目录

[目录 1](#_Toc369292332)

[一、 图论 2](#_Toc369292333)

[1) 强连通分量缩点tarjan 2](#_Toc369292334)

[2) 点双连通分量缩点tarjan 3](#_Toc369292335)

[3) 最大流 5](#_Toc369292336)

[4) KM算法 8](#_Toc369292337)

[5) 最小费用最大流 10](#_Toc369292338)

[6) 一般图匹配树开花算法 13](#_Toc369292339)

[二、 数据结构 15](#_Toc369292340)

[1) 划分树 15](#_Toc369292341)

[2) 左偏树 17](#_Toc369292342)

[3) Splay树 18](#_Toc369292343)

[4) KD树 22](#_Toc369292344)

[5) DLX——精确覆盖 23](#_Toc369292345)

[6) DLX——重复覆盖 28](#_Toc369292346)

[7) 数独 32](#_Toc369292347)

[8) RMQ 32](#_Toc369292348)

[三、 字符串 33](#_Toc369292349)

[1) KMP算法 33](#_Toc369292350)

[2) AC自动机 34](#_Toc369292351)

[3) 后缀自动机 36](#_Toc369292352)

[4) 后缀数组\* 38](#_Toc369292353)

[5) 回文串Manacher算法 38](#_Toc369292354)

[6) 最小表示法 39](#_Toc369292355)

[7) 最小覆盖子串 40](#_Toc369292356)

[四、 树相关 40](#_Toc369292357)

[1) LCA转RMQ 40](#_Toc369292358)

[2) LCA 41](#_Toc369292359)

[3) 树的重心 41](#_Toc369292360)

[4) 树链剖分 41](#_Toc369292361)

[5) LCT动态树 44](#_Toc369292362)

[五、 可持久化数据结构 44](#_Toc369292363)

[1) 可持久化线段树 44](#_Toc369292364)

[六、 几何 45](#_Toc369292365)

[1) 二维point类 45](#_Toc369292366)

[2) 扫描线\* 45](#_Toc369292367)

## 图论

### 强连通分量缩点tarjan

#define maxn 1005

struct node {

int v,next;

};

node e[maxn \* maxn];

int cnt ;

int head[maxn] , dfn[maxn] , low[maxn] , col[maxn];

bool instack[maxn];

int ind[maxn] , oud[maxn] , st[maxn] ;

int n , m , top, dep , scnt , dep ;

void init() {

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

cnt = 0 ;

}

void add(int u,int v) {

e[cnt].v = v;

e[cnt].next = head[u];

head[u] = cnt ++;

}

void tarjan(int u) {

int v;

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

st[++top]=u;

instack[u]=1;

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

v=e[i].v;

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

tarjan(v);

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

} else if(instack[v]) low[u]=min(low[u],dfn[v]);

}

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

int x ;

do {

x=st[top--];

instack[x]=0;

col[x]=scnt;

} while (x!=u);

scnt++;

}

}

void base() {

scnt = 0;

top = 0;

dep = 0 ;

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

memset(instack,0,sizeof(instack));

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

if (dfn[i] == -1) {

tarjan(i);

}

}

/\*

初始化init();

加边add，cnt为边数，从0开始

调用base缩点，求的col为每个点所在的强连通编号

scnt为强连通数目，从0开始，

\*/

### 点双连通分量缩点tarjan

#### 处理重边部分

void tarjan(int u ) {

int v ;

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

vis[u] = 1;

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

if(!vst[i>>1]) {

// printf("i>>1 = %d\n",i>>1);

vst[i >> 1] = 1;

v = e[i].v;

// printf("v = %d\n",v);

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

tarjan(v);

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

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

ans = min(ans , e[i].w);

}

} else if(vis[v] == 1) low[u] = min(low[u] , dfn[v]);

}

vis[u] = 2 ;

}

#### 完整部分

#define maxn 2000

struct node

{

int v,next;

bool isbri;

};

node e[maxn \* 4];

int head[maxn];

int cnt , dep , n , m ;

int low[maxn],dfn[maxn];

int vis[maxn];

bool vst[maxn \* 2]; // 边有没有访问过

int col[maxn],scnt;

int num[maxn],deg[maxn];

void init()

{

cnt = 0;

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

}

void add(int u , int v )

{

e[cnt].v = v;

e[cnt].isbri = false;

e[cnt].next = head[u] ;

head[u] = cnt ++ ;

e[cnt].v = u;

e[cnt].isbri = false;

e[cnt].next = head[v];

head[v] = cnt ++ ;

}

void tarjan(int u )

{

int v;

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

vis[u] = 1;

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

if(!vst[i >> 1 ])

{

vst[i>>1] = 1;

v = e[i].v;

if(dfn[v] == -1 )

{

tarjan(v);

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

if(dfn[u] < low[v] )

{

e[i].isbri = 1 ;

e[i^1].isbri = 1;

}

}

else if(vis[v] == 1 ) low[u] = min(low[u] , dfn[v]);

}

vis[u] = 2;

}

void dfs(int u )

{

col[u] = scnt;

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

if(e[i].isbri == false && col[e[i].v] == -1)

dfs(e[i].v);

}

void base()

{

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

memset(vst,0,sizeof(vst));

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

dep = 0 ;

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

if(dfn[i] == -1)

tarjan(i);

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

scnt = 0 ;

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

{

if(col[i] == -1)

{

scnt ++ ;

dfs(i);

}

}

//printf("scnt = %d\n",scnt);

memset(deg,0,sizeof(deg));

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

{

for(int j = head[i] ; j != -1 ; j =e[j].next)

{

if(e[j].isbri )

{

//jianbian

deg[col[i]] ++ ;

deg[col[e[j].v]] ++ ;

}

}

}

}

/\*

初始化 init();

加边 add() , cnt从0开始为边数

scnt为边双连通分两个数，从1开始

num为每个bcc中的点的个数

deg为每个bcc的度数

.isbri表示该边是不桥

可以处理重边

\*/

### 最大流

#### sap邻接表

#define inf 0x3f3f3f3f

#define maxn 3000

struct node {

int v,next,cap;

};

node e[maxn \* 300 \* 2];

int head[maxn];

int cnt ;

int s,t,nn;

int n ;

void init() {

cnt = 0 ;

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

}

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

e[cnt].v=v;

e[cnt].cap=w;

e[cnt].next=head[u];

head[u]=cnt++;

e[cnt].v=u;

e[cnt].cap=0;

e[cnt].next=head[v];

head[v]=cnt++;

}

int pre[maxn],cur[maxn],dis[maxn],gap[maxn];

int sap() {

int flow,aug,u;

int flag;

flow=0;

aug=inf;

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

cur[i]=head[i];

gap[i]=0;

dis[i]=0;

}

gap[s]=nn;

u=s;

pre[s]=s;

while(dis[s]<nn) {

flag=0;

for(int &j=cur[u]; j!=-1; j=e[j].next) {

//cur[u]=j;

int v=e[j].v;

if(e[j].cap>0&&dis[u]==dis[v]+1) {

flag=1;

if(e[j].cap<aug)aug=e[j].cap;

pre[v]=u;

u=v;

if (u==t) {

flow+=aug;

while(u!=s) {

u=pre[u];

e[cur[u]].cap-=aug;

e[cur[u]^1].cap+=aug;

//why?解释偶数异或1为偶数+1，奇数异或1为奇数-1，

//显然我们存的边是从0开始存的，

//所以偶数,偶数+1是残量网格中的两条边(无向边)

}

aug=inf;

}

break;

}

}

if (flag) continue;

int mindis=nn;

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

int v=e[j].v;

if (e[j].cap>0&&dis[v]<mindis) {

mindis=dis[v];

cur[u]=j;

}

}

if (--gap[dis[u]]==0) { //间隙优化

break;

}

dis[u]=mindis+1;

gap[dis[u]]++;

u=pre[u];

}

return flow;

}

#### dinic

#define MAXN 100

#define INF 0x3fffffff

struct edge {

int to,cap,rev;

};

vector<edge> G[MAXN];

int level[MAXN];

int iter[MAXN];

void add\_edge(int from,int to,int cap) {

G[from].push\_back((edge) {

to,cap,G[to].size()

});

G[to].push\_back((edge) {

from,0,G[from].size() - 1

});

}

void bfs(int s) {

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

queue<int> que;

level[s] = 0;

que.push(s);

while(!que.empty()) {

int v = que.front();

que.pop();

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

edge &e = G[v][i];

if(e.cap > 0 && level[e.to] < 0) {

level[e.to] = level[v] + 1;

que.push(e.to);

}

}

}

}

int dfs(int v,int t,int f) {

if(v == t)

return f;

for(int i = iter[v]; i < G[v].size(); i++) {

edge &e = G[v][i];

if(e.cap > 0 && level[v] < level[e.to]) {

int d = dfs(e.to,t,min(f,e.cap));

if(d > 0) {

e.cap -= d;

G[e.to][e.rev].cap += d;

return d;

}

}

}

return 0;

}

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

int flow = 0;

for(;;) {

bfs(s);

if(level[t] < 0)

return flow;

memset(iter,0,sizeof(iter));

int f;

while((f = dfs(s,t,INF)) > 0) {

flow += f;

}

}

}

### KM算法

#### 模板1

#define M 310

#define inf 0x3f3f3f3f

int n,nx,ny;

int Link[M],lx[M],ly[M],slack[M]; //lx,ly为顶标，nx,ny分别为x点集y点集的个数

int visx[M],visy[M],w[M][M];

int DFS(int x) {

visx[x] = 1;

for (int y = 1; y <= ny; y ++) {

if (visy[y])

continue;

int t = lx[x] + ly[y] - w[x][y];

if (t == 0) { //

visy[y] = 1;

if (Link[y] == -1||DFS(Link[y])) {

Link[y] = x;

return 1;

}

} else if (slack[y] > t) //不在相等子图中slack 取最小的

slack[y] = t;

}

return 0;

}

int KM() {

int i,j;

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

memset (ly,0,sizeof(ly));

for (i = 1; i <= nx; i ++) //lx初始化为与它关联边中最大的

for (j = 1,lx[i] = -inf; j <= ny; j ++)

if (w[i][j] > lx[i])

lx[i] = w[i][j];

for (int x = 1; x <= nx; x ++) {

for (i = 1; i <= ny; i ++)

slack[i] = inf;

while (1) {

memset (visx,0,sizeof(visx));

memset (visy,0,sizeof(visy));

if (DFS(x)) //若成功（找到了增广轨），则该点增广完成，进入下一个点的增广

break; //若失败（没有找到增广轨），则需要改变一些点的标号，使得图中可行边的数量增加。//方法为：将所有在增广轨中（就是在增广过程中遍历到）的X方点的标号全部减去一个常数d， //所有在增广轨中的Y方点的标号全部加上一个常数d

int d = inf;

for (i = 1; i <= ny; i ++)

if (!visy[i]&&d > slack[i])

d = slack[i];

for (i = 1; i <= nx; i ++)

if (visx[i])

lx[i] -= d;

for (i = 1; i <= ny; i ++) //修改顶标后，要把所有不在交错树中的Y顶点的slack值都减去d

if (visy[i])

ly[i] += d;

else

slack[i] -= d;

}

}

int res = 0;

for (i = 1; i <= ny; i ++)

if (Link[i] > -1)

res += w[Link[i]][i];

return res;

}

int main () {

int i,j;

while (scanf ("%d",&n)!=EOF) {

nx = ny = n;

// memset (w,0,sizeof(w));

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

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

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

int ans = KM();

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

}

return 0;

}

#### 模板2

int nx,ny;

int n,m,k,slack;

int linky[maxn],lx[maxn],ly[maxn],w[maxn][maxn];

bool visx[maxn],visy[maxn];

bool find(int x) {

visx[x]=true;

for(int y=1; y<=ny; y++) {

if(visy[y]) continue;

int t=lx[x]+ly[y]-w[x][y];

if(t==0) {

visy[y]=true;

if(linky[y]==-1 || find(linky[y])) {

linky[y]=x;

return true;

}

} else if(slack > t)

slack = t;

}

return false;

}

int KM() {

int i,x,y;

