《数据结构》实验报告

班　级:U10P53013.04　　姓　名:梁桐　　学　号:2023370018
E-mail:3211583077@qq.com　　日　期: 2024.5.26

◎实验题目:

求赋权图中一个结点到所有结点的最短路径的长度

◎实验内容:

自选算法实现单源最短路径长度的求解

一 需求分析

1.

给一个赋权图（无向图），求 0 号结点到其余结点的最短路径的长度。

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）。

按结点编号顺序输出 0 号结点所有结点的最短路径的长度。

2. 演示程序以用户和计算机的对话方式执行，由用户在键盘上输入相应数据

3. 程序执行的命令包括：

（1）输入图的顶点数及邻接矩阵；（2）执行选定算法计算单源最短路径；（3）输出最短路径距离。

4. 测试数据

（1）

顶点数：6

邻接矩阵：

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

最短路径距离：

0

1

3

7

4

9

二 概要设计

1. 基本操作

1.1. dijkstra():

这是 Dijkstra 算法的实现函数。它负责计算给定图中各个节点之间的最短路

1.2.

print_shortest_path(): 用于打印计算得到的最短路径。

1.3.

main(): 这是程序的入口函数。读取输入数据

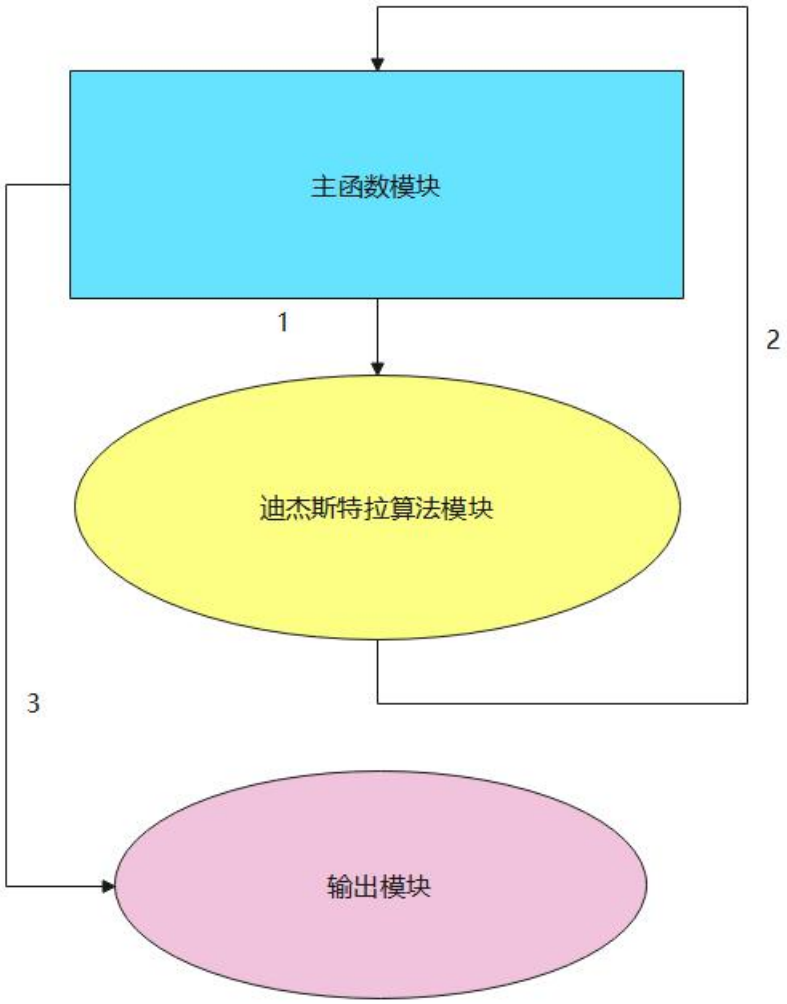
2. 本程序包含三个模块：

迪杰斯特拉算法模块

主函数模块

输出模块

函数模块调用图：



三 详细设计

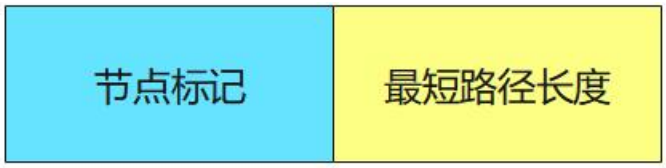
伪代码如下

1. 定义 MAXSIZE 为 105
2. 定义 INF 为 10000

3. 定义全局变量：
    3.1 vertex_count 初始化为 0
    3.2 distance[MAXSIZE][MAXSIZE] 初始化为全零
    3.3 visited[MAXSIZE] 初始化为全 false
    3.4 shortest_path[MAXSIZE] 初始化为全 INF

