安全路径

一. 考察内容：

图论 最短路径 树链剖分 线段树

二. 题目分析：

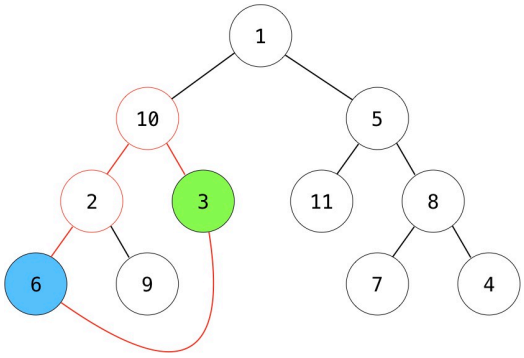
[题目大意]

一张无向连通图，保证每个节点到1节点的最短路是唯一的。求出1节点到所有点，且不经过1节点到该点的最短路上最后一条边的最短路径。

[写题思路]

刚拿到题时，我觉得这个题是一道简单的最短路径问题，求出每个点的最短路，然后枚举每个点的出边，对于一条非最短路径边，通过这条边更新该点的答案即可，但是提交上去之后除了样例全部WA掉，仔细思考之后发现，从其他节点转移的时候，并不能保证不经过那条最短路边，所以我考虑建出最短路树（原本是DAG，但是题目中保证是一棵树），然后通过DFS的子节点的横叉边和该点的横叉边、返祖边转移，发现情况考虑的不周全，于是看了题解，发现是树剖。。。

