }
        MODIFY(add);
    }
    void tomin(int l,int r,int val,int id=1){
        if(r<q[id].l || l>q[id].r || q[id].mx<=val) return;
        if(l<=q[id].l && q[id].r<=r && q[id].mx2<val){
            return spread_min(id,val);
        }
        MODIFY(tomin);
    }
    void tomax(int l,int r,int val,int id=1){
        if(r<q[id].l || l>q[id].r || q[id].mn>=val) return;
        if(l<=q[id].l && q[id].r<=r && q[id].mn2>val){
            return spread_max(id,val);
        }
        MODIFY(tomax);
    }
    LL qsum(int l,int r,int id=1){
        if(q[id].l==l && q[id].r==r) return q[id].sum;
        QUERY(qsum,ADD);
    }
    LL qmax(int l,int r,int id=1){
        if(q[id].l==l && q[id].r==r) return q[id].mx;
        QUERY(qmax,max);
    }
    LL qmin(int l,int r,int id=1){
        if(q[id].l==l && q[id].r==r) return q[id].mn;
        QUERY(qmin,min);
    }
};
#undef ls
#undef rs
#undef SPREAD_ID
#undef MODIFY
#undef ADD
#undef QUERY
```

虚树

模板题：CF613D，核心代码（二次排序法）如下：

```
vector<int>adj[N]; //adj存虚树
int flag[N]; //flag判断是否是关键点

void solve(int col){
    int m=read();
    vector<int>a(m),b;
    for(int i=0;i<m;i++) flag[a[i]=read()]=col;
    if(flag[1]!=col) a.push_back(1),m++; //把根节点1放进虚树内，不会影响答案且减少了特判
    sort(a.begin(),a.end(),cmp); //a中存关键点，按照dfn排序
    for(int i=0;i<m-1;i++){
        b.push_back(a[i]);
        b.push_back(LCA(a[i],a[i+1])); //插入关键点之间的lca
    }
    b.push_back(a.back()); //别忘了把最后一个点放进去
    sort(b.begin(),b.end(),cmp); //把所有虚树上的点按照dfn排序
    b.erase(unique(b.begin(),b.end()),b.end()); //去重
    for(int u:b) adj[u].clear(); //清空上一次的虚树
    for(int i=0;i<b.size()-1;i++){
        adj[LCA(b[i],b[i+1])].push_back(b[i+1]); //虚树连边
    }
    printf("%d\n",calc(1,col).first); //在虚树上处理出答案
}
```

图论

Tarjan

```cpp
namespace Tarjan{
    vector<int>dfn(n+1),low(n+1),inStack(n+1),scc(n+1),siz(n+1);
    int timStamp=0,col=0;
    stack<int>stk;
    auto tarjan=[&](auto self,int u)->void{
        low[u]=dfn[u]=++timStamp;
        stk.push(u),inStack[u]=1;
        for(int v:e[u]){
            if(!dfn[v]){
                self(self,v);
                low[u]=min(low[u],low[v]);
            }
            else if(inStack[v]){
                low[u]=min(low[u],low[v]);
            }
        }
        if(dfn[u]==low[u]){
            ++col;
            while(stk.top()!=u){
                siz[scc[stk.top()]=col]++;
                inStack[stk.top()]=0; stk.pop();
            }
            siz[scc[stk.top()]=col]++;
            inStack[stk.top()]=0; stk.pop();
        }
    };
}
```

字符串

字符串哈希（多重哈希）

修改 HL（$hash\_layer$） 以决定使用几重哈希

$init$ 时，传入字符数组首地址，字符串长度（以及是否倒着求哈希，0和缺省为正着求，1为倒着求）

注意字符数组内数据的下标是 $1\sim n$，$1\sim n$，$1\sim n$！！！

```
const int HL=2;//hash layer
namespace HC{//Hash Const
    const int P[4]={13331,233,131,19260817};
    const int MOD[4]={(int)(1e9+7),998244353,1004535809,754974721};
    LL ksm[N][4];
    void init(int use=HL){
        for(int j=0;j<use;j++){
            ksm[0][j]=1;
            for(int i=1;i<N;i++){
                ksm[i][j]=(ksm[i-1][j]*P[j])%MOD[j];
            }
        }
    }
}
template<int T>//T must be a constant
class Hash{
private:
    vector<array<LL,T>>h;
    bool sign;//0:normal 1:reverse
public:
    Hash(char *s,int n,bool sign_=0){
        init(s,n,sign_);
    }
    void init(char *s,int n,bool sign_=0){//s stores at pos [1,n], and just give para s (NOT s+1)
        h.resize(n+2);
        sign=sign_;
        if(!sign){
            for(int j=0;j<T;j++){
                h[0][j]=0;
                for(int i=1;i<=n;i++){
                    h[i][j]=(h[i-1][j]*HC::P[j]+s[i]-'a'+1)%HC::MOD[j];
                }
            }
        }
        else{
            for(int j=0;j<T;j++){
                h[n+1][j]=0;
                for(int i=n;i>0;i--){
                    h[i][j]=(h[i+1][j]*HC::P[j]+s[i]-'a'+1)%HC::MOD[j];
                }
            }
        }
    }
    array<LL,T>calc(const int& l,const int& r){
        array<LL,T>ret;
        if(!sign){
```