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

memset(lx,0,sizeof(lx));

memset(ly,0,sizeof(ly));

for(x=1; x<=nx; x++)

for(y=1; y<=ny; y++)

if(w[x][y]>lx[x]) lx[x]=w[x][y];

int old;

for(x=1; x<=nx; x++)

while(1 == 1) {

memset(visx,0,sizeof(visx));

memset(visy,0,sizeof(visy));

slack=inf;

if(find(x)) break;

for(i=1; i<=nx; i++)

if(visx[i]) lx[i] -= slack;

for(i=1; i<=ny; i++)

if(visy[i]) ly[i] += slack;

}

int ret=0;

for(y=1; y<=ny; y++)

if(linky[y]!=-1)

ret += w[linky[y]][y];

return ret;

}

/// 运用注意，把小的点集放在x上，大的点集放在y上

/// 否则会死循环&wa

### 最小费用最大流

#### 常用模板

#define maxn 1005

const int inf = 0x3f3f3f3f;

int n,s,t;

int nn;

bool vis[maxn];

bool mmm;

int pre[maxn];

struct node {

int v,next,cap,cost;

} e[maxn \* maxn];

int head[maxn],cnt;

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

e[cnt].v=v;

e[cnt].cap=cap;

e[cnt].cost=cost;

e[cnt].next=head[u];

head[u]=cnt++;

e[cnt].v=u;

e[cnt].cap=0;

e[cnt].cost=-cost;

e[cnt].next=head[v];

head[v]=cnt++;

}

void init() {

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

cnt = 0 ;

}

queue<int> q;

int dist[maxn];

bool spfa() {

memset(dist,inf,sizeof(dist));

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

while(!q.empty()) q.pop();

q.push(s);

vis[s]=1;

dist[s]=0;

while(!q.empty()) {

int u=q.front();

q.pop();

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

int v=e[i].v;

if(dist[v]>dist[u]+e[i].cost&&e[i].cap>0) {

dist[v]=dist[u]+e[i].cost;

pre[v]=i;

if(!vis[v]) {

q.push(v);

vis[v]=1;

}

}

}

vis[u]=0;

}

if(dist[t]<0) return true;

else return false;

}

int ans ; // 最小费用

int ans1 ; // 最大流

void mincost() {

int i;

i=t;

ans1 ++ ;

while(i!=s) {

ans+=e[pre[i]].cost;

e[pre[i]].cap--;

if(e[pre[i]].cap==0) mmm=true;

e[pre[i]^1].cap++;

if(e[pre[i]^1].cap==1) mmm=true;

i=e[pre[i]^1].v;

}

}

int sum;

void work() {

ans=0;

ans1 = 0 ;

mmm=true;

while(!mmm||spfa()) {

mmm=false;

mincost();

}

printf("ans1 = %d\n",ans1);

sum = -ans ;

}

/// 建图: 加边add，初始化 s 超级源点 t 超级汇点 节点个数

/// 让s为0 ， t为最后一个节点，nn = t + 1 ;

/// 加边初始化init();

/// ans 求出最小费用

/// ans1 为流量

int main() {

return 0 ;

}

#### 模板2

struct edge {

int x,y,f,c,nxt;

} e[E];

int en,head[N],pre[N],dist[N],visited[N];

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

e[en].x=x;

e[en].y=y;

e[en].f=f;

e[en].c=c;

e[en].nxt=head[x];

head[x]=en++;

}

void addedge(int x,int y,int f,int c) {

add(x,y,f,c);

add(y,x,0,-c);

}

bool spfa(int s,int t,int n) {

int i,v,cur;

queue<int> q;

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

pre[i]=-1;

visited[i]=false;

dist[i]=inf;

}

visited[s]=true;

dist[s]=0;

q.push(s);

while(!q.empty()) {

cur=q.front();

q.pop();

visited[cur]=false;

for(i=head[cur]; i!=-1; i=e[i].nxt)

if(e[i].f) {

v=e[i].y;

if(dist[v]>dist[cur]+e[i].c) {

dist[v]=dist[cur]+e[i].c;

pre[v]=i;

if(!visited[v]) {

visited[v]=true;

q.push(v);

}

}

}

}

return dist[t]!=inf;

}

int cost(int s,int t,int n) {

int i,flow=inf,ans=0;

while(spfa(s,t,n)) {

ans+=dist[t];

for(i=pre[t]; i!=-1; i=pre[e[i].x])

flow=min(flow,e[i].f);

for(i=pre[t]; i!=-1; i=pre[e[i].x]) {

e[i].f -= flow;

e[i^1].f+=flow;

}

}

return ans;

}

### 一般图匹配树开花算法

#### 模板1

#define MAXN 505

deque<int> Q;

//g[i][j]存放关系图：i,j是否有边,match[i]存放i所匹配的点

bool g[MAXN][MAXN],inque[MAXN],inblossom[MAXN];

int match[MAXN],pre[MAXN],base[MAXN];

//找公共祖先

int findancestor(int u,int v) {

bool inpath[MAXN]= {false};

while(1) {

u=base[u];

inpath[u]=true;

if(match[u]==-1)break;

u=pre[match[u]];

}

while(1) {

v=base[v];

if(inpath[v])return v;

v=pre[match[v]];

}

}

//压缩花

void reset(int u,int anc) {

while(u!=anc) {

int v=match[u];

inblossom[base[u]]=1;

inblossom[base[v]]=1;

v=pre[v];

if(base[v]!=anc)pre[v]=match[u];

u=v;

}

}

void contract(int u,int v,int n) {

int anc=findancestor(u,v);

clr(inblossom,0);

reset(u,anc);

reset(v,anc);

if(base[u]!=anc)pre[u]=v;

if(base[v]!=anc)pre[v]=u;

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

if(inblossom[base[i]]) {

base[i]=anc;

if(!inque[i]) {

Q.push\_back(i);

inque[i]=1;

}

}

}

bool dfs(int S,int n) {

for(int i=0; i<=n; i++)pre[i]=-1,inque[i]=0,base[i]=i;

Q.clear();

Q.push\_back(S);

inque[S]=1;

while(!Q.empty()) {

int u=Q.front();

Q.pop\_front();

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

if(g[u][v]&&base[v]!=base[u]&&match[u]!=v) {

if(v==S||(match[v]!=-1&&pre[match[v]]!=-1))contract(u,v,n);

else if(pre[v]==-1) {

pre[v]=u;

if(match[v]!=-1)Q.push\_back(match[v]),inque[match[v]]=1;

else {

u=v;

while(u!=-1) {

v=pre[u];

int w=match[v];

match[u]=v;

match[v]=u;

u=w;

}

return true;

}

}

}

}

}

return false;

}

int solve(int n) {

clr(match,-1);

int ans=0;

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

if(match[i]==-1&&dfs(i,n))

ans++;

return ans;

}

int main() {

//freopen("in.txt","r",stdin);

int n;

int u,v;

scanf("%d",&n);

memset(g,0,sizeof(g));

while(scanf("%d%d",&u,&v)!=EOF) {

g[u][v] = g[v][u] = 1;

}

int ans = solve(n);

printf("%d\n",ans \* 2 );

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

if(match[i] != -1) {

printf("%d %d\n",i,match[i]);

match[match[i]] = -1;

match[i] = -1;

}

}

return 0;

}

## 数据结构

### 划分树

#### 模板1

#define M 100001

struct Seg\_Tree {

int left,right;

int mid() {

return (left + right) >> 1;

}

} tt[M\*4];

int len;

int sorted[M];

int toLeft[20][M];

int val[20][M];

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

tt[idx].left = l;

tt[idx].right = r;

if(tt[idx].left == tt[idx].right) return ;

int mid = tt[idx].mid();

int lsame = mid - l + 1;//lsame表示和val\_mid相等且分到左边的

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

if(val[d][i] < sorted[mid]) {

lsame --;//先假设左边的数(mid - l + 1)个都等于val\_mid,然后把实际上小于val\_mid的减去

}

}

int lpos = l;

int rpos = mid+1;

int same = 0;

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

if(i == l) {

toLeft[d][i] = 0;//toLeft[i]表示[ tt[idx].left , i ]区域里有多少个数分到左边

} else {

toLeft[d][i] = toLeft[d][i-1];

}

if(val[d][i] < sorted[mid]) {

toLeft[d][i] ++;

val[d+1][lpos++] = val[d][i];

} else if(val[d][i] > sorted[mid]) {

val[d+1][rpos++] = val[d][i];

} else {

if(same < lsame) {//有lsame的数是分到左边的

same ++;

toLeft[d][i] ++;

val[d+1][lpos++] = val[d][i];

} else {

val[d+1][rpos++] = val[d][i];

}

}

}

build(l,mid,d+1,LL(idx));

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

}

int query(int l,int r,int k,int d,int idx) {

if(l == r) {

return val[d][l];

}

int s;//s表示[ l , r ]有多少个分到左边

int ss;//ss表示 [tt[idx].left , l-1 ]有多少个分到左边

if(l == tt[idx].left) {

s = toLeft[d][r];

ss = 0;

} else {

s = toLeft[d][r] - toLeft[d][l-1];

ss = toLeft[d][l-1];

}

if(s >= k) {//有多于k个分到左边,显然去左儿子区间找第k个

int newl = tt[idx].left + ss;

int newr = tt[idx].left + ss + s - 1;//计算出新的映射区间

return query(newl,newr,k,d+1,LL(idx));

} else {

int mid = tt[idx].mid();

int bb = l - tt[idx].left - ss;//bb表示 [tt[idx].left , l-1 ]有多少个分到右边

int b = r - l + 1 - s;//b表示 [l , r]有多少个分到右边

int newl = mid + bb + 1;

int newr = mid + bb + b;

return query(newl,newr,k-s,d+1,RR(idx));

}

}

#### 模板2

#define MAXN 100010

int len;

int sorted[MAXN];

int toLeft[20][MAXN],val[20][MAXN];

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

if(l==r)return;

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

int lsame=m-l+1;//lsame表示和val\_mid相等且分到左边的

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

if(val[d][i]<sorted[m])

lsame--;////先假设左边的数(mid - l + 1)个都等于val\_mid,然后把实际上小于val\_mid的减去

int lpos=l;

int rpos=m+1;

int same=0;

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

if(i==l) {

toLeft[d][i]=0;//toLeft[i]表示[ l , i ]区域里有多少个数分到左边

} else {

toLeft[d][i]=toLeft[d][i-1];

}

if(val[d][i]<sorted[m]) {

toLeft[d][i]++;

val[d+1][lpos++]=val[d][i];

} else if(val[d][i]>sorted[m]) {

val[d+1][rpos++]=val[d][i];

} else {

if(same<lsame) { //有lsame的数是分到左边的

same++;

toLeft[d][i]++;

val[d+1][lpos++]=val[d][i];

} else {

val[d+1][rpos++]=val[d][i];

}

}

}

build(d+1,l,m);

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

}

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

if(l==r) return val[d][l];

int s;//s表示[ L , R ]有多少个分到左边

int ss;//ss表示 [l , L-1 ]有多少个分到左边

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

if(L==l) {

s=toLeft[d][R];

ss=0;

} else {

s=toLeft[d][R]-toLeft[d][L-1];

ss=toLeft[d][L-1];

}

if(s>=k)//有多于k个分到左边,显然去左儿子区间找第k个

return query(l+ss,l+ss+s-1,k,d+1,l,m);

else

return query(m+L-l-ss+1,m-l-ss+R-s+1,k-s,d+1,m+1,r);

}

### 左偏树

struct tree {

int l, r, v, dis, f;

} heap[maxn];

int merge( int a, int b ) {

if( a == 0 ) return b;

if( b == 0 ) return a;

if( heap[a].v < heap[b].v ) swap( a, b );

heap[a].r = merge( heap[a].r, b );

heap[heap[a].r].f = a;

if( heap[heap[a].l].dis < heap[heap[a].r].dis ) swap( heap[a].l, heap[a].r );

if( heap[a].r == 0 ) heap[a].dis = 0;

else heap[a].dis = heap[heap[a].r].dis + 1;

return a;

}

int pop( int a ) {

int l = heap[a].l;

int r = heap[a].r;

heap[l].f = l;

heap[r].f = r;

heap[a].l = heap[a].r = heap[a].dis = 0;

return merge(l, r);

}

int find( int a ) {

return heap[a].f == a ? a : find( heap[a].f ) ;