4. 定义函数 dijkstra():
    4.1 初始化 shortest_path 数组为全 INF
    4.2 将起始节点标记为已访问，并将其到其他节点的距离更新到 shortest_path 数组中
    4.3 进行 vertex_count - 1 次迭代：
        4.3.1 找出当前未访问节点中距离起始节点最近的节点
            4.3.1.1 设置 min_path_length 为 INF
            4.3.1.2 对于每个节点 j：
                如果节点 j 未被访问且 shortest_path[j] 小于 min_path_length：
                    更新 min_path_length 为 shortest_path[j]
                    设置 next_vertex 为 j
        4.3.2 标记节点 next_vertex 为已访问
        4.3.3 更新当前节点的邻接节点的最短路径：
            4.3.3.1 对于每个节点 j：
                如果节点 j 未被访问且 distance[next_vertex][j] 不为 INF：
                    计算 tmp 为 shortest_path[next_vertex] + distance[next_vertex][j]
                    如果 tmp 小于 shortest_path[j]：
                        更新 shortest_path[j] 为 tmp

Dijkstra 算法的实现函数负责计算给定图中各个节点之间的最短路径。具体流程包括：
初始化最短路径数组为无穷大(INF)。
将起始节点标记为已访问，并将起始节点到其他节点的距离更新到最短路径数组中。
进行 vertex_count - 1 次迭代（vertex_count 是节点数量），每次迭代选择当前未访问节点中距离起始节点最近的节点，并更新该节点的邻接节点的最短路径。



5. 定义函数 print_shortest_path():
    5.1 对于每个节点 i：
        5.1.1 打印 shortest_path[i]

print_shortest_path(): 这个函数用于打印计算得到的最短路径。它会依次打印出每个节点到起始节点的最短距离。

6. 定义函数 main():

6.1 从标准输入读取节点数量 vertex_count

6.2 对于每个节点 i 从 0 到 vertex_count-1：

6.2.1 对于每个节点 j 从 0 到 vertex_count-1：

读取距离矩阵中的元素并存储到 distance[i][j]

6.3 调用 dijkstra() 函数计算最短路径

6.4 调用 print_shortest_path() 函数打印结果

main()是程序的入口函数。它首先从标准输入读取节点数量(vertex_count)，然后读取一个矩阵，矩阵的大小为 vertex_count × vertex_count，表示图中各个节点之间的距离。接着调用 dijkstra() 函数计算最短路径，并调用 print_shortest_path() 函数打印结果。

四 使用说明、测试分析及结果

1、 程序使用说明；

进入演示程序后：

输入表示图的顶点数 n

输入 n 行 n 列的邻接矩阵，

输出 0 号顶点到其余顶点的最短路径距离。

2、 测试结果与分析；

当输入图的顶点数为 6,

邻接矩阵为：

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

则输出正确为：

0

1

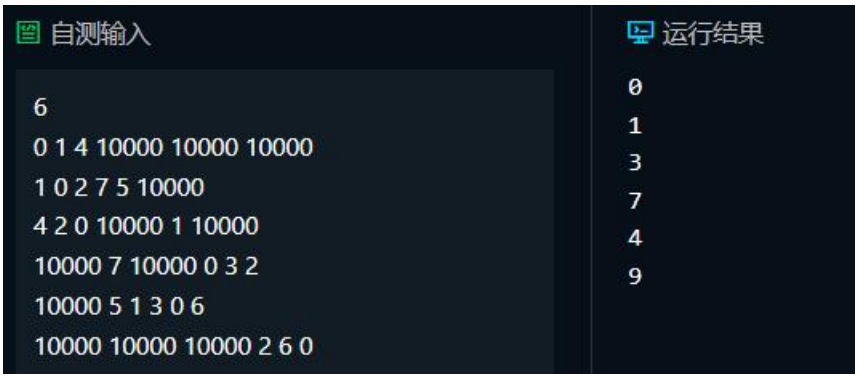
3

7

4

9

3、 运行界面



五、实验总结

1. Dijkstra 算法的理解：通过实现 Dijkstra 算法，我更深入地理解了它的工作原理。我学会了如何使用优先队列或简单的数组来实现该算法，以及如何在图中找到最短路径。

2. 问题分解与解决能力：在编写程序的过程中，我遇到了一些问题，比如处理输入数据、设计数据结构等。通过分析问题、查找资料和与同学讨论，我逐步解决了这些问题，提高了自己的问题分解和解决能力。

《数据结构》实验报告

班　级:U10P53013.04　　姓　名:梁桐　　学　号:2023370018
E-mail:3211583077@qq.com　　日　期: 2024.5.26

◎实验题目:

用迪杰斯特拉算法求一点到其余所有结点的最短路径

◎实验内容:

Dijkstra 算法实现单源最短路径的求解

一 需求分析

1.

