# noj实验9报告

## 0.0 题目//需求分析

File Name: T009.cpp

实验4.2：用迪杰斯特拉算法求赋权图中的最短路径

Time Limit: 3000ms, Memory Limit: 10000KB, Accepted: 0, Total Submissions: 0

VIDEO1

Description

用迪杰斯特拉算法求一点到其余所有结点的最短路径。

Input

先输入一个小于100的正整数n，然后输入图的邻接矩阵（10000表示无穷大，即两点之间没有边），最后输入两个0到n-1的整数表示两个点。

Output

先用迪杰斯特拉算法求给定的第一个点到其余所有结点的最短路径。
然后再输出给定的两个点之间的最短路径（按顺序输出最短路径上的每一个点，每个数据占一行）。

Sample Input

```
4
0??10?10000
2?0???3
```

需求分析：在第八题的基础上，存储最短路径，并输出。

## 1.0 实验思路

仍然是缔结斯塔拉算法。

不过要增加一个存储结构，用于存下路径。

操作步骤是：

**(1)** 初始时，S只包含起点s；U包含除s外的其他顶点，且U中顶点的距离为"起点s到该顶点的距离"[例如，U中顶点v的距离为(s,v)的长度，然后s和v不相邻，则v的距离为∞]。

**(2)** 从U中选出"距离最短的顶点k"，并将顶点k加入到S中；同时，从U中移除顶点k。

**(3)** 更新U中各个顶点到起点s的距离。之所以更新U中顶点的距离，是由于上一步中确定了k是求出最短路径的顶点，从而可以利用k来更新其它顶点的距离；例如，(s,v)的距离可能大于(s,k)+(k,v)的距离。

**(4)** 重复步骤(2)和(3)，直到遍历完所有顶点。

## 2.0 代码

```
#include <stdio.h>
#include <stdlib.h>

struct graphList
{
    int vexNum;
```

```
    int graph[120][120];
};

struct step
{
    int flags[3000];
    int stepN[3000];
};

void run();
void createNewGraphList (struct graphList *gList);
void DJ(struct step *gStep,struct graphList *gList);
void clearStep(struct step *gStep,struct graphList *gList);
void initializeStep(struct step *gStep,struct graphList *gList);
int judgeStep(struct step *gStep,struct graphList *gList);
int findMinStepN(struct step *gStep,struct graphList *gList);
void updateStepN(struct step *gStep,struct graphList *gList,int min);

int main()
{
    run ();
    return 0;
}

void run()
{
    struct graphList gList;
    struct step gStep;
    createNewGraphList (&gList);
    DJ (&gStep,&gList);
}

void createNewGraphList(struct graphList *gList)
{
    int i,j;
    scanf ("%d",&(gList->vexNum));
    for (i=0;i<gList->vexNum;i++)
    {
        for (j=0;j<gList->vexNum;j++)
        {
            scanf ("%d",&(gList->graph[i][j]));
        }
    }
}

void DJ(struct step *gStep,struct graphList *gList)
{
    int min;
    int end,endStep;
    clearStep (gStep,gList);
    initializeStep (gStep,gList);
    scanf ("%d",&end);
    endStep=gStep->stepN[end];
    while (judgeStep (gStep,gList))
    {
        min=findMinStepN (gStep,gList);
        updateStepN (gStep,gList,min);
        if (gStep->flags[end])
```

```
        {
            if (endStep!=gStep->stepN[end])
            {
                endStep=gStep->stepN[end];
                printf ("%d\n",min);
            }
        }
        else
        {
            printf ("%d\n",min);
            break;
        }
    }
}

void clearStep(struct step *gStep,struct graphList *gList)
{
    int i;
    for (i=0;i<gList->vexNum;i++)
    {
        gStep->flags[i]=-1;
        gStep->stepN[i]=0;
    }
}

void initializeStep(struct step *gStep,struct graphList *gList)
{
    int i;
    int start;
    scanf ("%d",&start);
    printf ("%d\n",start);
    for (i=0;i<gList->vexNum;i++)
    {
        if (gList->graph[start][i]!=10000)
        {
            gStep->flags[i]=1;
            gStep->stepN[i]=gList->graph[start][i];
        }
    }
    gStep->flags[start]=-1;
    gStep->stepN[start]=0;
}

int judgeStep(struct step *gStep,struct graphList *gList)
{
    int i;
    for (i=0;i<gList->vexNum;i++)
    {
        if (gStep->flags[i]==1)
        {
            return 1;
        }
    }
    return 0;
}

int findMinStepN(struct step *gStep,struct graphList *gList)
{
```

```
    int i,min=99999,n=-1;
    for (i=0;i<gList->vexNum;i++)
    {
        if (gStep->flags[i]==1)
        {
            if (gStep->stepN[i]<min)
            {
                min=gStep->stepN[i];
                n=i;
            }
        }
    }
    return n;
}

void updateStepN(struct step *gStep,struct graphList *gList,int min)
{
    int i;
    int minStepN=gStep->stepN[min];
    gStep->flags[min]=0;
    for (i=0;i<gList->vexNum;i++)
    {
        if (gStep->flags[i]==1)
        {
            if (gStep->stepN[i]>gList->graph[min][i]+minStepN)
            {
                gStep->stepN[i]=gList->graph[min][i]+minStepN;
            }
        }
        else
        {
            if (gStep->flags[i]==-1)
            {
                gStep->flags[i]=1;
                gStep->stepN[i]=gList->graph[min][i]+minStepN;
            }
        }
    }
}
```