**合肥工业大学**

**物联网工程专业**

姓 名： 党存远

学 号： 2022217587

班 级： 物联网工程22-2

实验地点： 计算机中心楼

实验时间： 12.13

1. 实验目的和要求

实验二:最短路径问题

实验目的:

运用最短路思想和求最短路程序解决实际问题

实验过程:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **城市-城市** | **通行费用** | **城市-城市** | **通行费用** | **城市-城市** | **通行费用** |
| A-C | 1892 | H-I | 1100 | Q-T | 375 |
| B-C | 216 | H-J | 967 | Q-Y | 622 |
| C-D | 676 | I-J | 639 | R-S | 674 |
| C-E | 1145 | J-K | 902 | R-U | 704 |
| D-H | 842 | J-L | 607 | S-T | 651 |
| D-G | 511 | K-L | 672 | U-V | 305 |
| E-F | 668 | K-N | 675 | U-X | 397 |
| F-G | 695 | K-Q | 528 | V-W | 242 |
| F-R | 137 | L-M | 255 | X-Z | 550 |
| G-S | 349 | N-O | 675 |  |  |
| G-Q | 534 | O-P | 140 |  |  |

请为旅客设计两个城市间的最短路径和所有城市间最短路径。

实验要求：

(1)选择一种编程语言，利用Dijkstra算法或Floyd-Warshall算法计算最优路径

输入：选择模式（单源或多源）

单源模式：

输入：邻接矩阵，源点城市与目的城市

输出：最短路径和最少花费

多源模式：

输入：邻接矩阵

输出：可达矩阵（包含每两个城市间最少花费）

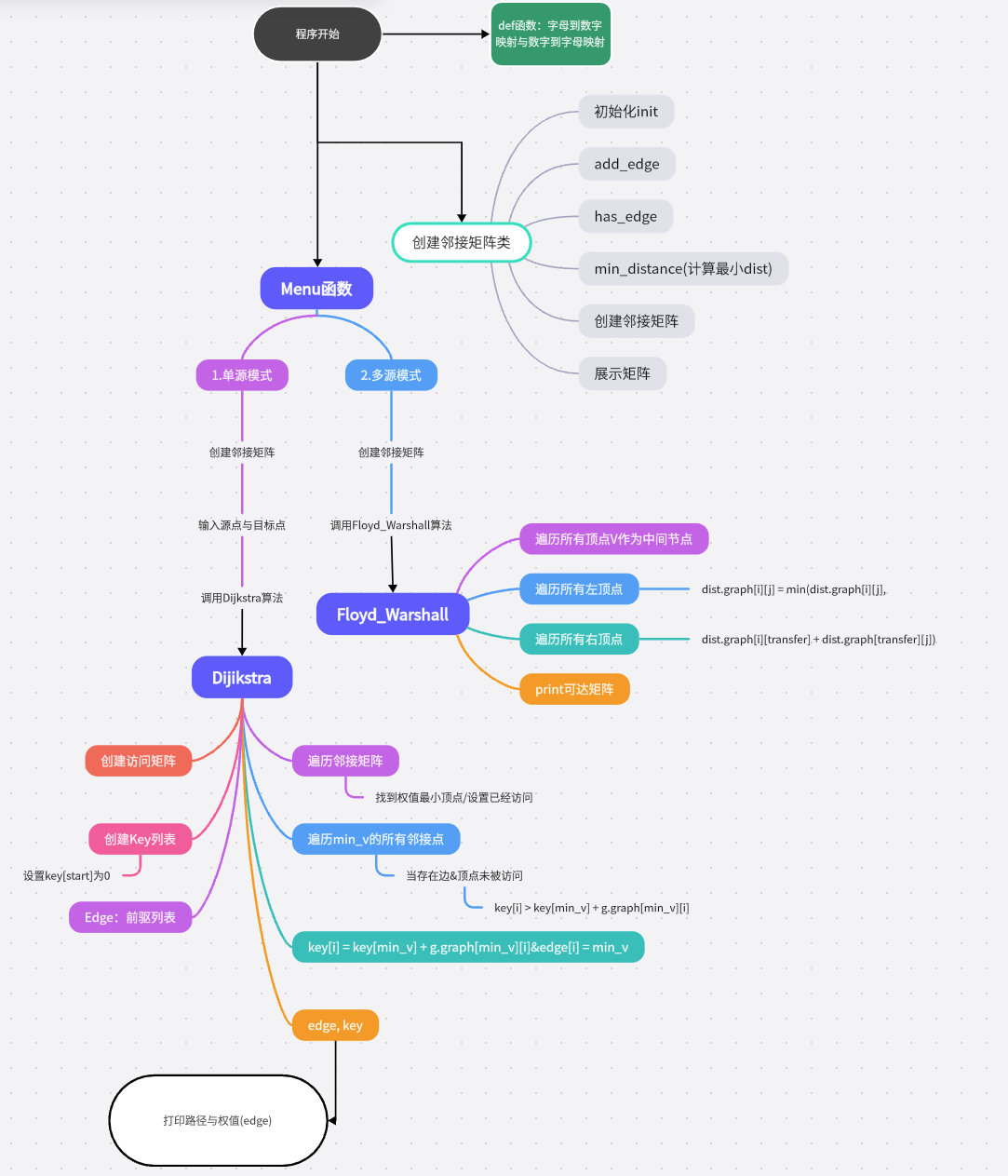
1. 实验环境和工具

编译器：Pycharm

运行环境：Anaconda

1. 实验结果
   1. 算法流程图

实验二算法流程图:



* 1. 程序核心代码

实验二:计算单源与多源最短路径

1、Dijkstra算法实现

def Dijkstra(g: Graph, start: int):  
 visit\_list = [False] \* g.size # 记录顶点是否已访问  
 key = [INF] \* g.size # 记录权值  
 edge = [-1] \* g.size  
 key[start] = 0  
 for \_ in range(g.size):  
 min\_v = g.min\_distance(key, visit\_list)  
 visit\_list[min\_v] = True  
 for i in range(g.size):  
 if (g.graph[min\_v][i] != INF) and not (visit\_list[i]) and key[i] > key[min\_v] + g.graph[min\_v][i]:  
 key[i] = key[min\_v] + g.graph[min\_v][i]  
 edge[i] = min\_v  
 return edge, key

2、Floyd\_Warshall算法实现:

def Floyd\_Warshall(g: Graph):  
 dist = CreateGraph(g.size)  
 for transfer in range(dist.size):  
 for i in range(g.size):  
 for j in range(g.size):  
 dist.graph[i][j] = min(dist.graph[i][j], dist.graph[i][transfer] + dist.graph[transfer][j])  
 return dist

3、递归调用(打印路径):

def print\_path(parent, j):  
 if parent[j] == -1: # 源点  
 print(vertex\_to\_letter(j), end=" ")  
 return  
 print\_path(parent, parent[j])  
 print(vertex\_to\_letter(j), end=" ")

