1. 快读

struct FastIO {

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;

1. ZKW线段树

int sum[maxn<<2],add[maxn<<2],mx[maxn<<2],mn[maxn<<2],w[maxn],N;

inline void build(int n)//建树

{

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

for(int i=1;i<=N+n;i++) add[i]=0;

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

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

mn[i]=mx[i]=sum[i];

}

for(int i=N-1;i;i--){

sum[i]=sum[i<<1]+sum[i<<1|1];

mn[i]=min(mn[i<<1],mn[i<<1|1]),mn[i<<1]-=mn[i],mn[i<<1|1]-=mn[i];

mx[i]=max(mx[i<<1],mx[i<<1|1]),mx[i<<1]-=mx[i],mx[i<<1|1]-=mx[i];

}

return;

}

inline void update(int x,int k)

{

x+=N;mx[x]+=k;mn[x]+=k;sum[x]+=k;

int A;

for(;x>1;x>>=1){

sum[x>>1]+=k;

A=min(mn[x],mn[x^1]);

mn[x]-=A;mn[x^1]-=A;mn[x>>1]+=A;

A=max(mx[x],mx[x^1]);

mx[x]-=A;mx[x^1]-=A;mx[x>>1]+=A;

}

return;

}

inline void update(int s,int t,int k)//维护区间修改

{

int lc=0,rc=0,num=1,A;

for(s+=N-1,t+=N+1;s^t^1;s>>=1,t>>=1,num<<=1){

if(~s&1) add[s^1]+=k,sum[s^1]+=k\*num,mn[s^1]+=k,mx[s^1]+=k,lc+=num;

if(t&1) add[t^1]+=k,sum[t^1]+=k\*num,mn[t^1]+=k,mx[t^1]+=k,rc+=num;

sum[s>>1]+=k\*lc;sum[t>>1]+=k\*rc;

A=min(mn[s],mn[s^1]),mn[s]-=A,mn[s^1]-=A,mn[s>>1]+=A;

A=max(mx[s],mx[s^1]),mx[s]-=A,mx[s^1]-=A,mx[s>>1]+=A;

A=min(mn[t],mn[t^1]),mn[t]-=A,mn[t^1]-=A,mn[t>>1]+=A;

A=max(mx[t],mx[t^1]),mx[t]-=A,mx[t^1]-=A,mx[t>>1]+=A;

}

for(lc+=rc;s>1;s>>=1){

sum[s>>1]+=k\*lc;

A=min(mn[s],mn[s^1]),mn[s]-=A,mn[s^1]-=A,mn[s>>1]+=A;

A=max(mx[s],mx[s^1]),mx[s]-=A,mx[s^1]-=A,mx[s>>1]+=A;

}

return;

}

inline int query\_sum(int s,int t)

{

int lc=0,rc=0,num=1,ans=0;

for(s+=N-1,t+=N+1;s^t^1;s>>=1,t>>=1,num<<=1){

if(add[s]) ans+=add[s]\*lc;

if(add[t]) ans+=add[t]\*rc;

if(~s&1) ans+=sum[s^1],lc+=num;

if(t&1) ans+=sum[t^1],rc+=num;

}

for(;s;s>>=1,t>>=1){

ans+=add[s]\*lc+add[t]\*rc;

}

return ans;

}

inline int query\_min(int s,int t)

{

int sans=0,tans=0;

s+=N,t+=N;

if(s!=t){

for(;s^t^1;s>>=1,t>>=1){

sans+=mn[s],tans+=mn[t];

if(~s&1) sans=min(sans,mn[s^1]);

if(t&1) tans=min(tans,mn[t^1]);

}

}

int ans=min(sans+mn[s],tans+mn[t]);

while(s>1) ans+=mn[s>>=1];

return ans;

}

inline int query\_max(int s,int t)

{

int sans=0,tans=0;

s+=N,t+=N;

if(s!=t){

for(;s^t^1;s>>=1,t>>=1){

sans+=mx[s],tans+=mx[t];

if(~s&1) sans=max(sans,mx[s^1]);

if(t&1) tans=max(tans,mx[t^1]);

}

}

int ans=max(sans+mx[s],tans+mx[t]);

while(s>1) ans+=mx[s>>=1];

return ans;

}

1. K短路（A\*）O(nk)

inline void dijkstra(int s)

{

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

dis[i]=INF;

vis[i]=0;

}

priority\_queue<Node>Q;

Q.push(Node(s,dis[s]=0));

while(!Q.empty())

{

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

if(vis[u]) continue;

vis[u]=1;

for(int i=head1[u];i!=-1;i=edge[i].next){

int v=edge[i].to;

if(!vis[v]&&dis[v]>dis[u]+edge[i].w){

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

Q.push(Node(v,dis[v]));

}

}

}

return;

}

inline int Astar(int s,int t,int k)