```
            for(int j=0;j<T;j++){
                ret[j]=h[r][j]-h[l-1][j]*HC::ksm[r-l+1][j];
            }
        }
        else{
            for(int j=0;j<T;j++){
                ret[j]=h[l][j]-h[r+1][j]*HC::ksm[r-l+1][j];
            }
        }
        for(int j=0;j<T;j++){
            ret[j]=(ret[j]%HC::MOD[j]+HC::MOD[j])%HC::MOD[j];
        }
        return ret;
    }
    static bool check(const array<LL,T>& a,const array<LL,T>& b){
        for(int i=0;i<T;i++){
            if(a[i]!=b[i]) return 0;
        }
        return 1;
    }
};
```

Z函数（扩展KMP）

约定：字符串下标以 0 为起点

定义函数 $z[i]$ 表示 s 和 $s[i,n-1]$（即以 $s[i]$ 开头的后缀）的最长公共前缀（LCP）的长度，则 z 被成为 s 的 **Z函数**。特别的，$z[0]=0$

```
vector<int>z_function(string s){
    int n=(int)s.length();
    vector<int>z(n);
    for (int i=1,l=0,r=0;i<n;i++){
        if(i<=r&&z[i-l]<r-i+1){
            z[i]=z[i-l];
        }
        else{
            z[i]=max(0,r-i+1);
            while(i+z[i]<n && s[z[i]]==s[i+z[i]]) z[i]++;
        }
        if(i+z[i]-1>r) l=i,r=i+z[i]-1;
    }
    return z;
}
```

数学

高精度

BigIntTiny Template Link，顺便从大佬的 README 里放个 Features preview，不支持的功能就现写吧。

operators	BigIntHex	BigIntDec	BigIntMini	BigIntTiny
constructor Bigint	✓	✓	✓	✓
constructor int	✓	✓	✓	✓
constructor char*	✓	✓	✕	✕
constructor string	✓	✓	✓	✓
=Bigint	✓	✓	✓	✓
=int	✓	✓	✓	✓
=string	✓	✓	✓	✓
=char*	✓	✓	✓	✕
<, ==, >, <=, >=, != Bigint	✓	✓	✓	✓
+, -, *, /, % int	✕	✕	✕	✓
+=, -=, *=, /=, %= int	✕	✕	✕	✕
+, -, *, /, % Bigint	✓	✓	✓	✓
+=, -=, *=, /=, %= Bigint	✓	✓	✕	✕
Base conversion	✓	✓	✕	✕
Efficiency	★★★★	★★★	★★	★

```
struct BigIntTiny {
    int sign;
    std::vector<int> v;

    BigIntTiny() : sign(1) {}
    BigIntTiny(const std::string &s) { *this = s; }
    BigIntTiny(int v) {
        char buf[21];
        sprintf(buf, "%d", v);
        *this = buf;
    }
    void zip(int unzip) {
        if (unzip == 0) {
            for (int i = 0; i < (int)v.size(); i++)
                v[i] = get_pos(i * 4) + get_pos(i * 4 + 1) * 10 + get_pos(i * 4 + 2) * 100 + get_pos(i * 4 + 3) * 1000;
        } else
            for (int i = (v.resize(v.size() * 4), (int)v.size() - 1), a; i >= 0; i--)
```

```
                a = (i % 4 >= 2) ? v[i / 4] / 100 : v[i / 4] % 100, v[i] =
(i & 1) ? a / 10 : a % 10;
        setsign(1, 1);
    }
    int get_pos(unsigned pos) const { return pos >= v.size() ? 0 : v[pos];
}
    BigIntTiny &setsign(int newsign, int rev) {
        for (int i = (int)v.size() - 1; i > 0 && v[i] == 0; i--)
            v.erase(v.begin() + i);
        sign = (v.size() == 0 || (v.size() == 1 && v[0] == 0)) ? 1 : (rev ?
newsign * sign : newsign);
        return *this;
    }
    std::string to_str() const {
        BigIntTiny b = *this;
        std::string s;
        for (int i = (b.zip(1), 0); i < (int)b.v.size(); ++i)
            s += char(*(b.v.rbegin() + i) + '0');
        return (sign < 0 ? "-" : "") + (s.empty() ? std::string("0") : s);
    }
    bool absless(const BigIntTiny &b) const {
        if (v.size() != b.v.size()) return v.size() < b.v.size();
        for (int i = (int)v.size() - 1; i >= 0; i--)
            if (v[i] != b.v[i]) return v[i] < b.v[i];
        return false;
    }
    BigIntTiny operator-() const {
        BigIntTiny c = *this;
        c.sign = (v.size() > 1 || v[0]) ? -c.sign : 1;
        return c;
    }
    BigIntTiny &operator=(const std::string &s) {
        if (s[0] == '-')
            *this = s.substr(1);
        else {
            for (int i = (v.clear(), 0); i < (int)s.size(); ++i)
                v.push_back(*(s.rbegin() + i) - '0');
            zip(0);
        }
        return setsign(s[0] == '-' ? -1 : 1, sign = 1);
    }
    bool operator<(const BigIntTiny &b) const {
        return sign != b.sign ? sign < b.sign : (sign == 1 ? absless(b) :
b.absless(*this));
    }
    bool operator==(const BigIntTiny &b) const { return v == b.v && sign ==
b.sign; }
    BigIntTiny &operator+=(const BigIntTiny &b) {
        if (sign != b.sign) return *this = (*this) - -b;
        v.resize(std::max(v.size(), b.v.size()) + 1);
        for (int i = 0, carry = 0; i < (int)b.v.size() || carry; i++) {
            carry += v[i] + b.get_pos(i);
            v[i] = carry % 10000, carry /= 10000;
        }
        return setsign(sign, 0);
    }
    BigIntTiny operator+(const BigIntTiny &b) const {
        BigIntTiny c = *this;
```