用迪杰斯特拉算法求一点到其余所有结点的最短路径。

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）。

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)。

2. 演示程序以用户和计算机的对话方式执行，由用户在键盘上输入相应数据

3. 程序执行的命令包括：

（1）输入图的顶点数及邻接矩阵；（2）执行 dijkstra 算法计算单源最短路径；（3）输出给定两点之间的最短路径。

4. 测试数据

顶点数：4

邻接矩阵：
0 2 10 10000
2 0 7 3
10 7 0 6
10000 3 6 0
给定的两个顶点：0 2
最短路径：
0
1
2

二 概要设计

2. 基本操作

2.1. dijkstra():

这是 Dijkstra 算法的实现函数。它负责计算给定图中各个节点之间的最短路

1.2.

print_shortest_path(): 用于打印计算得到的最短路径。

此函数使用了栈操作

1.3.

main(): 这是程序的入口函数。读取输入数据

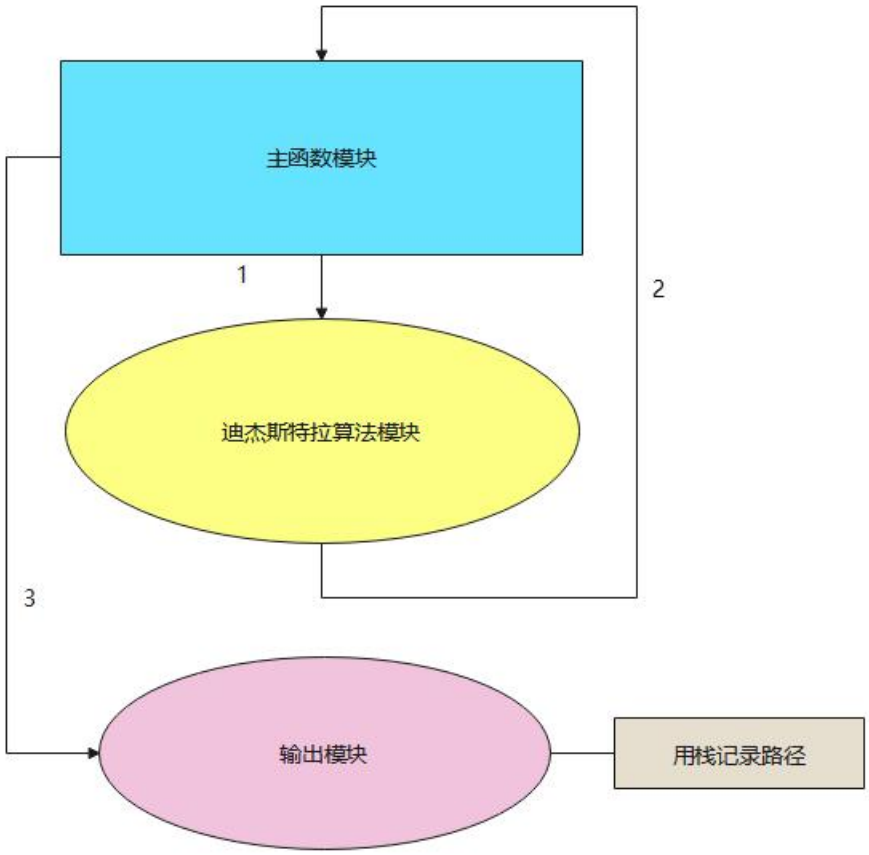
2. 本程序包含三个模块：

迪杰斯特拉算法模块

主函数模块

输出模块

函数模块调用图：



三 详细设计

1. 定义 MAXSIZE 为 105
2. 定义 INF 为 10000

3. 定义全局变量：
   3.1 vertex_count 初始化为 0
   3.2 distance[MAXSIZE][MAXSIZE] 初始化为全零
   3.3 visited[MAXSIZE] 初始化为全 false
   3.4 shortest_path[MAXSIZE] 初始化为全 INF
   3.5 path_stack1[MAXSIZE] 初始化为全零
   3.6 stack_top1 初始化为 -1
   3.7 path_stack2[MAXSIZE] 初始化为全零
   3.8 stack_top2 初始化为 -1

4. 定义函数 dijkstra_algorithm():
   4.1 初始化 shortest_path 数组为全 INF

4.2 将起始节点标记为已访问，并将其到其他节点的距离更新到 shortest_path 数组中

4.3 进行 vertex_count - 1 次迭代：

4.3.1 找出当前未访问节点中距离起始节点最近的节点

4.3.2 标记该节点为已访问

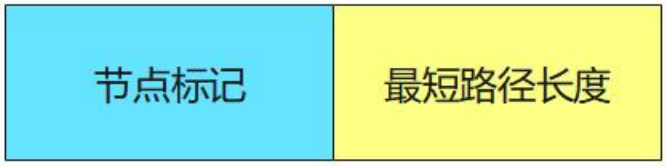
4.3.3 更新当前节点的邻接节点的最短路径

Dijkstra 算法的实现函数负责计算给定图中各个节点之间的最短路径。具体流程包括：

初始化最短路径数组为无穷大(INF)。

将起始节点标记为已访问，并将起始节点到其他节点的距离更新到最短路径数组中。

进行 vertex_count - 1 次迭代（vertex_count 是节点数量），每次迭代选择当前未访问节点中距离起始节点最近的节点，并更新该节点的邻接节点的最短路径。