{

if(dis[s]>=INF||k==1) return dis[s];

priority\_queue<Node>Q;

Q.push(Node(s,dis[s]));

int u,d;

while(!Q.empty())

{

u=Q.top().u,d=Q.top().d;Q.pop();

if(u==t){

if(--k==0) break;

continue;

}

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

int v=edge[i].to;

Q.push(Node(v,d+edge[i].w+dis[v]-dis[u]));

}

}

return d;

}

1. K短路（可持久化堆）O(nlogn+mlogm+klogk)

const int maxn=1010;

const int maxm=20010;

const int INF=0x3fffffff;

struct Node

{

int u,d;

Node(int x,int y):u(x),d(y){}

bool operator<(const Node&p)const

{

return d>p.d;

}

};

struct E

{

int to,next,w;

}edge[maxm];

int head[maxn],head1[maxn],tol;//正向反向边

int d[maxn],pre[maxn];

bool vis[maxn];//最短路建最短路径树部分

int s,t,K,T,n,m;//起点、终点、第K短

inline void Addedge(int u,int v,int w)

{

edge[tol].to=v;edge[tol].w=w;edge[tol].next=head[u];head[u]=tol++;

edge[tol].to=u;edge[tol].w=w;edge[tol].next=head1[v];head1[v]=tol++;

}

inline void dijsktra()

{

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

d[i]=INF;

vis[i]=0;

pre[i]=-1;

}

priority\_queue<Node>Q;

Q.push(Node(t,d[t]=0));

while(!Q.empty())

{

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

if(vis[u]) continue;

vis[u]=1;

for(int i=head1[u];i!=-1;i=edge[i].next){

int v=edge[i].to;

if(!vis[v]&&d[v]>d[u]+edge[i].w){

pre[v]=u;

d[v]=d[u]+edge[i].w;

Q.push(Node(v,d[v]));

}

}

}

return;

}

int rt[maxn],ls[maxn\*100],rs[maxn\*100],dis[maxn\*100];

int to[maxn\*100],key[maxn\*100],tot;//左偏树部分，to是非树边e的弧尾

inline int merge(int x,int y)//可持久化左偏树

{

if(!x||!y) return x+y;

if(key[x]>key[y]) swap(x,y);

int r=++tot;

ls[r]=ls[x];key[r]=key[x];to[r]=to[x];

rs[r]=merge(rs[x],y);

if(dis[ls[r]]<dis[rs[r]]) swap(ls[r],rs[r]);

dis[r]=dis[rs[r]]+1;

return r;

}

inline void dfs(int u)//O(mlogm)建堆

{

if(pre[u]!=-1) rt[u]=rt[pre[u]];

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

int v=edge[i].to;

if(v==pre[u]) continue;

int r=++tot;

ls[r]=rs[r]=dis[r]=0;key[r]=d[v]+edge[i].w-d[u];to[r]=v;

//非树边边权为增量

rt[u]=merge(rt[u],r);

}

for(int i=head1[u];i!=-1;i=edge[i].next){

int v=edge[i].to;

if(pre[v]==u) dfs(v);

}

return;

}

inline int solve()

{

dijsktra();

if(d[s]==INF) return -1;

if(K==1) return d[s];

--K;

rt[t]=tot=0;//记得初始化

dfs(t);

priority\_queue<Node>Q;

Q.push(Node(rt[s],d[s]+key[rt[s]]));

while(!Q.empty())

{

int u=Q.top().u,v=to[u],c=Q.top().d;Q.pop();

if(--K==0) return c;

if(rt[v]) Q.push(Node(rt[v],c+key[rt[v]]));//加新边

if(ls[u]) Q.push(Node(ls[u],c-key[u]+key[ls[u]]));//替换最后一条边

if(rs[u]) Q.push(Node(rs[u],c-key[u]+key[rs[u]]));

}

return -1;

}

1. 仙人掌图的判定

/\*性质 1 仙人掌图的 DFS 树没有横向边

性质 2 Low(u)<=DFS(v) (u 是 v 的儿子)

性质 3 设某个点 v 有 a(v)个儿子的 Low 值小于 DFS(v)，同时 v 自己有 b(v)条逆向边。

那么 a(v)+b(v)<2

\*/

inline void Tarjan(int u)

{

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

stk[top++]=u;

int v,sum=0;

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

v=edge[i].to;

if(vis[v]) ok=0;//仙人掌图无横向边

if(!dfn[v]){

Tarjan(v);

if(low[v]<low[u]) low[u]=low[v];

if(low[v]<dfn[u]&&++sum>=2) ok=0;//性质3

}

else if(!bel[v]&&dfn[v]<dfn[u]){

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

if(++sum>=2) ok=0;//性质3

}

}

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

++blocks;

do

{

v=stk[--top];

bel[v]=blocks;

}while(v!=u);

}

vis[u]=1;

return;

}