

《数据结构》实验报告

班 级:_____

姓 名:_____

学 号:_____

E-mail:_____

日 期:_____

◎实验题目:

4.1 求赋权图中一个结点到其余所有结点的路径长度

Description

给一个赋权图（无向图），求0号结点到其余所有结点的最短路径的长度。

◎实验内容:

4.1

Input

先输入一个小于等于100的正整数n，然后输入赋权图的邻接矩阵（10000表示无穷大,并且任意一条简单路径的长度都小于10000）

Output

按结点编号的顺序输出0号结点所有结点的最短路径的长度。

一、需求分析

本次实验需要实现一个使用 Dijkstra 算法寻找图中从一个点到其他所有点的最短路径的程序。程序需要从输入中读取图的信息（包括顶点和边的信息），然后使用 Dijkstra 算法计算从指定的源点到其他所有点的最短路径，然后将结果输出。

二、概要设计

解决这个问题的主要思路是首先构建一个图的数据结构来存储图的信息，然后实现 Dijkstra 算法来寻找最短路径。Dijkstra 算法的基本思想是以源点为中心，逐步扩大覆盖的范围，每一步都选择一个到源点距离最短的未访问节点，然后更新这个节点到其邻居节点的距离。

三、详细设计

数据结构设计:

两个主要的数据结构，一是图 Graph，二是用于执行 Dijkstra 算法的数据结构 Dijkstra。

Graph: 这是图的表示，其中 data 表示顶点数据，arc 为二维数组，表示图中每个顶点到其他顶点的权重，Vnum 为顶点数量，Anum 为边的数量。

Dijkstra: 其中 visited 数组用于标记每个节点是否已访问，length 数组用于存储从源点到每个顶点的最短路径。

算法设计:

initGraph 函数: 读入顶点数量和边的权重，构建图。

initDij 函数: 初始化 Dijkstra 数据结构，将源点到自身的距离设置为 0，源点到其他节点的距离设置为无穷大，所有节点都未访问。

searchMinLengthV 函数: 在未被访问的节点中找到距离源节点最近的节点。

judgeFinished 函数: 判断所有节点是否已经被访问过。

updateArcV 函数: 更新源节点到其他节点的最短距离。

findMinPath 函数: 执行 Dijkstra 算法，计算出源点到其他所有节点的最短路径。

printPath 函数：打印出源点到其他所有节点的最短路径。

伪代码描述如下：

```
函数 findMinPath(G, D):  
    调用 initDij(G, D)  
    对于 i 从 0 到 G.Vnum - 1:  
        t = searchMinLengthV(G, D)  
        如果 judgeFinished(G, D) 则 返回  
        调用 updateArcV(t, G, D)  
    结束对于
```

结束函数

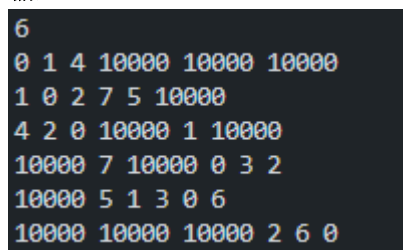
四 使用说明、测试分析及结果

1、说明如何使用你编写的程序；

先输入一个小于等于 100 的正整数，然后输入赋权图的邻接矩阵（10000 表示无穷大，并且任意一条简单路径的长度都小于 10000）；

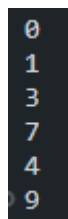
2、测试结果与分析；

输入



```
6  
0 1 4 10000 10000 10000  
1 0 2 7 5 10000  
4 2 0 10000 1 10000  
10000 7 10000 0 3 2  
10000 5 1 3 0 6  
10000 10000 10000 2 6 0
```

输出



```
0  
1  
3  
7  
4  
9
```

3、调试过程中遇到的问题是如何解决以及对设计与实现的回顾讨论和分析

1. 输入格式问题：在读取输入时，我们需要确保输入的稀疏矩阵数据格式正确。在创建稀疏矩阵时，我们使用了一个简单的输入终止条件（`i == 0 && j == 0 && e == 0`），这使得我们能够在读取完所有非零元素后正确地终止输入。这种方法对于本实验是有效的，但在实际应用中可能需要更加健壮的收入验证方法。

2. 初始化辅助数组：在 FastTranspseTSMatrix 算法中，我们需要正确地初始化辅助数组 num 和 cpot。一开始我们没有将 num 数组的元素初始化为 0，导致了计算错误。在发现这个问题后，我们添加了一个循环来初始化 num 数组，并确保了正确的计算结果。