}

/\*

支持合并，插入，删除最小值三个操作。

后两个操作都可以看成第一个操作的拓展，

如删除最小值是合并根的两棵子树，插入则直接将元素看作一个左偏树—

所以只要写个Merge就可以了！

性质:

节点的键值小于或等于它的左右子节点的键值。

节点的左子节点的距离不小于右子节点的距离。

节点的左子节点右子节点也是一颗左偏树。

\*/

### Splay树

#define inf 0x3f3f3f3f

#define keyTree (ch[ ch[root][1] ][0])

const int maxn = 222222;

int \_min;

int le;

struct SplayTree {

int sz[maxn];

int ch[maxn][2];

int pre[maxn];

int root , top1 , top2;

int ss[maxn] , que[maxn];

void Rotate(int x,int f) {

int y = pre[x];

push\_down(y);

push\_down(x);

ch[y][!f] = ch[x][f];

pre[ ch[x][f] ] = y;

pre[x] = pre[y];

if(pre[x]) ch[ pre[y] ][ ch[pre[y]][1] == y ] = x;

ch[x][f] = y;

pre[y] = x;

push\_up(y);

}

void Splay(int x,int goal) {

push\_down(x);

//puts("zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz");

while(pre[x] != goal) {

int y = pre[x], z = pre[y];

push\_down(z); push\_down(y); push\_down(x);

if(pre[pre[x]] == goal) {

Rotate(x , ch[pre[x]][0] == x);

} else {

int y = pre[x] , z = pre[y];

int f = (ch[z][0] == y);

if(ch[y][f] == x) {

Rotate(x , !f) , Rotate(x , f);

} else {

Rotate(y , f) , Rotate(x , f);

}

}

}

push\_up(x);

if(goal == 0) root = x;

}

void RotateTo(int k,int goal) {//把第k位的数转到goal下边

int x = root;

push\_down(x);

while(sz[ ch[x][0] ] != k) {

// printf("x = %d k = %d sz[x] = %d\n",x,k,sz[x]);

if(k < sz[ ch[x][0] ]) {

x = ch[x][0];

} else {

k -= (sz[ ch[x][0] ] + 1);

x = ch[x][1];

}

push\_down(x);

}

Splay(x,goal);

}

//以上一般不修改//////////////////////////////////////////////////////////////////////////////

void debug() {

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

Treaval(root);

}

void Treaval(int x) {

if(x) {

Treaval(ch[x][0]);

printf("结点%2d:左儿子 %2d 右儿子 %2d 父结点 %2d size = %2d ,val = %2d\n",x,ch[x][0],ch[x][1],pre[x],sz[x],val[x]);

Treaval(ch[x][1]);

}

}

//以上Debug

//以下是题目的特定函数:

void NewNode(int &x,int c) {

x = ++top1;

ch[x][0] = ch[x][1] = pre[x] = 0;

sz[x] = 1;

val[x] = c;

add[x] = 0;

}

//把延迟标记推到孩子

void push\_down(int x) {

if(add[x]) {

val[ ch[x][0] ] += add[x];

val[ ch[x][1] ] += add[x];

add[ ch[x][0] ] += add[x];

add[ ch[x][1] ] += add[x];

add[x] = 0;

}

return ;

}

//把孩子状态更新上来

void push\_up(int x) {

sz[x] = 1 + sz[ ch[x][0] ] + sz[ ch[x][1] ];

}

/\*初始化\*/

void makeTree(int &x,int l,int r,int f) {

if(l > r) return ;

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

NewNode(x , num[m]); /\*num[m]权值改成题目所需的\*/

makeTree(ch[x][0] , l , m - 1 , x);

makeTree(ch[x][1] , m + 1 , r , x);

pre[x] = f;

push\_up(x);

}

void clear() {

cnt = 0 ;

ch[0][0] = ch[0][1] = pre[0] = sz[0] = 0;

add[0] = val[0] = 0;

root = top1 = 0;

//为了方便处理边界,加两个边界顶点

NewNode(root , -inf);

NewNode(ch[root][1] , inf);

pre[top1] = root;

sz[root] = 2;

}

void init(int pos,int n) {

clear();

cnt = n;

RotateTo(pos , 0 );

RotateTo(pos + 1, root);

makeTree(keyTree , 1 , n , ch[root][1]);

push\_up(ch[root][1]);

push\_up(root);

}

int find\_low(int x) { //小于x的最大值位置

int s=0,p=root;

while(p) {

push\_down(p);

if(val[p]<=x) s=p,p=ch[p][1];

else p=ch[p][0];

}

return s;

}

int find\_up(int x) { //大于x的最小值位置

int s=0,p=root;

while(p) {

push\_down(p);

if(val[p]>=x) s=p,p=ch[p][0];

else p=ch[p][1];

}

return s;

}

void insert(int value) {

cnt ++ ;

int k = 0 ,x = root;

while(x) {

push\_down(x);

if(val[x] > value)

x = ch[x][0];

else {

k += (sz[ch[x][0]] + 1);

x = ch[x][1];

}

}

RotateTo(k - 1 , 0);

RotateTo(k , root);

NewNode(keyTree,value);

pre[keyTree] = ch[root][1];

// printf("push %d %d\n",ch[root][1],root);

// push\_up(ch[root][1]);

push\_up(root);

}

void del(int value) {

int k = 0 ,x = root;

while(x) {

push\_down(x);

if(val[x] >= value)

x = ch[x][0];

else {

k += (sz[ch[x][0]] + 1);

x = ch[x][1];

}

}

RotateTo(0 , 0);

RotateTo(k , root);

int key = keyTree;

keyTree = 0;

// push\_up(ch[root][1]);

push\_up(root);

cnt -= sz[key];

le += sz[key];

//erase(keyTree);

}

void update(int value) {

RotateTo(0,0);

RotateTo(cnt + 1, root);

val[keyTree] += value;

add[keyTree] += value;

push\_up(root);

}

int query(int k) {

if(cnt < k) return -1;

k = cnt - k + 1;

RotateTo(k - 1, 0);

RotateTo(k + 1, root);

return val[keyTree];

}

int cnt ;

/\*这是题目特定变量\*/

int num[maxn];

int val[maxn];

int add[maxn];

} spt;

### KD树

typedef long long ll;

const long long INF=1LL<<61;

const int MaxN=100010;

struct POINT {

ll x,y;

ll dis(const POINT &tp) {

return (x-tp.x)\*(x-tp.x)+(y-tp.y)\*(y-tp.y);

}

} tree[MaxN],p[MaxN],q[MaxN];

bool flag[MaxN];

bool cmpx(const POINT &p1,const POINT &p2)

{

return p1.x<p2.x;

}

bool cmpy(const POINT &p1,const POINT &p2)

{

return p1.y<p2.y;

}

ll ans;

void built(int L,int R)

{

if(L>R)return;

int M=(L+R)>>1;

ll miX=INF,maX=-INF,miY=INF,maY=-INF;

for(int i=L; i<=R; i++) {

miX=min(miX,p[i].x);

maX=max(maX,p[i].x);

miY=min(miY,p[i].y);

maY=max(maY,p[i].y);

}

flag[M]=(maX-miX)>=(maY-miY);

nth\_element(p+L,p+M,p+R+1,flag[M]? cmpx:cmpy);

tree[M]=p[M];

built(L,M-1);

built(M+1,R);

}

void fint(int L,int R,POINT v)

{

if(L>R)return;

int M=(L+R)>>1;

ll tmp=tree[M].dis(v);

if(tmp!=0) ans=min(ans,tmp);

if(flag[M]) {

ll d=(v.x-tree[M].x)\*(v.x-tree[M].x);

if(v.x<=tree[M].x) {

fint(L,M-1,v);

if(d<ans) fint(M+1,R,v);

} else {

fint(M+1,R,v);

if(d<ans) fint(L,M-1,v);

}

} else {

ll d=(v.y-tree[M].y)\*(v.y-tree[M].y);

if(v.y<=tree[M].y) {

fint(L,M-1,v);

if(d<ans) fint(M+1,R,v);

} else {

fint(M+1,R,v);

if(d<ans) fint(L,M-1,v);

}

}

}

int main()

{

int T;

scanf("%d",&T);

while(T--) {

int n;

scanf("%d",&n);

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

scanf("%lld%lld",&p[i].x,&p[i].y);

q[i]=p[i];

}

built(0,n-1);

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

ans=INF;

fint(0,n-1,q[i]);

printf("%I64d\n",ans);

}

}

return 0;

}

### DLX——精确覆盖

#### 模板1

/\*\*

在计算科学理论中，这一类问题的解答被称之为NPC问题中的Hitting Set Problem，中文名应该叫做碰集问题。

该类问题可以通过转换成为精确覆盖问题，其中以行表示概然，以列表示常规约束。

在数独问题中，行所表示的概然状态很明显为（r,c,k）即行，列，放置的数字。而列所表示的约束大致整理了下，

分做四种，即r行中放置数k可行性，c列放置数k可行性，放置于(r,c)格子可行性以及块b(即所属区域)放置数k的可行性。

因此行总共有N\*N\*N=9\*9\*9=729个，列总共有9\*9\*4=324个，问题转化为在729\*324的矩阵中取若干行，使每个列只有一个'1'，

此时对应一个数独的解，而(r,c)格的约束保证了我们最后解的行数一定<=N\*N。至此，模型转化完成。

\*\*/

const int maxn=50000+123;

const int maxc=5000+5;

int S[maxc], L[maxn], R[maxn], D[maxn], U[maxn];

int H[maxc], ok[maxc], sz, C[maxn], mark[maxn];

///H是横向的表头， 不知道必不必要

void Link(int row, int col) {

S[col]++;

C[sz]=col;///C域指向列头

U[sz]=U[col];

D[U[col]]=sz;

D[sz]=col;

U[col]=sz;

if(H[row]==-1)H[row]=L[sz]=R[sz]=sz;

else {

L[sz]=L[H[row]];

R[L[H[row]]]=sz;

R[sz]=H[row];

L[H[row]]=sz;

}

mark[sz]=row;/// 标记每个点是哪一行（题目要求输出解属于哪一行）

sz++;

}

void remove(int col) {

L[R[col]]=L[col];

R[L[col]]=R[col]; /// 在列对象链表中删除col

for (int i=D[col]; i!=col; i=D[i]) {

/// 删除col列中有1元素的行

for (int j=R[i]; j!=i; j=R[j]) {

///删除每行的1元素，并修改所在列的S域

U[D[j]]=U[j], D[U[j]]=D[j];

S[C[j]]--;

}

}

}

void resume(int col) {

for (int i=U[col]; i!=col; i=U[i]) {

for (int j=L[i]; j!=i; j=L[j]) {

U[D[j]]=j;

D[U[j]]=j;

S[C[j]]++;

}///恢复删除的元素，恢复S域

}///恢复删除的行

L[R[col]]=col;

R[L[col]]=col;

}

bool Dance(int k) {

if(R[0]==0) {

char ans[20][20];

///printf("%d\n", '@');

sort(ok, ok+k);

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

///printf("%d ", ok[i]);

int pos=(ok[i]-1)/16;

///printf("%d\n", pos);

///printf("x=%d,y=%d\n", pos/16, pos%16);

ans[pos/16][pos%16]=(ok[i]-1)%16+'A';

//printf("%c", ans[pos/16][pos%16]);

}

for(int i=0; i<16; ++i) {

ans[i][16]=0;

printf("%s\n", ans[i]);

}

puts("");

return true;

}

int c=R[0];

for(int i=R[0]; i; i=R[i])

if(S[i]<S[c])c=i;

remove(c);

///printf("remove col %d\n", c);

for(int i=D[c]; i!=c; i=D[i]) {

ok[k]=mark[i];

for (int j=R[i]; j!=i; j=R[j])

remove(C[j]);

if(Dance(k+1))return true;

for (int j=L[i]; j!=i; j=L[j])

resume(C[j]);

}

resume(c);

return false;

}

void initL(int x) {

for (int i=0; i<=x; ++i) { ///i<x???

