P1967-运输计划 解题报告

本题是洛谷P1967，NOIP2013提高组真题。

前置知识：最小生成树、树链剖分、静态线段树区间最值。

解题过程：

首先，我们做题都会有一个目的，我做这个题一是为了刷真题，二是为了练习刚刚学习的树链剖分算法。

首先，因为这是一道裸题，我就没有想什么部分分算法，直接打了正解。

其实，这里部分分算法甚至还没有正解好写、好打，前提是那些前置知识你都会。

话不多说，上思考过程：

拿到题的第一时间，我是没有什么思路的，后来发现这不是一棵树，但是却有树链剖分的标签，那么我们不妨大胆地做一个假设：这题需要我们把它建成一棵树。

之后就很好想了，我们对原图构建最大生成树，然后进行标准的树链剖分，之后构建线段树即可。

上伪代码：

初始化数据；

构建生成树；

进行树链剖分；

构造线段树；

回答询问；

由此，即可。

上代码：

#include <cstdio>

#include <iostream>

#include <algorithm>

#define MAXN 10005

#define MAXM 100005

#define mi2(a, b) (a < b ? a : b)

#define mx2(a, b) (a > b ? a : b)

using namespace std;

template <class T>

void read(T &ans)

{

   ans = 0;

   char us = getchar();

   bool f = false;

   while (us < 48 || us > 57)

   {

      f |= (us == 45);

      us = getchar();

   }

   while (us > 47 && us < 58)

   {

      ans = (ans << 1) + (ans << 3) + (us ^ 48);

      us = getchar();

   }

   ans \*= f ? -1 : 1;

   return;

}

template <class T, class... O>

void read(T &x, O &...oth)

{

   read(x);

   read(oth...);

}

struct line

{

   int a, b;

   int v;

   bool operator<(const line &oth) const

   {

      return v > oth.v;

   }

} lines[MAXM];

int fst[MAXN], nt[MAXM];

int to[MAXM];

int v[MAXM];

int zpld[MAXN], bigc[MAXN], depth[MAXN], fa[MAXN], dfsn[MAXN];

int sz[MAXN];

int data[MAXN];

void addline(int a, int b, int c)

{

   static int i = 0;

   ++i;

   to[i] = b;

   nt[i] = fst[a];

   fst[a] = i;

   v[i] = c;

   ++i;

   to[i] = a;

   nt[i] = fst[b];

   fst[b] = i;

   v[i] = c;

}

int anc[MAXN];

int getanc(int nw)

{

   if (anc[anc[nw]] != anc[nw])

   {

      anc[nw] = getanc(anc[nw]);

   }

   return anc[nw];

}

void dfs1(int nw, int f)

{

   fa[nw] = f;

   depth[nw] = depth[f] + 1;

   if (to[fst[nw]] == f)

   {

      fst[nw] = nt[fst[nw]];

   }

   sz[nw] = 1;

   int i;

   for (i = fst[nw]; i; i = nt[i])

   {

      if (to[nt[i]] == f)

      {

         nt[i] = nt[nt[i]];

      }

      dfs1(to[i], nw);

      sz[nw] += sz[to[i]];

      if (sz[to[i]] > sz[bigc[nw]])

      {

         bigc[nw] = to[i];

      }

   }

}

void dfs2(int nw, int ld)

{

   zpld[nw] = ld;

   static int x = 0;

   ++x;

   dfsn[nw] = x;

   if (!bigc[nw])

   {

      return;

   }

   dfs2(bigc[nw], ld);

   int i;

   for (i = fst[nw]; i; i = nt[i])

   {

      if (to[i] == bigc[nw])

      {

         data[dfsn[to[i]]] = v[i];

         continue;

      }

      dfs2(to[i], to[i]);

      data[dfsn[to[i]]] = v[i];

   }

}

struct node

{

   int l, r;

   node \*lc;

   node \*rc;

   int mi;

   void build(int L, int R)

   {

      l = L;

      r = R;

      if (L >= R)

      {

         mi = data[L];

         return;

      }

      lc = new node;

      rc = new node;

      lc->build(L, (L + R) >> 1);

      rc->build(((L + R) >> 1) + 1, R);

      pushup();

   }

   void pushup(void)

   {

      if (lc)

      {

         mi = mi2(lc->mi, rc->mi);

      }

   }

   int ask(int L, int R)

   {

      if (L <= l && R >= r)

      {

         return mi;

      }

      int a = 0x7fffffff, b = 0x7fffffff;

      if (L <= (l + r) >> 1)

      {

         a = lc->ask(L, R);

      }

      if (R > (l + r) >> 1)

      {

         b = rc->ask(L, R);

      }

      return mi2(a, b);

   }

} \* root;

int ask(int a, int b)

{

   int ans = 0x7fffffff;

   int u;

   while (1)

   {

      if (zpld[a] == zpld[b])

      {

         if (depth[a] > depth[b])

         {

            swap(a, b);

         }

         u = root->ask(dfsn[a] + 1, dfsn[b]);

         ans = mi2(ans, u);

         return ans;

      }

      if (depth[zpld[a]] < depth[zpld[b]])

      {

         swap(a, b);

      }

      u = root->ask(dfsn[zpld[a]], dfsn[a]);

      a = fa[zpld[a]];

      ans = mi2(ans, u);

   }

}

int n, m, q;

int main()

{

   freopen("F:\\tests\\P1967\_1.in", "r", stdin);

   read(n, m);

   int x, y;

   int i;

   for (i = 1; i <= m; ++i)

   {

      read(lines[i].a, lines[i].b, lines[i].v);

   }

   sort(lines + 1, lines + m + 1);

   for (i = 1; i <= n; ++i)

   {

      sz[i] = 1;

      anc[i] = i;

   }

   for (i = 1; i <= m; ++i)

   {

      if (getanc(lines[i].a) == getanc(lines[i].b))

      {

         continue;

      }

      addline(lines[i].a, lines[i].b, lines[i].v);

      if (sz[anc[lines[i].a]] < sz[anc[lines[i].b]])

      {

         anc[anc[lines[i].a]] = anc[lines[i].b];

         sz[anc[lines[i].b]] += sz[anc[lines[i].a]];

      }

      else

      {

         anc[anc[lines[i].b]] = anc[lines[i].a];

         sz[anc[lines[i].a]] += sz[anc[lines[i].b]];

      }

   }

   read(q);

   for (i = 1; i <= n; ++i)

   {

      if (depth[i] == 0)

      {

         depth[i] = 1;

         dfs1(i, i);

         dfs2(i, i);

      }

   }

   root = new node;

   root->build(1, n);

   for (i = 1; i <= q; ++i)

   {

      read(x, y);

      if (getanc(x) != getanc(y))

      {

         printf("-1\n");

         continue;

      }

      printf("%d\n", ask(x, y));

   }

}