目录

[DataStruct 5](#_Toc207919213)

[01tire 5](#_Toc207919214)

[BIT(树状数组) 7](#_Toc207919215)

[DSU(并查集) 9](#_Toc207919216)

[FHQTreap(无旋平衡树) 11](#_Toc207919217)

[[应用]Treap维护数对 16](#_Toc207919218)

[jiangly线段树(info) 21](#_Toc207919219)

[st表 24](#_Toc207919220)

[二叉搜索树 26](#_Toc207919221)

[可删改堆 30](#_Toc207919222)

[手写堆 33](#_Toc207919223)

[普通莫队 35](#_Toc207919224)

[李超线段树 38](#_Toc207919225)

[珂朵莉树(ODT) 41](#_Toc207919226)

[笛卡尔树 43](#_Toc207919227)

[线段树(max&min),tested,class 45](#_Toc207919228)

[线段树(max&min,upd+set),tested,class 48](#_Toc207919229)

[线段树(sum),tested ,class 52](#_Toc207919230)

[线段树(sum,update+set),tested,class 55](#_Toc207919231)

[Graph 58](#_Toc207919232)

[2-sat 58](#_Toc207919233)

[BCC（点双连通分量，tarjan） 61](#_Toc207919234)

[DSU on tree(树上启发式合并) 64](#_Toc207919235)

[EDCC (边双联通分量，tarjan） 67](#_Toc207919236)

[SCC（低注释） 69](#_Toc207919237)

[SCC（强联通分量，缩点，tarjan） 72](#_Toc207919238)

[ShortestPath（Bellman\_Ford,SPFA) 76](#_Toc207919239)

[ShortestPath（Floyed） 79](#_Toc207919240)

[ShortestPath（dijkstra） 81](#_Toc207919241)

[dfs&bfs 84](#_Toc207919242)

[johnson最短路 87](#_Toc207919243)

[二分图最大匹配 91](#_Toc207919244)

[二分图染色 95](#_Toc207919245)

[分层图 98](#_Toc207919246)

[差分约束 101](#_Toc207919247)

[拓扑排序 104](#_Toc207919248)

[最大流(dinic) 106](#_Toc207919249)

[最小斯坦纳树 109](#_Toc207919250)

[最小生成树（kruskal） 111](#_Toc207919251)

[最小生成树（prim） 114](#_Toc207919252)

[最小费用最大流(dinic) 117](#_Toc207919253)

[最小费用最大流(dinic,浮点) 120](#_Toc207919254)

[最小费用最大流(原始对偶) 124](#_Toc207919255)

[最近公共祖先（LCA）(targan,静态) 127](#_Toc207919256)

[最近公共祖先（LCA）(倍增) 130](#_Toc207919257)

[树上基本处理 133](#_Toc207919258)

[树链剖分（线段树ver.） 135](#_Toc207919259)

[点分治 141](#_Toc207919260)

[虚树(可拓展版) 144](#_Toc207919261)

[虚树(带边权) 147](#_Toc207919262)

[Other 151](#_Toc207919263)

[离散化 151](#_Toc207919264)

[hash 152](#_Toc207919265)

[hash表 152](#_Toc207919266)

[字符串双hash 157](#_Toc207919267)

[string 160](#_Toc207919268)

[AC自动机(dp版) 160](#_Toc207919269)

[AC自动机(可拓展版) 163](#_Toc207919270)

[Manacher 165](#_Toc207919271)

[exKMP 167](#_Toc207919272)

[kmp 169](#_Toc207919273)

[tiretree 171](#_Toc207919274)

[动态规划 173](#_Toc207919275)

[数位dp(例题1,数位和) 173](#_Toc207919276)

[数论 177](#_Toc207919277)

[(ex)CRT((扩展)中国剩余定理) 177](#_Toc207919278)

[FFT(快速傅里叶变换) 180](#_Toc207919279)

[乘法逆元 182](#_Toc207919280)

[安全取模类 184](#_Toc207919281)

[数论预处理 187](#_Toc207919282)

[整除分块 192](#_Toc207919283)

[矩阵快速幂 194](#_Toc207919284)

[筛法求积性函数 197](#_Toc207919285)

[线性基(gauss) 199](#_Toc207919286)

[线性基(贪心法) 202](#_Toc207919287)

[组合数预处理 204](#_Toc207919288)

[计算几何 209](#_Toc207919289)

[三角剖分 209](#_Toc207919290)

[凸包 213](#_Toc207919291)

[向量 216](#_Toc207919292)

[旋转卡壳 219](#_Toc207919293)

[极角排序 224](#_Toc207919294)

DataStruct

01tire

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
class O1Tire{  
 public:  
 struct node  
 {  
 node\* ch[2];  
 int cnt;  
 node():ch{nullptr,nullptr},cnt(0){}  
 };  
 node\* root;  
 O1Tire():root(new node){}  
 void set(int x,int t)//从高到低建树  
 {  
 node\* p=root;  
 for(int i=31;i>=0;i--)  
 {  
 int d=(x>>i)&1;  
 if(!p->ch[d])  
 p->ch[d]=new node();  
 p->ch[d]->cnt+=t;  
 p=p->ch[d];  
 }  
 }  
 int findMax(int x)//从高到低找,贪心选择,求解x对tire中所有数的最大异或值  
 {  
 node\* p=root;  
 int res=0;  
 for(int i=31;i>=0;i--)  
 {  
 int d=(x>>i)&1;  
 if(p->ch[d^1]&&p->ch[d^1]->cnt)  
 p=p->ch[d^1],res+=(1<<i);  
 else  
 p=p->ch[d];  
 if(!p)  
 return res;  
 }  
 return res;  
 }  
   
};  
int main()  
{  
 int T\_start=clock();  
 int t;cin>>t;  
 while(t--)  
 {  
 int n,k;cin>>n>>k;  
 vector<int> a(n);  
 for(int i=0;i<n;i++)  
 cin>>a[i];  
 O1Tire tire;  
 int ans=0xfffffff;  
 for(int i=0,j=0;i<n;i++)  
 {  
 tire.set(a[i],1);  
 while(j<=i&&tire.findMax(a[i])>=k)  
 {  
 ans=min(ans,i-j+1);  
 tire.set(a[j],-1);  
 j++;  
 }  
 }  
 if(ans==0xfffffff) cout<<-1<<endl;  
 else cout<<ans<<endl;  
 }  
 return 0;  
}

BIT(树状数组)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
using namespace std;  
class BIT{  
 private:  
 int n;  
 vector<int> tree;//tree[i] 是[i-lowbit(i)+1,i]的和,[1,n]存储  
 int lowbit(int x){  
 return x&(-x);  
 }  
 public:  
 BIT(int n): n(n),tree(n+1,0){}  
 void update(int i,int val)//单点修改 a[i]+=val  
 {  
 while(i<=n){   
 tree[i]+=val;  
 i+=lowbit(i);//跳到后一个lowbit(x)的位置  
 }  
 }  
 int query(int l,int r)//区间查询 [l,r]的和  
 {  
 int res=0;  
 while(r>=l){  
 res+=tree[r];  
 r-=lowbit(r);//跳到前一个lowbit(x)的位置  
 }  
 return res;  
 }  
 int query\_diff(int i)//单点查询 a\_diff[i] (维护差分数组)=sum[1,i]  
 {  
 return query(1,i);  
 }  
 void update\_diff(int l,int r,int val)//区间修改 (维护差分数组) a\_diff[l]+=val,a\_diff[r+1]-=val  
 {  
 update(l,val);  
 update(r+1,-val);  
 }  
 void init(vector<int> a)//初始化  
 {  
 vector<int> presum(a.size()+1,0);  
 for(int i=1;i<=a.size();i++)  
 {  
 presum[i]=presum[i-1]+a[i-1];  
 tree[i]=presum[i]-presum[i-lowbit(i)];//按定义  
 }   
 }  
};  
int main()  
{  
 int T\_start=clock();  
 //test  
 int n=10; vector<int> a={1,3,2,4,2,1,5,4,3,2},a\_diff={1,2,-1,2,-2,-1,4,-1,-1,-1};//a\_diff[i]=a[i]-a[i-1]  
 BIT bit(n),bit\_diff(n);  
 bit.init(a);bit\_diff.init(a\_diff);  
 cout<<bit.query(1,5)<<endl;  
 bit.update(1,5);  
 cout<<bit.query(1,5)<<endl;  
 bit\_diff.update\_diff(1,5,2);  
 cout<<bit\_diff.query\_diff(5)<<endl;  
 cout<<bit\_diff.query\_diff(6)<<endl;  
 //test end  
 return 0;  
}  
//进阶用法1.维护差分数组  
//进阶用法2.把数组离散化后按照值域建树状数组，可以用来求逆序对(第K大)  
//e.g.val[1,16,9,10,3]->dis[1,5,3,4,2]->bit[1,1,1,1,1]  
// BIT bit(5);  
// //bit.update(1,1);bit.update(2,1);bit.update(3,1);bit.update(4,1);bit.update(5,1);  
// val\_i[1,3]即更新为[1,0,1,0,1] 9即为第2大 即bit.query(1,3)=2  
//求逆序对，how to do? 即[1,r]中比a[r]大的数的个数

DSU(并查集)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
class DSU{  
 public:  
 int n;vector<int> fa,sz;  
 DSU(int n):n(n)  
 {  
 srand(time(NULL));  
 fa.resize(n+1);  
 sz.resize(n+1);  
 for(int i=1;i<=n;i++)  
 {  
 fa[i]=i;  
 sz[i]=1;  
 }  
 }  
 int find(int u){  
 return fa[u]==u?u:fa[u]=find(fa[u]);  
 }  
 void merge(int a,int b)  
 {  
 int u=find(a),v=find(b);  
 if(u==v) return;  
 fa[u]=v;  
 sz[v]+=sz[u];  
 }  
 int same(int a,int b)  
 {  
 return find(a)==find(b);  
 }  
 int size(int u){  
 return sz[find(u)];  
 }  
 vector<vector<int>> get(){  
 vector<vector<int>> ans(n+1);  
 for(int i=1;i<=n;i++)  
 {  
 ans[find(i)].push\_back(i);  
 }  
 ans.erase(remove(ans.begin(),ans.end(),vector<int>()),ans.end());  
 return ans;  
 }  
};  
int main()  
{  
 int T\_start=clock();  
 srand(time(NULL));  
 return 0;  
}

FHQTreap(无旋平衡树)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
class FHQTreap{  
 //无旋Treap：1.满足二叉搜索树性质(val) 2.满足堆性质（优先级）  
 //树堆：BST+Heap  
 public:  
 struct Node{  
 int val;  
 int size=0;  
 int priority;//随机数  
 Node \*left, \*right;  
 Node(int val):val(val),priority(rand()),left(NULL),right(NULL),size(1){}  
 Node(int val,int priority):val(val),priority(priority),left(NULL),right(NULL),size(1){}  
 };  
 bool cmp(int a,int b){  
 return a<=b;  
 }  
 Node \*root;  
 FHQTreap():root(NULL){}  
 void merge(Node \*&root,Node \*a,Node \*b){  
 //val a<=val b(内部满足Treap)  
 if(!a)root=b;  
 else if(!b)root=a;  
 else{  
 if(a->priority>b->priority){//a的优先级大  
 root=a;//a作为根(为了满足Heap(大))  
 merge(a->right,a->right,b);//b合并到a的右子树（为了满足BST：a的右子树的所有节点都大于a)  
 }  
 else{  
 root=b;  
 merge(b->left,a,b->left);  
 }  
 }  
 if(root)  
 {  
 root->size=1;  
 if(root->left)root->size+=root->left->size;  
 if(root->right)root->size+=root->right->size;  
 //cout<<root->val<<' '<<root->size<<endl;  
 }  
 }  
 void split(Node \*root,Node \*&a,Node \*&b,int val){  
 //将root按照val分割为a,b两部分  
 //a的val都小于等于val，b的val都大于val  
 if(!root){  
 a=b=NULL;  
 return;  
 }  
 if(cmp(root->val,val)){  
 a=root;  
 split(root->right,a->right,b,val);  
 }  
 else{  
 b=root;  
 split(root->left,a,b->left,val);  
 }  
 if(root)  
 {  
 root->size=1;  
 if(root->left)root->size+=root->left->size;  
 if(root->right)root->size+=root->right->size;  
 //cout<<root->val<<' '<<root->size<<endl;  
 }  
 }  
 void insert(int val){  
 Node \*a,\*b;  
 split(root,a,b,val);//将root按照val分割为a,b两部分  
 merge(a,a,new Node(val));//将val插入到a中  
 merge(root,a,b);//将a,b合并为root  
 //偶还能这样  
 }  
 void erase(int val){  
 Node \*a,\*b,\*c;  
 split(root,a,b,val);//将root按照val分割为a,b两部分  
 split(a,a,c,val-1);//将a按照val-1分割为a,c两部分  
 if(c)  
 {  
 merge(a,a,c->right);//将c的右子树合并到a中(删除一个节点)  
 merge(a,a,c->left);//将c的左子树合并到a中(删除一个节点)  
 }  
 merge(root,a,b);//将a,b合并为root  
 }  
 void print(Node \*root){  
 if(!root)return;  
 print(root->left);  
 cout<<root->val<<" ";  
 print(root->right);  
 }  
 int findMax(Node \*root){  
 if(!root)return -1;  
 while(root->right)root=root->right;  
 return root->val;  
 }  
 int findMin(Node \*root){  
 if(!root)return -1;  
 while(root->left)root=root->left;  
 return root->val;  
 }  
 int pre(int val){  
 Node \*a,\*b;  
 split(root,a,b,val-1);//将root按照val-1分割为a,b两部分  
 int res=findMax(a);  
 merge(root,a,b);  
 return res;  
 }  
 int next(int val){  
 Node \*a,\*b;  
 split(root,a,b,val);//将root按照val分割为a,b两部分  
 int res=findMin(b);  
 merge(root,a,b);  
 return res;  
 }  
 int rank(int val){  
 Node \*a,\*b;  
 split(root,a,b,val-1);//将root按照val-1分割为a,b两部分  
 int res=(a?a->size:0)+1;  
 merge(root,a,b);  
 return res;  
 }  
 int QueryKth(int k){  
 return KthQuery(root,k);  
 }  
 int KthQuery(Node\* root,int k){  
 if(root==nullptr) return -1;  
 int leftsize=root->left?root->left->size:0;  
 if(k<=leftsize) return KthQuery(root->left,k);  
 else if(k==leftsize+1) return root->val;  
 else return KthQuery(root->right,k-leftsize-1);  
 }  
 bool find(int val){  
 Node \*a,\*b;  
 split(root,a,b,val);//将root按照val分割为a,b两部分  
 bool res=a&&findMax(a)==val;  
 merge(root,a,b);  
 return res;  
 }  
   
};  
int main()  
{  
 int T\_start=clock();  
 FHQTreap treap;  
 vector<int> test={1,2,3,4,5,6,7,8,9,10};  
 /\*for(int i=0;i<test.size();i++){  
 treap.insert(test[i]);  
 }  
 treap.print(treap.root);  
 cout<<endl;  
 treap.erase(5);  
 treap.print(treap.root);  
 cout<<endl;  
 treap.insert(5);  
 treap.insert(5);  
 treap.insert(5);  
 treap.insert(5);  
 treap.print(treap.root);  
 cout<<endl;  
 treap.erase(5);  
 treap.print(treap.root);  
 cout<<endl;\*/  
 cout<<treap.pre(5)<<endl;  
 cout<<treap.next(5)<<endl;  
 cout<<treap.rank(5)<<endl;  
 treap.erase(5);  
 treap.print(treap.root);  
 cout<<endl;  
 treap.insert(5);  
 treap.insert(5);  
 treap.insert(5);  
 treap.print(treap.root);  
 cout<<endl;  
 cout<<treap.rank(6)<<endl;  
 treap.erase(5);  
 treap.print(treap.root);  
 cout<<endl;  
 cout<<treap.rank(8)<<endl;  
 cout<<treap.QueryKth(7)<<endl;  
 return 0;  
}

[应用]Treap维护数对

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
class FHQTreap{  
 //无旋Treap：1.满足二叉搜索树性质(val) 2.满足堆性质（优先级）  
 //树堆：BST+Heap  
 public:  
 struct Node{  
 int l,r;  
 int size=0;  
 int priority;//随机数  
 Node \*left, \*right;  
 Node(int l,int r):l(l),r(r),priority(rand()),left(NULL),right(NULL),size(1){}  
 };  
 bool cmp(Node \*a,pair<int,int> val){  
 if(a->l==val.first) return a->r<=val.second;  
 return a->l<val.first;  
 }  
 Node \*root;  
 FHQTreap():root(NULL){}  
 void merge(Node \*&root,Node \*a,Node \*b){  
 //val a<=val b(内部满足Treap)  
 if(!a)root=b;  
 else if(!b)root=a;  
 else{  
 if(a->priority>b->priority){//a的优先级大  
 root=a;//a作为根(为了满足Heap(大))  
 merge(a->right,a->right,b);//b合并到a的右子树（为了满足BST：a的右子树的所有节点都大于a)  
 }  
 else{  
 root=b;  
 merge(b->left,a,b->left);  
 }  
 }  
 if(root)  
 {  
 root->size=1;  
 if(root->left)root->size+=root->left->size;  
 if(root->right)root->size+=root->right->size;  
 //cout<<root->val<<' '<<root->size<<endl;  
 }  
 }  
 void split(Node \*root,Node \*&a,Node \*&b,pair<int,int> val){  
 //将root按照val分割为a,b两部分  
 //a的val都小于等于val，b的val都大于val  
 if(!root){  
 a=b=NULL;  
 return;  
 }  
 if(cmp(root,val)){  
 a=root;  
 split(root->right,a->right,b,val);  
 }  
 else{  
 b=root;  
 split(root->left,a,b->left,val);  
 }  
 if(root)  
 {  
 root->size=1;  
 if(root->left)root->size+=root->left->size;  
 if(root->right)root->size+=root->right->size;  
 //cout<<root->val<<' '<<root->size<<endl;  
 }  
 }  
 void insert(pair<int,int> val){  
 Node \*a,\*b;  
 split(root,a,b,val);//将root按照val分割为a,b两部分  
 merge(a,a,new Node(val.first,val.second));//将val插入到a中  
 merge(root,a,b);//将a,b合并为root  
 //偶还能这样  
 }  
 void erase(pair<int,int> val){  
 Node \*a,\*b,\*c;  
 split(root,a,b,val);//将root按照val分割为a,b两部分  
 split(a,a,c,{val.first,val.second-1});//将a按照val-1分割为a,c两部分  
 if(c)  
 {  
 merge(a,a,c->right);//将c的右子树合并到a中(删除一个节点)  
 merge(a,a,c->left);//将c的左子树合并到a中(删除一个节点)  
 }  
 merge(root,a,b);//将a,b合并为root  
 }  
 pair<int,int> findMax(Node \*root){  
 if(!root)return {-1,-1};  
 while(root->right)root=root->right;  
 return {root->l,root->r};  
 }  
 pair<int,int> findMin(Node \*root){  
 if(!root)return {-1,-1};  
 while(root->left)root=root->left;  
 return {root->l,root->r};  
 }  
 pair<int,int> pre(pair<int,int> val){  
 Node \*a,\*b;  
 split(root,a,b,{val.first,val.second-1});//将root按照val-1分割为a,b两部分  
 pair<int,int> res=findMax(a);  
 merge(root,a,b);  
 return res;  
 }  
 pair<int,int> next(pair<int,int> val){  
 Node \*a,\*b;  
 split(root,a,b,val);//将root按照val分割为a,b两部分  
 pair<int,int> res=findMin(b);  
 merge(root,a,b);  
 return res;  
 }  
 bool find(pair<int,int> val)  
 {  
 Node \*a,\*b;  
 split(root,a,b,val);  
 bool res=a&&findMax(a).second==val.second&&findMax(a).first==val.first;  
 merge(root,a,b);  
 return res;  
 }  
 int size()  
 {  
 return root?root->size:0;  
 }  
   
};  
int read()  
{  
 int s=0,f=1;  
 char ch=getchar();  
 while(ch<'0'||ch>'9')  
 {  
 if(ch=='-') f=-1;  
 ch=getchar();  
 }  
 while(ch>='0'&&ch<='9')  
 {  
 s=(s<<3)+(s<<1)+ch-'0';  
 ch=getchar();  
 }  
 return s\*f;  
}  
inline void write(int x)   
{  
 static int sta[35];   
 int top=0;  
 if(x<0&&x!=-2147483648) {putchar('-');x=-x;}  
 if(x==-2147483648) {printf("-2147483648");return;}  
 do{  
 sta[top++]=x%10, x/=10;  
 }while(x);  
 while(top) putchar(sta[--top]+48);  
}  
int main()  
{  
 //freopen("in.txt","r",stdin);  
 //freopen("out2.txt","w",stdout);  
 ios::sync\_with\_stdio(false);  
 cin.tie(0);  
 cout.tie(0);  
 int t;cin>>t;  
 FHQTreap tree;  
 while(t--)  
 {  
 char op;  
 cin>>op;  
 if(op=='B')  
 {  
 cout<<tree.size()<<'\n';  
 }  
 else if(op=='A')  
 {  
 int x,y;  
 cin>>x>>y;  
 auto check=[](pair<int,int> a,pair<int,int> b)->bool{  
 vector<int> v1={a.first,a.second,b.first,b.second};  
 sort(v1.begin(),v1.end());  
 return ((v1[0]==a.first&&v1[1]==a.second&&v1[2]==b.first&&v1[3]==b.second)||(v1[0]==b.first&&v1[1]==b.second&&v1[2]==a.first&&v1[3]==a.second))&&v1[1]!=v1[2];  
 };  
 int ans=0;  
 if(tree.find({x,y}))  
 {  
 tree.erase({x,y});  
 ans++;  
 }  
 while(tree.size()&&tree.pre({x,y}).first!=-1&&!check(tree.pre({x,y}),{x,y}))   
 {  
 ans++;  
 tree.erase(tree.pre({x,y}));  
 }  
 while(tree.size()&&tree.next({x,y}).first!=-1&&!check(tree.next({x,y}),{x,y}))   
 {  
 ans++;  
 tree.erase(tree.next({x,y}));  
 }  
 cout<<ans<<'\n';  
 tree.insert({x,y});  
 }  
 }  
 return 0;  
}

jiangly线段树(info)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
using namespace std;  
template<class Info, class Tag>  
struct LazySegmentTree {  
 const int n;  
 std::vector<Info> info;  
 std::vector<Tag> tag;  
 LazySegmentTree(int n) : n(n), info(4 << std::\_\_lg(n)), tag(4 << std::\_\_lg(n)) {}  
 LazySegmentTree(std::vector<Info> init) : LazySegmentTree(init.size()) {  
 std::function<void(int, int, int)> build = [&](int p, int l, int r) {  
 if (r - l == 1) {  
 info[p] = init[l];  
 return;  
 }  
 int m = (l + r) / 2;  
 build(2 \* p, l, m);  
 build(2 \* p + 1, m, r);  
 pull(p);  
 };  
 build(1, 0, n);  
 }  
 void pull(int p) {  
 info[p] = info[2 \* p] + info[2 \* p + 1];  
 }  
 void apply(int p, const Tag &v) {  
 info[p].apply(v);  
 tag[p].apply(v);  
 }  
 void push(int p) {  
 apply(2 \* p, tag[p]);  
 apply(2 \* p + 1, tag[p]);  
 tag[p] = Tag();  
 }  
 void modify(int p, int l, int r, int x, const Info &v) {  
 if (r - l == 1) {  
 info[p] = v;  
 return;  
 }  
 int m = (l + r) / 2;  
 push(p);  
 if (x < m) {  
 modify(2 \* p, l, m, x, v);  
 } else {  
 modify(2 \* p + 1, m, r, x, v);  
 }  
 pull(p);  
 }  
 void modify(int p, const Info &v) {  
 modify(1, 0, n, p, v);  
 }  
 Info rangeQuery(int p, int l, int r, int x, int y) {  
 if (l >= y || r <= x) {  
 return Info();  
 }  
 if (l >= x && r <= y) {  
 return info[p];  
 }  
 int m = (l + r) / 2;  
 push(p);  
 return rangeQuery(2 \* p, l, m, x, y) + rangeQuery(2 \* p + 1, m, r, x, y);  
 }  
 Info rangeQuery(int l, int r) {  
 return rangeQuery(1, 0, n, l, r);  
 }  
 void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {  
 if (l >= y || r <= x) {  
 return;  
 }  
 if (l >= x && r <= y) {  
 apply(p, v);  
 return;  
 }  
 int m = (l + r) / 2;  
 push(p);  
 rangeApply(2 \* p, l, m, x, y, v);  
 rangeApply(2 \* p + 1, m, r, x, y, v);  
 pull(p);  
 }  
 void rangeApply(int l, int r, const Tag &v) {  
 return rangeApply(1, 0, n, l, r, v);  
 }  
 void half(int p, int l, int r) {  
 if (info[p].act == 0) {  
 return;  
 }  
 if ((info[p].min + 1) / 2 == (info[p].max + 1) / 2) {  
 apply(p, {-(info[p].min + 1) / 2});  
 return;  
 }  
 int m = (l + r) / 2;  
 push(p);  
 half(2 \* p, l, m);  
 half(2 \* p + 1, m, r);  
 pull(p);  
 }  
 void half() {  
 half(1, 0, n);  
 }  
};  
struct Tag {  
 //tag清空态  
 void apply(Tag t) {  
 //tag t下发对tag的影响  
 }  
};  
struct Info {  
 //维护啥信息  
 void apply(Tag t) {  
 //tag t下发对info的影响  
 }  
};  
Info operator + (Info a, Info b) {  
 Info c;  
 //info a和info b合并  
 return c;  
}  
//tip:[l,r)区间->传入[l,r]改为[l,r+1)  
int main()  
{  
 int T\_start=clock();  
   
 return 0;  
}

st表

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
class st{  
 public:  
 vector<vector<int>> dp;//dp[i][j]是[i,i+2^j-1]的min/max  
 int inf(int a,int b)  
 {  
 return max(a,b);  
 }  
 void init(vector<int>& nums,int siz)  
 {  
 int len=log2(siz)+1;  
 dp.resize(siz);  
 for(auto &i:dp) i.resize(len);  
 for(int i=0;i<siz;i++)  
 {  
 dp[i][0]=nums[i];  
 }  
 for(int j=1;j<=len;j++)  
 {  
 for(int i=0;i+(1<<j)-1<siz;i++)  
 {  
 dp[i][j]=inf(dp[i][j-1],dp[i+(1<<(j-1))][j-1]);  
 }  
 }  
 }  
 int query(int l,int r)  
 {  
 int k=log2(r-l+1);  
 return inf(dp[l][k],dp[r-(1<<k)+1][k]);  
 }  
 st(vector<int>& nums){  
 init(nums,nums.size());  
 }  
};  
int read()  
{  
 int s=0,f=1;  
 char ch=getchar();  
 while(ch<'0'||ch>'9')  
 {  
 if(ch=='-') f=-1;  
 ch=getchar();  
 }  
 while(ch>='0'&&ch<='9')  
 {  
 s=(s<<3)+(s<<1)+ch-'0';  
 ch=getchar();  
 }  
 return s\*f;  
}  
inline void write(int x)   
{  
 static int sta[35];   
 int top=0;  
 if(x<0&&x!=-2147483648) {putchar('-');x=-x;}  
 if(x==-2147483648) {printf("-2147483648");return;}  
 do{  
 sta[top++]=x%10, x/=10;  
 }while(x);  
 while(top) putchar(sta[--top]+48);  
}  
int main()  
{  
 int T\_start=clock();  
 int n=read(),m=read();  
 vector<int> nums(n);  
 for(int i=0;i<n;i++)  
 {  
 nums[i]=read();  
 }  
 st s(nums);  
 for(int i=0;i<m;i++)  
 {  
 int l=read(),r=read();  
 write(s.query(l-1,r-1));  
 putchar('\n');  
 }  
 return 0;  
}

二叉搜索树

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
using namespace std;  
struct SBTreeNode{  
 int key;  
 SBTreeNode\* left;  
 SBTreeNode\* right;  
 int size; //子树大小  
 int count; //重复个数  
  
 SBTreeNode(int val):  
 key(val),left(nullptr),right(nullptr),size(1),count(1){}  
};  
//二叉查找树具有以下性质：  
//1. 若任意节点的左子树不空，则左子树上所有节点的值均小于它的根节点的值；  
//2. 若任意节点的右子树不空，则右子树上所有节点的值均大于它的根节点的值；  
//3. 任意节点的左、右子树也分别为二叉查找树；  
int FindMin(SBTreeNode\* root)  
{  
 if(root==nullptr) return -0x3f3f3f3f;  
 while(root->left!=nullptr) root=root->left;  
 return root->key;  
}  
int FindMax(SBTreeNode\* root)  
{  
 if(root==nullptr) return 0x3f3f3f3f;  
 while(root->right!=nullptr) root=root->right;  
 return root->key;  
}  
SBTreeNode\* FindMinNode(SBTreeNode\* root)  
{  
 if(root==nullptr) return nullptr;  
 while(root->left!=nullptr) root=root->left;  
 return root;  
}  
void Inorder(SBTreeNode\* root){  
 if(root==nullptr) return;  
 Inorder(root->left);  
 cout<<root->key<<" ";  
 Inorder(root->right);  
}  
  
bool search(SBTreeNode\* root,int val){  
 if(root==nullptr) return false;  
 if(root->key==val) return true;  
 if(val<root->key) return search(root->left,val);  
 else return search(root->right,val);  
}  
  
SBTreeNode\* insert(SBTreeNode\* root,int val){  
 if(root==nullptr) return new SBTreeNode(val);  
 if(val<root->key) root->left=insert(root->left,val);  
 else if(val>root->key) root->right=insert(root->right,val);  
 else root->count++;  
 root->size=root->count+(root->left?root->left->size:0)+(root->right?root->right->size:0);  
 return root;  
}  
//先搜索到，再判断子树有无，再删除  
SBTreeNode\* deleteNode(SBTreeNode\* root,int val){  
 if(root==nullptr) return nullptr;  
 if(val<root->key) root->left=deleteNode(root->left,val);  
 else if(val>root->key) root->right=deleteNode(root->right,val);  
 else{  
 if(root->count>1) root->count--;  
 else{  
 if(root->left==nullptr)   
 {  
 SBTreeNode\* temp=root->right;  
 delete root;  
 return temp;  
 }  
 else if(root->right==nullptr)   
 {  
 SBTreeNode\* temp=root->left;  
 delete root;  
 return temp;  
 }  
 else if(root->left!=nullptr&&root->right!=nullptr)  
 {  
 SBTreeNode\* temp=FindMinNode(root->right);//右子树最小值代替该节点  
 root->key=temp->key;  
 root->count=temp->count;  
 temp->count=1;  
 root->right=deleteNode(root->right,temp->key);  
 }  
 }  
 }  
 root->size=root->count+(root->left?root->left->size:0)+(root->right?root->right->size:0);  
 return root;  
}  
int QueryRank(SBTreeNode\* root,int val){  
 if(root==nullptr) return 0;  
 if(val==root->key) return (root->left?root->left->size:0)+1; //找到，返回左子树大小+1  
 if(val<root->key) return QueryRank(root->left,val); //向左走  
 else return QueryRank(root->right,val)+(root->left?root->left->size:0)+root->count; //向右走，加上左子树大小+根节点个数  
}  
//左子树的大小>=k,在左子树中找第k个  
//左子树的大小在[k-count,k-1],为根节点  
//左子树的大小<k-count,在右子树中找  
int KthQuery(SBTreeNode\* root,int k){  
 if(root==nullptr) return -0x3f3f3f3f;  
 if(root->left)  
 {  
 if(root->left->size>=k) return KthQuery(root->left,k);  
 else if(root->left->size+root->count>=k) return root->key;  
 else   
 {  
 if(k==1) return root->key;  
 }  
 }  
 return KthQuery(root->right,k-(root->left?root->left->size:0)-root->count);  
}  
  
int main()  
{  
 int T\_start=clock();  
 //生成测试代码  
 vector<int> v(0,10);  
 for(int i=0;i<10;i++) v.push\_back(rand()%100);  
 for(int i=0;i<10;i++) cout<<v[i]<<" ";  
 cout<<endl;  
 SBTreeNode\* root=nullptr;  
 for(int i=0;i<10;i++) root=insert(root,v[i]);  
 Inorder(root);  
 cout<<endl;  
 cout<<"v[0]:"<<v[0]<<endl;  
 cout<<QueryRank(root,v[0])<<endl;  
 cout<<KthQuery(root,4)<<endl;  
 cout<<search(root,v[0])<<endl;  
 root=deleteNode(root,v[0]);  
 Inorder(root);  
 cout<<endl;  
 int insertded=39;  
 root=insert(root,insertded);  
 Inorder(root);  
 int T\_end=clock();  
 cout<<"Time cost: "<<T\_end-T\_start<<endl;  
 //测试代码结束  
 return 0;  
}

