中山大学计算机学院
人工智能
本科生实验报告

（2023 学年春季学期）

课程名称：Artificial Intelligence

教学班级	**刘永梅**	专业（方向）	**信息与计算科学**
学号	**22336313**	姓名	**郑鸿鑫**

实验题目

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。

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

一. Dijkstra 算法原理：

初始化：

将起始节点加入"open set"，表示该节点的最短路径长度尚未确定。将"closed set"置为空。

迭代更新：

从"open set"中选择距离起始节点最近的节点，将其从"open set"移至"closed set"，表示该节点的最短路径长度已确定。然后更新该节点的邻接节点的最短路径长度和路径，如果经过当前节点到达邻接节点的路径长度小于当前记录的最短路径长度，则更新最短路径长度和路径。将刚刚加入"closed set"的节点的所有邻接节点加入"open set"，表示这些节点的最短路径长度还未确定。

重复以上步骤，直到"open set"为空，或者目标节点被移到"closed set"中（如果只需要求解到达目标节点的最短路径）。最终得到起始节点到所有其他节点的最短路径长度和路径。

这种说法下，"open set"表示尚未确定最短路径的节点集合，"closed set"表示已确定最短路径的节点集合。通过不断将"open set"中距离起始节点最近的节点移到"closed set"中，并更新其邻接节点的最短路径长度和路径，最终得到起始节点到其他节点的最短路径。

值得注意的是，在此算法中由于要记录路径，需要用另一个列表 parent 来记录每个节点的最短路径上的前驱（详细见下文）。

二．Floyd 算法原理：

初始化：

创建一个二维数组 Graph，用于存储任意两个节点之间的最短路径长度。初始时，Graph[i][j]表示节点 i 到节点 j 的直接路径

长度，如果节点 i 到节点 j 有直接连接，则为连接的权重值，否则为无穷大。另外，创建一个二维数组 parents，用于记录节点之间的中间节点，初始时 parents[i][j]为节点 i 到节点 j 的直接连接的终点。

三重循环：

对于每一个节点 k，遍历所有节点对 i 和 j，检查是否存在通过节点 k 的路径比直接路径更短。如果存在这样的路径，则更新 Graph[i][j]为经过节点 k 的路径长度，并更新 parents[i][j]为节点 k。

重复计算：

通过不断地更新 Graph 和 parents，直到所有节点对之间的最短路径都被计算出来。

最终结果：

经过上述计算，Graph[i][j]中存储了节点 i 到节点 j 的最短路径长度，parents[i][j]中存储了节点 i 到节点 j 最短路径上节点 j 的前一个节点。

最短路径重构：

根据 parents 数组，可以回溯得到任意两个节点之间的最短路径。

伪代码

一. Dijkstra 算法伪码：

文件读取输入建立无向图（采用邻接矩阵即二维列表存储）

建立字典 dict 以对应节点的名称和编号

While True：

接收循环输入（起点与终点），识别到"Stop"后跳出循环

建立 open 和 close 列表存储元组（a,b）（a 为节点编号，b 为到起点的距离）

建立 parent 列表存储元组（a,b）（a 为节点编号，b 为 a 在最短路径上的前驱）

将（起点，0）放入 open 中

While True：

If open 为空则 break

寻找 open 中离起点最近的点(min,min_dist)并从 open 中删除

If min == 终点: break

for i in 所有与 min 邻接的节点:

If min_dist + Graph[cur][i] <= dist[i]:

更新 open 和 parent

将(min,min_dist)加入到 close 中

建立 path 列表存储路径

While temp != 起点：

for I in parent：

寻找 temp，将其加入 path，更新 temp 为它的前驱

将 path 反转（path.reserve()）

打印最短路径和长度

二．Floyd 算法伪码：

文件读取输入建立无向图（采用邻接矩阵即二维列表存储）

建立字典 dict 以对应节点的名称和编号

初始化 parents 二维列表为：parents[i][j] = i

for k in range(m):

for i in range(m):

for j in range(m):

if i 到 k 的距离+k 到 j 的距离小于 i 直接到 j 的距离：

则更新最短距离，同时更新 parents 列表

建立 path 列表存储路径

temp 初始化为 end

While temp != 起点：

for l in parents[sta][temp]：

寻找 temp，将其加入 path，更新 temp 为它的前驱

将 path 反转（path.reserve()）

打印最短路径和长度

关键代码展示（带注释）

文件作为输入与构建无向图和字典:(对于两种算法都是通用的代码)

```
in_file = open("D:\Code\Python\Lab1\input.txt", "r")
mn = in_file.readline().split()
m = int(mn[0])
n = int(mn[1])
Max = float('Inf')
import numpy as np
# 创建一个m行m列的矩阵，所有元素的值都是无穷大
Graph = np.full((m, m), np.inf)
i = 0
while i < m:
    Graph[i][i] = 0
    i += 1
i = 0
j = 0
dict = {}#字典用于存放各节点及其编号
while i < n:
```

```
    s = in_file.readline()
    s = s.split()
    a = s[0]
    b = s[1]
    #ab 是两节点名称，a1b1 是两节点对应的编号
    weight = int(s[2])
    if a in dict:
        a1 = dict[a]
    else:
        dict[a] = j
        a1 = j
        j += 1
    if b in dict:
        b1 = dict[b]
    else:
        dict[b] = j
        b1 = j
        j += 1
    Graph[a1][b1] = weight
    Graph[b1][a1] = weight
    i += 1  #建立邻接矩阵
import copy
while True:
    s = in_file.readline()
    if s == "Stop":
        break
    s = s.split()
    start = s[0]
    sta = dict[start]#start 的编号
    end = s[1]
    en = dict[end]#end 的编号
    dist = copy.copy(Graph[sta])
```