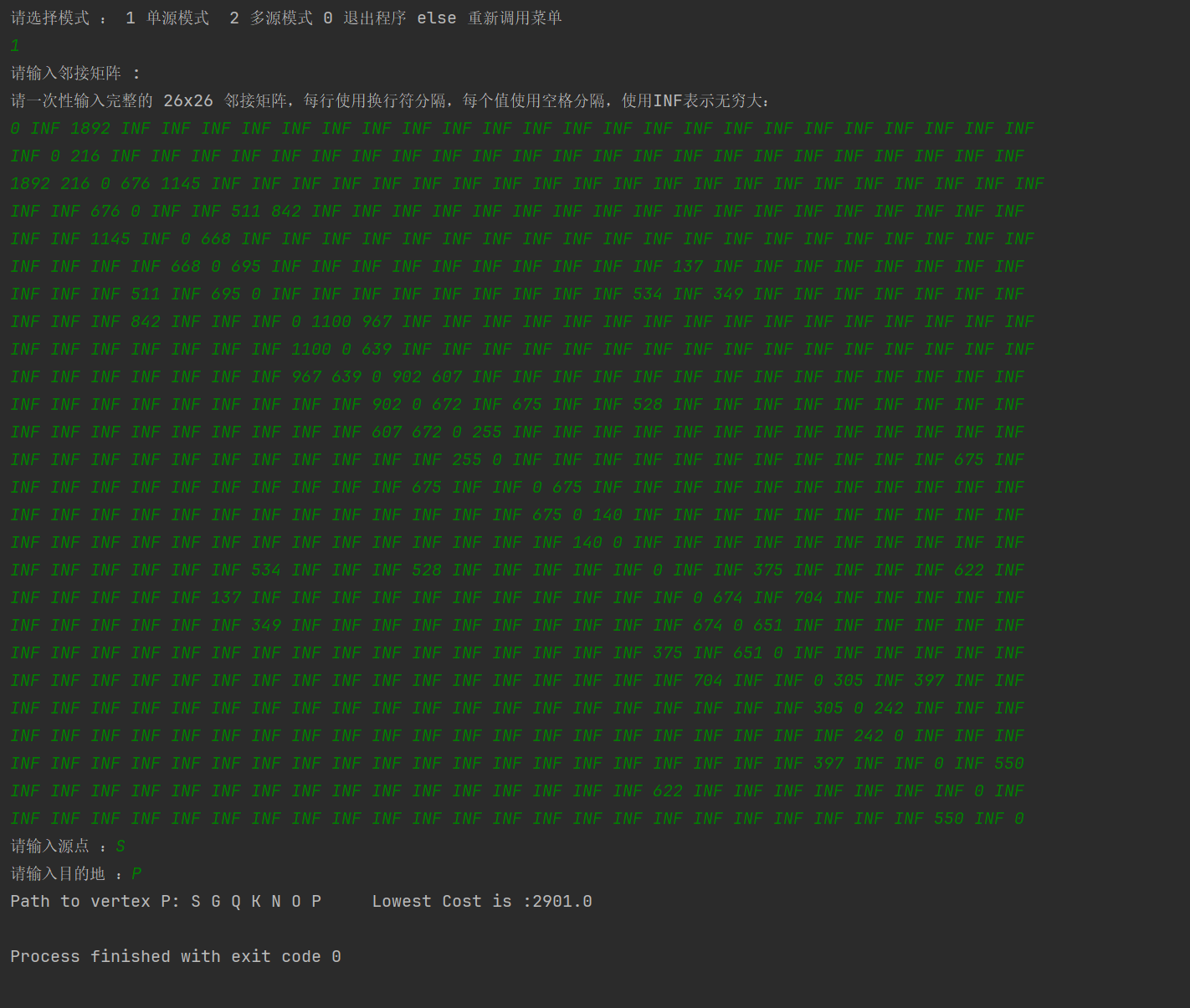
实验二测试数据

程序输入内容:

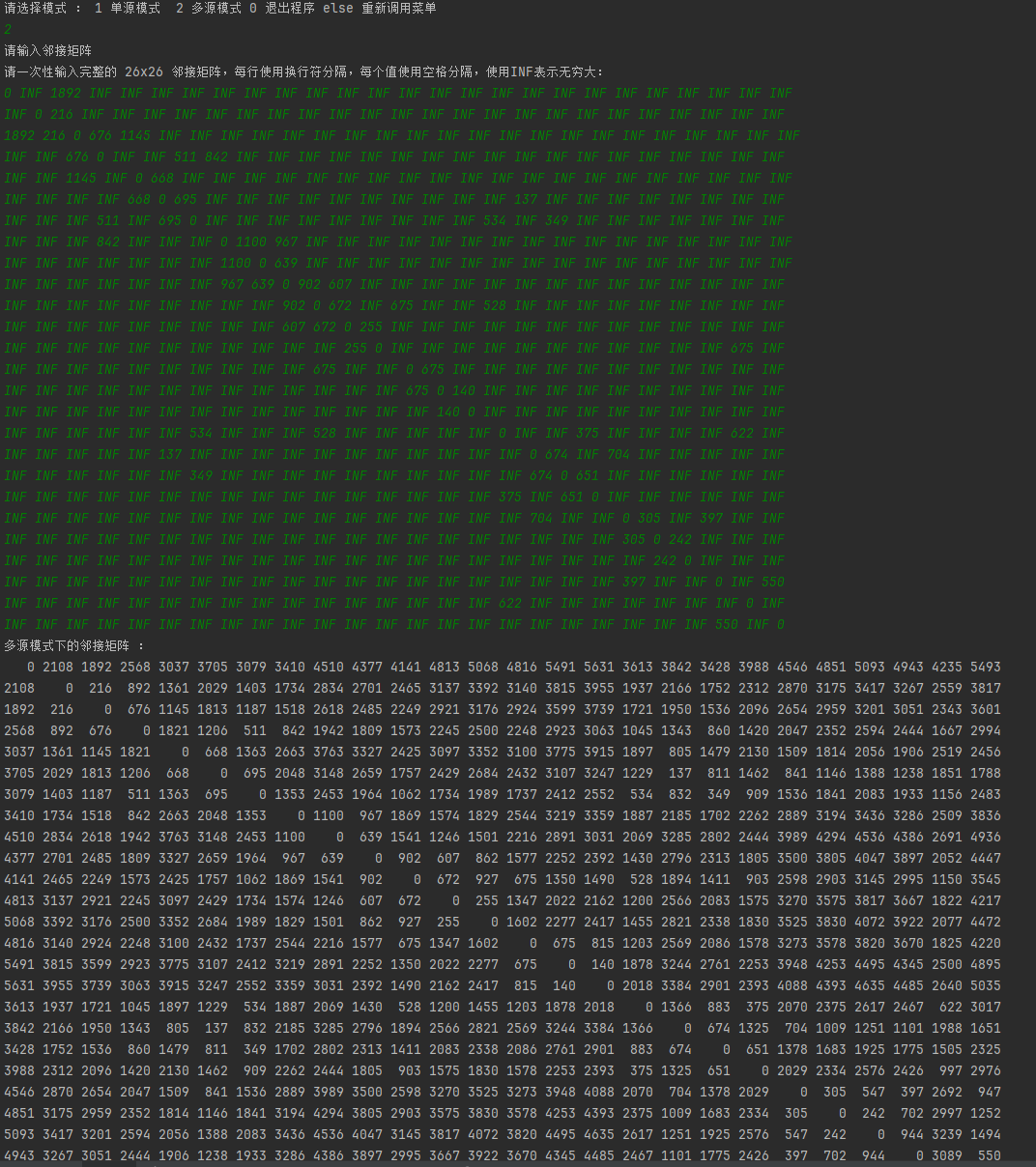
0 INF 1892 INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF  
INF 0 216 INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF  
1892 216 0 676 1145 INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF  
INF INF 676 0 INF INF 511 842 INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF  
INF INF 1145 INF 0 668 INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF  
INF INF INF INF 668 0 695 INF INF INF INF INF INF INF INF INF INF 137 INF INF INF INF INF INF INF INF  
INF INF INF 511 INF 695 0 INF INF INF INF INF INF INF INF INF 534 INF 349 INF INF INF INF INF INF INF  
INF INF INF 842 INF INF INF 0 1100 967 INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF  
INF INF INF INF INF INF INF 1100 0 639 INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF  
INF INF INF INF INF INF INF 967 639 0 902 607 INF INF INF INF INF INF INF INF INF INF INF INF INF INF  
INF INF INF INF INF INF INF INF INF 902 0 672 INF 675 INF INF 528 INF INF INF INF INF INF INF INF INF  
INF INF INF INF INF INF INF INF INF 607 672 0 255 INF INF INF INF INF INF INF INF INF INF INF INF INF  
INF INF INF INF INF INF INF INF INF INF INF 255 0 INF INF INF INF INF INF INF INF INF INF INF 675 INF  
INF INF INF INF INF INF INF INF INF INF 675 INF INF 0 675 INF INF INF INF INF INF INF INF INF INF INF  
INF INF INF INF INF INF INF INF INF INF INF INF INF 675 0 140 INF INF INF INF INF INF INF INF INF INF  
INF INF INF INF INF INF INF INF INF INF INF INF INF INF 140 0 INF INF INF INF INF INF INF INF INF INF  