可删改堆

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
class Heap{  
 struct node{  
 int val;  
 };  
 int n,size=0,inscnt=0;  
 vector<node> heap;  
 vector<int>ph,hp;//pos->heap, heap->pos  
 public:  
 Heap(int n):n(n){  
 heap.resize(n+1);  
 ph.resize(n+1);  
 hp.resize(n+1);  
 }  
 bool cmp(node a,node b){  
 return a.val>b.val;  
 }  
 void \_swap(int a,int b){  
 swap(ph[hp[a]],ph[hp[b]]);  
 swap(hp[a],hp[b]);  
 swap(heap[a],heap[b]);  
 }  
 void push(int val){  
 heap[++size].val=val;  
 ph[++inscnt]=size;hp[size]=inscnt;;  
 int i=size;  
 while(i>1&&cmp(heap[i],heap[i>>1])){  
 \_swap(i,i>>1);  
 i>>=1;  
 }//向上调整  
 }  
 void pop(){  
 \_swap(1,size);  
 ph[size]=0;hp[size]=0;  
 size--; int i=1;  
 while(i<<1<=size){  
 int j=i<<1;  
 if(j+1<=size&&cmp(heap[j+1],heap[j]))j++;  
 if(cmp(heap[i],heap[j]))break;  
 \_swap(i,j); i=j;  
 }//向下调整  
 }  
 int top(){  
 return heap[1].val;  
 }  
 void remove(int pos){//删除第pos个元素(第pos个插入的元素)  
 int i=ph[pos];  
 \_swap(i,size);  
 ph[size]=0;hp[size]=0;  
 size--; int j=i;  
 while(j>1&&cmp(heap[j],heap[j>>1])){  
 \_swap(j,j>>1);  
 j>>=1;  
 }  
 while(j<<1<=size){  
 int k=j<<1;  
 if(k+1<=size&&cmp(heap[k+1],heap[k]))k++;  
 if(cmp(heap[j],heap[k]))break;  
 \_swap(j,k); j=k;  
 }  
 }  
 void modify(int pos,int val){  
 int i=ph[pos];  
 heap[i].val=val;  
 int j=i;  
 while(j>1&&cmp(heap[j],heap[j>>1])){  
 \_swap(j,j>>1);  
 j>>=1;  
 }  
 while(j<<1<=size){  
 int k=j<<1;  
 if(k+1<=size&&cmp(heap[k+1],heap[k]))k++;  
 if(cmp(heap[j],heap[k]))break;  
 \_swap(j,k); j=k;  
 }  
 }  
   
};  
int main()  
{  
 int T\_start=clock();  
 vector<int> a={1,2,3,4,5,6,7,8,9,10};  
 Heap h(a.size());  
 for(int i=0;i<a.size();i++){  
 h.push(a[i]);  
 }  
 h.remove(1);  
 h.modify(2,11);  
 cout<<h.top()<<endl;  
 return 0;  
}

手写堆

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
class Heap{//小根堆  
 public:  
 struct node{  
 int val;  
 };  
 vector<node> heap;  
 int n,size=0;  
 Heap(int n):n(n){  
 heap.resize(n+1);  
 }  
 bool cmp(node a,node b){//自定义比较函数  
 return a.val<b.val;  
 }  
 //1...n的完全二叉树,i的父亲为i/2,i的左孩子为2i,右孩子为2i+1  
 void push(int val){  
 heap[++size].val=val;  
 int i=size;  
 while(i>1&&cmp(heap[i],heap[i/2])){  
 swap(heap[i],heap[i/2]);  
 i/=2;//向上调整  
 }  
 }  
 void pop(){  
 heap[1]=heap[size--];//将最后一个元素放到第一个位置  
 int i=1;  
 while(i\*2<=size){  
 int j=i\*2;  
 if(j<size&&cmp(heap[j+1],heap[j])) j++;//找到左右孩子中较大的一个  
 if(cmp(heap[i],heap[j])) break;//如果父亲节点比孩子节点大,则不需要调整  
 swap(heap[i],heap[j]);  
 i=j;  
 }//向下调整  
 }  
 int top(){  
 return heap[1].val;  
 }  
   
};  
int main()  
{  
 int T\_start=clock();  
 int n;  
 cin>>n;  
 Heap h(n);  
 for(int i=0;i<n;i++){  
 int x;  
 cin>>x;  
 h.push(x);  
 }  
 while(h.size>1){  
 cout<<h.top()<<" ";  
 h.pop();  
 }  
 return 0;  
}

普通莫队

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <unordered\_map>  
#include <array>  
using namespace std;  
class BIT{  
 private:  
 int n;  
 vector<int> tree;//tree[i] 是[i-lowbit(i)+1,i]的和,[1,n]存储  
 int lowbit(int x){  
 return x&(-x);  
 }  
 public:  
 BIT(int n): n(n),tree(n+1,0){}  
 void update(int i,int val)//单点修改 a[i]+=val  
 {  
 while(i<=n){   
 tree[i]+=val;  
 i+=lowbit(i);//跳到后一个lowbit(x)的位置  
 }  
 }  
 int query(int l,int r)//区间查询 [l,r]的和  
 {  
 int res=0;  
 while(r>=l){  
 res+=tree[r];  
 r-=lowbit(r);//跳到前一个lowbit(x)的位置  
 }  
 return res;  
 }  
 void init(vector<int> a)//初始化  
 {  
 vector<int> presum(a.size()+1,0);  
 for(int i=1;i<=a.size();i++)  
 {  
 presum[i]=presum[i-1]+a[i-1];  
 tree[i]=presum[i]-presum[i-lowbit(i)];//按定义  
 }   
 }  
};  
int read()  
{  
 int s=0,f=1;  
 char ch=getchar();  
 while(ch<'0'||ch>'9')  
 {  
 if(ch=='-') f=-1;  
 ch=getchar();  
 }  
 while(ch>='0'&&ch<='9')  
 {  
 s=(s<<3)+(s<<1)+ch-'0';  
 ch=getchar();  
 }  
 return s\*f;  
}  
inline void write(int x)   
{  
 static int sta[35];   
 int top=0;  
 if(x<0&&x!=-2147483648) {putchar('-');x=-x;}  
 if(x==-2147483648) {printf("-2147483648");return;}  
 do{  
 sta[top++]=x%10, x/=10;  
 }while(x);  
 while(top) putchar(sta[--top]+48);  
}  
unordered\_map<int,int> dis(vector<int> a)  
{  
 sort(a.begin(),a.end());  
 unordered\_map<int,int> mp;  
 for(int i=0;i<a.size();i++)  
 {  
 mp[a[i]]=i+1;  
 }  
 return mp;  
}  
int main()  
{  
 int t=read();  
 while(t--)  
 {  
 int n=read(),m=read();  
 vector<int> a(n+1);  
 for(int i=1;i<=n;i++)  
 {  
 a[i]=read();  
 }  
 vector<array<int,4>> q(m);  
 for(int i=0;i<m;i++)  
 {  
 q[i][0]=read();  
 q[i][1]=read();  
 q[i][2]=read();  
 q[i][3]=i;  
 }  
 int block=static\_cast<int>(sqrt(n))+1;//按值域分块  
 auto cmp=[block](array<int,4> a,array<int,4> b)  
 {  
 if(a[0]/block!=b[0]/block) return a[0]/block<b[0]/block;//按块排序  
 else{  
 if(a[0]/block%2==0) return a[1]<b[1];//按值排序  
 else return a[1]>b[1];//按块排序  
 }  
 };  
 sort(q.begin(),q.end(),cmp);  
 vector<int> ans(m,0);  
 int l=1,r=0;  
 BIT bit(n+1);  
 for(auto [ql,qr,x,idx]:q)//暴力  
 {  
 while(r<qr) bit.update(a[++r],1);  
 while(l>ql) bit.update(a[--l],1);  
 while(r>qr) bit.update(a[r--],-1);  
 while(l<ql) bit.update(a[l++],-1);  
 //cout<<l<<" "<<r<<endl;  
 ans[idx]=bit.query(1,a[x])+l-1;  
 }  
 for(auto i:ans) write(i),putchar('\n');  
 }  
 return 0;  
}

李超线段树

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
#define int long long //赫赫 要不要龙龙呢  
using namespace std;  
class LCSegTree{  
public:  
 const int L=1,R=1e9;  
 const double eps=1e-9;  
 struct Line{  
 long double a,b;  
 int id;  
 Line(long double a=0,long double b=/\*-1e18\*/1e18,int id=0):a(a),b(b),id(id){}  
 long double val(int x)const{return (long double)1.0\*a\*x+b;}  
 };  
 struct Node{  
 Line l;  
 Node \*lc,\*rc;  
 Node():l(),lc(nullptr),rc(nullptr){}  
 };  
 Node \*rt;  
 LCSegTree(){  
 rt=nullptr;  
 }  
 bool cmp(long double a,long double b){  
 // return a-b>eps;//max  
 return b-a>eps;//min  
 }  
 bool cmp1(Line a,Line b,int x){  
 long double va=a.val(x),vb=b.val(x);  
 return cmp(va,vb)||(fabs(va-vb)<eps&&a.id<b.id);  
 }  
 bool cmp2(pair<long double,int> a,long double b,int cid){  
 return cmp(a.first,b)||(fabs(a.first-b)<eps&&a.second<cid);  
 }  
 void ins(Node\*&o,int l,int r,int ql,int qr,Line v){  
 if(qr<l||r<ql)return;  
 if(!o)o=new Node();  
 if(ql<=l&&r<=qr){  
 int mid=(l+r)>>1;  
 bool LB=cmp1(v,o->l,l),MB=cmp1(v,o->l,mid),RB=cmp1(v,o->l,r);  
 if(MB)swap(o->l,v);  
 if(l==r)return;  
 if(LB!=MB)ins(o->lc,l,mid,ql,qr,v);  
 else ins(o->rc,mid+1,r,ql,qr,v);  
 return;  
 }  
 int mid=(l+r)>>1;  
 ins(o->lc,l,mid,ql,qr,v);  
 ins(o->rc,mid+1,r,ql,qr,v);  
 }  
 pair<long double,int> qry(Node\*o,int l,int r,int x){  
 if(!o)return {/\*-1e18\*/1e18,0};  
 long double cur=o->l.val(x);  
 int cid=o->l.id;  
 int mid=(l+r)>>1;  
 if(x<=mid){  
 auto res=qry(o->lc,l,mid,x);  
 if(cmp2(res,cur,cid))return res;  
 }else{  
 auto res=qry(o->rc,mid+1,r,x);  
 if(cmp2(res,cur,cid))return res;  
 }  
 return {cur,cid};  
 }  
 void add(int x1,int y1,int x2,int y2,int id){  
 if(x1==x2){  
 int y=max(y1,y2);  
 ins(rt,L,R,x1,x1,Line(0,y,id));  
 }else{  
 if(x1>x2)swap(x1,x2),swap(y1,y2);  
 long double a=1.0\*(y2-y1)/(x2-x1),b=1.0\*y1-a\*x1;  
 ins(rt,L,R,x1,x2,Line(a,b,id));  
 }  
 }  
 void add(Line v){  
 ins(rt,L,R,L,R,v);  
 }  
 pair<long double,int> ask(int x){return qry(rt,L,R,x);}  
};  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
 return 0;  
}  
//李超线段树插入O(logDlogD)查询O(logD)  
//解决：插入一条直线，查询某个点的最大/最小值  
//插入先划分logn区间，再懒标记下放(logn)  
//维护的是直线中点的最大/最小值  
//证明，对每个局部，区间中值最大代表着这个局部最优，于是可以遍历获得全局最优  
//空间复杂度O(nlogDlogD)

珂朵莉树(ODT)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
using namespace std;  
class ODT{  
 public:  
 struct node  
 {  
 int l,r;  
 mutable int val;  
 node(int l,int r,int val):l(l),r(r),val(val){}  
 bool operator<(const node &o)const {  
 return l<o.l;  
 }  
 };  
 set<node> s;  
 //0-based  
 ODT(vector<int> &a){  
 for(int i=0;i<a.size();i++){  
 s.insert(node(i,i,a[i]));  
 }  
 }  
 //把[l,r]区间分割成[l,mid)和[mid,r]两个区间  
 auto split(int pos){  
 auto it=s.lower\_bound(node(pos,0,0));  
 if(it!=s.end()&&it->l==pos) return it;  
 --it;  
 int l=it->l,r=it->r,val=it->val;  
 s.erase(it);  
 s.insert(node(l,pos-1,val));  
 return s.insert(node(pos,r,val)).first;  
 }  
 //把[l,r]区间赋值为val  
 void assign(int l,int r,int val){  
 auto itr=split(r+1),itl=split(l);  
 s.erase(itl,itr);  
 s.insert(node(l,r,val));  
 }  
 //对区间操作  
 void perform(int l,int r)  
 {  
 auto itr=split(r+1),itl=split(l);  
 for(auto it=itl;it!=itr;++it)  
 {  
 //perform   
 }  
 }  
};  
int main()  
{  
 int T\_start=clock();  
   
 return 0;  
}  
//ODT(珂朵莉树)  
//处理区间查询后立即覆盖问题  
//修改/查询的一次为O(logn) 一次查询m个区间 覆盖后最多产生3个区间，并减少m左右个区间  
//一次操作的代价是随机变量di 那么q次操作的期望是 n乘一个小常数  
//所以期望是 均摊O(nlogn)

笛卡尔树

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
using namespace std;  
struct DKRTreeNode{  
 int val;  
 int index;  
 DKRTreeNode \*left, \*right;  
 DKRTreeNode(int x,int i):val(x),index(i),left(NULL),right(NULL){}  
};  
//笛卡尔树具有一下性质：  
//1.二叉搜索树，2.堆  
//i:index,val:nums[i]  
//因为BST的中序遍历一定是原序列，所以新插入的节点一定在右边  
//需要调整最右边的纵向位置，使其满足堆的性质，用单调栈维护最右链  
//大根堆  
DKRTreeNode\* buildTree(vector<int> &nums){  
 stack<DKRTreeNode\*> s;  
 DKRTreeNode\* root=nullptr;  
 for(int i=0;i<nums.size();i++)  
 {  
 DKRTreeNode\* node=new DKRTreeNode(nums[i],i);  
 DKRTreeNode\* last=nullptr;  
 while(!s.empty()&&s.top()->val<node->val)  
 {  
 last=s.top();  
 s.pop();  
 }//单调栈维护  
 if(!s.empty()) s.top()->right=node;//栈顶元素的右子树为node  
 if(last) node->left=last;//栈弹出的元素为node的左子树  
 s.push(node);  
 }  
 while(!s.empty()) root=s.top(),s.pop();//最后一个元素为根节点  
 return root;  
}  
void printTree(DKRTreeNode\* root)  
{  
 if(root==nullptr) return;  
 printTree(root->left);  
 cout<<root->val<<" ";  
 printTree(root->right);  
}  
int main()  
{  
 int T\_start=clock();  
 vector<int> nums(10,0);  
 for(int i=0;i<nums.size();i++) nums[i]=rand()%100;  
 for(auto i:nums) cout<<i<<" ";  
 cout<<endl;  
 DKRTreeNode\* root=buildTree(nums);  
 printTree(root);  
 cout<<root->val<<endl;  
 return 0;  
}

线段树(max&min),tested,class

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
class SegTree{  
 public:  
 pair<int,int> error={-0x3f3f3f3f,0x3f3f3f3f};  
 struct Node{  
 int start;int end;  
 int high;int low;  
 int lazy=0;  
 Node\* left;  
 Node\* right;  
 Node(int start,int end,int high,int low)  
 :start(start),end(end),high(high),  
 low(low),left(nullptr),right(nullptr){}  
 };  
 Node\* root;  
 Node\* Build(vector<int>& nums,int start,int end)  
 {  
 if(start>end) return nullptr;  
 if(start==end) return new Node(start,end,nums[start],nums[start]);  
 int mid=(start+end)>>1;  
 Node\* root=new Node(start,end,-0x3f3f3f3f,0x3f3f3f3f);  
 Node\* leftchild=Build(nums,start,mid);  
 Node\* rightchild=Build(nums,mid+1,end);  
 if(leftchild)   
 {  
 root->left=leftchild;  
 root->high=max(root->high,leftchild->high);  
 root->low=min(root->low,leftchild->low);  
 }  
 if(rightchild)  
 {  
 root->right=rightchild;  
 root->high=max(root->high,rightchild->high);  
 root->low=min(root->low,rightchild->low);  
 }  
 return root;  
 }  
 void init(vector<int>& nums)  
 {  
 root=Build(nums,0,nums.size()-1);  
 return ;  
 }  
   
 void taglazy(Node\* root,int val)  
 {  
 if(root==nullptr) return ;  
 root->low+=val,root->high+=val;  
 root->lazy+=val;  
 }  
 void pushdown(Node\* root)  
 {  
 if(root==nullptr) return ;  
 if(root->lazy!=0)  
 {  
 taglazy(root->left,root->lazy);  
 taglazy(root->right,root->lazy);  
 root->lazy=0;  
 }  
 }  
 void update(Node\* root,int l,int r,int val)  
 {  
 if(root==nullptr) return;  
 if(root->end<l||root->start>r) return ;  
 if(root->start>=l&&root->end<=r)   
 {  
 taglazy(root,val);  
 return;  
 }  
 pushdown(root);  
 update(root->left,l,r,val);  
 update(root->right,l,r,val);  
 if(root->left)   
 {  
 root->high=max(root->left->high,root->high);  
 root->low=min(root->left->low,root->low);  
 }  
 if(root->right)  
 {  
 root->high=max(root->right->high,root->high);  
 root->low=min(root->right->low,root->low);  
 }  
 return ;  
 }  
 void update(int l,int r,int val) {update(root,l,r,val);}  
 pair<int,int> query(Node\* root,int l,int r)  
 {  
 pushdown(root);  
 if(root==nullptr) return error;  
 if(root->end<l||root->start>r) return error;  
 if(root->start>=l&&root->end<=r) return {root->high,root->low};  
 int tpmax=-0x3f3f3f3f,tpmin=0x3f3f3f3f;  
 if(root->left)   
 {  
 pair<int,int> tp1=query(root->left,l,r);  
 tpmax=max(tp1.first,tpmax);  
 tpmin=min(tp1.second,tpmin);  
 }  
 if(root->right)  
 {  
 pair<int,int> tp2=query(root->right,l,r);  
 tpmax=max(tp2.first,tpmax);  
 tpmin=min(tp2.second,tpmin);  
 }  
 return {tpmax,tpmin};  
 }  
 pair<int,int> query(int l,int r) {return query(root,l,r);}  
};  
int main()  
{  
   
 return 0;  
}

线段树(max&min,upd+set),tested,class

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
#define int long long  
class segtree{  
 public:  
 const int H=-0xfffffffffffffff;   
 const int L=0xfffffffffffffff;  
 pair<int,int> error={H,L};  
 struct node{  
 int s;int e;  
 int lazyadd;  
 int lazycover;  
 int lazycoveradd;  
 int high,low;  
 node\* lt;  
 node\* rt;  
 node(int s,int e,int high,int low):  
 s(s),e(e),lazyadd(0),lazycover(0),  
 high(high),low(low),lazycoveradd(-1),  
 lt(nullptr),rt(nullptr){}  
 };  
 node\* root;  
 void inf(node\* a,node\* l,node\* r)  
 {  
 if(!a) return ;  
 if(!l&&r) a->low=r->low,a->high=r->high;  
 if(l&&!r) a->low=l->low,a->high=l->high;  
 if(l&&r) a->low=min(l->low,r->low),a->high=max(l->high,r->high);  
 }  
 node\* build(vector<int> &nums,int l,int r)  
 {  
 if(l>r) return nullptr;  
 if(l==r) return new node(l,r,nums[l],nums[l]);  
 int mid=(l+r)>>1;  
 node\* root=new node(l,r,H,L);  
 node\* lc=build(nums,l,mid);  
 node\* rc=build(nums,mid+1,r);  
 root->lt=lc,root->rt=rc;  
 inf(root,lc,rc);  
 return root;  
 }  
 void init(vector<int> &nums){root=build(nums,0,nums.size()-1);}  
 void taglazy(node\* root,int val,int op)  
 {  
 //op 1 add,2 cover  
 if(!root) return ;  
 if(op==1)  
 {  
 root->lazyadd+=val;  
 root->high+=val;  
 root->low+=val;  
 }  
 if(op==2)  
 {  
 root->lazycover=val;  
 root->lazycoveradd=1;  
 root->high=val;  
 root->low=val;  
 root->lazyadd=0;  
 }  
 }  
 void pushdown(node\* root)  
 {  
 if(!root) return ;  
 if(root->lazycoveradd==1)  
 {  
 taglazy(root->lt,root->lazycover,2);  
 taglazy(root->rt,root->lazycover,2);  
 root->lazycover=0;  
 root->lazycoveradd=-1;  
 }  
 if(root->lazyadd)  
 {  
 taglazy(root->lt,root->lazyadd,1);  
 taglazy(root->rt,root->lazyadd,1);  
 root->lazyadd=0;  
 }  
 }  
 pair<int,int> query(node\* root,int l,int r)  
 {  
 pushdown(root);  
 if(!root) return error;  
 if(root->s>r||root->e<l) return error;  
 if(l<=root->s&&root->e<=r)   
 {  
 //cout<<"query:"<<root->s<<' '<<root->e<<' '<<root->high<<' '<<root->low<<endl;  
 return {root->high,root->low};  
 }  
 pair<int,int> tplt=query(root->lt,l,r);  
 pair<int,int> tprt=query(root->rt,l,r);  
 int anshigh=max(tplt.first,tprt.first);  
 int anslow=min(tplt.second,tprt.second);  
 return {anshigh,anslow};  
 }  
 pair<int,int> query(int l,int r) {return query(root,l,r);}  
 void update(node\* root,int l,int r,int val,int op)  
 {  
 if(!root) return ;  
 if(root->s>r||root->e<l) return ;  
 if(l<=root->s&&root->e<=r)  
 {  
 //cout<<"taglazy:"<<root->s<<' '<<root->e<<' '<<val<<' '<<op<<endl;  
 taglazy(root,val,op);  
 return;  
 }  
 pushdown(root);  
 update(root->lt,l,r,val,op);  
 update(root->rt,l,r,val,op);  
 inf(root,root->lt,root->rt);  
 return ;  
 }  
 void update(int l,int r,int val,int op){  
 update(root,l,r,val,op);  
 }  
  
};  
int read()  
{  
 int s=0,f=1;  
 char ch=getchar();  
 while(ch<'0'||ch>'9')  
 {  
 if(ch=='-') f=-1;  
 ch=getchar();  
 }  
 while(ch>='0'&&ch<='9')  
 {  
 s=(s<<3)+(s<<1)+ch-'0';  
 ch=getchar();  
 }  
 return s\*f;  
}  
inline void write(int x)   
{  
 static int sta[35];   
 int top=0;  
 if(x<0&&x!=-2147483648) {putchar('-');x=-x;}  
 if(x==-2147483648) {printf("-2147483648");return;}  
 do{  
 sta[top++]=x%10, x/=10;  
 }while(x);  
 while(top) putchar(sta[--top]+48);  
}  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 int n=read(),m=read();  
 vector<int>a(n);  
 for(auto&i:a) i=read();  
 segtree st;st.init(a);  
 while(m--)  
 {  
 int op=read(),l=read(),r=read();  
 if(op==1)  
 {  
 int x=read();  
 st.update(l-1,r-1,x,2);  
 }  
 else if(op==2)  
 {  
 int x=read();  
 st.update(l-1,r-1,x,1);  
 }  
 else printf("%lld\n",st.query(l-1,r-1).first);  
 // for(int i=0;i<n;i++)  
 // {  
 // cout<<st.query(i,i).first<<' ';  
 // }  
 // cout<<endl;  
 }   
 return 0;  
}

线段树(sum),tested ,class

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
class SegTree{  
 public:  
 struct Node  
 {  
 int sum;  
 int s,e;  
 int lazy=0;  
 Node\* lt;  
 Node\* rt;  
 Node(int sum,int s,int e):s(s),e(e),sum(sum),lt(nullptr),rt(nullptr){}  
 };  
 Node\* root;  
 Node\* buildtree(vector<int> &nums,int l,int r)  
 {  
 if(l>r) return nullptr;  
 if(l==r) return new Node(nums[l],l,l);  
 int mid=(l+r)>>1;  
 Node\* root=new Node(0,l,r);  
 Node\* lc=buildtree(nums,l,mid);  
 Node\* rc=buildtree(nums,mid+1,r);  
 if(lc) root->lt=lc,root->sum+=lc->sum;  
 if(rc) root->rt=rc,root->sum+=rc->sum;  
 return root;  
 }  
 void init(vector<int>nums)  
 {  
 root=buildtree(nums,0,nums.size()-1);  
 return;  
 }  
 void taglazy(Node\* root,int val)  
 {  
 if(root==nullptr) return;  
 root->lazy+=val;  
 root->sum+=(root->e-root->s+1)\*val;  
 }  
 void pushdown(Node\* root)  
 {  
 if(!root) return ;  
 if(root->lazy)  
 {  
 taglazy(root->lt,root->lazy);  
 taglazy(root->rt,root->lazy);  
 root->lazy=0;  
 }  
 }  
 void update(Node\* root,int l,int r,int val)  
 {  
 if(!root) return ;  
 if(root->s>r||root->e<l) return ;  
 if(root->s>=l&&root->e<=r)  
 {  
 taglazy(root,val);  
 return;  
 }  
 pushdown(root);  
 update(root->lt,l,r,val);  
 update(root->rt,l,r,val);  
 root->sum=((root->lt?root->lt->sum:0)+(root->rt?root->rt->sum:0));  
 return ;  
 }  
 void update(int l,int r,int val)  
 {  
 update(root,l,r,val);  
 return ;  
 }  
 int query(Node\* root,int l,int r)  
 {  
 pushdown(root);  
 if(!root) return 0;  
 if(root->s>r||root->e<l) return 0;  
 if(root->s>=l&&root->e<=r) return root->sum;  
 return query(root->lt,l,r)+query(root->rt,l,r);  
 }  
 int query(int l,int r)  
 {  
 return query(root,l,r);  
 }  
};  
int main()  
{  
   
 return 0;  
}

线段树(sum,update+set),tested,class

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#define int long long  
using namespace std;  
class segtree{  
 public:  
 struct node  
 {  
 int sum;  
 int s;int e;  
 node\* lt;  
 node\* rt;  
 int lazysum=0;  
 int lazycover=0;  
 node(int sum,int s,int e):   
 sum(sum),s(s),e(e),lt(nullptr),  
 rt(nullptr),lazycover(0),lazysum(0){}  
 };  
 node\* root;  
 node\* build(vector<int>& nums,int l,int r)  
 {  
 if(l>r) return nullptr;  
 if(l==r) return new node(nums[l],l,l);  
 int mid=(l+r)>>1;  
 node\* root=new node(0,l,r);  
 node\* lc=build(nums,l,mid);  
 node\* rc=build(nums,mid+1,r);  
 root->lt=lc,root->rt=rc;  
 root->sum=(lc?lc->sum:0)+(rc?rc->sum:0);  
 return root;  
 }  
 void init(vector<int>& nums) {root=build(nums,0,nums.size()-1);}  
 void taglazy(node\* root,int val,int op){  
 //1:sum,2:cover  
 if(!root) return;  
 if(op==1)  
 {  
 root->lazysum+=val;  
 root->sum+=(val)\*(root->e-root->s+1);  
 }  
 else if(op==2)  
 {  
 root->lazycover=val;  
 root->lazysum=0;  
 root->sum=val\*(root->e-root->s+1);  
 }  
 }  
 void pushdown(node\* root)  
 {  
 if(root->lazycover){  
 taglazy(root->lt,root->lazycover,2);  
 taglazy(root->rt,root->lazycover,2);  
 root->lazycover=0;  
 }  
 if(root->lazysum){  
 taglazy(root->lt,root->lazysum,1);  
 taglazy(root->rt,root->lazysum,1);  
 root->lazysum=0;  
 }  
 }  
 int query(node\* root,int l,int r)  
 {  
 pushdown(root);  
 if(!root) return 0;  
 if(root->s>r||root->e<l) return 0;  
 if(root->s>=l&&root->e<=r) return root->sum;  
 return query(root->lt,l,r)+query(root->rt,l,r);  
 }  
 int query(int l,int r) {return query(root,l,r);}  
 void update(node\* root,int l,int r,int val,int op)  
 {  
   
 if(!root) return ;  
 if(root->s>r||root->e<l) return ;  
 if(root->s>=l&&root->e<=r)   
 {  
 taglazy(root,val,op);  
 return ;  
 }  
 pushdown(root);  
 update(root->lt,l,r,val,op);  
 update(root->rt,l,r,val,op);  
 root->sum=(root->lt?root->lt->sum:0)+(root->rt?root->rt->sum:0);  
 return ;  
 }  
 void update(int l,int r,int val,int op){  
 update(root,l,r,val,op);  
 }  
  
};  
int read()  
{  
 int s=0,f=1;  
 char ch=getchar();  
 while(ch<'0'||ch>'9')  
 {  
 if(ch=='-') f=-1;  
 ch=getchar();  
 }  
 while(ch>='0'&&ch<='9')  
 {  
 s=(s<<3)+(s<<1)+ch-'0';  
 ch=getchar();  
 }  
 return s\*f;  
}  
inline void write(int x)   
{  
 static int sta[35];   
 int top=0;  
 if(x<0&&x!=-2147483648) {putchar('-');x=-x;}  
 if(x==-2147483648) {printf("-2147483648");return;}  
 do{  
 sta[top++]=x%10, x/=10;  
 }while(x);  
 while(top) putchar(sta[--top]+48);  
}  
signed main()  
{  
   
 return 0;  
}

# Graph

2-sat

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
using namespace std;  
pair<vector<int>,int> tarjan(vector<vector<int>> &mp,int n)  
{  
 vector<int> bel(n+1,-1);//bel[i]:i属于哪个强连通分量  
 vector<int> dfn(n+1,-1),low(n+1,-1);  
 stack<int> st;int cnt=0,scc\_cnt=0;  
 auto dfs=[&](auto dfs,int u)->void{  
 dfn[u]=low[u]=++cnt; //时间戳+1  
 st.push(u); //inst[u]=1; //入栈  
 for(int v:mp[u])  
 {  
 if(dfn[v]==-1)//case1:u的邻接点v未被访问过  
 {  
 dfs(dfs,v);  
 low[u]=min(low[u],low[v]);  
 }  
 else if(bel[v]==-1)//v所属的强连通分量还未被确定（等价于case2）  
 {  
 low[u]=min(low[u],dfn[v]);  
 }  
 //case3:u的邻接点v不在栈中,且访问过  
 //说明v已经确定在某个强连通分量中，所以u的low不需要更新  
 }  
 if(dfn[u]==low[u])  
 {  
 scc\_cnt++;  
 while(true)  
 {  
 int v=st.top();  
 st.pop();  
 bel[v]=scc\_cnt;  
 if(v==u) break;  
 }  
 }  
 };  
 //图有可能不是强联通的  
 for(int i=1;i<=n;i++)  
 {  
 if(dfn[i]==-1)  
 {  
 dfs(dfs,i);  
 }  
 }  
 return {bel,scc\_cnt};  
}  
int main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 int n,m;cin>>n>>m;  
 vector<vector<int>> mp(2\*n+1);  
 for(int i=0;i<m;i++)  
 {  
 int u,b1,v,b2;  
 cin>>u>>b1>>v>>b2;  
 //u=b1 or v=b2  
 //=>u!=b1->v=b2 and v!=b2->u=b1  
 mp[u+(!b1)\*n].push\_back(v+b2\*n);  
 mp[v+(!b2)\*n].push\_back(u+b1\*n);  
 //u=b1-> u+b1\*n x  
 }  
 auto [bel,scc\_cnt]=tarjan(mp,2\*n);  
 vector<int> ans(n+1,-1);  
 int flag=1;  
 for(int i=1;i<=n;i++)  
 {  
 if(bel[i]==bel[i+n]) {flag=0;break;}  
 else ans[i]=bel[i]>bel[i+n];  
 }  
 //此处处理的是i的正确性  
 //当bel[u==0]>bel[u==1]时,u==0的拓扑序小,i应当被赋值为false  
 //因为i->!i为永真式的前提为i=0  
 //此处i的含义是命题变元i的取值=0  
 //所以ans[i]=1  
 if(flag)  
 {  
 cout<<"POSSIBLE"<<endl;  
 for(int i=1;i<=n;i++)  
 {  
 cout<<ans[i]<<" ";  
 }  
 cout<<endl;  
 }  
 else cout<<"IMPOSSIBLE"<<endl;  
 return 0;  
}  
//2-sat  
//处理n个命题变元的赋值问题，形式上判断形如(p->q) and (!p->q)是否可永真赋值  
//即判断是否存在一种赋值使得p->q和!p->q同时为真  
//很显然若p->q,q->p均成立,则p,q在一个scc里

