K短路有两种方法，一种是A\*，一种是可持久化可并堆，前面的一种相对好写，但是有极小的概率被卡，后面的方法复杂度正确但是相对代码要多一点

1.A\*

给你一张图，起点为1，终点为n 。求从1到n的次短路。（这里每条边可以重复走）

#include<bits/stdc++.h>

#define CL(a,num) memset((a),(num),sizeof(a))

#define ll long long

#define inf 1e18

#define M 200004

#define N 5007

using namespace std;

struct node

{

int v,w;

int next;

}g[M];

int head[N],ct;

int dis[N];

bool vt[N];

int n,m;

struct A\_node

{

int v,g;

bool operator < (A\_node b) const

{

return (g + dis[v]) > (b.g + dis[b.v]);

}

}p1,p2;

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

{

g[ct].v = v;

g[ct].w = w;

g[ct].next = head[u];

head[u] = ct++;

g[ct].v = u;

g[ct].w = w;

g[ct].next = head[v];

head[v] = ct++;

}

void spfa(int s)

{

int i;

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

{

dis[i] = inf;

vt[i] =false;

}

dis[s] = 0; vt[s] = true;

queue<int>q;

q.push(s);

while (!q.empty())

{

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

vt[u] = false;

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

{

int v = g[i].v;

int w = g[i].w;

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

{

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

if (!vt[v])

{

vt[v] = true;

q.push(v);

}

}

}

}

}

int cnt[N];

priority\_queue<A\_node> pq;

int A\_star(int s,int t,int k)//求出s的t的k短路

{

int i;

int ans = 0;

CL(cnt,0);

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

if (s == t) k++;//注意当s==t时需要计算K+1短路,因为s到t这条距离为0的路不能算在这K短路中,这时只需将K++

p1.g = 0; p1.v = s;

pq.push(p1);

while (!pq.empty())

{

p2 = pq.top(); pq.pop();

int u = p2.v;

int gi = p2.g;

cnt[u]++;

if (cnt[u] > k) continue;

if (cnt[t] == k)

{

ans = gi;

break;

}

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

{

int v = g[i].v;

int w = g[i].w;

p1.v = v; p1.g = gi + w;

pq.push(p1);

}

}

return ans;

}

int main()

{

int i;

int x,y,z;

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

{

CL(head,-1); ct = 0;

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

{

scanf("%d%d%d",&x,&y,&z);

add(x,y,z);

}

spfa(n);//从终点遍历反图

printf("%d\n",A\_star(1,n,2));//因为是次短路，所以k=2

}

return 0;

}

2.可持久化可并堆

求s到t的第k短路

struct Node{

int val,to;

Node \*left,\*right;

Node(){}

Node(int val, int to, Node \*left, Node \*right) : val(val), to(to), left(left), right(right) {}

};

#define Limit 1000000

Node pool[Limit];

Node \*top = pool;

Node \*newNode(int val, int to) {

if (top >= pool + Limit)

return new Node(val, to, NULL, NULL);

top->val = val;

top->to = to;

top->left = NULL;

top->right = NULL;

return top++;

}

Node \*meGTe(Node \*a, Node \*b) {

if (!a)

return b;

if (!b)

return a;

if (a->val > b->val)

swap(a, b);

Node \*p = newNode(a->val, a->to);

p->left = a->left;

p->right = a->right;

p->right = meGTe(p->right, b);

swap(p->left, p->right);

return p;

}

struct Status {

int dist;

Node \*p;

Status(){}

Status(int dist, Node \*p) : dist(dist), p(p) {}

bool operator<(Status b) const { return dist > b.dist; }

};

struct Edge {

int to, next;

int w;

Edge() {}

Edge(int to, int next, int w) : to(to), next(next), w(w) {}

};

struct Map {

int tot;

int \*head;

Edge \*edge;

Map() {}

Map(int n, int m) : tot(0) {

head = new int[(n + 1)];

edge = new Edge[(m + 5)];

memset(head, 0, sizeof(int) \* (n + 1));

}

void addEdge(int x, int y, int w) {

edge[++tot].to = y;

edge[tot].next = head[x];

edge[tot].w = w;

head[x] = tot;

}

Edge &operator[](int pos) { return edge[pos]; }

};

int n, m;

int s, t, k;

Map G, GT;

bool \*vis;

int \*dis, \*pre;

queue<int> SPFA(int S) {

vis = new bool[(n + 1)];

dis = new int[(n + 1)];

pre = new int[(n + 1)];

memset(dis, INF, sizeof(int) \* (n + 1));

memset(vis, false, sizeof(bool) \* (n + 1));

queue<int> Q;

Q.push(S);

dis[S] = 0;

pre[S] = 0;

while (!Q.empty()) {

int x = Q.front();

Q.pop();

vis[x] = false;

for (int i = GT.head[x]; i; i = GT[i].next) {

int y = GT[i].to;

int w = GT[i].w;

if (dis[x] + w < dis[y]) {

dis[y] = dis[x] + w;

pre[y] = i;

if (!vis[y]) {

vis[y] = true;

Q.push(y);

}

}

}

}

return Q;

}

Node \*\*Hash;

void rebuild(queue<int> Q) { //建堆

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

for (int j = G.head[i]; j; j = G[j].next) {

int to = G[j].to;

if (pre[i] != j)

G[j].w += dis[to] - dis[i];

}

}

Hash = new Node \*[(n + 1)];

Q.push(t);

Hash[t] = NULL;

while (!Q.empty()) {

int x = Q.front();

Q.pop();

if (pre[x])

Hash[x] = Hash[G[pre[x]].to];

for (int i = G.head[x]; i; i = G[i].next)

if (pre[x] != i && dis[G[i].to] != INF)

Hash[x] = meGTe(Hash[x], new Node(G[i].w, G[i].to, NULL, NULL));

for (int i = GT.head[x]; i; i = GT[i].next) {

int y = GT[i].to;

if (pre[y] == i)

Q.push(y);

}

}

}

int kthPath(int k) {

if (s == t)

k++;

if (dis[s] == INF)

return -1;

if (k == 1)

return dis[s];

priority\_queue<Status> Q;

if (!Hash[s])

return -1;

Q.push(Status(Hash[s]->val, Hash[s]));

while (--k && !Q.empty()) {

Status x = Q.top();

Q.pop();

if (k == 1)

return x.dist + dis[s];

int y = x.p->to;

if (Hash[y])

Q.push(Status(x.dist + Hash[y]->val, Hash[y]));

if (x.p->left)

Q.push(Status(x.dist - x.p->val + x.p->left->val, x.p->left));

if (x.p->right)

Q.push(Status(x.dist - x.p->val + x.p->right->val, x.p->right));

}

return -1;

}

int main() {

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

G = Map(n, m);

GT = Map(n, m);

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

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

G.addEdge(u, v, w);

GT.addEdge(v, u, w);

}

scanf("%d%d%d", &s, &t, &k);

queue<int> Q = SPFA(t);

rebuild(Q);

printf("%d\n", kthPath(k));

return 0;

}