中山大学计算机学院

人工智能

本科生实验报告

（2022学年春季学期）

课程名称：Artificial Intelligence

教学班级	计科1班	专业（方向）	计算机科学与技术
学号	21307035	姓名	邓栩瀛

一、实验题目

罗马尼亚旅行问题（最短路径搜索）

实验要求：实现一个搜索罗马尼亚城市间最短路径的导航程序

- 城市地图信息从文件中读取，出发城市和到达城市由用户在查询时输入
- 对于城市名称，用户可以输入全称，也可以只输入首字母，且均不区分大小写
- 向用户输出最短路径时，要输出途经的城市，以及路径总路程
- 输出内容在直接反馈给用户的同时，还需追加写入一个文本文件中，作为记录日志
- 为提升代码灵活性，应在代码中合理引入函数和类（各定义至少一个）
- 定义的一些函数和类，存储在独立的模块文件中

二、实验内容

1.算法原理

Dijsktra算法：从起始点开始，采用贪心算法的策略，每次遍历到始点距离最近且未访问过的顶点的邻接节点，直到扩展到终点为止

1. 设置两个顶点集合 `previous` 和 `unvisited_nodes` 用来存放前置节点（已找到最短路径的结点）和未访问结点（还未找到最短路径的结点）
2. 初始， `previous` 包含源点
3. 从 `unvisited_nodes` 中选择到源点长度最短的点 `current` 加入到 `previous` 中
4. `previous` 中每加入一个新的顶点 `current` ，都要修改源点 `start` 到 `unvisited_nodes` 中剩余顶点的最短路径值， `unvisited_nodes` 中各顶点的新的当前最短路径长度值为原来的当前最短路径长度值和从源点经过 `current` 到达该顶点的带权路径长度中的较小者
5. 重复步骤3，直到 `unvisited_nodes` 中的顶点全部进入到 `previous`

2.伪代码

```
dijkstra(G,dist[],source)
   initialization;
   for i=1 to G.vexnum do
       current=使dist[current]最小的还未被访问的结点
       visited[current]=1
       for 从current出发能到达的所有顶点v do
           if (!visited[v] && source→current→v的距离dist[v]更短)
               更新dist[v]
```

3.关键代码展示（带注释）

最短路径算法

```
  # dijkstra_shortest_way位于类Graph内
  def dijkstra_shortest_way(self, start, end):
        #start:起点  end:终点
        # method: Dijkstra,计算start→所有点的最短距离存储在distance[]中，返回 (path,distance[end])
        # 未访问结点
        unvisited_nodes = self.nodes.copy()
        # 距离初始化,distance[start]=0,其余未INFINITY
        distance = {
            node: (0 if node == start else INFINITY) for node in self.nodes
        }

        # 初始化节点,后面的结点为前面节点的前置节点
        previous = {node: None for node in self.nodes}
        # 循环访问未被访问的节点
        while unvisited_nodes:
          # 当前节点为还未被访问过的节点中路径最短的点
            current = min(
                unvisited_nodes, key=lambda node: distance[node]
            )
            # current节点已访问
            unvisited_nodes.remove(current)
            # 当前节点距离无穷大，退出循环
            if distance[current] == INFINITY:
                break
            # neighbour:当前节点的邻节点（adjlist为邻接表）
            for neighbour, dist in self.adjlist[current]:
```

```
                new_path = distance[current] +
int(dist)
                if new_path < distance[neighbour]:
                    # 更新路径
                    distance[neighbour] = new_path
                    previous[neighbour] = current
        # 用迭代器遍历每个点的前置节点来找到最短路径经过的节
点，并用deque存储
        path = deque()
        current = end
        while previous[current] is not None:
            path.appendleft(current)
            current = previous[current]
        path.appendleft(start)
        return path, distance[end]
```

Graph类及相关函数

```
class Graph:
    def __init__(self, filename):
        # 初始化边的长度
        graph_edge = []
        count = 0
        # 按行读取
        with open(filename) as file:
            for line in file:
                if count == 0:
                    vexnum, arcnum, *_ =
line.strip().split(" ")
                    count = 1
                    continue
                # 循环终止条件
                if line == "\n":
                    break
```

```
                # 无向图
                edge_from, edge_to, cost, *_ = line.strip().split(" ")
                graph_edge.append((edge_from, edge_to, cost))
                edge_to, edge_from, cost, *_ = line.strip().split(" ")
                graph_edge.append((edge_from, edge_to, cost))
        self.nodes = set()
        for edge in graph_edge:
            self.nodes.update([edge[0], edge[1]])
            # 邻接表
        self.adjlist = {node: set() for node in self.nodes}
        for edge in graph_edge:
            self.adjlist[edge[0]].add((edge[1], edge[2]))

    def dijkstra_shortest_way(self, start, end):
```

```python
# 部分
def dijkstra(filename, start, end):
    graph = Graph(filename)
    return_path, return_distance = graph.dijkstra_shortest_way(start, end)

def main():
    dijkstra(
        filename="Romania.txt",
        start=input("Please enter the departure city\n"),
        end=input("Please enter the destination city\n"),
    )
```