BCC（点双连通分量，tarjan）

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
vector<vector<int>> tarjan(vector<vector<int>> &mp,int n)  
{  
 //点双连通分量：无割点，且任意两点间至少有两条路径  
 vector<int> low(n+1,-1),dfn(n+1,-1);  
 //low:从当前点出发能到达的最早时间戳  
 //dfn:当前点的时间戳  
 vector<vector<int>> bccs;  
 stack<int> st;  
 int cnt=0;  
 auto dfs=[&](auto dfs,int u,int fa)->void{  
 int ch=0; //儿子数  
 dfn[u]=low[u]=++cnt;  
 st.push(u);  
 for(auto v:mp[u])  
 {  
 if(dfn[v]==-1)//case1:未访问  
 {  
 ch++;  
 dfs(dfs,v,u);  
 low[u]=min(low[u],low[v]);//更新low[u]  
 if((fa==-1&&ch>1)||(fa!=-1&&low[v]>=dfn[u]))//是割点,v以及他的被处理过的子树是一个bcc  
 {  
 vector<int> bcc;  
 while(1)  
 {  
 int x=st.top();st.pop();  
 bcc.push\_back(x);  
 if(x==v)break;//处理到v  
 }  
 bcc.push\_back(u);//把割点也加入bcc:割点有可能在多个bcc中  
 bccs.push\_back(bcc);  
 }  
 }  
 else if(v!=fa)//case2:已访问且不是父节点  
 {  
 low[u]=min(low[u],dfn[v]);//更新low[u]  
 }  
 }  
 // if(fa==-1&&ch==0) {  
 // bccs.push\_back({u});  
 // }  
 };  
 for(int i=1;i<=n;i++)  
 {  
 if(dfn[i]==-1)  
 {  
 dfs(dfs,i,-1);  
 //处理剩下的bcc  
 vector<int> bcc;  
 while(!st.empty())  
 {  
 int x=st.top();st.pop();  
 bcc.push\_back(x);  
 }  
 if(!bcc.empty()) bccs.push\_back(bcc);  
 }  
 }  
 return bccs;  
}  
//无向图中割点：删除该点后，图的bcc数增加  
//一个图中割点的判断  
//1.对于某个顶点 u，如果存在至少一个顶点 v（u 的儿子），使得low[v]>=dfn[u] ，即只能回到祖先（到不了dfn更早的点），那么 u 点为割点。  
//2.对于搜索的起始点，如果它的儿子数大于等于 2，那么它就是割点。  
int main()  
{  
 int T\_start=clock();  
 int n,m;cin>>n>>m;  
 vector<vector<int>> mp(n+1);  
 vector<int> val(n+1);  
 for(int i=0;i<m;i++)  
 {  
 int u,v;cin>>u>>v;  
 mp[u].push\_back(v);  
 mp[v].push\_back(u);  
 }  
 vector<vector<int>> bccs=tarjan(mp,n);  
 cout<<bccs.size()<<endl;  
 for(auto bcc: bccs)  
 {  
 cout<<bcc.size()<<' ';  
 for(auto x: bcc)  
 {  
 cout<<x<<" ";  
 }  
 cout<<endl;  
 }  
 return 0;  
}

DSU on tree(树上启发式合并)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
//#define int long long //赫赫 要不要龙龙呢  
using namespace std;  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
 int n,m;cin>>n;  
 vector<int> sz(n+5,0),hson(n+5,0),fa(n+5,0);  
 vector<int> dfn(n+5,0),id(n+5,0),out(n+5,0);  
 //子树大小,重儿子,父节点,dfs序,dfs序对应的节点,出栈序  
 vector<int> c(n+5,0),s(n+5,0),ans(n+5,0);  
 vector<vector<int>> tr(n+5);  
 int tot=0,cnt=0;  
 for(int i=1;i<n;i++)  
 {  
 int u,v;cin>>u>>v;  
 tr[u].push\_back(v);  
 tr[v].push\_back(u);  
 }  
 for(int i=1;i<=n;i++) cin>>c[i];  
 auto dfs1=[&](this auto&& dfs1,int u,int f)->void{  
 dfn[u]=++tot;  
 id[tot]=u,sz[u]=1;  
 for(auto v:tr[u])  
 {  
 if(v==f) continue;  
 dfs1(v,u);  
 sz[u]+=sz[v];  
 if(sz[v]>sz[hson[u]]) hson[u]=v;  
 }  
 out[u]=tot;  
 };//预处理一些东西  
 dfs1(1,0);  
 auto dfs2=[&](this auto& dfs2,int u,int f,bool keep)->void{  
 for(auto v:tr[u]) //先遍历轻儿子，不保留其对集合的影响  
 {  
 if(v==f||v==hson[u]) continue;  
 dfs2(v,u,0);  
 }  
 if(hson[u]) dfs2(hson[u],u,1);// 然后遍历重儿子，保留其对集合的影响  
 if(!s[c[u]]) ++cnt,s[c[u]]=1; // 加入根结点对集合的贡献  
 for(auto v:tr[u])  
 {  
 if(v==f||v==hson[u]) continue;  
 for(int i=dfn[v];i<=out[v];i++) //遍历轻儿子的子树  
 {  
 int x=id[i];  
 if(!s[c[x]]) ++cnt,s[c[x]]=1; //加入轻儿子的贡献  
 }  
 }  
 ans[u]=cnt;  
 if(!keep) //如果当前节点不是重儿子，则撤销当前节点的贡献  
 {  
 for(int i=dfn[u];i<=out[u];i++)  
 {  
 int x=id[i];  
 s[c[x]]=0;  
 }  
 cnt=0;  
 }  
 };  
 dfs2(1,0,1);  
 cin>>m;  
 for(int i=1;i<=m;i++)  
 {  
 int x;cin>>x;  
 cout<<ans[x]<<'\n';  
 }  
 return 0;  
}  
//树上启发式合并(dsu on tree)  
//时间复杂度O(nlogn)  
//考虑将树上的问题转化为集合合并信息的问题  
//想到子树，就想到dsu on tree  
//考虑把小集合合并到大集合里，这样小集合的大小至少变成原来的两倍，这样合并的次数就变少了  
//当然可以不用dsu on tree 来暴力合并，不过要一些数据结构支持/时空复杂度会多一个log  
//这边加上-撤销贡献的操作是为了保证空间复杂度  
//同时保证了此时s数组是空的，所以不会影响后续的合并操作  
//注意：这里撤销贡献的操作是必须的，如果不撤销贡献，空间复杂度会退化到O(n^2)  
//考虑时间复杂度证明  
//考虑一个节点被作为轻儿子做出贡献的次数  
//实际上就转化为从该节点到根节点路径上的轻边数量  
//why?路径上有轻边意味着该节点作为轻儿子的子树被合并到该节点上  
//撤销操作和合并操作是互反的  
//所以轻边数量就是该节点被作为轻儿子做出贡献的次数  
//根据HLD的结论：该节点到根节点路径上的轻边数量<logn  
//所以时间复杂度是O(nlogn),Q.E.D

EDCC (边双联通分量，tarjan）

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
vector<vector<int>> tarjan(vector<vector<pair<int,int>>> &mp,int n)  
{  
 //边双联通分量：无向图中，边双联通分量是指一个极大子图，删除该子图中的任意一条边，该子图仍然连通  
 //连接边双联通分量的边称为桥  
 vector<int> low(n+1,-1),dfn(n+1,-1);  
 //low:从当前点出发能到达的最早时间戳  
 //dfn:当前点的时间戳  
 vector<vector<int>> dccs;  
 stack<int> st; int cnt=0;  
 auto dfs=[&](auto dfs,int u,int fre)->void{  
 //fre:来时的边  
 dfn[u]=low[u]=++cnt;  
 st.push(u);  
 for(auto [v,rev]:mp[u])  
 {  
 if(dfn[v]==-1)//case1:未访问  
 {  
 dfs(dfs,v,rev);  
 low[u]=min(low[u],low[v]);//更新low[u]  
 }  
 else if(rev!=(fre^1))//case2:已访问且不是该边的反边  
 {  
 //阻断向父节点更新的可能  
 //多重边可能有一种特殊的组合让v!=fa失效  
 //eg.1-2,1-2,2-3,2-3  
 low[u]=min(low[u],dfn[v]);//更新low[u]  
 }  
 }  
 if(dfn[u]==low[u])//case3:u的子树中不存在能到达u的祖先的边  
 //u的子树全为dcc  
 {  
 vector<int> dcc;  
 while(true){  
 int t=st.top();st.pop();  
 dcc.push\_back(t);  
 if(t==u) break;  
 }  
 dccs.push\_back(dcc);  
 }  
 };  
 for(int i=1;i<=n;i++)  
 {  
 if(dfn[i]==-1)  
 {  
 dfs(dfs,i,-1);  
 }  
 }  
 return dccs;  
}  
int main()  
{  
 int T\_start=clock();  
 int n,m;cin>>n>>m;  
 vector<vector<pair<int,int>>> mp(n+1);  
 vector<int> val(n+1);  
 int tot=0;  
 for(int i=0;i<m;i++)  
 {  
 int u,v;cin>>u>>v;  
 if(u==v) continue;  
 mp[u].push\_back({v,tot+1});  
 mp[v].push\_back({u,tot});  
 tot+=2;//存各自的边的编号  
 }  
 vector<vector<int>> bccs=tarjan(mp,n);  
 cout<<bccs.size()<<endl;  
 for(auto bcc: bccs)  
 {  
 cout<<bcc.size()<<' ';  
 for(auto x: bcc)  
 {  
 cout<<x<<" ";  
 }  
 cout<<endl;  
 }  
 return 0;  
}

SCC（低注释）

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
pair<vector<int>,int> tarjan(vector<vector<int>> &mp,int n)  
{  
 vector<int> bel(n+1,-1);//bel[i]:i属于哪个强连通分量  
 vector<int> dfn(n+1,-1),low(n+1,-1);  
 stack<int> st;int cnt=0,scc\_cnt=0;  
 auto dfs=[&](auto dfs,int u)->void{  
 dfn[u]=low[u]=++cnt; //时间戳+1  
 st.push(u); //inst[u]=1; //入栈  
 for(int v:mp[u])  
 {  
 if(dfn[v]==-1)//case1:u的邻接点v未被访问过  
 {  
 dfs(dfs,v);  
 low[u]=min(low[u],low[v]);  
 }  
 else if(bel[v]==-1)//v所属的强连通分量还未被确定（等价于case2）  
 {  
 low[u]=min(low[u],dfn[v]);  
 }  
 //case3:u的邻接点v不在栈中,且访问过  
 //说明v已经确定在某个强连通分量中，所以u的low不需要更新  
 }  
 if(dfn[u]==low[u])  
 {  
 scc\_cnt++;  
 while(true)  
 {  
 int v=st.top();  
 st.pop();  
 bel[v]=scc\_cnt;  
 if(v==u) break;  
 }  
 }  
 };  
 //图有可能不是强联通的  
 for(int i=1;i<=n;i++)  
 {  
 if(dfn[i]==-1)  
 {  
 dfs(dfs,i);  
 }  
 }  
 return {bel,scc\_cnt};  
}  
int main()  
{  
 int T\_start=clock();  
 int n,m;cin>>n>>m;  
 vector<vector<int>> mp(n+1);  
 vector<int> val(n+1);  
 for(int i=1;i<=n;i++) cin>>val[i];  
 for(int i=0;i<m;i++)  
 {  
 int u,v;cin>>u>>v;  
 mp[u].push\_back(v);  
 }  
 auto [bel,cnt]=tarjan(mp,n);  
 vector<vector<int>> mp2(cnt+1);  
 vector<int> val2(cnt+1,0);  
 vector<int> in(cnt+1,0);  
 vector<int> dp(cnt+1,0);  
 for(int i=1;i<=n;i++)  
 {  
 val2[bel[i]]+=val[i];  
 }  
 for(int i=1;i<=n;i++)  
 {  
 for(int v:mp[i])  
 {  
 if(bel[i]!=bel[v])  
 {  
 mp2[bel[i]].push\_back(bel[v]);  
 in[bel[v]]++;  
 }  
 }  
 }  
 queue<int> q;  
 for(int i=1;i<=cnt;i++)  
 {  
 if(in[i]==0) q.push(i),dp[i]=val2[i];  
 }  
 while(!q.empty())  
 {  
 int u=q.front();q.pop();  
 for(int v:mp2[u])  
 {  
 dp[v]=max(dp[v],dp[u]+val2[v]);  
 in[v]--;  
 if(in[v]==0) q.push(v);  
 }  
 }  
 cout<<\*max\_element(dp.begin()+1,dp.end())<<endl;  
 return 0;  
}

SCC（强联通分量，缩点，tarjan）

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
///vector<vector<int>> tarjan(vector<vector<int>> &mp,int n,int m)  
pair<vector<int>,int> tarjan(vector<vector<int>> &mp,int n)  
{  
 //求强连通分量，强连通分量是有向图中的极大顶点子集，其中任意两个顶点都是互相可达的  
 ///vector<vector<int>> scc;//强连通分量  
 vector<int> bel(n+1,-1);//bel[i]:i属于哪个强连通分量  
 vector<int> dfn(n+1,-1),low(n+1,-1);  
 //vector<int> inst(n+1,0);  
 //dfn:时间戳（dfs序），low:从i开始能到达的最小时间戳，inst:是否在栈中  
 stack<int> st;int cnt=0,scc\_cnt=0;  
 //st:未放到scc的点，cnt：计时器，初始为0  
 auto dfs=[&](auto dfs,int u)->void{  
 dfn[u]=low[u]=++cnt; //时间戳+1  
 st.push(u); //inst[u]=1; //入栈  
 for(int v:mp[u])  
 {  
 if(dfn[v]==-1)//case1:u的邻接点v未被访问过  
 {  
 dfs(dfs,v);  
 low[u]=min(low[u],low[v]);//用v的low更新u的low  
 }  
 // else if(inst[v])//case2:u的邻接点v在栈中,且访问过  
 // {  
 // low[u]=min(low[u],dfn[v]);  
 // //有可能存在一个环  
 // }  
 else if(bel[v]==-1)//v所属的强连通分量还未被确定（等价于case2）  
 {  
 low[u]=min(low[u],dfn[v]);  
 }  
 //case3:u的邻接点v不在栈中,且访问过  
 //说明v已经确定在某个强连通分量中，所以u的low不需要更新  
 }  
 if(dfn[u]==low[u])  
 {  
 //u是某个强连通分量的根（第一个被访问的结点）  
 //why,low[u]==dfn[u]说明u没有指向自己的边，所以u是某个强连通分量的根  
 //某个强连通分量的根的low值不会被更新  
 // vector<int> s;  
 // while(true)  
 // {  
 // int v=st.top();  
 // st.pop();inst[v]=0;  
 // s.push\_back(v);  
 // if(v==u) break;  
 // }  
 // scc.push\_back(s);  
 scc\_cnt++;  
 while(true)  
 {  
 int v=st.top();  
 st.pop();  
 bel[v]=scc\_cnt;  
 if(v==u) break;  
 }  
 }  
 };  
 //图有可能不是强联通的  
 for(int i=1;i<=n;i++)  
 {  
 if(dfn[i]==-1)  
 {  
 dfs(dfs,i);  
 }  
 }  
 //return scc;  
 return {bel,scc\_cnt};  
}  
//证明：如果结点 u 是某个强连通分量在搜索树中遇到的第一个结点，那么这个强连通分量的其余结点肯定是在搜索树中以 u 为根的子树中。  
//结点 u 被称为这个强连通分量的根。  
//反证法：假设有个结点 v 在该强连通分量中但是不在以 u 为根的子树中，那么 u 到 v 的路径中肯定有一条离开子树的边。  
//但是这样的边只可能是横叉边或者反祖边，然而这两条边都要求指向的结点已经被访问过了，这就和 v 不在以 u 为根的子树中矛盾了。得证。  
//其实手玩一下，若有个结点 v 在该强连通分量中但是不在以 u 为根的子树中，他的与u形成的那个环，其实是v一定在u的子树中，从u的dfs一定能够遍历到v  
//画个图就知道了  
//并且，很容易想到，对于一个连通分量图，有且只有一个根，即第一个被访问的结点  
//所以算法正确性显然  
//时间复杂度：O(n+m)  
int main()  
{  
 int T\_start=clock();  
 int n,m;cin>>n>>m;  
 vector<vector<int>> mp(n+1);  
 vector<int> val(n+1);  
 for(int i=1;i<=n;i++) cin>>val[i];  
 for(int i=0;i<m;i++)  
 {  
 int u,v;cin>>u>>v;  
 mp[u].push\_back(v);  
 }  
 auto [bel,cnt]=tarjan(mp,n);  
 vector<vector<int>> mp2(cnt+1);  
 vector<int> val2(cnt+1,0);  
 vector<int> in(cnt+1,0);  
 vector<int> dp(cnt+1,0);  
 for(int i=1;i<=n;i++)  
 {  
 val2[bel[i]]+=val[i];  
 }  
 for(int i=1;i<=n;i++)  
 {  
 for(int v:mp[i])  
 {  
 if(bel[i]!=bel[v])  
 {  
 mp2[bel[i]].push\_back(bel[v]);  
 in[bel[v]]++;  
 }  
 }  
 }  
 queue<int> q;  
 for(int i=1;i<=cnt;i++)  
 {  
 if(in[i]==0) q.push(i),dp[i]=val2[i];  
 }  
 while(!q.empty())  
 {  
 int u=q.front();q.pop();  
 for(int v:mp2[u])  
 {  
 dp[v]=max(dp[v],dp[u]+val2[v]);  
 in[v]--;  
 if(in[v]==0) q.push(v);  
 }  
 }  
 cout<<\*max\_element(dp.begin()+1,dp.end())<<endl;  
 return 0;  
}

ShortestPath（Bellman\_Ford,SPFA)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
int read()  
{  
 int s=0,f=1;  
 char ch=getchar();  
 while(ch<'0'||ch>'9')  
 {  
 if(ch=='-') f=-1;  
 ch=getchar();  
 }  
 while(ch>='0'&&ch<='9')  
 {  
 s=(s<<3)+(s<<1)+ch-'0';  
 ch=getchar();  
 }  
 return s\*f;  
}  
inline void write(int x)   
{  
 static int sta[35];   
 int top=0;  
 if(x<0&&x!=-2147483648) {putchar('-');x=-x;}  
 if(x==-2147483648) {printf("-2147483648");return;}  
 do{  
 sta[top++]=x%10, x/=10;  
 }while(x);  
 while(top) putchar(sta[--top]+48);  
}  
vector<int> Bellman\_Ford(vector<vector<pair<int,int>>>& mp,int s,int n)  
{  
 vector<int> dis(n+1,0x7fffffff);  
 dis[s]=0;  
 for(int i=1;i<=n-1;i++)  
 {  
 for(int j=1;j<=n;j++)  
 {  
 for(auto [u,w]:mp[j])  
 {  
 if(dis[j]!=0x7fffffff&&dis[j]+w<dis[u])  
 {  
 dis[u]=dis[j]+w;  
 }  
 }  
 }   
 }  
 for(int j=1;j<=n;j++)  
 {  
 for(auto [u,w]:mp[j])  
 {  
 if(dis[j]!=0x7fffffff&&dis[j]+w<dis[u])  
 {  
 cout<<"negative cycle!"<<endl;  
 }  
 }  
 }  
 return dis;  
 //Bellman\_Ford 对所有的边进行n-1次松弛操作，如果在进行第n次松弛操作时，仍然存在边可以松弛，则说明图中存在负权环（从s点出发存在负权环）  
 //时间复杂度：O(nm),形式上就是暴力）  
 //第i次循环，我们能找到经历i条边到达的点的最短距离  
 //所以第n次循环，我们能找到经历n条边到达的点的最短距离，如果存在负权环，那么一定能在第n次循环找到经历n条边到达的点的最短距离  
}  
vector<int> SPFA(vector<vector<pair<int,int>>>& mp,int s,int n)  
{  
 vector<int> dis(n+1,0x7fffffff);  
 vector<int> vis(n+1,0);  
 vector<int> cnt(n+1,0);  
 dis[s]=0;queue<int> q;  
 q.push(s);vis[s]=1;cnt[s]=0;  
 while(!q.empty())  
 {  
 int u=q.front();  
 q.pop();vis[u]=0;  
 for(auto [v,w]:mp[u])  
 {  
 if(dis[v]>dis[u]+w)//松弛  
 {  
 dis[v]=dis[u]+w;  
 cnt[v]=cnt[u]+1;  
 if(cnt[v]>n-1)//存在负权环  
 {  
 //1-n的节点，最短路最多经过n-1条边，如果经过n条边，说明存在负权环  
 cout<<"negative cycle!"<<endl;  
 return dis;  
 }  
 if(!vis[v])  
 {  
 q.push(v);  
 vis[v]=1;  
 }  
 }  
 }  
 }  
 return dis;  
 //SPFA: 形式上Bellman\_Ford是一棵树，很显然，只有上一次被松弛的节点u，才有可能对v进行松弛，所以可以采用SPFA  
 //为啥只有上一次被松弛的节点u，才有可能对v进行松弛？  
 //手玩一下就好了（悲  
 //考虑简单图，他可以是个递推的过程  
}  
int main()  
{  
 int T\_start=clock();  
 int n=read(),m=read();  
 vector<vector<pair<int,int>>> mp(n+1);  
 for(int i=0;i<m;i++)  
 {  
 int u=read(),v=read(),w=read();  
 mp[u].push\_back({v,w});  
 }  
 return 0;  
}

ShortestPath（Floyed）

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
using namespace std;  
const int MAXN=1e4;  
int Graph[MAXN][MAXN];  
int dp[MAXN][MAXN];  
int main()  
{  
 int T\_start=clock();  
 int T=10;  
 srand(time(NULL));  
 for(int i=1;i<=T;i++)  
 {  
 for(int j=1;j<=T;j++)  
 {  
 int op=rand()%3;  
 if(false) Graph[i][j]=0;  
 else if(op==1) Graph[i][j]=0x3f3f3f3f;  
 else Graph[i][j]=(rand()\*10+rand())%10;  
 //Graph[i][j]=(rand()\*10+rand())%10;  
 }  
 Graph[i][i]=0;  
 }  
  
   
 for(int i=1;i<=T;i++)  
 {  
 for(int j=1;j<=T;j++)  
 {  
 dp[i][j]=Graph[i][j];  
 }  
 }  
 for(int i=1;i<=T;i++)  
 {  
 for(int j=1;j<=T;j++)  
 {  
 cout<<dp[i][j]<<" ";  
 }  
 cout<<endl;  
 }  
 cout<<endl;  
 for(int i=1;i<=T;i++)  
 {  
 for(int j=1;j<=T;j++)  
 {  
 for(int k=1;k<=T;k++)  
 {  
 dp[j][k]=min(dp[j][k],dp[j][i]+dp[i][k]);  
 }  
 }  
 }  
  
 for(int i=1;i<=T;i++)  
 {  
 for(int j=1;j<=T;j++)  
 {  
 cout<<dp[i][j]<<" ";  
 }  
 cout<<endl;  
 }  
 int T\_end=clock();  
 return 0;  
}  
  
//Floyed 算法  
//处理任意两点之间的最短路径(无负环）  
//时间复杂度O(n^3)

ShortestPath（dijkstra）

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
int read()  
{  
 int s=0,f=1;  
 char ch=getchar();  
 while(ch<'0'||ch>'9')  
 {  
 if(ch=='-') f=-1;  
 ch=getchar();  
 }  
 while(ch>='0'&&ch<='9')  
 {  
 s=(s<<3)+(s<<1)+ch-'0';  
 ch=getchar();  
 }  
 return s\*f;  
}  
inline void write(int x)   
{  
 static int sta[35];   
 int top=0;  
 if(x<0&&x!=-2147483648) {putchar('-');x=-x;}  
 if(x==-2147483648) {printf("-2147483648");return;}  
 do{  
 sta[top++]=x%10, x/=10;  
 }while(x);  
 while(top) putchar(sta[--top]+48);  
}  
vector<int> dijkstra(int n,vector<vector<pair<int,int>>>mp,int s)  
{  
 vector<int> dis(n+1,0x7fffffff);//初始化距离为无穷大  
 dis[s]=0;//起点到起点的距离为0  
 priority\_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>> pq;  
 pq.push({0,s});//将起点加入优先队列  
 while(!pq.empty())  
 {  
 int u=pq.top().second;//取出当前距离最小的点  
 int d=pq.top().first;//取出当前距离最小的点的距离  
 pq.pop();  
 if(d>dis[u]) continue;//u已经被更新过  
 if(mp[u].empty()) continue;  
 for(auto it:mp[u])  
 {  
 int v=it.first;  
 int w=it.second;  
 if(dis[v]>dis[u]+w)//更新s->v的最短距离(min(s->v,s->u->v))  
 {  
 dis[v]=dis[u]+w;  
 pq.push({dis[v],v});  
 }  
 }  
 }  
 //正确性证明：  
 //假设目前更新s->t(=3),假设存在s->u->t(=2),则s->u<s->t,而s->u一定在之前被更新过，所以s->u->t一定在之前被更新过，与假设矛盾。  
 //单源最短路(正边权)  
 //时间复杂度O(ElogV),E为边数,V为点数(二叉堆)  
 //使用斐波那契堆的 Dijkstra 算法的时间复杂度为 O(E+VlogV)。  
 //不用堆优化：O(v^2+E)  
 //当E<<v^2时，使用堆优化  
 //当E~v^2时，不用堆优化  
 return dis;  
}  
int main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 int n=read(),m=read(),s=read();  
 vector<vector<pair<int,int>>> mp(n+1);  
 while(m--)  
 {  
 int u=read(),v=read(),w=read();  
 mp[u].push\_back({v,w});  
 }  
 vector<int> dis=dijkstra(n,mp,s);  
 for(int i=1;i<=n;i++)  
 write(dis[i]),putchar(' ');  
 int T\_end=clock();  
 return 0;  
}

dfs&bfs

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
using namespace std;  
vector <int> edge[100000+5];  
queue <int> q;int vis[100000+5]={0},sum=0;  
int ans[100000+5];  
int read()  
{  
 int s=0,f=1;  
 char ch=getchar();  
 while(ch<'0'||ch>'9')  
 {  
 if(ch=='-') f=-1;  
 ch=getchar();  
 }  
 while(ch>='0'&&ch<='9')  
 {  
 s=(s<<3)+(s<<1)+ch-'0';  
 ch=getchar();  
 }  
 return s\*f;  
}  
inline void write(int x)   
{  
 static int sta[35];   
 int top=0;  
 do{  
 sta[top++]=x%10, x/=10;  
 }while(x);  
 while(top) putchar(sta[--top]+48);  
}  
void dfs(int x)  
{  
 write(x);putchar(' ');vis[x]=1;  
 for(int i=0;i<edge[x].size();i++)  
 {  
 if(!vis[edge[x][i]]) dfs(edge[x][i]);  
 }  
 return ;  
}  
int bfs(int x)  
{  
 q.push(x);  
 while(!q.empty())  
 {  
 int temp=q.front(); q.pop();  
 if(vis[temp]) continue;  
 else{  
 vis[temp]=1;sum++;  
 }  
 for(int i=0;i<edge[temp].size();i++)  
 {  
 q.push(edge[temp][i]);  
 if(!vis[edge[temp][i]]) ans[edge[temp][i]]=ans[temp];  
 }  
 // cout<<q.size()<<endl;  
 // for(int i=0;i<=q.size();i++)  
 // {  
 // cout<<q.front()<<" ";  
 // q.pop();  
 // }  
 }  
 return sum;  
}  
int main()  
{  
 int T\_start=clock();  
 int n=read(),m=read();  
 for(int i=0;i<m;i++)  
 {  
 int u=read(),v=read();  
 edge[v].push\_back(u);  
 }  
 for(int i=1;i<=n;i++)  
 {  
 ans[i]=i;  
 }  
 for(int i=n;i>=1;i--)  
 {  
 bfs(i);  
 if(sum==n) break;  
 }  
 for(int i=1;i<=n;i++)  
 {  
 write(ans[i]);putchar(' ');  
 }  
 putchar('\n');  
 return 0;  
}  
//preview:2024.12.29 23:01

johnson最短路

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
using namespace std;  
#define int long long  
int has\_neg=0;  
vector<int> SPFA(vector<vector<pair<int,int>>>& mp,int s,int n)  
{  
 vector<int> dis(n+1,1e9);  
 vector<int> vis(n+1,0);  
 vector<int> cnt(n+1,0);  
 dis[s]=0;queue<int> q;  
 q.push(s);vis[s]=1;cnt[s]=0;  
 while(!q.empty())  
 {  
 int u=q.front();  
 q.pop();vis[u]=0;  
 for(auto [v,w]:mp[u])  
 {  
 if(dis[v]>dis[u]+w)//松弛  
 {  
 dis[v]=dis[u]+w;  
 cnt[v]=cnt[u]+1;  
 if(cnt[v]>n-1)//存在负权环  
 {  
 //1-n的节点，最短路最多经过n-1条边，如果经过n条边，说明存在负权环  
 has\_neg=1;  
 return dis;  
 }  
 if(!vis[v])  
 {  
 q.push(v);  
 vis[v]=1;  
 }  
 }  
 }  
 }  
 return dis;  
 //SPFA: 形式上Bellman\_Ford是一棵树，很显然，只有上一次被松弛的节点u，才有可能对v进行松弛，所以可以采用SPFA  
 //为啥只有上一次被松弛的节点u，才有可能对v进行松弛？  
 //手玩一下就好了（悲  
 //考虑简单图，他可以是个递推的过程  
}  
vector<int> dijkstra(int n,vector<vector<pair<int,int>>>& mp,int s)  
{  
 vector<int> dis(n+1,1e9);//初始化距离为无穷大  
 dis[s]=0;//起点到起点的距离为0  
 priority\_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>> pq;  
 pq.push({0,s});//将起点加入优先队列  
 while(!pq.empty())  
 {  
 int u=pq.top().second;//取出当前距离最小的点  
 int d=pq.top().first;//取出当前距离最小的点的距离  
 pq.pop();  
 if(d>dis[u]) continue;//u已经被更新过  
 if(mp[u].empty()) continue;  
 for(auto it:mp[u])  
 {  
 int v=it.first;  
 int w=it.second;  
 if(dis[v]>dis[u]+w)//更新s->v的最短距离(min(s->v,s->u->v))  
 {  
 dis[v]=dis[u]+w;  
 pq.push({dis[v],v});  
 }  
 }  
 }  
 //正确性证明：  
 //假设目前更新s->t(=3),假设存在s->u->t(=2),则s->u<s->t,而s->u一定在之前被更新过，所以s->u->t一定在之前被更新过，与假设矛盾。  
 //单源最短路(正边权)  
 //时间复杂度O(ElogV),E为边数,V为点数(二叉堆)  
 //使用斐波那契堆的 Dijkstra 算法的时间复杂度为 O(E+VlogV)。  
 //不用堆优化：O(v^2+E)  
 //当E<<v^2时，使用堆优化  
 //当E~v^2时，不用堆优化  
 return dis;  
}  
vector<vector<int>> johnson(vector<vector<pair<int,int>>>& mp,int n)  
{  
 //1.添加一个虚拟节点0，连接到所有节点，边权为0  
 for(int i=1;i<=n;i++)  
 {  
 mp[0].push\_back({i,0});  
 }  
 //2.使用Bellman-Ford算法计算从虚拟节点0到所有节点的最短路径  
 vector<int> h=SPFA(mp,0,n+1);  
 if(has\_neg==1)   
 {  
 cout<<-1<<endl;  
 exit(0);  
 }  
 //3.删除虚拟节点0，并更新所有边的权重  
 for(int i=1;i<=n;i++)  
 {  
 for(auto& it:mp[i])  
 {  
 it.second+=h[i]-h[it.first];  
 }  
 }  
 //4.对每个节点i，使用Dijkstra算法计算从i到所有节点的最短路径  
 vector<vector<int>> dis(n+1,vector<int>(n+1,1e9));  
 for(int i=1;i<=n;i++)  
 {  
 dis[i]=dijkstra(n,mp,i);  
 for(int j=1;j<=n;j++)  
 {  
 dis[i][j]-=h[i]-h[j];  
 if(dis[i][j]>=1e8) dis[i][j]=1e9;  
 }  
 }  
 //5.更新所有边的权重  
 for(int i=1;i<=n;i++)  
 {  
 for(auto& it:mp[i])  
 {  
 it.second+=h[it.first]-h[i];  
 }  
 }  
 return dis;  
}  
signed main()  
{  
 int T\_start=clock();  
 int n,m;  
 cin>>n>>m;  
 vector<vector<pair<int,int>>> mp(n+1);  
 for(int i=0;i<m;i++)  
 {  
 int u,v,w;  
 cin>>u>>v>>w;  
 mp[u].push\_back({v,w});  
 }  
 vector<vector<int>> dis=johnson(mp,n);  
 for(int i=1;i<=n;i++)  
 {  
 int ans=0;  
 for(int j=1;j<=n;j++)  
 {  
 //cout<<dis[i][j]<<" ";  
 ans+=j\*dis[i][j];  
 }  
 //cout<<endl;  
 cout<<ans<<endl;  
 }  
 return 0;  
}  
//johnson全源最短路算法：O(nmlogm)  
//重新标记边权后,u-v两点的任意路径一定有hu-hv项，最短路不变  
//由于三角形不等式，所以重新标记边权，边权一定非负  
//所以重新标记边权后，可以使用Dijkstra算法求最短路

