Cwolf9

[ACM模板](#_目录)

[目录 4](#_Toc526875220)

[代码 5](#_Toc526875221)

[图论 5](#_Toc526875222)

[最短路 5](#_Toc526875223)

[Floyd 5](#_Toc526875224)

[Dijkstra求最短路次短路及条数 5](#_Toc526875225)

[A\*求k短路 6](#_Toc526875226)

[拓扑排序输出 7](#_Toc526875227)

[生成树 7](#_Toc526875228)

[(严格)次小生成树 7](#_Toc526875229)

[链上第k小点权 9](#_Toc526875230)

[二分图 11](#_Toc526875231)

[匈牙利 12](#_Toc526875232)

[染色法 12](#_Toc526875233)

[KM 13](#_Toc526875234)

[Tarjan 14](#_Toc526875235)

[模板 14](#_Toc526875236)

[缩点 15](#_Toc526875237)

[点双联通分量 15](#_Toc526875238)

[边双联通分量 16](#_Toc526875239)

[LCA 16](#_Toc526875240)

[2-sat 17](#_Toc526875241)

[树链剖分 18](#_Toc526875242)

[模板 18](#_Toc526875243)

[网络流 21](#_Toc526875244)

[最大流Dinic 21](#_Toc526875245)

[最大流SAP 22](#_Toc526875246)

[最小费用流Dinic 23](#_Toc526875247)

[Zkw 24](#_Toc526875248)

[数据结构 27](#_Toc526875249)

[Bitset 27](#_Toc526875250)

[线段树区间合并 27](#_Toc526875251)

[线段树 28](#_Toc526875252)

[矩形面积并(全部) 28](#_Toc526875253)

[矩形面积交(相交部分) 29](#_Toc526875254)

[矩形周长并 30](#_Toc526875255)

[树状数组 32](#_Toc526875256)

[一维树状数组该段求段 32](#_Toc526875257)

[二维改点求段 32](#_Toc526875258)

[二维改段求点 33](#_Toc526875259)

[二维改段求段 33](#_Toc526875260)

[RMQ区间最值 34](#_Toc526875261)

[主席树 35](#_Toc526875262)

[第k大 35](#_Toc526875263)

[区间数字种类 36](#_Toc526875264)

[划分树 37](#_Toc526875265)

[分块区间众数 38](#_Toc526875266)

[分块动态第k大 39](#_Toc526875267)

[莫队 41](#_Toc526875268)

[修改莫队-颜色种类 41](#_Toc526875269)

[数论 43](#_Toc526875270)

[欧拉函数 43](#_Toc526875271)

[质因数分解 43](#_Toc526875272)

[取模运算 44](#_Toc526875273)

[杂烩 45](#_Toc526875274)

[Miller-rabin 46](#_Toc526875275)

[O(1)快速乘 47](#_Toc526875276)

[求平方根 47](#_Toc526875277)

[矩阵快速幂 47](#_Toc526875278)

[Lucas 48](#_Toc526875279)

[exLucas 49](#_Toc526875280)

[除法分块 50](#_Toc526875281)

[容斥 50](#_Toc526875282)

[一般的积性函数 51](#_Toc526875283)

[卡特兰数 51](#_Toc526875284)

[计算几何 52](#_Toc526875285)

[模板 52](#_Toc526875286)

[凸包 56](#_Toc526875287)

[字符串 57](#_Toc526875288)

[Kmp 57](#_Toc526875289)

[Manacher 57](#_Toc526875290)

[后缀数组 58](#_Toc526875291)

[Ac自动机 59](#_Toc526875292)

[Hash 60](#_Toc526875293)

[DP 62](#_Toc526875294)

[数位DP 62](#_Toc526875295)

[部分头文件 63](#_Toc526875296)

[读入挂 63](#_Toc526875297)

[JAVA 65](#_Toc526875298)

# 目录

目录下大的标题1，中3，小的标题7

代码大类标题5，6，7

# 代码

[图论](#_目录)

[最短路](#_最短路_1)

[Floyd](#_Floyd_3)

int n,m,mp[N][N];

vector<int> v[N][2];

void add(int x, int y){

v[x][1].push\_back(y),v[y][0].push\_back(x);

}

int main(){

scanf("%d%d", &n, &m);

for(int i=1,x,y;i<=m;i++)

scanf("%d%d",&x,&y),add(x,y),mp[x][y]=1;

for(int k = 1; k <= n; k++)

for(int i = 0; i<v[k][0].size(); i++)

for(int j = 0;j<v[k][1].size(); j++)

if(mp[v[k][0][i]][v[k][1][j]] == 0){

mp[v[k][0][i]][v[k][1][j]] = 1;//记得更新每个点的出边入边

v[v[k][0][i]][1].push\_back(v[k][1][j]);

v[v[k][1][j]][0].push\_back(v[k][0][i]);

}

return 0;

}

[Dijkstra求最短路次短路及条数](#_图论_1)

struct lp{

int to, id;

LL w;

lp(int a,LL b){to=a;w=b;id=0;}

lp(int a,LL b,int c){to=a;w=b;id=c;}

bool operator <(const lp &a)const {

if(w!=a.w) return w>a.w;

return to>a.to;

}

};

vector<lp>mp[N];

//mp[a].push\_back(lp(b,c));

LL dis1[N],dis2[N];

int n, m, cnt1[N], cnt2[N], vis[N][3];

int st,ed;

void dij(){

priority\_queue<lp>Q;

Q.push(lp(st,0,1));

memset(..,0,sizeof(..));

cnt1[st]=1;

dis1[st]=0;

while(!Q.empty()){

lp aa = Q.top();Q.pop();

int u = aa.to, len = mp[u].size();

if(vis[u][aa.id])continue;

vis[u][aa.id] = 1;

for(int i=0;i<len;++i){

int v = mp[u][i].to;

LL w = mp[u][i].w + aa.w;

if(w<dis1[v]){

dis2[v] = dis1[v];

cnt2[v] = cnt1[v];

dis1[v] = w;

if(aa.id==1)cnt1[v] = cnt1[u];

else cnt1[v] = cnt2[u];

Q.push(lp(v, dis1[v], 1));

Q.push(lp(v, dis2[v], 2));

}else if(w == dis1[v]){

if(aa.id==1)cnt1[v] += cnt1[u];

else cnt1[v] += cnt2[u];

}else if(w < dis2[v]){

dis2[v] = w;

if(aa.id==1)cnt2[v] = cnt1[u];

else cnt2[v] = cnt2[u];

Q.push(lp(v, dis2[v], 2));

}else if(w==dis2[v]){

if(aa.id==1)cnt2[v] += cnt1[u];

else cnt2[v] += cnt2[u];

}

}

}

printf("%d\n", cnt1[ed]);

}

struct lp{

int to, id;

LL w;

lp(int a,LL b){to=a;w=b;id=0;}

lp(int a,LL b,int c){to=a;w=b;id=c;}

bool operator <(const lp &a)const {

if(w!=a.w) return w>a.w;

return to>a.to;

}

};

vector<lp>mp[N];

//mp[a].push\_back(lp(b,c));

LL dis1[N],dis2[N];

int n, m, cnt1[N], cnt2[N], vis[N][3];

int st,ed;

void dij(){

[A\*求k短路](#_图论_1)

const int N = (int)1e3 +107;

int n, m, k, st, ed;

int dis[N],vis[N],time[N];

struct lp{

int f,g,v;

friend bool operator <(const lp &a,const lp &b){

if(a.f==b.f)return a.g>b.g;

return a.f>b.f;

}

}aa,bb;

struct lh{

int v,w,nex;

}cw[100000+5],rev[100000+5];

int head[N],tot,headd[N],tum;

int q[2500005];

void add(int u,int v,int w){

cw[++tot].v=v;cw[tot].nex=head[u];cw[tot].w=w;

head[u]=tot;

rev[++tum].v=u;rev[tum].nex=headd[v];rev[tum].w=w;

headd[v]=tum;

}

void init(){

tot=tum=-1;

memset(head,-1,sizeof(head));

memset(headd,-1,sizeof(headd));

}

void spfa(){

for(int i = 1; i <= n; i++) dis[i] = INF;

memset(vis, 0, sizeof(vis));

int h = 0, t = 1;

q[0] = ed;

dis[ed] = 0;

while(h < t){

int u = q[h++];

vis[u] = 0;

for(int i = headd[u] ; ~i ; i = rev[i].nex){

int v = rev[i].v, w = rev[i].w;

if(dis[v] > dis[u] + w){

dis[v] = dis[u] + w;

if(!vis[v]){

q[t++] = v;

vis[v] = 1;

}

}

}

}

}

int Astar(){

if(st == ed)k++;

if(dis[st]==INF)return -1;

memset(time,0,sizeof(time));

aa.v=st;aa.f=dis[st];aa.g=0;

priority\_queue<lp>Q;

Q.push(aa);

while(!Q.empty()){

bb = Q.top();Q.pop();

int u = bb.v;

time[u]++;

if(time[u]==k&&u==ed)return bb.g;

if(time[u]>k)continue;

for(int i=head[u];~i;i=cw[i].nex){

int v = cw[i].v;

aa.v = v;

aa.g=bb.g+cw[i].w;

aa.f=aa.g+dis[v];

Q.push(aa);

}

}

return -1;

}

拓扑排序输出

1.若需要字典序最小输出的话，使用小堆维护，正常建边即可。

2.若要求最小值在前的输出(312优先级高于231)，可反向建边，大堆维护，把答案reverse下即可

生成树

(严格)次小生成树

性质1: 最小瓶颈路的值一定在MST上

const int MXN = 1e5 + 5;

const int MXE = 6e5 + 5;

const int INF = 0x3f3f3f3f;

int n, m;

LL dis[MXN], dis1[MXN][23], dis2[MXN][23];

int dep[MXN], up[MXN][23];

int FA[MXN], rk[MXN];

int head[MXN], tot;

struct lp{

int v, w, nex;

}cw[MXE];

struct EDGE{

int u, v, w, is;

}edge[MXE];

bool cmp(const EDGE& a, const EDGE& b) {

return a.w < b.w;

}

void add(int u,int v,int w) {

cw[++tot].v = v; cw[tot].nex = head[u]; cw[tot].w = w;

head[u] = tot;

cw[++tot].v = u; cw[tot].nex = head[v]; cw[tot].w = w;

head[v] = tot;

}

void dfs(int u, int ba, int d) {

dep[u] = d; up[u][0] = ba;

for(int i = 1; i < 20; ++i) {

int cf = up[u][i-1];

up[u][i] = up[cf][i-1];

dis1[u][i] = max(dis1[u][i-1], dis1[cf][i-1]);

if(dis1[u][i-1] == dis1[cf][i-1]) dis2[u][i] = max(dis2[u][i-1], dis2[cf][i-1]);

else {

dis2[u][i] = min(dis1[u][i-1], dis1[cf][i-1]);

dis2[u][i] = max(dis2[u][i], max(dis2[u][i-1], dis2[cf][i-1]));

}

}

for(int i = head[u]; ~i; i = cw[i].nex) {

int v = cw[i].v;

if(v == ba) continue;

dis[v] = dis[u] + cw[i].w;

dis1[v][0] = cw[i].w;

up[v][0] = u;

dfs(v, u, d + 1);

}

}

int LCA(int x, int y) {

if(dep[x] < dep[y]) swap(x, y);

int k = dep[x] - dep[y];

for(int i = 0; i < 20; ++i) {

if((1<<i)&k) {

x = up[x][i];

}

}

if(x == y) return x;

for(int i = 19; i >= 0; --i) {

if(up[x][i] != up[y][i]) {

x = up[x][i]; y = up[y][i];

}

}

return up[x][0];

}

int Fi(int x) {

return FA[x] == x? x: FA[x] = Fi(FA[x]);

}

LL calc(int u, int v,int w) {

if(dep[u] < dep[v]) swap(u, v);

int k = dep[u] - dep[v];

LL mx1 = 0, mx2 = 0;

for(int i = 0; i < 20; ++i) {

if((1<<i)&k) {

if(dis1[u][i] > mx1) {

mx2 = mx1;

mx1 = dis1[u][i];

}

mx2 = max(mx2, dis2[u][i]);

u = up[u][i];

}

}

if(mx1 != w) return w - mx1;

return w - mx2;

}

void solve() {

scanf("%d%d", &n, &m);

init();

for(int i = 0; i < m; ++i) {

scanf("%d%d%d", &edge[i].u, &edge[i].v, &edge[i].w);

edge[i].is = 0;

}

for(int i = 1; i <= n; ++i) FA[i] = i, rk[i] = 1;

sort(edge, edge + m, cmp);

int cnt = 0;

LL MAX = 0;

for(int i = 0; i < m; ++i) {

int pa = Fi(edge[i].u), pb = Fi(edge[i].v);

if(pa == pb) continue;

if(rk[pa] > rk[pb]) swap(pa,pb);

FA[pa] = pb;

rk[pb] += rk[pa];

add(edge[i].u, edge[i].v,edge[i].w);

MAX += edge[i].w;

cnt ++;

edge[i].is = 1;

if(cnt == n-1) break;

}

up[1][0] = 1;

dis1[1][0] = 0;

dfs(1, 1, 1);

LL tmp = 1e18;

for(int i = 0; i < m; ++i) {

if(edge[i].is) continue;

int cf = LCA(edge[i].u,edge[i].v);

tmp = min(tmp, calc(edge[i].u,cf,edge[i].w));

tmp = min(tmp, calc(edge[i].v,cf,edge[i].w));

}

printf("%lld\n", MAX + tmp);

}

链上第k小点权

const int MXN = 1e5 + 5;

const int MXE = 6e5 + 5;

const int INF = 0x3f3f3f3f;

int n, m;

std::vector<int> mp[MXN];

struct lp{

int l, r, sum;

}cw[MXN\*40];

int rot[MXN], k;

int ar[MXN], br[MXN];

int sz[MXN], son[MXN], top[MXN], dep[MXN], fa[MXN], CNT;

void init() {

CNT = 0;

cw[0].l = cw[0].r = cw[0].sum = 0;

for(int i = 1; i <= n; ++i) mp[i].clear();

memset(rot, 0 ,sizeof(rot));

memset(son, 0, sizeof(son));

memset(top, 0, sizeof(top));

}

void insert(int &cur,int last,int p,int l,int r) {

cw[++CNT] = cw[last];

cur = CNT;

cw[cur].sum ++;

if(l == r) return;

int mid = (l + r) >> 1;

if(p <= mid) insert(cw[cur].l, cw[last].l, p, l, mid);

else insert(cw[cur].r, cw[last].r, p, mid+1, r);

}

int query(int pl,int pr,int lca,int lca\_fa,int p,int l,int r) {

if(l == r) return l;

int mid = (l + r) >> 1;

int tmp = cw[lson(pl)].sum+cw[lson(pr)].sum-cw[lson(lca)].sum-cw[lson(lca\_fa)].sum;

if(p <= tmp) return query(lson(pl),lson(pr),lson(lca),lson(lca\_fa),p,l,mid);

else {

return query(rson(pl),rson(pr),rson(lca),rson(lca\_fa),p-tmp,mid+1,r);

}

}

void dfs(int u,int ba,int d) {

sz[u] = 1, dep[u] = d;

insert(rot[u], rot[ba], ar[u], 1, k);

int len = mp[u].size();

for(int i = 0; i < len; ++i) {

int v = mp[u][i];

if(v == ba) continue;

fa[v] = u;

dfs(v, u, d+1);

sz[u] += sz[v];

if(!son[u] || sz[v] > sz[son[u]]) son[u] = v;

}

}

void dfs(int u,int tu) {

top[u] = tu;

if(!son[u]) return;

dfs(son[u], tu);

int len = mp[u].size();

for(int i = 0; i < len; ++i) {

int v = mp[u][i];

if(v == son[u] || v == fa[u]) continue;

dfs(v, v);

}

}

int LCA(int x,int y) {

while(top[x] != top[y]) {

if(dep[top[x]]>=dep[top[y]]) x = fa[top[x]];

else y = fa[top[y]];

}

if(dep[x] >= dep[y]) return y;

return x;

}

void solve() {

for(int i = 1; i <= n; br[i] = ar[i], ++i) scanf("%d", &ar[i]);

sort(br+1, br+1+n);

k = unique(br+1,br+1+n)-br;

for(int i = 1; i <= n; ++i) {

ar[i] = lower\_bound(br+1,br+k,ar[i]) - br;

}

init();

for(int i = 1, u , v; i < n; ++ i) {

scanf("%d%d", &u, &v);

mp[u].push\_back(v);

mp[v].push\_back(u);

}

dfs(1, 0, 1);

dfs(1, 1);

int lastans = 0;

while(m --) {

int u, v, p;

scanf("%d%d%d", &u, &v, &p);

u = u ^ lastans;

int lca = LCA(u, v);

lastans = br[query(rot[u],rot[v],rot[lca],rot[fa[lca]],p,1,k)];

printf("%d\n", lastans);

}

}

最小生成树的最佳替换边

double prim() {

double sum = 0;

for(int i = 1; i <= n; ++i) {

dis[i] = ar[1][i];

vis[i] = 1;

pre[i] = 1;

}

vis[1] = -1;

for(int J = 1; J < n; ++J) {

int p = -1;

double mm = INF;

for(int i = 1; i <= n; ++i)

if(vis[i] != -1 && dis[i] < mm) {

p = i; mm = dis[i];

}

if(p == -1) break;

sum += mm;

son[p].push\_back(vis[p]);

son[vis[p]].push\_back(p);

is[p][vis[p]] = is[vis[p]][p] = 1;

vis[p] = -1;

for(int i = 1; i <= n; ++i)

if(ar[p][i] < dis[i] && vis[i] != -1) {

dis[i] = ar[p][i];

vis[i] = p;

}

}

return sum;

}

double dfs(int u,int ba,int r) {

int len = son[u].size();

double mi = INF;

for(int i = 0; i < len; ++i) {

int v = son[u][i];

if(v == ba) continue;

double tmp = dfs(v, u, r);

mi = min(mi, tmp);

dis1[u][v] = dis1[v][u] = min(dis1[v][u], tmp);

}

if(r != ba) mi = min(mi, ar[r][u]);

return mi;

}

void solve() {

scanf("%d%lf", &n, &m);

init();

for(int i = 0; i < m; ++i) ar[u][v] = ar[v][u] = w;

double tmp = prim(), ans = tmp;

for(int i = 1; i <= n; ++i) dfs(i, -1, i);

for(int i = 2; i < n; ++i)

for(int j = i + 1; j <= n; ++j)

if(is[i][j])ans = max(tmp - ar[i][j] + dis1[i][j], ans);

printf("%.2f\n", ans);

}

[二分图](#_二分图)

**二分图的判定：**

 只需判定图是否有奇环即可，若无奇环则为二分图。判定方法有染色法，并查集(拆点)等。

**边覆盖集：**

 选出某些边集合，满足G中所有点都是边集合中某条边的端点。注意极小的最小的区别。极小边覆盖集的任何真子集都不是边覆盖集。

**最小路径覆盖和最小边覆盖的区别：**

 最小路径覆盖不要求给的图是二分图，而要求是PXP的有向图。不能有环，根据原图拆点构成二分图。

 点u分为u1和u2，点v分为v1和v2。若u->v有边，则u1->v2。

**最小路径覆盖 = n - m**

 n是原图的顶点数，m是新构造二分图的最大匹配。

 根据定义最小路径覆盖里要求同一个点只可以属于一条路径，

 即路径时不可以开叉的，如果在二分图里选两条有公共点的边那么反应在原图上

 就是路径有岔路，那么就不符合匹配的定义了,所以二分图里选的边必须是无公

 共交点的，这转化到最大匹配了。

**DAG图中：最小路径覆盖 = 节点数 - 最大匹配数**

 Floyd对原题求一次传递闭包，然后求新图的最小路径覆盖。

**最大匹配数 = 最小点覆盖数 = 左边匹配点 + 右边未匹配点**

**最小路径覆盖 = |G| - 最大匹配数**

**最大独立集 = 顶点数 - 最大匹配 = 顶点数 - 最小点覆盖**

显然把最大匹配两端的点都从顶点集中去掉, 剩余的点是独立集, 这是|V|-2\*|M|, 同时必然可以从每条匹配边的两端取一个点加入独立集并保持独立集性质.