3. 数组下标问题：在处理三元组顺序表时，需要确保数组下标从 1 开始，因为将 0 位置用于输入终止条件。在调试过程中，发现在某些情况下，错误地从 0 开始访问数组，这导致了错误的结果。

回顾讨论与分析：

在设计和实现稀疏矩阵快速转置算法的过程中，我意识到了将问题分解为更小的子任务的重要性。通过将问题分解为初始化、创建、转置和打印稀疏矩阵的子任务，我们能够更清晰地理解算法的逻辑，并且更容易地调试代码。

运行界面

```
6
0 1 4 10000 10000 10000
1 0 2 7 5 10000
4 2 0 10000 1 10000
10000 7 10000 0 3 2
10000 5 1 3 0 6
10000 10000 10000 2 6 0
0
1
3
7
4
9
(base) PS C:\Users\csp\Desktop>
```

五、实验总结

本次实验通过编程实现了 Dijkstra 算法，实现了求解图的最短路径问题，加深了对图和最短路径问题的理解，提高了我的编程能力和解决问题的能力。实验过程中遇到的问题和困难，我们通过查阅资料、分析问题和尝试解决方案，最终成功地完成了实验目标。

4.2 ◎实验题目：

用迪杰特斯拉算法求赋权图的最短路径

◎实验内容

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）。

一、需求分析

本次实验的主要目标是实现基于邻接矩阵的 Dijkstra 算法，用于求解图中两个节点之间的最短路径。输入数据为图的邻接矩阵表示和待求解的两个节点，输出结果是这两个节点之间的最短路径。

二、概要设计

为了实现 Dijkstra 算法，首先需要构建图的邻接矩阵表示。邻接矩阵是一种常见的图的存储方式，能够直观地表示图中的所有节点及其相互之间的关系。然后，使用 Dijkstra 算法求解最短路径。Dijkstra 算法是一种贪心算法，每一步都选择当前的最优解，以此最终求得全局的最优解。

三、详细设计

数据结构设计：

两个主要的数据结构，一是邻接矩阵 MGraph，二是 Dijkstra 路径 Dijkstra。

MGraph: 这是图的邻接矩阵表示，其中 G 为二维数组，表示图中每个顶点到其他顶点的权重，Nv 为顶点数量。

Dijkstra: 这是用于执行 Dijkstra 算法的数据结构。其中 dist 数组用于存储从源点到每个顶点的最短路径，path 数组用于存储最短路径的路径，visited 数组用于标记每个节点是否已访问。

算法设计:

CreateGraph 函数: 输入顶点数量，然后构建邻接矩阵。

InitDijkstra 函数: 初始化 Dijkstra 数据结构，将源点到自身的距离设置为 0，源点到其他节点的距离设置为无穷大，所有节点都未访问。

FindMinDist 函数: 在未被访问的节点中找到距离源节点最近的节点。

Dijkstra 函数: 执行 Dijkstra 算法，计算出源点到其他所有节点的最短路径。

PrintPath 函数: 递归打印从源点到指定节点的最短路径。

伪代码描述如下:

函数 Dijkstra(Graph, D, S):

调用 InitDijkstra(Graph, D, S)

当 真时:

 V = FindMinDist(Graph, D)

 如果 V == -1 则 跳出循环

 D.visited[V] = 真

 对于 Graph 中的每个节点 W:

 如果 !D.visited[W] 且 Graph[V][W] < INF 则

 如果 D.dist[V] + Graph[V][W] < D.dist[W] 则

 D.dist[W] = D.dist[V] + Graph[V][W]

 D.path[W] = V

 结束如果

 结束如果

结束对于

结束当

结束函数

四 使用说明、测试分析及结果

第一行输入四个整数表示矩阵的行数和列数，矩阵 A 中非零元素的个数，矩阵 B 中非零元素的个数；然后依次输入矩阵 A 三元组，矩阵 B 的三元组；最后输入回车得到矩阵相加后的矩阵；

测试结果与分析:

输入

```
4
0 2 10 10000
2 0 7 3
10 7 0 6
10000 3 6 0
0 2
```

输出：

```
0
1
2
```

遇到的问题及解决方法

初始的问题是如何设计和实现 Dijkstra 算法。通过查阅相关的资料和文献，理解了 Dijkstra 算法的基本工作原理，最终实现了这个算法。

在实现算法的过程中，我们遇到了如何记录最短路径的问题。为了解决这个问题，设计了一个路径数组，每次更新最短路径时，同时更新路径数组。

在打印路径时，需要按照从源点到目标点的顺序打印出路径。为了实现这一点，采用了递归的方法，先打印出源点到目标点的父节点的路径，然后再打印目标点。

运行界面

```
nnerFile.cpp -o tempCodef
4
0 2 10 10000
2 0 7 3
10 7 0 6
10000 3 6 0
0 2
0
1
2
```