二分图最大匹配

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
//#define int long long //赫赫 要不要龙龙呢  
using namespace std;  
class maxflow{  
 public:  
 struct node  
 {  
 int to,cap,id;  
 };  
 vector<vector<node>> mp;  
 vector<int> dep,cur;//dep:层次图，cur:当前弧优化  
 int n,m,s,t;  
 maxflow(int n,int m,int s,int t,vector<array<int,3>>& eds):  
 mp(n+1),n(n),m(m),s(s),t(t),dep(n+1),cur(n+1){  
 //u->v capacity  
 for(auto [u,v,cap]:eds){  
 int uid=mp[u].size();  
 int vid=mp[v].size();  
 mp[u].push\_back({v,cap,vid});  
 mp[v].push\_back({u,0,uid});  
 //建反边  
 }  
 }  
 bool bfs(){  
 fill(dep.begin(),dep.end(),-1);  
 fill(cur.begin(),cur.end(),0);  
 queue<int> q;  
 q.push(s);  
 dep[s]=0;  
 while(!q.empty()){  
 int u=q.front();  
 q.pop();  
 for(auto [v,cap,id]:mp[u]){  
 if(cap>0&&dep[v]==-1){  
 dep[v]=dep[u]+1;  
 q.push(v);  
 }  
 }  
 }  
 return dep[t]!=-1;  
 }  
 int dfs(int u,int lim)//到u点的最大流量lim  
 {  
 if(u==t) return lim;  
 int sum=0;//u点流出的流量  
 for(int &i=cur[u];i<mp[u].size();i++){  
 //当前弧优化,考虑u->v有重边,那么这个优化会使  
 //被榨干过的v的出边不再被访问  
 auto [v,cap,id]=mp[u][i];  
 if(cap>0&&dep[v]==dep[u]+1){  
 int f=dfs(v,min(lim,cap));  
 mp[u][i].cap-=f;  
 mp[v][id].cap+=f;  
 sum+=f;  
 lim-=f;  
 if(lim==0) break;  
 }  
 }  
 if(sum==0) dep[u]=-1;//无增广路  
 return sum;  
 }  
 int dinic(){  
 int res=0;  
 while(bfs()){  
 res+=dfs(s,INT\_MAX);  
 }  
 return res;  
 }  
};  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
 int n,m,e;cin>>n>>m>>e;  
 vector<array<int,3>> eds;  
 int s=n+m+1,t=n+m+2;  
 //s->left cap 1  
 for(int i=1;i<=n;i++){  
 eds.push\_back({s,i,1});  
 }  
 //right->t cap 1  
 for(int i=1;i<=m;i++){  
 eds.push\_back({i+n,t,1});  
 }  
 //u->v cap 1  
 for(int i=1;i<=e;i++){  
 int u,v;cin>>u>>v;  
 eds.push\_back({u,v+n,1});  
 }  
 maxflow mf(n+m+2,e,s,t,eds);  
 cout<<mf.dinic()<<endl;  
 return 0;  
}  
//二分图最大匹配 最大流O(n^1/2\*m)  
//二分图的划分可以用二分图染色进行  
//二分图最大匹配：设有若干男生,若干女生,若干配对关系,求最大匹配,即求出最多的配对关系  
  
//二分图最小点覆盖：在一张无向图中选择最少的顶点，满足每条边至少有一个端点被选  
//->二分图中，最小点覆盖中的顶点数量等于最大匹配中的边数量。  
//从网络流的角度看，最小点覆盖问题就是最小割问题：选择左部点，相当于切割它与源点的连边；选择右部点，相当于切割它与汇点的连边。  
//why?因为一条边被割掉，意味着原二分图上这个点的配对点无法跑一条流->最小割  
//[引] 最小割：把图分为s集和t集 s->t的边为割边，割边的最小权值和为最小割  
  
//最大独立集问题：在一张无向图中选择最多的顶点，满足两两之间互不相邻。  
//->二分图中，最大独立集中的顶点数量等于n-最小点覆盖中的顶点数量  
//[引理1] 图G(V,E)中v的子集s为点覆盖<=>v/s为独立集  
//证明：s为点覆盖=>v/s为独立集 假设v/s不是独立集，则存在v1,v2∈v/s，v1,v2相邻，但是v1-v2这条边的两个端点都不在s中 ->矛盾  
//v/s为独立集=>s为点覆盖 要证 对所有u-v∈E，至少有一个端点在s中  
//假设u-v都不在s中，则u,v都在v/s中 但此时v/s不是独立集 ->矛盾  
//[推论1] 图G(V,E)中v的子集s为最大点覆盖<=>v/s为最小独立集  
  
//有向无环图最小路径覆盖：在一张有向图中，选择最少数量的简单路径，使得所有顶点都恰好出现在一条路径中。  
//->有向无环图的最小路径覆盖数等于顶点数减去最大匹配数  
//通过dag构造的二分图如下：  
//将每个顶点拆成两个顶点，v\_in v\_out  
//对于原图中的每条有向边u->v，在二分图中连边v\_in-u\_out  
//此证明为构造性的：二分图的每个匹配对应这dag中不交的各个路径  
//考虑最极端的平凡图 n个点 路径是n条 增加一个匹配->路径数减少1  
//Q.E.D

二分图染色

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
using namespace std;  
struct node{  
 int v;  
 int w;  
};  
vector<node> mp[20005];  
bool vis[20005]={false};  
int dyed[20005]={0};  
int Data[100005];  
int read()  
{  
 int s=0,f=1;  
 char ch=getchar();  
 while(ch<'0'||ch>'9')  
 {  
 if(ch=='-') f=-1;  
 ch=getchar();  
 }  
 while(ch>='0'&&ch<='9')  
 {  
 s=(s<<3)+(s<<1)+ch-'0';  
 ch=getchar();  
 }  
 return s\*f;  
}  
inline void write(int x)   
{  
 static int sta[35];   
 int top=0;  
 if(x<0&&x!=-2147483648) {putchar('-');x=-x;}  
 if(x==-2147483648) {printf("-2147483648");return;}  
 do{  
 sta[top++]=x%10, x/=10;  
 }while(x);  
 while(top) putchar(sta[--top]+48);  
}  
bool dye(int start,int mid)  
{  
 queue<int> q;  
 q.push(start);  
 vis[start]=1;dyed[start]=1;  
 while(!q.empty())  
 {  
 int temp=q.front();  
 q.pop();  
 for(auto i:mp[temp])  
 {  
 if(i.w>=mid)  
 {  
 if(!vis[i.v])  
 {  
 q.push(i.v);  
 vis[i.v]=true;  
 dyed[i.v]=3-dyed[temp];  
 }  
 else if(dyed[i.v]==dyed[temp]) return false;  
 }  
 }  
 }  
 return true;   
}  
bool isBinGraph(int n,int mid)  
{  
 memset(vis,0,sizeof(vis));  
 memset(dyed,0,sizeof(dyed));  
 for(int i=1;i<=n;i++)  
 {  
 if(!vis[i])  
 {  
 if(!dye(i,mid)) return false;  
 }  
 }  
 return true;  
}  
int main()  
{  
 int T\_start=clock();  
 freopen("in.txt","r",stdin);  
 // freopen("out.txt","w",stdout);  
 int n=read(),m=read();  
 for(int i=0;i<m;i++)  
 {  
 int u=read(),v=read(),w=read();  
 mp[u].push\_back({v,w});  
 mp[v].push\_back({u,w});  
 Data[i]=w;  
 }  
 sort(Data,Data+m);  
 // for(int i=0;i<m;i++)  
 // {  
 // cout<<Data[i]<<endl;  
 // }  
 if(isBinGraph(n,0))  
 {  
 cout<<"0"<<endl;  
 }  
 else  
 {  
 int l=0,r=m;  
 while(l<=r)  
 {  
 int mid=(l+r)>>1;  
 if(!isBinGraph(n,Data[mid])) l=mid+1;  
 else r=mid-1;  
 }  
 cout<<Data[r]<<endl;  
 }   
 return 0;  
}

分层图

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
int read()  
{  
 int s=0,f=1;  
 char ch=getchar();  
 while(ch<'0'||ch>'9')  
 {  
 if(ch=='-') f=-1;  
 ch=getchar();  
 }  
 while(ch>='0'&&ch<='9')  
 {  
 s=(s<<3)+(s<<1)+ch-'0';  
 ch=getchar();  
 }  
 return s\*f;  
}  
inline void write(int x)   
{  
 static int sta[35];   
 int top=0;  
 if(x<0&&x!=-2147483648) {putchar('-');x=-x;}  
 if(x==-2147483648) {printf("-2147483648");return;}  
 do{  
 sta[top++]=x%10, x/=10;  
 }while(x);  
 while(top) putchar(sta[--top]+48);  
}  
int dij(vector<vector<pair<int,int>>>& mp,int s,int n,int t,int kk)  
{  
 vector<int> vis((kk+1)\*n+1,0x7fffffff);  
 vis[s]=0;  
 priority\_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>> pq;  
 pq.push({0,s});  
 while(!pq.empty())  
 {  
 auto [val,k]=pq.top();  
 pq.pop();  
 if(val>vis[k]) continue;  
 for(auto i:mp[k])  
 {  
 auto [v,w]=i;  
 if(vis[v]>vis[k]+w)  
 {  
 vis[v]=vis[k]+w;  
 pq.push({vis[v],v});  
 }  
 }  
 }  
 int ans=0x7fffffff;  
 for(int i=0;i<=kk;i++)  
 {  
 //i表示免费次数  
 ans=min(ans,vis[i\*n+t]);  
 }  
 return ans;  
}  
int main()  
{  
 //分层图：解决k次免费（有代价）最短路问题  
 int T\_start=clock();  
 int n=read(),m=read(),k=read();  
 int s=read(),t=read();  
 vector<vector<pair<int,int>>> mp((k+1)\*n+1);  
 while(m--)  
 {  
 int u,v,w;  
 u=read(),v=read(),w=read();  
 for(int i=0;i<=k;i++)  
 {  
 mp[i\*n+u].push\_back({i\*n+v,w});  
 mp[i\*n+v].push\_back({i\*n+u,w});  
 if(i!=k)   
 {  
 mp[i\*n+u].push\_back({(i+1)\*n+v,0});  
 mp[i\*n+v].push\_back({(i+1)\*n+u,0});//分层图连边  
 }  
 }  
 }  
 cout<<dij(mp,s,n,t,k)<<endl;  
 return 0;  
}

差分约束

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
int flag=0;  
int read()  
{  
 int s=0,f=1;  
 char ch=getchar();  
 while(ch<'0'||ch>'9')  
 {  
 if(ch=='-') f=-1;  
 ch=getchar();  
 }  
 while(ch>='0'&&ch<='9')  
 {  
 s=(s<<3)+(s<<1)+ch-'0';  
 ch=getchar();  
 }  
 return s\*f;  
}  
inline void write(int x)   
{  
 static int sta[35];   
 int top=0;  
 if(x<0&&x!=-2147483648) {putchar('-');x=-x;}  
 if(x==-2147483648) {printf("-2147483648");return;}  
 do{  
 sta[top++]=x%10, x/=10;  
 }while(x);  
 while(top) putchar(sta[--top]+48);  
}  
vector<int> SPFA(vector<vector<pair<int,int>>>& mp,int s,int n)  
{  
 vector<int> dis(n+1,0x7fffffff);  
 vector<int> vis(n+1,0);  
 vector<int> cnt(n+1,0);  
 dis[s]=0;queue<int> q;  
 q.push(s);vis[s]=1;cnt[s]=1;  
 while(!q.empty())  
 {  
 int u=q.front();  
 q.pop();vis[u]=0;  
 for(auto [v,w]:mp[u])  
 {  
 if(dis[v]>dis[u]+w)//松弛  
 {  
 dis[v]=dis[u]+w;  
 cnt[v]=cnt[u]+1;  
 if(cnt[v]>=n+1)  
 {  
 flag=1;  
 return vector<int>(n+1,-1);  
 }  
 if(!vis[v])  
 {  
 q.push(v);  
 vis[v]=1;  
 }  
 }  
 }  
 }  
 return dis;  
}  
int main()  
{  
 int T\_start=clock();  
 int n=read(),m=read();  
 vector<vector<pair<int,int>>> mp(n+1);  
 for(int i=1;i<=m;i++)  
 {  
 int v=read(),u=read(),w=read();  
 mp[u].push\_back(make\_pair(v,w));  
 }  
 for(int i=1;i<=n;i++)  
 {  
 mp[0].push\_back(make\_pair(i,0));  
 }  
 vector<int>ans=SPFA(mp,0,n);  
 if(flag==1)  
 {  
 printf("NO\n");  
 }  
 else  
 {  
 //printf("YES\n");  
 for(int i=1;i<=n;i++)  
 {  
 printf("%d ",ans[i]);  
 }  
 printf("\n");  
 }  
 return 0;  
}  
//对一个差分约束系统，判断是否存在一组解，使得所有约束条件都成立。  
//ex. x1-x2<=3  
// x2-x3<=-2  
// x1-x3<=1  
//将xn看作超级源点（到所有点的权值为w=0）到n的最短路  
//那么第一个式子的意义就是x1<=x2+3,0到1的最短路<=3+0到2的最短路  
//在图上的意义就是建2->1的边权为3的边，0->1,0->2的边权为0的边  
//0->1,0->2的边权为0的边也是添加了以下条件  
//x1-x0<=0  
//x2-x0<=0  
//x0=0  
//那么整个系统就转化为了一张图  
//求xn即求0到n的最短路，如果存在负环，则无解，否则有解  
//负环还原的形式为  
//x1-x2<=-1...1  
//x2-x3<=-4...2  
//x3-x1<=-5...3  
//1+2+3->0<=-10,不成立  
//还有结论，设定w即求x1,x2..xn<=w的最大解  
//如果差分约束系统换换不等号，求最长路，spfa改一下即可  
  
//结论形式证明  
//假设X0是定死的；X1到Xn在满足所有约束的情况下可以取到的最大值分别为M1、M2、……、Mn（当然我们不知道它们的值是多少）；解出的源点到每个点的最短路径长度为D1、D2、……、Dn。  
//基本的Bellman-Ford算法是一开始初始化D1到Dn都是无穷大。然后检查所有的边对应的三角形不等式，一但发现有不满足三角形不等式的情况，则更新对应的D值。最后求出来的D1到Dn就是源点到每个点的最短路径长度。  
//如果我们一开始初始化D1、D2、……、Dn的值分别为M1、M2、……、Mn，则由于它们全都满足三角形不等式（我们刚才已经假设M1到Mn是一组合法的解），则Bellman-Ford算法不会再更新任合D值，则最后得出的解就是M1、M2、……、Mn。  
//好了，现在知道了，初始值无穷大时，算出来的是D1、D2、……、Dn；初始值比较小的时候算出来的则是M1、M2、……、Mn。大家用的是同样的算法，同样的计算过程，总不可能初始值大的算出来的结果反而小吧。所以D1、D2、……、Dn就是M1、M2、……、Mn。

拓扑排序

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
using namespace std;  
#define MOD 80112002  
vector<int> edge[5005];  
int \_to[5005]={0},\_in[5005]={0};  
long long ans[5005]={0};queue<int> q;  
int read()  
{  
 int s=0,f=1;  
 char ch=getchar();  
 while(ch<'0'||ch>'9')  
 {  
 if(ch=='-') f=-1;  
 ch=getchar();  
 }  
 while(ch>='0'&&ch<='9')  
 {  
 s=(s<<3)+(s<<1)+ch-'0';  
 ch=getchar();  
 }  
 return s\*f;  
}  
inline void write(int x)   
{  
 static int sta[35];   
 int top=0;  
 do{  
 sta[top++]=x%10, x/=10;  
 }while(x);  
 while(top) putchar(sta[--top]+48);  
}  
int main()  
{  
 int T\_start=clock();  
 int n=read(),m=read();  
 for(int i=0;i<m;i++)  
 {  
 int u=read(),v=read();  
 edge[v].push\_back(u);  
 \_to[u]++;\_in[v]++;  
 }  
 for(int i=1;i<=n;i++)  
 {  
 if(!\_to[i])   
 {  
 q.push(i);  
 ans[i]=1;  
 }  
 }  
 while(!q.empty())  
 {  
 int temp=q.front();  
 //cout<<temp<<endl;  
 q.pop();  
 for(int i=0;i<edge[temp].size();i++)  
 {  
 //cout<<temp<<' '<<edge[temp][i]<<' '<<ans[temp]<<' '<<ans[edge[temp][i]]<<endl;  
 ans[edge[temp][i]]=(ans[edge[temp][i]]+ans[temp])%MOD;  
 \_to[edge[temp][i]]--;  
 if(!\_to[edge[temp][i]]) q.push(edge[temp][i]);  
 }  
 }  
 long long res=0;  
 for(int i=1;i<=n;i++)  
 {  
 if(!\_in[i])   
 {  
 //cout<<i<<' '<<ans[i]<<endl;  
 res=(res+ans[i])%MOD;  
 }  
 }  
 write(res),putchar('\n');  
 return 0;  
}

最大流(dinic)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
using namespace std;  
#define int long long  
class maxflow{  
 public:  
 struct node  
 {  
 int to,cap,id;  
 };  
 vector<vector<node>> mp;  
 vector<int> dep,cur;//dep:层次图，cur:当前弧优化  
 int n,m,s,t;  
 maxflow(int n,int m,int s,int t,vector<array<int,3>>& eds):  
 mp(n+1),n(n),m(m),s(s),t(t),dep(n+1),cur(n+1){  
 //u->v capacity  
 for(auto [u,v,cap]:eds){  
 int uid=mp[u].size();  
 int vid=mp[v].size();  
 mp[u].push\_back({v,cap,vid});  
 mp[v].push\_back({u,0,uid});  
 //建反边  
 }  
 }  
 bool bfs(){  
 fill(dep.begin(),dep.end(),-1);  
 fill(cur.begin(),cur.end(),0);  
 queue<int> q;  
 q.push(s);  
 dep[s]=0;  
 while(!q.empty()){  
 int u=q.front();  
 q.pop();  
 for(auto [v,cap,id]:mp[u]){  
 if(cap>0&&dep[v]==-1){  
 dep[v]=dep[u]+1;  
 q.push(v);  
 }  
 }  
 }  
 return dep[t]!=-1;  
 }  
 int dfs(int u,int lim)//到u点的最大流量lim  
 {  
 if(u==t) return lim;  
 int sum=0;//u点流出的流量  
 for(int &i=cur[u];i<mp[u].size();i++){  
 //当前弧优化,考虑u->v有重边,那么这个优化会使  
 //被榨干过的v的出边不再被访问  
 auto [v,cap,id]=mp[u][i];  
 if(cap>0&&dep[v]==dep[u]+1){  
 int f=dfs(v,min(lim,cap));  
 mp[u][i].cap-=f;  
 mp[v][id].cap+=f;  
 sum+=f;  
 lim-=f;  
 if(lim==0) break;  
 }  
 }  
 if(sum==0) dep[u]=-1;//无增广路  
 return sum;  
 }  
 int dinic(){  
 int res=0;  
 while(bfs()){  
 res+=dfs(s,INT\_MAX);  
 }  
 return res;  
 }  
};  
signed main()  
{  
 int T\_start=clock();  
 int n,m,s,t;  
 cin>>n>>m>>s>>t;  
 vector<array<int,3>> eds(m);  
 for(auto &[u,v,cap]:eds){  
 cin>>u>>v>>cap;  
 }  
 maxflow mf(n,m,s,t,eds);  
 cout<<mf.dinic()<<endl;  
 return 0;  
}  
//最大流，解决从有向图源点到汇点的最大流量问题(假定源点流量无限)  
//dinic算法，时间复杂度O(n^2\*m)  
//增广路：是从源点到汇点的路径，其上所有边的残余容量均大于0  
//初级思路：贪心选择所有增广路，然后更新边权，引入反向边进行反悔贪心  
//基本思路：每次bfs把图变成一个带层数的DAG(限制dfs深度)  
//然后找到极大增广流量，更新图，重复上述过程

最小斯坦纳树

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
#define int long long //赫赫 要不要龙龙呢  
const int INF=1e18;  
using namespace std;  
  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
 int n,m,k;cin>>n>>m>>k;  
 vector<vector<array<int,2>>> mp(n+1);  
 for(int i=1;i<=m;i++)  
 {  
 int u,v,w;cin>>u>>v>>w;  
 mp[u].push\_back({v,w});  
 mp[v].push\_back({u,w});  
 }  
 vector<vector<int>> dp((1<<k),vector<int>(n+1,INF));  
 //dp[i][j]表示以i为集合，j为根的最小贡献  
 vector<int> sp(k+1);  
 for(int i=1;i<=k;i++)  
 {  
 cin>>sp[i];  
 dp[1<<(i-1)][sp[i]]=0;  
 }  
 for(int st=1;st<(1<<k);st++){  
 for(int t=st;t;t=(t-1)&st){  
 for(int i=1;i<=n;i++)  
 {  
 dp[st][i]=min(dp[st][i],dp[t][i]+dp[st-t][i]);  
 }  
 }//枚举子集，合并 只保证了v节点的状态是最优的  
 priority\_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>> q;  
 vector<int> vis(n+1,0);  
 for(int i=1;i<=n;i++){  
 if(dp[st][i]!=INF) q.push({dp[st][i],i});   
 }  
 while(!q.empty()){  
 int u=q.top().second;q.pop();  
 if(vis[u]) continue;  
 vis[u]=1;  
 for(auto [v,w]:mp[u]){  
 if(dp[st][v]>dp[st][u]+w){  
 dp[st][v]=dp[st][u]+w;  
 q.push({dp[st][v],v});  
 }  
 }  
 }//状态传播  
 }  
 cout<<dp[(1<<k)-1][sp[1]]<<endl;  
 //状态传播完了,由于树的性质,所以sp[1]一定是根节点  
 return 0;  
}  
//最小斯坦纳树 给定一个图 和k个关键点 求一个包含所有关键点的最小生成树(可以用其他点)  
//时间复杂度(n\*3^k+2^k\*mlogm)

最小生成树（kruskal）

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
class BSU{  
 public:  
 int n;vector<int> fa;  
 BSU(int n):n(n)  
 {  
 fa.resize(n+1);  
 for(int i=1;i<=n;i++)  
 {  
 fa[i]=i;  
 }  
 }  
 int find(int u){  
 return fa[u]==u?u:fa[u]=find(fa[u]);  
 }  
 void merge(int a,int b)  
 {  
 int op=rand()%2;  
 if(op==0) fa[find(a)]=find(b);  
 else fa[find(b)]=find(a);  
 }  
};  
int read()  
{  
 int s=0,f=1;  
 char ch=getchar();  
 while(ch<'0'||ch>'9')  
 {  
 if(ch=='-') f=-1;  
 ch=getchar();  
 }  
 while(ch>='0'&&ch<='9')  
 {  
 s=(s<<3)+(s<<1)+ch-'0';  
 ch=getchar();  
 }  
 return s\*f;  
}  
inline void write(int x)   
{  
 static int sta[35];   
 int top=0;  
 if(x<0&&x!=-2147483648) {putchar('-');x=-x;}  
 if(x==-2147483648) {printf("-2147483648");return;}  
 do{  
 sta[top++]=x%10, x/=10;  
 }while(x);  
 while(top) putchar(sta[--top]+48);  
}  
int kruskal(vector<array<int,3>> &edge,int m,int n)  
{  
 sort(edge.begin(),edge.end(),[](auto a,auto b)->bool{  
 return a[2]<b[2];  
 });  
 BSU bsu(n);int cnt=0;int ans=0;  
 for(auto [u,v,w]:edge)  
 {  
 if(bsu.find(u)!=bsu.find(v))  
 {  
 bsu.merge(u,v);  
 cnt++;ans+=w;  
 //cout<<u<<" "<<v<<endl;  
 }  
 if(cnt==n-1) break;  
 }  
 return cnt==n-1?ans:-1;  
}  
//时间复杂度O(mlogm)，证明同prim  
int main()  
{  
 int T\_start=clock();  
 srand(time(NULL));  
 //freopen("in.txt","r",stdin);  
 int n=read(),m=read();  
 vector<array<int,3>> edge(m);  
 for(int i=0;i<m;i++)  
 {  
 int u=read(),v=read(),w=read();  
 edge[i]={u,v,w};  
 }  
 int ans=kruskal(edge,m,n);  
 if(ans==-1) puts("orz");  
 else cout<<ans<<endl;  
 return 0;  
}

最小生成树（prim）

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
int read()  
{  
 int s=0,f=1;  
 char ch=getchar();  
 while(ch<'0'||ch>'9')  
 {  
 if(ch=='-') f=-1;  
 ch=getchar();  
 }  
 while(ch>='0'&&ch<='9')  
 {  
 s=(s<<3)+(s<<1)+ch-'0';  
 ch=getchar();  
 }  
 return s\*f;  
}  
inline void write(int x)   
{  
 static int sta[35];   
 int top=0;  
 if(x<0&&x!=-2147483648) {putchar('-');x=-x;}  
 if(x==-2147483648) {printf("-2147483648");return;}  
 do{  
 sta[top++]=x%10, x/=10;  
 }while(x);  
 while(top) putchar(sta[--top]+48);  
}  
vector<int> prim(vector<vector<pair<int,int>>> &mp,int s,int n)  
{  
 vector<int> dis(n+1,0x7fffffff);//点离当前生成树的距离  
 vector<int> in(n+1,0);//点是否在生成树中  
 priority\_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>> pq;  
 dis[s]=0;pq.push({0,s});  
 while(!pq.empty())  
 {  
 auto [\_,u]=pq.top();//找到最小生成树连的边中未加入生成树的边权最小的边  
 pq.pop();  
 if(in[u]) continue;  
 in[u]=true;//进入最小生成树  
 for(auto [v,w]:mp[u])  
 {  
 if(dis[v]>w&&!in[v])//更新不在当前生成树中的点离生成树的距离  
 {  
 dis[v]=w;  
 pq.push({dis[v],v});  
 }  
 }  
 }  
 return dis;  
}  
//和dij一样，时间复杂度O((n+m)logn),暴力prim时间复杂度O(n^2),看看稀疏图和稠密图哪个更快  
//正确性证明：反证法：假设prim生成的不是最小生成树  
// 1).设prim生成的树为G0  
// 2).假设存在Gmin使得cost(Gmin)<cost(G0) 则在Gmin中存在<u,v>不属于G0  
// 3).将<u,v>加入G0中可得一个环，且<u,v>不是该环的最长边(这是因为<u,v>∈Gmin)  
// 4).这与prim每次生成最短边矛盾  
// 5).故假设不成立，命题得证.  
int main()  
{  
 int T\_start=clock();  
 int n=read(),m=read();  
 vector<vector<pair<int,int>>> mp(n+1);  
 for(int i=1;i<=m;i++)  
 {  
 int u=read(),v=read(),w=read();  
 mp[u].push\_back({v,w});  
 mp[v].push\_back({u,w});  
 }  
 vector<int> ans=prim(mp,1,n);  
 int sum=0;  
 for(int i=1;i<=n;i++)  
 {  
 if(ans[i]==0x7fffffff)  
 {  
 puts("不连通！");  
 return 0;  
 }  
 sum+=ans[i];  
 }  
 cout<<sum<<endl;  
 return 0;  
}

最小费用最大流(dinic)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
using namespace std;  
#define int long long  
class Mcmf{  
public:  
 struct node{  
 int to;  
 int cap;  
 int cost;  
 int rev;  
 };  
 int n,s,t;  
 int maxf=0,minc=0;  
 const int INF=1e18;  
 vector<vector<node>> mp;  
 vector<int> dis,cur,inq,vis;  
 Mcmf(int n,int s,int t,vector<array<int,4>>& eds):  
 n(n),s(s),t(t),mp(n+1),dis(n+1),  
 cur(n+1),inq(n+1,0),vis(n+1,0){  
 for(auto [u,v,cap,w]:eds){  
 int uid=mp[u].size();  
 int vid=mp[v].size();  
 mp[u].push\_back({v,cap,w,vid});  
 mp[v].push\_back({u,0,-w,uid});  
 //反边的费用是负的  
 }  
 }  
  
 bool spfa(){  
 fill(dis.begin(),dis.end(),INF);  
 fill(inq.begin(),inq.end(),0);  
 deque<int> q;dis[s]=0,inq[s]=1;  
 q.push\_back(s);  
 while(!q.empty()){  
 int u=q.front();q.pop\_front();  
 inq[u]=0;  
 for(auto [v,cap,w,rev]:mp[u]){  
 if(cap>0&&dis[u]+w<dis[v]){  
 dis[v]=dis[u]+w;  
 if(!inq[v]){  
 if(!q.empty()&&dis[v]<dis[q.front()]){  
 q.push\_front(v);  
 }else{  
 q.push\_back(v);  
 }  
 inq[v]=1;  
 }  
 }  
 }  
 }  
 return dis[t]!=INF;  
 }  
  
 int dfs(int u,int f){  
 if(u==t)return f;  
 vis[u]=1;  
 int res=0;  
 for(int &i=cur[u];i<mp[u].size();i++){  
 auto [v,cap,w,rev]=mp[u][i];  
 if(!vis[v]&&cap>0&&dis[u]+w==dis[v]){  
 int tmp=dfs(v,min(f,cap));  
 f-=tmp;  
 res+=tmp;  
 mp[u][i].cap-=tmp;  
 mp[v][rev].cap+=tmp;  
 minc+=tmp\*w;  
 if(!f)break;  
 }  
 }  
 vis[u]=0;  
 return res;  
 }  
  
 void dinic(){  
 while(spfa()){  
 fill(vis.begin(),vis.end(),0);  
 fill(cur.begin(),cur.end(),0);  
 maxf+=dfs(s,INF);  
 }  
 }  
  