**最大团 = 补图的最大独立集**

**最小割 = 最小点权覆盖集 = 点权和 - 最大点权独立集**

**二分图最小边覆盖 = 最大独立数 = n - 最大匹配**

**无向图的最小边覆盖集 = (二分图(拆点后)两边顶点数-最大匹配)/2**

[匈牙利](#_匈牙利)

bool dfs(int u){

for(auto x : son[u]){

if(vis[x]) continue;

vis[x] = 1;

if(is[x] == -1 || dfs(is[x])){

is[x] = u;

be[u] = x;

return true;

}

}

return false;

}

void solve() {

mme(is, -1);

mme(be, -1);

int cnt = 0;

for(int i = 1; i <= n; ++i){

if(be[i] != -1) continue;

mme(vis, 0);

if(dfs(i)) cnt++;

}

printf("%d\n", cnt);

}

染色法

const int N = 500;

int flag,vis[N],is[N][N];

vector<int>vc[N];

bool bfs(int s){

queue<int>Q;

Q.push(s);

vis[s]=0;

while(!Q.empty()){

int u=Q.front();Q.pop();

for(int i=0;i<vc[u].size();++i){

int v=vc[u][i];

if(vis[v]==-1){

vis[v]=!vis[u];

Q.push(v);

}

if(vis[v]==vis[u])return false;

}

}

return true;

}

void solve(){

for(int i=0;i<m;++i){

vc[u].push\_back(v);

vc[v].push\_back(u);

}

flag=0;

mm1(vis);

for(int i=1;i<=n;++i){

if(vis[i]==-1&&(!bfs(i))){

flag=1;

break;

}

}

if(flag)printf("No\n");

}

[KM](#_KM)

const int N=310;

const int INF = 0x3f3f3f3f;

int match[N];

int lx[N],ly[N];

int sx[N],sy[N];

int weight[N][N];

int n;

int dfs(int x) {

sx[x]=true;

for(int i=0; i<n; i++) {

if(!sy[i]&&lx[x]+ly[i]==weight[x][i]) {

sy[i]=true;

if(match[i]==-1||dfs(match[i])) {

match[i]=x;

return true;

}

}

}

return false;

}

int fax(int x) {

if(!x) {

for(int i=0; i<n; i++) {

for(int j=0; j<n; j++) {

weight[i][j]=-weight[i][j];

}

}

}

memset(match,-1,sizeof(match));

for(int i=0; i<n; i++) {

ly[i]=0;

lx[i]=-INF;

for(int j=0; j<n; j++) {

if(weight[i][j]>lx[i]) {

lx[i]=weight[i][j];

}

}

}

for(int i=0; i<n; i++) {

while(1) {

memset(sx,0,sizeof(sx));

memset(sy,0,sizeof(sy));

if(dfs(i))break;

/\*将所有增广轨中的X方点的标号全部减去一个常数d，Y方点的标号全部加上一个常数d\*/

int mic=INF;

for(int j=0; j<n; j++) {

if(sx[j]) {

for(int k=0; k<n; k++) {

if(!sy[k]&&lx[j]+ly[k]-weight[j][k]<mic) {

mic=lx[j]+ly[k]-weight[j][k];

}

}

}

}

if(mic==0) return -1;

for(int j=0; j<n; j++) {

if(sx[j]) lx[j]-=mic;

if(sy[j]) ly[j]+=mic;

}

}

}

int sum=0;

for(int i=0; i<n; i++) {

if(match[i]>=0) {

sum+=weight[match[i]][i];

}

}

if(!x) sum=-sum;

return sum;

}

void solve(){

for(int i=0;i<n;++i){

for(int j=0;j<n;++j){

scanf("%d",&weight[i][j]);

}

}

printf("%d\n",fax(1));

}

[Tarjan](#_Tarjan)

[模板](#_模版)

const int N = 100005;

struct lp{int v,w,nex;}cw[N\*2];

struct edge{int u,v,w;}now,aa;

int stak[N],top;

int head[N],vis[N],dfn[N],low[N],qltMap[N];

int n,m,tot,qltNum,vs,vt,inde;

int add\_block[N];///删除一个点后增加的连通块

void dfs(int u,int fa,int fae){

dfn[u]=low[u]=++inde;

vis[u]=1;int v,son=0;

stak[++top]=u;

for(int i=head[u];~i;i=cw[i].nex){

int v=cw[i].v;

if(fae!=-2&&(fae==(i^1))){//或者v==fa

continue;

}

if(!vis[v]){

son++;

dfs(v,u,i);

low[u]=min(low[u],low[v]);

if(u!=root&&low[v]>=dfn[u]){

iscut[u]=1;add\_block[u]++;

}else if(u==root&&son==2){

iscut[u]=1;

}

if(low[v]>dfn[u]){

printf("%d -> %d\n", u, v);

}

}else if(vis[v]==1){

low[u]=min(low[u],dfn[v]);

}

}

if(u==root)add\_block[u]=son-1;

if(low[u]==dfn[u]){

qltNum++;

do{

v=stak[top--];

qltMap[v]=qltNum;

}while(v!=u);

}

}

void tarjan(){

for(int i=1;i<=n;++i){

if(!vis[i])dfs(i,-1,-2);

}

}

void add(int a,int b,int c){

cw[++tot].v=b;cw[tot].nex=head[a];cw[tot].w=c;

head[a]=tot;

}

[缩点](#_缩点)

void dfs(int u,int Fa){//这是另一个普通缩点方法，上面也有缩点

dfn[u] = low[u] = ++inde;

vis[u]=1;int v;

st[++top]=u;

for(int i=head[u];~i;i=cw[i].pre){

if(cw[i].is)continue;

v=cw[i].to;

if(v==Fa)continue;

if(!vis[v]){

cw[i].is=cw[i^1].is=1;

dfs(v,u);

low[u]=min(low[u],low[v]);

if(low[v]>dfn[u]){//只有单独一个点的情况这个for循环不会跑

qltNum++;

do{

qltMap[st[top]]=qltNum;

}while(st[top--]!=v);

}

}else if(vis[v]==1){

low[u]=min(low[u],dfn[v]);

}

}

}

点双联通分量

int dfs(int u, int fa){

int lowu = pre[u] = ++dfs\_clock;

int child = 0;

for(int i = head[u]; i != -1; i = edges[i].next) {

int v = edges[i].v;

Edge e = (Edge) {u, v};

if(!pre[v]) {

s.push(e);

child++;

int lowv = dfs(v, u);

lowu = min(lowu, lowv); //用后代更新lowu

if(lowv >= pre[u]) { //找到了一个子树满足割顶的条件

iscut[u] = 1;

bcc\_cnt++;

bcc[bcc\_cnt].clear();

for(;;) { //保存bcc信息

Edge x = s.top(); s.pop();

if(bccno[x.u] != bcc\_cnt) {bcc[bcc\_cnt].push\_back(x.u); bccno[x.u] = bcc\_cnt;}

if(bccno[x.v] != bcc\_cnt) {bcc[bcc\_cnt].push\_back(x.v); bccno[x.v] = bcc\_cnt;}

if(x.u == u && x.v == v) break;

}

}

} else if(pre[v] < pre[u] && v != fa) { //用反向边更新lowu

s.push(e);

lowu = min(lowu, pre[v]);

}

}

if(fa < 0 && child == 1) iscut[u] = 0; //对于根节点若只有一个子树则不是割顶

return lowu;

}

边双联通分量

//找出所有的桥

int dfs\_findbridge(int u, int fa){

int lowu = pre[u] = ++dfs\_clock;

for(int i = head[u]; i != -1; i = edges[i].next) {

int v = edges[i].v;

if(!pre[v]) {

int lowv = dfs\_findbridge(v, u);

lowu = min(lowu, lowv);

if(lowv > pre[u]) {

isbridge[edges[i].no] = 1; //桥

}

} else if(pre[v] < pre[u] && v != fa) {

lowu = min(lowu, pre[v]);

}

}

return lowu;

}

//保存边\_双连通分量的信息

void dfs\_coutbridge(int u, int fa){

ebc[ebcnum].push\_back(u);

pre[u] = ++dfs\_clock;

for(int i = head[u]; i != -1; i = edges[i].next) {

int v = edges[i].v;

if(!isbridge[edges[i].no] && !pre[v]) dfs\_coutbridge(v, u);

}

}

void solve(){

dfs\_findbridge(1, -1);

memset(pre, 0, sizeof(pre));

for(int i = 1; i <= n; i++) {

if(!pre[i]) {

ebc[ebcnum].clear();

dfs\_coutbridge(i, -1);

ebcnum++;

}

}

}

[LCA](#_LCA)

const int N = 1e5+7;

const LL INF = 0x3f3f3f3f3f3f3f3f;

struct lp{

int v,nex;

LL w;

}cw[N\*10];

int fa[N],head[N],dep[N],f[N][30];

LL size[N],dis[N],Z;

int n,m,tot,X,Y;

void add(int x,int y,int z){

cw[++tot].v=y;cw[tot].w=z\*1LL;cw[tot].nex=head[x];

head[x]=tot;

cw[++tot].v=x;cw[tot].w=z\*1LL;cw[tot].nex=head[y];

head[y]=tot;

}

void init(){

tot=-1;mm1(head,-1);mm0(fa);mm0(f);

mm0(dep);mm0(size);mm0(dis);

}

void dfs(int u,int Fa,int h){

dep[u]=h;f[u][0]=Fa;fa[u]=Fa;

for(int i=1;i<20;++i){

int cf=f[u][i-1];

f[u][i]=f[cf][i-1];

}

for(int i=head[u];~i;i=cw[i].nex){

int v=cw[i].v;

if(v==Fa)continue;

dis[v]=dis[u]+cw[i].w;

dfs(v,u,h+1);

}

}

int LCA(int x,int y){

if(dep[x]<dep[y])x^=y^=x^=y;

for(int i=19;i>=0;--i){

if(dep[f[x][i]]>=dep[y]){

x=f[x][i];

}

}

if(x==y)return x;

for(int i=19;i>=0;--i){

if(f[x][i]!=f[y][i]){

x=f[x][i];y=f[y][i];

}

}

return f[x][0];

}

LL Len(int a,int b){

int ff=LCA(a,b);

return (dis[a]+dis[b]-2\*dis[ff]);

}

void solve(){

scanf("%d%d", &n, &m);

init();

for(int i=0;i<n-1;++i)

add(u,v,w);

dfs(1,0,0);

int a,b;

scanf("%d%d",&a,&b);

LL ans=min(INF,Len(a,b));

ans=min(ans,min(Len(a,X)+Len(b,Y)+Z,Len(a,Y)+Len(b,X)+Z));

printf("%lld\n",ans );

}

2-sat

https://www.cnblogs.com/-ZZB-/p/6635483.html

简介

什么是2-SAT呢？就是有一些集合，每个集合中有且仅有两个元素，且不能同时选取两个元素

集合间的元素存在一定的选择关系，求解可行性及可行方案。

算法

1、连边

2、跑tarjan

3、判可行性，即同一集合中的两个点是否同属一个强连通块

4、缩点建新图，连反边

5、拓扑序，若当前点没有被访问过，则选择该点，不选择其另外的点

连边

2-SAT关键是连边，需要充分理解好边的概念：

a->b即选a必选b。

a、b不能同时选：选了a就要选b'，选了b就要选a'

a、b必须同时选：选了a就要选b，选了b就要选a，选了a'就要选b'，选了b'就要选a'

a、b必须选一个：选了a就要选b'，选了b就要选a'，选了a'就要选b，选了b'就要选a

※a必须选：a'->a

树链剖分

模板

#include<bits/stdc++.h>

#define lowbit(x) (x&(-(x)))

#define lson rt<<1

#define rson rt<<1|1

#define mme(a,b) memset((a),(b),sizeof((a)))

#define fuck(x) cout<<"\* "<<x<<"\n"

#define iis std::ios::sync\_with\_stdio(false)

#define fi first

#define se second

using namespace std;

typedef long long LL;

const int INF = 0x3f3f3f3f;

const int MOD = 998244353;

const int N = 1e5 + 7;

const int MX = 2e5 + 7;

int n, m, q, mod, root;

struct Tree{

int l,r,d,Sum,lazy;

}sgt[N<<2];

int ar[N];

struct lp{

int v,nex;

}cw[MX];

int head[MX],tot;

int inde;

int sz[MX],top[MX],son[MX],dep[MX],faz[MX];

int tid[MX],rnk[MX],rtid[MX];//id->dfsid//dfsid->id//lastid

void init(){

tot=-1;inde=0;

mme(head,-1);mme(son,-1);

mme(sz,0);mme(dep,0);mme(faz,0);mme(top,0);

mme(tid,0);mme(rtid,0);mme(rnk,0);

}

void push\_up(int rt){

sgt[rt].Sum=(sgt[lson].Sum+sgt[rson].Sum)%mod;

}

void push\_down(int rt){

if(sgt[rt].lazy){

int c = sgt[rt].lazy;

sgt[lson].lazy=(sgt[lson].lazy+sgt[rt].lazy)%mod;

sgt[rson].lazy=(sgt[rson].lazy+sgt[rt].lazy)%mod;

sgt[lson].Sum=(sgt[lson].Sum+sgt[lson].d\*c%mod)%mod;

sgt[rson].Sum=(sgt[rson].Sum+sgt[rson].d\*c%mod)%mod;

sgt[rt].lazy=0;

}

}

void build(int l,int r,int rt){

sgt[rt].l=l;sgt[rt].r=r;

sgt[rt].d=r-l+1;

sgt[rt].lazy=0;

if(l==r){

sgt[rt].Sum=ar[rnk[l]];

return;

}

int mid=(l+r)>>1;

build(l,mid,lson);build(mid+1,r,rson);

push\_up(rt);

}

void update(int L,int R,int c,int rt){

int l = sgt[rt].l,r=sgt[rt].r,mid=(l+r)>>1;

if(sgt[rt].l>R||sgt[rt].r<L)return;

if(L<=l&&r<=R){

sgt[rt].lazy=(c+sgt[rt].lazy)%mod;

sgt[rt].Sum=(sgt[rt].Sum+sgt[rt].d\*c%mod)%mod;

return;

}

if(sgt[rt].l==sgt[rt].r)return;

push\_down(rt);

if(L<=mid)update(L,R,c,lson);

if(R>mid)update(L,R,c,rson);

push\_up(rt);

}

int query(int L,int R,int rt){

int l = sgt[rt].l,r=sgt[rt].r,mid=(l+r)>>1;

if(L<=l&&r<=R){

return sgt[rt].Sum;

}

if(sgt[rt].l>R||sgt[rt].r<L)return 0;

if(sgt[rt].l==sgt[rt].r)return 0;

push\_down(rt);

int \_sum = 0;

if(L<=mid)\_sum = query(L,R,lson)%mod;

if(R>mid)\_sum=(\_sum+query(L,R,rson))%mod;

return \_sum;

}

void dfs1(int u,int fat,int depth){

dep[u] = depth;faz[u] = fat;sz[u] = 1;

for(int i=head[u];~i;i=cw[i].nex){

int v = cw[i].v;

if(v==fat)continue;

dfs1(v,u,depth+1);

sz[u] += sz[v];

if(son[u]==-1||sz[v]>sz[son[u]]){

son[u] = v;