S[i]=0;

D[i]=U[i]=i;

L[i+1]=i;

R[i]=i+1;

}///对列表头初始化

R[x]=0;

sz=x+1;///真正的元素从m+1开始

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

///mark每个位置的名字

}

#### 模板2

#define maxnode 60000

#define maxn 5005

struct DLX {

int n , sz; // 行数，节点总数

int S[maxn]; // 各列节点总数

int row[maxnode],col[maxnode]; // 各节点行列编号

int L[maxnode],R[maxnode],U[maxnode],D[maxnode]; // 十字链表

int ansd,ans[maxn]; // 解

void init(int n ) {

this->n = n ;

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

U[i] = i ;

D[i] = i ;

L[i] = i - 1;

R[i] = i + 1;

}

R[n] = 0 ;

L[0] = n;

sz = n + 1 ;

memset(S,0,sizeof(S));

}

void addRow(int r,vector<int> c1) {

int first = sz;

for(int i = 0 ; i < c1.size(); i++ ) {

int c = c1[i];

L[sz] = sz - 1 ;

R[sz] = sz + 1 ;

D[sz] = c ;

U[sz] = U[c];

D[U[c]] = sz;

U[c] = sz;

row[sz] = r;

col[sz] = c;

S[c] ++ ;

sz ++ ;

}

R[sz - 1] = first ;

L[first] = sz - 1;

}

// 顺着链表A，遍历除s外的其他元素

#define FOR(i,A,s) for(int i = A[s]; i != s ; i = A[i])

void remove(int c) {

L[R[c]] = L[c];

R[L[c]] = R[c];

FOR(i,D,c)

FOR(j,R,i) {

U[D[j]] = U[j];

D[U[j]] = D[j];

--S[col[j]];

}

}

void restore(int c) {

FOR(i,U,c)

FOR(j,L,i) {

++S[col[j]];

U[D[j]] = j;

D[U[j]] = j;

}

L[R[c]] = c;

R[L[c]] = c;

}

bool dfs(int d) {

if(R[0] == 0 ) {

ansd = d;

return true;

}

// 找S最小的列c

int c = R[0] ;

FOR(i,R,0) if(S[i] < S[c]) c = i;

remove(c);

FOR(i,D,c) {

ans[d] = row[i];

FOR(j,R,i) remove(col[j]);

if(dfs(d + 1)) return true;

FOR(j,L,i) restore(col[j]);

}

restore(c);

return false;

}

bool solve(vector<int> & v) {

v.clear();

if(!dfs(0)) return false;

for(int i = 0 ; i< ansd ; i ++ ) v.push\_back(ans[i]);

return true;

}

};

DLX solver;

const int SLOT = 0;

const int ROW = 1;

const int COL = 2;

const int SUB = 3;

int encode(int a ,int b,int c) {

return a \* 81 + 9 \* b + c + 1;

}

void decode(int code ,int & a ,int & b, int & c ) {

code -- ;

c = code % 9;

code /= 9;

b = code % 9;

code /= 9;

a = code ;

}

char mf[20][20];

char str[100];

void print() {

for(int i = 0 ; i < 9; i ++ )

printf("%s",mf[i]);

printf("\n");

}

bool read() {

scanf("%s",str);

if(str[0] == 'e' && str[1] == 'n' && str[2] =='d') return false;

for(int i = 0 ; i < 81 ; i ++ )

mf[i/9][i%9] = str[i];

for(int i = 0 ; i < 9 ; i ++ ) mf[i][10] = 0;

//print();

return true;

}

int main() {

int cas;

cas = 0 ;

while(read()) {

// if(cas != 0 ) cout << endl;

cas ++ ;

solver.init(324);

for(int r = 0 ; r < 9 ; r ++ )

for(int c = 0 ; c < 9; c ++ )

for(int v = 0 ; v < 9; v ++ )

if(mf[r][c] == '.' || mf[r][c] == '1' + v) {

vector<int> c1;

c1.clear();

c1.push\_back(encode(SLOT,r,c));

c1.push\_back(encode(ROW,r,v));

c1.push\_back(encode(COL,c,v));

c1.push\_back(encode(SUB,(r/3)\*3+c/3,v));

solver.addRow(encode(r,c,v),c1);

}

vector<int> ans;

solver.solve(ans);

for(int i = 0 ; i < ans.size(); i ++ ) {

int r,c,v;

decode(ans[i],r,c,v);

mf[r][c] = '1' + v;

}

print();

}

return 0;

}

### DLX——重复覆盖

#### 模板1

const int N = 1005;

const int M = 4000;

int U[M], D[M], R[M], L[M], rs[N];

int size[N], head, row[M], col[M];

int n, m, k;

int nodecnt, ans;

void init(int m) {

memset(size, 0, sizeof (size));

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

nodecnt = m + 1;

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

L[i] = (i - 1);

R[i] = (i + 1);

col[i] = i;

D[i] = U[i] = i;

}

L[0] = m;

R[m] = 0;

size[0] = INT\_MAX;

}

inline void Remove(int x) {

for (int i = D[x]; i != x; i = D[i]) {

L[ R[i] ] = L[i];

R[ L[i] ] = R[i];

size[ col[i] ]--;

}

}

inline void Resume(int x) {

for (int i = U[x]; i != x; i = U[i]) {

L[ R[i] ] = R[ L[i] ] = i;

size[ col[i] ]++;

}

}

int rec[N];

inline int evalute() { /\* 估价函数 \*/

bool vis[40] = {false};

int tt = 0;

for (int i = R[0]; i != 0; i = R[i])

if (!vis[col[i]]) {

tt++;

vis[col[i]] = true;

for (int j = D[i]; j != i; j = D[j])

if (col[j] != 0)

for (int k = R[j]; k != j; k = R[k]) vis[ col[k] ] = true;

}

return tt;

}

int dfs(int num) { /\*这里的dfs是一个迭代加深的过程\*/

if (num + evalute() > ans) return 0;

if (R[0] == 0) return 1;

int i, j, opt = 0;

for (i = R[0]; i != 0; i = R[i])

if (size[col[i]] < size[col[opt]]) {

opt = i;

if (size[col[i]] <= 1) break;

}

for (i = D[opt]; i != opt; i = D[i]) {

Remove(i);

for (j = R[i]; i != j; j = R[j]) Remove(j);

if (dfs(num + 1)) return 1;

for (j = L[i]; j != i; j = L[j]) Resume(j);

Resume(i);

}

return 0;

}

inline void insert(int i, int \*tt, int c) {

for (int j = 0; j < c; j++, nodecnt++) {

int x = tt[j];

row[nodecnt] = i;

col[nodecnt] = x;

size[x]++;

U[nodecnt] = x;

D[nodecnt] = D[x];

U[D[x]] = nodecnt;

D[x] = nodecnt;

if (j == 0) L[nodecnt] = R[nodecnt] = nodecnt;

else {

L[nodecnt] = nodecnt - 1;

R[nodecnt] = nodecnt - j;

R[nodecnt - 1] = nodecnt;

L[nodecnt - j] = nodecnt;

}

}

}

void build() {

/\*建图过程，同精确覆盖，通过 insert(i,tt,c) 完成\*/

}

void solve() {

ans = 0;

while (!dfs(0)) {

ans++;

if (ans > m) break;

}

/\*解题过程\*/

}

int main() {

while (scanf("%d%d%d", &n, &m, &k)!=EOF) {

read(); /\*读入初始化\*/

build(); /\*建图过程\*/

solve(); /\*解题过程\*/

}

return 0;

}

#### 模板2

const int maxn=360000;

const int maxc=500;

const int maxr=500;

const int inf=0x3f3f3f3f;

int L[maxn], R[maxn], D[maxn], U[maxn], C[maxn];

int S[maxc], H[maxr], size;

///不需要S域

int sgn(const double & s)

{

return (int)(s > eps) - (s < -eps);

}

void Link(int r, int c)

{

S[c]++;

C[size]=c;

U[size]=U[c];

D[U[c]]=size;

D[size]=c;

U[c]=size;

if(H[r]==-1) H[r]=L[size]=R[size]=size;

else

{

L[size]=L[H[r]];

R[L[H[r]]]=size;

R[size]=H[r];

L[H[r]]=size;

}

size++;

}

void remove(int c)

{

for (int i=D[c]; i!=c; i=D[i])

L[R[i]]=L[i], R[L[i]]=R[i];

}

void resume(int c)

{

for (int i=U[c]; i!=c; i=U[i])

L[R[i]]=R[L[i]]=i;

}

int h() ///用精确覆盖去估算剪枝

{

int ret=0;

bool vis[maxc];

memset (vis, false, sizeof(vis));

for (int i=R[0]; i; i=R[i])

{

if(vis[i])continue;

ret++;

vis[i]=true;

for (int j=D[i]; j!=i; j=D[j])

for (int k=R[j]; k!=j; k=R[k])

vis[C[k]]=true;

}

return ret;

}

int ans;

int limit;

bool Dance(int k1)

{

if(k1+h() > limit ) return false;;

if(!R[0])

return true;

int c=R[0];

for (int i=R[0]; i; i=R[i])

if(S[i]<S[c])c=i;

//printf("c = %d\n",c);

for (int i=D[c]; i!=c; i=D[i])

{

remove(i);

for (int j=R[i]; j!=i; j=R[j])

remove(j);

if(Dance(k1+1)) return true;

for (int j=L[i]; j!=i; j=L[j])

resume(j);

resume(i);

}

return false;

}

void initL(int x) ///col is 1~x,row start from 1

{

for (int i=0; i<=x; ++i)

{

S[i]=0;

D[i]=U[i]=i;

L[i+1]=i;

R[i]=i+1;

}///对列表头初始化

R[x]=0;

size=x+1;///真正的元素从m+1开始

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

///mark每个位置的名字

}

### 数独

/\* class Shudo \*/

class Shudo

{

private:

int type,square,block;

public:

Shudo( int n ) {

type = n;

square = type\*type;

block = square\*square;

}

int FF( int s ) {return s;}

int rr( int s ) {return s/square;}

int ll( int s ) {return s%square;}

int bb( int s ) {return rr(s)/type\*type+ll(s)/type;}

int rs( int s, int c ) {return rr(s)\*square+FF(c);}

int ls( int s, int c ) {return ll(s)\*square+FF(c)+block;}

int bs( int s, int c ) {return bb(s)\*square+FF(c)+block\*2;}

public:

int save( int \*line, int s, int c, int move ) {

line[ move ++ ] = rs(s,c);

line[ move ++ ] = ls(s,c);

line[ move ++ ] = bs(s,c);

line[ move ++ ] = s+1+3\*block;

return move;

}

int getfill( int\* line, int \*data ) {

int count = 0;

for ( int i = 0 ; i < block ; ++ i )

if ( data[ i ] >= 1 && data[ i ] <= 9 )

count = save( line, i, data[ i ], count );

return count;

}

};

/\* shudo end \*/

### RMQ

#### 一维RMQ

#### 二维RMQ

\*用dp[row][col][i][j]表示(row,col)到(row+2^i,col+2^j)矩形内的最小值

\*查询的时候

\* int kx = log(double(x2 - x1 +1)) / log(2.0);

int ky = log(double(y2 - y1 +1)) / log(2.0);

int m1 = dp[x1][y1][kx][ky];

int m2 = dp[x2-(1<<kx)+1][y1][kx][ky];

int m3 = dp[x1][y2-(1<<ky)+1][kx][ky];

int m4 = dp[x2-(1<<kx)+1][y2-(1<<ky)+1][kx][ky];

\*取4个值里面的最小值(有种二分的思想)

\*/

const int maxn = 310;

int mat[maxn][maxn];

int dp[maxn][maxn][9][9];

int n , q;

void rmq\_2d(void) {

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

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

dp[row][col][0][0] = mat[row][col];

int m = log((double)n) / log(2.0);

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

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

if(i == 0 && j == 0)

continue;

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

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

if(i == 0) {

dp[row][col][i][j] = min(dp[row][col][i][j-1] , dp[row][col+(1<<(j-1))][i][j-1]);

} else {

dp[row][col][i][j] = min(dp[row][col][i-1][j] , dp[row+(1<<(i-1))][col][i-1][j]);

}

}

}

}

}

}

int query\_2d(int x1,int x2,int y1,int y2) {

int kx = log(double(x2 - x1 +1)) / log(2.0);

int ky = log(double(y2 - y1 +1)) / log(2.0);

int m1 = dp[x1][y1][kx][ky];

int m2 = dp[x2-(1<<kx)+1][y1][kx][ky];

int m3 = dp[x1][y2-(1<<ky)+1][kx][ky];

int m4 = dp[x2-(1<<kx)+1][y2-(1<<ky)+1][kx][ky];

return min( min(m1,m2), min(m3,m4) );