5. 定义函数 print_shortest_path()：

5.1 从标准输入读取起始节点和结束节点 start_vertex, end_vertex

5.2 将 start_vertex1, end_vertex1 初始化为 start_vertex, end_vertex

5.3 将 start_vertex2, end_vertex2 初始化为 end_vertex, start_vertex

5.4 将 end_vertex1 入栈到 path_stack1 中

5.5 将 start_vertex2 入栈到 path_stack2 中

5.6 寻找第一条最短路径：

5.6.1 当 end_vertex1 不等于 start_vertex1 时，循环执行以下操作：

5.6.1.1 对于每个节点 i：

如果距离矩阵中 distance[i][end_vertex1] 不为 INF，且 shortest_path[i] 小于 shortest_path[end_vertex1]，

且 shortest_path[i] 加上 distance[i][end_vertex1] 等于 shortest_path[end_vertex1]：

将节点 i 入栈到 path_stack1 中

5.6.1.2 如果循环次数超过 105，则跳出循环

5.7 如果超过节点数，则寻找另一条最短路径：

5.7.1 当 end_vertex2 不等于 start_vertex2 时，循环执行以下操作：

5.7.1.1 对于每个节点 i：

如果距离矩阵中 distance[i][end_vertex2] 不为 INF，且 shortest_path[i] 小于 shortest_path[end_vertex2]，

且 shortest_path[i] 加上 distance[i][end_vertex2] 等于 shortest_path[end_vertex2]：

将节点 i 入栈到 path_stack2 中

5.8 输出最短路径：

5.8.1 当 path_stack1 的栈顶大于 -1 且不需要寻找另一条最短路径时，循环执行以

下操作：

5.8.1.1 出栈并打印 path_stack1 的栈顶元素

5.8.2 当 path_stack2 的栈顶大于 -1 且需要寻找另一条最短路径时，循环执行以下操作：

5.8.2.1 出栈并打印 path_stack2 的栈顶元素

print_shortest_path() 函数的作用是根据 Dijkstra 算法计算出的最短路径信息，从起始节点到目标节点打印出最短路径上的所有节点。具体分析如下：

函数输入：

该函数从标准输入中读取起始节点和目标节点的信息，并将其存储在变量 start_vertex 和 end_vertex 中。这些信息用于确定要打印的最短路径。

路径存储：

函数中定义了两个栈 path_stack1 和 path_stack2，分别用于存储第一条最短路径和可能的第二条最短路径上的节点。

使用 stack_top1 和 stack_top2 分别记录两个栈的栈顶位置。

寻找路径：

首先，从目标节点 end_vertex 出发，通过循环迭代，依次寻找与当前节点相邻且最短路径长度更小的节点，直到回到起始节点。

如果迭代次数超过了节点数量，即 105 次，表示可能存在环路或其他问题，因此将寻找另一条最短路径。

打印路径：

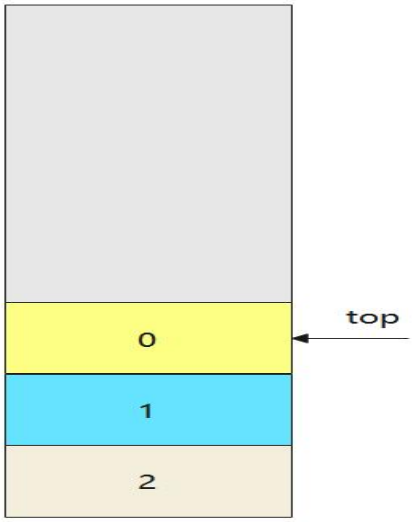
当找到第一条最短路径时，函数通过依次出栈 path_stack1 中的节点，从起始节点开始依次打印出最短路径上的节点。

如果迭代次数超过了节点数量，需要寻找另一条最短路径，则从 path_stack2 中出栈并打印节点。

输出结果：

打印函数将最短路径上的节点依次输出到标准输出中。

通过这些步骤，print_shortest_path() 函数完成了根据 Dijkstra 算法计算的最短路径信息的解析和打印操作，向用户提供了从起始节点到目标节点的最短路径。



6. 定义函数 main():
   6.1 从标准输入读取节点数量 vertex_count
   6.2 读取节点间的距离信息：
      6.2.1 对于每个节点 i 从 0 到 vertex_count-1：
            对于每个节点 j 从 0 到 vertex_count-1：
               从标准输入读取距离信息并存储到 distance[i][j]
   6.3 调用 dijkstra_algorithm() 函数计算最短路径
   6.4 调用 print_shortest_path() 函数打印结果

main()是程序的入口函数。它首先从标准输入读取节点数量(vertex_count)，然后读取一个矩阵，矩阵的大小为 vertex_count × vertex_count，表示图中各个节点之间的距离。接着调用 dijkstra() 函数计算最短路径，并调用 print_shortest_path() 函数打印结果。