}

}

}

void dfs2(int u,int t){

top[u] = t;

tid[u] = ++inde;

rnk[inde]=u;

if(son[u]==-1){

rtid[u]=inde;

return;

}

dfs2(son[u],t);

for(int i=head[u];~i;i=cw[i].nex){

int v = cw[i].v;

if(v!=son[u]&&v!=faz[u]){

dfs2(v,v);

}

}

rtid[u]=inde;

}

int path\_query(int x,int y){

int ans=0;

int fx=top[x],fy=top[y];

while(fx!=fy){

if(dep[fx]>=dep[fy]){

ans=(ans+query(tid[fx],tid[x],1))%mod;

x=faz[fx];

}else {

ans=(ans+query(tid[fy],tid[y],1))%mod;

y=faz[fy];

}

fx=top[x],fy=top[y];

}

if(tid[x]<tid[y]){

ans=(ans+query(tid[x],tid[y],1))%mod;

}else {

ans=(ans+query(tid[y],tid[x],1))%mod;

}

return ans;

}

void path\_update(int x,int y,int c){

int fx=top[x],fy=top[y];

while(fx!=fy){

if(dep[fx]>dep[fy]){

update(tid[fx],tid[x],c,1);

x=faz[fx];

}else {

update(tid[fy],tid[y],c,1);

y=faz[fy];

}

fx=top[x],fy=top[y];

}

if(tid[x]<tid[y]){

update(tid[x],tid[y],c,1);

}else {

update(tid[y],tid[x],c,1);

}

}

void add\_edge(int u,int v){

cw[++tot].v=v;cw[tot].nex=head[u];

head[u]=tot;

cw[++tot].v=u;cw[tot].nex=head[v];

head[v]=tot;

}

void pre\_build(){

dfs1(root,root,1);

dfs2(root,root);

build(1,n,1);

}

int main(){

while(~scanf("%d%d%d%d",&n,&q,&root,&mod)){

init();

for(int i=1;i<=n;++i){

scanf("%d",&ar[i]);

ar[i]%=mod;

}

for(int i=1,u,v;i<n;++i){

scanf("%d%d",&u,&v);

add\_edge(u,v);

}

pre\_build();

while(q--){

int op;scanf("%d",&op);

int u,v,w;

if(op==1){

scanf("%d%d%d",&u,&v,&w);

w%=mod;

path\_update(u,v,w);

}else if(op==2){

scanf("%d%d",&u,&v);

printf("%d\n", path\_query(u,v));

}else if(op==3){

scanf("%d%d",&u,&w);

w%=mod;

update(tid[u],rtid[u],w,1);

}else if(op==4){

scanf("%d",&u);

printf("%d\n", query(tid[u],rtid[u],1));

}

}

}

return 0;

}

[网络流](#_网络流)

[最大流Dinic](#_网络流)

const int INF = 0x3f3f3f3f;

const int MXN = 1e4+7;

const int MXE = 1e7+7;

struct DINIC{

int tot,vt,vs;

int d[MXN],head[MXN];

struct lp{

int v,w,nex;

}cw[MXE];

void add\_edge(int a,int b,int c){

cw[++tot].v=b;cw[tot].nex=head[a],cw[tot].w=c;

head[a]=tot;

cw[++tot].v=a;cw[tot].nex=head[b],cw[tot].w=0;

head[b]=tot;

}

bool bfs(){

memset(d,-1,sizeof(d));

queue<int>Q;

Q.push(vt);d[vt]=0;

while(!Q.empty()){

int u=Q.front();

Q.pop();

for(int i=head[u];i!=-1;i=cw[i].nex){

int v=cw[i].v;

if(cw[i^1].w&&d[v]==-1){

d[v]=d[u]+1;

Q.push(v);

}

}

}

return d[vs]!=-1;

}

/\*int dfs(int x,int low){

if(x==vt||low==0)return low;

int flow=0,used=0;

for(int i=head[x];i!=-1;i=cw[i].nex){

int v=cw[i].to;

if(cw[i].w&&d[v]+1==d[x]&&(used=dfs(v,min(low,cw[i].w)))>0){

//used=dfs(v,min(low,cw[i].w));

if(!used)continue;

flow+=used,low-=used;

cw[i].w-=used;cw[i^1].w+=used;

if(!low)break;

}

}

if(!flow)d[x]=-1;

return flow;

}\*/

int dfs(int x,int f){

if(x==vt||f==0) return f;

int use=0,w;

for(int i=head[x];i!=-1;i=cw[i].nex){

int to=cw[i].v;

if(d[to]==d[x]-1 && cw[i].w){

w=dfs(to,min(cw[i].w,f-use));

cw[i].w-=w,cw[i^1].w+=w;

use+=w;

if(use==f) return f;

}

}

//if(!use)d[x]=-1;

return use;

}

void init(int st,int ed){

tot=-1;

memset(head,-1,sizeof(head));

vs=st;vt=ed;

}

int max\_flow(){

int ans=0;

while(bfs())ans+=dfs(vs,INF);

return ans;

}

}dinic;

dinic.init(vs,vt);

最大流SAP

const int N = 1e5+5;

const int INF = 0x3f3f3f3f;

typedef long long LL;

struct lp {

int v,nex;

int cap,flow;

} cw[N\*4];

int n,m,vs,vt,NE,NV;

int head[N],pre[N],cur[N],level[N],gap[N];

inline void add\_edge(int u,int v,int cap){

cw[NE].v = v;cw[NE].cap = cap;cw[NE].flow = 0;cw[NE].nex = head[u];

head[u] = NE++;

cur[u] = head[u];

cw[NE].v = u;cw[NE].cap = 0;cw[NE].flow = 0;cw[NE].nex = head[v];

head[v] = NE++;

cur[v] = head[v];

}

inline void bfs(int vt){

memset(level,-1,sizeof(level));

memset(gap,0,sizeof(gap));

level[vt]=0;

gap[level[vt]]++;

queue<int>que;

que.push(vt);

while(!que.empty()) {

int u=que.front();

que.pop();

for(int i=head[u]; i!=-1; i=cw[i].nex) {

int v=cw[i].v;

if(level[v]!=-1)continue;

level[v]=level[u]+1;

gap[level[v]]++;

que.push(v);

}

}

}

int SAP(int vs,int vt){

bfs(vt);

memset(pre,-1,sizeof(pre));

//for(int i=0;i<=n;++i)cur[i]=head[i];

int u=pre[vs]=vs,flow=0,aug=INF;

gap[0]=NV;

while(level[vs]<NV) {

bool flag=false;

for(int &i=cur[u]; i!=-1; i=cw[i].nex) {

int v=cw[i].v;

if(cw[i].cap>cw[i].flow&&level[u]==level[v]+1) {

flag=true;

pre[v]=u;

u=v;

// aug=(aug==-1?cw[i].cap:min(aug,cw[i].cap));

aug=min(aug,cw[i].cap-cw[i].flow );

if(v==vt) {

flow+=aug;

for(u=pre[v]; v!=vs; v=u,u=pre[u]) {

cw[cur[u]].flow+=aug;

cw[cur[u]^1].flow-=aug;

}

// aug=-1;

aug=INF;

}

break;

}

}

if(flag)continue;

int minlevel=NV;

for(int i=head[u]; i!=-1; i=cw[i].nex) {

int v=cw[i].v;

if(cw[i].cap>cw[i].flow&&level[v]<minlevel) {

minlevel=level[v];

cur[u]=i;

}

}

if(--gap[level[u]]==0)break;

level[u]=minlevel+1;

gap[level[u]]++;

u=pre[u];

}

return flow;

}

void init(){

NE=0;NV=n;

vs=0;vt=0;

memset(head,-1,sizeof(head));

memset(cur,-1,sizeof(cur));

}

void solve(){

init();

add\_edge(u,v,w);

printf("%d\n", SAP(vs, vt));

}

最小费用流Dinic

const int N = 405;//最小权匹配

const int M = 45000;

const int INF = 0x3f3f3f3f;

int n,m,tot;

struct lp{

int u,v,w,c,nex,f;

lp(){}

lp(int a,int b,int c,int d,int e,int f):

u(a),v(b),w(c),c(d),nex(e),f(f){}

}cw[M];

int vis[N],dis[N],pre[N],head[N];

int aa[N], mp[500][500];

void add(int x,int y,int z,int f){

cw[++tot]=lp(x,y,z,f,head[x],0);

head[x]=tot;

cw[++tot]=lp(y,x,-z,0,head[y],0);

head[y]=tot;

}

bool spfa(int vs,int vt){

mm0(vis);

mmx(dis);

mm1(pre);

queue<int>Q;

Q.push(vs);

dis[vs]=0;

aa[vs]=INF;

while(!Q.empty()){

int u=Q.front();Q.pop();

vis[u]=0;

for(int i=head[u];i!=-1;i=cw[i].nex){

int v=cw[i].v,w=cw[i].w;

if(cw[i].c>cw[i].f&&dis[v]>dis[u]+w){

dis[v]=dis[u]+w;

pre[v]=i;

aa[v]=min(cw[i].c-cw[i].f ,aa[u]);

if(!vis[v]){

vis[v]=1;

Q.push(v);

}

}

}

}

return dis[vt]!=INF;

}

int MCMF(int vs,int vt){

int maxFlow=INF,maxCost=0,Sum=0;

/\*for(int i=pre[vt];i!=-1;i=pre[cw[i].u]){

maxFlow=min(maxFlow,cw[i].c-cw[i].f);

}\*/

maxFlow=aa[vt];

for(int i=pre[vt];i!=-1;i=pre[cw[i].u]){

cw[i].f+=maxFlow;

cw[i^1].f-=maxFlow;

Sum+=maxFlow\*cw[i].w;

}

maxCost=dis[vt]\*maxFlow;

return maxCost;

}

void solve(){

scanf("%d%d",&n,&m);

int vs=0,vt=2\*n+1;

init();

for(int i=1;i<=n;++i){

add(vs,i,0,1);

add(i+n,vt,0,1);

}

memset(mp,0x3f,sizeof(mp));

for(int i=0;i<m;++i){

scanf("%d%d%d",&u,&v,&w);

mp[u][v+n]=min(mp[u][v+n],w);

}

for(int i=1;i<=n;++i){

for(int j=1+n;j<=n\*2;++j){

if(mp[i][j]!=INF){

add(i,j,mp[i][j],1);

}

}

}

int ans=0;

while(spfa(vs,vt))ans+=MCMF(vs,vt);

printf("%d\n",ans );

}

Zkw

const int MAXN = 605;

const int MAXE = 4e5 + 7;

const int INF = 0x3f3f3f3f;

struct MCMF {

int S, T;//源点，汇点

int tot, n;

int st, en, maxflow, mincost;

bool vis[MAXN];

int head[MAXN], cur[MAXN], dis[MAXN];

int roade[MAXN], roadv[MAXN], rsz; //用于打印路径

queue <int> Q;

struct Edge {

int v, cap, cost, nxt, flow;

Edge() {}

Edge(int a, int b, int c, int d) {

v = a, cap = b, cost = c, nxt = d, flow = 0;

}

} E[MAXE], SE[MAXE];

void init(int \_n) {

n = \_n, tot = 0;

for(int i = 0; i <= n; i++) head[i] = -1;

}

void add\_edge(int u, int v, int cap, int cost) {

E[tot] = Edge(v, cap, cost, head[u]);

head[u] = tot++;

E[tot] = Edge(u, 0, -cost, head[v]);

head[v] = tot++;

}

bool adjust() {

int v, min = INF;

for(int i = 0; i <= n; i++) {

if(!vis[i]) continue;

for(int j = head[i]; ~j; j = E[j].nxt) {

v = E[j].v;

if(E[j].cap - E[j].flow) {

if(!vis[v] && dis[v] - dis[i] + E[j].cost < min) {

min = dis[v] - dis[i] + E[j].cost;

}

}

}

}

if(min == INF) return false;

for(int i = 0; i <= n; i++) {

if(vis[i]) {

cur[i] = head[i];

vis[i] = false;

dis[i] += min;

}

}

return true;

}

int augment(int i, int flow) {

if(i == en) {

mincost += dis[st] \* flow;

maxflow += flow;

return flow;

}

vis[i] = true;

for(int j = cur[i]; j != -1; j = E[j].nxt) {

int v = E[j].v;

if(E[j].cap == E[j].flow) continue;

if(vis[v] || dis[v] + E[j].cost != dis[i]) continue;

int delta = augment(v, std::min(flow, E[j].cap - E[j].flow));

if(delta) {

E[j].flow += delta;

E[j ^ 1].flow -= delta;

cur[i] = j;

return delta;

}

}

return 0;

}

void spfa() {

int u, v;

for(int i = 0; i <= n; i++) {

vis[i] = false;

dis[i] = INF;

}

Q.push(st);

dis[st] = 0;

vis[st] = true;

while(!Q.empty()) {

u = Q.front(), Q.pop();

vis[u] = false;

for(int i = head[u]; ~i; i = E[i].nxt) {

v = E[i].v;

if(E[i].cap == E[i].flow || dis[v] <= dis[u] + E[i].cost) continue;

dis[v] = dis[u] + E[i].cost;

if(!vis[v]) {

vis[v] = true;

Q.push(v);

}

}

}

for(int i = 0; i <= n; i++) {

dis[i] = dis[en] - dis[i];

}

}

int zkw\_flow(int s, int t) {

st = s, en = t;

spfa();

mincost = maxflow = 0;

for(int i = 0; i <= n; i++) {

vis[i] = false;

cur[i] = head[i];

}

do {

while(augment(st, INF)) {

memset(vis, false, n \* sizeof(bool));

}

} while(adjust());

return mincost;

}

}zkw;

//u v 流量 费用

数据结构

[Bitset](#_Bitset)

#include<bitset>

bitset<32> ar; //默认全为0

bitset<32> ar(n); //n的二进制

bitset<32> ar(str); //01串

bitset<n> ar(str,pos,n); //从str第p位开始的n位

ar.size();//返回位数

ar.count();//返回1的个数

ar.any();//返回是否有1

ar.none();//返回是否没有1

ar.test(p);//返回第p位是不是1

ar.set();//全部设为1

ar.set(p);//第p位设为1

ar.reset();//全部设为0

ar.reset(p);//第p位设为0

ar.flip();//全部反转

ar.flip(p);//第p位反转

ar.to\_ulong();//返回unsigned long

ar.to\_ullong();//返回unsigned long long

ar.to\_string();//返回string

//方法一：

int bitCount(unsigned int n){

unsigned int tmp = n - ((n >> 1) & 033333333333) - ((n >> 2) & 011111111111);

return ((tmp + (tmp >> 3)) & 030707070707) % 63;

}

//方法二：

bitset<32> a(n);

a.count();

//方法三：

\_\_builtin\_popcount(n)//返回二进制位中有多少个1

\_\_builtin\_popcountll//longlong

\_\_builtin\_parity(n)//返回二进制位中1的数量的奇偶性，奇数返回1，偶数返回0

\_\_builtin\_ffs(n)//返回二进制末尾最后一个1的位置，从一开始

\_\_builtin\_ctz(n)//返回二进制末尾后面0的个数，当n为0时，和n的类型有关

#define LeftPos(x) 32 - \_\_builtin\_clz(x) - 1

#define LeftPosll(x) 64 - \_\_builtin\_clzll(x) - 1

[线段树区间合并](#_数据结构_1)

const int N = (int)1e5 +107;int n, m;

struct lp{

int l, r, d;

int lm,rm,mmax,lazy;

}cw[N<<2];

inline void push\_up(int rt){//更新父节点

cw[rt].lm = cw[lson].lm;//更新父亲左端点区间最长长度

cw[rt].rm = cw[rson].rm;//更新右端点

//父亲的最大长度，一定是左儿子最大长度，右儿子最大长度，和(左儿子rsum+右儿子lsum)这三者中的一个

cw[rt].mmax = max(max(cw[lson].mmax,cw[rson].mmax), cw[lson].rm+cw[rson].lm);

//如果左儿子是满的，就要把右儿子的lsum加给父亲

if(cw[lson].lm==cw[lson].d)cw[rt].lm+=cw[rson].lm;

//rsum同理

if(cw[rson].rm==cw[rson].d)cw[rt].rm+=cw[lson].rm;

}

void build(int l,int r,int rt){

int mid = (l + r)>>1;

cw[rt].l=l;cw[rt].r=r;

cw[rt].d=r-l+1;//区间长度

cw[rt].lazy = -1;

cw[rt].lm=cw[rt].rm=cw[rt].mmax=r-l+1;

if(l==r)return；

build(l,mid,lson);build(mid+1,r,rson);//push\_up(o,rt);

}

void push\_down(int rt){//把信息传给儿子节点

if(cw[rt].lazy!=-1){

if(cw[rt].lazy==0){//lazy==0清空区间

cw[lson].lazy=cw[rson].lazy=0;

cw[lson].lm=cw[lson].rm=cw[lson].mmax=0;

cw[rson].lm=cw[rson].rm=cw[rson].mmax=0;

}else{//lazy=1表示全部可以安排

cw[lson].lazy=cw[rson].lazy=1;

cw[lson].lm=cw[lson].rm=cw[lson].mmax=cw[lson].d;

cw[rson].lm=cw[rson].rm=cw[rson].mmax=cw[rson].d;

}

cw[rt].lazy=-1;

}

}

void update(int L,int R,int c,int rt){

int l=cw[rt].l,r=cw[rt].r,mid=(l+r)>>1;

if(L==l&&r==R){

cw[rt].lazy=c;

if(c)cw[rt].lm=cw[rt].rm=cw[rt].mmax=r-l+1;

else cw[rt].lm=cw[rt].rm=cw[rt].mmax=0;

return;