};  
signed main()  
{  
 int T\_start=clock();  
 int n,m,s,t;  
 cin>>n>>m>>s>>t;  
 vector<array<int,4>> eds;  
 for(int i=0;i<m;i++){  
 int u,v,cap,w;  
 cin>>u>>v>>cap>>w;  
 eds.push\_back({u,v,cap,w});  
 }  
 Mcmf mcmf(n,s,t,eds);  
 mcmf.dinic();  
 cout<<mcmf.maxf<<" "<<mcmf.minc<<endl;  
 return 0;  
}  
//最小费用最大流，O(nmf)  
//基本思路：找到最短增广路，然后增广，直到找不到为止  
//最短增广路：spfa(slf优化)，每次找到最短路径，然后更新，直到找不到为止  
//增广：用dinic思路在最短路上多路增广

最小费用最大流(dinic,浮点)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <iomanip>  
using namespace std;  
#define int long long  
struct ta{  
 int u,v;  
 int cap;  
 double w;  
};  
class Mcmf{  
public:  
 struct node{  
 int to;  
 int cap;  
 double cost;  
 int rev;  
 };  
 int n,s,t;  
 int maxf=0;double minc=0;  
 const int INF=1e18;  
 vector<vector<node>> mp;  
 vector<int> cur,inq,vis;  
 vector<double> dis;  
 Mcmf(int n,int s,int t,vector<ta>& eds):  
 n(n),s(s),t(t),mp(n+1),dis(n+1),  
 cur(n+1),inq(n+1,0),vis(n+1,0){  
 for(auto [u,v,cap,w]:eds){  
 int uid=mp[u].size();  
 int vid=mp[v].size();  
 mp[u].push\_back({v,cap,w,vid});  
 mp[v].push\_back({u,0,-w,uid});  
 //反边的费用是负的  
 }  
 }  
  
 bool spfa(){  
 fill(dis.begin(),dis.end(),INF);  
 fill(inq.begin(),inq.end(),0);  
 deque<int> q;dis[s]=0,inq[s]=1;  
 q.push\_back(s);  
 while(!q.empty()){  
 int u=q.front();q.pop\_front();  
 inq[u]=0;  
 for(auto [v,cap,w,rev]:mp[u]){  
 if(cap>0&&dis[u]+w+1e-10<dis[v]){  
 dis[v]=dis[u]+w;  
 if(!inq[v]){  
 if(!q.empty()&&dis[v]+1e-10<dis[q.front()]){  
 q.push\_front(v);  
 }else{  
 q.push\_back(v);  
 }  
 inq[v]=1;  
 }  
 }  
 }  
 }  
 return dis[t]!=INF;  
 }  
  
 int dfs(int u,int f){  
 if(u==t)return f;  
 vis[u]=1;  
 int res=0;  
 for(int &i=cur[u];i<mp[u].size();i++){  
 auto [v,cap,w,rev]=mp[u][i];  
 if(!vis[v]&&cap>0&&abs(dis[u]+w-dis[v])<1e-10){  
 int tmp=dfs(v,min(f,cap));  
 f-=tmp;  
 res+=tmp;  
 mp[u][i].cap-=tmp;  
 mp[v][rev].cap+=tmp;  
 minc+=1.0\*tmp\*w;  
 if(!f)break;  
 }  
 }  
 vis[u]=0;  
 return res;  
 }  
  
 void dinic(){  
 while(spfa()){  
 fill(vis.begin(),vis.end(),0);  
 fill(cur.begin(),cur.end(),0);  
 maxf+=dfs(s,INF);  
 }  
 }  
  
};  
signed main()  
{  
 int T\_start=clock();  
 int n;cin>>n;  
 vector<array<int,2>> xy(n+1);  
 for(int i=1;i<=n;i++){  
 cin>>xy[i][0]>>xy[i][1];  
 }  
 int s=0,t=2\*n+1;  
 vector<ta> eds;  
 for(int i=1;i<=n;i++)  
 {  
 eds.push\_back({s,i,2,0});  
 }  
 for(int i=1;i<=n;i++)  
 {  
 eds.push\_back({i+n,t,1,0});  
 }  
 auto dis=[&](int u,int v)->double{  
 return sqrt(1.0\*(xy[u][0]-xy[v][0])\*(xy[u][0]-xy[v][0])+1.0\*(xy[u][1]-xy[v][1])\*(xy[u][1]-xy[v][1]));  
 };  
 for(int u=1;u<=n;u++)  
 {  
 for(int v=1;v<=n;v++)  
 {  
 if(xy[u][1]>xy[v][1])  
 {  
 eds.push\_back({u,v+n,1,dis(u,v)});  
 }  
 }  
 }  
 Mcmf mc(2\*n+2,s,t,eds);  
 mc.dinic();  
 if(mc.maxf==n-1) cout<<fixed<<setprecision(10)<<mc.minc<<endl;  
 else cout<<-1<<endl;  
 return 0;  
}  
//最小费用最大流，O(nmf)  
//基本思路：找到最短增广路，然后增广，直到找不到为止  
//最短增广路：spfa(slf优化)，每次找到最短路径，然后更新，直到找不到为止  
//增广：用dinic思路在最短路上多路增广  
//浮点数比较  
//a==b->abs(a-b)<1e-10  
//a<b->a+eps<b  
//a>b->a>b+eps

最小费用最大流(原始对偶)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
using namespace std;  
#define int long long  
class Mcmf{  
public:  
 struct node{  
 int to;  
 int cap;  
 int cost;  
 int rev;  
 };  
 int n,s,t;  
 int maxf=0,minc=0;  
 const int INF=1e18;  
 vector<vector<node>> mp;  
 vector<int> dis,h,prev,previd;  
 Mcmf(int n,int s,int t,vector<array<int,4>>& eds):  
 n(n),s(s),t(t),mp(n+1),dis(n+1),  
 h(n+1,INF),prev(n+1),previd(n+1){  
 //only 1-based  
 for(auto [u,v,cap,w]:eds){  
 int uid=mp[u].size();  
 int vid=mp[v].size();  
 mp[u].push\_back({v,cap,w,vid});  
 mp[v].push\_back({u,0,-w,uid});  
 //反边的费用是负的  
 }  
 }  
 bool dijk(){  
 fill(dis.begin(),dis.end(),INF);  
 dis[s]=0;  
 priority\_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>> pq;  
 pq.push({0,s});  
 while(!pq.empty()){  
 auto [d,u]=pq.top();  
 pq.pop();  
 if(dis[u]<d) continue;  
 for(int i=0;i<mp[u].size();i++){  
 auto [v,cap,w,rev]=mp[u][i];  
 int cost=w+h[u]-h[v];  
 if(cap>0&&dis[u]+cost<dis[v]){  
 dis[v]=dis[u]+cost;  
 prev[v]=u;//记录前驱  
 previd[v]=i;//记录当前弧  
 pq.push({dis[v],v});  
 }  
 }  
 }  
 return dis[t]<INF;  
 }  
  
 void SPFA(){  
 h[s]=0;  
 queue<int> q;  
 vector<bool> inq(n+1,0);  
 q.push(s),inq[s]=1;  
 while(!q.empty()){  
 int u=q.front();  
 q.pop();  
 inq[u]=0;  
 for(auto [v,cap,w,rev]:mp[u]){  
 if(cap>0&&h[u]+w<h[v]){  
 h[v]=h[u]+w;  
 if(!inq[v]){  
 q.push(v);  
 inq[v]=1;  
 }  
 }  
 }  
 }  
 }  
  
 void PD(){  
 SPFA();  
 while(dijk())  
 {  
 //now dis(u-v)=dis(u,v)(real)+h[u]-h[v]  
 //when u=0, dis(u,v)=dis(u,v)(real)-h[v]  
 int d=INF;  
 for(int i=t;i!=s;i=prev[i])  
 {  
 d=min(d,mp[prev[i]][previd[i]].cap);  
 }  
 //计算增广路径上的最小流量  
 maxf+=d;  
 minc+=d\*(dis[t]+h[t]);  
 for(int i=t;i!=s;i=prev[i])  
 {  
 mp[prev[i]][previd[i]].cap-=d;  
 mp[i][mp[prev[i]][previd[i]].rev].cap+=d;  
 }  
 //更新残余网络  
 for(int i=1;i<=n;i++)  
 {  
 if(dis[i]<INF)  
 {  
 h[i]+=dis[i];  
 }  
 }  
 }  
 }  
};  
signed main()  
{  
 int T\_start=clock();  
 int n,m,s,t;  
 cin>>n>>m>>s>>t;  
 vector<array<int,4>> eds;  
 for(int i=0;i<m;i++){  
 int u,v,cap,w;  
 cin>>u>>v>>cap>>w;  
 eds.push\_back({u,v,cap,w});  
 }  
 Mcmf mcmf(n,s,t,eds);  
 mcmf.PD();  
 cout<<mcmf.maxf<<" "<<mcmf.minc<<endl;  
 return 0;  
}  
//Primal-Dual 原始对偶算法,O(F\*E\*logE)  
//利用johnson最短路，将每条边的权值加上一个常数，使得每条边的权值非负，从而可以使用dijkstra算法

最近公共祖先（LCA）(targan,静态)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <unordered\_map>  
using namespace std;  
const int MaxN=5e5+5;  
vector<int> tree[MaxN];  
vector<pair<int,int>> q[MaxN];  
int vis[MaxN],ans[MaxN];  
int fa[MaxN];  
void prepare\_tree(int n)  
{  
 for(register int i=1;i<=n;i++)  
 {  
 fa[i]=i;  
 }  
}  
int find(int G)  
{  
 if(G==fa[G]) return G;  
 else  
 {  
 fa[G]=find(fa[G]);  
 return fa[G];  
 }  
 //return G==fa[G]? G:(fa[G]=find(fa[G]));  
}  
void merge(int a,int b)//合并  
{  
 fa[find(a)]=find(b);//有时路径压缩可能破坏rank'(rank->树深)  
 /\*register int x=find(a),y=find(b);  
 Rank[x]<=Rank[y]?fa[x]=y:fa[y]=x;  
 if(Rank[x]==Rank[y]&&x!=y) Rank[y]++;\*/  
  
}  
int read()  
{  
 int s=0,f=1;  
 char ch=getchar();  
 while(ch<'0'||ch>'9')  
 {  
 if(ch=='-') f=-1;  
 ch=getchar();  
 }  
 while(ch>='0'&&ch<='9')  
 {  
 s=(s<<3)+(s<<1)+ch-'0';  
 ch=getchar();  
 }  
 return s\*f;  
}  
inline void write(int x)   
{  
 static int sta[35];   
 int top=0;  
 if(x<0&&x!=-2147483648) {putchar('-');x=-x;}  
 if(x==-2147483648) {printf("-2147483648");return;}  
 do{  
 sta[top++]=x%10, x/=10;  
 }while(x);  
 while(top) putchar(sta[--top]+48);  
}  
void dfs(int node)  
{  
 vis[node]=1;  
 for(auto child:tree[node])  
 {  
 if(!vis[child])  
 {  
 dfs(child);  
 fa[child]=node;//调换顺序会使路径压缩到child的父节点，此时子树还没遍历完  
 }  
 }  
 for(auto i:q[node])  
 {  
 if(vis[i.first])//node及其子树已经dfs完了,如果此时i已经搜到，显然，根据dfs原则，find(i)是lca(i,node)  
 {  
 ans[i.second]=find(i.first);  
 }  
 }  
}  
int main()  
{  
 int T\_start=clock();  
 int n=read(),m=read(),s=read();  
 for(int i=0;i<n-1;i++)  
 {  
 int u=read(),v=read();  
 tree[v].push\_back(u);  
 tree[u].push\_back(v);  
 }  
 for(int i=0;i<m;i++)  
 {  
 int u=read(),v=read();  
 q[v].push\_back(make\_pair(u,i));  
 q[u].push\_back(make\_pair(v,i));  
 }  
 prepare\_tree(n);  
 for(int i=1;i<=n;i++)  
 {  
 vis[i]=0;  
 }  
 dfs(s);  
 for(int i=0;i<m;i++)  
 {  
 write(ans[i]);  
 putchar('\n');  
 }  
 return 0;  
}

最近公共祖先（LCA）(倍增)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
using namespace std;  
const int MAXN=5e5+5;  
const int LOG=25;//MAXN<=2^LOG  
vector<int> tree[MAXN];  
int dep[MAXN],st[MAXN][LOG];//节点深度，st表，st[i][j]=i的2^j级祖先  
int read()  
{  
 int s=0,f=1;  
 char ch=getchar();  
 while(ch<'0'||ch>'9')  
 {  
 if(ch=='-') f=-1;  
 ch=getchar();  
 }  
 while(ch>='0'&&ch<='9')  
 {  
 s=(s<<3)+(s<<1)+ch-'0';  
 ch=getchar();  
 }  
 return s\*f;  
}  
inline void write(int x)   
{  
 static int sta[35];   
 int top=0;  
 if(x<0&&x!=-2147483648) {putchar('-');x=-x;}  
 if(x==-2147483648) {printf("-2147483648");return;}  
 do{  
 sta[top++]=x%10, x/=10;  
 }while(x);  
 while(top) putchar(sta[--top]+48);  
}  
void init(int node,int parent)//用dfs预处理dep和st  
{  
 dep[node]=(parent==-1)?0:dep[parent]+1;  
 st[node][0]=parent;//一级祖先为自身  
 for(int i=1;i<LOG;i++)//更新node的祖先表  
 {  
 if(st[node][i-1]!=-1)  
 {  
 st[node][i]=st[st[node][i-1]][i-1];  
 //node的2^j级祖先为node的2^j-1祖先的2^j-1祖先  
 }  
 else st[node][i]=-1;//你的码的码没了，你还有码？（可删吗？）  
 }  
 for(auto child:tree[node])  
 {  
 if(child!=parent)  
 {  
 init(child,node);//从父节点向下dfs  
 }  
 }  
}  
int lca(int u,int v)  
{  
 if(dep[u]<dep[v]) swap(u,v);//确保u比v深  
 int diff=dep[u]-dep[v];  
 for(int i=0;i<LOG;i++)  
 {  
 if((diff>>i)&1)  
 {  
 u=st[u][i];//u向上跳转2^i,其中i为diff的二进制表示中第i位为一  
 }  
 }  
 if(u==v) return u;//深度相等，可能找到  
 //不相等，假设他们与lca(u,v)的距离为diff  
 //注意到5=4+1，5-4=1  
 //7=4+2+1,7-4-2=1  
 //6=4+2,6-4-1=1  
 //12=8+4,12-8-2-1  
 //做以下操作总能使diff=1  
 // for(int i=LOG-1;i>=0;i--)  
 // {  
 // if(st[u][i]!=st[v][i])  
 // {  
 // u=st[u][i];  
 // v=st[v][i];  
 // }  
 // }  
 // return st[u][0];  
 //优化版  
 for(int i=LOG-1;i>=0;i--)  
 {  
 if(st[u][i]!=st[v][i])  
 {  
 u=st[u][i];  
 v=st[v][i];  
 }  
 }  
 return st[u][0];  
}  
int main()  
{  
 int T\_start=clock();  
 int n=read(),m=read(),s=read();//n个点，n-1条边,m个询问，s为根  
 for(int i=0;i<n-1;i++)  
 {  
 int u=read(),v=read();  
 tree[u].push\_back(v);  
 tree[v].push\_back(u);//存树  
 }  
 for(int i=1;i<=n;i++)  
 {  
 dep[i]=-1;  
 for(int j=0;j<LOG;j++)  
 {  
 st[i][j]=-1;  
 }  
 }  
 init(s,-1);  
 while(m--)  
 {  
 write(lca(read(),read()));  
 putchar('\n');  
 }  
 return 0;  
}

树上基本处理

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
#define int long long //赫赫 要不要龙龙呢  
using namespace std;  
class tree{  
 public:  
 vector<vector<int>> tr;  
 vector<vector<int>> fa;  
 vector<int> dfn,dep,siz;  
 int n,k,tot;  
 tree(int n,vector<vector<int>>& tr):  
 tr(tr),n(n),k(\_\_lg(n)+1),dfn(n+1,0),dep(n+1,0),tot(0),siz(n+1,0),  
 fa(n+1,vector<int>(\_\_lg(n)+2,0)){  
 dfs(1,0);  
 }  
  
 int dfs(int u,int f){  
 dfn[u]=++tot;  
 dep[u]=dep[f]+1;  
 fa[u][0]=f;siz[u]=1;  
 for(int i=1;i<=k;i++){   
 fa[u][i]=fa[fa[u][i-1]][i-1];   
 }  
 for(auto v:tr[u]){  
 if(v==f) continue;  
 siz[u]+=dfs(v,u);  
 }  
 return siz[u];  
 }  
  
 int lca(int u,int v){  
 if(dep[u]<dep[v]) swap(u,v);  
 for(int i=k;i>=0;i--){  
 if(dep[fa[u][i]]>=dep[v]) u=fa[u][i];  
 }  
 if(u==v) return u;  
 for(int i=k;i>=0;i--){  
 if(fa[u][i]!=fa[v][i]){  
 u=fa[u][i];  
 v=fa[v][i];  
 }  
 }  
 return fa[u][0];  
 }  
 //求u,v的最近公共祖先  
 int dis(int u,int v){  
 return dep[u]+dep[v]-2\*dep[lca(u,v)];  
 }  
 //求u,v距离  
 bool con(int s,int t,int x){  
 return dis(s,x)+dis(t,x)==dis(s,t);  
 }  
 //判断x是否在s,t的路径上  
};  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
  
 return 0;  
}

树链剖分（线段树ver.）

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
#define int long long  
const int N=1e5+5;  
int dep[N],fa[N],hson[N],top[N],siz[N],dfn[N],rk[N],val[N];  
int mod;  
class SegTree{  
 public:  
 struct Node  
 {  
 int sum;  
 int s,e;  
 int lazy=0;  
 Node\* lt;  
 Node\* rt;  
 Node(int sum,int s,int e):s(s),e(e),sum(sum),lt(nullptr),rt(nullptr){}  
 };  
 Node\* root;  
 Node\* buildtree(vector<int> &nums,int l,int r)  
 {  
 if(l>r) return nullptr;  
 if(l==r) return new Node(nums[l],l,l);  
 int mid=(l+r)>>1;  
 Node\* root=new Node(0,l,r);  
 Node\* lc=buildtree(nums,l,mid);  
 Node\* rc=buildtree(nums,mid+1,r);  
 if(lc) root->lt=lc,root->sum=(root->sum+lc->sum)%mod;  
 if(rc) root->rt=rc,root->sum=(root->sum+rc->sum)%mod;  
 return root;  
 }  
 void init(vector<int>nums)  
 {  
 root=buildtree(nums,0,nums.size()-1);  
 return;  
 }  
 void taglazy(Node\* root,int val)  
 {  
 if(root==nullptr) return;  
 val%=mod;  
 root->lazy=(root->lazy+val)%mod;  
 root->sum=(root->sum+(root->e-root->s+1)%mod\*val)%mod;  
 }  
 void pushdown(Node\* root)  
 {  
 if(!root) return ;  
 if(root->lazy)  
 {  
 taglazy(root->lt,root->lazy);  
 taglazy(root->rt,root->lazy);  
 root->lazy=0;  
 }  
 }  
 void update(Node\* root,int l,int r,int val)  
 {  
 if(!root) return ;  
 if(root->s>r||root->e<l) return ;  
 if(root->s>=l&&root->e<=r)  
 {  
 taglazy(root,val);  
 return;  
 }  
 pushdown(root);  
 update(root->lt,l,r,val);  
 update(root->rt,l,r,val);  
 root->sum=((root->lt?root->lt->sum:0)+(root->rt?root->rt->sum:0))%mod;  
 return ;  
 }  
 void update(int l,int r,int val)  
 {  
 update(root,l,r,val);  
 return ;  
 }  
 int query(Node\* root,int l,int r)  
 {  
 pushdown(root);  
 if(!root) return 0;  
 if(root->s>r||root->e<l) return 0;  
 if(root->s>=l&&root->e<=r) return root->sum;  
 return query(root->lt,l,r)+query(root->rt,l,r);  
 }  
 int query(int l,int r)  
 {  
 return query(root,l,r);  
 }  
};  
class cutTree  
{  
 //树链剖分，把树剖分成若干条链，每条链上维护一个线段树  
 //可以支持链上修改和查询，也可以支持树上修改和查询  
 //还可以求lca  
 //重链剖分有一个重要的性质：对于节点数为n的树，从任意节点向上走到根节点，经过的轻边数量不超过logn。这是因为，如果一个节点连向父节点的边是轻边，  
 //就必然存在子树不小于它的兄弟节点，那么父节点对应子树的大小一定超过该节点的两倍(由dfs1可得)。每经过一条轻边，子树大小就翻倍，所以最多只能经过logn条。  
 public:  
 int n,tot,s;  
 //s:根节点  
 vector<vector<int>> tree;  
 //dep:树深,fa:父节点,hson:i的重儿子,top:重链顶端,siz:子树大小,dfn:dfs序,rk:dfs序对应的节点  
 SegTree seg;  
 void dfs1(int u,int f)  
 {  
 //cntt++;cout<<cntt<<endl;  
 dep[u]=dep[f]+1;//更新树深  
 fa[u]=f;siz[u]=1;  
 for(auto v:tree[u])  
 {  
 if(v==f)continue;  
 dfs1(v,u);  
 siz[u]+=siz[v];  
 if(hson[u]==-1||siz[v]>siz[hson[u]]) hson[u]=v;  
 //u的重儿子是所有子树大小最大的儿子  
 }  
 }  
 void dfs2(int u)  
 {  
 dfn[u]=++tot;rk[tot]=u;  
 //优先访问重儿子,保证重链顶端的dfn最小  
 if(hson[u]!=-1)  
 {  
 top[hson[u]]=top[u];  
 //重儿子的top是它所在重链的顶端  
 dfs2(hson[u]);  
 }  
 for(auto v:tree[u])  
 {  
 if(v==fa[u]||v==hson[u])//跳过父节点和重儿子  
 continue;  
 top[v]=v;//轻儿子的top是自己  
 dfs2(v);  
 }  
 }  
 void init()  
 {  
 tot=0;  
 dfs1(s,0);  
 dfs2(s);  
 }  
 int lca(int u,int v)  
 {  
 while(top[u]!=top[v])//不在同一条重链上  
 {  
 if(dep[top[u]]<dep[top[v]])swap(u,v);  
 u=fa[top[u]];  
 //链头深度大的往上跳  
 }  
 return dep[u]<dep[v]?u:v;  
 }  
 int queryPath(int u,int v)  
 {  
 int ans=0;  
 while(top[u]!=top[v])//遍历所有的边  
 {  
 if(dep[top[u]]<dep[top[v]])swap(u,v);  
 ans=(ans+seg.query(dfn[top[u]],dfn[u]))%mod;  
 u=fa[top[u]];  
 }  
 if(dep[u]>dep[v])swap(u,v);  
 ans=(ans+seg.query(dfn[u],dfn[v]))%mod;  
 return ans;  
 }  
 void updatePath(int u,int v,int val)  
 {  
 while(top[u]!=top[v])  
 {  
 if(dep[top[u]]<dep[top[v]])swap(u,v);  
 seg.update(dfn[top[u]],dfn[u],val);  
 u=fa[top[u]];  
 }  
 if(dep[u]>dep[v])swap(u,v);  
 seg.update(dfn[u],dfn[v],val);  
 }  
 void updateSub(int u,int val)  
 {  
 //子树的dfn一定是连续的  
 seg.update(dfn[u],dfn[u]+siz[u]-1,val);  
 }  
 int querySub(int u)  
 {  
 return seg.query(dfn[u],dfn[u]+siz[u]-1);  
 }  
 cutTree(int n,vector<vector<int>> tree,int s):n(n),tree(tree),s(s)  
 {  
 for(int i=0;i<=n;i++)  
 {  
 dep[i]=0;fa[i]=-1;hson[i]=-1;top[i]=-1;  
 siz[i]=0;dfn[i]=-1;rk[i]=-1;  
 }  
 top[s]=s;init(); vector<int> inf(n+1,0);  
 for(int i=1;i<=n;i++)inf[dfn[i]]=val[i]%mod;  
 seg.init(inf);  
 }  
};  
signed main()  
{  
 int T\_start=clock();  
 ios::sync\_with\_stdio(false);  
 cin.tie(0);  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 int n,m,s;  
 cin>>n>>m>>s>>mod;  
 vector<vector<int>> tree(n+1);  
 for(int i=1;i<=n;i++)  
 {  
 cin>>val[i];  
 }  
 for(int i=1;i<n;i++)  
 {  
 int u,v;  
 cin>>u>>v;  
 tree[u].push\_back(v);  
 tree[v].push\_back(u);  
 }  
 cutTree ct(n,tree,s);  
 while(m--)  
 {  
 int op,x,y,z;  
 cin>>op;  
 if(op==1)  
 {  
 cin>>x>>y>>z;  
 ct.updatePath(x,y,z);  
 }  
 else if(op==2)  
 {  
 cin>>x>>y;  
 cout<<ct.queryPath(x,y)%mod<<endl;  
 }  
 else if(op==3)  
 {  
 cin>>x>>y;  
 ct.updateSub(x,y);  
 }  
 else if(op==4)  
 {  
 cin>>x;  
 cout<<ct.querySub(x)%mod<<endl;  
 }  
 }  
 int T\_end=clock();  
 //cout<<"time: "<<(double)(T\_end-T\_start)/CLOCKS\_PER\_SEC<<"s"<<endl;  
 return 0;  
}

点分治

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <unordered\_set>  
#include <numeric>  
using namespace std;  
  
int main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 int n,m;cin>>n>>m;  
 vector<vector<pair<int,int>>> tr(n+1);  
 for(int i=1;i<n;i++)  
 {  
 int u,v,w;  
 cin>>u>>v>>w;  
 tr[u].push\_back({v,w});  
 tr[v].push\_back({u,w});  
 }  
 vector<int> siz(n+1,0),q(m+1),vis(n+1,0),ans(m+1,0);  
 for(int i=1;i<=m;i++) cin>>q[i];  
 auto getsz=[&](auto getsz,int u,int p=-1)->int  
 {  
 siz[u]=1;  
 for(auto [v,w]:tr[u])  
 {  
 if(v==p||vis[v])continue;  
 siz[u]+=getsz(getsz,v,u);  
 }  
 return siz[u];  
 };//统计以u为根的子树大小  
 auto getrt=[&](auto getrt,int u,int p=-1,int sizrt)->int  
 {  
 for(auto [v,w]:tr[u])  
 {  
 if(v==p||vis[v])continue;  
 if(siz[v]>sizrt/2)return getrt(getrt,v,u,sizrt);  
 }  
 return u;  
 };//寻找重心  
 //重心：对于一棵树，如果存在一个顶点，其子树中最大的子树大小不超过整棵树大小的一半，则称该顶点为这棵树的重心。  
 auto clac=[&](auto clac,int uu,int dis,int p=-1,vector<int>& tpd)->void  
 {  
 tpd.push\_back(dis);  
 for(auto [vv,ww]:tr[uu])  
 {  
 if(vv==p||vis[vv])continue;  
 clac(clac,vv,dis+ww,uu,tpd);  
 }  
 };  
 auto div=[&](auto div,int u)->void{  
 vis[u]=1;  
 unordered\_set<int> s{0};  
 for(auto [v,w]:tr[u])  
 {  
 if(vis[v])continue;  
 vector<int> tpd;  
 clac(clac,v,w,u,tpd);  
 for(auto d:tpd)  
 {  
 for(int i=1;i<=m;i++)  
 {  
 if(!ans[i]&&d<=q[i]&&s.find(q[i]-d)!=s.end())  
 {  
 ans[i]=1;  
 }  
 }  
 }  
 for(auto d:tpd)s.insert(d);  
 }  
 for(auto [v,w]:tr[u])  
 {  
 //用重心划分u的子树  
 if(vis[v])continue;  
 getsz(getsz,v);  
 int subrt=getrt(getrt,v,-1,siz[v]);  
 div(div,subrt);  
 }  
 };//处理以u为根的子树  
 getsz(getsz,1);  
 int rt=getrt(getrt,1,-1,siz[1]);  
 div(div,rt);  
 for(int i=1;i<=m;i++)   
 {  
 if(ans[i])cout<<"AYE\n";  
 else cout<<"NAY\n";  
 }  
 return 0;  
}  
//淀粉质：把树上路径问题转化为子树分治问题  
//把树按重心划分，那么树高（或树的大小）不超过n/2，递归深度不超过logn(最坏：退化为链），于是可以暴力处理子树  
//根据实现方式的不同，时间复杂度可以做到O(nlogn)或O(nlog^2n)

虚树(可拓展版)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
#define int long long //赫赫 要不要龙龙呢  
using namespace std;  
class vtree{  
 public:  
 vector<vector<int>> tr,vtr;  
 vector<vector<int>> fa;  
 vector<int> dfn,dep;  
 int n,k,tot;  
 vtree(int n,vector<vector<int>>& tr):  
 tr(tr),vtr(n+1),n(n),k(\_\_lg(n)+1),dfn(n+1,0),dep(n+1,0),tot(0),  
 fa(n+1,vector<int>(\_\_lg(n)+2,0)){  
 dfs(1,0);  
 }  
  
 void dfs(int u,int f){  
 dfn[u]=++tot;  
 dep[u]=dep[f]+1;  
 fa[u][0]=f;  
 for(int i=1;i<=k;i++){   
 fa[u][i]=fa[fa[u][i-1]][i-1];   
 }  
 for(auto v:tr[u]){  
 if(v==f) continue;  
 dfs(v,u);  
 }  
 }  
  
 int lca(int u,int v){  
 if(dep[u]<dep[v]) swap(u,v);  
 for(int i=k;i>=0;i--){  
 if(dep[fa[u][i]]>=dep[v]) u=fa[u][i];  
 }  
 if(u==v) return u;  
 for(int i=k;i>=0;i--){  
 if(fa[u][i]!=fa[v][i]){  
 u=fa[u][i];  
 v=fa[v][i];  
 }  
 }  
 return fa[u][0];  
 }  
  
 void getvTree(vector<int>& o){  
 sort(o.begin(),o.end(),[&](int a,int b){return dfn[a]<dfn[b];});  
 int p=o.size();  
 for(int i=1;i<p;i++){  
 o.push\_back(lca(o[i-1],o[i]));  
 }  
 sort(o.begin(),o.end(),[&](int a,int b){return dfn[a]<dfn[b];});  
 o.erase(unique(o.begin(),o.end()),o.end());  
 for(int i=1;i<o.size();i++){  
 int tp=lca(o[i-1],o[i]);  
 vtr[tp].push\_back({o[i]});  
 //vtr[o[i]].push\_back({tp});  
 }  
 }  
 void clear(vector<int>& o){  
 for(auto x:o) vtr[x].clear();  
 }  
};  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
 int t;cin>>t;  
 while(t--){  
 int n,k;cin>>n>>k;  
 vector<vector<int>> tr(n+1),id(k+1);  
 vector<int> c(n+1),w(n+1),cnt(n+1,0);  
 //c[0]=1;  
 for(int i=1;i<=n;i++) cin>>w[i];  
 for(int i=1;i<=n;i++) cin>>c[i],id[c[i]].push\_back(i);  
 for(int i=1;i<n;i++){  
 int u,v;cin>>u>>v;  
 tr[u].push\_back(v);  
 tr[v].push\_back(u);  
 }  
 vtree vt(n,tr);  
 for(int i=1;i<=k;i++){  
 vt.getvTree(id[i]);  
 for(auto x:id[i]){  
 cnt[x]+=1;  
 if(c[x]==0)   
 {  
 c[x]=i;  
 }  
 }  
 vt.clear(id[i]);  
 }  
 int ans=0;  
 for(int i=1;i<=n;i++){  
 if(cnt[i]>=2) ans+=w[i];  
 }  
 cout<<ans<<endl;  
 auto dfs=[&](this auto& dfs,int u,int f)->void{  
 for(auto v:tr[u]){  
 if(v==f) continue;  
 if(c[v]==0&&c[u]!=0) c[v]=c[u];  
 dfs(v,u);  
 if(c[u]==0&&c[v]!=0) c[u]=c[v];  
 }  
 if(c[u]==0) c[u]=1;  
 };  
 dfs(1,0);  
 for(int i=1;i<=n;i++){  
 cout<<c[i]<<" ";  
 }  
 cout<<endl;  
 }  
 return 0;  
}  
//虚树，处理q次询问，每次询问给出k个关键点，答案只跟关键点有关的问题  
//构建虚树：  
//1.将关键点按dfn排序  
//2.相邻的关键点的lca加入虚树  
//why?虚树的定义是关键点的集合和其两两lca构成的树  
//考虑任意两个关键点，其lca一定把这两个关键点分成两个分支，  
//而根据dfn的连续性，一定有某个相邻的关键点的lca是这个lca  
//Q.E.D  
//3.按父子关系构建虚树