```
        return c += b;
    }
    void add_mul(const BigIntTiny &b, int mul) {
        v.resize(std::max(v.size(), b.v.size()) + 2);
        for (int i = 0, carry = 0; i < (int)b.v.size() || carry; i++) {
            carry += v[i] + b.get_pos(i) * mul;
            v[i] = carry % 10000, carry /= 10000;
        }
    }
    BigIntTiny operator-(const BigIntTiny &b) const {
        if (b.v.empty() || b.v.size() == 1 && b.v[0] == 0) return *this;
        if (sign != b.sign) return (*this) + -b;
        if (absless(b)) return -(b - *this);
        BigIntTiny c;
        for (int i = 0, borrow = 0; i < (int)v.size(); i++) {
            borrow += v[i] - b.get_pos(i);
            c.v.push_back(borrow);
            c.v.back() -= 10000 * (borrow >>= 31);
        }
        return c.setsign(sign, 0);
    }
    BigIntTiny operator*(const BigIntTiny &b) const {
        if (b < *this) return b * *this;
        BigIntTiny c, d = b;
        for (int i = 0; i < (int)v.size(); i++, d.v.insert(d.v.begin(), 0))
            c.add_mul(d, v[i]);
        return c.setsign(sign * b.sign, 0);
    }
    BigIntTiny operator/(const BigIntTiny &b) const {
        BigIntTiny c, d;
        BigIntTiny e=b;
        e.sign=1;

        d.v.resize(v.size());
        double db = 1.0 / (b.v.back() + (b.get_pos((unsigned)b.v.size() - 2) / 1e4) +
                           (b.get_pos((unsigned)b.v.size() - 3) + 1) / 1e8);
        for (int i = (int)v.size() - 1; i >= 0; i--) {
            c.v.insert(c.v.begin(), v[i]);
            int m = (int)((c.get_pos((int)e.v.size()) * 10000 + c.get_pos((int)e.v.size() - 1)) * db);
            c = c - e * m, c.setsign(c.sign, 0), d.v[i] += m;
            while (!(c < e))
                c = c - e, d.v[i] += 1;
        }
        return d.setsign(sign * b.sign, 0);
    }
    BigIntTiny operator%(const BigIntTiny &b) const { return *this - *this / b * b; }
    bool operator>(const BigIntTiny &b) const { return b < *this; }
    bool operator<=(const BigIntTiny &b) const { return !(b < *this); }
    bool operator>=(const BigIntTiny &b) const { return !(*this < b); }
    bool operator!=(const BigIntTiny &b) const { return !(*this == b); }
};
```

快速幂&矩阵乘法

矩阵乘法中，f 为答案矩阵，是一维长为 n 的 $vector$；a 为转移矩阵，是二维 $n\cdot n$ 的 $vector$

传参数时，直接传入当前的 f 与 a 即可

```
LL ksm(LL a,LL b){
    a%=mod; LL ret=1;
    while(b){
        if(b&1) ret=ret*a%mod;
        a=a*a%mod;
        b>>=1;
    }
    return ret;
}

void mul(const vector<vector<LL>>& a,vector<LL>& f,int n){//f=a*f
    vector<LL>b(n,0);
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            b[i]=(b[i]+a[i][j]*f[j]%mod)%mod;
        }
    }
    swap(f,b);
}
void self_mul(vector<vector<LL>>& a,int n){//a=a*a
    vector b(n,vector<LL>(n,0));
    for(int k=0;k<n;k++){
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                b[i][j]=(b[i][j]+a[i][k]*a[k][j]%mod)%mod;
            }
        }
    }
    swap(a,b);
}
```

扩展欧几里得

最终会给出一组满足 $a \cdot x + b \cdot y = g$ 的解

```
//迭代法实现
int exgcd(int a, int b, int &x, int &y) {
  if (!b) {
    x = 1;
    y = 0;
    return a;
  }
  int d = Exgcd(b, a % b, x, y);
  int t = x;
  x = y;
  y = t - (a / b) * y;
  return d;
}

//非迭代法实现
int exgcd(int a, int b, int& x, int& y) {
  x = 1, y = 0;
  int x1 = 0, y1 = 1, a1 = a, b1 = b;
  while (b1) {
    int q = a1 / b1;
    tie(x, x1) = make_tuple(x1, x - q * x1);
    tie(y, y1) = make_tuple(y1, y - q * y1);
    tie(a1, b1) = make_tuple(b1, a1 - q * b1);
  }
  return a1;
}
```

线性筛欧拉函数

```
bool notP[N]={}; int phi[N],mn_p[N]={1};
vector<int>prime;
void init_prime(const int& n=N){//init [1,n-1]
    phi[1]=1;
    for(int i=2;i<n;i++){
        if(!notP[i]){
            prime.push_back(mn_p[i]=i);
            phi[i]=i-1;
        }
        for(const int& p:prime){
            if(i>(n-1)/p) break;
            notP[i*p]=1,mn_p[i*p]=p;
            if(i%p==0){
                phi[i*p]=phi[i]*p;
                break;
            }
            phi[i*p]=phi[i]*(p-1);
        }
    }
}
```

组合数学

阶乘相关的初始化

这个给模板只用来充数的qwq