四 使用说明、测试分析及结果

4、 程序使用说明；

进入演示程序后：

输入表示图的顶点数 n

输入 n 行 n 列的邻接矩阵，

输入两个给定顶点

输出两个顶点之间的最短路径。

5、 测试结果与分析；

当输入图的顶点数为 4,

邻接矩阵为：

0 2 10 10000

2 0 7 3

10 7 0 6

10000 3 6 0
给定的两个顶点：0 2
则输出正确为：
0
1
2

6、 运行界面

五、实验总结

1. Dijkstra 算法的理解：通过实现 Dijkstra 算法，我更深入地理解了它的工作原理。我学会了如何使用优先队列或简单的数组来实现该算法，以及如何在图中找到最短路径。
2. 问题分解与解决能力：在编写程序的过程中，我遇到了一些问题，比如处理输入数据、设计数据结构等。通过分析问题、查找资料和与同学讨论，我逐步解决了这些问题，提高了自己的问题分解和解决能力。
3.在实验 4.1 的基础上，深化了对栈的使用理解。

《数据结构》实验报告

班　级:U10P53013.04　　姓　名:梁桐　　学　号:2023370018
E-mail:3211583077@qq.com　　日　期: 2024.5.26

◎实验题目:

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:

floyd 算法实现单源最短路径长度的求解

一 需求分析

1.

用弗洛伊德算法求任意两点间的最短路径长度，并输出指定的 m 对结点间的最短路径长度。

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。

用弗洛伊德算法求任意两点间的最短路径长度，并输出这些两个点之间的最短路径长度。

2. 演示程序以用户和计算机的对话方式执行，由用户在键盘上输入相应数据

3. 程序执行的命令包括：

（1）输入图的顶点数及邻接矩阵；（2）执行 floyd 算法计算最短路径长度；（3）输出给定 m 对两点之间的最短路径长度。

4. 测试数据

顶点数：4

邻接矩阵：

0 2 10 10000

2 0 7 3

10 7 0 6

10000 3 6 0

给定的顶点对数：2

顶点对：

0 2

3 0

最短路径长度：

9

5

二 概要设计

1. 基本操作：

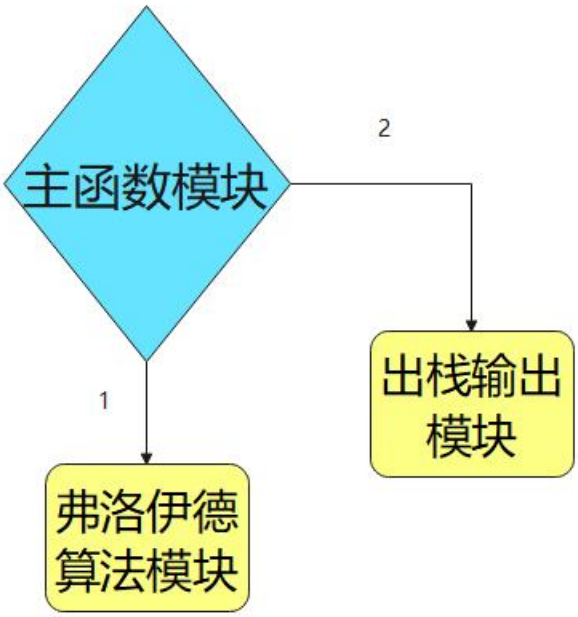
floydWarshall() 函数

作用：

该函数实现了弗洛伊德-沃尔沙尔算法，用于计算图中所有节点之间的最短路径。它会更新每对节点之间的最短路径，并记录路径中经过的中间节点。

printShortestPaths()函数

作用：
读取用户输入的查询，并打印出每个查询对应的最短路径长度。
main 函数
作用：
主函数负责程序的初始化和整体控制。
弗洛伊德算法模块
输出模块
主函数模块
函数模块调用图：



三 详细设计
程序伪代码

```
// 伪代码开始
// 定义常量
CONST MAX_NODES = 102
// 初始化图的邻接矩阵
DECLARE graph[MAX_NODES][MAX_NODES]
// 初始化节点数
DECLARE numNodes = 0
```

常量和变量初始化

定义 MAX_NODES 常量表示节点数的最大值。
定义 graph 矩阵存储图的邻接矩阵。
定义 numNodes 变量存储节点的数量。
floydWarshall 函数

```
// 弗洛伊德-沃尔沙尔算法函数
FUNCTION floydWarshall():
    FOR i FROM 0 TO numNodes - 1:
        FOR j FROM 0 TO numNodes - 1:
            FOR k FROM 0 TO numNodes - 1:
                IF graph[j][i] + graph[i][k] < graph[j][k]:
                    graph[j][k] = graph[j][i] + graph[i][k]
                END IF
            END FOR
        END FOR
    END FOR
END FUNCTION
```