}

if(cw[rt].l==cw[rt].r)return;

push\_down(rt);

if(L>mid)update(L,R,c,rson);

else if(R<=mid)update(L,R,c,lson);

else update(L,mid,c,lson),update(mid+1,R,c,rson);

push\_up(rt);

}

//query函数要返回第一个可行的时间点

int query(int L,int R,int len,int rt){//len表示要安排的长度

int l=cw[rt].l,r=cw[rt].r,mid=(l+r)>>1;

if(l==r)return l;

if(cw[rt].l==cw[rt].r)return 0;

push\_down(rt);

//如果左儿子的max\_sum够用就去左儿子，如果左右儿子衔接处够用就返回衔接处，反之就去右儿子

if(cw[lson].mmax>=len)return query(1,mid,len,lson);

else if(cw[lson].rm+cw[rson].lm>=len)return mid-cw[lson].rm+1;

else return query(mid+1,R,len,rson);

}

[线段树](#_数据结构_1)

[矩形面积并(全部)](#_矩形面积并(全部)_1)

#define lson rt<<1

#define rson rt<<1|1

#define lsonl l,mid,rt<<1

#define rsonr mid+1,r,rt<<1|1

const int N = (int)1e5 +107;

struct lp{double l,r,h;int id;}seg[N];

bool cmp(const lp &a, const lp &b){return a.h<b.h;}

int n, m, tot;

int cnt[N<<2];

double sum[N<<2], all[N];

void push\_up(int l,int r,int rt){

if(cnt[rt]) sum[rt] = all[r+1]-all[l];

else if(l==r) sum[rt] = 0;

else sum[rt]=sum[lson]+sum[rson];

}

void update(int L,int R,int c,int l,int r,int rt){

if(L<=l&&r<=R){

cnt[rt] += c;

push\_up(l,r,rt);

return;

}

int mid = (l+r)>>1;

if(L>mid) update(L,R,c,rsonr);

else if(R<=mid) update(L,R,c,lsonl);

else {update(L,mid,c,lsonl);update(mid+1,R,c,rsonr);}

push\_up(l,r,rt);

}

int main(){

while(~scanf("%d",&n)){

for(int i=0;i<n;++i){

double x1,x2,y1,y2;

scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);

seg[i].l=x1;seg[i].r=x2;seg[i].h=y1;seg[i].id=1;

seg[i+n].l=x1;seg[i+n].r=x2;seg[i+n].h=y2;seg[i+n].id=-1;

all[i]=x1,all[i+n]=x2;

}

m = 2\*n;

sort(seg,seg+m,cmp);

sort(all,all+m);

int k = unique(all,all+m)-all-1;

memset(cnt,0,sizeof(cnt));

memset(sum,0,sizeof(sum));

double ans = 0;

for(int i=0;i<n\*2-1;++i){

int l = lower\_bound(all,all+k+1,seg[i].l)-all;

int r = lower\_bound(all,all+k+1,seg[i].r)-all;

if(l<r) update(l,r-1,seg[i].id,0,k,1);

ans += sum[1]\*(seg[i+1].h-seg[i].h);

}

printf("%.2f\n", ans);

}

return 0;

}

[矩形面积交(相交部分)](#_矩形面积交(相交部分)_1)

#define lson rt<<1

#define rson rt<<1|1

#define lsonl l,mid,rt<<1

#define rsonr mid+1,r,rt<<1|1

const int N = (int)1e5 +107;

struct lp{double l,r,h;int id;}seg[N];

bool cmp(const lp &a, const lp &b){return a.h<b.h;}

int n, m, tot, cnt[N<<2];

double one[N<<2], two[N<<2];

double all[N];

void push\_up(int l,int r,int rt){

if(cnt[rt]>=2){

one[rt] = two[rt] = all[r+1]-all[l];

}else if(cnt[rt]==1){

one[rt] = all[r+1]-all[l];

if(l==r)two[rt] = 0;

else two[rt]=one[lson]+one[rson];

}else{

if(l==r)two[rt]=one[rt]=0;

else{

one[rt]=one[lson]+one[rson];

two[rt]=two[lson]+two[rson];

}

}

}

void update(int L,int R,int c,int l,int r,int rt){

if(L<=l&&r<=R){

cnt[rt] += c;

push\_up(l,r,rt);

return;

}

int mid = (l+r)>>1;

if(L<=mid)update(L,R,c,lsonl);

if(R>mid)update(L,R,c,rsonr);

push\_up(l,r,rt);

}

int main(){

while(~scanf("%d",&n)){

for(int i=0;i<n;++i){

double x1,x2,y1,y2;

scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);

seg[i].l=x1;seg[i].r=x2;seg[i].h=y1;seg[i].id=1;

seg[i+n].l=x1;seg[i+n].r=x2;seg[i+n].h=y2;seg[i+n].id=-1;

all[i]=x1,all[i+n]=x2;

}

m = 2\*n;

sort(seg,seg+m,cmp);

sort(all,all+m);

int k = unique(all, all+m)-all-1;

memset(cnt,0,sizeof(cnt));

memset(one,0,sizeof(one));

memset(two,0,sizeof(two));

double ans = 0;

for(int i=0;i<n\*2-1;++i){

int l = lower\_bound(all,all+k+1,seg[i].l)-all;

int r = lower\_bound(all,all+k+1,seg[i].r)-all;

if(l<r) update(l,r-1,seg[i].id,0,k,1);

ans += two[1]\*(seg[i+1].h-seg[i].h);

}

printf("%.2f\n", ans);

}

return 0;

}

[矩形周长并](#_矩形周长并)

#define lson rt<<1

#define rson rt<<1|1

#define lsonl l,mid,rt<<1

#define rsonr mid+1,r,rt<<1|1

const int N = (int)1e5 +107;

struct lp{double l,r,h;int id;}seg[N];

struct lh{

int l,r;

double len;//有效长度

int s;//关键在是否被覆盖，其次记录被覆盖次数

int lc,rc;//左右端点是否被覆盖

int num;//区间有多少条线段

}cw[N<<2];

bool cmp(const lp &a, const lp &b){return a.h<b.h;}

int n, m;double all[N<<2];

void build(int l,int r,int rt){

cw[rt].l=l;cw[rt].r=r;

cw[rt].len=0;

cw[rt].s=cw[rt].lc=cw[rt].rc=cw[rt].num=0;

if(l==r)return;

int mid = (l+r)>>1;build(lsonl);build(rsonr);

}

void push\_up(int rt){

if(cw[rt].s){

cw[rt].lc=cw[rt].rc=1;

cw[rt].len = all[cw[rt].r+1]-all[cw[rt].l];

cw[rt].num=1;

}else if(cw[rt].l==cw[rt].r){

cw[rt].lc=cw[rt].rc=0;

cw[rt].len=0;

cw[rt].num=0;

}else{

cw[rt].lc=cw[lson].lc;

cw[rt].rc=cw[rson].rc;

cw[rt].len=cw[lson].len+cw[rson].len;

cw[rt].num = cw[lson].num+cw[rson].num-(cw[lson].rc&cw[rson].lc);

}

}

void update(int L,int R,int c,int rt){

int l = cw[rt].l, r = cw[rt].r, mid=(l+r)>>1;

if(L<=l&&r<=R){

cw[rt].s += c;

push\_up(rt);

return;

}

if(L<=mid)update(L,R,c,lson);

if(R>mid)update(L,R,c,rson);

push\_up(rt);

}

int main(){

while(~scanf("%d",&n)){

for(int i=0;i<n;++i){

double x1,x2,y1,y2;

scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);

seg[i].l=x1;seg[i].r=x2;seg[i].h=y1;seg[i].id=1;

seg[i+n].l=x1;seg[i+n].r=x2;seg[i+n].h=y2;seg[i+n].id=-1;

all[i]=x1,all[i+n]=x2;

}

m = 2\*n;

sort(seg,seg+m,cmp);

sort(all,all+m);

int k = unique(all, all+m)-all-1;

double ans = 0;

double last=0;

build(0,k,1);

for(int i=0;i<n\*2;++i){

int l = lower\_bound(all,all+k+1,seg[i].l)-all;

int r = lower\_bound(all,all+k+1,seg[i].r)-all;

if(l<r) update(l,r-1,seg[i].id,1);

ans += fabs(cw[1].len-last);

last = cw[1].len;

ans += cw[1].num\*2\*(seg[i+1].h-seg[i].h);

}

printf("%.0f\n", ans);

}

return 0;

}

优化建边

int seg[MXN][2];

void build(int id,int l,int r,int rt) {///注意有向图和无向图

seg[rt][id] = ++inde;///inde = n;

if(l == r) {

if(id) add\_edge(seg[rt][id], l, 0);///入树,被其他边指向

else add\_edge(l, seg[rt][id], 0);///出树,指向其他边

return;

}

int mid = (l + r) >> 1;

build(id, l, mid, lson); build(id, mid + 1, r, rson);

if(id) {

add\_edge(seg[rt][id], seg[lson][id], 0); add\_edge(seg[rt][id], seg[rson][id], 0);

}else {

add\_edge(seg[lson][id], seg[rt][id], 0); add\_edge(seg[rson][id], seg[rt][id], 0);

}

}

[树状数组](#_数据结构_1)

[一维树状数组该段求段](#_数据结构_1)

#define lowbit(x) (x)&(-(x))

//sum[i] = sigma(ar[x])+(i+1)\*sigma(delta[x])-sigma(x\*delta[x])

//delta[]是差分数组

void add(LL \*a, int x, LL v){

while(x <= n){

a[x] += v;

x += lowbit(x);

}

}

LL query(LL \*a, int x){

LL sum = 0;

while(x > 0){

sum += a[x];

x -= lowbit(x);

}

return sum;

}

void init(){

pre[0] = 0;

for(int i = 1; i <= n; ++i){

scanf("%lld", &ar[i]);

pre[i] = pre[i-1] + ar[i];

}

}

void update(int l, int r, LL x){

add(delta, l, x);add(delta, r+1, -x);

add(deltai, l, l\*x);add(deltai, r+1, -x\*(r+1));

}

LL range(int l, int r){

LL sum1 = pre[l-1]+l\*query(delta, l-1)-query(deltai, l-1);

LL sum2 = pre[r]+(r+1)\*query(delta, r)-query(deltai, r);

return sum2-sum1;

}

[二维改点求段](#_数据结构_1)

void add(int x, int y, int z){

int tmp = y;

while(x<=n){

y = tmp;

while(y<=n){

cw[x][y] += z, y += lowbit(y);

}

x += lowbit(x);

}

}

int query(int x, int y){

int res = 0, tmp = y;

while(x){

y = tmp;

while(y){

res += cw[x][y], y -= lowbit(y);

}

x -= lowbit(x);

}

return res;

}

[二维改段求点](#_数据结构_1)

//d[i][j]表示 a[i][j]与a[i−1][j]+a[i][j−1]−a[i−1][j−1]的差

//delta[][]是差分数组

void add(int x, int y, int z){

int tmp = y;

while(x <= n){

y = tmp;

while(y <= n){

delta[x][y] += z, y += lowbit(y);

}

x += lowbit(x);

}

}

void update(int xa,int ya,int xb,int yb,int z){

add(xa,ya,z);add(xa,yb+1,-z);add(xb+1,ya,-z);add(xb+1,yb+1,z);

}

int query(int x, int y){

int res = 0, tmp = y;

while(x){

y = tmp;

while(y){

res += delta[x][y], y -= lowbit(y);

}

x -= lowbit(x);

}

return res;

}

void init(){

for(int i = 1; i <= n; ++i){

for(int j = 1; j <= n; ++j){

int tmp = ar[i][j]-ar[i-1][j]-ar[i][j-1]+ar[i-1][j-1];

add(i,j,tmp);

}

}

}

[二维改段求段](#_数据结构_1)

//sum[x][y] = (x+1)(y+1)sigma(d[i][j])+sigma(i\*j\*d[i][j])

//-(y+1)sigma(i\*d[i][j])-(x+1)sigma(j\*d[i][j])

#define lowbit(x) (x)&(-(x))

using namespace std;

typedef long long LL;

const int INF = 0x3f3f3f3f;

const int N = (int)1e3 +107;

int ar[N][N], da[N][N], di[N][N], dj[N][N],dij[N][N];

int n, m, q;

void add(int x, int y, int z){

for(int i=x;i<=n;i+=lowbit(i)){

for(int j=y;j<=n;j+=lowbit(j)){

da[i][j] += z; di[i][j] += z\*x;

dj[i][j] += z\*y; dij[i][j] += z\*x\*y;

}

}

}

void update(int xa,int ya,int xb,int yb,int z){

add(xa,ya,z);add(xa,yb+1,-z);add(xb+1,ya,-z);add(xb+1,yb+1,z);

}

int query(int x, int y){

int res = 0;

for(int i = x; i>0; i -= lowbit(i)){

for(int j = y; j>0; j -= lowbit(j)){

res += (x+1)\*(y+1)\*da[i][j] - (y+1)\*di[i][j] - (x+1)\*dj[i][j] + dij[i][j];

}

}

return res;

}

int ask(int xa,int ya,int xb,int yb){

return query(xb,yb)-query(xb,ya-1)-query(xa-1,yb)+query(xa-1,ya-1);

}

void init(){

for(int i = 1; i <= n; ++i){

for(int j = 1; j <= n; ++j){

int tmp = ar[i][j]-ar[i-1][j]-ar[i][j-1]+ar[i-1][j-1];

add(i,j,tmp);

//update(i,j,i,j,ar[i][j]);

}

}

}

void solve(){

memset(ar,0,sizeof(ar));

for(int i=1;i<=n;++i){

for(int j=1;j<=n;++j){

scanf("%d",&ar[i][j]);

}

}

init();

}

[RMQ区间最值](#_RMQ区间最值)

const int N = 50005;

const int M = 1000005;

const int INF = 0x3f3f3f3f;

int mmax[N][30];

int ar[N];

int n,m;

void RMQ(){

for(int j=1;j<20;++j){

for(int i=1;i<=n;++i){

if(i+(1<<j)-1<=n){

mmax[i][j]=max(mmax[i][j-1],mmax[i+(1<<(j-1))][j-1]);

}

}

}

}

void solve(){

memset(mmin,0x3f,sizeof(mmin));

memset(mmax,0,sizeof(mmax));

for(int i=1;i<=n;++i)

mmax[i][0]=ar[i];

RMQ();

while(m--){

int a,b;

scanf("%d%d",&a,&b);

int k=(int)(log(b-a+1.0)/log(2.0));

int num\_max=max(mmax[a][k],mmax[b-(1<<k)+1][k]);

printf("%d\n",num\_max);

}

}

[主席树](#_主席树)

[第k大](#_第k大_1)

typedef long long LL;

const int N = 200005;

const int INF = 0x3f3f3f3f;

int n,q,tot;

struct lp{

int l,r,sum;

lp(){l=r=sum=0;}

}cw[N\*20];

int ar[N],br[N],now[N];

int root[N];

void update(int l,int r,int last,int &cur,int x){

cw[++tot]=cw[last];

cw[tot].sum++;

cur=tot;

if(l==r)return;

int mid = (l+r)>>1;

if(x<=mid){

update(l,mid,cw[last].l,cw[cur].l,x);

}else {

update(mid+1,r,cw[last].r,cw[cur].r,x);

}

}

int query(int l,int r,int last,int cur,int k){

if(l==r)return l;

int l1=cw[last].l,l2=cw[cur].l,r1=cw[last].r,r2=cw[cur].r;

int summ=cw[l2].sum-cw[l1].sum;

int mid=(l+r)>>1;

if(k<=summ){

return query(l,mid,cw[last].l,cw[cur].l,k);

}else{

return query(mid+1,r,cw[last].r,cw[cur].r,k-summ);

}

}

int main(){

while(~scanf("%d%d",&n,&q)){

memset(root,0,sizeof(root));

cw[0].l=cw[0].r=cw[0].sum=0;

for(int i=1;i<=n;++i){

scanf("%d",&br[i]);

ar[i]=br[i];

}

sort(br+1,br+1+n);

int k=1;

for(int i=2;i<=n;++i){

if(br[i]!=br[i-1])br[++k]=br[i];

}

tot=0;

for(int i=1;i<=n;++i){

now[i]=lower\_bound(br+1,br+1+k,ar[i])-br;

}

for(int i=1;i<=n;++i){

update(1,k,root[i-1],root[i],now[i]);

}

while(q--){

int u,v;

scanf("%d%d",&u,&v);

int L=1,R=v-u+1,mid,ans=1;

while(L<=R){//二分的写法大同小异，用自己习惯的写法

mid=(L+R)>>1;

int tmp=query(1,k,root[u-1],root[v],v-u+1-mid+1);

if(mid>br[tmp]){ans=mid-1;R=mid-1;

}else{ans=mid;L=mid+1;}

}

printf("%d\n",ans);

}

}

return 0;

}

[区间数字种类](#_区间数字种类)

#include<bits/stdc++.h>

using namespace std;

typedef long long LL;

const int N = 1e6+5;

int n,m;

struct lp{

int l,r,sum;

}cw[N\*20];

int rak[N],ar[N],br[N],vis[N],tot;

void update(int l,int r,int last,int &cur,int x,int v){

cw[++tot]=cw[last];

cw[tot].sum=cw[last].sum+v;

cur=tot;

if(l==r)return;

int mid=(l+r)>>1;

if(x<=mid)update(l,mid,cw[last].l,cw[cur].l,x,v);

else update(mid+1,r,cw[last].r,cw[cur].r,x,v);