五、回顾讨论与实验总结

本次实验通过编程实现了 Dijkstra 算法，解决了求最短路径的问题。通过实验，更深入理解了 Dijkstra 算法的工作原理以及图的邻接矩阵表示。同时，也学习了如何设计和实现相关的数据结构，以及如何使用这些数据结构和算法解决实际问题。

4.3

◎实验题目：

用弗洛伊德算法求赋权图的两点间的最短路径长度

Description

用弗洛伊德算法求任意两点间的最短路径的长度

◎实验内容：

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），之后再输入一个小于100的正整数m，最后的m行每行输入两个不同的0到n-1之间的整数表示两个点。

Output

用弗洛伊德算法求任意两点间的最短路径的长度，并输出这两个点之间的最短路径的长度。

一、需求分析

本次实验的需求是实现一个能解决图中任意两点间最短路径的程序。从输入中读取图的信息，其中包括节点的数量，每个节点之间的距离。然后，通过使用 Floyd 算法计算图中任意两点之间的最短路径，并输出用户指定的两点之间的最短距离。

二、概要设计

为解决这个问题，首先需要建立一个图形数据结构来保存图形的信息，然后实现 Floyd 算法以找到任意两点之间的最短路径。Floyd 算法的核心思想是，对于图中的每一个节点，检查是否可以通过该节点将任何两点之间的距离缩短。

三、详细设计

数据结构设计：

Graph: 用于存储图的信息，其中包含节点数量(vnum)、两节点间的距离(arc)、最短路径信息(path)。

算法设计及伪代码描述：

使用的算法主要有三个函数：

1、init_Graph 函数，用于初始化图的数据结构：

```
function init_Graph(Graph G):  
    read G.vnum from input  
    for i=0 to G.vnum do  
        for j=0 to G.vnum do  
            read G.arc[i][j] from input  
            G.path[i][j] = -1
```

2、floyd 函数，用于执行 Floyd 算法找到最短路径：

```
function floyd(Graph G):  
    for m=0 to G.vnum do  
        for a=0 to G.vnum do  
            for b=0 to G.vnum do  
                if G.arc[a][b] > G.arc[a][m] + G.arc[m][b] then  
                    G.arc[a][b] = G.arc[a][m] + G.arc[m][b]  
                    G.path[a][b] = m
```

3、print_result 函数，用于打印出用户指定的两点间的最短距离：

```
function print_result(Graph G):  
    read n from input  
    for i=0 to n do  
        read a, b from input  
        print G.arc[a][b]
```

四 使用说明、测试分析及结果

先输入一个小于等于 100 的正整数，然后输入赋权图的邻接矩阵（10000 表示无穷大，并且任意一条简单路径的长度都小于 10000），最后输入 m 表示 m 对结点，之后一次输入 2m 个元素；

测试结果与分析；

输入：

```

4
0 2 10 10000
2 0 7 3
10 7 0 6
10000 3 6 0
2
0 2
9
3 0

```

输出

```

9
3 0
5

```

运行界面

```

(base) PS C:\Users\csp\Desktop>
4
0 2 10 10000
2 0 7 3
10 7 0 6
10000 3 6 0
2
0 2
9
3 0
5

```

遇到的问题及解决方法

在实现 Floyd 算法的过程中，主要的问题在于如何更新图中每两点之间的距离信息。通过查阅相关文献以及进行反复的实验，发现需要使用三重循环来遍历每个节点，并检查是否可以通过当前节点来缩短任何两点之间的距离。

五、回顾讨论与实验总结

通过这次实验，成功地实现了一个程序，可以在图中找到任意两点之间的最短路径。这个程序能够接受用户输入的图信息，并且应用 Floyd 算法来查找最短路径。通过这个实验，我对 Floyd 算法的原理以及实现方式有了进一步的理解，并成功地将其应用在实际问题上。

4.4

◎实验题目：

用弗洛伊德算法求赋权图中任意两点的最短路径

Description

用弗洛伊德算法求任意两点间的最短路径，并输出指定的m对结点间的最短路径。

◎实验内容：

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），之后再输入一个小于100的正整数m，最后的m行每行输入两个不同的0到n-1之间的整数表示两个点。