这是弗洛伊德-沃尔沙尔算法的实现，用于计算所有节点对之间的最短路径。
遍历每个节点作为中间节点 i，检查每对节点 j 和 k 之间的路径长度是否可以通过 i 节点变短，如果是，则更新路径长度。

```
// 打印最短路径的函数
FUNCTION printShortestPaths():
    // 读取查询的数量
    INPUT numQueries
    FOR i FROM 0 TO numQueries - 1:
        // 读取起始节点和目标节点
        INPUT nodeA, nodeB
        // 输出最短路径长度
        OUTPUT graph[nodeA][nodeB]
    END FOR
END FUNCTION
```

读取用户输入的查询数量 numQueries。
针对每个查询，读取起始节点 nodeA 和目标节点 nodeB，并输出从 nodeA 到 nodeB 的最短路径长度。

```
// 主函数
FUNCTION main():
    // 读取节点的数量
    INPUT numNodes

    // 读取图的邻接矩阵
    FOR i FROM 0 TO numNodes - 1:
```

```
        FOR j FROM 0 TO numNodes - 1:
            INPUT graph[i][j]
        END FOR
    END FOR

    // 计算所有节点之间的最短路径
    CALL floydWarshall()

    // 打印查询的最短路径
    CALL printShortestPaths()

    RETURN 0
END FUNCTION

// 调用主函数
CALL main()

// 伪代码结束
读取节点的数量 numNodes。
读取图的邻接矩阵 graph。
调用 floydWarshall 函数计算所有节点对之间的最短路径。
调用 printShortestPaths 函数处理用户的查询并打印最短路径长度。
返回 0，表示程序正常结束。
```

四 使用说明、测试分析及结果

程序使用说明；

进入演示程序后：

输入表示图的顶点数 n

输入 n 行 n 列的邻接矩阵，

输入 m

输入 m 对给定顶点

输出 m 对顶点之间的最短路径。

测试结果与分析；

4. 测试数据

顶点数：4

邻接矩阵：

0 2 10 10000

2 0 7 3

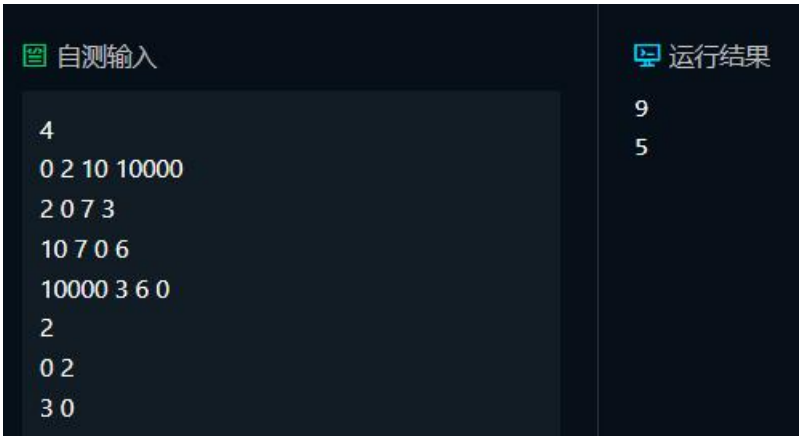
10 7 0 6

10000 3 6 0

给定的顶点对数：2

顶点对：
0 2
3 0
最短路径长度：
9
5

运行界面



五、实验总结

本次实验的主要目的是实现和理解弗洛伊德-沃尔沙尔算法（Floyd-Warshall Algorithm），通过它来计算图中所有节点对之间的最短路径，并能根据用户输入查询特定节点对之间的最短路径长度。

理解算法的基本原理是编写正确程序的前提。弗洛伊德-沃尔沙尔算法通过动态规划的方式逐步更新路径长度，这一过程如果不理解清楚，很容易在实现时出现逻辑错误。

代码结构的设计:

将代码分为读取输入、计算最短路径、处理查询和输出结果等几个模块，使得程序结构清晰，便于调试和维护。尤其是在处理复杂算法时，良好的代码结构能够显著提高开发效率。

边界情况的处理:

边界情况和异常情况往往是程序出错的主要原因。通过实验，深刻体会到在设计和实现算法时，需要充分考虑各种边界情况，并进行相应的处理。

《数据结构》实验报告

班　级:U10P53013.04　　姓　名:梁桐　　学　号:2023370018
E-mail:3211583077@qq.com　　日　期: 2024.5.26

◎实验题目:

用弗洛伊德算法求赋权图中任意两点间的最短路径

◎实验内容:

floyd 算法实现单源最短路径的求解

一 需求分析

1.