}

## 字符串

### KMP算法

int n,m;

string s,t;

// s wei text m

// t wei pattern n

int pre[maxn];

void getnext() {

pre[0] = -1;

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

while(j!=-1 && t[i]!=t[j+1]) j = pre[j];

if(t[i]==t[j+1]) ++j;

pre[i] = j;

}

}

void kmp() {

for(int j=-1,i=0; i<m; ++i) {

while(j!=-1 && s[i]!=t[j+1]) j = pre[j];

if(s[i]==t[j+1]) ++j;

if(j==n-1) {

printf("%d\n",i - n + 1 );

}

}

}

### AC自动机

#### 模板1

const int UNDEF = 0;

const int MAXN = 10004 \* 50;

const int CHARSET = 26;

int ans ;

struct ACmation {

int cnt[MAXN];

int fail[MAXN];

int ch[MAXN][CHARSET];

int end;

void init() {

cnt[0] = UNDEF;

fill(ch[0], ch[0] + CHARSET, -1);

end = 1;

}

void add(char \*str) {

int p = 0;

int m = strlen(str);

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

if (ch[p][\*str - 'a'] == -1) {

cnt[end] = UNDEF;

fill(ch[end], ch[end] + CHARSET, -1);

ch[p][\*str - 'a'] = end++;

}

p = ch[p][\*str -'a'];

++str;

}

cnt[p] ++ ;

}

void build() {

queue<int> bfs;

fail[0] = 0;

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

if (ch[0][i] != -1) {

fail[ch[0][i]] = 0;

bfs.push(ch[0][i]);

} else {

ch[0][i] = 0;

}

}

while (!bfs.empty()) {

int p = bfs.front();

bfs.pop();

//cnt[p] |= cnt[fail[p]];

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

if (ch[p][i] != -1) {

fail[ch[p][i]] = ch[fail[p]][i];

bfs.push(ch[p][i]);

} else {

ch[p][i] = ch[fail[p]][i];

}

}

}

}

void find(char \*s ) {

int len , j , u ;

len = strlen(s);

u = 0 ;

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

j = s[i] - 'a';

u = ch[u][j];

for(int k = u ; k ; k = fail[k]) {

ans += cnt[k] ;

cnt[k] = 0 ;

}

}

return ;

}

} ac;

#### 模板2

#define maxn 10004\*50

struct node {

int ch[26];

int cnt;

int fail;

void init() {

memset(ch,0,sizeof(ch));

fail = 0 ;

cnt = 0 ;

}

};

node tt[maxn];

int ROOT ,sind ;

void init() {

ROOT = 0 ;

tt[0].init();

sind = 1 ;

}

void ins(char \*str) {

int idx = ROOT;

int len = strlen(str);

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

int j = str[i] - 'a' ;

if(tt[idx].ch[j] == 0 ) {

tt[sind].init();

tt[idx].ch[j] = sind++;

}

idx = tt[idx].ch[j];

}

tt[idx].cnt ++ ;

}

void makefail() {

queue<int> q;

while(!q.empty()) q.pop();

for(int i = 0 ; i < 26 ; i ++) {

if(tt[ROOT].ch[i]) {

tt[tt[ROOT].ch[i]].fail = ROOT;

q.push(tt[ROOT].ch[i]);

}

}

while(!q.empty()) {

int u = q.front();

q.pop();

for(int i = 0 ; i < 26 ; i ++) {

if(tt[u].ch[i]) {

int v = tt[u].ch[i];

q.push(v);

int fail = tt[u].fail;

tt[v].fail = tt[fail].ch[i];

//tt[v].cnt = tt[tt[fail].ch[i]].cnt;//AC自动机状态合并

} else {

int fail = tt[u].fail;

tt[u].ch[i] = tt[fail].ch[i];

}

}

}

}

int ans ;

void find(char \*s) {

int u , j , len ;

len = strlen(s);

u = ROOT ;

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

j = s[i] - 'a';

u = tt[u].ch[j];

for(int k = u ; k ; k = tt[k].fail ) {

ans = ans + tt[k].cnt;

tt[k].cnt = 0 ;

}

}

return ;

}

#### 模板3

const int maxn = 10004 \* 51;

int n;

char s[1000006], str[55];

int ans ;

struct AC {

int ch[maxn][26], tot, f[maxn];

int is[maxn];

int idx(char c) {

return c-'a';

}

int new\_node() {

is[tot] = 0;

memset(ch[tot], 0, sizeof(ch[tot]));

return tot++;

}

void init() {

tot = 0;

new\_node();

}

void insert(char \*s, int pos) {

int i, j, u = 0, n = strlen(s);

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

int k = idx(s[i]);

if(!ch[u][k]) ch[u][k] = new\_node();

u = ch[u][k];

}

is[u] ++ ;

}

void getfail() {

int i, j, u, v;

queue <int> q;

f[0] = 0;

for(i = 0; i < 26; i++) {

u = ch[0][i];

if(u) {

f[u] = 0;

q.push(u);

}

}

while(!q.empty()) {

u = q.front();

q.pop();

for(i = 0; i < 26; i++) {

v = ch[u][i];

if(!v) {

ch[u][i] = ch[f[u]][i]; // 这行是AC自动机改造： 没改造的是这样的 if(!v) continue;

continue;

}

j = f[u];

while(j && !ch[j][i]) j = f[j];

f[v] = ch[j][i];

q.push(v);

}

}

}

void find(char \*s ) {

int len , j , u ;

len = strlen(s);

u = 0 ;

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

j = idx(s[i]);

u = ch[u][j];

for(int k = u ; k ; k = f[k]) {

if(is[k]) {

ans += is[k] ;

is[k] = 0 ;

}

}

}

}

} ac;

#### 模板4

#define maxn 10005\*50

int ans ;

struct Trie {

int ch[maxn][26],fail[maxn],cnt[maxn],sind,root;

int NewNode() {

++sind;

memset(ch[sind],-1,sizeof(ch[sind]));

cnt[sind]=0;

fail[sind]=-1;

return sind;

}

void init() {

sind=0;

root=NewNode();

}

void ins(char \*s ) {

int len=strlen(s),pos=root;

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

int val=s[i]-'a';

if (ch[pos][val]==-1) ch[pos][val]=NewNode();

pos=ch[pos][val];

}

cnt[pos]++;

}

queue<int> Q;

void getfail() {

fail[root]=root;

for (int i=0; i<26; ++i)

if (ch[root][i]==-1) ch[root][i]=root;

else {

fail[ch[root][i]]=root;

Q.push(ch[root][i]);

}

while (!Q.empty()) {

int pos=Q.front();

Q.pop();

for (int i=0; i<26; ++i)

if (ch[pos][i]==-1) ch[pos][i]=ch[fail[pos]][i];

else {

fail[ch[pos][i]]=ch[fail[pos]][i];

Q.push(ch[pos][i]);

}

}

}

void find(char \*s ) {

int len=strlen(s),pos=root ;

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

int tmp=ch[pos][s[i]-'a'];

pos=tmp;

while (tmp!=root) {

ans += cnt[tmp];

cnt[tmp]=0;

tmp=fail[tmp];

}

}

}

} ac;

### 后缀自动机

#### 模板1

#define maxn 20005

struct suffix\_automaton {

int f[maxn << 1 ] , ant , last ;

int pos[maxn<<1];

int ch[maxn << 1][26],step[maxn<<1];

void init() {

last = ant =1 ;

memset(f,0,sizeof(f));

memset(ch,0,sizeof(ch));

memset(step,0,sizeof(step));

}

void add(int x ) {

int t = ++ ant ,pa =last;

step[t] = step[last] + 1 ;

pos[t] = step[t];

last = t;

for(; pa && !ch[pa][x]; pa = f[pa])

ch[pa][x] = t;

if(pa == 0 ) f[t] = 1;

else if(step[pa] + 1 == step[ch[pa][x]])

f[t] = ch[pa][x];

else {

int nq = ++ ant,q = ch[pa][x];

memcpy(ch[nq],ch[q],sizeof(ch[nq]));

step[nq] = step[pa] + 1 ;

f[nq] = f[q];

f[q] = f[t] = nq;

pos[q] = pos[t];

for(; pa && ch[pa][x] == q; pa = f[pa])

ch[pa][x] = nq;

}

}

};

suffix\_automaton SAM;

#### 模板2

#define N 2010

#define MAXQ 10010

struct Suffix\_Automaton

{

int F[N << 1],ant,last,ch[N << 1][26],step[N << 1];

void init()

{

last = ant = 1;

memset(F,0,sizeof(F));

memset(ch,0,sizeof(ch));

memset(step,0,sizeof(step));

}

void ins(int x)

{

int t = ++ant, pa = last;

step[t] = step[last] + 1;

last = t;

for( ; pa && !ch[pa][x]; pa = F[pa] )

ch[pa][x] = t;

if( pa == 0 ) F[t] = 1;

else if( step[pa] + 1 == step[ ch[pa][x] ] )

F[t] = ch[pa][x];

else

{

int nq = ++ant, q = ch[pa][x];

memcpy( ch[nq], ch[q], sizeof(ch[nq]) );

step[nq] = step[pa] + 1;

F[nq] = F[q];

F[q] = F[t] = nq;

for( ; pa && ch[pa][x] == q; pa = F[pa] )

ch[pa][x] = nq;

}

}

};

#### 模板3

const int MAX\_N = 250000 + 10;

char buf[MAX\_N];

struct State {

State\*suf, \*go[26];

int val;

State() :

suf(0), val(0) {

memset(go, 0, sizeof go);

}

}\*root, \*last;

State statePool[MAX\_N \* 2], \*cur;

void init() {

cur = statePool;

root = last = cur++;

}

void extend(int w) {

State\*p = last, \*np = cur++;

np->val = p->val + 1;

while (p && !p->go[w])

p->go[w] = np, p = p->suf;

if (!p)

np->suf = root;

else {

State\*q = p->go[w];

if (p->val + 1 == q->val) {

np->suf = q;

} else {

State\*nq = cur++;

memcpy(nq->go, q->go, sizeof q->go);

nq->val = p->val + 1;

nq->suf = q->suf;

q->suf = nq;

np->suf = nq;

while (p && p->go[w] == q)

p->go[w] = nq, p = p->suf;

}

}

last = np;

}

### 后缀数组\*

### 回文串Manacher算法

/\* 回文串Manacher算法

\* 时间复杂度O(n)，空间复杂度O(n)

\* \*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

// cs 为输入的字符串，从0开始索引，n为字符串长度

// len[i]为以i位置为中心，回文串的半径长度，len数组需要是2\*n大小

// 如果i为偶数，则表示以i/2的字符为中心，i为奇数，则以i/2字符和(i+1)/2的字符的中间线为中心

// len[i]表是以i为中心，回文长度是len[i]

void palindrome(char cs[], int len[], int n) {

memset(len, 0, sizeof(len[0]) \* n \* 2);

for (int i = 0, j = 0, k; i < n \* 2; i += k, j = max(j - k, 0) ) {

while (i - j >= 0 && i + j + 1 < n \* 2 && cs[(i - j) / 2] == cs[(i + j + 1) / 2]) ++j; /\*每次多比较两次\*/

len[i] = j;

for (k = 1; i - k >= 0 && j - k >= 0 && len[i - k] != j - k; ++k) {

len[i+k] = min(len[i-k], j - k);

}

}

}

void palindrome2(char cs[], int len[], int n) { /\* 这个貌似快点,应该是循环少吧，代码也少 \*/

for (int i = 0, mx = 0, k, l; i < n \* 2; ++i) {

if (i < mx) l = min(len[k - (i - k)], mx - i);

else l = 0;

while (i-l >= 0 && cs[(i-l)/2] == cs[(i+l+1)/2]) ++l;

len[i] = l;

if (i + l > mx) {

mx = i + l;

k = i;

}

}

}

char str[128] = "aabaaab";

int len[256];

void pp(char cs[], int i, int l) {

printf("%2d:[%2d,%2d] %2d:", i, i/2 - l / 2 + i % 1, i/2 + l/2 + i%2, l);

for (int j = i/2 - l/2 + i % 1; j <= i/2 + l/2 + i % 1; ++j) printf("%c", cs[j]);

puts("");

