/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

kruskal

O(ElogE) ~~(?)~~

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <algorithm>

using namespace std;

const int MAX = 1e5 + 5;

struct edge{

int u, v, w;

edge(int u\_ = 0, int v\_ = 0, int w\_ = 0) : u(u\_), v(v\_), w(w\_) {}

bool operator< (const edge b)const

{

return w < b.w;

}

};// 存边

int n, m;

edge e[MAX];

int pre[MAX]; //并查集

void init()

{

ans = 0;

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

pre[i] = i;

}

int find(int x) // 并查集路径压缩

{

return pre[x] == x ? x : pre[x] = find(pre[x]);

}

int kruskal()

{

init();

sort(e + 1, e + 1 + m); // 边集按权值排序

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

{

int a = find(e[i].u), b = find(e[i].v);

if (a != b) // 不在同一树上， 需合并

{

pre[b] = a;

ans += e[i].w;

}

}

return ans;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Prim(朴素)

O(VE)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <string.h>

const int MAX = 1e5 + 5;

struct edge{

int v, w;

int next;

edge(int v\_ = 0, int w\_ = 0, int n\_ = -1) : v(v\_), w(w\_), next(n\_){}

};

int n, cnt;

edge e[MAX << 1]; //需存双向边

int head[MAX];

int dis[MAX];

void init()

{

cnt = 0;

memset(head, -1, sizeof head);

}

void addedge(int u, int v, int w)

{

e[cnt++] = edge(v, w, head[u]); head[u] = cnt++;

e[cnt++] = edge(u, w, head[v]); head[v] = cnt++;

}

int prim()

{

/\* 初始化dis数组 \*/

memset(dis, 0x3f3f3f3f, sizeof dis);

dis[1] = 0;

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

dis[e[i].v] = e[i].w;

int ans = 0; // answer

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

{

int tmp = 0x3f3f3f3f, ind = -1;

for (int i = 1; i <= n; i++) // 查找当前最近点

if (dis[i] && dis[i] <= tmp)

{

tmp = dis[i];

ind = i;

}

ans += tmp; //更新answer

dis[ind] = 0;

for (int i = head[ind]; i != -1; i = e[i].next) // 更新dis

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

}

return ans;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Prim(Heap)

O(ElogV)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <string.h>

#include <algorithm>

#include <queue>

using namespace std;

const int MAX = 1e5 + 5;

struct edge{

int v, w;

int next;

edge(int v\_ = 0, int w\_ = 0, int n\_ = -1) : v(v\_), w(w\_), next(n\_) {}

};

int n, cnt;

edge e[MAX << 1]; //需存双向边

int head[MAX];

int dis[MAX];

bool vis[MAX];

void init()

{

cnt = 0;

memset(head, -1, sizeof head);

}

void addedge(int u, int v, int w)

{

e[cnt++] = edge(v, w, head[u]); head[u] = cnt++;

e[cnt++] = edge(u, w, head[v]); head[v] = cnt++;

}

struct node{

int u, w;

node(int u\_ = 0, int w\_ = 0) : u(u\_), w(w\_) {}

bool operator< (const node b)const

{

return w > b.w;

}

}; // 优先队列用

int prim()

{

memset(dis, 0x3f3f3f3f, sizeof dis);

memset(vis, 0, sizeof vis);

priority\_queue<node> q;

q.push(node(1, 0));

dis[1] = 0;

int ans = 0;

while (!q.empty())

{

int u = q.top().u;

q.pop();

if (vis[u])

continue;

ans += dis[u];

vis[u] = 1; dis[u] = 0;

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

{

if (!vis[e[i].v] && dis[e[i].v] > e[i].w)

{

dis[e[i].v] = e[i].w;

q.push(node(e[i].v, dis[e[i].v]));

}

}

}

return ans;

}