算法部分

题意就是如果某条路的长度减去了L（最少为0），给定的部落对之间最短路总和的最小值是多少

那么只需要预处理出减少长度前部落两两之间的最短路，就可以通过枚举那条特殊的道路，公式推得答案

假设第i条道路进行了改造，这条路连接u,v两个部落，当前处理的是部落对a,b，那么a,b之间的最短路为

$$Dist[a][b]=min\begin{cases}dist[a][b]\\dist[a][u]+dist[v][b]+max(0,w-L)\\dist[a][v]+dist[u][b]+max(0,w-L)\end{cases}$$

故本题主要为预处理两两之间的最短路即可

Floyd

采取简单粗暴的$O(n^3)$方法求解，数据降为800，只要常数不大就能过（原题没卡掉Floyd）

```cpp
#include<bits/stdc++.h>
using namespace std;

int n,m,k,L;
int dis[850][850];
int u[850],v[850],w[850];
int a[2050],b[2050];

void floyd()
{
    for(int k=1;k<=n;k++)
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);
}

int main()
{
    scanf("%d%d%d%d",&n,&m,&k,&L);

    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            dis[i][j]=1e9;
    for(int i=1;i<=n;i++)
```

```
25          dis[i][i]=0;
26
27      for(int i=1;i<=m;i++)
28      {
29          scanf("%d%d%d",&u[i],&v[i],&w[i]);
30          dis[u[i]][v[i]]=dis[v[i]][u[i]]=w[i];
31      }
32
33      for(int i=1;i<=k;i++)
34          scanf("%d%d",&a[i],&b[i]);
35
36      floyd();
37
38      int ans=2e9;
39      for(int i=1;i<=m;i++)
40      {
41          int sum=0;
42          for(int j=1;j<=k;j++)
43              sum+=min({dis[a[j]][b[j]],dis[a[j]][u[i]]+dis[v[i]][b[j]]+max(dis[u[i]][v[i]]-
    L,0),dis[a[j]][v[i]]+dis[u[i]][b[j]]+max(dis[u[i]][v[i]]-L,0)});
44          ans=min(ans,sum);
45      }
46      printf("%d\n",ans);
47
48      return 0;
49  }
```

Dijkstra

时间复杂度为$O(nlogn)$，需要对于每个点都做一遍，故最终时间复杂度为$O(n^2logn)$

```
1   #include<bits/stdc++.h>
2   using namespace std;
3   typedef pair<int,int> P;
4
5   int n,m,k,L;
6   vector<P> G[850];
7   int dis[850][850];
8   int u[850],v[850],w[850];
9   int a[2050],b[2050];
10
11  void dijkstra(int st)
```

```
{
    priority_queue<P> pq;
    for(int i=1;i<=n;i++)
        dis[st][i]=1e9;
    dis[st][st]=0;
    pq.push(P(0,st));
    while(!pq.empty())
    {
        int v=pq.top().second;
        pq.pop();
        for(P pd:G[v])
            if(dis[st][v]+pd.second<dis[st][pd.first])
            {
                dis[st][pd.first]=dis[st][v]+pd.second;
                pq.push(P(-dis[st][pd.first],pd.first));
            }
    }
}

int main()
{
    scanf("%d%d%d%d",&n,&m,&k,&L);

    for(int i=1;i<=m;i++)
    {
        scanf("%d%d%d",&u[i],&v[i],&w[i]);
        G[u[i]].push_back(P(v[i],w[i]));
        G[v[i]].push_back(P(u[i],w[i]));
    }

    for(int i=1;i<=k;i++)
        scanf("%d%d",&a[i],&b[i]);

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

    int ans=2e9;
    for(int i=1;i<=m;i++)
    {
        int sum=0;
        for(int j=1;j<=k;j++)
            sum+=min({
                dis[a[j]][b[j]],
                dis[a[j]][u[i]]+dis[v[i]][b[j]]+max(dis[u[i]][v[i]]-L,0),
                dis[a[j]][v[i]]+dis[u[i]][b[j]]+max(dis[u[i]][v[i]]-L,0)
```

```
            });
        ans=min(ans,sum);
    }
    printf("%d\n",ans);

    return 0;
}
```

交互处理部分

- 因为数据量过大，限时 5s，因此手动操作的难度非常大，借助 Pwntools 操作就很方便了
- 首先把写完的解题程序编译好，这边编译到了 `./wqt_judger_acm/solves`
- 然后开个 process 开个 remote 交互，通过逆向可以知道有 18 组测试数据
- 循环 18 次，把数据给本地的解题程序求解，将结果发过去，代码如下

```
from pwn import *

io = remote("121.43.169.147", 8848)

io.sendlineafter("choice : ", "1")
io.sendlineafter("judger name: ", "1")
io.sendlineafter("judger type: ", "1")

io.sendlineafter("choice : ", "2")
for i in range(18):
    rectmp = io.recvuntil("below input:").decode()
    inp = io.recvuntil("Your Answer (Time limit 5s):", drop=True)
    solver = process("./wqt_judger_acm/solves")
    solver.sendline(inp)
    res = solver.recvuntil("\n", drop=True)
    print(res)
    io.sendline(res)
io.interactive()
```