```
namespace Fac{//factorial
    LL fc[N]={1},fc_inv[N]={1};
    void init(){
        for(int i=1;i<N;i++){
            fc[i]=fc[i-1]*i%mod;
            fc_inv[i]=fc_inv[i-1]*ksm(i,mod-2)%mod;
        }
    }
    LL F(const int& x){
        return fc[x];
    }
    LL P(const int& x,const int& y){
        return fc[x]*fc_inv[x-y]%mod;
    }
    LL C(const int& x,const int& y){
        return fc[x]*fc_inv[y]%mod*fc_inv[x-y]%mod;
    }
}
```

错位排列

- 递推公式（两种任选即可）

$$D_n = (n-1)\cdot(D_{n-1}+D_{n-2})$$
$$D_n = n\cdot D_{n-1}+(-1)^n$$

其中D 的前几项是 $D_1=0, D_2=1, D_3=9, D_4=44, D_5=265$（OEIS A000166）

- 其他关系

错位排列数有一个向下取整的简单表达式，增长速度与阶乘仅相差常数：

$$D_n = \left\lfloor \frac{n!}{e} \right\rfloor$$

随着元素数量的增加，形成错位排列的概率 P 接近：

$$P = \lim_{n\to\infty}\frac{D_n}{n!} = \frac{1}{e}$$

卡特兰数

$H_n =$ 进栈序列为 $1,2,3,\cdots,n$ 的栈的出栈序列个数，以下是一些常见公式

$$H_n = \frac{\binom{2n}{n}}{n+1}(n\geq 2, n\in\mathbb{N}_+)$$
$$H_n = \begin{cases}\sum_{i=1}^{n} H_{i-1}H_{n-i} & n\geq 2, n\in\mathbb{N}_+\\ 1 & n=0,1\end{cases}$$
$$H_n = \frac{H_{n-1}(4n-2)}{n+1}$$
$$H_n = \binom{2n}{n} - \binom{2n}{n-1}$$

斯特林数

第二类斯特林数

$\left\{ {n \atop k} \right\}$，$S(n,k)$，表示将 n 个两两不同的元素，划分为 k 个互不区分的非空子集的方案数

- 递推公式

$$\left\{ {n \atop m} \right\} = \left\{ {n-1 \atop k-1} \right\} + k\left\{ {n-1 \atop k} \right\}$$

边界是 $\left\{ {n \atop 0} \right\} = [n=0]$

- 通项公式

$$\left\{ {n \atop m} \right\} = \sum_{i=0}^{m} \frac{(-1)^{m-i} \cdot i^n}{i! \cdot (m-i)!}$$

待补充：同一行/列的第二类斯特林数的计算

第一类斯特林数

$\left[{n \atop k} \right]$，$s(n,k)$，表示将 n 个两两不同的元素，划分为 k 个互不区分的非空轮换的方案数

一个轮换即一个首尾相接的环形排列，如 $[A,B,C,D]=[B,C,D,A]=[C,D,A,B]=[D,A,B,C]$，即两个可以通过旋转而相互得到的轮换是等价的。但是，翻转不算旋转，如 $[A,B,C,D] \neq [D,C,B,A]$

- 递推公式

$$\left[{n \atop m} \right] = \left[{n-1 \atop k-1} \right] + (n-1)\left[{n-1 \atop k} \right]$$

边界是 $\left[{n \atop 0} \right] = [n=0]$

- 通项公式

没有

待补充：同一行/列的第一类斯特林数的计算

计算几何

计算几何基础

知识点

摘自繁凡的博客和 yxc 的讲义

1. 前置知识点
    1. $\pi = arccos(-1)$
    2. 余弦定理 $c^2 = a^2 + b^2 - 2abcos\theta$
2. 浮点数的比较

```
const double eps=1e-8; //精度，可按需要增加至1e-12之类的
int sign(double x){ //符号函数
    if(fabs(x)<eps) return 0;
    if(x<0) return -1;
    return 1;
}
int cmp(double x,double y){ //比较函数
    if(fabs(x-y)<eps) return 0;
    if(x<y) return -1;
    return 1;
}
```

3. 向量
    1. 点与向量的保存，以及运算符重载

    ```
    using PDD=pair<double,double>;
    #define x first
    #define y second

    PDD operator +(PDD a,PDD b){return PDD{a.x+b.x,a.y+b.y};}
    PDD operator -(PDD a,PDD b){return PDD{a.x-b.x,a.y-b.y};}
    PDD operator *(PDD a,double k){return PDD{k*a.x,k*a.y};}
    PDD operator *(double k,PDD a){return PDD{k*a.x,k*a.y};}
    ```

    2. 内积（点积）$A \cdot B = |A||B|cos\theta$
        1. 几何意义：向量 A 在向量 B 上的投影与 B 的长度的乘积
        2. 代码实现

        ```
        double dot(PDD a,PDD b){
            return a.x*b.x+a.y*b.y;
        }
        ```

    3. 外积（叉积）$A \times B = |A||B| \sin\theta$
        1. 几何意义：向量 A 与 B 张成的平行四边形的有向面积。B 在 A 的逆时针方向为正
        2. 代码实现

        ```
        double cross(PDD a,PDD b){
            return a.x*b.y-b.x*a.y;
        }
        ```

4. 常用函数

1. 取模

```
double get_length(PDD a){
    return sqrt(dot(a,a));
}
```

2. 计算向量夹角

```
double get_angle(PDD a,PDD b){
    return acos(dot(a,b)/get_length(a)/get_length(b));
}
```

3. 计算两个向量构成的平行四边形有向面积