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。

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。

用弗洛伊德算法求任意两点间的最短路径，并输出这些两个点之间的最短路径。

2. 演示程序以用户和计算机的对话方式执行，由用户在键盘上输入相应数据

3. 程序执行的命令包括：

（1）输入图的顶点数及邻接矩阵；（2）执行 floyd 算法计算最短路径；（3）输出给定 m 对两点之间的最短路径。

4. 测试数据

顶点数：4

邻接矩阵：

0 2 10 10000

2 0 7 3

10 7 0 6

10000 3 6 0

给定的顶点对数：2

顶点对：

0 2

3 0

最短路径：

0

1

2

3

1

0

二 概要设计

2. 基本操作：

floydWarshall() 函数

作用：

该函数实现了弗洛伊德-沃尔沙尔算法，用于计算图中所有节点之间的最短路径。它会更新每对节点之间的最短路径，并记录路径中经过的中间节点。

findPath()函数

作用：

该函数递归地重建从起始节点到目标节点的路径，并将路径节点存储在栈中。

printPaths 函数

作用：

该函数处理用户查询，输出从起始节点到目标节点的最短路径。

main 函数

作用：

主函数负责程序的初始化和整体控制。

2. 本程序包含四个模块：

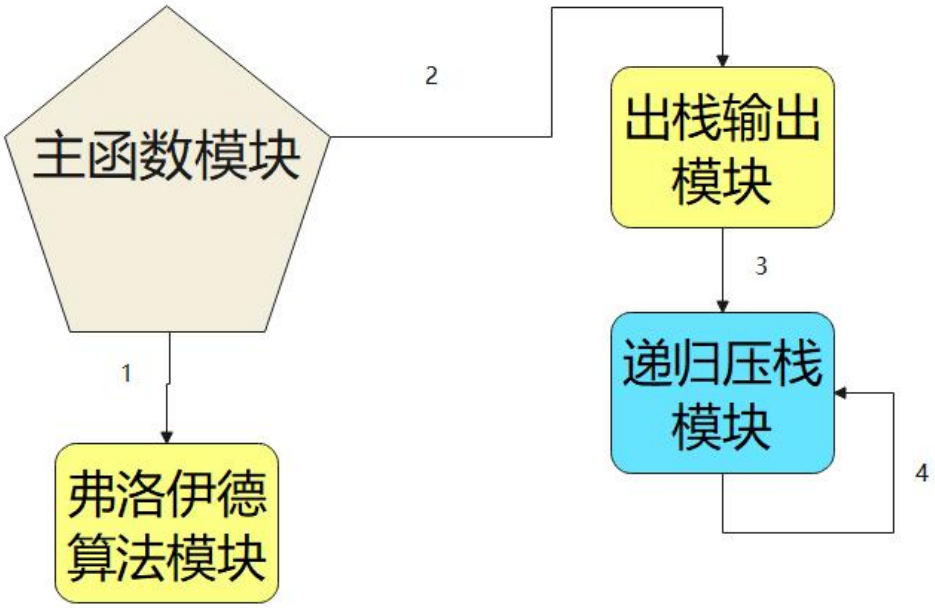
弗洛伊德算法模块

递归压栈模块

出栈输出模块

主函数模块

函数模块调用图：



三 详细设计

伪代码如下：

定义常量 MAX_NODES 为 102
声明 graph[MAX_NODES][MAX_NODES] 用于存储图的邻接矩阵
声明 path[MAX_NODES][MAX_NODES] 用于存储路径信息
声明 stack[MAX_NODES] 用于存储路径的栈
声明变量 numNodes 用于存储节点数量
声明变量 top 为栈顶指针，初始值为 -1

函数 floydWarshall():
    遍历每个节点 k:
        遍历每个节点 i:
            遍历每个节点 j:
                如果通过节点 k 的路径比直接路径更短:
                    更新 graph[i][j] 为通过 k 的最短路径
                    更新 path[i][j] 为中间节点 k

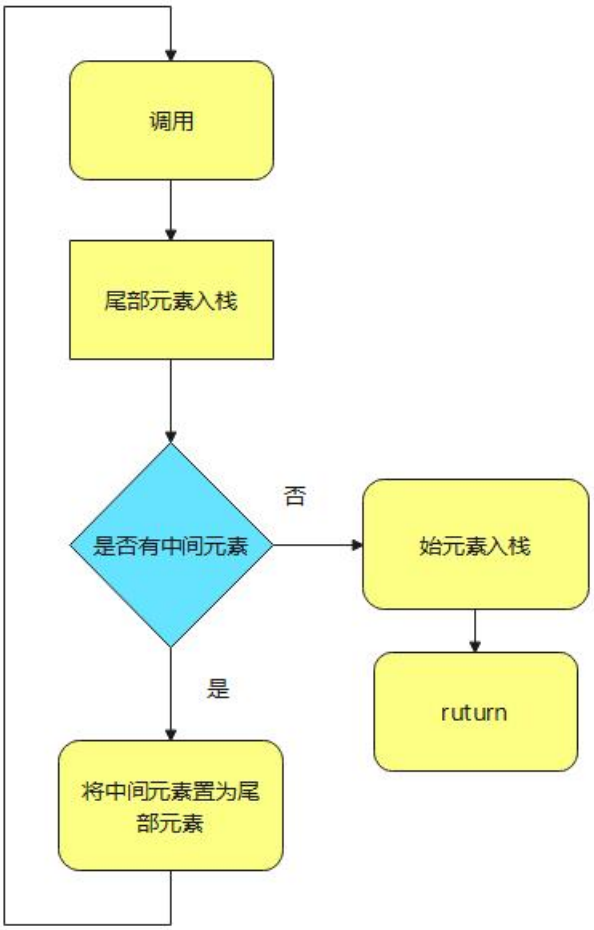
该函数实现了弗洛伊德算法，用于计算图中所有节点之间的最短路径。它会更新每对节点之间的最短路径，并记录路径中经过的中间节点。

