哈尔滨工业大学

数据科学与大数据技术专业

实验报告

课程名称：动态规划算法的设计与实现

实验项目：动态规划

实验题目：多段图的最短路径问题的动态规划算法的设计与实现

实验日期：2020.12

班级：1804105

学号：1180400510

姓名：石瑞河

|  |  |  |
| --- | --- | --- |
| 设计成绩 | 报告成绩 | 指导老师 |
|  |  | 张岩 |

**一、实验目的**

1.学会动态规划算法的分析与设计，并实现。

2.比较图的不同存储方式的时空性能

**二、实验要求及实验环境**

实验要求：

1. 设计多段图的最短路径问题的动态规划算法，并证明其具有最优子结构性质 和重叠子问题性质，以及算法的正确性，写在实验报告中。

2. 采用邻接矩阵存储多段图，实现多段图最短路径和最短路径长度的动态规划 算法。

3. 采用邻接表存储多段图，实现多段图最短路径和最短路径长度的动态规划算 法。

4. 比较两种存储方法所实现的算法的时空性能

实验环境：

硬件：



软件：

VSCode，Mingw-w64

**三、设计的基本思想和算法的原理**

动态规划设计有着一定的模式，一般要经历以下几个步骤：

1.划分阶段：按照问题的时间或空间特征，把问题分为若干个阶段。

2.确定状态：将问题发展到各个阶段时所处的各种客观情况用不同的状态表示出来。

3.确定决策并写出状态转移方程：因为决策和状态转移有着天然的联系，状态转移就是根据上一阶段的状态和决策来导出本阶段的状态，所以如果确定了决策，状态转移方程也就可以写出。

4. 寻找边界条件：给出的状态转移方程是一个递推式，需要一个递推的终止条件或边界条件。

5. 程序设计实现：动态规划的主要难点在于理论上的设计，一旦设计完成，实现部分就会非常简单

**最优子结构性质：**

从源点s到汇点t的最短距离，可以由源点到其后继结点v的距离与结点v到汇点t的最短距离之和的最小值得到，其中，结点v到汇点t的最小值的求解就是原问题的一个子问题，原问题的最优解由子问题的最优解到处，故具有最优子结构性质。

**子问题重叠性：**

dis[v]的先驱节点可能有很多个，计算每一个先驱节点的最优解时都需要用到节点v的最优解取尝试更新，v节点的最优解被多次用到，故问题具有子问题重叠性。

**问题分析与设计：**

对于一个DAG，设定源点为是，汇点为t

用dis[u]表示节点u到汇点t的最短距离

对于一个节点u,有若干个后继节点v,通过权值为的边连接

显然有



即为状态转移方程

对于一条有向边（u,v,w），假设dis[v]已知，则可以用来更新dis[u]

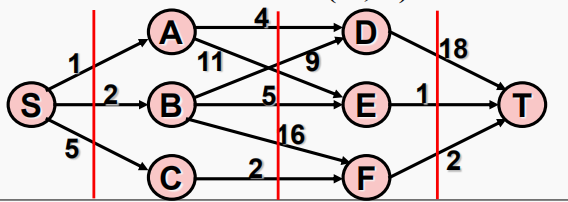
用递归的思想，先得到v到t的最小值dis[v],然后就能得到dis[u]

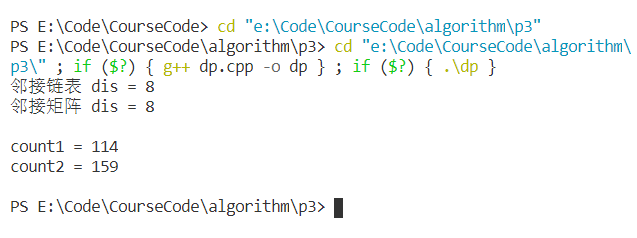
同时，用dp的思想，若dis[v]已经被计算过则不必重复计算

边界条件：dis[t] = 0;

**四、实验结果与分析**

对ppt样例中的8个节点，12条边的DAG计算几点1到节点8的最短距离





无论是使用邻接链表还是邻接矩阵，都计算出了节点1到节点8的最小距离值为8，但是用邻接链表比邻接矩阵的计数值小约30%，这是因为邻接链表省去了邻接矩阵寻找后继结点的判断操作，在稀疏图上的能够节省很大的不必要的判断。

**五、经验体会与不足**

写出状态转移方程后，动态规划的实现往往非常简单和容易

**六、附录：源代码（带注释）**

见压缩包附录文件 dp.cpp

#include <iostream>

#include <cstdio>

#include <algorithm>

#include <vector>

using namespace std;

#define pii pair<int,int>

const int maxn = 1000+10;

const int inf = 1e9;

int n, m;

int dis1[maxn],dis2[maxn];

int mp[maxn][maxn];

int acc1, acc2;

vector<pii> G[maxn];

void addedge(int u, int v, int w)

{

    G[u].push\_back(make\_pair(v,w));

    mp[u][v] = w;

}

void dp(int s, int t)

{

    if(s == t){dis1[t] = 0; return;}

    if(dis1[s] < inf)return;

    int u = s, v, w;

    for(auto& p : G[s] )

    {

        v = p.first;

        w = p.second;

        dp(v,t);

        dis1[u] = min(dis1[v]+w, dis1[u]);

    }

}

// void dp(int s, int t)

// {

//     acc1++;

//     if(s == t){acc1++; dis1[t] = 0; return;}

//     acc1++;

//     if(dis1[s] < inf)return;

//     acc1++;

//     int u = s, v, w;

//     acc1++;

//     for(auto& p : G[s] )

//     {

//         acc1++;

//         v = p.first;

//         acc1++;

//         w = p.second;

//         acc1+=4;

//         dp(v,t);

//         acc1 += 2;

//         if(dis1[v]+w < dis1[u]){acc1 ++; dis1[u] = dis1[v]+w;}

//     }

// }

void dp\_mp(int s, int t)

{

    acc2++;

    if(s == t){acc2++; dis2[t] = 0; return;}

    acc2++;

    if(dis2[s] < inf)return;

    acc2++;

    int u = s, v, w, i;

    acc2 += 2;

    for(v = 1; v <= n; i++)

    {

        acc2++;

        if(mp[u][v] == inf) continue;

        acc2++;

        w = mp[u][v];

        acc2 += 4;

        dp\_mp(v,t);

        acc2 == 2;

        if(dis2[v]+w < dis2[u]){acc2++; dis2[u] = dis2[v]+w;}

    }

}

void pre()

{

    int i, j;

    for(i = 0; i <= maxn; i++)dis1[i] = dis2[i] = inf;

    for(i = 0; i <= maxn; i++)mp[i][i] = 0;

    for(i = 0; i <= maxn; i++)

        for(j = 0; j <= maxn; j++)

            mp[i][j] = inf;

    acc1 = acc2 = 0;

}

int main()

{

    freopen("data1.in","r",stdin);

    scanf("%d %d",&n,&m);

    cin >> n >> m;

    int i,u,v,w;

    for(i = 1; i <= m; i++)

    {

        scanf("%d %d %d",&u,&v,&w);

        addedge(u,v,w);

    }

    int s,t;

    scanf("%d %d",&s,&t);

    pre();

    dp(s,t);

    dp\_mp(s,t);

    printf("邻接链表 dis = %d \n邻接矩阵 dis = %d\n\n", dis1[s], dis2[s]);

    printf("count1 = %d\ncount2 = %d\n\n",acc1,acc2);

}