}

int query(int l,int r,int p,int cur){

if(l==r)return cw[cur].sum;

int mid=(l+r)>>1;

if(p>mid)return query(mid+1,r,p,cw[cur].r);

else {

int ans=query(l,mid,p,cw[cur].l);

ans+=cw[cw[cur].r].sum;

return ans;

}

}

int main(){

while(~scanf("%d",&n)){

for(int i=1;i<=n;++i){

scanf("%d",&ar[i]);

}

tot=0;cw[0].l=cw[0].r=cw[0].sum=0;

memset(rak,0,sizeof(rak));

memset(vis,0,sizeof(vis));

for(int i=1;i<=n;++i){

if(vis[ar[i]]==0){

update(1,n,rak[i-1],rak[i],i,1);

vis[ar[i]]=i;

}else{

update(1,n,rak[i-1],rak[i],vis[ar[i]],-1);

vis[ar[i]]=i;

update(1,n,rak[i],rak[i],i,1);

}

}

scanf("%d",&m);

for(int i=0,u,v;i<m;++i){

scanf("%d%d",&u,&v);

printf("%d\n",query(1,n,u,rak[v]) );

}

}

return 0;

}

[划分树](#_划分树)

struct lp{

int cnt[N];//本层中从本段左端点到i有多少个进入左子树

int num[N];//本层中从本段的数字排列

LL sum[N];////本层中从1到i进入左子树的权值和

}cw[22];

LL pre[N],sum;

int n,m,num;

int sor[N];

int build(int l,int r,int d){

if(l==r)return cw[d].num[l];

cw[d].sum[0]=cw[d].cnt[0]=0;

int same=0,mid=(l+r)>>1;

for(int i=mid;i>=l;--i){//左边有多少的和mid值相等

if(sor[i]==sor[mid])same++;

else break;

}

int cnt=0;

int li=l,ri=mid+1;//下一层分段的起点，反正左右两边肯定是均分的(可能差1)

for(int i=l;i<=r;++i){

cw[d].sum[i]=cw[d].sum[i-1];//前缀进入左子树权值和

if(cw[d].num[i]<sor[mid]){//小于mid一律入左子树

cw[d+1].num[li++]=cw[d].num[i];

cw[d].sum[i]+=cw[d].num[i];

cnt++;

}else if(cw[d].num[i]==sor[mid]&&same){

same--;

cnt++;

cw[d+1].num[li++]=cw[d].num[i];

cw[d].sum[i]+=cw[d].num[i];

}else{

cw[d+1].num[ri++]=cw[d].num[i];

}

cw[d].cnt[i]=cnt;//统计本段到i进入左子树数量

}

build(l,mid,d+1);//递归

build(mid+1,r,d+1);

}

int query(int l,int r,int L,int R,int k,int d){

if(l==r)return cw[d].num[l];//如果只有一个数，直接返回

int left=0,sum\_in\_left=0,mid=(l+r)>>1;

if(L==l){

sum\_in\_left=cw[d].cnt[R];

}else{

sum\_in\_left=cw[d].cnt[R]-cw[d].cnt[L-1];//这是LR区间中进入左子树的数量

left=cw[d].cnt[L-1];//这是本段[l,L)区间中进入左子树的数量

}

int newl,newr;

if(sum\_in\_left>=k){//第k大数在左子树

newl=l+left;//L左边有left个进入左子树，过滤掉那些(不理解的话，取left为0理解

newr=l+left+sum\_in\_left-1;//下一段此区间有效的右边界

return query(l,mid,newl,newr,k,d+1);

}else{//在右子树

newl=mid+L-l+1-left;//过滤掉[l,L)中在右子树的数量

newr=mid+R-l+1-left-sum\_in\_left;//右边界(我把它化简了，不过很好推得

num+=sum\_in\_left;//把LR区间在左子树的数量累加

sum+=cw[d].sum[R]-cw[d].sum[L-1];//把LR区间在左子树权值累加

return query(mid+1,r,newl,newr,k-sum\_in\_left,d+1);

}

}

int main(int argc, char const \*argv[]){

int tim;

scanf("%d",&tim);

for(int T=1;T<=tim;++T){

scanf("%d",&n);

pre[0]=0;

for(int i=1;i<=n;++i){

scanf("%d",&sor[i]);

cw[0].num[i]=sor[i];

pre[i]=pre[i-1]+sor[i];//区里出前缀和，方便O1求值

}

sort(sor+1,sor+1+n);//记得排序呀兄dei

build(1,n,0);

scanf("%d",&m);

printf("Case #%d:\n",T);

while(m--){

int x,y,z;

scanf("%d%d",&x,&y);

x++;y++;//要自加一下，因为我的下标从1开始

sum=0;num=0;//初始化

z=(y-x+2)>>1;//取中间一位

int mid=(x+y)>>1;

LL tmp=query(1,n,x,y,z,0);

assert(num==mid-x);

num=mid-x;

sum=pre[y]-pre[x-1]-sum\*2-(y-x+1-num-num)\*tmp;//推出来的公式

printf("%lld\n",sum );

}

printf("\n");//PE一发你就知道咯

}

return 0;

}

[分块区间众数](#_分块区间众数)

const int N = 1e5+7;

int n,belong[N],br[N],now[N],cnt[N],f[500][500];

int l[N],r[N],block,Num,k;

vector<int> g[N];

int ar[N],cw[500][N];

inline void build(){

block=sqrt(n\*1.0);

Num=n/block;if(n%block)Num++;

for(int i=1;i<=Num;++i){

l[i]=(i-1)\*block+1;r[i]=i\*block;

}

r[Num]=n;

for(int i=1;i<=n;++i){

belong[i]=(i-1)/block+1;

now[i]=lower\_bound(br+1,br+1+k,ar[i])-br;

cw[belong[i]][now[i]]++;

}

for(int i=1;i<=n;++i)g[now[i]].push\_back(i);

for(int i=1;i<=Num;++i){

for(int j=1;j<=k;++j){

cw[i][j]+=cw[i-1][j];

}

}

}

inline int get(int a,int b,int x){

if(belong[a]==belong[b]){

return cw[belong[b]][x]-cw[belong[a]][x];

}

return cw[belong[b]-1][x]-cw[belong[a]][x];

//return (upper\_bound(g[x].begin(),g[x].end(),b)-lower\_bound(g[x].begin(),g[x].end(),a));

}

inline void chaobaoli(int a,int b){

for(int i=0;i<=k;++i)cnt[i]=0;

int p=f[belong[a]+1][belong[b]-1];

int num\_max=get(a,b,p);

int da=min(r[belong[a]],b);

for(int i=a;i<=da;++i){

cnt[now[i]]++;

int tnum=get(a,b,now[i])+cnt[now[i]];

if(tnum>num\_max||(tnum==num\_max&&br[p]>br[now[i]])){

p=now[i];

num\_max=tnum;

}

}

if(belong[a]!=belong[b])

for(int i=l[belong[b]];i<=b;++i){

cnt[now[i]]++;

int tnum=get(a,b,now[i])+cnt[now[i]];

if(tnum>num\_max||(tnum==num\_max&&br[p]>br[now[i]])){

p=now[i];

num\_max=tnum;

}

}

printf("%d\n",br[p] );

}

int main(){

scanf("%d",&n);

for(int i=1;i<=n;++i)scanf("%d",&ar[i]);,br[i]=ar[i];

sort(br+1,br+1+n);

k=1;

for(int i=2;i<=n;++i)

if(br[i]!=br[i-1])

br[++k]=br[i];

build();

for(int i=1;i<=Num;++i){

for(int i=0;i<=k;++i)cnt[i]=0;

int num\_max=0,p=0;

for(int j=l[i];j<=n;++j){

cnt[now[j]]++;

if(cnt[now[j]]>num\_max||(cnt[now[j]]==num\_max&&br[p]>br[now[j]])){

p=now[j];num\_max=cnt[now[j]];

}

f[i][belong[j]]=p;

}

}

for(int i=0,a,b;i<n;++i){

scanf("%d%d",&a,&b);

if(a>b)a^=b^=a^=b;

chaobaoli(a,b);

}

return 0;

}

[分块动态第k大](#_分块动态第k大)

const int N = 50007;

int n,m;

int ar[N],l[N],r[N],belong[N];

int block,num;

vector<int>b[N];

void build(){//这是qsc学姐分块写法

block=sqrt(n);//块的大小

num=n/block;if(n%block)num++;//块的数量

for(int i=1;i<=num;++i){

l[i]=(i-1)\*block+1;r[i]=i\*block;//每一块的左右区间

}

r[num]=n;

for(int i=1;i<=n;++i){

belong[i]=(i-1)/block+1;//每一点属于哪一块

}

for(int i=0;i<=num;++i)b[i].clear();//一定要记得初始化

for(int i=1;i<=n;++i){

b[belong[i]].push\_back(ar[i]);//把值放入块中

}

for(int i=1;i<=num;++i){

sort(b[i].begin(),b[i].end());//要排序，因为后面查询要用到lowerboud

}

}

int q(int m,int x,int y){//查询节点x到y中小于等于m的数的数量

int cnt=0;

if(belong[x]==belong[y]){//如果在同一块，直接暴力枚举

for(int i=x;i<=y;++i){

if(ar[i]<=m)cnt++;

}

return cnt;

}

for(int i=belong[x]+1;i<belong[y];++i){//两节点之间的完整的块使用lowerbound

cnt+=upper\_bound(b[i].begin(),b[i].end(),m)-b[i].begin();

}

if(x==l[belong[x]]){//如果x是其所属块的左端点，也使用lowerbound

int i=belong[x];

cnt+=upper\_bound(b[i].begin(),b[i].end(),m)-b[i].begin();

}else{

for(int i=x;i<=r[belong[x]];++i){

if(ar[i]<=m)cnt++;

}

}

if(y==r[belong[y]]){//如果y是其所属块的右端点

int i=belong[y];

cnt+=upper\_bound(b[i].begin(),b[i].end(),m)-b[i].begin();

}else{

for(int i=l[belong[y]];i<=y;++i)

if(ar[i]<=m)cnt++;

}

return cnt;

}

int get(int k,int a,int B,int r){//二分查找第k小的数字，大部分人写二分的习惯都不同，尽量使用自己习惯的写法

int l=0,ans=0,mid;

while(l<=r){

mid=(l+r)>>1;

int cnt=q(mid,a,B);

if(cnt>=k){ans=mid;r=mid-1;

}else{ans=mid+1;l=mid+1;}

}

return ans;

}

void ch(int &x,int &y){x^=y^=x^=y;}

int main(){

int T;

scanf("%d",&T);

while(T--){

scanf("%d%d",&n,&m);

int mmax=0;

for(int i=1;i<=n;++i){

scanf("%d",&ar[i]);

mmax=max(mmax,ar[i]);//mmax是二分查找的上界

}

build();

for(int i=0;i<m;++i){

char s[2];int a,B,c;

scanf("%s %d %d",s,&a,&B);

if(s[0]=='Q'){

scanf("%d",&c);

printf("%d\n",get(c,a,B,mmax) );

}else{

mmax=max(mmax,B);

int tmp=belong[a];//先获取节点在块中的位置

int pos=lower\_bound(b[tmp].begin(),b[tmp].end(),ar[a])-b[tmp].begin();

ar[a]=b[tmp][pos]=B;//左右移动调整

while(pos>0&&b[tmp][pos]<b[tmp][pos-1]){

ch(b[tmp][pos],b[tmp][pos-1]);

pos--;

}

int size=b[tmp].size()-1;

while(pos<size&&b[tmp][pos]>b[tmp][pos+1]){

ch(b[tmp][pos],b[tmp][pos+1]);

pos++;

}

}

}

}

return 0;

}

[莫队](#_莫队)

[修改莫队-颜色种类](#_修改莫队-颜色种类)

#include <bits/stdc++.h>

using namespace std;

const int N = 1e5+7;

struct lp{

int l,r,id,last;

int old,now;

}cw[N],tim[N];

int n,m,ar[N],arr[N],belong[N],ans,Ans[N],cnt[N\*10];

bool cmp(const lp &a,const lp &b){

if(belong[a.l]!=belong[b.l])return belong[a.l]<belong[b.l];

if(belong[a.r]!=belong[b.r])return belong[a.r]<belong[b.r];

return a.last<b.last;

}

void update(int x,int f){

cnt[ar[x]]+=f;

if(f==1&&cnt[ar[x]]==1)ans++;

if(f==-1&&cnt[ar[x]]==0)ans--;

}

void update\_time(int t,int f,int l,int r){

if(tim[t].l>=l&&tim[t].l<=r){

cnt[tim[t].old]-=f;

cnt[tim[t].now]+=f;

if(f==1){

if(cnt[tim[t].old]==0)ans--;

if(cnt[tim[t].now]==1)ans++;

}else{

if(cnt[tim[t].old]==1)ans++;

if(cnt[tim[t].now]==0)ans--;

}

}

if(f==1)ar[tim[t].l]=tim[t].now;

else ar[tim[t].l]=tim[t].old;

}

int main() {

while(~scanf("%d%d",&n,&m)){

int block=2\*sqrt(n\*1.0);

for(int i=1;i<=n;++i){

scanf("%d",&ar[i]);

arr[i]=ar[i];

belong[i]=(i-1)/block+1;

}

char op[2];

int tot=0,change=0;

for(int i=0,l,r;i<m;++i){

scanf("%s%d%d",op,&l,&r);

if(op[0]=='Q'){

cw[tot].last=change;cw[tot].l=l;cw[tot].r=r;

cw[tot].id=tot;tot++;

}else{

tim[++change].l=l;

tim[change].old=arr[l];

tim[change].now=r;

arr[l]=r;

}

}

ans=0;

memset(cnt,0,sizeof(cnt));

sort(cw,cw+tot,cmp);

for(int i=0,L=1,R=0,t=0;i<tot;++i){

for(;t<cw[i].last;)

update\_time(++t,1,L,R);

for(;t>cw[i].last;)

update\_time(t--,-1,L,R);

while(L<cw[i].l)update(L++,-1);

while(L>cw[i].l)update(--L,1);

while(R<cw[i].r)update(++R,1);

while(R>cw[i].r)update(R--,-1);

Ans[cw[i].id]=ans;

}

for(int i=0;i<tot;++i)

printf("%d\n",Ans[i] );

}

return 0;

}

[数论](#_目录)

[欧拉函数](#_欧拉函数)

1.欧拉函数

对正整数n，欧拉函数是少于或等于n的数中与n互质的数的数目

显然对素数n,phi(n)=n-1

通式：φ(x)=x(1-1/p1)(1-1/p2)(1-1/p3)(1-1/p4)…..(1-1/pn),其中p1, p2……pn为x的所有质因数，x是不为0的整数。φ(1)=1（唯一和1互质的数就是1本身）

phi[p^a]=p^a - p^(a-1);

2.降幂公式

当x≥ϕ(p)时，有a x ≡ a xmodϕ(p)+ϕ(p)(mod p)

gcd(a,mod)=1 a^(x)=a^(x%ϕ(mod)) %mod

gcd(a,mod)!=1 a^(x)=a^(x%ϕ(mod) + ϕ(mod)) %mod only if x>=ϕ(mod)

gcd(a,mod)!=1 a^(x)=a^(x%ϕ(mod)) %mod only if x<ϕ(mod)

这部分可以参考“上帝与集合”

void init(){

for(int i = 1; i < maxn; i++) phi[i] = i;

for(int i = 2; i < maxn; i++)

if(phi[i] == i)

for(int j = i; j < maxn; j += i)

phi[j] -= (phi[j] / i);// phi[j]=phi[j]/i\*(i-1);

}

LL euler(LL n){

LL res = n, a = n;

for(LL i = 2; i \* i <= a; i++) {

if(a % i == 0) {

res = res / i \* (i - 1); //先进行除法是为了防止中间数据的溢出

while(a % i == 0) a /= i;

}

}

if(a > 1) res = res / a \* (a - 1);

return res;

}

int prime[N + 5], phi[N + 5]; bool noprime[N + 5];

//1. phi(p)=p-1 因为质数p除了1以外的因数只有p，故1至p的整数只有p与p不互质

//2. 如果i mod p = 0, 那么phi(i \* p)=p \* phi(i)

//3.若i mod p ≠0, 那么phi(i \* p)=phi(i) \* (p-1)

void euler(){

int size = 0;

phi[1] = 1;

for(int i = 2; i <= N; i++) {

if(!noprime[i]) {

prime[++size] = i;

phi[i] = i - 1;

}

for(int j = 1; j <= size && i \* prime[j] <= N - 5; j++) {

noprime[i \* prime[j]] = 1;

if(i % prime[j] == 0) {

phi[i\*prime[j]] = prime[j]\*phi[i]; //phi[p\*i]=phi[p\*p\*k]=phi[k\*p\*p]=phi[k]\*phi[p]\*p=p\*phi[i]

break;

}

phi[i\*prime[j]] = phi[i]\*(prime[j] - 1);

//phi[nm]=phi[n]\*phi[m] gcd(n,m) = 1

}

}

}

sigma(gcd (i , n)) = sigma (d \* phi[n / d]) d | n

[质因数分解](#_质因数分解_1)

const int N = (int)2e5 + 7;

int noprime[N], pcnt, p[N / 2];

int nump[N / 2], yinzi[N / 2];

int n, m, top;

void getprime(){

pcnt = 0;

memset(noprime, 0, sizeof(noprime));

noprime[0] = noprime[1] = 1;

for(int i = 2; i < N; ++i) {

if(!noprime[i])p[pcnt++] = i,mu[i]=-1;;

for(int j = 0; j < pcnt && i \* p[j] < N; ++j) {

noprime[i \* p[j]] = 1;

if(i % p[j] == 0){ mu[i\*prime[j]] = 0;break;}

mu[i\*prime[j]] = mu[i] \* -1;

}

}

}

void cal(int t){

memset(nump, 0, sizeof(nump));

top = -1;

int tmp = (int)sqrt(t\*1.0);

for(int i = 0; i < pcnt && p[i] <= tmp; ++i) {

if(t % p[i] == 0) {

yinzi[++top] = p[i];

while (t % p[i] == 0) {

nump[top] ++;

t /= p[i];

}

}

if(t == 1)break;

}

if(t > 1) {

yinzi[++top] = t;

nump[top] ++;

}

}

n!中因子p的个数可以这样求：

int t = n,sum = 0;

while ( t > 0 ) {

sum += t / p;

t /= p;

}

计算n!中质因子p的个数x的公式为x=⌊n/p⌋+⌊n/p2⌋+⌊n/p3⌋+...

