中山大学计算机学院 本科生实验报告（2023学年春季学期）

课程名称：Artificial Intelligence 人工智能

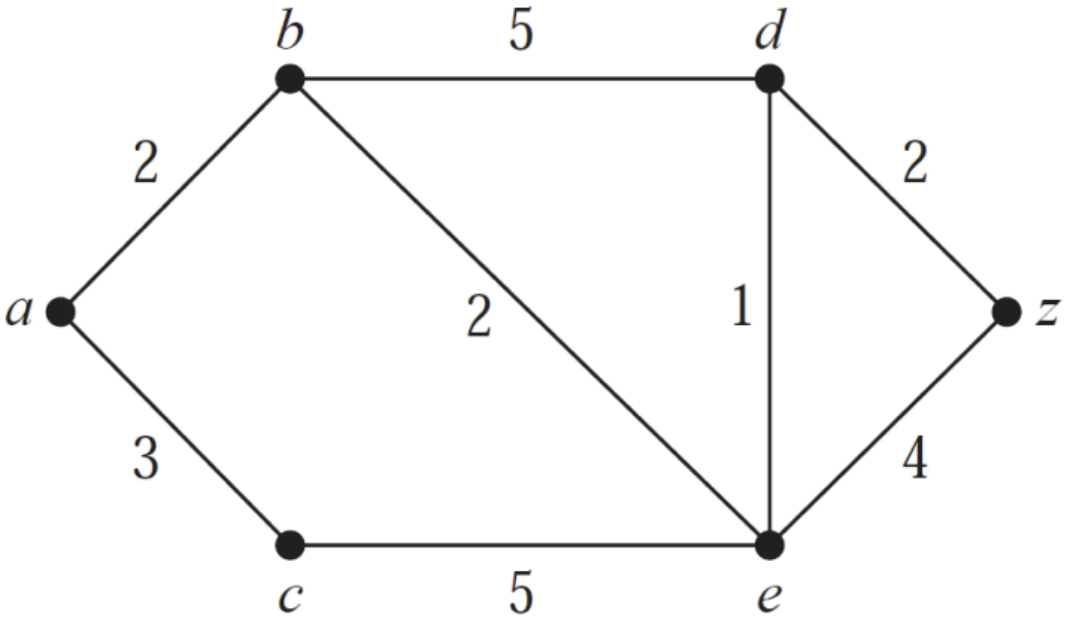
教学班级		专业（方向）	
学号 2233 6173		姓名 罗弘杰	

实验题目

实验1 最短路径算法

- 给定无向图，及图上两个节点，求其最短路径及长度
- 要求：使用Python实现，至少实现Dijkstra算法
- DDL：**3月11号23:59**，即给两周时间完成

- 输入（统一格式，便于之后的验收）
  - 第1行：节点数m 边数n（中间用空格隔开，下同）；
  - 第2行到第n+1行是边的信息，每行是：节点1名称 节点2名称 边权；
  - 第n+2行开始可接受循环输入，每行是：起始节点名称 目标节点名称。
- 输出（格式不限）
  - 最短路径及其长度。



实验内容

1. 算法原理：使用了Dijkstra算法，规定了起点和终点以后，一起点为中心，每次选择最近的点加入内部点集，然后更新最近距离，重复以上操作，直到遇到终点
2. 步骤：

   1,以文件为输入格式，程序读取文件每一行的末尾的换行符来分割每一行，读取每一行中间的空格来分割起点，终点和距离

   2.选择起始点；建立起始点到各个顶点的距离数组；

   3，从数组中找到起始点最近的下一个点，将这个点加入到中间点集；

   4，计算中间点集到集合外各个点的最短路径，更新这个数组，找到最近的点加入到集合，循环如此；
3. 伪代码
4. 关键代码展示（带注释）

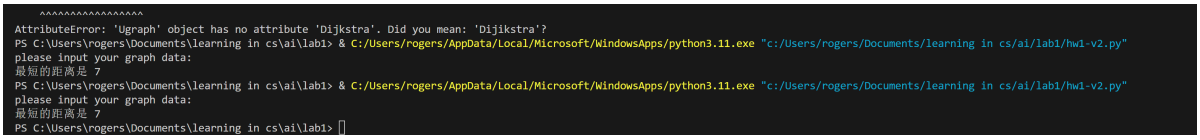
```
# 主函数
print("please input your graph data:")

with open('hw1_data.txt') as file_object:  #读取文件作为字符串
    contents = file_object.read()

content_list = contents.split('\n')  # 拆分为列表，
line0 = content_list[0].split()  # 获取第一行的顶点数和边数
vertex = int(line0[0])
edge = int(line0[1])
my_graph = Ugraph(content_list, vertex, edge)  #输入无向图这个类
my_graph.Dijkstra()   #调用最短路径函数
```

