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

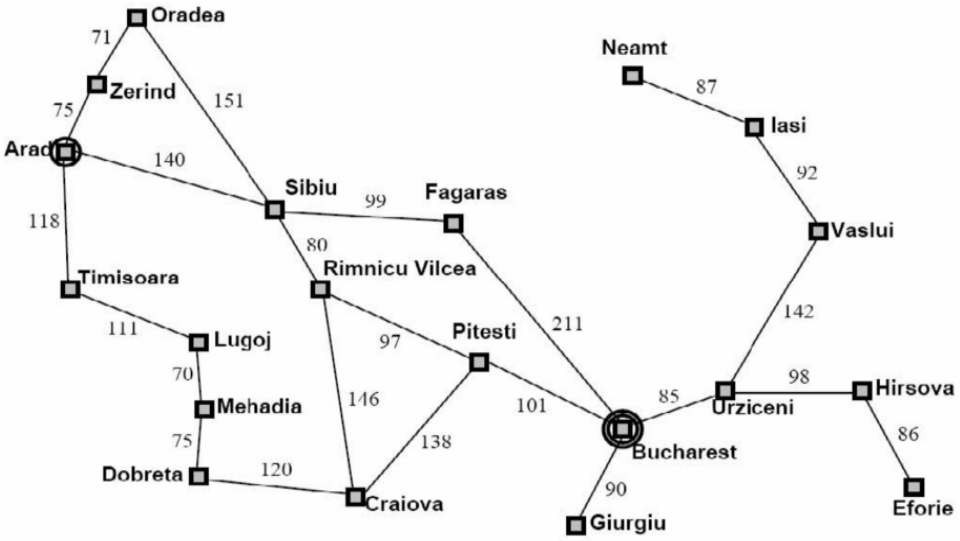
教学班级	计科 1 班	专业（方向）	计算机科学与技术
学号	21307077	姓名	凌国明

一、实验题目

罗马尼亚旅行问题

你是一名正在罗马尼亚旅行的游客，现在你急需从城市1赶到城市2搭乘航班，因此，你需要找到从**城市1到城市2的最短路径。**

罗马尼亚的地图信息文件：Romania.txt
文件第1行：城市数 道路数
文件第2行到第24行，每行是：城市1名称 城市2名称 路程



- 实现一个搜索罗马尼亚城市间最短路径的导航程序，要求：
- 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，你应在代码中合理引入**函数和类**（各定义至少一个）
- 此外，将你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

Dijkstra算法

Dijkstra算法本质上是一种贪心算法。从起始节点开始，层层向外扩展，每次确定一个距离源点最短且未访问过的节点，再通过这个节点的相邻边更新相邻节点的目前最短路径，最终确定单源点到其他所有节点的最短路径。其中贪心的思想体现在最短路径的更新上。

2、伪代码

```
for i in range(n): # 循环n次
    min=float("inf")
    min_index=-1
  for k in range(n):
    for j in range(n):
        # 找出未访问过的，距离最短的点
        if min>distance[i][j] and not visited[i][j]:
            min=distance[i][j]
            min_index=j
        visited[min_index]=1
    # 更新各点的距离
    for l in range(n):
        if distance[i][l]>dis[i][min_index]+edge[min_index][l]:
            distance[i][l]=dis[i][min_index]+edge[min_index][l]
            path[i][l]=min_index
```

3、关键代码展示（带注释）

```
# class Graph 中的 init 函数
with open(txt_location, 'r', encoding='ascii') as f:
    for item in f.readlines():
        item = item.strip('\n')  # 去除文本中的换行符
        item = item.split(sep=' ')
        # 第一行
        if len(item) < 3:
            self.node_num = int(item[0])
            self.edge_num = int(item[1])
            continue
        # 新遇到的城市添加到list中记录下来
        if item[0].lower() not in self.list:
            self.list[item[0].lower()] = number
            self.list[number] = item[0].lower()
            number = number + 1
        if item[1].lower() not in self.list:
            self.list[item[1].lower()] = number
            self.list[number] = item[1].lower()
            number = number + 1
        # 更新邻接表
        string = item[0] + '->' + item[1]
        string = string.lower()
        self.adj[string] = int(item[2])
        string = item[1] + '->' + item[0]
        string = string.lower()
        self.adj[string] = int(item[2])
```

```
# Dijkstra 算法过程
# 起始节点
start = self.list[self.city_name(a)]
# 目标节点
target = self.list[self.city_name(b)]
# 是否确定最短路径
determined = []
# 目前最短路径的距离
distance = []
# 最短路径的前驱节点
pre = []
# 初始化数组
for i in range(0, self.node_num):
    determined.append(False)
    distance.append(0x7FFFFFFF)
    pre.append(-1)
# 起始节点确定,距离为0
determined[start] = True
distance[start] = 0
pre[start] = start
# 更新初始距离
for i in range(0, self.node_num):
    if self.list[start]+'->'+self.list[i] in self.adj:
        distance[i] = self.adj[self.list[start]+'->'+self.list[i]]
        pre[i] = start
# dijkstra过程
for e in range(1, self.node_num):
    # 找到未确定的距离最短的点
    min_dist = 0x7FFFFFFF
    min_position = -1
    for i in range(0, self.node_num):
        # 还没有确定最短路径，且目前最短路径小于min
        if not determined[i] and distance[i] < min_dist:
            min_dist = distance[i]
            min_position = i
    if min_position == -1:
        break
    # 将距离最短的点的最短路径确定下来
    print(self.list[min_position] + "的最短路径已确定")
    determined[min_position] = True
    # 更新其他节点的目前最短路径
    for i in range(0, self.node_num):
        # 如果存在这样的一条道路
        if self.list[min_position]+'->'+self.list[i] in self.adj:
            # 如果start到min_position的距离 + min_position到i的距离 < start到i的距离
            if np.longlong(self.adj[self.list[min_position]+'->'+self.list[i]] + min_dist) < np.
                print(self.list[i] + "的最短路径长度从" + str(distance[i]) + "更新为" + str(self.a
                # 更新距离
                distance[i] = self.adj[self.list[min_position]+'->'+self.list[i]] + min_dist
```

```
            # 更新前驱节点
            pre[i] = min_position
```

