逛公园

一. 考察内容：

动态规划 图论 最短路径

二. 题目分析：

[题目大意]

在有向非负权图上求出从1点到n点之间比最短路径长不超过k的路径条数。

[写题思路]

第一步，我们先求出原图的最短路，可以考虑用速度较快的Dijkstra求。

下面，我们将要用动态规划的方法求出答案，考虑题目中的k非常小，我们拿k作为一个维度设状态，设状态f[i][j]为1到j比最短路长i的路径数，很显然，通过每一条边转移即可，f[i][a]=f[i-v[e]+d[a]-d[b]][b]，但是转移的顺序是值得注意的，对于最短路长度不同的节点，我们可以按照节点的最短路从小到大转移（转移时可能需要反向建图），但是对于最短路长度相同的节点，我们需要考虑求出以所有边权为0的边建立的图的拓扑排序，很显然，有有限条路径数的充要条件是没有0权环，所以拓扑排序是一定可以求出来的，**总结：最短路长度不同的，以最短路从小到大转移，最短路相同的，以拓扑排序顺序转移。**

三. 代码实现：

```
#define _CRT_SECURE_NO_DEPRECATE
/************************
*创建时间：2018 08 22
*文件类型：源代码文件
*题目来源：洛谷
*当前状态：已通过
*备忘录：最短路 动态规划 Dijkstra
*作者：HtBest
************************/
#include <stdio.h>
#include <iostream>
#include <string>
#include <stdlib.h>
#include <string.h>
#include <algorithm>
#include <queue>
// #include <sys/wait.h>
// #include <sys/types.h>
// #include <unistd.h>
using namespace std;
namespace   IO
{
    #define SIZE 1
    int T=0,end=0;
    char s[SIZE+1];
    int in()
    {
        T=0;
        return end=fread(s,1,SIZE,stdin);
    }
    int read(int &a)
    {
        int i=a=0,flag=1;
        while(1)
        {
            if(T==end&&!in())return i;
            if(s[T]==' '||s[T]=='\n'||s[T]=='\r')
            {
                if(i)return i;
```

```
            }
            else if(s[T]=='-')flag=-1;
            else ++i,a=flag*(flag*10*a+s[T]-'0');
            ++T;
        }
    }
}
using IO::read;
#define MAXN 100001
#define MAXM 200002
int n,m,k,MOD,head[MAXN],uhead[MAXN],h0[MAXN],_uedge,_edge,_e0,f[55]
[MAXN],num[MAXN],topo[MAXN],_topo,in_d[MAXN];
long long d[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[MAXM],uedge[MAXM],e0[MAXM];
struct A
{
    int id,v;
    A(int id=0,int v=0):id(id),v(v){}
    bool operator < (A a) const
    {
        return v>a.v;
    }
};
struct NODE
{
    int id;
    long long d;
    bool operator < (NODE a) const
    {
        return d==a.d?topo[id]<topo[a.id]:d<a.d;
    }
}node[MAXN];
/* Variable explain:
n:节点数
m:边数
k:允许的次短路范围
MOD:模数
head[i]:每个节点的第一条出边
uhead[i]:反向建图后每个节点的第一条出边
h0[i]:以0的边建图后每个节点的第一条出边
edge:邻接表
_edge:邻接表标记
uedge:反向邻接表
_uedge:反向邻接表标记
e0:边权为0的邻接表
_e0:边权为0的邻接表标记
f[i][j]:1到j比最短路长i的路径数
num[i]:1到i的最短路个数
d[i]:i距离1的最短路长度
node[i]:标记第i个节点的信息
topo[i]:拓扑排序
_topo:拓扑排序标记
in_d[i]:每个点的入度
*/
void adde(int a,int b,int v)
{
    edge[++_edge]=EDGE(a,b,v,head[a]);
    head[a]=_edge;
}
void addu(int a,int b,int v)
{
    uedge[++_uedge]=EDGE(a,b,v,uhead[a]);
```

```
    uhead[a]=_uedge;
}
void add0(int a,int b,int v)
{
    e0[++_e0]=EDGE(a,b,v,h0[a]);
    h0[a]=_e0;
}
int add(int a,int b){return (a+b)%MOD;}
void reset()
{
    _edge=_uedge=_e0=0;
    for(int i=1;i<=n;++i)head[i]=uhead[i]=h0[i]=num[i]=in_d[i]=0;

}
void read()
{
    int ls1,ls2,ls3;
    read(n),read(m),read(k),read(MOD);
    reset();
    for(int i=1;i<=m;++i)
    {
        read(ls1),read(ls2),read(ls3),adde(ls1,ls2,ls3),addu(ls2,ls1,ls3);
        if(!ls3)add0(ls1,ls2,ls3),++in_d[ls2];
    }
    return;
}
void dijkstra()
{
    bool vis[MAXN]={0};
    priority_queue <A> q;
    for(int i=2;i<=n;++i)d[i]=1e9;
    num[1]=1;
    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)num[b]=add(num[b],num[a.id]);
            if(d[b]>d[a.id]+v)
            {
                d[b]=d[a.id]+v;
                num[b]=num[a.id];
                q.push(A(b,d[b]));
            }
        }
    }
}
void toposort()
{
    deque <int> q;
    for(int i=1;i<=n;++i)
        if(in_d[i]==0)topo[i]=++_topo,q.push_back(i);
    while(!q.empty())
    {
        int a=q.front();
        q.pop_front();
        for(int i=h0[a];i;i=e0[i].next)
        {
            int b=e0[i].b;
            --in_d[b];
            if(!in_d[b])topo[b]=++_topo,q.push_back(b);
        }
    }
}
int main()
{
```

```
    // freopen(".in","r",stdin);
    // freopen(".out","w",stdout);
    int T=1;
    read(T);
    while(T--)
    {
        read();
        dijkstra();
        // for(int i=1;i<=n;++i)printf("%d ",num[i]);
        for(int i=1;i<=n;++i)f[0][i]=num[i],node[i].d=d[i],node[i].id=i;
        for(int i=0;i<=k;++i)for(int j=1;j<=n;++j)f[i][j]=0;
        f[0][1]=1;
        toposort();
        int flag=1;
        for(int i=1;i<=n;++i)if(in_d[i]){printf("-1\n");flag=0;break;}
        // for(int i=1;i<=n;++i)if(in_d[i]){printf("%d %d\n",i,in_d[i]);}
        if(!flag)continue;
        sort(node+1,node+1+n);
        for(int i=0;i<=k;++i)
        {
            for(int j=1;j<=n;++j)
            {
                // printf("%d\n",node[j].id);
                for(int k=uhead[node[j].id];k;k=uedge[k].next)
                {
                    int a=uedge[k].a,b=uedge[k].b,v=uedge[k].v;
                    if(v-d[a]+d[b]<=i)f[i][a]=add(f[i][a],f[i-v+d[a]-d[b]][b]);
                }
            }
        }
        int ans=0;
        for(int i=0;i<=k;++i)ans=add(ans,f[i][n]);
        printf("%d\n",ans);
    }

    return 0;
}
```

<题目跳转> <查看代码>