}

void work() {

int n = strlen(str);

puts(str);

palindrome(str, len, n);

for (int i = 0; i < 2\*n; ++i) pp(str, i, len[i]);

puts("");

palindrome2(str, len, n);

for (int i = 0; i < 2\*n; ++i) pp(str, i, len[i]);

}

// test palindrome

int main() {

work();

while (EOF != scanf("%s", str)) {

if (strlen(str) == 0) break;

work();

}

return 0;

}

### 最小表示法

最小表示法：

初始时，i=0,j=1，分别以i，j，为起始点顺着i，j，往下比较直到找的str[i+k]!=str[j+k]，然后分两种情况考虑：

1、 str[i+k]>str[j+k]，i变成i=i+k+1，j不变，然后继续往下比较。

2、 str[i+k]<str[j+k]，j变成j=j+k+1，i不变，然后继续往下比较。

直到i或j大于串长，找较小者。

说说我的疑惑吧：

用枚举方法我们很好理解，一某一点i为起点，不断查看以i后的点为起点的串是否小于以i为起点的串，这理解起来不难。我们假设以i点为起点的串为S1，以j点为起点的串S2，而用最小表示法的话，需要找的S1、S2第一个不相等的字符，假设距离S1、S2起点的k点为第一个不相同的点，然后做一下比较，使i或者j跳到k的下一个字符，这就避免了枚举，这也就是我疑惑的地方，为什么0到k之间（相对于S1，S2来说）为起点的串都不可能成为最小串呢？假如S1[k]>S2[k] 只能说明以S1[k]为起点的串不可能成为最小串，虽然S1[0..k-1] 和 S2[0…k-1]对应字符是相等的又能说明什么问题呢？我想了很久没有想明白。我突发奇想，假如说S1[0..k-1]和S2[0..k-1]所有字符都是相等的，这就可以解释我刚才的疑惑了：假如说S1[k]>S2[k]，若S1[k]>S1[k-1]，想想就知道S1[0…k]是不可能成为最小串的起点。若S1[k]<S1[k-1]，S1[0…k-1]（注意是0到k-1）那就更不可能了，然而S2[k]又小于S1[k]，所以S1[0…k]就不可能了。

我不知道我的想法对还是不对，反正我没有找到反例，姑且这样认为吧，有更好的解释留言吧。

int work(int m)

{

int i,j,l;

i=0; j=1;

while(i<m && j<m)

{

for(l=0;l<m;l++)

if(str[(i+l)%m]!=str[(j+l)%m]) break;

if(l>m) break;

if(str[(i+l)%m] > str[(j+l)%m])

i=i+l+1;

else

j=j+l+1;

if(i==j) j=i+1;

}

if(i<j) return i;

return j;

}

### 最小覆盖子串

最小覆盖子串（串尾多一小段时，用前缀覆盖）长度为n-next[n]（n-pre[n]），n为串长。

证明分两部分：

１－长为n-next[n]的前缀必为覆盖子串。

当next[n]<n-next[n]时，如图a，长为next[n]的前缀A与长为next[n]的后缀B相等，故长为n-next[n]的前缀C必覆盖后缀B；

当next[n]>n-next[n]时，如图b，将原串X向后移n-next[n]个单位得到Y串，根据next的定义，知长为next[n]的后缀串A与长为前缀串B相等，X串中的长为n-next[n]的前缀C与Y串中的前缀D相等，而X串中的串E又与Y串中的D相等……可见X串中的长为n-next[n]的前缀C可覆盖全串。

２－长为n-next[n]的前缀是最短的。

如图c，串A是长为n-next[n]的前缀，串B是长为next[n]的后缀，假设存在长度小于n-next[n]的前缀C能覆盖全串，则将原串X截去前面一段C，得到新串Ｙ，则Ｙ必与原串长度大于next[n]的前缀相等，与next数组的定义（使str[1..i]前k个字母与后k个字母相等的最大k值。）矛盾。得证！有人问，为什么Ｙ与原串长大于next[n]的前缀相等？由假设知原串的构成必为CCC……E（E为C的前缀），串Ｙ的构成必为CC……E（比原串少一个Ｃ），懂了吧！

## 树相关

### LCA转RMQ

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

E[++num] = u , L[num] = dep , R[u] = num;

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

if(e[i].v != fa) {

int v = e[i].v;

dfs(v, u, dep + 1 );

E[++num] = u , L[num] = dep;

}

return;

}

void initRMQ() {

for(int i = 1 ; i <= num ; i ++ ) f[i][0] = i;

for(int j = 1 ; (1<<j) <= num ; j++ )

for(int i = 1; i + (1<<j) -1 <= num; i ++ )

if(L[f[i][j-1]] < L[f[i+(1<<(j-1))][j-1]]) //ps:注意下标

f[i][j] = f[i][j-1];

else f[i][j] = f[i+(1<<(j-1))][j-1];

}

int lca(int a ,int b) {

a = R[a] , b = R[b];

if(a>b) swap(a,b);

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

if(L[f[a][k]] < L[f[b-(1<<k)+1][k]] )

return E[f[a][k]];

else return E[f[b-(1<<k)+1][k]];

}

### LCA

int check(int v,int a,int b) {

int fa = top[a],fb = top[b],s = 0;

t = v;

while (fa != fb) {

if (dep[fa] < dep[fb]) {

std::swap(fa,fb); std::swap(a,b);

}

st = w[fa]; ed= w[a];

s += find(1,1,n);

a = f[fa]; fa = top[a];//爬升是把a弄成a重链顶端的父亲。

}

if (dep[a] > dep[b]) std::swap(a,b);

st = w[a]; ed = w[b];//如果是维护边权的，这里查询的是w[son[a]]，因为边权信息保存在a上面。

s += find(1,1,n);//当a,b在同一条重链上后，要记得查询他们之间的信息

return s;

}

### 树的重心

int findroot(int cur) {

int i, j, x, rear = 0, max, min = INF, root;

q[rear ++] = cur, fa[cur] = 0;

for(i = 0; i < rear; i ++) {

x = q[i];

for(j = first[x]; j != -1; j = next[j])

if(!del[v[j]] && v[j] != fa[x])

q[rear ++] = v[j], fa[v[j]] = x;

}

for(i = rear - 1; i >= 0; i --) {

x = q[i];

size[x] = 1, max = 0;

for(j = first[x]; j != -1; j = next[j])

if(!del[v[j]] && v[j] != fa[x])

size[x] += size[v[j]], max = Max(max, size[v[j]]);

max = Max(max, rear - size[x]);

if(max < min)

min = max, root = x;

}

return root;

}

### 树链剖分

#define maxn 100010

int w[maxn],son[maxn],sz[maxn],top[maxn],fa[maxn],dep[maxn];

int d1[maxn][3];

int z;

int n ;

char str[15];

struct node {

int v ,next;

};

node e[maxn \* 2 ];

int cnt ;

int head[maxn];

int tt[maxn \* 3 ];

inline void add(int u ,int v) {

e[cnt].v = v;

e[cnt].next = head[u];

head[u] = cnt ++ ;

return ;

}

void push\_up(int root ) {

tt[root] = tt[root\*2] + tt[root\*2+1];

}

inline void dfs1(int u ) { // 计算 sz[],son[],dep[],fa[];

sz[u] = 1 ;

son[u] = 0;

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

if(e[i].v != fa[u]) {

fa[e[i].v] = u;

dep[e[i].v] = dep[u] + 1;

dfs1(e[i].v);

if(sz[e[i].v] > sz[son[u]] ) son[u] = e[i].v;

sz[u] += sz[e[i].v];

}

}

void dfs2(int u,int tp) { // 计算 w[],top[]

w[u] = ++ z ;

top[u] = tp;

if(son[u] != 0 ) dfs2(son[u] , top[u]);

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

if(e[i].v != fa[u] && e[i].v != son[u] )

dfs2(e[i].v,e[i].v);

return ;

}

void update(int root,int l,int r,int pos,int val) {

if(l == r ) {

tt[root] = val;

return ;

}

int mid = (l + r ) / 2;

if(pos <= mid)

update(root + root ,l,mid, pos , val);

else

update(root + root + 1 ,mid +1 , r , pos ,val);

push\_up(root);

return ;

}

int query(int root,int l1,int r1,int l,int r) {

if(l <= l1 && r1 <= r)

return tt[root];

int mid = (l1 + r1 ) / 2 ;

int res = 0 ;

if(l <= mid )

res = res + query(root + root,l1 , mid , l , r );

if(r > mid )

res = res + query(root + root + 1 ,mid + 1 , r1 , l , r );

return res;

}

int find(int a,int b) {

int f1 = top[a] , f2 = top[b] , tmp = 0 ;

while( f1 != f2 ) {

if(dep[f1] < dep[f2])

swap(f1,f2),swap(a,b);

tmp = tmp + query(1,1,z,w[f1],w[a]);

a = fa[f1],f1 = top[a];

}

if(a == b) return tmp ;

if(dep[a] > dep[b]) swap(a,b);

return tmp + query(1,1,z,w[son[a]],w[b]) ; // 父边

}

int main() {

int root;

int u,v,w1;

int cas;

int q,s;

while(scanf("%d%d%d",&n,&q,&s)!=EOF) {

cnt = 0 ;

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

memset(tt,0,sizeof(tt));

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

scanf("%d%d%d",&u,&v,&w1);

d1[i][0] = u;

d1[i][1] = v;

d1[i][2] = w1;

add(u,v);

add(v,u);

}

root = 1;

dep[root] = 0 ;

fa[root] = 0 ;

dfs1(root);

z = 0 ;

dfs2(root,root);

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

if(dep[d1[i][0]] > dep[d1[i][1]] ) swap(d1[i][0],d1[i][1]);

update(1,1,z,w[d1[i][1]] ,d1[i][2]);

}

int id;

int ans ;

while(q -- ) {

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

if(id == 0 ) {

ans = find(s,u);

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

s = u ;

} else {

scanf("%d",&v);

update(1,1,z,w[d1[u][1]],v);

}

}

}

return 0;

}

### LCT动态树

#### Xtt\_lct 适合点权

#include <cstdio>

#include <algorithm>

const int maxn = 300500;

int e[maxn][2],v,X,Y,w,n,TC,m,t,ans;

struct node {

int max,v,rev,del;

node\* ch[2],\*fa;

void add(int w) {

v += w;

del += w;

max += w;

}

} T[maxn],\*a[maxn],\*x,\*y;

node\* null = T;

inline void down(node\* x) {

if (x->del) {

x->ch[0]->add(x->del);

x->ch[1]->add(x->del);

x->del = 0;

}

if (x->rev) {

std::swap(x->ch[0],x->ch[1]);

x->ch[0]->rev ^= 1;

x->ch[1]->rev ^= 1;

x->rev = 0;

}

}

inline int max(int x,int y,int z) {

if (y > x) x = y;

if (z > x) x = z;

return x;

}

inline void update(node\* x) {

x->max = x->v;

if (x->ch[0] != null && x->ch[0]->max > x->max) x->max = x->ch[0]->max;

if (x->ch[1] != null && x->ch[1]->max > x->max) x->max = x->ch[1]->max;

}

inline void rotate(node\* x,int t) {

node\* y = x->fa,\*z = y->fa;

y->ch[t] = x->ch[t^1];

x->ch[t^1]->fa = y;

x->ch[t^1] = y;

if (y == z->ch[0]) z->ch[0] = x;

if (y == z->ch[1]) z->ch[1] = x;

y->fa = x;

x->fa = z;

update(y);

}

inline void splay(node\* x) {

node\* y,\*z;

down(x);

int t1,t2;

while (((y = x->fa) != null) && (y->ch[0] == x || y->ch[1] == x)) {

if (((z = y->fa) != null) && (z->ch[0] == y || z->ch[1] == y)) {

down(z);

down(y);

down(x);

t1 = x == y->ch[0] ? 0 : 1;

t2 = y == z->ch[0] ? 0 : 1;

if (t1 == t2) rotate(y,t1),rotate(x,t1);

else rotate(x,t1),rotate(x,t2);