先考虑在最短路树上，每加一条非树边，会出现一个环，环上所有节点（除了环上所有点的公共祖先以外）都会有一个新的可行解，也就是通过这条非树边，从另一条树链走向该点。



如图，添加一条非树边：3——6，则节点6的可行解为1—>10—>3—>6，节点2的可行解为1—>10—>3—>6—>2。

很显然，我们可以通过树剖时记录的信息，O(1)时间内求出环上任何一个点可行解大小，用线段树维护他们（神奇的区间修改操作），求出每个点的所有可行解的最小值即可。

三. 代码实现：

```
#define _CRT_SECURE_NO_DEPRECATE
/*************************
*创建时间：2018 09 05
*文件类型：源代码文件
*题目来源：BZOJ
*当前状态：已通过
*备忘录：图论 最短路径 树链剖分 线段树
*作者：HtBest
*************************/
#include <stdio.h>
#include <iostream>
#include <string>
#include <stdlib.h>
#include <string.h>
#include <algorithm>
#include <queue>
#include <bitset>
// #include <sys/wait.h>
// #include <sys/types.h>
// #include <unistd.h>
using namespace std;
#define MAXN 100001
#define MAXM 200001
```

```
int n,m,_edge,head[MAXN],d[MAXN],
son[MAXN],bro[MAXN],fa[MAXN],
deep[MAXN],dfsx[MAXN],_dfsx,top[MAXN],h_son[MAXN],num[MAXN],real[MAXN];
struct EDGE
{
    int a,b,v,next;
    EDGE(int a=0,int b=0,int v=0,int next=0):a(a),b(b),v(v),next(next){}
}edge[2*MAXM];
struct A
{
    int id,d;
    A(int id=0,int d=0):id(id),d(d){}
    bool operator < (A x) const
    {
        return d>x.d;
    }
};
struct NODE
{
    int lson,rson,v,lazy;
    void clear()    {lson=rson=0;v=lazy=1e9;}
    void addl(int x)
    {
        lazy=min(lazy,x);
        v=min(x,v);
    }
}o[2*MAXN];
struct segmentTree
{
    int root,cnt;
    void downl(NODE &x,int l,int r)
    {
        if(x.lazy)
        {
            int m=(l+r)>>1;
            // printf("%d %d %d %d\n",x.lson,x.lazy,x.rson,x.lazy-d[real[m+1]]
+d[real[l]]);
            o[x.lson].addl(x.lazy);
            o[x.rson].addl(x.lazy-d[real[m+1]]+d[real[l]]);
        }
        x.lazy=1e9;
    }
    void build(int &x,int l,int r)
    {
        x=++cnt;
        o[x].clear();
        if(l!=r)
        {
            int m=(l+r)>>1;
            build(o[x].lson,l,m);
            build(o[x].rson,m+1,r);
        }
    }
    void update(int x,int l,int r,int ul,int ur,int v)
    {
        if(l>=ul&&r<=ur)o[x].addl(v-d[real[l]]+d[real[ul]]);//,printf("#%d %d %d
%d#\n",l,r,v-d[real[l]]+d[real[ul]],x);
        else
        {
            int m=(l+r)>>1;
            downl(o[x],l,r);
            if(ul<=m)update(o[x].lson,l,m,ul,ur,v);
            if(ur>m)update(o[x].rson,m+1,r,ul,ur,v);
        }
    }
    int query(int x,int l,int r,int p)
    {
        if(l==r)return o[x].v;
        else
        {
            int m=(l+r)>>1;
            downl(o[x],l,r);
```

```
            if(p<=m)    return query(o[x].lson,l,m,p);
            else        return query(o[x].rson,m+1,r,p);
        }
    }
    void build(){cnt=0,build(root,1,n);}
    void update(int l,int r,int v)
    {
        update(root,1,n,l,r,v);
    }
    int query(int p){return query(root,1,n,p);}
}seg;
/* Variable explain:
n:节点数
m:边数
head[i]:节点i的第一条出边
edge[i]:邻接表
_edge:邻接表标记
d[i]:节点i距1点的最短路
son[i]:节点i在最短路树上的第一个儿子
bro[i]:节点i在最短路树上的下一个兄弟
fa[i]:节点i在最短路树上的父亲
deep[i]:节点i在最短路树上的深度
dfsx[i]:最短路树上节点i的树剖dfs序
_dfsx:dfsx序标记
top[i]:最短路树上节点i的树剖链顶
h_son[i]:最短路树上节点i的树剖重儿子
num[i]:最短路树上子树i的大小
real[i]:dfs序为i的节点的真实标号
o[i]:线段树节点
seg:线段树
*/
void adde(int a,int b,int v)
{
    edge[++_edge]=EDGE(a,b,v,head[a]);
    head[a]=_edge;
}
void addt(int a,int b,int v)
{
    fa[b]=a;
    bro[b]=son[a];
    son[a]=b;
}
void read()
{
    int ls1,ls2,ls3;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;+
+i)scanf("%d%d%d",&ls1,&ls2,&ls3),adde(ls1,ls2,ls3),adde(ls2,ls1,ls3);
    return;
}
void dijkstra()
{
    priority_queue <A> q;
    bitset <MAXN> vis;
    for(int i=1;i<=n;++i)d[i]=1e9;
    d[1]=0;
    q.push(A(1,0));
    while(!q.empty())
    {
        A a=q.top();
        q.pop();
        if(vis[a.id])continue;
        vis[a.id]=1;
        for(int i=head[a.id];i;i=edge[i].next)
        {
            int b=edge[i].b,v=edge[i].v;
            if(d[b]>d[a.id]+v)
```

```
            {
                d[b]=d[a.id]+v;
                q.push(A(b,d[b]));
            }
        }
    }
}
bool vis[MAXN];
void dfs1(int a,int c)
{
    deep[a]=c;
    for(int i=head[a];i;i=edge[i].next)
    {
        int b=edge[i].b,v=edge[i].v;
        if(d[b]!=d[a]+v||fa[a]==b)continue;//非树边和来边不能建树
        addt(a,b,v);
        dfs1(b,c+1);
        num[a]+=num[b];
        if(num[h_son[a]]<num[b])h_son[a]=b;
    }
    ++num[a];
}
void dfs2(int a,int t)
{
    dfsx[a]=++_dfsx;
    top[a]=t;
    if(h_son[a])dfs2(h_son[a],t);
    for(int i=son[a];i;i=bro[i])
        if(i!=h_son[a])dfs2(i,i);
}
void lca_update(int a,int b,int v)
{
    int va=d[b]+v,vb=d[a]+v;//记录到达a、b点需要花费的值
    while(1)
    {
        if(top[a]==top[b])break;
        if(deep[top[a]]>deep[top[b]])swap(a,b),swap(va,vb);
        seg.update(dfsx[top[b]],dfsx[b],vb+d[b]-d[top[b]]);
        vb+=d[b]-d[fa[top[b]]];
        b=fa[top[b]];
    }
    if(a!=b)
    {
        if(deep[a]>deep[b])swap(a,b),swap(va,vb);
        seg.update(dfsx[h_son[a]],dfsx[b],vb+d[b]-d[h_son[a]]);
    }
}
int main()
{
    // freopen(".in","r",stdin);
    // freopen(".out","w",stdout);
    read();
    seg.build();
    dijkstra();
    dfs1(1,0);
    dfs2(1,1);
    // for(int i=1;i<=n;++i)printf("%d ",num[i]);puts("");exit(0);
    for(int i=1;i<=n;++i)real[dfsx[i]]=i;
    int counts=0;
    for(int i=1;i<=_edge;i+=2)
    {
        int a=edge[i].a,b=edge[i].b,v=edge[i].v;
        if(d[a]==d[b]+v||d[b]==d[a]+v)continue;
        lca_update(a,b,v);counts++;
    }
    // printf("%d\n",counts);
    int ls1;
    for(int i=2;i<=n;++i)ls1=seg.query(dfsx[i]),printf("%d ",ls1>=1e8?-1:ls1);
    return 0;
}
```

<题目跳转> <查看代码>