```
double area(PDD a,PDD b,PDD c){
    return cross(b-a,c-a);
}
```

4. 向量 A **顺时针**旋转 $angle$（弧度制）的角度：

```
PDD rotate(PDD a,double angle){
    return PDD{ a.x*cos(angle) + a.y*sin(angle),
                -a.x*sin(angle) + a.y*cos(angle) };
}
```

4. 点与线

1. 直线定理

1. 一般式 $ax+by+c=0$
2. 点向式 p_0+vt
3. 斜截式 $y=kx+b$

2. 常用操作

(1) 判断点在直线上 $A\times B=0$

(2) 两直线相交（取直线 $p+vt$, $q+wt$ 的交点）

```
//cross(v,w)==0 则两直线平行或者重合，注意特判，这里没加
PDD get_line_inter(PDD p, PDD v, PDD q, PDD w){
    PDD u = p - q;
    double t = cross(w, u) / cross(v, w);
    return p + v * t;
}
```

(3) 点到直线的距离

```
//点p; 直线由a, b两点表示
double dis2line(PDD p, PDD a, PDD b){
    PDD v1 = b - a, v2 = p - a;
    return fabs(cross(v1, v2) / get_length(v1));
}
```

(4) 点到线段的距离

```
double dis2seg(PDD p, PDD a, PDD b){
    if (a == b) return get_length(p - a);
    PDD v1 = b - a, v2 = p - a, v3 = p - b;
    if (sign(dot(v1, v2)) < 0) return get_length(v2);
    if (sign(dot(v1, v3)) > 0) return get_length(v3);
    return dis2line(p, a, b);
}
```

(5) 点在直线上的投影

```
PDD get_line_proj(PDD p, PDD a, PDD b){
    PDD v = b - a;
    return a + v * (dot(v, p - a) / dot(v, v));
}
```

(6) 点是否在线段上

```
bool on_seg(PDD p, PDD a, PDD b){
    return sign(cross(p - a, p - b)) == 0 && sign(dot(p - a, p - b))
<= 0;
}
```

(7) 判断两线段是否相交

```
bool seg_inter(PDD a1, PDD a2, PDD b1, PDD b2){
    double c1 = cross(a2-a1, b1-a1), c2 = cross(a2-a1, b2-a1);
    double c3 = cross(b2-b1, a1-b1), c4 = cross(b2-b1, a2-b1);
    // 有if则允许线段在端点处相交，无if则不允许，根据需要添加
    if(!sign(c1) && !sign(c2) && !sign(c3) && !sign(c4)){
        bool f1 = on_seg(b1, a1, a2);
        bool f2 = on_seg(b2, a1, a2);
        bool f3 = on_seg(a1, b1, b2);
        bool f4 = on_seg(a2, b1, b2);
        bool f = (f1|f2|f3|f4);
        return f;
    }
    return (sign(c1)*sign(c2) < 0 && sign(c3)*sign(c4) < 0);
}
```

5. 多边形
    1. 三角形
        1. 面积
            1. 叉积
            2. 海伦公式

$$p = \frac{a+b+c}{2}$$

$$S = \sqrt{p(p-a)\cdot(p-b)\cdot(p-c)}$$

        2. 三角形四心
            1. 外心，外接圆圆心
               三边中垂线交点。到三角形三个顶点的距离相等

2. 内心，内切圆圆心
   角平分线交点，到三边距离相等
3. 垂心
   三条垂线交点
4. 重心
   三条中线交点（到三角形三顶点距离的平方和最小的点，三角形内到三边距离之积最大的点）

2. 普通多边形，通常按逆时针存储所有点

   1. 定义

      1. 多边形
         由在同一平面且不再同一直线上的多条线段首尾顺次连接且不相交所组成的图形叫多边形
      2. 简单多边形
         简单多边形是除相邻边外其它边不相交的多边形
      3. 凸多边形
         过多边形的任意一边做一条直线，如果其他各个顶点都在这条直线的同侧，则把这个多边形叫做凸多边形
         任意凸多边形外角和均为 360°
         任意凸多边形内角和为 $(n-2)180^\circ$

   2. 常用函数

      1. 求多边形面积（不一定是凸多边形）
         我们可以从第一个顶点出发把凸多边形分成n－2个三角形，然后把面积加起来。

```cpp
double polygon_area(vector<PDD>p){
    double s = 0;
    for (int i = 1; i + 1 < p.size(); i ++ )
        s += cross(p[i] - p[0], p[i + 1] - p[i]);
    return s / 2;
}
```

      2. 判断点是否在多边形内（不一定是凸多边形）
         a. 射线法，从该点任意做一条和所有边都不平行的射线。交点个数为偶数，则在多边形外，为奇数，则在多边形内
         b. 转角法

      3. 判断点是否在凸多边形内
         只需判断点是否在所有边的左边（逆时针存储多边形）

3. 皮克定理
   皮克定理是指一个计算点阵中，顶点在格点上的多边形面积的公式:

$$S = a + \frac{b}{2} - 1$$

   其中 a 表示多边形内部的点数，b 表示多边形边界上的点数，S 表示多边形的面积。

6. 圆

   1. 圆与直线交点
   2. 两圆交点
   3. 点到圆的切线
   4. 两圆公切线
   5. 两圆相交面积

完整代码