} else down(y),down(x),rotate(x,x == y->ch[0] ? 0 : 1);

}

update(x);

}

inline node\* access(node\* x) {

node\* p;

for (p = null; x != null; x = x->fa) {

splay(x);

x->ch[1] = p;

update(p = x);

}

return p;

}

inline void evert(node\* x) {

access(x)->rev ^= 1;

splay(x);

}

inline node\* root(node\* x) {

for (x = access(x); down(x),x->ch[0] != null; x = x->ch[0]) ;

return x;

}

inline bool link(node\* x,node\* y) {

if (root(x) == root(y)) return false;

evert(x);

x->fa = y;

access(x);

return true;

}

inline bool cut(node\* x,node\* y) {

if (x == y || root(x) != root(y)) return false;

evert(x);

access(y);

splay(y);

y->ch[0]->fa = null;

y->ch[0] = null;

update(y);

return true;

}

inline bool modify(node\* x,node\* y) {

if (root(x) != root(y)) return false;

evert(x);

access(y);

splay(y);

y->add(w);

return true;

}

inline bool getmax(node\* x,node \* y,int& max) {

if (root(x) != root(y)) return false;

evert(x);

access(y);

splay(y);

max = y->max;

return true;

}

int main() {

freopen("4010.in","r",stdin);

freopen("4010.out","w",stdout);

while (scanf("%d",&n) != EOF) {

TC = 0;

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

scanf("%d%d",&e[i][0],&e[i][1]);

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

scanf("%d",&v);

a[i] = T + (++TC);

a[i]->fa = a[i]->ch[0] = a[i]->ch[1] = null;

a[i]->max = a[i]->v = v;

a[i]->rev = a[i]->del = 0;

}

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

node\* x = a[e[i][0]],\*y = a[e[i][1]];

link(x,y);

}

scanf("%d",&m);

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

scanf("%d",&t);

if (t != 3) {

scanf("%d%d",&X,&Y);

x = a[X];

y = a[Y];

}

if (t == 1) {

if (!link(x,y)) printf("-1\n");

} else if (t == 2) {

if (!cut(x,y)) printf("-1\n");

} else if (t == 3) {

scanf("%d%d%d",&w,&X,&Y);

x = a[X];

y = a[Y];

if (!modify(x,y)) printf("-1\n");

} else if (t == 4) {

if (!getmax(x,y,ans)) printf("-1\n");

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

}

}

printf("\n");

}

fclose(stdin);

fclose(stdout);

}

#### 网上的模板

#include <cstdio>

#include <algorithm>

#include <iostream>

#include <string.h>

#include <stdio.h>

const int maxn=100011;

const int inf=0x7fffffff;

using namespace std;

struct SplayTree

{

int val,mn,lazy;

bool remark;

int ch[2],pre;

};

SplayTree \*tree;

int N;

int val[maxn];

void Init\_Splay(int x)

{

tree[x].ch[0]=tree[x].ch[1]=tree[x].pre=0;

tree[x].remark=0;

tree[x].val=val[x];

tree[x].mn=val[x];

}

bool IsRoot(int x)

{

return !tree[x].pre || (tree[tree[x].pre].ch[0]!=x && tree[tree[x].pre].ch[1]!=x);

}

void DynamicTree(int n) // init()

{

N=n;

tree=new SplayTree[n+1];

for(int i=0; i<=n; i++) Init\_Splay(i);

tree[0].val=-inf;

tree[0].mn=-inf;

}

void Inc(int x,int d)

{

tree[x].val+=d;

tree[x].mn+=d;

tree[x].lazy+=d;

}

void Rev(int x)

{

swap(tree[x].ch[0],tree[x].ch[1]);

tree[x].remark^=1;

}

void PushDown(int x)

{

if(!x) return;

if(tree[x].lazy)

{

if(tree[x].ch[0]) Inc(tree[x].ch[0],tree[x].lazy);

if(tree[x].ch[1]) Inc(tree[x].ch[1],tree[x].lazy);

tree[x].lazy=0;

}

if(tree[x].remark)

{

if(tree[x].ch[0]) Rev(tree[x].ch[0]);

if(tree[x].ch[1]) Rev(tree[x].ch[1]);

tree[x].remark=0;

}

}

void Update(int x)

{

if(!x) return;

tree[x].mn=tree[x].val;

if(tree[x].ch[0]) tree[x].mn=max(tree[tree[x].ch[0]].mn,tree[x].mn);

if(tree[x].ch[1]) tree[x].mn=max(tree[tree[x].ch[1]].mn,tree[x].mn);

}

void Rotate(int p,int c)

{

int x=tree[p].pre,y=tree[x].pre;

tree[p].pre=y;

tree[x].pre=p;

if(y) if(x==tree[y].ch[0]) tree[y].ch[0]=p;

else if(x==tree[y].ch[1]) tree[y].ch[1]=p;

tree[x].ch[!c]=tree[p].ch[c];

if(tree[x].ch[!c]) tree[tree[x].ch[!c]].pre=x;

tree[p].ch[c]=x;

Update(x);

}

int stack[maxn];

void Splay(int x)

{

int top=1;

stack[0]=x;

for(int q=x; !IsRoot(q);) stack[top++]=(q=tree[q].pre);

while(top) PushDown(stack[--top]);

while(!IsRoot(x))

{

int q=tree[x].pre;

if(IsRoot(q)) if(tree[q].ch[0]==x) Rotate(x,1);

else Rotate(x,0);

else

{

if(q==tree[tree[q].pre].ch[0])

if(tree[q].ch[0]==x) Rotate(q,1),Rotate(x,1);

else Rotate(x,0),Rotate(x,1);

else if(x==tree[q].ch[1]) Rotate(q,0),Rotate(x,0);

else Rotate(x,1),Rotate(x,0);

}

}

Update(x);

}

int Head(int x)

{

Splay(x);

for(PushDown(x); tree[x].ch[0]; x=tree[x].ch[0]) PushDown(x);

Splay(x);

return x;

}

int Expose(int x) // x到root的边全部调整为实边

{

int y;

for(y=0; x; x=tree[x].pre) Splay(x),PushDown(x),tree[x].ch[1]=y,Update(y=x);

return y;

}

void ChangeRoot(int x)

{

Rev(Expose(x));

}

void Change(int x,int y,int val)

{

ChangeRoot(y);

Expose(x);

Splay(x);

tree[x].val+=val;

tree[x].lazy+=val;

tree[x].mn+=val;

// PushDown(x);

// Update(x);

}

int AskMax(int x,int y)

{

ChangeRoot(x);

Expose(y);

Splay(y);

return tree[y].mn;

}

void Link(int x,int y)//link操作即为链接两个树，那么要先进性expose操作，把到路径上的边都变为实边，这样才能进行把x调整到根部，进而通过更改祖先来进行链接

{

ChangeRoot(x);

Splay(x);

tree[x].pre=y;

}

void Cut(int x,int y)

{

ChangeRoot(y);

Splay(x);

if(tree[x].ch[0])

{

tree[tree[x].ch[0]].pre=tree[x].pre;

tree[x].pre=tree[x].ch[0]=0;

}

else tree[x].pre=0;

}

int LCA(int x,int y)

{

int p=Head(Expose(x));

int q=Expose(y),w=Head(q);

if(p==w) return q;

return 0;

}

struct data

{

int x,y;

} a[maxn];

int main()

{

// freopen("in","r",stdin);

// freopen("out","w",stdout);

int n,m;

while(scanf("%d",&n)==1)

{

for(int i=1; i<n; i++) scanf("%d%d",&a[i].x,&a[i].y);

val[0]=val[n+1]=-inf;

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

DynamicTree(n+1);

for(int i=1; i<n; i++) Link(a[i].x,a[i].y);

scanf("%d",&m);

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

{

int c;

int x,y,val;

scanf("%d",&c);

if(c==1)

{

scanf("%d%d",&x,&y);

if(!LCA(x,y)) Link(x,y);

else printf("-1\n");

}

else if(c==2)

{

scanf("%d%d",&x,&y);

if(LCA(x,y) && x!=y) Cut(y,x);

else printf("-1\n");

}

else if(c==3)

{

scanf("%d%d%d",&val,&x,&y);

if(LCA(x,y)) Change(x,y,val);

else printf("-1\n");

}

else if(c==4)

{

scanf("%d%d",&x,&y);

int tmp=LCA(x,y);

if(tmp) printf("%d\n",AskMax(x,y));

else printf("-1\n");

}

}

printf("\n");

}

return 0;

}

#### 链表 适合边权

namespace lct {

const int N = maxn;

struct node {

node \*c[2], \*p;

int w, mx;

int type() {

if (!p || p->c[0] != this && p->c[1] != this) return 2;

return p->c[0] != this;

}

node \*update() {

mx = w;

for (int i = 0; i < 2; i++)

if (c[i]) mx = max(mx, c[i]->mx);

return this;

}

node \*rotate() {

node \*y = p;

int b = type(), pb = y->type();

if (p = y->p, pb < 2) p->c[pb] = this;

if (y->c[b] = c[!b]) c[!b]->p = y;

return c[!b] = y, y->update()->p = this;

}

node \*splay() {

for ( ; type() < 2; type() < 2? rotate(): 0)

type() == p->type()? p->rotate(): rotate();

return update();

}

node \*end(int b) {

node \*x = this;

for ( ; x->c[b]; x = x->c[b]) ;

return x;

}

} pool[N], \*top;

node \*init() {

return top = pool;

}

node \*make\_tree() {

return &(\*top++ = (node) {

0, 0, 0, -inf32, -inf32

});

}

node \*access(node \*x) {

for (node \*y = x, \*z = 0; y; z = y, y = y->update()->p)

y->splay()->c[1] = z;

return x;

}

node \*find\_root(node \*x) {

return access(x)->splay()->end(0)->splay();

}

node \*cut(node \*x) {

access(x);

x->c[0] = 0;

return x;

}

node \*join(node \*x, node \*y) {

access(x)->p = y;

return access(x);

}

node \*ask(node \*x, int &ans) {

for (node \*y = x, \*z = 0; y; z = y, y = y->update()->p) {

y->splay();

if (!y->p) {

ans = -inf32;

if (y->c[1]) ans = max(ans, y->c[1]->mx);

if (z) ans = max(ans, z->mx);

}

y->c[1] = z;

}

return x;

}

}

struct edge {

int v, to, i, w;

};

vector<edge> E;

int L[maxn];

void graph\_inti() {

E.clear();

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

}

void graph\_add(int u, int v, int i, int w) {

edge t = {v, L[u], i, w};

L[u] = E.size();

E.push\_back(t);

}

int n;

lct::node \*rt[maxn], \*etr[maxn];

void travel(int u = 0, int p = -1) {

rt[u] = lct::make\_tree();

if (p != -1) {

etr[E[p].i] = rt[u];

rt[u]->w = E[p].w;

lct::join(rt[u], rt[E[p].v]);

}

for (int i = L[u]; i != -1; i = E[i].to)

if (i != p) travel(E[i].v, i^1);

}

int main() {

#if 0

freopen("input.in", "r", stdin);

#endif

for (int T = (scanf("%d", &T), T); T--; ) {

scanf("%d", &n);

graph\_inti();

for (int u, v, w, i = 0; i < n-1; i++) {

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

u--, v--;

graph\_add(u, v, i, w);

graph\_add(v, u, i, w);

}

lct::init();

travel();

char op[10];

for (int x, y; scanf("%s", op), op[0] != 'D'; ) {

scanf("%d%d", &x, &y);

if (op[0] == 'C') {

etr[x-1]->w = y;

access(etr[x-1]);

} else {

int ans;

lct::access(rt[x-1]);

lct::ask(rt[y-1], ans);

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

}

}

}

return 0;

}

// 边权

#### Acforfun

//变量：

int ch[maxn][2],fa[maxn],M[maxn],key[maxn],lz[maxn];

bool rev[maxn],is[maxn];//is[]表示该点是否为splay的根

//标记下放函数：

void ad(int n,int k)

{

if(!n)

return;

lz[n]+=k,key[n]+=k,M[n]+=k;

}

void push(int n)

{

if(rev[n])

{

rev[n]=0;swap(ch[n][0],ch[n][1]);

rev[ch[n][0]]^=1,rev[ch[n][1]]^=1;

}

if(lz[n])