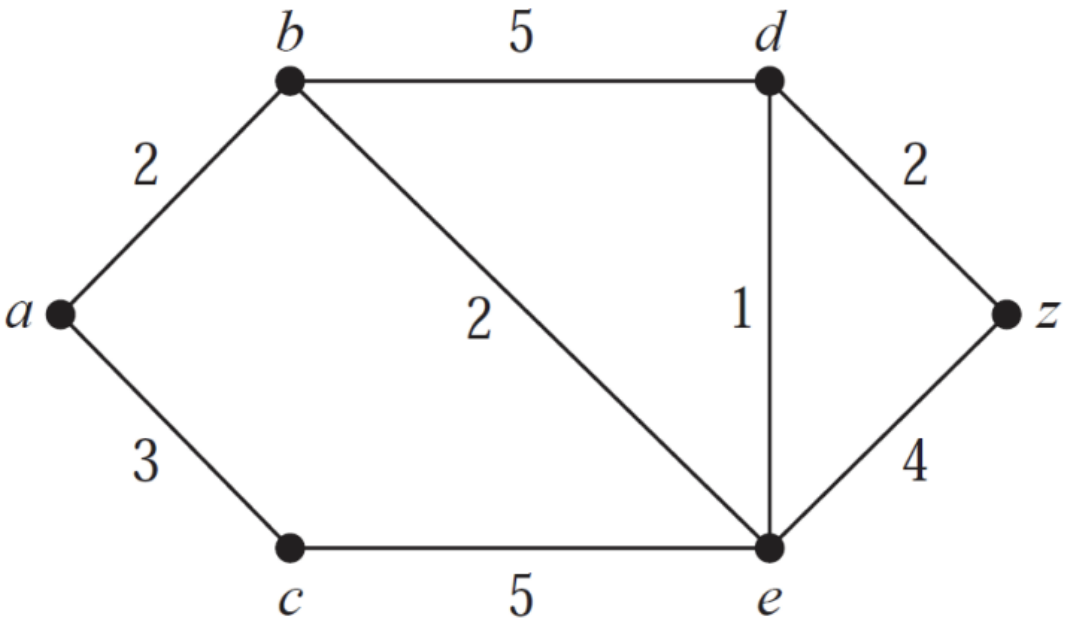
```
# 计算函数
# coding=gbk
class Ugraph():  # 无向图
    def __init__(self, contents, vertex: int, edge: int):
        self.contents = contents
        self.vertex = vertex
        self.edge = edge
        self.graph = {}
        for i in range(edge):  # 获取图像，保存为邻接表
            a, b, dis = contents[i + 1].split(' ', 2)
            dis = int(dis)
            if a not in self.graph.keys():  # 不在表内
                self.graph[a] = {}
                self.graph[a][b] = dis
            else:
                self.graph[a][b] = dis
            if b not in self.graph.keys():  # 不在表内
                self.graph[b] = {}
                self.graph[b][a] = dis
            else:
                self.graph[b][a] = dis
```

```
    def Dijkstra(self):
        src, dst = self.contents[edge + 1].split(' ', 1)
        In_Part = {}   #内围顶点
        Out_Part = {}  #外圈顶点
        In_Part[src] = 0  #先放入起点到内围
        while True:
            if not In_Part:   #没有可以加入的新顶点了，退出
                print("搜索失败了，不存在这样的路径")
                break

            distance, min_node = min(zip(In_Part.values(), In_Part.keys())) #通过距离判断最近的顶点，获取该距离和顶点编号
            In_Part.pop(min_node)     #弹出改顶点
            Out_Part[min_node] = distance  #加入距离信息

            if min_node == dst:                #已经找到
                print("最短的距离是",distance)
                break

            for node in self.graph[min_node].keys():   #遍历新加入顶点的邻接顶点，更新该顶点作为中间顶点带来的变化
                if node not in Out_Part.keys():       #讨论外围顶点
                    if node in In_Part.keys():
                        if self.graph[min_node][node] + distance<In_Part[node]:  #距离变小的要更新距离
                            In_Part[node] = self.graph[min_node][node] + distance

                    else:
                        In_Part[node] = distance + self.graph[min_node][node]  #距离不存在的要加入距离
```

实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）





如图a,z的最短距离是a2b21d2z,总共确实是7，正确输出