Dijkstra 算法的主要代码段：

```
    open = []#open 中存储编号和距离起点的 int 二元组
    close = []#close 中存储 open 中已经确定最短路径的 int 二元组
    parent = []#存储每个节点最短路径上的前驱，起点的 parent 为 None
    open.append((sta,0))
    while True:
        if len(open) == 0:
            break
        min_dist = Max
        min = -1 # min 为 open 中距离起点最近的点的编号
        for i in range(len(open)):
            if open[i][1] < min_dist:
                min_dist = open[i][1]
                min = i
        cur, cur_dist = open.pop(min)
        #cur 为该轮能确定最短路径的点
        if cur == en:
            break
        for j in range(len(Graph[cur])):#所有和 min 相邻的节点更新距离
            if (Graph[cur][j] < Max) and (j != cur):
                new_dist = cur_dist + Graph[cur][j]
```

```
                if new_dist <= dist[j]:
                    dist[j] = new_dist
                    open.append((j, new_dist))
                    flag = False
                    for i in parent:
                        if i[0] == j:
                            flag = True
                            parent.remove(i)
                            parent.append((j,cur))
                            break
                    if flag == False:
                        if j == sta:
                            parent.append((j,None))
                        else:
                            parent.append((j,cur))
        close.append((cur, cur_dist))
    path = []
    path.append(en)
    temp = en
    while temp != sta:
        for i in parent:
            if i[0] == temp:
                temp = i[1]#将 temp 更新为它的前驱
                path.append(i[1])
                break
    path.reverse()
```

Floyd 算法主要代码段：

```
parents = np.full((m,m),np.inf)#新建 parents 数组
for i in range(m):
    for j in range(m):
        parents[i][j] = i #初始化 parents 数组
for k in range(m):
        for i in range(m):
            for j in range(m):
                if Graph[i][k] + Graph[k][j] < Graph[i][j]:
                    Graph[i][j] = Graph[i][k] + Graph[k][j]
                    parents[i][j] = parents[k][j]
```

最短路径及其长度的输出：（对于 Floyd 算法，只需将最后一行 dist[en]改为 Graph[sta][en]即可通用）

```
    print(start,"到",end,"的最短路径为: ")
    for i in path:
        for key, value in dict.items():
            if value == i:
                if value == en:
                    print(key)
                else:
                    print(key, end="->")
                break
    print("长度为",int(dist[en]))
```

实验结果及分析

实验结果展示示例

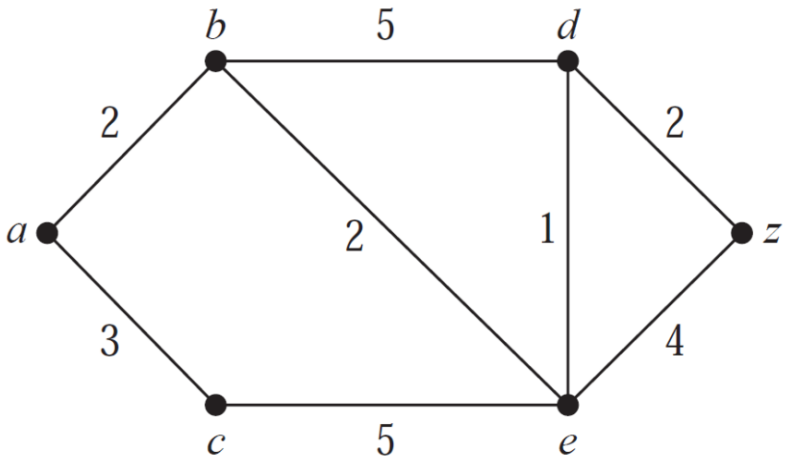
Input.txt 文件内容如下：

（可通过修改 input 文件修改无向图和所求起点和终点,识别到 Stop 时程序结束）

Code > Python > Lab1 > input.txt

```
6 8
a b 2
a c 3
b d 5
b e 2
c e 5
d e 1
d z 2
e z 4
a z
z a
a d
c z
e d
Stop
```

经过两种算法得到的结果相同，如图所示：（附示例无向图）：




```
PS D:\Code> & D:/python/python.exe d:/Code/Python/Lab1/Project1_1.py
d:\Code\Python\Lab1\Project1_1.py:2: SyntaxWarning: invalid escape sequence '\C'
  in_file = open("D:\Code\Python\Lab1\input.txt", "r")
a 到 z 的最短路径为:
a->b->e->d->z
长度为 7
z 到 a 的最短路径为:
z->d->e->b->a
长度为 7
a 到 d 的最短路径为:
a->b->e->d
长度为 5
c 到 z 的最短路径为:
c->e->d->z
长度为 8
e 到 d 的最短路径为:
e->d
长度为 1
```

参考资料

Dijkstra 算法参考自：

Dijkstra 路径规划算法原理详解及 Python 代码实现_python dijkstra-CSDN 博客

Floyd 算法参考自：

floyd 算法(多源最短路径) python 实现_floyd python-CSDN 博客

numpy 库参考自：

python numpy 创建二维矩阵 numpy 创建二维数组的方法_mob64ca1404baa2 的技术博客_51CTO 博客