虚树(带边权)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
#define int long long //赫赫 要不要龙龙呢  
using namespace std;  
class vtree{  
 public:  
 vector<vector<array<int,2>>> tr,vtr;  
 vector<vector<int>> len,fa;  
 vector<int> dfn,dep;  
 int n,k,tot;  
 vtree(int n,vector<vector<array<int,2>>>& tr):  
 tr(tr),vtr(n+1),n(n),k(\_\_lg(n)+1),dfn(n+1,0),dep(n+1,0),tot(0),  
 fa(n+1,vector<int>(\_\_lg(n)+2,0)),len(n+1,vector<int>(\_\_lg(n)+2,1e18)){  
 dfs(1,0);  
 }  
  
 void dfs(int u,int f){  
 dfn[u]=++tot;  
 dep[u]=dep[f]+1;  
 fa[u][0]=f;  
 for(int i=1;i<=k;i++){   
 fa[u][i]=fa[fa[u][i-1]][i-1];   
 len[u][i]=min(len[u][i-1],len[fa[u][i-1]][i-1]);  
   
 }  
 for(auto [v,w]:tr[u]){  
 if(v==f) continue;  
 len[v][0]=w;  
 dfs(v,u);  
 }  
 }  
  
 int lca(int u,int v){  
 if(dep[u]<dep[v]) swap(u,v);  
 for(int i=k;i>=0;i--){  
 if(dep[fa[u][i]]>=dep[v]) u=fa[u][i];  
 }  
 if(u==v) return u;  
 for(int i=k;i>=0;i--){  
 if(fa[u][i]!=fa[v][i]){  
 u=fa[u][i];  
 v=fa[v][i];  
 }  
 }  
 return fa[u][0];  
 }  
  
 int w(int u,int v){  
 if(dep[u]<dep[v]) swap(u,v);  
 int ans=1e18;  
 for(int i=k;i>=0;i--){  
 if(dep[fa[u][i]]>=dep[v]){  
 ans=min(ans,len[u][i]);  
 u=fa[u][i];  
 }  
 }  
 return ans;  
 }  
 void getvTree(vector<int>& o){  
 sort(o.begin(),o.end(),[&](int a,int b){return dfn[a]<dfn[b];});  
 int p=o.size();  
 for(int i=1;i<p;i++){  
 o.push\_back(lca(o[i-1],o[i]));  
 }  
 sort(o.begin(),o.end(),[&](int a,int b){return dfn[a]<dfn[b];});  
 o.erase(unique(o.begin(),o.end()),o.end());  
 for(int i=1;i<o.size();i++){  
 int tp=lca(o[i-1],o[i]);  
 vtr[tp].push\_back({o[i],w(o[i],tp)});  
 //vtr[o[i]].push\_back({tp,w(o[i],tp)});  
 }  
 }  
 void clear(vector<int>& o){  
 for(auto x:o) vtr[x].clear();  
 }  
};  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
 int n;cin>>n;  
 vector<vector<array<int,2>>> tr(n+1);  
 for(int i=1;i<n;i++){  
 int u,v,w;cin>>u>>v>>w;  
 tr[u].push\_back({v,w});  
 tr[v].push\_back({u,w});  
 }  
 vtree vt(n,tr);  
 vector<bool> iskey(n+1,0);  
 int q;cin>>q;  
 while(q--){  
 vector<int> o,co;  
 int k;cin>>k;  
 for(int i=0;i<k;i++){  
 int x;cin>>x;  
 o.push\_back(x);  
 iskey[x]=1;  
 }  
 co=o;  
 vt.getvTree(o);  
 auto dp=[&](this auto& self,int u)->int{  
 if(iskey[u]){  
 return vt.len[u][0];  
 }  
 int ans=0;  
 for(auto [v,w]:vt.vtr[u]){  
 ans += min(self(v), w);  
 }  
 return ans;  
 };  
 int fin=dp(o[0]);  
 if(o[0]==1) cout<<fin<<endl;  
 else cout<<min(fin,vt.w(o[0],1))<<endl;  
 vt.clear(o);  
 for(auto x:co) iskey[x]=0;  
 }  
 return 0;  
}  
//虚树，处理q次询问，每次询问给出k个关键点，答案只跟关键点有关的问题  
//构建虚树：  
//1.将关键点按dfn排序  
//2.相邻的关键点的lca加入虚树  
//why?虚树的定义是关键点的集合和其两两lca构成的树  
//考虑任意两个关键点，其lca一定把这两个关键点分成两个分支，  
//而根据dfn的连续性，一定有某个相邻的关键点的lca是这个lca  
//Q.E.D  
//3.按父子关系构建虚树  
//时间复杂度：O(nlogn)

# Other

离散化

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
using namespace std;  
unordered\_map<int,int> dis(vector<int> a)  
{  
 sort(a.begin(),a.end());  
 unordered\_map<int,int> mp;  
 for(int i=0;i<a.size();i++)  
 {  
 mp[a[i]]=i+1;  
 }  
 return mp;  
}  
int main()  
{  
 int T\_start=clock();  
   
 return 0;  
}

# hash

hash表

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
using namespace std;  
/\*hash mod number 53 97 193 389 769 1543 3079 6151 12289 24593 49157 98317 196613 393241 786433 1572869 3145739 6291469 12582917 25165843 50331653 100663319 402653189 805306457 1610612741 (1e9+7,1e9+9)\*/   
/\*prime such as 4k+3 (some)553963 553991 554003 554011 554051 554087 554123 554167 554171 554179 554207 554263 554299 554303 554347 554383 554419 554431 554447 554467 554503 554527 554531 554611 554627 554639 554663 554699 554707 554711 554731 554747 554759 554767 554779 554791 554803 554839 554843 554887 554891 554899 554923 554927 554951 554959 555043 555083 555091 555119 555143 555167 555251 555287 555307 555383 555391 555419 555439 555487 555491 555523 555671 555683 555691 555707 555739 555743 555767 555823 555827 555871 555931 555967 556007 556027 556043 556051 556067 556103 556123 556159 556211 556219 556243 556267 556271 556279 556327 556331 556343 556351 556399 556403 556459 556483 556487 556519 556559 556579 556583 556607 556627 556639 556651 556679 556687 556691 556723 556727 556763 556799 556811 556819 556823 556859 556867 556883 556891 556931 556939 556943 556967 556987 556999 557027 557059 557087 557159 557303 557339 557371 557423 557443 557483 557519 557551 557567 557591 557611 557639 557663 557671 557731 557743 557747 557759 557779 557803 557831 557863 557891 557899 557903 557927 557987 558007 558067 558083 558091 558139 558167 558179 558203 558223 558251 558287 558307 558319 558343 558427 558431 558479 558491 558499 558539 558563 558583 558587 558599 558611 558643 558683 558703 558731 558787 558791 558827 558863 558931 558947 558979 559051 559067 559099 559123 559183 559211 559219 559231 559243 559259 559319 559343 559367 559451 559459 559483 559511 559523 559547 559571 559583 559591 559631 559639 559667 559679 559687 559703 559739 559747 559799 559807 559831 559859 559883 559907 559939 559967 559991 560023 560039 560047 560083 560107 560123 560159 560171 560179 560191 560207 560227 560239 560243 560299 560311 560411 560447 560459 560471 560479 560491 560503 560531 560543 560551 560639 560683 560719 560767 560771 560783 560803 560827 560863 560887 560891 560939 561019 561047 561059 561079 561083 561091 561103 561191 561199 561251 561307 561343 561347 561359 561367 561419 561439 561551 561559 561599 561607 561667 561703 561767 561787 561839 561907 561923 561931 561943 561947 561983 562007 562019 562043 562091 562103 562147 562231 562259 562271 562283 562291 562307 562351 562399 562403 562427 562439 562459 562519 562579 562591 562607 562631 562651 562663 562691 562699 562703 562711 562739 562759 562763 562831 562871 562931 562943 562963 562967 562979 562987 563011 563039 563047 563051 563099 563119 563131 563183 563219 563263 563287 563327 563351 563359 563411 563419 563447 563467 563503 563543 563551 563587 563599 563623 563663 563723 563743 563747 563831 563851 563887 563947 563971 563987 563999 564059 564103 564127 564163 564191 564227 564251 564271 564299 564307 564323 564359 564367 564371 564391 564407 564419 564463 564467 564491 564523 564607 564643 564667 564671 564679 564703 564779 564827 564871 564899 564919 564923 564959 564979 564983 565039 565111 565127 565163 565171 565183 565207 565247 565259 565283 565303 565319 565343 565379 565387 565391 565427 565451 565463 565483 565507 565511 565519 565559 565567 565571 565583 565603 565651 565667 565723 565727 565771 565787 565867 565891 565907 565919 565979 566011 566023 566047 566107 566131 566179 566183 566227 566231 566311 566323 566347 566387 566431 566443 566539 566543 566551 566563 566567 566639 566659 566707 566719 566723 566759 566767 566791 566851 566879 566911 566939 566947 566963 566971 566987 566999 567011 567031 567059 567067 567107 567143 567179 567187 567263 567319 567323 567367 567383 567407 567439 567451 567467 567487 567499 567527 567607 567631 567659 567667 567719 567751 567767 567779 567811 567863 567871 567883 567899 567943 567947 567979 567991 568019 568027 568091 568151 568163 568171 568187 568207 568231 568279 568303 568363 568367 568387 568391 568439 568471 568523 568619 568627 568643 568679 568691 568699 568723 568751 568783 568787 568807 568823 568831 568891 568903 568907 568963 568979 568987 568991 568999 569003 569011 569047 569071 569083 569111 569159 569243 569251 569263 569267 569323 569419 569423 569431 569447 569479 569507 569579 569599 569603 569623 569659 569663 569671 569683 569711 569731 569747 569759 569771 569819 569831 569839 569843 569851 569887 569903 569927 569939 569983 570043 570047 570071 570079 570083 570091 570107 570131 570139 570191 570359 570379 570391 570403 570407 570419 570463 570467 570487 570491 570499 570511 570527 570539 570547 570587 570643 570659 570667 570671 570683 570719 570743 570827 570839 570851 570859 570887 570919 570959 570967 570991 571019 571031 571099 571111 571147 571163 571199 571211 571223 571231 571267 571279 571303 571331 571339 571399 571471 571531 571579 571583 571603 571679 571699 571751 571759 571783 571799 571811 571847 571867 571871 571903 571939 572023 572027 572051 572059 572063 572087 572107 572179 572183 572207 572239 572251 572303 572311 572323 572387 572399 572419 572423 572471 572479 572491 572519 572567 572587 572599 572639 572651 572659 572683 572687 572699 572707 572711 572791 572807 572827 572843 572867 572879 572903 572927 572939 572963 573007 573031 573047 573107 573119 573143 573163 573179 573247 573263 573299 573343 573371 573379 573383 573451 573479 573487 573511 573523 573527 573571 573647 573679 573691 573719 573739 573763 573787 573791 573847 573851 573863 573871 573883 573887 573899 573967 574003 574031 574051 574099 574127 574159 574163 574183 574219 574279 574283 574307 574363 574367 574423 574439 574507 574543 574547 574619 574627 574631 574643 574667 574687 574699 574703 574711 574723 574727 574799 574859 574907 574939 574963 574967 575027 575063 575087 575119 575123 575131 575203 575219 575231 575243 575251 575303 575359 575371 575431 575479 575503 575551 575579 575591 575611 575623 575647 575651 575699 575711 575723 575747 575791 575863 575867 575903 575923 575959 575963 575987 576019 576031 576119 576131 576151 576167 576179 576203 576211 576223 576227 576287 576299 576319 576379 576391 576427 576431 576439 576523 576539 576551 576647 576659 576671 576683 576703 576727 576731 576739 576743 576787 576791 576883 576899 576943 576967 577007 577043 577063 577067 577111 577123 577147 577151 577219 577259 577271 577279 577307 577327 577331 577351 577363 577387 577399 577427 577463 577471 577483 577523 577531 577547 577559 577627 577639 577667 577739 577751 577799 577807 577831 577867 577879 577919 577931 577939 577979 578047 578063 578131 578167 578183 578191 578203 578251 578267 578299 578311 578327 578363 578371 578399 578407 578419 578467 578483 578503 578563 578587 578603 578647 578659 578687 578719 578779 578803 578819 578827 578839 578843 578923 578959 578971 578999 579011 579023 579079 579083 579107 579119 579179 579199 579239 579251 579259 579263 579283 579287 579311 579331 579379 579407 579427 579451 579499 579503 579539 579563 579571 579583 579587 579611 579643 579707 579763 579779 579851 579883 579907 579947 579967 579983 580031 580079 580163 580183 580187 580219 580231 580259 580291 580303 580331 580339 580343 580379 580471 580487 580607 580627 580631 580639 580663 580687 580691 580711 580747 580759 580763 580787 580807 580843 580859 580871 580891 580919 580927 580939 581047 581071 581099 581143 581171 581183 581227 581239 581263 581303 581311 581323 581351 581407 581411 581443 581447 581459 581491 581527 581551 581599 581639 581663 581683 581687 581699 581731 581743 \*/  
const int MOD1=98317,MOD2=196613;  
vector <int> a[MOD1];  
vector <int> vis;  
bool \_find(int x)  
{  
 if(!a[(x%MOD1+MOD1)%MOD1].size()) return false;  
 for(int i=0;i<a[(x%MOD1+MOD1)%MOD1].size();i++)  
 {  
 if(a[(x%MOD1+MOD1)%MOD1][i]==x) return true;  
 }  
 return false;  
}  
void \_insert(int x)  
{  
 if(!\_find(x))   
 {  
 a[(x%MOD1+MOD1)%MOD1].push\_back(x);  
 vis.push\_back((x%MOD1+MOD1)%MOD1);  
 }  
 return ;  
}  
int read()  
{  
 int s=0,f=1;  
 char ch=getchar();  
 while(ch<'0'||ch>'9')  
 {  
 if(ch=='-') f=-1;  
 ch=getchar();  
 }  
 while(ch>='0'&&ch<='9')  
 {  
 s=(s<<3)+(s<<1)+ch-'0';  
 ch=getchar();  
 }  
 return s\*f;  
}  
inline void write(int x)   
{  
 static int sta[35];   
 int top=0;  
 if(x<0) {putchar('-');x=-x;}  
 do{  
 sta[top++]=x%10,x/=10;  
 }while(x);  
 while(top) putchar(sta[--top]+48);  
}  
int main()  
{  
 int T\_start=clock();  
 int t=read();  
 while(t--)  
 {  
 int n=read();  
 for(int i=0;i<n;i++)  
 {  
 int x=read();  
 if(!\_find(x))  
 {  
 \_insert(x);  
 write(x),putchar(' ');  
 }  
 }  
 putchar('\n');  
 for(auto i:vis)  
 {  
 a[i].clear();  
 }  
 vis.clear();  
 }  
   
   
 return 0;  
}

字符串双hash

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
#include <random>  
#define int long long //赫赫 要不要龙龙呢  
#define ull unsigned long long  
using namespace std;  
class SHash{  
 public:  
 const int m1=1e9+7,m2=1e9+9;  
 int b1,b2;  
 SHash(){  
 mt19937\_64 rand(time(0));  
 b1=rand()%(int)1e9+1e6,b2=rand()%(int)1e9+1e6;  
 }  
 ull get(string s){  
 int h1=0,h2=0;  
 for(auto c:s){  
 h1=(h1\*b1+c)%m1;  
 h2=(h2\*b2+c)%m2;  
 }  
 return ((ull)h1)<<32|(ull)(h2);  
 }  
};  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
 int t;cin>>t;  
 SHash hs;  
 while(t--)  
 {  
 int n;cin>>n;  
 vector<array<ull,2>> a(n+1);  
 for(int i=1;i<=n;i++)  
 {  
 string s1,s2;cin>>s1>>s2;  
 a[i]={hs.get(s1),hs.get(s2)};  
 }  
 vector<vector<int>> mp(n+1);  
 for(int i=1;i<=n;i++)  
 {  
 for(int j=1;j<=n;j++)  
 {  
 if(i==j) continue;  
 if(a[i][0]==a[j][0]||a[i][1]==a[j][1]) mp[i].push\_back(j);  
 }  
 }  
 int st=(1<<n);  
 vector<vector<int>> dp(st,vector<int>(n+1,0));  
 for(int i=1;i<=n;i++) dp[1<<(i-1)][i]=1;  
 for(int i=0;i<st;i++)  
 {  
 for(int u=1;u<=n;u++)  
 {  
 if(!dp[i][u]) continue;  
 for(int v:mp[u])  
 {  
 if((i|(1<<(v-1)))!=i)   
 dp[i|(1<<(v-1))][v]|=dp[i][u];  
 }  
 }  
 }  
 int ans=0;  
 for(int i=0;i<st;i++)  
 {  
 for(int j=1;j<=n;j++)  
 {  
 if(dp[i][j])  
 {  
 ans=max(ans,(int)\_\_builtin\_popcountll(i));  
 }  
 }  
 }  
 cout<<n-ans<<endl;  
 }  
 return 0;  
}

# string

AC自动机(dp版)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
//#define int long long //赫赫 要不要龙龙呢  
using namespace std;  
class AC{  
 public:  
 vector<vector<int>> ch;  
 int n,tot,pidx; //节点数>=模式串总长  
 vector<int> ans,ne,idx,deg,sidx,fin;  
 //idx:节点的新编号 sidx:原字符串对应的编号 fin:最终答案  
 AC(int n):n(n),ch(n+1,vector<int>(26,0)),  
 ans(n+1,0),ne(n+1,0),tot(0),pidx(0),  
 idx(n+1,0),deg(n+1,0),sidx(n+1,0),fin(n+1,0){}  
 void insert(string s,int i){  
 int p=0;  
 for(auto c:s){  
 if(!ch[p][c-'a']) ch[p][c-'a']=++tot;  
 p=ch[p][c-'a'];  
 }  
 if(!idx[p]) idx[p]=++pidx;  
 sidx[i]=idx[p];  
 }  
 //构建tire  
 void build(){  
 queue<int> q;  
 for(int i=0;i<26;i++){  
 if(ch[0][i]) q.push(ch[0][i]);  
 }  
 while(!q.empty()){  
 int u=q.front();q.pop();  
 for(int i=0;i<26;i++){  
 int v=ch[u][i];  
 if(v)   
 {  
 ne[v]=ch[ne[u]][i];  
 q.push(v);deg[ch[ne[u]][i]]++;  
 }//构建回跳边  
 else ch[u][i]=ch[ne[u]][i];//构建转移边(压缩fail指针)  
 }  
 }  
 }  
 //统计主串中模式串的出现次数  
 void query(string s){  
 for(int k=0,i=0;k<s.size();k++){  
 i=ch[i][s[k]-'a'];//走树边/转移边  
 ans[i]++;  
 }  
 }  
 void topu(){  
 queue<int> q;  
 for(int i=0;i<=tot;i++){  
 if(!deg[i]) q.push(i);  
 }  
 while(!q.empty()){  
 int u=q.front();q.pop();  
 fin[idx[u]]=ans[u];  
 ans[ne[u]]+=ans[u];  
 if(--deg[ne[u]]==0) q.push(ne[u]);  
 }  
 }  
 int qans(int i){  
 return fin[sidx[i]];  
 }  
 vector<int> getans(int k){  
 vector<int> res(k+1,0);  
 for(int i=1;i<=k;i++) res[i]=qans(i);  
 return res;  
 }  
};  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
 int n;  
 while(true){  
 cin>>n;  
 if(n==0) break;  
 AC ac(n\*70);  
 vector<string> p(n+1);  
 for(int i=1;i<=n;i++){  
 string s;cin>>s;  
 p[i]=s;  
 ac.insert(s,i);  
 }  
 ac.build();  
 string l;  
 cin>>l;ac.query(l);  
 ac.topu();  
 vector<int> res=ac.getans(n);  
 int maxx=\*max\_element(res.begin(),res.end());  
 cout<<maxx<<endl;  
 for(int i=1;i<=n;i++){  
 if(res[i]==maxx) cout<<p[i]<<endl;  
 }  
 }  
 return 0;  
}  
//AC自动机 处理多模式串匹配问题  
//具体来说 就是给定多个模式串和一个文本串  
//求文本串中有多少个模式串的出现  
//AC自动机是Trie树和KMP的结合  
//时间复杂度构建：O(n+26n) n是模式串总长   
//时间复杂度匹配：O(m) m是文本串长度  
//时间复杂度查询：O(m\*p) m是文本串长度 p是一个串的后缀串的数量  
//-->查询可以通过fail树上做树dp/拓扑排序达到O(n+m)的时间复杂度  
//ac自动机的结构其实就是一个 trans 函数，而构建好这个函数后，在匹配字符串的过程中，我们会舍弃部分前缀达到最低限度的匹配。  
//本质上就是一个状态，接受一个输入，转移到另一个状态，  
//注意到fail链构成的图是一个DAG，所以fail链的长度是O(n)的，所以fail指针的构建是O(n)的。

AC自动机(可拓展版)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
//#define int long long //赫赫 要不要龙龙呢  
using namespace std;  
class AC{  
 public:  
 vector<vector<int>> ch;  
 int n,tot; //节点数>=模式串总长  
 vector<int> cnt,ne;  
 AC(int n):n(n),ch(n+1,vector<int>(26,0)),  
 cnt(n+1,0),ne(n+1,0),tot(0){}  
 void insert(string s){  
 int p=0;  
 for(auto c:s){  
 if(!ch[p][c-'a']) ch[p][c-'a']=++tot;  
 p=ch[p][c-'a'];  
 }  
 cnt[p]++;  
 }  
 //构建tire  
 void build(){  
 queue<int> q;  
 for(int i=0;i<26;i++){  
 if(ch[0][i]) q.push(ch[0][i]);  
 }  
 while(!q.empty()){  
 int u=q.front();q.pop();  
 for(int i=0;i<26;i++){  
 int v=ch[u][i];  
 if(v) ne[v]=ch[ne[u]][i],q.push(v);//构建回跳边  
 else ch[u][i]=ch[ne[u]][i];//构建转移边(压缩fail指针)  
 }  
 }  
 }  
 //统计主串中有多少个模式串  
 int query(string s){  
 int ans=0;  
 for(int k=0,i=0;k<s.size();k++){  
 i=ch[i][s[k]-'a'];//走树边/转移边  
 for(int j=i;j&&~cnt[j];j=ne[j]){  
 ans+=cnt[j],cnt[j]=-1;//统计后缀匹配的个数  
 }  
 }  
 return ans;  
 }  
};  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
 int n;cin>>n;  
 AC ac(1e6+5);  
 while(n--){  
 string s;cin>>s;  
 ac.insert(s);  
 }  
 ac.build();  
 string s;cin>>s;  
 cout<<ac.query(s)<<endl;  
 return 0;  
}  
//AC自动机 处理多模式串匹配问题  
//具体来说 就是给定多个模式串和一个文本串  
//求文本串中有多少个模式串的出现  
//AC自动机是Trie树和KMP的结合  
//时间复杂度构建：O(n+26n) n是模式串总长   
//时间复杂度匹配：O(m) m是文本串长度  
//时间复杂度查询：O(m\*p) m是文本串长度 p是一个串的后缀串的数量  
//-->查询可以通过fail树上做树dp/拓扑排序达到O(n+m)的时间复杂度  
//ac自动机的结构其实就是一个 trans 函数，而构建好这个函数后，在匹配字符串的过程中，我们会舍弃部分前缀达到最低限度的匹配。  
//本质上就是一个状态，接受一个输入，转移到另一个状态，  
//注意到fail链构成的图是一个DAG，所以fail链的长度是O(n)的，所以fail指针的构建是O(n)的。

Manacher

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
//#define int long long //赫赫 要不要龙龙呢  
using namespace std;  
class Manacher{  
 public:  
 vector<char> s;   
 vector<int> d;  
 int k,n;  
 Manacher(int n):d(2\*n+5,0),s(2\*n+5){}  
 void manacher(string str){  
 k=0,n=str.size();  
 str="x"+str;  
 s[0]='$',s[++k]='#';  
 for(int i=1;i<=n;i++){  
 s[++k]=str[i],s[++k]='#';  
 }  
 d[1]=1;  
 for(int i=2,l,r=1;i<=k;i++){  
 if(i<=r) d[i]=min(d[r-i+l],r-i+1);  
 while(s[i+d[i]]==s[i-d[i]]) d[i]++;  
 if(i+d[i]-1>r) l=i-d[i]+1,r=i+d[i]-1;  
 //与exkmp一致 不再赘述  
 }  
 }  
 int get\_max(){  
 int maxn=0;  
 for(int i=1;i<=k;i++){  
 maxn=max(maxn,d[i]);  
 }  
 return maxn-1;  
 }  
};  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
 Manacher ma(1.1e7+5);  
 string str;  
 cin>>str;  
 ma.manacher(str);  
 cout<<ma.get\_max()<<endl;  
 return 0;  
}  
//Manacher算法 求最长回文子串长度 O(n)

exKMP

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
#define int long long //赫赫 要不要龙龙呢  
using namespace std;  
class exkmp{  
 public:  
 vector<int> z,p;  
 string s1,s2;  
 int len1,len2;  
 exkmp(int n):z(n+5,0),p(n+5,0){}  
 //z[i]表示s[i,len]与s[1,len]的最长公共前缀长度  
 //加速盒 右端点最靠右的LCP区间  
 //p[i]表示s2[i,len]与s[1,len]的最长公共前缀长度  
 void getz(string s){  
 len1=s.size();  
 s="x"+s;s1=s;  
 z[1]=len1;  
 for(int i=2,l=0,r=0;i<=len1;i++){  
 if(i<=r) z[i]=min(z[i-l+1],r-i+1);//case1+2  
 //case1:i在l~r内，对称的区间长度<=加速盒  
 //case2:i在l~r内，对称的区间长度>加速盒  
 //case3:i在l~r外  
 while(s[z[i]+1]==s[i+z[i]]) z[i]++;  
 //暴力扩展(case2+3）  
 if(i+z[i]-1>r) l=i,r=i+z[i]-1;  
 //更新加速盒  
 //printf("i=%d z=%d [%d %d]\n",i,z[i],l,r);  
 }  
 }  
 void getp(string oths){  
 len2=oths.size();  
 oths="x"+oths,s2=oths;  
 for(int i=1,l,r=0;i<=len2;i++)  
 {  
 if(i<=r) p[i]=min(z[i-l+1],r-i+1);  
 while(i+p[i]<=len2&&1+p[i]<=len1&&s1[p[i]+1]==s2[i+p[i]]) p[i]++;  
 if(i+p[i]-1>r) l=i,r=i+p[i]-1;  
 }  
 }  
};  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
 exkmp exk(2e7+5);  
 string s1,s2;  
 cin>>s1>>s2;  
 exk.getz(s2);  
 exk.getp(s1);  
 int ans1=0,ans2=0;  
 for(int i=1;i<=s2.size();i++)  
 ans1^=(i\*(exk.z[i]+1));  
 for(int i=1;i<=s1.size();i++)  
 ans2^=(i\*(exk.p[i]+1));  
 cout<<ans1<<endl<<ans2<<endl;  
 return 0;  
}  
//exkmp   
//z[i]表示s[i,len]与s[1,len]的最长公共前缀长度  
//加速盒 右端点最靠右的LCP区间  
//p[i]表示s2[i,len]与s[1,len]的最长公共前缀长度  
//o(n)

kmp

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
using namespace std;  
vector<int> prefix\_init\_f(string s) //前缀函数初始化  
// 前缀函数就是，子串s[0..i]最长的相等的真前缀与真后缀的长度。  
// 在kmp算法中，前缀函数是核心，它决定了模式串（key）在匹配过程若不匹配应该跳转的位置。  
// e.g. abcabc的前缀函数为[0,0,0,1,2,3]  
{  
 int len=s.length();  
 vector<int> dp(len,0);  
 dp[0]=0;  
 for(int i=1;i<len;i++)  
 {  
 int j=dp[i-1];  
 while(j>0&&s[i]!=s[j]) j=dp[j-1];//如果s[i]和s[j]不相同，j跳到前一个符合的位置  
 if(s[i]==s[j]) j++; //如果s[i]和s[j]相同，j+1  
 dp[i]=j;  
 }  
 return dp;  
}   
void kmp(string s,string key)  
{  
 if(key.length()==0) return;  
 vector<int> dp=prefix\_init\_f(key);  
 int i=0,j=0;  
 while(i<s.length())  
 {  
 if(s[i]==key[j]) {i++;j++;} //如果匹配，接着匹配下一个字符  
 else if(j>0) j=dp[j-1]; //如果不匹配，j跳到前一个符合的位置  
 else i++;  
 if(j==key.length())  
 {  
 /\*some operation\*/  
 j=dp[j-1];//匹配成功后，j跳到前一个符合的位置  
 }  
 }  
}  
int main()  
{  
 int T\_start=clock();  
   
 return 0;  
}

tiretree

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
#include <cassert>  
using namespace std;  
class Trie{  
public:  
 struct Node{  
 int son[26],a1,a2;  
 Node(){  
 memset(son,0,sizeof(son));  
 a1=a2=0;  
 }  
 };  
 vector<Node> trie;  
 int tot,root;  
 long long fin;  
 Trie(int len):trie(len+5),tot(0),root(0),fin(0){}  
 void ins(string s,int op)  
 {  
 int p=root;  
 for(auto c:s)  
 {  
 int id=c-'a';  
 if(!trie[p].son[id]) trie[p].son[id]=++tot;  
 p=trie[p].son[id];  
 if(op==1) trie[p].a1++;  
 else trie[p].a2++;  
 }  
 }  
 void dfs(int p)  
 {  
 fin+=1ll\*trie[p].a1\*trie[p].a2;  
 for(int i=0;i<26;i++)  
 {  
 if(trie[p].son[i])  
 {  
 dfs(trie[p].son[i]);  
 }  
 }  
 }  
};  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
 int n;cin>>n;  
 Trie trie(1e6+5);  
 long long fin=0;  
 for(int i=1;i<=n;i++)  
 {  
 string s;cin>>s;  
 fin+=s.size();  
 trie.ins(s,1);  
 reverse(s.begin(),s.end());  
 trie.ins(s,2);  
 }  
 fin=2ll\*n\*fin;  
 //cout<<fin<<endl;  
 trie.dfs(trie.root);  
 cout<<fin-2\*trie.fin<<endl;  
 return 0;  
}