递推式也可以写为f(n)=f(⌊n/p⌋)+⌊n/p⌋

sum即为n!中因子p的个数

x = p1^a1 \* p2^a2 \* ... \* pn^an

x的因数个数为(a1+1)\*...\*(an+1)

取模运算

void Add(LL &x, LL y){

x += y;

if(x>=mod)x-=mod;

}

void Sub(LL &x, LL y){

x = x + mod - y;

if(x>=mod)x-=mod;

}

void Mul(LL &x, LL y){

x \*= y;

if(x>=mod)x%=mod;

}

(a/b) mod k = d;

(a/b) = kx + d;

a = (kb)x + bd;

a mod kb = bd;

(a mod kb)/b = d;

调和级数

Hn=ln(n)+c+1.0/(2\*n)

sum=log(n+1)+R-1.0/(2\*n);

r=0.57721566490153286060651209（r就是欧拉常数）。

当n大于1e5时，答案等于

0.57721566490153286060651209+log(n)

杂烩

gcd(a^n - 1,a^m - 1) = a^gcd(n,m) - 1

前n个数的逆元

const int MOD = 998244353;

const int N = 1e6 + 7;

LL inv[N];

void getinv(){//mod为素数

inv[1]=1;

for(int i=2;i<N;i++)

inv[i]=(MOD-MOD/i)\*1LL\*inv[MOD%i]%MOD;

}

LL inv(LL t,LL p){//求t关于p的逆元

return t==1?1:(p-p/t)\*inv(p%t,p)%p;

}

printf("%lld\n",inv(a%p,p));

inline int inv(int a) {

a %= md;

if (a < 0) a += md;

int b = md, u = 0, v = 1;

while (a) {

int t = b / a;

b -= t \* a; swap(a, b);

u -= t \* v; swap(u, v);

}

assert(b == 1);

if (u < 0) u += md;

return u;

}

int exgcd(int a,int b,int &d,int &x,int &y){

if(b==0){

x=1;y=0;d=a;

return;

}

exgcd(b,a%b,d,y,x);

y -= a/b\*x;

return;

}

LL f[N],inv[N],b[M];

LL exgcd(LL a,LL b,LL &x,LL &y){

if(b==0){

x=1;y=0;

return a;

}

LL d = exgcd(b,a%b,y,x);

y -= a/b\*x;

return d;

}

LL CRT\_Prime(LL a[],LL b[],LL n){

//a[i] 是 mod b[i]（模数） 后的结果,b[i]两两互质

LL m = 1, ans = 0, m\_i , x , y , tmp;

for(int i=0;i<n;++i) m\*=b[i];

for(int i=0;i<n;++i){//O(n\*log(m\_i)\*log(a[i]))

m\_i = m/b[i];

exgcd(m\_i,b[i],x,y);

ans += qmul(x,(qmul(m\_i,a[i],m)%m),m)%m;

//快速乘根据是否会爆精度来使用 不爆的话不建议使用

ans %= m;

}

return (ans%m+m)%m;

}

void Pre\_Print(LL f[],LL inv[],LL mod){

//打表 小于等于 n 的所有阶乘 及其逆元 mod 是个大素数

f[0] = 0;f[1] = 1;inv[1] = 1;

LL x,y;

for(int i=2;i<N;++i){//O(nlog(max(f[i],mod)))

f[i] = (1ll\*i\*f[i-1])%mod;

//exgcd(f[i],mod,x,y);//fermat is allowed too

//x = (x%mod+mod)%mod;

//inv[i] = x;

}

exgcd(f[N-1],mod,x,y);

inv[N-1]=(x%mod+mod)%mod;

for(int i=N-2;i>1;--i)inv[i]=inv[i+1]\*(i+1)%mod;

return ;

}

LL Select\_comb(LL n,LL m,LL f[],LL inv[],LL mod){

//打表后求组合数的结果

if(n < m) return 0;

if(n == m) return 1;

LL ans = (((f[n]\*inv[m])%mod)\*inv[n-m])%mod;

return ans%mod;

}

LL inv\_Cal(LL val,LL mod){

LL x,y;

exgcd(val,mod,x,y);

return ((x%mod+mod)%mod);

}

LL comb(LL n,LL m,LL mod){

//单一C(n,m)求法 O(mlogm)

if(m > n) return 0;

LL ans = 1,x,y;

//简化计算直接求n\*(n-1)\*..\*(n-m+1)/m!;

for(int i=1;i<=m;++i){

x = (n+1-i\*1ll)%mod;

y = inv\_Cal(1ll\*i,mod);

ans = (((ans\*x)%mod)\*y)%mod;

}

return ans%mod;

}

Miller-rabin

long long mul(long long a, long long b, long long mod){

long long res = 0;

for(;b;b>>=1,a=(a+a)%mod){

if(b&1)res = (res + a)%mod;

}

return res;

}

long long power(long long v, long long p, long long m){

long long r = 1;

while(p){

if(p & 1) r = r \* v % m;

//if(p & 1) r = mul(r, v, m);

v = v \* v % m;

//v = mul(v, v, m);

p >>= 1;

}

return r;

}

bool witness(long long a, long long p){

int k = 0;

long long q = p - 1;

while((q & 1) == 0)

++k, q >>= 1;

long long v = power(a, q, p);

if(v == 1 || v == p - 1)

return false; // probably prime number

while(k-- != 0){

v = v \* v % p;

if(v == p - 1)

return false;

}

return true; // composite number

}

bool miller\_rabin(long long p){

if(p == 2) return true;

if(p % 2 == 0) return false;

for(int i = 0; i != 50; ++i){

long long a = std::rand() % (p - 1) + 1;

if(witness(a, p))

return false;

}

return true;

}

O(1)快速乘

LL mul(LL a,LL b,LL mo){

LL tmp=(a\*b-(LL)((long double)a/mo\*b+1e-8)\*mo);

return tmp<0?tmp+mo:(tmp>=mo?tmp-mo:tmp);

}

LL mul(LL x, LL y,LL mo) {

LL z = (long double) x \* y / mo; z = x \* y - z \* mo;

if(z < 0) z += mo; else if(z > mo) z -= mo;

return z;

}

求平方根

double sqr(double x){

double k = x;

while(k \* k - x > 1e-9)

k = 0.5 \* (k + x / k);

return k;

}

[矩阵快速幂](#_数论)

struct lp{

LL ar[55][55];

lp(){memset(ar,0,sizeof(ar));}

};

lp exe(lp a,lp b){

lp c;//c[i][j]= Σ a[i][k]\*b[k][j]

for(int k=0;k<n;++k){

for(int i=0;i<n;++i){

if(a.ar[i][k]==0)continue;

for(int j=0;j<n;++j){

if(b.ar[k][j]==0)continue;

c.ar[i][j]+=a.ar[i][k]\*b.ar[k][j];

if(c.ar[i][j]>=mod){//取模的复杂度比较高，所以尽量减少去模运算

c.ar[i][j]%=mod;

}

}

}

}

return c;

}

lp cal(lp a,int b){

lp ret;

for(int i=0;i<n;++i)

for(int j=0;j<n;++j)

ret.ar[i][j]=(i==j);

while(b>0){

if(b&1)ret=exe(ret,a);

a=exe(a,a);

b>>=1;

}

return ret;

}

[Lucas](#_Lucas)

//数据范围较小的组合数

const int N = 1e5 + 10;

LL ar[N], sum[N], f[N];

LL comb(LL n,LL m){

if (n<m)return 0;

if(m>n-m)m=n-m;

LL s1=1,s2=1,i;

for(i=1;i<=m;i++){

s1=s1\*(n-i+1)%mod;

s2=s2\*i%mod;

}

return s1\*ksm(s2,mod-2,mod)%mod;

}

LL lucas(LL n,LL m){

if (m==0)return 1;

return (lucas(n/mod,m/mod)\*comb(n%mod,m%mod)%mod);

}

inv[1]=1; fac[0]=facInv[0]=1;

for(int i=1; i<= N; i++) {

if(i!=1) inv[i] = (mod-mod/i)\*inv[mod%i]%mod;

fac[i] = fac[i-1]\*i%mod;

facInv[i] = facInv[i-1]\*inv[i]%mod;

}

LL lucas(int n, int m) {

if(n<m) return 0;

LL ans=1;

for(; m; n/=mod, m/=mod) ans = ans\*COMB(n%mod, m%mod)%mod;

return ans;

}

const LL mod = 1e9 + 7;

const int MX = 1e6 + 5;

LL F[MX], invF[MX];

LL ksm(LL a, LL b){

LL res = 1;

for(;b;b>>=1,a=a\*a%mod){

if(b&1)res = res \* a % mod;

}

return res;

}

void init() {//阶乘和阶乘的逆元

F[0] = 1;

for (int i = 1; i < MX; i++) F[i] = F[i - 1] \* i % mod;

invF[MX - 1] = ksm(F[MX - 1], mod - 2);

for (int i = MX - 2; i >= 0; i--) invF[i] = invF[i + 1] \* (i + 1) % mod;

}

LL COMB(LL n, LL m) {

LL ret = 1;

while (n && m) {

LL nn = n % mod, mm = m % mod;

if (nn < mm) return 0;

ret = ((ret \* F[nn] % mod) \* invF[mm] % mod) \* invF[nn - mm] % mod;

n /= mod, m /= mod;

}

return ret;

if(n == m)return 1;

if(n < m) return 0;

return F[n]\*invF[m]%mod\*invF[n-m]%mod;

}

exLucas

//http://codeforces.com/gym/100633/problem/J

#include<bits/stdc++.h>

using namespace std;

typedef long long LL;

const int N = 5e1 + 7;

const int ME = 1e6 + 7;

const int MOD = 1000000007;

const int INF = 0x3f3f3f3f;

LL n, m, Mod;

LL ksm(LL a, LL b, LL mod){

LL res = 1;

for(;b;b>>=1,a=a\*a%mod){

if(b&1)res = res \* a % mod;

}

return res;

}

void exgcd(LL a,LL b,LL &x,LL &y){

if (!b) x=1LL,y=0LL;

else exgcd(b,a%b,y,x),y-=a/b\*x;

}

LL Inv(LL A,LL mod){

if (!A) return 0LL;

LL a=A,b=mod,x=0LL,y=0LL;

exgcd(a,b,x,y);

x=((x%b)+b)%b;

if (!x) x+=b;

return x;

}

LL Mul(LL n,LL pi,LL pk){

if (!n) return 1LL;

LL ans=1LL;

if (n/pk){

for (LL i=2;i<=pk;++i)

if (i%pi) ans=ans\*i%pk;

ans=ksm(ans,n/pk,pk);

}

LL r = n%pk;

for (LL i=2;i<=r;++i)//不足的部分

if (i%pi) ans=ans\*i%pk;

return ans\*Mul(n/pi,pi,pk)%pk;//再递归一层算(n/pi)!

}

LL COMB(LL n,LL m,LL mod,LL pi,LL pk){

if (n<m) return 0;

LL a=Mul(n,pi,pk),b=Mul(m,pi,pk),c=Mul(n-m,pi,pk);

LL k=0LL,ans;

for (LL i=n;i;i/=pi) k+=i/pi;

for (LL i=m;i;i/=pi) k-=i/pi;

for (LL i=n-m;i;i/=pi) k-=i/pi;

ans=a\*Inv(b,pk)%pk\*Inv(c,pk)%pk\*ksm(pi,k,pk)%pk;

return ans\*(mod/pk)%mod\*Inv(mod/pk,pk)%mod;//CRT合并

}

LL exLucas(LL n, LL m, LL mod){

LL ans = 0;

for (LL x=Mod,i=2;i<=Mod;++i){

if (x%i==0){

LL pk = 1LL;

while (x%i==0) pk\*=i,x/=i;

ans=(ans+COMB(n,m,Mod,i,pk))%Mod;

}

}

return ans;

}

int main(){

while(~scanf("%I64d%I64d%I64d",&n,&m,&Mod)){

printf("%I64d\n",exLucas(n, m, Mod));

}

}

求解n!可以分为3部分：第一部分是pi的幂的部分，也就是3^6即pi^⌊n/pi⌋，可以直接求解；

第二部分是一个新的阶乘，也就是6!即⌊n/pi⌋!，可以递归下去求解；

第三部分是除前两部分之外剩下的数 考虑第三部分如何求解

发现第三部分在模pi^ki意义下是以pi^ki为周期的，

即(1∗2∗4∗5∗7∗8)≡(10∗11∗13∗14∗16∗17)(mod pi^ki)，所以只求pi^ki长度的即可；

可以按pr分块，一共n/pr块

但是还剩下一个孤立的19，可以发现剩下孤立的数长度不会超过pi^ki，只需要暴力求解即可

e.最后一个问题是对于求出的m!%pi^ki和(n−m)!%pi^ki有可能与pi^ki不互质，无法求逆元

所以要将m!%pi^ki和(n−m)!%pi^ki中质因子pi先全部除去，求出逆元后再全部乘回去

计算n!中质因子p的个数x的公式为x=⌊n/p⌋+⌊n/p2⌋+⌊n/p3⌋+...

递推式也可以写为f(n)=f(⌊n/p⌋)+⌊n/p⌋

除法分块

f(X)表示X所有约数的和.求f(X)+f(X+1)+……+f(Y)的值.

ans=∑⌊n/i⌋×i

列举12的⌊n/i⌋值：12,6,4,3,2,2,1,1,1,1,1,1

这个意思是1出现了12次，2出现了6次，3出现了4次…12出现了一次。

除法分块就把出现次数相同的在一起算。因为数字的连着的没所以可以用等差数列求和来计算答案。

对于出现次数相同的区间，通过打表找规律得到：[L,R] R=N/(N/L) ，一个L等于上一个R加1.

LL get\_sum(LL x){

LL ans=0;

for(LL left=1,right;left<=x;left=right+1){

right = x/(x/left);

ans += (x/left)\*(left+right)\*(right-left+1)/2;

}

return ans;

}

ar[i][j]表示i的第j个素数因子，sum[i][j]表示这个因子的个数

int cnt[N], sum[N][20];

uLL val[N], ar[N][20];

void exe(){

mme(cnt, 0);mme(sum, 0);mme(ar, 0);

uLL len = ri-le+1;

for(int i = 0; i < len; ++i)val[i] = i + le;

for(int i = 0; i < pcnt; ++i){

for(uLL h = (le+pm[i]-1)/pm[i]\*pm[i]; h <= ri; h += pm[i]){

uLL j = h-le;

ar[j][cnt[j]] = pm[i];

while(val[j]%pm[i] == 0)sum[j][cnt[j]]++, val[j] /= pm[i];

cnt[j]++;

}

}

for(int i = 0; i < len; ++i){

if(val[i] > 1){

ar[i][cnt[i]] = val[i];

sum[i][cnt[i]]++;

cnt[i]++;

}

}

}

容斥

奇加偶减

vector<int> ve;

LL inv(LL t){

return t == 1LL? 1LL: (MOD-MOD/t)\*inv(MOD%t)%MOD;

}

LL get\_num1(LL n){

return n\*(n+1)%MOD\*inv2%MOD;

}

LL get\_num2(LL n){

return n\*(n+1)%MOD\*(2\*n%MOD+1)%MOD\*inv6%MOD;

}

LL slove(LL n, LL m) {

inv2 = inv(2), inv6 = inv(6);

ve.clear();

int tm = m;

for(int i = 2; (LL)i \* i <= m && i <= n; ++i){

if(tm % i == 0){

ve.push\_back(i);

while(tm % i == 0)tm /= i;

}

if(tm == 1)break;

}

if(tm != 1)ve.push\_back(tm);

int len = ve.size(), state = 1 << len;

LL ans = (get\_num1(n)+get\_num2(n))%MOD;

ans = 0;

for(int i = 0; i < state; ++i){

LL tot = 1, cnt;

int num = 0;

for(int j = 0; j < len; ++j){

if(i&(1<<j)){

++num;

tot \*= ve[j];

}

}

cnt = n/tot;

//printf("\*%lld %lld %d\n", tot,cnt,num);

if(num % 2 == 0){

ans = (ans+ tot\*tot%MOD\*get\_num2(cnt)%MOD+tot\*get\_num1(cnt)%MOD)%MOD;

}else{

ans = (ans- tot\*tot%MOD\*get\_num2(cnt)%MOD-tot\*get\_num1(cnt)%MOD)%MOD;

ans = (ans + MOD)%MOD;

}

}

return ans;

}

一般的积性函数

for(int i = 2; i < N; i ++){

if(!noprime[i]){

pm[++pcnt] = i;

low[i] = lowp[i] = i;

f[i] = ...;

}

for(int j = 1; i\*pm[j] < N; j ++){

noprime[i\*pm[j]] = 1;

if(i%pm[j] == 0){

low[i\*pm[j]] = pm[j];

lowp[i\*pm[j]] = lowp[i] \* pm[j];

if(i == lowp[i]) f[i] = ...;

else f[i\*pm[j]] = f[i/lowp[i]] \* f[lowp[i]\*pm[j]];

break;