4、创新点&优化（如果有）

设点的个数为 n,边的个数为 m。最初我运用二维列表构建邻接矩阵，这样的空间复杂度为 $O(n^2)$，若 n 过大，则占用空间大，于是改用Python中的字典来构建邻接表，空间复杂度为 $O(m)$。

三、实验结果与分析

1、实验结果展示示例（可图可表可文字，尽量可视化）

Arad 到 Bucharest

```
最短路径为: Bucharest<--Pitesti<--Rimnicuvilcea<--Sibiu<--Arad
最短路径长度为: 418
```

Fagaras 到 Dobreta

```
最短路径为: Dobreta<--Craiova<--Rimnicuvilcea<--Sibiu<--Fagaras
最短路径长度为: 445
```

Mehadia 到 Sibiu

```
最短路径为: Sibiu<--Rimnicuvilcea<--Craiova<--Dobreta<--Mehadia
最短路径长度为: 421
```

res文件记录本次运行的结果

```
start: arad  end: bucharest
Shortest path: Pitesti<--Rimnicuvilcea<--Sibiu<--Arad
Length of the shorest path:418

start: fagaras  end: dobreta
Shortest path: Craiova<--Rimnicuvilcea<--Sibiu<--Fagaras
Length of the shorest path:445

start: mehadia  end: sibiu
Shortest path: Rimnicuvilcea<--Craiova<--Dobreta<--Mehadia
Length of the shorest path:421

start: urziceni  end: arad
Shortest path: Sibiu<--Rimnicuvilcea<--Pitesti<--Bucharest<--Urziceni
Length of the shorest path:503
```

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）

设点的个数为 n,边的个数为 m。

- 若使用邻接矩阵来构建图，则每一次 $Dijkstra$ 算法的时间复杂度为$O(n^2)$, n 次 $Dijkstra$ 的时间复杂度则为 $O(n^3)$.

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?

当字典的键为列表时，系统会报错，因为列表是可以改变的，是"unhashable"的，而元组是不可变的，是"hashable"的。

代码如下

```
list_ = [1, 2, 3]
tuple_ = (1, 2, 3)
dict_1 = {tuple_: 1}
dict_2 = {list_: 1}
```

运行结果如下

```
Traceback (most recent call last):
  File "C:\Users\linggm\Desktop\AI\Lab1\Code\sfsf.py", line 4, in <module>
    dict_2 = {list_: 1}
TypeError: unhashable type: 'list'

进程已结束,退出代码1
```

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

Python 的六种标准数据类型：数字、字符串、列表、元组、字典、集合。

可变数据：数字，字符串，元组

数字代码如下

```
a = 1
print(id(a))
add_1 = id(a)
a = 2
add_2 = id(a)
print(id(a))
if add_1 == add_2:
    print("不可变")
else:
    print("可变")
```

运行结果如下

```
2397727189296
2397727189328
可变

进程已结束,退出代码0
```

字符串代码如下

```
str1 = "first"
print(id(str1))
add_1 = id(str1)
str1 = "second"
add_2 = id(str1)
print(id(str1))
if add_1 == add_2:
    print("不可变")
else:
    print("可变")
```

运行结果如下

```
2418769467632
2418766783152
可变

进程已结束,退出代码0
```

元组代码如下

```python
a = (1, 2, 3)
print(id(a))
add_1 = id(a)
a = (5, 4, 6)
add_2 = id(a)
print(id(a))
if add_1 == add_2:
    print("不可变")
else:
    print("可变")
```

运行结果如下

```
1550125296064
1550125296960
可变

进程已结束,退出代码0
```

不可变数据：列表、字典、集合

集合代码如下

```python
a = {1, 2, 3}
print(id(a))
add_1 = id(a)
a.add(31)
add_2 = id(a)
print(id(a))
if add_1 == add_2:
    print("不可变")
else:
    print("可变")
```

运行结果如下

```
1986066461632
1986066461632
不可变

进程已结束,退出代码0
```

列表代码如下

```python
a = [1, 2, 3]
print(id(a))
add_1 = id(a)
a.append(4)
add_2 = id(a)
print(id(a))
if add_1 == add_2:
    print("不可变")
else:
    print("可变")
```

运行结果如下

```
2155343489024
2155343489024
不可变

进程已结束,退出代码0
```

字典代码如下

```
a = {'a': 1, 'b': 2}
print(id(a))
add_1 = id(a)
a['c'] = 3
add_2 = id(a)
print(id(a))
if add_1 == add_2:
    print("不可变")
else:
    print("可变")
```

运行结果如下

```
2293728423168
2293728423168
不可变

进程已结束,退出代码0
```

五、参考资料

https://zhuanlan.zhihu.com/p/129373740

https://blog.csdn.net/qq_42420425/article/details/107687230