# 动态规划

数位dp(例题1,数位和)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
#define int long long //赫赫 要不要龙龙呢  
using namespace std;  
const int mod=1e9+7;;  
template <int MOD>  
struct SMC {  
 int64\_t val;  
 constexpr SMC(int64\_t v=0){  
 val=(v%MOD+MOD)%MOD;  
 }  
 SMC& operator=(int64\_t v){  
 val=(v%MOD+MOD)%MOD;  
 return \*this;  
 }  
 SMC& operator+=(const SMC &rhs){  
 val+=rhs.val;  
 if(val>=MOD) val-=MOD;  
 return \*this;  
 }  
 SMC& operator-=(const SMC &rhs){  
 val-=rhs.val;  
 if(val<0) val+=MOD;  
 return \*this;  
 }  
 SMC& operator\*=(const SMC &rhs){  
 val=1LL\*val\*rhs.val%MOD;  
 return \*this;  
 }  
 static int64\_t qpow(int64\_t a,int64\_t b){  
 int64\_t res=1;  
 while(b){  
 if(b&1) res=res\*a%MOD;  
 a=a\*a%MOD;  
 b>>=1;  
 }  
 return res;  
 }  
 SMC pow(int64\_t k) const{  
 return SMC(qpow(val,k));  
 }  
 SMC inv() const{  
 return pow(MOD-2);  
 }  
 SMC& operator/=(const SMC &rhs){  
 return \*this\*=rhs.inv();  
 }  
 friend SMC operator+(SMC a,const SMC &b){ return a+=b;}  
 friend SMC operator-(SMC a,const SMC &b){ return a-=b;}  
 friend SMC operator\*(SMC a,const SMC &b){ return a\*=b;}  
 friend SMC operator/(SMC a,const SMC &b){ return a/=b;}  
 SMC& operator++() { return \*this += 1; }  
 SMC& operator--() { return \*this -= 1; }  
 SMC operator++(int32\_t dummy) { SMC t=\*this; ++\*this; return t; }  
 SMC operator--(int32\_t dummy) { SMC t=\*this; --\*this; return t; }  
 friend bool operator==(const SMC &a,const SMC &b){ return a.val==b.val;}  
 friend bool operator<(const SMC &a,const SMC &b){ return a.val<b.val;}  
 friend bool operator>(const SMC &a,const SMC &b){ return a.val>b.val;}  
 friend bool operator<=(const SMC &a,const SMC &b){ return a.val<=b.val;}  
 friend bool operator>=(const SMC &a,const SMC &b){ return a.val>=b.val;}  
 friend bool operator!=(const SMC &a,const SMC &b){ return a.val!=b.val;}  
  
 friend std::istream& operator>>(std::istream &in,SMC &a){  
 int64\_t v;  
 in>>v,a=SMC(v);  
 return in;  
 }  
  
 friend std::ostream& operator<<(std::ostream &out,const SMC &a){  
 out<<a.val;  
 return out;  
 }  
 explicit operator long long() const{  
 return val;  
 }  
 SMC operator-() const{  
 return SMC(-val);  
 }  
 SMC& operator+=(int64\_t x) { return \*this+=SMC(x); }  
 SMC& operator-=(int64\_t x) { return \*this-=SMC(x); }  
 SMC& operator\*=(int64\_t x) { return \*this\*=SMC(x); }  
 SMC& operator/=(int64\_t x) { return \*this/=SMC(x); }  
  
 friend SMC operator+(SMC a, int64\_t b) { return a+=b; }  
 friend SMC operator-(SMC a, int64\_t b) { return a-=b; }  
 friend SMC operator\*(SMC a, int64\_t b) { return a\*=b; }  
 friend SMC operator/(SMC a, int64\_t b) { return a/=b; }  
  
 friend SMC operator+(int64\_t a, SMC b) { return b+a; }  
 friend SMC operator-(int64\_t a, SMC b) { return SMC(a)-b; }  
 friend SMC operator\*(int64\_t a, SMC b) { return b\*a; }  
 friend SMC operator/(int64\_t a, SMC b) { return SMC(a)/b; }  
};  
using Z=SMC<mod>;  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
 int t;cin>>t;  
 vector<vector<Z>> dp(20,vector<Z>(18\*9+5,-1));  
 //dp[i][j]表示[i+1,len](除低i位的高位)数位和=j时,[1,i]任选的所有方案的数位和  
 while(t--)  
 {  
 int l,r;cin>>l>>r;  
 vector<int> bit;  
 auto work=[&](int x)->int{  
 bit.clear();  
 bit.push\_back(0);//1-based  
 while(x) bit.push\_back(x%10),x/=10;  
 return bit.size()-1;  
 };  
 auto dfs=[&](this auto&& dfs,int pos,bool lim,int sum)->Z{  
 //从len位填到pos+1位,lim表示是否受上界限制,sum表示当前数位和  
 //现在填pos位 也就是说dfs的含义是[pos+1,len]数位和=sum时,pos位受lim限制的方案数  
 if(pos==0) return sum;//第0位,直接返回sum  
 if(!lim&&dp[pos][sum]!=-1) return dp[pos][sum];  
 int up=lim?bit[pos]:9;  
 Z res=0;  
 for(int i=0;i<=up;i++)  
 {  
 res+=dfs(pos-1,lim&&i==up,sum+i);  
 //传递受上界限制的状态  
 }  
 if(!lim) dp[pos][sum]=res;  
 return res;  
 };  
 auto solve=[&](int x)->Z{  
 int len=work(x);  
 return dfs(len,1,0);  
 };  
 cout<<solve(r)-solve(l-1)<<endl;  
 }  
 return 0;  
}  
//数位dp 计算[l,r]内所有数的数位和  
//dfs 形参总结  
//

# 数论

(ex)CRT((扩展)中国剩余定理)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
using namespace std;  
#define int \_\_int128  
// 用于存储 \_\_int128 的字符串表示  
std::string to\_string(\_\_int128 value) {  
 std::string result;  
 bool isNegative = value < 0;  
 value = isNegative ? -value : value;  
  
 do {  
 result.push\_back(static\_cast<char>(value % 10) + '0');  
 value /= 10;  
 } while (value > 0);  
  
 if (isNegative) {  
 result.push\_back('-');  
 }  
  
 std::reverse(result.begin(), result.end());  
 return result;  
}  
  
// 从字符串转换为 \_\_int128  
\_\_int128 to\_int128(const std::string& str) {  
 \_\_int128 result = 0;  
 bool isNegative = str[0] == '-';  
 size\_t start = isNegative ? 1 : 0;  
  
 for (size\_t i = start; i < str.size(); ++i) {  
 result = result \* 10 + (str[i] - '0');  
 }  
  
 return isNegative ? -result : result;  
}  
  
// 重载 >> 操作符以支持 \_\_int128 输入  
std::istream& operator>>(std::istream& in, \_\_int128& value) {  
 std::string str;  
 in >> str;  
 value = to\_int128(str);  
 return in;  
}  
  
// 重载 << 操作符以支持 \_\_int128 输出  
std::ostream& operator<<(std::ostream& out, \_\_int128 value) {  
 out << to\_string(value);  
 return out;  
}  
class exCRT{  
   
 public:  
 exCRT(vector<int> r,vector<int> m){  
 this->r=r;  
 this->m=m;  
 n=r.size();  
 }  
 vector<int> r,m;  
 int x,n;  
 int exgcd(int a,int b,int &x,int &y)//扩展欧几里得  
 {  
 if(b==0)  
 {  
 x=1;y=0;  
 return a;  
 }  
 int d=exgcd(b,a%b,x,y),t=x;  
 x=y;y=t-a/b\*y;  
 return d;  
 }  
 int CRT()  
 {  
 int mul=accumulate(m.begin(),m.end(),1LL,  
 [](int a,int b){return a\*b;}),ans=0;  
 for(int i=0;i<n;i++)  
 {  
 int M=mul/m[i],b,y;  
 exgcd(M,m[i],b,y);  
 ans=(ans+r[i]\*M%mul\*b%mul+mul)%mul;  
 }  
 return (ans%mul+mul)%mul;  
 }  
 int \_exCRT()  
 {  
 int M=m[0],ans=r[0];  
 for(int i=1;i<n;i++)  
 {  
 int a=M,b=m[i];  
 int c=((r[i]-ans)%b+b)%b;  
 int x,y;  
 int gcd=exgcd(a,b,x,y);  
 int bg=b/gcd;  
 if(c%gcd!=0) return -1;  
 x=(x%bg+bg)%bg;  
 x=(x\*c/gcd%bg+bg)%bg;  
 ans+=x\*M;  
 M\*=bg;  
 ans=(ans%M+M)%M;  
 }  
 return (ans%M+M)%M;  
 }  
  
};  
signed main()  
{  
 int T\_start=clock();  
 int n;cin>>n;  
 vector<int> r(n),m(n);  
 for(int i=0;i<n;i++) cin>>m[i]>>r[i];  
 exCRT ex(r,m);  
 cout<<ex.CRT()<<endl;  
 return 0;  
}  
//exCRT 求解同余方程组  
//形式：x≡a1(mod m1),x≡a2(mod m2),...,x≡ak(mod mk)  
//其中m1,m2,...,mk互质(CRT)  
//其中m1,m2,...,mk不互质(\_exCRT)  
//时间复杂度：O(nln(amax))

FFT(快速傅里叶变换)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
#include <complex>  
#define int long long //赫赫 要不要龙龙呢  
using namespace std;  
typedef complex<double> Complex;  
class FFT{  
 public:  
 FFT(){}  
 // struct complex{  
 // double x,y;  
 // complex(double x=0,double y=0):x(x),y(y){}  
 // complex operator+(const complex &a) const{return complex(x+a.x,y+a.y);}  
 // complex operator-(const complex &a) const{return complex(x-a.x,y-a.y);}  
 // complex operator\*(const complex &a) const{return complex(x\*a.x-y\*a.y,x\*a.y+y\*a.x);}  
 // };  
 const double PI=acos(-1);  
 vector<int> R;  
 void fft(vector<Complex> &a,int n,int op){  
 for(int i=0;i<n;i++) R[i]=(R[i>>1]>>1)|((i&1)\*(n>>1));  
 //=R[i] = R[i/2]/2 + ((i&1)?n/2:0);  
 for(int i=0;i<n;i++) if(i<R[i]) swap(a[i],a[R[i]]);  
 for(int i=2;i<=n;i<<=1){  
 int m=i>>1;  
 Complex w1(cos(2\*PI/i),op\*sin(2\*PI/i));  
 for(int j=0;j<n;j+=i){  
 Complex wk(1,0);  
 for(int k=j;k<j+m;k++){  
 Complex x=a[k],y=wk\*a[k+m];  
 a[k]=x+y;a[k+m]=x-y;  
 wk=wk\*w1;  
 }  
 }  
 }  
 }  
 vector<int> calc(vector<int> a,vector<int> b){  
 int n=a.size(),m=b.size();  
 int len=1;  
 while(len<n+m-1) len<<=1;  
 R.clear();R.resize(len);  
 vector<Complex> fa(len),fb(len);  
 for(int i=0;i<n;i++) fa[i].real(a[i]);  
 for(int i=0;i<m;i++) fb[i].real(b[i]);  
 fft(fa,len,1);fft(fb,len,1);  
 for(int i=0;i<len;i++) fa[i]=fa[i]\*fb[i];  
 fft(fa,len,-1);  
 vector<int> ans(n+m-1);  
 for(int i=0;i<n+m-1;i++) ans[i]=(int)(fa[i].real()/len+0.5);  
 return ans;  
 }  
};  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
 int n,m;cin>>n>>m;  
 vector<int> a(n+1),b(m+1);  
 for(int i=0;i<=n;i++) cin>>a[i];  
 for(int i=0;i<=m;i++) cin>>b[i];  
 FFT fft;  
 vector<int> ans=fft.calc(a,b);  
 for(int i=0;i<ans.size();i++) cout<<ans[i]<<" ";  
 return 0;  
}

乘法逆元

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
using namespace std;  
int ModQpow(int a,int b,int m)//快速幂  
{  
 int ans=1;  
 while(b)  
 {  
 if(b&1) ans=ans\*a%m;  
 a=a\*a%m;b>>=1;  
 }  
 return ans;  
}  
int invMod1(int a,int m)//a在模m意义下的逆元（费马小定理，m为质数），即a^(m-2)  
{  
 return ModQpow(a,m-2,m);  
}  
int exgcd(int a,int b,int &x,int &y)//扩展欧几里得  
{  
 if(b==0)  
 {  
 x=1;y=0;  
 return a;  
 }  
 int d=exgcd(b,a%b,x,y),t=x;  
 x=y;y=t-a/b\*y;  
 return d;  
}  
int invMod2(int a,int m)//a在模m意义下的逆元（扩展欧几里得）  
{  
 int x,y;  
 exgcd(a,m,x,y);  
 return (x%m+m)%m;  
}  
int main()  
{  
 int T\_start=clock();  
   
 return 0;  
}

安全取模类

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
#define int long long //赫赫 要不要龙龙呢  
using namespace std;  
const int mod=998244353;  
template <int MOD>  
struct SMC {  
 int64\_t val;  
 constexpr SMC(int64\_t v=0){  
 val=(v%MOD+MOD)%MOD;  
 }  
 SMC& operator=(int64\_t v){  
 val=(v%MOD+MOD)%MOD;  
 return \*this;  
 }  
 SMC& operator+=(const SMC &rhs){  
 val+=rhs.val;  
 if(val>=MOD) val-=MOD;  
 return \*this;  
 }  
 SMC& operator-=(const SMC &rhs){  
 val-=rhs.val;  
 if(val<0) val+=MOD;  
 return \*this;  
 }  
 SMC& operator\*=(const SMC &rhs){  
 val=1LL\*val\*rhs.val%MOD;  
 return \*this;  
 }  
 static int64\_t qpow(int64\_t a,int64\_t b){  
 int64\_t res=1;  
 while(b){  
 if(b&1) res=res\*a%MOD;  
 a=a\*a%MOD;  
 b>>=1;  
 }  
 return res;  
 }  
 SMC pow(int64\_t k) const{  
 return SMC(qpow(val,k));  
 }  
 SMC inv() const{  
 return pow(MOD-2);  
 }  
 SMC& operator/=(const SMC &rhs){  
 return \*this\*=rhs.inv();  
 }  
 friend SMC operator+(SMC a,const SMC &b){ return a+=b;}  
 friend SMC operator-(SMC a,const SMC &b){ return a-=b;}  
 friend SMC operator\*(SMC a,const SMC &b){ return a\*=b;}  
 friend SMC operator/(SMC a,const SMC &b){ return a/=b;}  
 SMC& operator++() { return \*this += 1; }  
 SMC& operator--() { return \*this -= 1; }  
 SMC operator++(int32\_t dummy) { SMC t=\*this; ++\*this; return t; }  
 SMC operator--(int32\_t dummy) { SMC t=\*this; --\*this; return t; }  
 friend bool operator==(const SMC &a,const SMC &b){ return a.val==b.val;}  
 friend bool operator<(const SMC &a,const SMC &b){ return a.val<b.val;}  
 friend bool operator>(const SMC &a,const SMC &b){ return a.val>b.val;}  
 friend bool operator<=(const SMC &a,const SMC &b){ return a.val<=b.val;}  
 friend bool operator>=(const SMC &a,const SMC &b){ return a.val>=b.val;}  
 friend bool operator!=(const SMC &a,const SMC &b){ return a.val!=b.val;}  
  
 friend std::istream& operator>>(std::istream &in,SMC &a){  
 int64\_t v;  
 in>>v,a=SMC(v);  
 return in;  
 }  
  
 friend std::ostream& operator<<(std::ostream &out,const SMC &a){  
 out<<a.val;  
 return out;  
 }  
 explicit operator long long() const{  
 return val;  
 }  
 SMC operator-() const{  
 return SMC(-val);  
 }  
 SMC& operator+=(int64\_t x) { return \*this+=SMC(x); }  
 SMC& operator-=(int64\_t x) { return \*this-=SMC(x); }  
 SMC& operator\*=(int64\_t x) { return \*this\*=SMC(x); }  
 SMC& operator/=(int64\_t x) { return \*this/=SMC(x); }  
  
 friend SMC operator+(SMC a, int64\_t b) { return a+=b; }  
 friend SMC operator-(SMC a, int64\_t b) { return a-=b; }  
 friend SMC operator\*(SMC a, int64\_t b) { return a\*=b; }  
 friend SMC operator/(SMC a, int64\_t b) { return a/=b; }  
  
 friend SMC operator+(int64\_t a, SMC b) { return b+a; }  
 friend SMC operator-(int64\_t a, SMC b) { return SMC(a)-b; }  
 friend SMC operator\*(int64\_t a, SMC b) { return b\*a; }  
 friend SMC operator/(int64\_t a, SMC b) { return SMC(a)/b; }  
};  
using Z=SMC<mod>;  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
   
 return 0;  
}

数论预处理

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
#define int long long //赫赫 要不要龙龙呢  
using namespace std;  
class Pre{  
public:  
 int n,tag;  
 const int mod=1e9+7;  
 vector<int> inv,fac,invfac;  
 Pre(int n):n(n){}  
 Pre(int n,int tag):n(n),tag(tag){  
 inv.resize(n+1);  
 fac.resize(n+1);  
 invfac.resize(n+1);  
 preC();  
 }  
 int ModQpow(int a,int b,int m)//快速幂  
 {  
 int ans=1;  
 while(b)  
 {  
 if(b&1)ans=ans\*a%m;  
 a=a\*a%m,b>>=1;  
 }  
 return ans;  
 }  
 //O(nlnn)求1-n所有数的约数  
 vector<vector<int>> divs()  
 {  
 vector<vector<int>> ans(n+1);  
 for(int i=1;i<=n;i++)  
 {  
 for(int j=i;j<=n;j+=i)  
 {  
 ans[j].push\_back(i);  
 }  
 }  
 return ans;  
 }  
 //O(n)求1-n所有的质数  
 vector<int> primes()  
 {  
 vector<int> primes;  
 vector<bool>v(n+1,0);  
 for(int i=2;i<=n;i++)  
 {  
 if(!v[i])primes.push\_back(i);  
 for(int j=0;j<primes.size()&&primes[j]\*i<=n;j++)  
 {  
 v[primes[j]\*i]=1;  
 if(i%primes[j]==0)break;  
 }  
 }  
 return primes;  
 }  
 //O(n)求0-n的阶乘和阶乘逆元  
 void preC()  
 {  
 inv[1]=1;  
 for(int i=2;i<=n;i++)  
 {  
 inv[i]=(mod-mod/i)\*inv[mod%i]%mod;  
 }  
 fac[0]=invfac[0]=1;  
 for(int i=1;i<=n;i++)  
 {  
 fac[i]=fac[i-1]\*i%mod;  
 invfac[i]=invfac[i-1]\*inv[i]%mod;  
 }  
 }  
 //组合数C(n,m) n个数中选m个  
 int C(int n,int m)  
 {  
 if(n<0||m<0||n<m)return 0;  
 return fac[n]\*invfac[m]%mod\*invfac[n-m]%mod;  
 }  
 //O(n)求1-n的欧拉函数  
 vector<int> euler()  
 {  
 vector<int> phi(n+1);  
 phi[1]=1;  
 vector<int> primes;  
 vector<bool>v(n+1,0);  
 for(int i=2;i<=n;i++)  
 {  
 if(!v[i])primes.push\_back(i),phi[i]=i-1;  
 for(int j=0;j<primes.size()&&primes[j]\*i<=n;j++)  
 {  
 int m=primes[j]\*i;  
 v[m]=1;  
 if(i%primes[j]==0)  
 {  
 phi[m]=phi[i]\*primes[j];  
 break;  
 }  
 else phi[m]=phi[i]\*(primes[j]-1);  
 }  
 }  
 return phi;  
 }  
 //O(n)求1-n的约数个数  
 vector<int> d()  
 {  
 vector<int> a(n+1),d(n+1);  
 vector<int> primes;  
 vector<bool>v(n+1,0);  
 d[1]=1;  
 for(int i=2;i<=n;i++)  
 {  
 if(!v[i])  
 {  
 primes.push\_back(i);  
 a[i]=1,d[i]=2;  
 }  
 for(int j=0;j<primes.size()&&primes[j]\*i<=n;j++)  
 {  
 int m=primes[j]\*i;  
 v[m]=1;  
 if(i%primes[j]==0)  
 {  
 a[m]=a[i]+1;  
 d[m]=d[i]/(a[i]+1)\*(a[m]+1);  
 break;  
 }  
 else  
 {  
 a[m]=1;  
 d[m]=d[i]\*2;  
 }  
 }  
 }  
 return d;  
 }  
 //O(n)求1-n的约数和  
 vector<int> sumd()  
 {  
 vector<int> g(n+1),f(n+1);  
 vector<int> primes;  
 vector<bool>v(n+1,0);  
 g[1]=f[1]=1;  
 for(int i=2;i<=n;i++)  
 {  
 if(!v[i])  
 {  
 primes.push\_back(i);  
 f[i]=g[i]=i+1;  
 }  
 for(int j=0;j<primes.size()&&primes[j]\*i<=n;j++)  
 {  
 int m=primes[j]\*i;  
 v[m]=1;  
 if(i%primes[j]==0)  
 {  
 g[m]=g[i]\*primes[j]+1;  
 f[m]=f[i]\*g[m]/g[i];  
 break;  
 }  
 else  
 {  
 g[m]=primes[j]+1;  
 f[m]=f[i]\*g[m];  
 }  
 }  
 }  
 return f;  
 }  
 //O(n)求1-n的莫比乌斯函数  
 vector<int> mu()  
 {  
 vector<int> mu(n+1);  
 mu[1]=1;  
 vector<int> primes;  
 vector<bool>v(n+1,0);  
 for(int i=2;i<=n;i++)  
 {  
 if(!v[i])primes.push\_back(i),mu[i]=-1;  
 for(int j=0;j<primes.size()&&primes[j]\*i<=n;j++)  
 {  
 int m=primes[j]\*i;  
 v[m]=1;  
 if(i%primes[j]==0)  
 {  
 mu[m]=0;  
 break;  
 }  
 else mu[m]=-mu[i];  
 }  
 }  
 return mu;  
 }  
};  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
   
 return 0;  
}

整除分块

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#define int long long   
using namespace std;  
struct blocknode{  
 int l;  
 int r;  
 int val;  
};  
//对[l,r]的i,floor(n/i)相等  
//n%i=n-i\*floor(n/i)  
//首项n-l\*val 公差-val 项数r-l+1  
class divb{  
public:  
 struct node{  
 int l,r;  
 int val1,val2;  
 };  
 int s,e;  
 vector<node> a;  
 divb(int s,int e):s(s),e(e){}  
 void b1(int n,int m){  
 for(int l=s,r;l<=e;l=r+1)  
 {  
 r=min(n/(n/l),m/(m/l));  
 a.push\_back({l,r,n/l,m/l});  
 }  
 }  
 void b2(int n)  
 {  
 for(int l=s,r;l<=e;l=r+1)  
 {  
 r=n/(n/l);  
 a.push\_back({l,r,n/l,0});  
 }  
 }  
};  
signed main()  
{  
 int T\_start=clock();  
 int n,k;cin>>n>>k;  
 vector<blocknode>a;  
 for(int l=1,r;l<=n;l=r+1)  
 {  
 blocknode tp;  
 r=min(n/(n/l),n);  
 tp.l=l;tp.r=r;  
 tp.val=n/l;  
 a.push\_back(tp);  
 }  
 //for(auto i:a)  
 //{  
 //cout<<i.l<<' '<<i.r<<' '<<i.val<<endl;  
 //}  
 return 0;  
}

矩阵快速幂

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
using namespace std;  
#define int long long  
const int mod=1e9+7;  
class MaTQpow{  
 public:  
 vector<vector<int>> mat;  
 int \_mod;  
 MaTQpow(vector<vector<int>> \_mat,int mod):mat(\_mat),\_mod(mod){}  
 MaTQpow(int n,int mod):\_mod(mod)  
 {  
 mat.resize(n,vector<int>(n,0));  
 for(int i=0;i<n;i++) mat[i][i]=1;  
 \_mod=mod;  
 }  
 MaTQpow operator\*(const MaTQpow& other)const{  
 vector<vector<int>> res(mat.size(),vector<int>(other.mat[0].size(),0));  
 for(int i=0;i<mat.size();i++)  
 {  
 for(int j=0;j<other.mat[0].size();j++)  
 {  
 for(int k=0;k<other.mat.size();k++)  
 {  
 res[i][j]=(res[i][j]+mat[i][k]\*other.mat[k][j])%\_mod;  
 }  
 }  
 }  
 return MaTQpow(res,\_mod);  
 }  
 MaTQpow Qpow(int n){  
 MaTQpow res(mat.size(),\_mod);  
 MaTQpow base(mat,\_mod);  
 while(n)  
 {  
 if(n&1) res=res\*base;  
 base=base\*base;  
 n>>=1;  
 }  
 return res;  
 }  
 vector<vector<int>> get(int n){  
 return Qpow(n).mat;  
 }  
};  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 int n;cin>>n;  
 vector<int> a(n+1);  
 for(int i=1;i<=n;i++) cin>>a[i];  
 int c,m,k,t;cin>>c>>m>>k>>t;c%=m;  
 vector<int> dp(m,0);  
 for(int i=1;i<=n;i++)  
 {  
 vector<int> ndp(m,0);  
 ndp[a[i]%m]=1;  
 for(int j=0;j<m;j++)  
 {  
 if(dp[j])  
 {  
 ndp[(j+a[i])%m]=(ndp[(j+a[i])%m]+dp[j])%mod;  
 }  
 }  
 for(int j=0;j<m;j++) dp[j]=(dp[j]+ndp[j])%mod;  
 }  
 vector<vector<int>> p(m,vector<int>(m,0));  
 for(int i=0;i<m;i++)  
 {  
 for(int j=0;j<m;j++)  
 {  
 p[i][(i\*j)%m]=(p[i][(i\*j)%m]+dp[j])%mod;  
 }  
 }  
 MaTQpow mat(p,mod);  
 auto res=mat.Qpow(t);  
 cout<<res.mat[c][k]<<endl;  
 return 0;  
}  
//矩阵快速幂：处理快速形式变换  
//时间复杂度：O(n^3logk)

筛法求积性函数

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
//#define int long long //赫赫 要不要龙龙呢  
using namespace std;  
vector<int> primes(int n)  
{  
 //积性函数gcd(a,b)=1,f(ab)=f(a)f(b)  
 //当f(p^k),p为质数时时的函数值可以快速求出，  
 //即可以通过递推求出所有积性函数  
 vector<int> primes;  
 vector<bool>v(n+1,0);  
 for(int i=2;i<=n;i++)  
 {  
 if(!v[i])  
 {  
 primes.push\_back(i);  
 //考虑f(p)=...  
 //单个质数的情况  
 }  
 for(int j=0;j<primes.size()&&primes[j]\*i<=n;j++)  
 {  
 v[primes[j]\*i]=1;  
 int m=primes[j]\*i;  
 if(i%primes[j]==0)  
 {  
 //此时p[j]是m的最小质因子,运用反证法p[j]也是m的最小质因子  
 //考虑f(m)=...  
 //多个质因子的情况  
 break;  
 }  
 else{  
 //此时gcd(i,pj)=1,f(m)=f(i)f(pj)  
 //新增质因子的情况  
 }  
 }  
 }  
 return primes;  
}  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
 return 0;  
}

线性基(gauss)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
#define int long long //赫赫 要不要龙龙呢  
using namespace std;  
class basic{  
 public:  
 vector<int> num,bas;  
 int bit,cnt,n;  
 basic(vector<int> a,int bit):  
 num(a),bit(bit),cnt(0),n(a.size()){  
 gauss();  
 //usually bit=32/64  
 }  
 void gauss(){  
 for(int i=bit-1;i>=0;i--){  
 //把当前第i位是1的数换上去  
 for(int j=cnt;j<n;j++){  
 if(num[j]>>i&1){  
 swap(num[j],num[cnt]);  
 break;  
 }  
 }  
 //如果这一位全0，跳过  
 if((num[cnt]>>i&1)==0) continue;  
 //消去其他数第i位1  
 for(int j=0;j<n;j++){  
 if(j!=cnt&&(num[j]>>i&1))   
 num[j]^=num[cnt];  
 }  
 cnt++;  
 if(cnt==n) break;  
 }  
 bas.assign(num.begin(),num.begin()+cnt);  
 }  
 //求第k小的数 k:1base  
 int kth(int k){  
 //k个基向量能构造出2^k-1个数  
 //case1 :cnt<n 意味这能构造出 0 所以能构造2^k个数  
 //case2 :cnt=n 意味这不能构造出 0 所以只能构造2^k-1个数  
 if(cnt<n) k--;  
 if(k>=(1ll<<cnt)) return -1;  
 int ans=0;  
 for(int i=0;i<cnt;i++){  
 if(k>>i&1) ans^=bas[cnt-1-i];  
 }  
 return ans;  
 }  
 //求一个数用一个数列异或得到的方案数  
 //约简为0的向量是不必要的 于是可以任选  
 int count(int x){  
 for(auto b:bas){  
 if(x&b) x^=b;  
 }  
 return x==0?(1ll<<(n-cnt)):0;  
 }  
 //求一个数在数列xor和中的排名  
 int rk(int x){  
 int tp=x;  
 for(auto b:bas){  
 if(tp&b) tp^=b;  
 }  
 if(tp) return -1;  
 int id=0;  
 for(int i=0;i<cnt;i++){  
 if(x&(bas[i]))  
 {  
 id|=(1ll<<(cnt-1-i));  
 x^=bas[i];  
 }  
 }  
 if(cnt<n) id++;  
 return id;  
 }  
};  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
 int n;cin>>n;  
 vector<int> a(n);  
 for(int i=0;i<n;i++) cin>>a[i];  
 basic b(a,64);  
 int ans=0;  
 for(auto i:b.bas) ans^=i;  
 cout<<ans<<endl;  
 return 0;  
}  
//异或线性基 O(bit\*n)  
//异或线性基是原数列的一个基向量，  
//意味这基向量的线性组合能构造出原数列的任意数  
//意味着原数列线性组合构造出的数和线性基线性组合构造出的数是一样的  
//xor=mod 2+/GF(2)域  
//guass消元法给出的线性基是行最简式  
//即形如  
//01001  
//00100  
//00011  
//00000  
//满足三个性质：  
//1.线性基中任意两个基向量的异或结果不会是0  
//2.线性基每一个基向量的高位1在别的基向量中都是0  
//3.基向量是从大到小存储的

线性基(贪心法)

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
#define int long long //赫赫 要不要龙龙呢  
using namespace std;  
class basic{  
 public:  
 struct node  
 {  
 int val;  
 int inf;  
 };  
 vector<node> bas; int tot;  
 basic(int bit):bas(bit,{0,0}),tot(0){}  
 void ins(node x)  
 {  
 tot++;  
 for(int i=63;i>=0;i--)  
 {  
 if(x.val>>i&1)  
 {  
 if(bas[i].val==0)  
 {  
 bas[i]=x;  
 return;  
 }  
 else x.val^=bas[i].val;  
 }  
 }  
 }  
};  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
 int n;cin>>n;  
 vector<pair<int,int>> a(n);  
 for(auto& [x,y]:a) cin>>x>>y;  
 sort(a.begin(),a.end(),[](auto& x,auto& y){return x.second>y.second;});  
 basic b(64);  
 for(auto& [x,y]:a)  
 {  
 b.ins({x,y});  
 }  
 int ans=0;  
 for(auto [x,y]:b.bas)  
 {  
 ans+=y;  
 }  
 cout<<ans<<endl;  
 return 0;  
}  
//贪心法构造的线性基：  
//按照元素顺序构造，适用于依赖元素顺序的题