}else {

low[i\*pm[j]] = lowp[i\*pm[j]] = pm[j];

f[i\*pm[j]] = f[i] \* f[pm[j]];

}

}

}

卡特兰数

递推公式h[1] = 1,h[n] = h[n-1] \*(4\*n-2)/(n+1)

计算几何

模板

const double eps = 1e-10;

const double PI = acos (-1.0);

int dcmp(double x) {//三态函数，减少精度问题

if (fabs (x) < eps) return 0;

else return x < 0 ? -1 : 1;

}

struct Point {//点的定义

double x, y;

Point () {}

Point (double x, double y) : x (x), y (y) {}

Point operator + (const Point &r) const {//向量加法

return Point (x + r.x, y + r.y);

}

Point operator - (const Point &r) const {//向量减法

return Point (x - r.x, y - r.y);

}

Point operator \* (double p) const {//向量乘以标量

return Point (x \* p, y \* p);

}

Point operator / (double p) const {//向量除以标量

return Point (x / p, y / p);

}

bool operator < (const Point &r) const {//点的坐标排序

return x < r.x || (x == r.x && y < r.y);

}

bool operator == (const Point &r) const {//判断同一个点

return dcmp (x - r.x) == 0 && dcmp (y - r.y) == 0;

}

};

typedef Point Vector;//向量的定义

Point read\_point(void){//点的读入

double x, y;scanf ("%lf%lf", &x, &y);

return Point (x, y);

}

double dot(Vector A, Vector B) {//向量点积

return A.x \* B.x + A.y \* B.y;

}

double cross(Vector A, Vector B) {//向量叉积

return A.x \* B.y - A.y \* B.x;

}

double polar\_angle(Vector A) {//向量极角

return atan2 (A.y, A.x);

}

double Length(Vector A) {//向量长度，点积

return sqrt (dot (A, A));

}

double angle(Vector A, Vector B) {//向量转角，逆时针，点积

return acos (dot (A, B) / Length (A) / Length (B));

}

Vector rotate(Vector A, double rad) {//向量旋转，逆时针

return Vector (A.x \* cos (rad) - A.y \* sin (rad), A.x \* sin (rad) + A.y \* cos (rad));

}

Vector nomal(Vector A) {//向量的单位法向量

double len = Length (A);

return Vector (-A.y / len, A.x / len);

}

Point line\_line\_inter(Point p, Vector V, Point q, Vector W) {//两直线交点，参数方程

Vector U = p - q;

double t = cross (W, U) / cross (V, W);

return p + V \* t;

}

double point\_to\_line(Point p, Point a, Point b) {//点到直线的距离，两点式

Vector V1 = b - a, V2 = p - a;

return fabs (cross (V1, V2)) / Length (V1);

}

double point\_to\_seg(Point p, Point a, Point b) { //点到线段的距离，两点式

if (a == b) return Length (p - a);

Vector V1 = b - a, V2 = p - a, V3 = p - b;

if (dcmp (dot (V1, V2)) < 0) return Length (V2);

else if (dcmp (dot (V1, V3)) > 0) return Length (V3);

else return fabs (cross (V1, V2)) / Length (V1);

}

Point point\_line\_proj(Point p, Point a, Point b) {//点在直线上的投影，两点式

Vector V = b - a;

return a + V \* (dot (V, p - a) / dot (V, V));

}

bool can\_seg\_seg\_inter(Point a1, Point a2, Point b1, Point b2) {//判断线段相交，两点式

double c1 = cross (a2 - a1, b1 - a1), c2 = cross (a2 - a1, b2 - a1),

c3 = cross (b2 - b1, a1 - b1), c4 = cross (b2 - b1, a2 - b1);

return dcmp (c1) \* dcmp (c2) < 0 && dcmp (c3) \* dcmp (c4) < 0;

}

bool can\_line\_seg\_inter(Point a1, Point a2, Point b1, Point b2) {//判断直线与线段相交，两点式

double c1 = cross (a2 - a1, b1 - a1), c2 = cross (a2 - a1, b2 - a1);

return dcmp (c1 \* c2) <= 0;

}

bool on\_seg(Point p, Point a, Point b) {//判断点在线段上，两点式

return dcmp (cross (a - p, b - p)) == 0 && dcmp (dot (a - p, b - p)) < 0;

}

double area\_triangle(Point a, Point b, Point c) {//三角形面积，叉积

return fabs (cross (b - a, c - a)) / 2.0;

}

double area\_poly(vector<Point> ps) {//多边形面积，叉积

double ret = 0;

for (int i=1; i<ps.size ()-1; ++i) {

ret += fabs (cross (ps[i] - ps[0], ps[i+1] - ps[0])) / 2;

}

return ret;

}

/\*

点集凸包，输入点的集合，返回凸包点的集合。

凸包边上无点：<= 凸包边上有点：<

\*/

vector<Point> convex\_hull(vector<Point> ps) {

sort (ps.begin (), ps.end ());//x - y排序

ps.erase (unique (ps.begin (), ps.end ()), ps.end ());//删除重复点

int n = ps.size (), k = 0;

vector<Point> qs (n \* 2);

for (int i=0; i<n; ++i) {

while (k > 1 && cross (qs[k-1] - qs[k-2], ps[i] - qs[k-1]) <= 0) k--;

qs[k++] = ps[i];

}

for (int t=k, i=n-2; i>=0; --i) {

while (k > t && cross (qs[k-1] - qs[k-2], ps[i] - qs[k-1]) <= 0) k--;

qs[k++] = ps[i];

}

qs.resize (k-1);

return qs;

}

struct Circle {

Point c;

double r;

Circle () {}

Circle (Point c, double r) : c (c), r (r) {}

Point point(double a) {

return Point (c.x + cos (a) \* r, c.y + sin (a) \* r);

}

};

struct Line {

Point p;

Vector v;

double r;

Line () {}

Line (const Point &p, const Vector &v) : p (p), v (v) {

r = polar\_angle (v);

}

Point point(double a) {

return p + v \* a;

}

};

/\*

直线与圆相交求交点，返回交点个数，交点保存在P中

\*/

int line\_cir\_inter(Line L, Circle C, double &t1, double &t2, vector<Point> &P) {

double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y - C.c.y;

double e = a \* a + c \* c, f = 2 \* (a \* b + c \* d), g = b \* b + d \* d - C.r \* C.r;

double delta = f \* f - 4 \* e \* g;

if (dcmp (delta) < 0) return 0;

if (dcmp (delta) == 0) {

t1 = t2 = -f / (2 \* e); P.push\_back (L.point (t1));

return 1;

}

t1 = (-f - sqrt (delta)) / (2 \* e);

t2 = (-f + sqrt (delta)) / (2 \* e);

if (t1 > t2) swap (t1, t2);

if (dcmp (t1) > 0 && dcmp (t1 - 1) < 0) P.push\_back (L.point (t1));

if (dcmp (t2) > 0 && dcmp (t2 - 1) < 0) P.push\_back (L.point (t2));

return (int) P.size ();

}

/\*

两圆相交求交点，返回交点个数。交点保存在P中

\*/

int cir\_cir\_inter(Circle C1, Circle C2, vector<Point> &P) {

double d = Length (C1.c - C2.c);

if (dcmp (d) == 0) {

if (dcmp (C1.r - C2.r) == 0) return -1;//两圆重叠

else return 0;

}

if (dcmp (C1.r + C2.r - d) < 0) return 0;

if (dcmp (fabs (C1.r - C2.r) - d) < 0) return 0;

double a = polar\_angle (C2.c - C1.c);

double da = acos ((C1.r \* C1.r + d \* d - C2.r \* C2.r) / (2 \* C1.r \* d)); //C1C2到C1P1的角？

Point p1 = C1.point (a - da), p2 = C2.point (a + da);

P.push\_back (p1);

if (p1 == p2) return 1;

else P.push\_back (p2);

return 2;

}

/\*

过点到圆的切线，返回切线条数，切线保存在V中

\*/

int point\_cir\_tan(Point p, Circle C, Vector \*V) {

Vector u = C.c - p;

double dis = Length (u);

if (dis < C.r) return 0;

else if (dcmp (dis - C.r) == 0) {

V[0] = rotate (u, PI / 2); return 1;

}else {

double ang = asin (C.r / dis);

V[0] = rotate (u, -ang);

V[1] = rotate (u, +ang);

return 0;

}

}

/\*

两圆的公切线，返回公切线条数，切线短点保存在a和b中

\*/

int cir\_cir\_tan(Circle A, Circle B, Point \*a, Point \*b) {

int cnt = 0;

if (A.r < B.r) {

swap (A, B); swap (a, b);

}

double d = dot (A.c - B.c, A.c - B.c);

double rsub = A.r - B.r, rsum = A.r + B.r;

if (dcmp (d - rsub) < 0) return 0;//内含

double base = polar\_angle (B.c - A.c);

if (dcmp (d) == 0 && dcmp (A.r - B.r) == 0) return -1;//两圆重叠

if (dcmp (d - rsub) == 0) {//内切，一条切线

a[cnt] = A.point (base); b[cnt] = B.point (base); cnt++;

return 1;

}

//有外公切线

double ang = acos (rsub / d);

a[cnt] = A.point (base + ang); b[cnt] = B.point (base + ang); cnt++;

a[cnt] = A.point (base - ang); b[cnt] = B.point (base - ang); cnt++;

if (d == rsum) {

a[cnt] = A.point (base); b[cnt] = B.point (base + PI); cnt++;

}else if (dcmp (d - rsum) > 0) {//两条内公切线

double ang2 = acos (rsum / d);

a[cnt] = A.point (base + ang2); b[cnt] = B.point (base + ang2 + PI); cnt++;

a[cnt] = A.point (base - ang2); b[cnt] = B.point (base - ang2 + PI); cnt++;

}

return cnt;

}

/\*

多边形与圆的公共面积，上交红书模板

调用fabs (cir\_poly\_area (ps))，ps为多边形的点集，

需要用到line\_cir\_inter ()函数，圆心在原点(可平移)

\*/

double sector\_area(Point a, Point b, double r) {//三角剖分，求扇形面积

double theta = polar\_angle (a) - polar\_angle (b);

while (dcmp (theta) <= 0) theta += 2 \* PI;

while (theta > 2 \* PI) theta -= 2 \* PI;

theta = min (theta, 2 \* PI - theta);

return r \* r \* theta / 2;

}

double cal(Point a, Point b, double r) {

double t1, t2;

bool ina = dcmp (Length (a) - r) < 0;

bool inb = dcmp (Length (b) - r) < 0;

if (ina && inb) return fabs (cross (a, b)) / 2.0;

vector<Point> p;

int num = line\_cir\_inter (Line (a, b - a), Circle (Point (0, 0), r), t1, t2, p);

if (ina) return sector\_area (b, p[0], r) + fabs (cross (a, p[0])) / 2.0;

if (inb) return sector\_area (p[0], a, r) + fabs (cross (p[0], b)) / 2.0;

if (num == 2) return sector\_area (a, p[0], r) + sector\_area (p[1], b, r) + fabs (cross (p[0], p[1])) / 2.0;

return sector\_area (a, b, r);

}

double cir\_poly\_area(vector<Point> &ps, double r) {

double ret = 0;

for (int i=0; i<ps.size ()-1; ++i) {//多边形最后放入ps[0]起点

int sgn = dcmp (cross (ps[i], ps[i+1]));

if (sgn != 0) {

ret += sgn \* cal (ps[i], ps[i+1], r);

}

}

return ret;

}

//线段相交包括非规范相交

bool can\_seg\_seg\_inter2(Point a1, Point a2, Point b1, Point b2) {

if (min (a1.x, a2.x) > max (b1.x, b2.x) ||

min (a1.y, a2.y) > max (b1.y, b2.y) ||

min (b1.x, b2.x) > max (a1.x, a2.x) ||

min (b1.y, b2.y) > max (a1.y, a2.y)) return false;

double c1 = (a1 - a2) ^ (a1 - b1);//叉积

double c2 = (a1 - a2) ^ (a1 - b2);

double c3 = (b1 - b2) ^ (b1 - a1);

double c4 = (b1 - b2) ^ (b1 - a2);

return dcmp (c1 \* c2) <= 0 && dcmp (c3 \* c4) <= 0;

}

凸包

#include <bits/stdc++.h>

#define fuck(x) cout<<'['<<#x<<' '<<(x)<<"]\n"

using namespace std;

typedef long long LL;

const int INF = 0x3f3f3f3f;

const int mod = 1e9 + 7;

const int MX = 1e6 + 5;

struct lp {

LL x, y;

bool operator<(const lp&b) const {

if(x == b.x) return y < b.y;

return x < b.x;

}

} cw[MX],R[MX],op[5],ok;

LL cross(lp a, lp b, lp c) {

return ((b.x - a.x) \* (c.y - a.y) - (c.x - a.x) \* (b.y - a.y));

}

int convex(int n) {

int tn = 0, k;

sort(cw, cw + n);

for(int i = 0; i < n; i++) {

while(tn > 1 && cross(R[tn - 1], cw[i], R[tn - 2]) <= 0) tn--;

R[tn++] = cw[i];

}

k = tn;

for(int i = n - 2; i >= 0; i--) {

while(tn > k && cross(R[tn - 1], cw[i], R[tn - 2]) <= 0) tn--;

R[tn++] = cw[i];

}

if(n > 1) tn--;

return tn;

}

int main() {

int T, n;

scanf("%d",&T);

while(T--) {

scanf("%d",&n);

for(int i=0;i<n;++i){

scanf("%lld%lld",&cw[i].x,&cw[i].y);

}

n = convex(n);

sort(R, R + n);

for(int i=0;i<n;++i){

printf("%lld %lld\n", R[i].x,R[i].y);

}

}

return 0;

}

[字符串](#_字符串)

[Kmp](#_Kmp)

strcat(s1, "{");

memcpy(s2, s1, sizeof(char)\*(len1+1));

void get\_next(string t){

nex[0] = -1;

for(int i = 0,k = -1;i < lent;){

if(k==-1||t[i] == t[k]){

++k;++i;

nex[i]=k;

}else k = nex[k];

}

}

bool kmp(string s,string t){

lens=s.length();

lent=t.length();

if(lens<lent)return 0;

get\_next(t);

int i = 0, j = 0;

while(i < lens&&j<lent) {

if(j==-1||s[i] == t[j]){

i++;j++;

if(j==lent){

return 1;//return i-j+1;

}

}else j=nex[j];

}

return 0;

}

int len=m-nxt[m];

if(nxt[m]\*2<m) len=m;

//短串的最小循环节len

int ans = 0, t = 0;

for(int i = len; i > 0; ) {

ans += i - nex[i];

i = nex[i];

p[t++] = nex[i];

}

printf("ans = %d\n", ans);///ans 的值和这个字符串的长度一样哦！！！

///只有p数组里存的前缀才是这个字符串的后缀！

[Manacher](#_Manacher)

const int N = 1e6+7;

int n, m;

char ar[N<<1], br[N<<1];

int p[N<<1];

void manacher(){

int Len = strlen(ar), id = 0, ans\_id = 0, right = 0;

memset(p, 0, sizeof(p));

memset(br, 0, sizeof(br));

for(int i = Len; i >= 0; --i){

ar[2 \* i + 1] = '#';

ar[2 \* i + 2] = ar[i];

}

ar[0] = '\*';p[0] = p[1] = 1;

for(int i = 2; i < 2 \* Len + 1; ++i){

if(i < right)p[i] = min(p[2 \* id - i], right - i);

else p[i] = 1;

while(i - p[i] >= 0 && ar[i + p[i]] == ar[i - p[i]])p[i]++;

if(p[ans\_id] < p[i])ans\_id = i;

if(i + p[i] > right){

id =i;

right = i + p[i];

}

}

memcpy(br, ar + id - p[id] + 1, (2 \* p[id] - 1)\*sizeof(char));

id = ans\_id;

printf("%d\n", p[id] - 1);

}

后缀数组

后缀数组(SA[i]存放排名第i大的后缀首字符下标)

名次数组（rank[i]存放suffix(i)的优先级(名次)）

height数组：height[i]是sa[i-1]和sa[i]的最长公共前缀长度

/\*SA,R,H的下标都是 0~n 其中多包括了一个空字符串\*/

struct Suffix\_Array {

static const int N = 3e5 + 7;

int n, len, s[N], M;

int sa[N], rnk[N], height[N];

int tmp\_one[N], tmp\_two[N], c[N];

int dp[N][33];

void init\_str(char \*str);

void build\_sa(int m = 128);

void calc\_height(int n);

void Out(char \*str);

void RMQ\_init(int n);

int RMQ\_query(int l, int r);

int cmp\_suffix(char\* pattern, int p){//判断是否为后缀p的前缀

return strncmp(pattern, s + sa[p], M);

}

int find(char\* P){//Omlog(n)

M = strlen(P);

if(cmp\_suffix(P, 0) < 0) return -1;

if(cmp\_suffix(P, n-1) > 0) return -1;

int L = 0, R = n - 1;

while(R >= L){

int mid = L + (R - L) / 2;

int res = cmp\_suffix(P, mid);

if(!res) return mid;

if(res < 0) R = mid - 1; else L = mid + 1;

}

return -1;

}

}SA;

void Suffix\_Array::Out(char \*str) {

puts ("/\*Suffix\*/");

for (int i=0; i<n; ++i) {

printf ("%s\n", str+sa[i]);

}

}

//LCP(suffix(i), suffix(j))=RMQ\_query(rnk[i], rnk[j]);

int Suffix\_Array::RMQ\_query(int l, int r) {

l = rnk[l]; r = rnk[r];

if (l > r) swap(l, r);

l++;

int k = 0; while (1<<(k+1) <= r - l + 1) k++;

return min(dp[l][k], dp[r-(1<<k)+1][k]);