```
const double eps=1e-8; //精度，可按需要增加至1e-12之类的
int sign(double x){ //符号函数
    if(fabs(x)<eps) return 0;
    if(x<0) return -1;
    return 1;
}
int cmp(double x,double y){ //比较函数
    if(fabs(x-y)<eps) return 0;
    if(x<y) return -1;
    return 1;
}

using PDD=pair<double,double>;
#define x first
#define y second

//基本运算符重载
PDD operator +(PDD a,PDD b){return PDD{a.x+b.x,a.y+b.y};}
PDD operator -(PDD a,PDD b){return PDD{a.x-b.x,a.y-b.y};}
PDD operator *(double k,PDD a){return PDD{k*a.x,k*a.y};}
PDD operator *(PDD a,double k){return PDD{k*a.x,k*a.y};}
PDD operator /(PDD a,double k){return PDD{a.x/k,a.y/k};}

double dot(PDD a,PDD b){ //内积
    return a.x*b.x+a.y*b.y;
}

double cross(PDD a,PDD b){ //叉积
    return a.x*b.y-b.x*a.y;
}

//取模（长度）
double get_length(PDD a){
    return sqrt(dot(a,a));
}

//计算向量夹角
double get_angle(PDD a,PDD b){
    return acos(dot(a,b)/get_length(a)/get_length(b));
}

//计算两个向量构成的平行四边形的面积
double area(PDD a,PDD b,PDD c){
    return cross(b-a,c-a);
}

//A绕原点**顺时针**旋转angle（弧度制）
PDD rotate(PDD a,double angle){
    return PDD{a.x*cos(angle) + a.y*sin(angle),
        -a.x*sin(angle) + a.y*cos(angle)};
}

//取直线p+vt, q+wt的交点
//cross(v,w)==0 则两直线平行或者重合，注意特判，这里没加特判
```

```cpp
PDD get_line_inter(PDD p, PDD v, PDD q, PDD w){
    PDD u = p - q;
    double t = cross(w, u) / cross(v, w);
    return p + v * t;
}

//点p; 直线由a, b两点表示
//点到直线的距离
double dis2line(PDD p, PDD a, PDD b){
    PDD v1 = b - a, v2 = p - a;
    return fabs(cross(v1, v2) / get_length(v1));
}

//点到线段的距离
double dis2seg(PDD p, PDD a, PDD b){
    if (a == b) return get_length(p - a);
    PDD v1 = b - a, v2 = p - a, v3 = p - b;
    if (sign(dot(v1, v2)) < 0) return get_length(v2);
    if (sign(dot(v1, v3)) > 0) return get_length(v3);
    return dis2line(p, a, b);
}

//点在直线上的投影
PDD get_line_proj(PDD p, PDD a, PDD b){
    PDD v = b - a;
    return a + v * (dot(v, p - a) / dot(v, v));
}

//点是否在线段上
bool on_seg(PDD p, PDD a, PDD b){
    return sign(cross(p - a, p - b)) == 0 && sign(dot(p - a, p - b)) <= 0;
}

//判断两线段是否相交
bool seg_inter(PDD a1, PDD a2, PDD b1, PDD b2){
    double c1 = cross(a2-a1, b1-a1), c2 = cross(a2-a1, b2-a1);
    double c3 = cross(b2-b1, a1-b1), c4 = cross(b2-b1, a2-b1);
    // 有if则允许线段在端点处相交，无if则不允许，根据需要添加
    if(!sign(c1) && !sign(c2) && !sign(c3) && !sign(c4)){
        bool f1 = on_seg(b1, a1, a2);
        bool f2 = on_seg(b2, a1, a2);
        bool f3 = on_seg(a1, b1, b2);
        bool f4 = on_seg(a2, b1, b2);
        bool f = (f1|f2|f3|f4);
        return f;
    }
    return (sign(c1)*sign(c2) < 0 && sign(c3)*sign(c4) < 0);
}


//计算**任意**多边形面积（不一定凸）
double polygon_area(vector<PDD>p){
    double s = 0;
    for (int i = 1; i + 1 < p.size(); i ++ )
        s += cross(p[i] - p[0], p[i + 1] - p[i]);
    return s / 2;
}
```

凸包

代码1（普通凸包）（by fhy）

```
using p_t = int;
using p_t2 = long long;
struct P{
    mutable p_t x, y;
    P operator - (P b)const{
        return {x - b.x, y - b.y};
    }
    p_t2 operator ^ (P b)const{
        return (p_t2)x * b.y - (p_t2)y * b.x;
    }
    bool operator < (P b)const{
        return x < b.x;
    }
    bool operator < (p_t b)const{
        return x < b;
    }
};
int sgn(p_t2 x){
    return (x > 0) - (x < 0);
}
struct HULL:set<P, less<> >{//上凸包
    int out(P v){
        auto it = lower_bound(v.x);
        if(it == end()) return 1;
        if(it->x == v.x) return sgn(v.y - it->y);
        if(it == begin()) return 1;
        auto p = prev(it);
        return sgn(*p - v ^ *it - v);
    }
    void add(P v){
        if(out(v) <= 0) return;
        auto it = find(v);
        if(it != end()) it->y = v.y;
        else it = insert(v).first;
        auto p = next(it);
        if(p != end()){
            auto nxt = next(p);
            while(nxt != end()){
                if(sgn(*it - *p ^ *nxt - *p) > 0) break;
                p = erase(p); nxt = next(p);
            }
        }
        if(it != begin()){
            p = prev(it);
            while(p != begin()){
                auto pre = prev(p);
                if(sgn(*pre - *p ^ *it - *p) > 0) break;
                erase(p); p = prev(it);
            }
        }
    }
}hull[2];
```