Output

用弗洛伊德算法求任意两点间的最短路径，并输出这两个点之间的最短路径。

一、需求分析

本次实验的目标是实现一种解决问题的程序，找出图中任意两点之间的最短路径，并且能够打印出这条路径。根据图的信息，包括节点的数量和每对节点之间的距离，使用 Floyd 算法计算图中任意两点之间的最短路径，然后打印出指定的两点之间的最短路径。

二、概要设计

为解决这个问题，首先需要建立一个图形数据结构和一个栈数据结构来保存图形的信息，然后实现 Floyd 算法以找到最短路径。在找到最短路径之后，程序会递归地找出这条路径上的每一个节点，并将这些节点存储在栈中。最后，程序会依次弹出栈中的元素，打印出路径。

三、详细设计

数据结构设计：

Graph：用于存储图的信息，其中包含节点数量(vnum)、两节点间的距离(arc)、最短路径信息(path)。

Stack：用于存储路径上的节点，其中包含栈顶指针(top)和数据区(data)。

算法设计及伪代码描述：

使用的算法主要有五个函数：

init_Graph 函数，用于初始化图的数据结构；

```
1、function init_Graph(G)
2、    read G.vnum
3、    for i from 0 to G.vnum-1 do
4、        for j from 0 to G.vnum-1 do
5、            read G.arc[i][j]
6、            G.path[i][j] = -1
```

floyd 函数，用于执行 Floyd 算法找到最短路径；

```
1、function floyd(G)
2、    for m from 0 to G.vnum-1 do
3、        for a from 0 to G.vnum-1 do
4、            for b from 0 to G.vnum-1 do
5、                if G.arc[a][b] > G.arc[a][m] + G.arc[m][b] then
6、                    G.arc[a][b] = G.arc[a][m] + G.arc[m][b]
7、                    G.path[a][b] = m
```

init_Stack、push_Stack 和 pop_Stack 函数，分别用于初始化栈、入栈和出栈；


```

1、 function init_Stack(S)
2、     S.top = -1
3、
4、 function push_Stack(S, e)
5、     S.top = S.top + 1
6、     S.data[S.top] = e
7、
8、 function pop_Stack(S)
9、     S.top = S.top - 1
10、    return S.data[S.top + 1]

```

find_path 函数,用于递归地找出路径上的每一个节点,并将这些节点存储在栈中;

```

1、 function find_path(G, S, a, b)
2、     push_Stack(S, b)
3、     if G.path[a][b] == -1 then
4、         push_Stack(S, a)
5、         return
6、     else
7、         find_path(G, S, a, G.path[a][b])

```

print_result 函数,用于依次弹出栈中的元素,打印出路径。

```

1、 function print_result(G, S)
2、     read n
3、     for i from 0 to n-1 do
4、         read a, b
5、         init_Stack(S)
6、         find_path(G, S, a, b)
7、         while S.top > -1 do
8、             print pop_Stack(S)

```

四 使用说明、测试分析及结果

先输入一个小于等于 100 的正整数,然后输入赋权图的邻接矩阵(10000 表示无穷大,并且任意一条简单路径的长度都小于 10000),然后输入正整数 m,最后 m 行每行输入两个 0 到 n-1 的数字;

测试结果与分析;

输入:

```

0 2 10 10000
2 0 7 3
10 7 0 6
10000 3 6 0
2
0 2

```

```
3 0
```

输出：

```
0  
1  
2
```

```
3  
1  
0
```

遇到的问题及解决方法

在实现路径查找和打印的过程中，需要解决如何有效地存储和查找路径上的节点。这个问题的解决方法是使用栈数据结构，栈的先入后出特性恰好符合路径的查找和打印顺序。

运行界面

```
(base) PS C:\Users\csp\Desktop>  
0 2 10 10000  
2 0 7 3  
10 7 0 6  
10000 3 6 0  
2  
0 2  
0  
1  
2  
3 0  
3  
1  
0
```

五、回顾讨论与实验总结

通过这次实验，我实现的算法可以在图中找到任意两点之间的最短路径，并打印出这条路径，应用 Floyd 算法来查找最短路径。我对 Floyd 算法的原理以及实现方式有了进一步的理解，并成功地将其应用在实际问题上。同时，我也对栈这种数据结构有了更深入的理解，知道了如何利用其特性来解决实际问题。

教师评语：

实验成绩：

指导教师签名：

批阅日期：