}

void Suffix\_Array::RMQ\_init(int n) {

for (int i=0; i<n; ++i) dp[i][0] = height[i];

for (int j=1; (1<<j)<=n; ++j) {

for (int i=0; i+(1<<j)-1<n; ++i) {

dp[i][j] = std::min (dp[i][j-1], dp[i+(1<<(j-1))][j-1]);

}

}

}

void Suffix\_Array::init\_str(char \*str) {

len = strlen(str);

n = len + 1;

for (int i=0; i<len; ++i) {

s[i] = str[i] - 'a' + 1;

}

s[len] = '\0';

}

void Suffix\_Array::calc\_height(int n) {

for (int i=0; i<=n; ++i) rnk[sa[i]] = i;

int k = height[0] = 0;

for (int i=0; i<n; ++i) {

if (k) k--;

int j = sa[rnk[i]-1];

while (s[i+k] == s[j+k]) k++;

height[rnk[i]] = k;

}

}

//m = max(r[i]) + 1，一般字符128足够了

void Suffix\_Array::build\_sa(int m) {

int i, j, p, \*x = tmp\_one, \*y = tmp\_two;

for (i=0; i<m; ++i) c[i] = 0;

for (i=0; i<n; ++i) c[x[i]=s[i]]++;

for (i=1; i<m; ++i) c[i] += c[i-1];

for (i=n-1; i>=0; --i) sa[--c[x[i]]] = i;

for (j=1; j<=n; j<<=1) {

for (p=0, i=n-j; i<n; ++i) y[p++] = i;

for (i=0; i<n; ++i) if (sa[i] >= j) y[p++] = sa[i] - j;

for (i=0; i<m; ++i) c[i] = 0;

for (i=0; i<n; ++i) c[x[y[i]]]++;

for (i=1; i<m; ++i) c[i] += c[i-1];

for (i=n-1; i>=0; --i) sa[--c[x[y[i]]]] = y[i];

std::swap (x, y);

for (p=1, x[sa[0]]=0, i=1; i<n; ++i) {

x[sa[i]] = (y[sa[i-1]] == y[sa[i]] && y[sa[i-1]+j] == y[sa[i]+j] ? p - 1 : p++);

}

if(p >= n) break;

m=p;

}

calc\_height(n-1);

RMQ\_init(n);

}

Ac自动机

const int MXN = 1e6 + 6;

const int MXT = 5e5 + 5;

struct AHO {

struct trie {

int nex[26];

int fail, cnt;

void New() {

memset(nex, -1, sizeof(nex));

fail = cnt = 0;

}

}cw[MXT];

int rt, tot;

void init() {

rt = tot = 0;

cw[0].New();

}

void add\_str(char \*S) {

int len = strlen(S);

rt = 0;

for(int i = 0, now; i < len; ++i) {

now = S[i] - 'a';

if(cw[rt].nex[now] == -1) {

cw[rt].nex[now] = ++tot;

cw[tot].New();

}

rt = cw[rt].nex[now];

}

cw[rt].cnt++;

}

void build\_ac(){

queue<int> Q;

Q.push(0);

cw[0].fail = -1;

while(!Q.empty()){

int u = Q.front(); Q.pop();

for(int i = 0, pos; i < 26; ++i) {

pos = cw[u].nex[i];

if(~pos) {

if(u == 0) cw[pos].fail = 0;

else {

int v = cw[u].fail;

while(~v){

if(~cw[v].nex[i]) {

cw[pos].fail = cw[v].nex[i];

break;

}

v = cw[v].fail;

}

if(v == -1) cw[pos].fail = 0;

}

Q.push(pos);

}

}

}

}

int Get(int u) {

int ans = 0;

while(u) {

ans += cw[u].cnt;

cw[u].cnt = 0;

u = cw[u].fail;

}

return ans;

}

int Query(char \*S) {

int len = strlen(S);

int ans = 0;

rt = 0;

for(int i = 0, now, p; i < len; ++i) {

now = S[i] - 'a';

if(~cw[rt].nex[now]) {

rt = cw[rt].nex[now];

}else {

p = cw[rt].fail;

while(p != -1 && cw[p].nex[now] == -1) p = cw[p].fail;

if(p == -1) rt = 0;

else rt = cw[p].nex[now];

}

if(cw[rt].cnt) ans += Get(rt);

}

return ans;

}

}aho;

[Hash](#_Hash)

public long BKDRHash(String str){

long seed = 131;// 31 131 1313 13131 131313 etc..

long hash = 0;

for(int i = 0; i < str.length(); i++){

hash = (hash \* seed) + str.charAt(i);

}

return hash;

}

//最有效的

public long DJBHash(String str){

long hash = 5381;

for(int i = 0; i < str.length(); i++){

hash = ((hash << 5) + hash) + str.charAt(i);

}

return hash;

}

//还不错

public long SDBMHash(String str){

long hash = 0;

for(int i = 0; i < str.length(); i++){

hash = str.charAt(i) + (hash << 6) + (hash << 16) - hash;

}

return hash;

}

//加快散列过程

public long RSHash(String str){

int b = 378551;

int a = 63689;

long hash = 0;

for(int i = 0; i < str.length(); i++){

hash = hash \* a + str.charAt(i);

a = a \* b;

}

return hash;

}

双hash

typedef pair<int,int>pii;typedef pair<LL,LL>pll;

const int mod1 = 1e9+7, mod2 = 1e9+9;

const int MX = 1e5+7;

const int has = 99959;

char str1[MX],str2[MX];

LL P1[MX],P2[MX];

pll H[MX];

unordered\_map<LL,int>mp;

void Hash(){

//预处理出str1的hash值，方便求解子串的hash值

P1[0]=P2[0]=1;

for(int i=1;i<=n;i++){

P1[i]=(P1[i-1]\*has)%mod1;

P2[i]=(P2[i-1]\*has)%mod2;

LL h1=((H[i-1].fi\*has)%mod1+(LL)str1[i])%mod1;

LL h2=((H[i-1].se\*has)%mod2+(LL)str1[i])%mod2;

H[i]=MP(h1,h2);

}

}

LL get\_hash(int l,int r){

//得到子串str1[l,r]的hash值

LL h1=(H[r].fi-(H[l-1].fi\*P1[r-l+1])%mod1+mod1)%mod1;

LL h2=(H[r].se-(H[l-1].se\*P2[r-l+1])%mod2+mod2)%mod2;

return h1+h2;

}

void solve(){

scanf("%s%s",str1+1,str2+1);

Hash();

短串的最小循环节len

pll h=MP(0,0);

for(int i=1;i<=m;i++){

LL h1=((h.fi\*has)%mod1+(LL)str2[i])%mod1;

LL h2=((h.se\*has)%mod2+(LL)str2[i])%mod2;

h=MP(h1,h2);

}

}

hash[x+y+x/y+y/x+x%y+y%x+x|y]

uLL base = 13131,hs[MX],pw[MX];

unordered\_map<uLL,int>Hash;

inline uLL calc(int l,int r){

return hs[r]-pw[r-l+1]\*hs[l-1];

}

inline void init(int len){

pw[0]=1;

for(int i=1;i<=len;++i){

pw[i]=pw[i-1]\*base;

hs[i]=hs[i-1]\*base+(s[i]-'a');

}

}

struct Hash\_map{

static const int mask=0x7fffff;

int p[mask+1],q[mask+1];

void clear(){

memset(q,0,sizeof(q));

}

int& operator [](int k){

int i;

for(i=k&mask;q[i]&&p[i]!=k;i=(i+1)&mask);

p[i]=k;

return q[i];

}

}Hash;

[DP](#_DP)

[数位DP](#_数位DP)

const int N = 1e4 + 7;

const int INF = 0x3f3f3f3f;

LL n, m;

int ar[N];

LL dp[19][19][3000];

LL dfs(int pos,int mid,int cur,bool limit){

if(pos==-1) return cur==0;

if(cur<0) return 0;

if(!limit&&dp[pos][mid][cur]!=-1)return dp[pos][mid][cur];

int up = limit? ar[pos]: 9;

LL sum = 0;

for(int i = 0; i <= up; ++i){

int tmp = (pos-mid)\*i;

sum += dfs(pos-1,mid,cur+tmp,limit&&i==ar[pos]);

}

if(!limit) dp[pos][mid][cur] = sum;

return sum;

}

LL solve(LL x){

if(x<0) return 0;

else if(x==0) return 1;

int pos = 0;

while(x){

ar[pos++] = x % 10;

x /= 10;

}

LL sum = 0;

for(int i = pos-1; i >= 0; --i){

sum += dfs(pos-1, i, 0, 1);

}

return sum-(pos-1);

}

void solve(){

memset(dp, -1, sizeof(dp));

scanf("%lld%lld", &n, &m);

printf("%lld\n", solve(m)-solve(n-1));

}

其他东西

while(l <= r){

mid = (l + r)>>1;

if(ok(mid))r = mid - 1, ans = mid;

else l = mid + 1, ans = mid + 1;

}

while(r > l ){

midl = (l + r)/2;

midr = (midl + r + 1)/2;

midl = l + (r - l)/3;

midr = r - (r - l)/3;

if(get(midl)?get(midr)) r = midr-1;

else l = midl+1;

}

[部分头文件](#_祖传头文件_1)

//#include<bits/stdc++.h>修改版

#include<algorithm>

#include<vector>

#include<cctype>

#include<bitset>

#include<cassert>

#include<ctime>

#include<iomanip>

#include<memory>

#include<utility>

#include<fstream>

#include<unordered\_map>

#include <tr1/unordered\_set>

#define haah(i) \_\_builtin\_popcount(i)

#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

#define mme(a,b) memset((a),(b),sizeof((a)))

#define precision(x,d) cout<<fixed<<setprecision(d)<<x<<"\n"

#define iis std::ios::sync\_with\_stdio(false)

using namespace std;

#pragma optimize("-O3")

const int X = 999983;

const LL INFLL = 0x3f3f3f3f3f3f3f3fll;

const int xiao = 0x80808080;

const int MOD = 1e9 + 7;//1e15+37

int myrand() { return rand() % 1000 \* 1000000 + rand() % 1000 \* 1000 + rand() % 1001; }

int some\_primes[]={24443, 100271, 1000003,199373,149627, 1000333, 5000321, 98765431,998244353};

const long long HMOD[] = {2078526727, 2117566807};

const long long BASE[] = {1572872831, 1971536491,19260817,99959};

unordered\_map<int, int> mp;int main(){

#ifndef ONLINE\_JUDGE

freopen("E://ADpan//in.in", "r", stdin);

freopen("E://ADpan//out.out", "w", stdout);

#endif

return 0;

}

[读入挂](#_读入挂)

template<typename T>

inline T read(T&x){

x=0;int f=0;char ch=getchar();

while (ch<'0'||ch>'9') f|=(ch=='-'),ch=getchar();

while (ch>='0'&&ch<='9') x=(x<<3)+(x<<1)+ch-'0',ch=getchar();

return x=f?-x:x;

}

template<typename T>

inline void write(T x){

if(x==0){ putchar('0'),putchar('\n');return;}

if(x<0){ putchar('-'),x = -x; }

static char s[22];int l = 0;

while(x!=0) s[l++] = x%10+48,x /= 10;

while(l)putchar(s[--l]);

putchar('\n');

}

for(int i=1; i<=m; i++) {

memset(tools,0,sizeof(tools));

int ulen=0,tool;

cin.getline(tools,10000);

while(sscanf(tools+ulen,"%d",&tool)==1){

v[i].push\_back(tool);

if(tool==0)ulen++;

else {

while(tool){

tool/=10;

ulen++;

}

}

ulen++;

}

}

struct FastIO { //1e7的char占用内存11000kb

static const int S = 1e7;

int wpos;

char wbuf[S];

FastIO() : wpos(0) {}

inline int xchar() {

static char buf[S];

static int len = 0, pos = 0;

if (pos == len)

pos = 0, len = fread(buf, 1, S, stdin);

if (pos == len) exit(0);

return buf[pos++];

}

inline int xuint() {

int c = xchar(), x = 0;

while (c <= 32) c = xchar();

for (; '0' <= c && c <= '9'; c = xchar()) x = x \* 10 + c - '0';

return x;

}

inline int xint() {

int s = 1, c = xchar(), x = 0;

while (c <= 32) c = xchar();

if (c == '-') s = -1, c = xchar();

for (; '0' <= c && c <= '9'; c = xchar()) x = x \* 10 + c - '0';

return x \* s;

}

inline void xstring(char \*s) {

int c = xchar();

while (c <= 32) c = xchar();

for (; c > 32; c = xchar()) \* s++ = c;

\*s = 0;

}

inline void wchar(int x) {

if (wpos == S) fwrite(wbuf, 1, S, stdout), wpos = 0;

wbuf[wpos++] = x;

}

inline void wint(LL x) {

if (x < 0) wchar('-'), x = -x;

char s[24];

int n = 0;

while (x || !n) s[n++] = '0' + x % 10, x /= 10;

while (n--) wchar(s[n]);

wchar('\n');

}

inline void wstring(const char \*s) {

while (\*s) wchar(\*s++);

}

~FastIO() {

if (wpos) fwrite(wbuf, 1, wpos, stdout), wpos = 0;

}

} io;

[JAVA](#_JAVA)

import java.io.\*;

import java.text.DecimalFormat;

import java.util.\*;

import java.math.\*;

import java.util.Arrays;

public class hhd{

static int N = (int)1e5+7;

static PrintStream putOut = System.out;//可以换一种写法~

public static void main(String[] args){

Scanner cin = new Scanner(System.in);//读入

StringBuilder sb;//String与StringBuilder

String tmp = new String();

tmp = cin.nextLine();

sb = new StringBuilder(tmp);

sb.setCharAt(1,'3');//首位是第0位，改变第1位为'3'

tmp = sb.toString();

System.out.println(tmp+" "+sb);

BigInteger bigNum = new BigInteger(tmp);

System.out.println(bigNum.multiply(BigInteger.valueOf(2)));//大整数运算

String a = bigNum.toString();//两种进制转化的方法

//a = new BigInteger(a,10).toString(2);

a=bigNum.toString(2);

System.out.println(a);

int AA = 199;//控制输出格式

double b = 9.9999;

System.out.printf("%04d %.5f\n",AA,b);

System.out.format("%04d %.5f\n",AA,b);

System.out.printf("%.3f\n",(double)AA);

DecimalFormat p3 = new DecimalFormat("#.000#");

System.out.println(p3.format(AA));

bigNum = cin.nextBigInteger();

System.out.printf("%d\n",bigNum);

String str;//进制转化

int A=8,B=2;

str = Integer.toString(A,2);//把int型数据转换成X进制数并转换成String型

System.out.println(str);

A = Integer.parseInt(str,2);//把字符串当作X进制数转换成10进制int型

System.out.println(A);

str = "aryawu";//String与字符串

char[] ch = str.toCharArray();

System.out.println(ch.length+str.substring(2,4));

int temp = -1;//位运算

System.out.println((temp>>1)+" "+(temp>>>1));

System.out.println(Integer.toString(31,16));

//数组的一些操作

int[] ar = {2,5,9,3,34,1,2};

boolean[] vis = new boolean[100];

boolean[] Vis = new boolean[100];

Arrays.sort(ar);

Arrays.fill(vis,true);

putOut.println("相同？："+Arrays.equals(vis,Vis));

putOut.println(Arrays.binarySearch(ar,5));

//数据结构

Map map=new HashMap();//以下的俩种使用方式都是可以的

map.put("a",1);

map.put(11,"abc");

Map<String,Integer> mp=new HashMap<String,Integer>();

mp.put("a",1);mp.put("as",2);mp.put("b",3);

mp.put("a",5);

mp.remove("a");

mp.get("as");

//获取所有的key和value

for(Map.Entry<String, Integer> entry : mp.entrySet()){

System.out.println("key = " + entry.getKey() + ", value = " + entry.getValue());

}

//获取所有的key

Set<String> keys = mp.keySet();

for(String key: keys){

System.out.println(key);

}

//获取所有的value

Collection<Integer> values = mp.values();

for(Integer value: values){

System.out.println(value);

}

putOut.println("===============");

Queue<Integer> que = new LinkedList<Integer>();

/\*

\* 建议使用offer而不使用add

\* 建议使用poll而不使用remove

\* 建议使用peek而不使用element

\* 因为前者不会抛出异常，会有返回值

\* \*/

que.offer(5);que.offer(2);

putOut.println(que.poll());

que.peek();

for(int i:que){//遍历队列

putOut.println(i);

}

putOut.println("===============");

Stack<Integer>st = new Stack<>();

st.push(12);st.push(23);

st.peek();

st.pop();

st.empty();

putOut.println(st.search(23));

putOut.println(st.search(12));

//数组和List的转换

List list = new ArrayList();

list.add("a");

list.add("12");

int len = list.size();

String[] arr\_tmp = new String[len+1];

for(int i=0;i<len;++i){

arr\_tmp[i]=(String)list.get(i);

//arr\_tmp[i]=list.get(i).toString();

}

List<String> list1 = new ArrayList<String>();

list1.add("a");

list1.add("12");

String[] arr\_tmp1 = (String[])list1.toArray(new String[len]);

List<String>list2 = Arrays.asList(arr\_tmp);

for(int i=0;i<list2.size();++i){

putOut.println(list2.get(i));

}

List<String> list3 = Arrays.asList("12","34","56");

for(int i=0;i<list3.size();i++){

System.out.println(list3.get(i));

}

while (cin.hasNext()){//多组输入

int n = cin.nextInt();

System.out.println(Math.pow(n,4));

}

}

}