组合数预处理

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
//#define int long long //赫赫 要不要龙龙呢  
using namespace std;  
const int mod=1e9+7;  
template <int MOD>  
struct SMC {  
 int64\_t val;  
 constexpr SMC(int64\_t v=0){  
 val=(v%MOD+MOD)%MOD;  
 }  
 SMC& operator=(int64\_t v){  
 val=(v%MOD+MOD)%MOD;  
 return \*this;  
 }  
 SMC& operator+=(const SMC &rhs){  
 val+=rhs.val;  
 if(val>=MOD) val-=MOD;  
 return \*this;  
 }  
 SMC& operator-=(const SMC &rhs){  
 val-=rhs.val;  
 if(val<0) val+=MOD;  
 return \*this;  
 }  
 SMC& operator\*=(const SMC &rhs){  
 val=1LL\*val\*rhs.val%MOD;  
 return \*this;  
 }  
 static int64\_t qpow(int64\_t a,int64\_t b){  
 int64\_t res=1;  
 while(b){  
 if(b&1) res=res\*a%MOD;  
 a=a\*a%MOD;  
 b>>=1;  
 }  
 return res;  
 }  
 SMC pow(int64\_t k) const{  
 return SMC(qpow(val,k));  
 }  
 SMC inv() const{  
 return pow(MOD-2);  
 }  
 SMC& operator/=(const SMC &rhs){  
 return \*this\*=rhs.inv();  
 }  
 friend SMC operator+(SMC a,const SMC &b){ return a+=b;}  
 friend SMC operator-(SMC a,const SMC &b){ return a-=b;}  
 friend SMC operator\*(SMC a,const SMC &b){ return a\*=b;}  
 friend SMC operator/(SMC a,const SMC &b){ return a/=b;}  
 SMC& operator++() { return \*this += 1; }  
 SMC& operator--() { return \*this -= 1; }  
 SMC operator++(int32\_t dummy) { SMC t=\*this; ++\*this; return t; }  
 SMC operator--(int32\_t dummy) { SMC t=\*this; --\*this; return t; }  
 friend bool operator==(const SMC &a,const SMC &b){ return a.val==b.val;}  
 friend bool operator<(const SMC &a,const SMC &b){ return a.val<b.val;}  
 friend bool operator>(const SMC &a,const SMC &b){ return a.val>b.val;}  
 friend bool operator<=(const SMC &a,const SMC &b){ return a.val<=b.val;}  
 friend bool operator>=(const SMC &a,const SMC &b){ return a.val>=b.val;}  
 friend bool operator!=(const SMC &a,const SMC &b){ return a.val!=b.val;}  
  
 friend std::istream& operator>>(std::istream &in,SMC &a){  
 int64\_t v;  
 in>>v,a=SMC(v);  
 return in;  
 }  
  
 friend std::ostream& operator<<(std::ostream &out,const SMC &a){  
 out<<a.val;  
 return out;  
 }  
 explicit operator long long() const{  
 return val;  
 }  
 SMC operator-() const{  
 return SMC(-val);  
 }  
 SMC& operator+=(int64\_t x) { return \*this+=SMC(x); }  
 SMC& operator-=(int64\_t x) { return \*this-=SMC(x); }  
 SMC& operator\*=(int64\_t x) { return \*this\*=SMC(x); }  
 SMC& operator/=(int64\_t x) { return \*this/=SMC(x); }  
  
 friend SMC operator+(SMC a, int64\_t b) { return a+=b; }  
 friend SMC operator-(SMC a, int64\_t b) { return a-=b; }  
 friend SMC operator\*(SMC a, int64\_t b) { return a\*=b; }  
 friend SMC operator/(SMC a, int64\_t b) { return a/=b; }  
  
 friend SMC operator+(int64\_t a, SMC b) { return b+a; }  
 friend SMC operator-(int64\_t a, SMC b) { return SMC(a)-b; }  
 friend SMC operator\*(int64\_t a, SMC b) { return b\*a; }  
 friend SMC operator/(int64\_t a, SMC b) { return SMC(a)/b; }  
};  
using Z=SMC<mod>;  
class Pre{  
 public:  
 int n,m;  
 vector<vector<Z>> s,s2;  
 vector<Z> inv,fac,invfac,d;  
 //c组合数,s第一类斯特林数  
 Pre(int n,int m):n(n),m(m){  
 preS();  
 preC();  
 preD();  
 preS2();  
 }  
 void preS(){  
 s.resize(n+1,vector<Z>(m+1));  
 s[0][0]=1;  
 for(int i=1;i<=n;i++){  
 s[i][0]=0;  
 if(i<=m) s[i][i]=1;  
 for(int j=1;j<=min(m,i);j++){  
 s[i][j]=s[i-1][j]\*(i-1)+s[i-1][j-1];  
 }  
 }  
 }  
 void preC()  
 {  
 fac.resize(n+1);  
 invfac.resize(n+1);  
 inv.resize(n+1);  
 inv[1]=1;  
 for(int i=2;i<=n;i++)  
 {  
 inv[i]=(mod-mod/i)\*inv[mod%i];  
 }  
 fac[0]=invfac[0]=1;  
 for(int i=1;i<=n;i++)  
 {  
 fac[i]=fac[i-1]\*i;  
 invfac[i]=invfac[i-1]\*inv[i];  
 }  
 }  
 void preD(){  
 d.resize(n+1);  
 d[1]=0,d[2]=1;  
 for(int i=3;i<=n;i++){  
 d[i]=(i-1)\*(d[i-1]+d[i-2]);  
 }  
 }  
 void preS2(){  
 s2.resize(n+1,vector<Z>(m+1));  
 s2[0][0]=1;  
 for(int i=1;i<=n;i++){  
 s2[i][0]=0;  
 if(i<=m) s2[i][i]=1;  
 for(int j=1;j<=min(m,i);j++){  
 s2[i][j]=s2[i-1][j]\*j+s2[i-1][j-1];  
 }  
 }  
 }  
 //第一类斯特林数S(n,m) n个不同元素划分为m个非空圆排列的方案数  
 Z S(int i,int j){  
 return s[i][j];  
 }  
 //第二类斯特林数S2(n,m) n个不同元素划分为m个非空子集的方案数  
 Z S2(int i,int j){  
 return s2[i][j];  
 }  
 //排列数 A(n,m) n个数中选m个的排列  
 Z A(int n,int m){  
 if(n<0||m<0||n<m)return 0;  
 return fac[n]\*invfac[n-m];  
 }  
 //组合数C(n,m) n个数中选m个  
 Z C(int n,int m)  
 {  
 if(n<0||m<0||n<m)return 0;  
 return fac[n]\*invfac[m]\*invfac[n-m];  
 }  
 //圆排列数 Q(n,m) n个数中选m个，m个数的圆排列  
 //Q(n,n)=(n-1)!,n个数的圆排列  
 Z Q(int n,int m)  
 {  
 if(n<0||m<0||n<m)return 0;  
 return fac[n]\*invfac[n-m]\*inv[m];  
 }  
 //错位排列数 D(n,m) n个数中选m个，m个数的错位排列  
 //D(n,n)=d[n],n个数的错位排列  
 Z D(int n,int m)  
 {  
 if(n<0||m<0||n<m)return 0;  
 return d[n]\*C(n,m);  
 }  
};  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
 return 0;  
}

# 计算几何

三角剖分

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
#include <cassert>  
//#define int long long //赫赫 要不要龙龙呢  
//#define double long double  
//const double eps=1e-12;  
using namespace std;  
struct pit;struct vec;  
const double eps=1e-8;  
const double pi=acos(-1);  
double R;  
struct pit;  
pit CO;//圆心   
struct vec{  
 double x,y;  
 vec(double x=0,double y=0):x(x),y(y){}  
 vec(pit a) {x=a.x;y=a.y;}//点转向量(OA向量)  
 vec operator+(const vec& o)const{return vec(x+o.x,y+o.y);}  
 vec operator-(const vec& o)const{return vec(x-o.x,y-o.y);}  
 vec operator/(const double& o)const{return vec(x/o,y/o);} //数除  
 vec operator\*(const double& o)const{return vec(x\*o,y\*o);} //数乘  
 double operator\*(const vec& o)const{return x\*o.y-y\*o.x;} //叉积  
 double operator&(const vec& o)const{return x\*o.x+y\*o.y;} //点积  
};  
struct pit  
{  
 double x,y;  
 pit(double x=0,double y=0):x(x),y(y){}  
 vec operator-(const pit& o)const{return vec(x-o.x,y-o.y);}  
 pit operator+(const vec& o)const{return pit(x+o.x,y+o.y);}  
 pit operator+(const pit& o)const{return pit(x+o.x,y+o.y);}  
 pit operator/(const double& o)const{return pit(x/o,y/o);}  
};  
double len(const vec& o){return sqrt(o.x\*o.x+o.y\*o.y);} //向量模长  
double dis(const pit& a,const pit& b){return len(b-a);} //两点距离  
//向量逆时针旋转theta弧度  
vec rotate(const vec& o,double theta){  
 return vec(o.x\*cos(theta)-o.y\*sin(theta),o.x\*sin(theta)+o.y\*cos(theta));  
}   
//单位向量  
vec norm(vec a){  
 return a/len(a);  
}  
//用单位圆证明  
//向量夹角 dot(a,b)=len(a)\*len(b)\*cos(θ)  
double angle(vec a,vec b){  
 double val=(a&b)/len(a)/len(b);  
 val=max(-1.0,min(1.0,val));  
 return acos(val);  
}   
//向量围成的平行四边形面积,b在a的逆时针方向为正，否则为负  
double area(vec a,vec b){return a\*b;}   
//点线关系(点c,直线ab)  
int cross(pit a,pit b,pit c){  
 if((b-a)\*(c-a)>eps) return 1; //c在ab的逆时针方向  
 else if((b-a)\*(c-a)<-eps) return -1; //c在ab的顺时针方向  
 return 0; //c,a,b共线  
}  
//判断点在线段上(p在ab上)  
bool onSeg(pit a,pit b,pit p){  
 return cross(a,b,p)==0&&((a-p)&(b-p))<=eps;  
}  
//OA OB扇形面积  
double sector(vec a,vec b){  
 double angle=acos((a&b)/len(a)/len(b)); //[0,pi]  
 if(a\*b<=-eps) angle=-angle;   
 return angle\*R\*R/2;  
}  
//求两直线ab,cd的交点(两点式)  
pit getNode(pit a,pit b,pit c,pit d){  
 vec u=b-a,v=d-c;  
 //assert(fabs(u\*v)<=eps); //平行 无交点  
 //if(fabs(u\*v)<=eps) return pit(NAN,NAN); //平行 无交点  
 double t=((a-c)\*v)/(u\*v);  
 return a+u\*t;  
}  
//求两直线ab,cd的交点(点向式) a起点u方向向量 c起点v方向向量  
pit getNode(pit a,vec u,pit c,vec v){  
 //assert(fabs(u\*v)<=eps); //平行 无交点  
 //if(fabs(u\*v)<=eps) return pit(NAN,NAN); //平行 无交点  
 double t=((a-c)\*v)/(u\*v);  
 return a+u\*t;  
}  
//计算线段ab与圆的交点和距离(此处的距离是有意义的距离 即线段离圆心的距离)  
double getDP2(pit a,pit b,pit& pa,pit &pb){  
 pit e=getNode(a,b-a,CO,rotate(b-a,pi/2));  
 //圆心到线段的垂足  
 double d=dis(e,CO);  
 if(!onSeg(a,b,e)) d=min(dis(a,CO),dis(b,CO)); //垂足不在线段上  
 if(R-d<=-eps) return d; //线段在圆外 0个交点  
 double h=sqrt(max(0.0,R\*R-d\*d));  
 pa=e+norm(a-b)\*h;  
 pb=e+norm(b-a)\*h;//计算两个交点  
 return d;  
}  
//计算线段ab与圆心构成的三角形与圆的面积交  
double getS(pit a,pit b){  
 if(cross(a,b,CO)==0) return 0; //case1:三点共线  
 double da=dis(a,CO),db=dis(b,CO);  
 if(R-da>=-eps&&R-db>=-eps) return (vec(a))\*(vec(b))/2; //case2:线段在圆内 构成一个三角形  
 pit pa,pb;  
 double d=getDP2(a,b,pa,pb);   
 if(R-d<=-eps) return sector(vec(a),vec(b)); //case3:线段在圆外 构成一个扇形  
 if(R-da>=-eps) return (vec(a))\*(vec(pb))/2+sector(vec(pb),vec(b)); //case4.1:a在圆内 一个三角形+扇形  
 if(R-db>=-eps) return (vec(pa))\*(vec(b))/2+sector(vec(a),vec(pa)); //case4.2:b在圆内 一个三角形+扇形  
 return (vec(pa))\*(vec(pb))/2+sector(vec(a),vec(pa))+sector(vec(b),vec(pb)); //case5:两个端点都在圆内 一个三角形+两个扇形  
}  
//极角排序  
void psort(vector<pit>& a)  
{  
 pit cen(0,0);  
 for(auto& i:a) cen=cen+i;  
 cen=cen/a.size();  
 sort(a.begin(),a.end(),[&](pit a,pit b){  
 double angA=atan2(a.y-cen.y,a.x-cen.x);  
 double angB=atan2(b.y-cen.y,b.x-cen.x);  
 return angA<angB;  
 });  
}  
double S(pit o,double r,vector<pit> a)  
{  
 for(auto& i:a) i.x-=o.x,i.y-=o.y;  
 R=r;CO=pit(0,0);  
 //psort(a); //当且仅当多边形为凸多边形且传入顺序不是正/逆时针时  
 double res=0;  
 for(int i=0;i<a.size();i++)  
 res+=getS(a[i],a[(i+1)%a.size()]);  
 return fabs(res);  
}  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
 return 0;  
}  
//三角剖分：将多边形分割成若干边 求边与圆心构成的三角形  
//通过这个三角形，求多边形与圆的面积交  
//传多边形的时候要逆时针传参(极角排序,当且仅当多边形为凸多边形时成立),同时圆心要平移到(0,0)

凸包

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
//#define int long long //赫赫 要不要龙龙呢  
//#define double long double  
//const double eps=1e-12;  
using namespace std;  
const double eps=1e-8;  
struct vec{  
 double x,y;  
 vec(double x=0,double y=0):x(x),y(y){}  
 vec operator+(const vec& o)const{return vec(x+o.x,y+o.y);}  
 vec operator-(const vec& o)const{return vec(x-o.x,y-o.y);}  
 vec operator/(const double& o)const{return vec(x/o,y/o);} //数除  
 vec operator\*(const double& o)const{return vec(x\*o,y\*o);} //数乘  
 double operator\*(const vec& o)const{return x\*o.y-y\*o.x;} //叉积  
 double operator&(const vec& o)const{return x\*o.x+y\*o.y;} //点积  
};  
struct pit  
{  
 double x,y;  
 pit(double x=0,double y=0):x(x),y(y){}  
 vec operator-(const pit& o)const{return vec(x-o.x,y-o.y);}  
 pit operator+(const vec& o)const{return pit(x+o.x,y+o.y);}  
 pit operator+(const pit& o)const{return pit(x+o.x,y+o.y);}  
 pit operator/(const double& o)const{return pit(x/o,y/o);}  
};  
double len(const vec& o){return sqrt(o.x\*o.x+o.y\*o.y);} //向量模长  
double dis(const pit& a,const pit& b){return len(b-a);} //两点距离  
bool cmp(const pit& a,const pit& b){  
 return fabs(a.x-b.x)>=eps?a.x<b.x:a.y<b.y;  
}  
//ab x ac  
double cross(pit a,pit b,pit c){  
 return (b-a)\*(c-a);  
}  
pair<double,vector<pit>> Andrew(vector<pit> p)  
{  
 sort(p.begin(),p.end(),cmp);  
 vector<pit> st(p.size()+5,{0,0});  
 int top=0;  
 for(int i=0;i<p.size();i++)  
 {  
 while(top>1&&cross(st[top-2],st[top-1],p[i])<=eps)top--;  
 //<=eps 三点共线不算 <=-eps 三点共线算  
 st[top++]=p[i];  
 }//下凸包  
 int k=top;  
 for(int i=p.size()-2;i>=0;i--)  
 {  
 while(top>k&&cross(st[top-2],st[top-1],p[i])<=eps)top--;  
 st[top++]=p[i];  
 }//上凸包  
 double res=0;  
 for(int i=0;i<top-1;i++)res+=len(st[i+1]-st[i]);  
 st.resize(top-1);  
 return {res,st};  
}  
//计算多边形面积  
double TA(vector<pit> p)  
{  
 int n=p.size();  
 double res=0;  
 for(int i=0;i<n;i++){  
 res+=p[i].x\*p[(i+1)%n].y-p[i].y\*p[(i+1)%n].x;  
 }  
 return fabs(res)/2;  
}  
//判断凸包是否逆时针,不然要翻转  
void rev(vector<pit>& p)  
{  
 int n=p.size();  
 double res=0;  
 for(int i=0;i<n;i++){  
 res+=p[i].x\*p[(i+1)%n].y-p[i].y\*p[(i+1)%n].x;  
 }  
 if(res<=-eps) reverse(p.begin(),p.end());  
}  
//判断p点是否在三角形abc内  
bool isCon(pit a,pit b,pit c,pit p){  
 return cross(a,b,p)>=-eps&&cross(b,c,p)>=-eps&&cross(c,a,p)>=-eps;  
}  
//二分判断点是否在凸包内O(logn)  
//需保证凸包逆时针  
bool isConvex(vector<pit> p,pit a)  
{  
 int n=p.size();  
 if(n<3) return false;  
 if((p[1]-p[0])\*(a-p[0])<=-eps) return false;  
 if((p[n-1]-p[0])\*(a-p[0])>=eps) return false;  
 int l=1,r=n-1,idx=-1;  
 while(l<=r)  
 {  
 int mid=(l+r)>>1;  
 if((p[mid]-p[0])\*(a-p[0])>=-eps)  
 {  
 idx=mid;  
 l=mid+1;  
 }  
 else r=mid-1;  
 }  
 if(idx==-1||idx>=n-1) return false;  
 return isCon(p[0],p[idx],p[idx+1],a);  
}  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
   
 return 0;  
}  
//凸包：给定点集，求周长最小凸多边形围住它们 Andrew算法  
//O(nlogn)，不保证逆时针/顺时针，保证有序

向量

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
#include <cassert>  
//#define int long long //赫赫 要不要龙龙呢  
//#define double long double  
//const double eps=1e-12;  
using namespace std;  
const double eps=1e-8;  
struct vec{  
 double x,y;  
 vec(double x=0,double y=0):x(x),y(y){}  
 vec operator+(const vec& o)const{return vec(x+o.x,y+o.y);}  
 vec operator-(const vec& o)const{return vec(x-o.x,y-o.y);}  
 vec operator/(const double& o)const{return vec(x/o,y/o);} //数除  
 vec operator\*(const double& o)const{return vec(x\*o,y\*o);} //数乘  
 double operator\*(const vec& o)const{return x\*o.y-y\*o.x;} //叉积  
 double operator&(const vec& o)const{return x\*o.x+y\*o.y;} //点积  
};  
struct pit  
{  
 double x,y;  
 pit(double x=0,double y=0):x(x),y(y){}  
 vec operator-(const pit& o)const{return vec(x-o.x,y-o.y);}  
 pit operator+(const vec& o)const{return pit(x+o.x,y+o.y);}  
 pit operator+(const pit& o)const{return pit(x+o.x,y+o.y);}  
 pit operator/(const double& o)const{return pit(x/o,y/o);}  
};  
double len(const vec& o){return sqrt(o.x\*o.x+o.y\*o.y);} //向量模长  
double dis(const pit& a,const pit& b){return len(b-a);} //两点距离  
//向量逆时针旋转theta弧度  
vec rotate(const vec& o,double theta){  
 return vec(o.x\*cos(theta)-o.y\*sin(theta),o.x\*sin(theta)+o.y\*cos(theta));  
}   
//向量单位化  
vec norm(vec a){  
 return a/len(a);  
}  
//用单位圆证明  
//向量夹角 dot(a,b)=len(a)\*len(b)\*cos(θ)  
double angle(vec a,vec b){  
 double val=(a&b)/len(a)/len(b);  
 val=max(-1.0,min(1.0,val));  
 return acos(val);  
}   
//向量围成的平行四边形面积,b在a的逆时针方向为正，否则为负  
double area(vec a,vec b){return a\*b;}   
//点线关系(点c,直线ab)  
int cross(pit a,pit b,pit c){  
 if((b-a)\*(c-a)>eps) return 1; //c在ab的逆时针方向  
 else if((b-a)\*(c-a)<-eps) return -1; //c在ab的顺时针方向  
 return 0; //c,a,b共线  
}  
//判断点在线段上(p在ab上)  
bool onSeg(pit a,pit b,pit p){  
 return cross(a,b,p)==0&&((a-p)&(b-p))<=eps;  
}  
//线线关系  
//case1:直线ab与线段cd  
bool lcross(pit a,pit b,pit c,pit d){  
 if(cross(a,b,c)\*cross(a,b,d)>0) return 0;//c,d在ab的同一侧 无交点  
 return 1; //有交点  
}  
//case2:线段ab与线段cd  
bool scross(pit a,pit b,pit c,pit d){  
 if(cross(a,b,c)\*cross(a,b,d)>0||cross(c,d,a)\*cross(c,d,b)>0) return 0;//c,d在ab 或 a,b在cd 的同一侧 无交点  
 return 1; //有交点  
}  
//case3:直线ab与直线cd  
bool pcross(pit a,pit b,pit c,pit d){  
 if(fabs((b-a)\*(d-c))<=eps) return 0; //平行 无交点  
 return 1; //有交点  
}  
//求两直线ab,cd的交点(两点式)  
pit getNode(pit a,pit b,pit c,pit d){  
 vec u=b-a,v=d-c;  
 //assert(fabs(u\*v)<=eps);  
 //if(fabs(u\*v)<=eps) return pit(NAN,NAN); //平行 无交点  
 double t=((a-c)\*v)/(u\*v);  
 return a+u\*t;  
}  
//求两直线ab,cd的交点(点向式) a起点u方向向量 c起点v方向向量  
pit getNode(pit a,vec u,pit c,vec v){  
 //assert(fabs(u\*v)<=eps);  
 //if(fabs(u\*v)<=eps) return pit(NAN,NAN); //平行 无交点  
 double t=((a-c)\*v)/(u\*v);  
 return a+u\*t;  
}  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
   
 return 0;  
}

旋转卡壳

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
//#define int long long //赫赫 要不要龙龙呢  
#define double long double  
const double eps=1e-12;  
using namespace std;  
//const double eps=1e-8;  
const double PI=acos(-1);  
struct vec{  
 double x,y;  
 vec(double x=0,double y=0):x(x),y(y){}  
 vec operator+(const vec& o)const{return vec(x+o.x,y+o.y);}  
 vec operator-(const vec& o)const{return vec(x-o.x,y-o.y);}  
 vec operator/(const double& o)const{return vec(x/o,y/o);} //数除  
 vec operator\*(const double& o)const{return vec(x\*o,y\*o);} //数乘  
 double operator\*(const vec& o)const{return x\*o.y-y\*o.x;} //叉积  
 double operator&(const vec& o)const{return x\*o.x+y\*o.y;} //点积  
};  
struct pit  
{  
 double x,y;  
 pit(double x=0,double y=0):x(x),y(y){}  
 vec operator-(const pit& o)const{return vec(x-o.x,y-o.y);}  
 pit operator+(const vec& o)const{return pit(x+o.x,y+o.y);}  
 pit operator+(const pit& o)const{return pit(x+o.x,y+o.y);}  
 pit operator/(const double& o)const{return pit(x/o,y/o);}  
};  
double len(const vec& o){return sqrt(o.x\*o.x+o.y\*o.y);} //向量模长  
double dis(const pit& a,const pit& b){return len(b-a);} //两点距离  
bool cmp(const pit& a,const pit& b){  
 return fabs(a.x-b.x)>=eps?a.x<b.x:a.y<b.y;  
}  
//向量逆时针旋转theta弧度  
vec rotate(const vec& o,double theta){  
 return vec(o.x\*cos(theta)-o.y\*sin(theta),o.x\*sin(theta)+o.y\*cos(theta));  
}   
//ab x ac  
double cross(pit a,pit b,pit c){  
 return (b-a)\*(c-a);  
}  
//ab·ac  
double dot(pit a,pit b,pit c){  
 return (b-a)&(c-a);  
}  
vec norm(vec a){  
 return a/len(a);  
}  
pair<double,vector<pit>> Andrew(vector<pit> p)  
{  
 sort(p.begin(),p.end(),cmp);  
 vector<pit> st(p.size()+5,{0,0});  
 int top=0;  
 for(int i=0;i<p.size();i++)  
 {  
 while(top>1&&cross(st[top-2],st[top-1],p[i])<=eps)top--;  
 //<=eps 三点共线不算 <=-eps 三点共线算  
 st[top++]=p[i];  
 }//下凸包  
 int k=top;  
 for(int i=p.size()-2;i>=0;i--)  
 {  
 while(top>k&&cross(st[top-2],st[top-1],p[i])<=eps)top--;  
 st[top++]=p[i];  
 }//上凸包  
 double res=0;  
 for(int i=0;i<top-1;i++)res+=len(st[i+1]-st[i]);  
 st.resize(top-1);  
 return {res,st};  
}  
//计算多边形面积  
double TA(vector<pit> p)  
{  
 int n=p.size();  
 double res=0;  
 for(int i=0;i<n;i++){  
 res+=p[i].x\*p[(i+1)%n].y-p[i].y\*p[(i+1)%n].x;  
 }  
 return fabs(res)/2;  
}  
//判断凸包是否逆时针,不然要翻转  
void rev(vector<pit>& p)  
{  
 int n=p.size();  
 double res=0;  
 for(int i=0;i<n;i++){  
 res+=p[i].x\*p[(i+1)%n].y-p[i].y\*p[(i+1)%n].x;  
 }  
 if(res<=-eps) reverse(p.begin(),p.end());  
}  
//判断p点是否在三角形abc内  
bool isCon(pit a,pit b,pit c,pit p){  
 return cross(a,b,p)>=-eps&&cross(b,c,p)>=-eps&&cross(c,a,p)>=-eps;  
}  
//二分判断点是否在凸包内O(logn)  
//需保证凸包逆时针  
bool isConvex(vector<pit> p,pit a)  
{  
 int n=p.size();  
 if(n<3) return false;  
 if((p[1]-p[0])\*(a-p[0])<=-eps) return false;  
 if((p[n-1]-p[0])\*(a-p[0])>=eps) return false;  
 int l=1,r=n-1,idx=-1;  
 while(l<=r)  
 {  
 int mid=(l+r)>>1;  
 if((p[mid]-p[0])\*(a-p[0])>=-eps)  
 {  
 idx=mid;  
 l=mid+1;  
 }  
 else r=mid-1;  
 }  
 if(idx==-1||idx>=n-1) return false;  
 return isCon(p[0],p[idx],p[idx+1],a);  
}  
//双指针/多指针在凸包上找最优->旋转卡壳 形如一个游标卡尺绕着凸包旋转  
//旋转卡壳,用叉积可以找离一条线垂直最高或最低，用点积可以找离一条线水平最左或最右的点（点积的几何意义是b在a的投影长度)  
pair<double,vector<pit>> rot(vector<pit> p)  
{  
 double ans=1e14;vector<pit> fin(4);  
 int n=p.size(),a=1,b=1,c;  
 for(int i=0;i<n;i++)  
 {  
 while(cross(p[i],p[(i+1)%n],p[a])-cross(p[i],p[(i+1)%n],p[(a+1)%n])<=-eps) a=(a+1)%n;  
 while(dot(p[i],p[(i+1)%n],p[b])-dot(p[i],p[(i+1)%n],p[(b+1)%n])<=-eps) b=(b+1)%n;  
 if(i==0) c=a;  
 while(dot(p[(i+1)%n],p[i],p[c])-dot(p[(i+1)%n],p[i],p[(c+1)%n])<=-eps) c=(c+1)%n;  
 double d=dis(p[i],p[(i+1)%n]);  
 double H=fabs(cross(p[a],p[i],p[(i+1)%n]))/d;  
 double R=dot(p[i],p[(i+1)%n],p[b])/d;  
 double L=dot(p[(i+1)%n],p[i],p[c])/d;  
 if(ans>(R+L-d)\*H)  
 {  
 ans=(R+L-d)\*H;  
 vec nor1=norm(p[(i+1)%n]-p[i]);//i->i+1  
 vec nor2=norm(p[i]-p[(i+1)%n]);//i+1->i  
 fin[0]=p[i]+nor1\*R;  
 fin[1]=p[(i+1)%n]+nor2\*L;  
 fin[2]=fin[1]+rotate(nor1,PI/2)\*H;  
 fin[3]=fin[0]+rotate(nor1,PI/2)\*H;  
 }  
 }  
 return {ans,fin};  
}  
void zero(pit& a)  
{  
 if(fabs(a.x)<eps) a.x=0;  
 if(fabs(a.y)<eps) a.y=0;  
}  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
 int n;cin>>n;  
 vector<pit> p(n);  
 for(int i=0;i<n;i++)cin>>p[i].x>>p[i].y;  
 auto [\_,st]=Andrew(p);  
 auto [ans,fin]=rot(st);  
 printf("%.5Lf\n",ans);  
 int k=0;  
 reverse(fin.begin(),fin.end());  
 for(int i=0;i<=3;i++) if(cmp(fin[i],fin[k])) k=i;  
 for(int i=k;i<=k+3;i++)   
 {  
 zero(fin[i%4]);  
 printf("%.5Lf %.5Lf\n",fin[i%4].x,fin[i%4].y);  
 }  
 return 0;  
}  
//凸包：给定点集，求周长最小凸多边形围住它们 Andrew算法  
//O(nlogn)，不保证逆时针/顺时针，保证有序

极角排序

#include <algorithm>  
#include <bitset>  
#include <cmath>  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <ctime>  
#include <deque>  
#include <map>  
#include <iostream>  
#include <queue>  
#include <set>  
#include <stack>  
#include <vector>  
#include <array>  
#include <unordered\_map>  
#include <numeric>  
#include <functional>  
#include <ranges>  
#include <iomanip>  
//#define int long long //赫赫 要不要龙龙呢  
//#define double long double  
//const double eps=1e-12;  
using namespace std;  
struct vec{  
 double x,y;  
 vec(double x=0,double y=0):x(x),y(y){}  
 vec operator+(const vec& o)const{return vec(x+o.x,y+o.y);}  
 vec operator-(const vec& o)const{return vec(x-o.x,y-o.y);}  
 vec operator/(const double& o)const{return vec(x/o,y/o);} //数除  
 vec operator\*(const double& o)const{return vec(x\*o,y\*o);} //数乘  
 double operator\*(const vec& o)const{return x\*o.y-y\*o.x;} //叉积  
 double operator&(const vec& o)const{return x\*o.x+y\*o.y;} //点积  
};  
struct pit  
{  
 double x,y;  
 pit(double x=0,double y=0):x(x),y(y){}  
 vec operator-(const pit& o)const{return vec(x-o.x,y-o.y);}  
 pit operator+(const vec& o)const{return pit(x+o.x,y+o.y);}  
 pit operator+(const pit& o)const{return pit(x+o.x,y+o.y);}  
 pit operator/(const double& o)const{return pit(x/o,y/o);}  
};  
void psort(vector<pit>& a)  
{  
 pit cen(0,0);  
 for(auto& i:a) cen=cen+i;  
 cen=cen/a.size();  
 sort(a.begin(),a.end(),[&](pit a,pit b){  
 double angA=atan2(a.y-cen.y,a.x-cen.x);  
 double angB=atan2(b.y-cen.y,b.x-cen.x);  
 return angA<angB;  
 });  
}  
signed main()  
{  
 int T\_start=clock();  
 //freopen("in.txt","r",stdin);  
 //freopen("out.txt","w",stdout);  
 //ios::sync\_with\_stdio(false),cin.tie(0),cout.tie(0);  
   
 return 0;  
}  
//极角排序：将点按照极角排序(逆时针)，即以某点为极点，将点按照与x轴的夹角从小到大排序