```cpp
int main(){
    int q;
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin >> q;
    while(q--){
        int op, x, y;
        cin >> op >> x >> y;
        if(op == 1) hull[0].add({x, y}), hull[1].add({x, -y});
        else printf(hull[0].out({x, y}) <= 0 && hull[1].out({x, -y}) <= 0 ? "YES\n" : "NO\n");
    }
    return 0;
}
```

代码2（点全是整数） （by fhy）

```cpp
using line_t = long long;
struct Line{
    mutable line_t k, b, x;
    bool operator < (Line rhs)const{
        return k > rhs.k || k == rhs.k && b < rhs.b;
    }
    bool operator < (line_t rhs)const{
        return x < rhs;
    }
};
struct LCH : set<Line, less<> >{
    static const line_t sup = 1e6, inf = 0;
    long long floor(line_t x, line_t y){
        return x / y - ((x ^ y) < 0 && x % y);
    }
    bool over(iterator x, iterator y){
        if(y == end()){
            x->x = sup;
            return 0;
        }
        if(x->k == y->k) x->x = inf;
        else x->x = floor(y->b - x->b, x->k - y->k);
        return x->x >= y->x;
    }
    void add(Line v){
        auto z = insert(v).first, y = z++, x = y;
        while(over(y, z))
            z = erase(z);
        if(x == begin()) return;
        if(over(--x, y))
            over(x, erase(y));
        while((y = x) != begin() && (--x)->x >= y->x)
            over(x, erase(y));
    }
    line_t query(int x){
        auto it = lower_bound(x);
        return 1ll * it->k * x + it->b;
    }
};
```

代码3 (by jyc)

```
void Andrew(vector<PDD>& a,vector<PDD>& up,vector<PDD>& down){
    sort(a.begin(),a.end(),[](const PDD& A,const PDD& B){
        if(dcmp(A.y,B.y)==0) return dcmp(A.x,B.x)==-1;
        return dcmp(A.y,B.y)==-1;
    });
    up={a[0]};
    for(int i=1;i<a.size();i++){
        while(up.size()>=2 && sign(area(up[up.size()-2],up.back(),a[i]))>=0)
            up.pop_back();
        up.push_back(a[i]);
    }
    down={up.back()};
    for(int i=a.size()-1;i>=0;i--){
        while(down.size()>=2 &&
sign(area(down[down.size()-2],down.back(),a[i]))>=0)
            down.pop_back();
        down.push_back(a[i]);
    }
}
```

半平面交

by fhy

```
using p_t = long double;
const p_t eps = 1e-9;
inline int sgn(p_t x) {
    return (x > eps) - (x < -eps);
}
struct P{
    p_t x, y;
    P(p_t x, p_t y): x(x), y(y){}
    P(): x(0), y(0){}
    inline P operator - (P rhs)const{
        return {x - rhs.x, y - rhs.y};
    }
    inline P operator + (P rhs) const{
        return {x + rhs.x, y + rhs.y};
    }
    inline P operator * (p_t k)const{
        return {x * k, y * k};
    }
    inline p_t operator ^ (P rhs)const{
        return x * rhs.y - rhs.x * y;
    }
    inline p_t operator * (P rhs)const{
        return x * rhs.x + y * rhs.y;
    }
    inline friend istream& operator >> (istream &in, P &rhs){
        return in >> rhs.x >> rhs.y;
    }
}p[200010], t[200010], c[5];
P e = {0, -1};
inline int cmp(P i, P j){
    p_t x = i ^ e, y = j ^ e;
    int a = x == 0 && i * e >= 0, b = y == 0 && j * e >= 0;
    if(a || b) return b - a;
    if(x == 0 || y == 0 || sgn(x) == sgn(y)) return sgn(i ^ j);
    return sgn(y - x);
}
struct L{
    P a, x;
    L(P a, P b): a(a), x(b - a){}
    L(): a(), x(){}
    inline P operator ^ (L rhs)const{
        return a - x * ((rhs.x ^ (a - rhs.a)) / (rhs.x ^ x));
    }
    inline bool operator < (L rhs)const{
        int o = cmp(x, rhs.x);
        if (o) return o > 0;
        return pos(rhs.a) < 0;
    }
    inline bool operator == (L rhs) const {
        return cmp(x, rhs.x) == 0;
    }
    inline void inv() {
```

```
        x = {-x.y, x.x};
    }
    inline int pos(P rhs)const{
        return sgn(x ^ rhs - a);
    }
}s[200010], q[200010];

int n;
p_t xl = -1e4, yl = -1e4, xr = 1e4, yr = 1e4;
int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin >> n;
    int m = 0;
    for(int i = 1; i <= n; i++){
        p_t a, b, c, d;
        cin >> a >> b >> c >> d;
        s[m++] = L(P(a, b), P(c, d));
    }
    for(int i = 0; i <= 4; i++)
        c[i] = {i == 1 || i == 2 ? xr : xl, i >> 1 & 1 ? yr : yl};
    for(int i = 0; i < 4; i++)
        s[m++] = {c[i], c[i + 1]};
    sort(s, s + m);

    int l = 0, r = 0;
    for (int i = 0; i < m; i++) {
        if (i && s[i] == s[i - 1]) continue;
        while (l + 1 < r && s[i].pos(t[r - 1]) <= 0) {
            r -= 1;
        }
        while (l + 1 < r && s[i].pos(t[l + 1]) <= 0) {
            l += 1;
        }
        q[r++] = s[i];
        if (l + 1 < r) {
            t[r - 1] = q[r - 2] ^ q[r - 1];
        }
    }
    while (l + 1 < r && q[l].pos(t[r - 1]) <= 0) {
        r -= 1;
    }
    t[l] = t[r] = q[r - 1] ^ q[l];

    p_t res = 0;
    for(int i = l + 1; i < r - 1; i++){
        res += (t[i] - t[r]) ^ (t[i + 1] - t[r]);
    }
    printf("%.3Lf\n", res / 2);
    return 0;
}
```

杂项

火车头

娱乐用，用来平时做题凹卡常题