1. 遍历每一个节点 `k`，将其作为中间节点。
2. 对于每一对节点 `(i, j)`，检查是否通过节点 `k` 的路径比当前已知的直接路径更短。
3. 如果通过节点 `k` 的路径更短，则更新从 `i` 到 `j` 的最短路径长度。
4. 记录中间节点 `k`，表示路径从 `i` 到 `j` 是通过 `k` 达成的。

函数 findPath(startNode, endNode):
    将 endNode 压入栈
    如果 path[startNode][endNode] 为 -1:
        将 startNode 压入栈
        返回
    递归调用 findPath(startNode, path[startNode][endNode])

该函数递归地重建从起始节点到目标节点的路径，并将路径节点存储在栈中。

1. 将目标节点 `endNode` 压入栈。
2. 检查路径数组 `path`，如果起始节点到目标节点之间没有中间节点（即 `path[startNode][endNode] == -1`），则将起始节点 `startNode` 压入栈，路径查找结束。
3. 如果有中间节点，则递归调用 `findPath` 函数，查找从起始节点到中间节点的路径。




```
函数 printPaths():
    读取查询次数 numQueries
    遍历每个查询:
        读取起始节点 startNode 和目标节点 endNode
        重置栈顶指针 top 为 -1
        调用 findPath(startNode, endNode)
        当栈不为空时:
            输出栈顶节点
            弹出栈顶节点

主函数 main():
    读取节点数量 numNodes
    遍历每个节点 i:
        遍历每个节点 j:
            读取 graph[i][j] 的值
            初始化 path[i][j] 为 -1
    调用 floydWarshall() 计算所有节点之间的最短路径
```

调用 printPaths() 打印查询的路径

返回 0

四 使用说明、测试分析及结果

程序使用说明；

进入演示程序后：

输入表示图的顶点数 n

输入 n 行 n 列的邻接矩阵，

输入 m

输入 m 对给定顶点

输出 m 对顶点之间的最短路径。

测试结果与分析；

当输入图的顶点数为 4,

邻接矩阵为：

0 2 10 10000

2 0 7 3

10 7 0 6

10000 3 6 0

顶点对数为：2

顶点对：

0 2

3 0

则输出正确为：

0

1

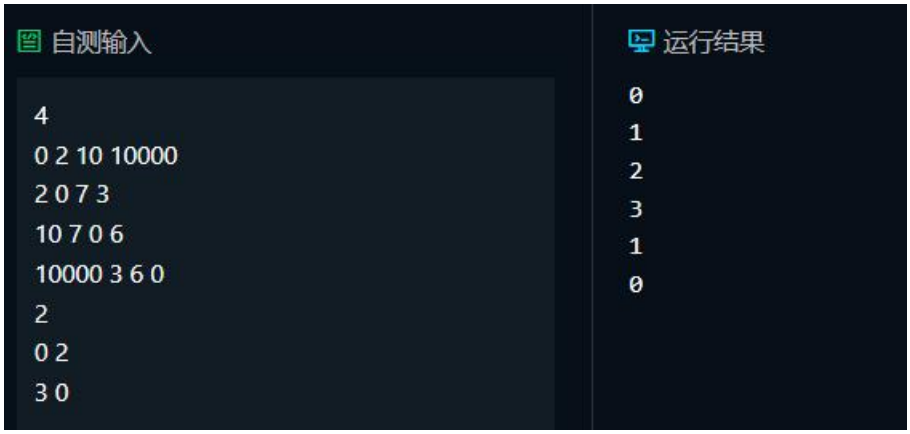
2

3

1

0

运行界面



六、实验总结

本次实验的主要目的是实现和理解弗洛伊德-沃尔沙尔算法（Floyd-Warshall Algorithm），通过它来计算图中所有节点对之间的最短路径，并能根据用户输入查询特定节点对之间的最短路径长度。

在本次实验中，我使用栈结构来重建并输出从起始节点到目标节点的最短路径，这一方法不仅直观且有效地解决了路径输出问题。通过递归函数 findPath，我们将路径上的节点依次压入栈中，然后通过栈的后进先出特性，逆序输出栈中的节点，从而正确重建并输出了路径。

教师评语：

实验成绩：

指导教师签名：

批阅日期：