{

int t=lz[n];lz[n]=0;

ad(ch[n][0],t);

ad(ch[n][1],t);

}

}

//更新函数：

void flow(int n)

{

M[n]=max(max(M[ch[n][0]],M[ch[n][1]]),key[n]);

}

//旋转：

inline void rot(int n)

{

int t=fa[n],isr=ch[t][1]==n;

ch[t][isr]=ch[n][!isr],fa[ch[t][isr]]=t;

fa[n]=fa[t],fa[t]=n,ch[n][!isr]=t;

if(is[t])

is[t]=0,is[n]=1;

else

ch[fa[n]][ch[fa[n]][1]==t]=n;

flow(t);

}

//splay：注意此处有个P函数，涵义是在splay之前先将根节点到n的路径上所有节点的标记逐级下放，以确保此时从n到根不断旋转的路径上不会有标记（对带标记节点进行旋转是会产生错误的）。另一种方法是在splay的过程中对fa[fa[n]],fa[n],n逐级下放，但是我觉得一次性下放好更直观。

void P(int n)

{

if(!is[n])P(fa[n]);

push(n);

}

void splay(int n)

{

P(n);

int f,ff;

while(!is[n])

{

f=fa[n],ff=fa[f];

if(is[f])

rot(n);

else if((ch[ff][1]==f)==(ch[f][1]==n))

rot(f),rot(n);

else

rot(n),rot(n);

}

flow(n);

}

//access：和大多数人的access一样，没什么特别之处。在access里面直接嵌套个splay(v)是不必要的。

int access(int n)

{

int v=0;

for(;n;n=fa[v=n])

{

splay(n);

is[ch[n][1]]=1,is[ch[n][1]=v]=0;

flow(n);

}

return v;

}

//jud：判断是否同根（真实的树，非splay）

// mroot：使n成为它所在的树的跟：

int jud(int u,int v)

{

while(fa[u])u=fa[u];

while(fa[v])v=fa[v];

return u==v;

}

void mroot(int n)

{

access(n),splay(n);

rev[n]^=1,push(n);

}

// lca(int &u,int &v)：调用后u是原来u和v的lca，v和ch[u][1]分别存着lca的2个儿子（原来u和v所在的2棵子树）。

// 则路径信息存在子树v，子树ch[u][1]和结点u中。

inline void lca(int &u,int &v)

{

access(v),v=0;

while(u)

{

splay(u);

if(!fa[u])

return;

is[ch[u][1]]=1,is[ch[u][1]=v]=0;

flow(u);u=fa[v=u];

}

}

//link(u,v):使u成为u所在树的根并且v为它的父亲

void link(int u,int v)

{

if(jud(u,v))

{

puts("-1");

return;

}

mroot(u);

fa[u]=v;

}

//void cut(int u,int v)：使u成为u所在树的根，并且v和它父亲的边断开

void cut(int u,int v)

{

if(u==v||!jud(u,v))

{

puts("-1");

return;

}

mroot(u);

splay(v);

fa[ch[v][0]]=fa[v],fa[v]=0;is[ch[v][0]]=1;

ch[v][0]=0,flow(v);

}

//区间增加和查询，这个很无聊了：

void add(int u,int v,int k)

{

if(!jud(u,v))

{

puts("-1");

return;

}

lca(u,v);

ad(ch[u][1],k),ad(v,k);

key[u]+=k,flow(u);

}

void query(int u,int v)

{

if(!jud(u,v))

{

puts("-1");

return;

}

lca(u,v);

printf("%d\n",max(max(M[v],M[ch[u][1]]),key[u]));

}

//主函数什么的：

int first[maxn],nxt[maxn<<1],vv[maxn<<1];

void dfs(int n)

{

for(int e=first[n];e;e=nxt[e])if(!fa[vv[e]])

fa[vv[e]]=n,dfs(vv[e]);

}

int main()

{

int n,q,u,v,e,i,j;

while(scanf("%d",&n)!=EOF)

{

memset(first,0,sizeof(first)),e=2;

memset(fa,0,sizeof(fa));

memset(ch,0,sizeof(ch));

memset(rev,0,sizeof(rev));

memset(lz,0,sizeof(lz));

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

M[0]=-2100000000;

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

{

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

nxt[e]=first[u],vv[e]=v,first[u]=e++;

nxt[e]=first[v],vv[e]=u,first[v]=e++;

}

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

scanf("%d",&key[i]),M[i]=key[i];

scanf("%d",&q);

fa[1]=-1,dfs(1);fa[1]=0;

while(q--)

{

scanf("%d",&i);

if(i==3)

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

else

{

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

if(i==1)

link(u,v);

else if(i==2)

cut(u,v);

else

query(u,v);

}

}

puts("");

}

}

#### sy

struct node

{

node \*p,\*ch[2];

int mx,rev,val,add;

}nodes[N],\*cur,\*null;

int n,m,u,v,w;

node \*newnode(int key)

{

cur->p=cur->ch[0]=cur->ch[1]=null;

cur->mx=cur->val=key;

cur->rev=0;

return cur++;

}

void init()

{

null=nodes;

null->p=null->ch[0]=null->ch[1]=null;

null->mx=null->val=-INF;

null->add=0;

null->rev=0;

cur=nodes+1;

}

struct dynamictree

{

bool isroot(node \*x)//判根

{

return x==null || x->p->ch[0]!=x && x->p->ch[1]!=x;

}

void pushup(node \*x)

{

x->mx=max(x->val,max(x->ch[0]->mx,x->ch[1]->mx));

}

void pushdown(node \*x)

{

if(x==null) return;

if(x->rev)

{

x->rev=0;

if(x->ch[0]!=null) x->ch[0]->rev^=1;

if(x->ch[1]!=null) x->ch[1]->rev^=1;

swap(x->ch[0],x->ch[1]);

}

if(x->add)

{

if(x->ch[0]!=null) x->ch[0]->add+=x->add,x->ch[0]->val+=x->add,x->ch[0]->mx+=x->add;

if(x->ch[1]!=null) x->ch[1]->add+=x->add,x->ch[1]->val+=x->add,x->ch[1]->mx+=x->add;

x->add=0;

}

}

void rotate(node \*x,int f)

{

if(isroot(x)) return;

node \*y=x->p;

y->ch[!f]=x->ch[f];

x->p=y->p;

if(x->ch[f]!=null) x->ch[f]->p=y;

if(y!=null)

{

if(y==y->p->ch[1]) y->p->ch[1]=x;

else if(y==y->p->ch[0]) y->p->ch[0]=x;

}

x->ch[f]=y;

y->p=x;

pushup(y);

}

void splay(node \*x)

{

static node \*sta[N];

int top=1;

sta[0]=x;

for(node \*y=x;!isroot(y);y=y->p)

sta[top++]=y->p;

while (top) pushdown(sta[--top]);

while (!isroot(x))

{

node \*y=x->p;

if(isroot(y)) rotate(x,x==y->ch[0]);

else

{

int f=y->p->ch[0]==y;

if(y->ch[f]==x) rotate(x,!f);

else rotate(y,f);

rotate(x,f);

}

}

pushup(x);

}

node \*access(node \*u)

{

node \*v=null;

while (u!=null)

{

splay(u);

v->p=u;

u->ch[1]=v;

pushup(u);

v=u;

u=u->p;

}

return v;

}

node \*link(node \*u,node \*v)//合并

{

access(u);

splay(u);

u->rev=1;

u->p=v;

}

node \*cut(node \*u)//分离

{

access(u);

splay(u);

u->ch[0]=u->ch[0]->p=null;

pushup(u);

}

void changeroot(node \*u)//换根

{

access(u)->rev^=1;

}

node \*getroot(node \*u)//找根

{

access(u);

splay(u);

while (u->p!=null) u=u->p;

splay(u);

return u;

}

bool queryuv(node \*u,node \*v)//判断是否在同一子树

{

while (u->p!=null) u=u->p;

while (v->p!=null) v=v->p;

return u==v;

}

}splay;

int eu[N],ev[N];

int main ()

{

while (scanf("%d",&n)!=-1)

{

init();

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

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

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

{

int a;

scanf("%d",&a);

newnode(a);

}

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

splay.link(nodes+eu[i],nodes+ev[i]);

scanf("%d",&m);

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

{

scanf("%d",&u);

if(u==1)

{

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

if(splay.queryuv(nodes+u,nodes+v))

{

printf("-1\n");

continue;

}

splay.link(nodes+u,nodes+v);

}

else if(u==2)

{

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

if(u==v || !splay.queryuv(nodes+u,nodes+v))

{

printf("-1\n");

continue;

}

splay.changeroot(nodes+u);

splay.cut(nodes+v);

}

else if(u==3)

{

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

if(! splay.queryuv(nodes+u,nodes+v))

{

printf("-1\n");

continue;

}

splay.changeroot(nodes+u);

splay.access(nodes+v);

node \*q=splay.getroot(nodes+v);

q->add+=w;

q->mx+=w;

q->val+=w;

}

else

{

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

if(! splay.queryuv(nodes+u,nodes+v))

{

printf("-1\n");

continue;

}

splay.changeroot(nodes+u);

splay.access(nodes+v);

printf("%d\n",splay.getroot(nodes+v)->mx);

}

}

printf("\n");

}

return 0;

}

## 可持久化数据结构

### 可持久化线段树

#define maxn 100105

struct node {

int l,r;

int cnt;

void init(int ll,int rr,int cc) {

l = ll;

r = rr;

cnt = cc;

}

};

node tt[maxn \* 30];

int t[maxn ];

int a[maxn];

int v[maxn];

int n,q;

int m;

int tot;

void build(int &root ,int l ,int r ) {

root = tot ++ ;

tt[root].init(0,0,0);

if(l == r ) return ;

int mid = (l + r ) / 2 ;

build(tt[root].l,l,mid);

build(tt[root].r,mid + 1 , r);

}

void ins(int &root ,int p ,int l ,int r , int x) {

root = tot ++ ;

tt[root].init(tt[p].l,tt[p].r,tt[p].cnt + 1 );

if(l == r ) return ;

int mid = (l + r ) / 2 ;

if(mid >= x ) ins(tt[root].l,tt[p].l, l , mid , x);

else ins(tt[root].r,tt[p].r, mid + 1 , r , x );

}

int query(int lroot,int rroot,int l, int r ,int k) {

if(l == r ) return l ;

int mid = (l + r ) / 2 ;

int lcnt = tt[tt[rroot].l].cnt - tt[tt[lroot].l].cnt;

if(lcnt >= k )

query(tt[lroot].l,tt[rroot].l , l , mid , k );

else

query(tt[lroot].r,tt[rroot].r , mid + 1 , r , k - lcnt );

}

int main() {

int cas;

scanf("%d",&cas);

while(cas -- ) {

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

for(int i = 1 ; i <= n ; i ++ ) scanf("%d",&a[i]),v[i-1] = a[i];

sort(v ,v + n );

m = unique(v , v + n ) - v ;

//printf("m = %d\n",m);

memset(t,0,sizeof(t));

tot = 0 ;

build(t[0],1,m);

int pos;

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

pos = lower\_bound(v ,v + m ,a[i]) - v + 1 ;

//printf("pos = %d\n",pos);

ins(t[i],t[i-1],1,m,pos);

}

int x,y,k;

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

scanf("%d%d%d",&x,&y,&k);

pos = query(t[x-1],t[y],1,m,k);

printf("%d\n",v[pos-1]);

}

}

return 0;

}

## 几何

### 二维point类

struct pt

{

double x, y;

pt(){}

pt(double \_x, double \_y):x(\_x), y(\_y){}

pt operator - (const pt p1){return pt(x - p1.x, y - p1.y);}

pt operator + (const pt p1){return pt(x + p1.x, y + p1.y);}

pt operator \* (double s){return pt(x \* s, y \* s);}

pt operator / (double s){return pt(x / s, y / s);}

bool operator < (const pt p1)const{return y < p1.y-eps || y < p1.y+eps && x < p1.x;}

bool operator == (const pt p1)const{return !SGN(y - p1.y) && !SGN(x - p1.x);}

bool operator != (const pt p1)const{return SGN(y - p1.y) || SGN(x - p1.x);}

void read(){scanf("%lf %lf", &x, &y);}

};

### 扫描线\*