```
#pragma GCC target("avx")
#pragma GCC optimize(1)
#pragma GCC optimize(2)
#pragma GCC optimize(3)
#pragma GCC optimize("Ofast")
#pragma GCC optimize("inline")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
#pragma GCC optimize("-fipa-sra")
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vrp")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-ffast-math")
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("unroll-loops")
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-funroll-loops")
#pragma GCC optimize("-fwhole-program")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-fstrict-overflow")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-skip-blocks")
#pragma GCC optimize("-fcse-follow-jumps")
#pragma GCC optimize("-fsched-interblock")
#pragma GCC optimize("-fpartial-inlining")
#pragma GCC optimize("no-stack-protector")
#pragma GCC optimize("-freorder-functions")
#pragma GCC optimize("-findirect-inlining")
#pragma GCC optimize("-fhoist-adjacent-loads")
#pragma GCC optimize("-frerun-cse-after-loop")
#pragma GCC optimize("inline-small-functions")
#pragma GCC optimize("-finline-small-functions")
#pragma GCC optimize("-ftree-switch-conversion")
#pragma GCC optimize("-foptimize-sibling-calls")
#pragma GCC optimize("-fexpensive-optimizations")
#pragma GCC optimize("-funsafe-loop-optimizations")
#pragma GCC optimize("inline-functions-called-once")
#pragma GCC optimize("-fdelete-null-pointer-checks")
```

模拟退火

用一句话概括过程：如果新状态的解更优则修改答案，否则以一定的概率接收新状态

定义当前温度为 T，新状态 S' 和已知状态 S （新状态由已知状态通过随机的方式得到）之间的能量（值）差为 ΔE $(\Delta E \geq 0)$，则发生状态转移（修改最优解）的概率为

$$P(\Delta E) = \begin{cases} 1 & S' \text{ is better than S} \\ exp(-\Delta E/T) & \text{otherwise} \end{cases}$$

$P.S.$

1. 这里 E 越小代表越接近最优解，实际若 E 越大越接近最优解，可考虑把 $-E$ 作为能量值函数
2. 为了使得解更为精确，通常不直接取当前解作为答案，而是在退火过程中维护遇到的所有解的最优值
3. 有时为了使得到的解更有质量，会在模拟退火结束后，以当前温度在得到的解附近多次随机状态，尝试得到更优的解（其过程与模拟退火相似）

以下代码以P1337 [JSOI2004] 平衡点 / 吊打XXX（求 n 个点的带权费马点）为例（改了改 wiki 里的代码）

```cpp
mt19937 rng(chrono::system_clock::now().time_since_epoch().count());
double Rand(){return (long double)(rng())/UINT_MAX;}//生成一个[0,1]的小数
int Rand(int l,int r){return rng()%(r-l+1)+l;}//生成一个[l,r]的整数

#define N 1010
int n,x[N],y[N],w[N];
double ansx,ansy,dis;

//calc返回能量值函数，在calc中更新答案
double calc(double xx,double yy){
    double res=0;
    for (int i=1;i<=n;i++){
        double dx=x[i]-xx,dy=y[i]-yy;
        res+=sqrt(dx*dx+dy*dy)*w[i];
    }
    if(res<dis) dis=res,ansx=xx,ansy=yy;
    return res;
}

void SA(){//simulated_annealing
    double nowx=ansx,nowy=ansy;
    double eps_t=1e-4;
    //模拟退火参数，起始温度，终止温度，每次乘以的系数
    for(auto t=2e5;t>eps_t;t*=0.99){
        double nxtx=nowx+t*(Rand()*2-1);
        double nxty=nowy+t*(Rand()*2-1);
        double delta=calc(nxtx,nxty)-calc(nowx,nowy);
        if(exp(-delta/t)>Rand()) nowx=nxtx,nowy=nxty;
    }
    //在得到的解附近多次随机状态，尝试得到更优的解
    for (int i=1;i<=1000;i++){
        double nxtx=ansx+eps_t*(Rand()*2-1);
        double nxty=ansy+eps_t*(Rand()*2-1);
        calc(nxtx,nxty);
    }
}
```

```cpp

int main() {
    n=read();
    for(int i=1;i<=n;i++){
        x[i]=read(),y[i]=read(),w[i]=read();
        ansx+=x[i],ansy+=y[i];
    }
    ansx/=n,ansy/=n,dis=calc(ansx,ansy);
    //卡时技巧，填一个比题目时限略小的数
    while((double)clock()/CLOCKS_PER_SEC<0.9) SA();
    printf("%.3lf %.3lf\n", ansx, ansy);
    return 0;
}
```

写在最后（来自Benq大神的几句话）

```
stuff you should look for
int overflow, array bounds
special cases (n=1?)
do smth instead of nothing and stay organized
WRITE STUFF DOWN
DON'T GET STUCK ON ONE APPROACH
```