三、实验结果及分析

1.实验结果展示示例

- Arad → Bucharest

```
Please enter the departure city
Arad
Please enter the destination city
Bucharest
起点->终点: Arad->Bucharest
最短路径: deque(['Arad', 'Sibiu', 'RimnicuVilcea', 'Pitesti', 'Bucharest'])
路径长度: 418
```

- Fagaras → Dobreta

```
Please enter the departure city
FAGARAS
Please enter the destination city
DObreTa
起点->终点: Fagaras->Dobreta
最短路径: deque(['Fagaras', 'Sibiu', 'RimnicuVilcea', 'Craiova', 'Dobreta'])
路径长度: 445
```

- Mehadia → Sibiu

```
Please enter the departure city
M
Please enter the destination city
s
起点->终点: Mehadia->Sibiu
最短路径: deque(['Mehadia', 'Dobreta', 'Craiova', 'RimnicuVilcea', 'Sibiu'])
路径长度: 421
```

记录日志中的内容

```
Please enter the departure city
Arad
Please enter the destination city
Bucharest
起点->终点: Arad->Bucharest
最短路径: deque(['Arad', 'Sibiu', 'RimnicuVilcea', 'Pitesti', 'Bucharest'])
路径长度: 418

Please enter the departure city
FAGARAS
Please enter the destination city
DObreTa
起点->终点: Fagaras->Dobreta
最短路径: deque(['Fagaras', 'Sibiu', 'RimnicuVilcea', 'Craiova', 'Dobreta'])
路径长度: 445

Please enter the departure city
M
Please enter the destination city
s
起点->终点: Mehadia->Sibiu
最短路径: deque(['Mehadia', 'Dobreta', 'Craiova', 'RimnicuVilcea', 'Sibiu'])
路径长度: 421
```

2.评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

程序运行时间

```
Running Time:  0.101224s
```

四、思考题

1． 如果用列表作为字典的键，会发生什么现象？用元组呢？

```
In [2]: la=[1,2]
In [3]: d={}
In [4]: d[la]=3
```

程序报错：

```
    exec(code_obj, self.user_global_ns, self.user_ns)
  File "<ipython-input-4-79b9de59ffa1>", line 1, in <module>
    d[la]=3
TypeError: unhashable type: 'list'
```

对于Python中的所有内置可变类型（列表、字典、集合），都不能将其用作字典的键值

> To be used as a dictionary key, an object must support the hash function (e.g. through **hash), equality comparison (e.g. through eq** or **cmp**), and must satisfy the correctness condition above.
>
> ——来着python官网
>
> Translate:一个数据类型要用作字典键，对象必须同时支持取哈希值（例如通过 `__hash__` 方法）和判断是否相等（例如通过 `__eq__` 或 `__cmp__` 方法）这两个操作

列表类型没有实现 `__hash__` 方法，因此无法取哈希值，进而无法作为字典键

对于元组而言

```
In [16]: d = {('a','b'):(3,7)}
In [17]: id(d)
Out[17]: 140609459079296
```

可以作为字典的键，但元组中只能包括像数字和字符串这样的不可变参数，才可以作为字典中有效的键

2. 在本课件第2章和第4章提到的数据类型中，哪些是可变数据类型，哪些是不可变数据类型？试结合代码分析。

- 可变/不可变数据类型：变量值发生改变时，变量的内存地址不变/改变。
- 提示：① 你可能会用到id()函数。② Python的赋值运算符（=）是引用传递。

不可变类型	数字（int、float）	字符串（str）	布尔（bool）	元组（tuple）
可变类型	列表（list）	字典（dict）	集合（set）	用户自定义的类

```
In [6]: a=123
In [7]: id(a)
Out[7]: 140609445025968
In [8]: a=4567
In [9]: id(a)
Out[9]: 140609179929520
```

对于不可变类型：给一个整数对象 `a` 赋予或修改数值时， `a` 的 `id` 会发生变化，因为数字是不可变类型，当我们修改变量 `a` 的值时，将原先内存中的 `123` 销毁，然后开辟出新的内存空间写入 `4567` 并赋予对象 `a` 。

```
In [10]: arr=[1,2]
In [11]: id(arr)
Out[11]: 140609180121792
In [12]: arr[1]=3
In [13]: id(arr)
Out[13]: 140609180121792
In [14]: arr.append(6)
In [15]: id(arr)
Out[15]: 140609180121792
```

对于可变类型：如果对其修改，会在原来的内存空间上修改。因为列表 `arr` 对象是一个可变对象。存储对象 `arr` 的内存空间中，并不直接存储 `arr` 的内容，而是包含一个指针指向真实存放 `arr` 内容的地址。