INF INF INF INF INF INF 534 INF INF INF 528 INF INF INF INF INF 0 INF INF 375 INF INF INF INF 622 INF  
INF INF INF INF INF 137 INF INF INF INF INF INF INF INF INF INF INF 0 674 INF 704 INF INF INF INF INF  
INF INF INF INF INF INF 349 INF INF INF INF INF INF INF INF INF INF 674 0 651 INF INF INF INF INF INF  
INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF 375 INF 651 0 INF INF INF INF INF INF  
INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF 704 INF INF 0 305 INF 397 INF INF  
INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF 305 0 242 INF INF INF  
INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF 242 0 INF INF INF  
INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF 397 INF INF 0 INF 550  
INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF 622 INF INF INF INF INF INF INF 0 INF  
INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF INF 550 INF 0

* 1. 运行结果

实验二单源模式测试结果:



实验二多源模式测试结果:



* 1. 运行结果分析

1、实验二1单源模式下，通过Dijkstra算法计算出单源最

短路径，程序提示输入所需矩阵以及需要进行计算的源点到达的目标点，通过递归调用显示edge列表中所有的前驱节点计算出最短的路径长度以及到达方式。

2、实验二2多源模式下，通过Floyd\_Warshall算法求出可达矩阵并进行显示操作，对每条路径的权值进行计算存入可达矩阵中。

1. 实验心得

通过这次实验，我学会了如何使用Dijkstra算法以

及Floyd\_Warshall算法解决实际问题，在编写程序的过程中遇到了一些问题例如:

1. 利用Dijkstra算法计算单源最短路径时同样不知如何打印输出源点到达叶子节点的路径，之后仿照Prim算法中的实现，但是Prim算法中并不是按照顺序对路径进行打印，于是想到利用递归调用的方法进行实现操作，递归退出的截止条件为判断当前节点是否为传入的源点，若是源点则进行return操作反之继续进行递归操作。
2. 利用Floyd\_Warshall算法时通过三重循环对于每个顶点都将其作为一次中端节点对于对每个循环中的内容进行访问并且更改权值的操作。

总结来说，通过这次实验不仅了解了如何在实际生活中利用这些常见算法，并且学会了如何通过代码完成这些代码的编写与运行，掌握了简单的Debug方法对程序进行调试操作。