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	21 级计科 2 班	专业（方向）	计算机科学与技术
学号	21307185	姓名	张礼贤

一、 实验题目

Python 求最短路径

二、 实验内容

1. 算法原理

Dijkstra 算法：

求单源最短路径要用到 dijkstra 算法，即基于一种贪心的思想。可用邻接矩阵实现，初始化距离数组 dis 为无穷大，标记数组 used 为 false（因为是无向图，所以要进行访问标记），parent 数组初始化未-1，表示没有父节点（记录节点的父节点，方便回溯路径）

进入循环后，每次选取未访问过的最小 dis 节点，记为 x，再设置指针 y 遍历每个节点，利用 dis[y]=min(dis[y],dis[x]+graph[x][y])更新 dis 数组的距离，如果比原来的值要小则将 parent[y]=x，以获取父结点的值，并反复迭代。

之后通过目标节点利用 parent 数组不断回溯，直到-1 停止，并将遍历到的节点存入 list，之后将 list 结果 reverse，即为路径。再一并将起点和终点的距离和 path 输出，得到结果。

Floyd 算法：

在本题中，求最短路径也可以使用 Floyd 算法。弗洛伊德算法定义了两个二维矩阵：
矩阵 D 记录顶点间的最小路径，例如 D[0][3]= 10，说明顶点 0 到 3 的最短路径为 10；
矩阵 P 记录顶点间最小路径中的中转点，例如 P[0][3]= 1 说明，0 到 3 的最短路径轨迹为：0 -> 1 -> 3。

它通过 3 重循环，k 为中转点，v 为起点，w 为终点，循环比较 D[v][w] 和 D[v][k] + D[k][w] 最小值，如果 D[v][k] + D[k][w] 为更小值，则把 D[v][k] + D[k][w] 覆盖保存在 D[v][w]中

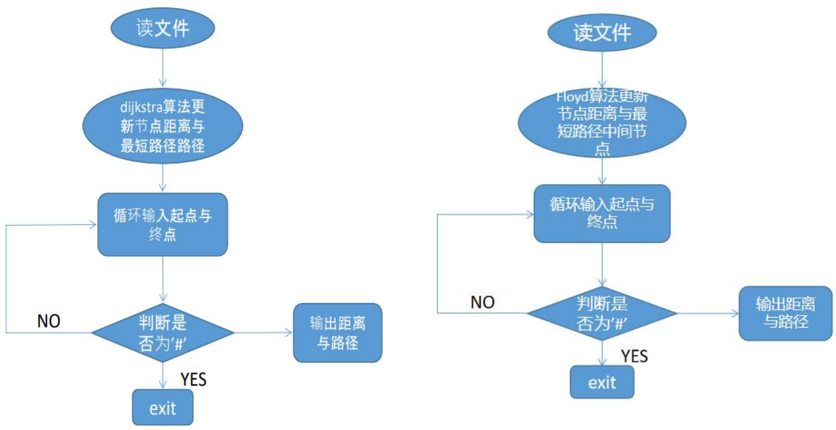
2. 伪代码

```
伪代码
#dijkstra 算法
dis[start]=0
while(i<vertex_num) do
begin
    find min(dis[x])
```

```
    set x => true
    while(j<26) do
    begin
        dis[j]=min(dis[i]+graph[i][j])
        set parent
    end
end
Print distance and parent

#Floyd 算法:
Initialize graph and path
For i in range vertex_num
  For j in range vertex_num
    For k in range vertex_num
      graph[j][k]=min(graph[j][i]+graph[i][k],graph[j][k])
      Set path
```

流程图:



3. 关键代码展示（带注释）

Dijkstra 算法:

```
import numpy
from numpy import Infinity
def print_path(parent,start):
    """打印路径
       parent 表示父节点数组
       start 表示从某个点开始
    """
    store=[]
    if(parent[start]==-1):
        return store
    store.append(chr(start+ord('a')))
    while(parent[start]!=-1):
        #当遇到-1 时终止,表示已经到了起点或不能到达
```

```
        store.append(chr(parent[start]+ord('a')))
        start=parent[start]
    store.reverse() #反转列表，表示从起点到终点的路径
    return store

def create_graph(f):
    """创建无向图"""
    # graph=[[Infinity]*26 for _ in range(26)] #用列表解析式初始化图
    graph=numpy.full((26,26),Infinity) #调用 numpy 创建二维数组

    for line in f.readlines():
        """逐行读取文件，创建无向图"""
        line=line.strip()
        print(line)
        s=line.split( ) #分割空格
        graph[ord(s[0])-ord('a')][ord(s[1])-ord('a')]=int(s[2])
        graph[ord(s[1])-ord('a')][ord(s[0])-ord('a')]=int(s[2])
```

```
    return graph
```

```
def dijkstra(vertex_num,edge_num,start,end):
    """实现最短路径算法"""
    dis=[Infinity]*26   #初始化距离数组
    parent=[-1]*26    #初始化父节点数组，以-1 表示终点
    used=[False]*26     #初始化 used 数组，确定节点是否被访问
```

```
    for i in range(vertex_num):
        graph[i][i]=0 #到自身的距离为零
    dis[ord(start)-ord('a')]=0 #初始化源点的 dis 为 0
    for i in range(26):
        x = -1
        for y in range(26):
            if(not used[y] and (x==-1 or dis[y]<dis[x])):
                x = y   #找到 dis 最小的点
        used[x]=True    #对其进行以访问标记
        for y in range(26):
            if(dis[y]>dis[x]+graph[x][y]):
                dis[y]=dis[x]+graph[x][y]   #更新距离
                parent[y]=x                  #记录父节点
```

```
    distance=dis[ord(end)-ord('a')]
    return [distance,parent]
```

```
f=open('test.txt')
```

```
s=f.readline()  #读取文件的第一行，获取点数和边数
s=s.split()
vertex_num=int(s[0])
edge_num=int(s[1])
graph=create_graph(f)
while(True):
    #循环输入，当遇到'#'时退出循环
    ans =input("Please input the start and end:(input '#' exit)\n")
    if(ans=='#'):break
    ans =ans.split()
    start=ans[0]
    end=ans[1]
    res=dijkstra(vertex_num, edge_num, start, end) #res 接受结果,res[0]表示 distance; res[1]表示路径
    print("The distance is:",res[0])
    p=ord(end)-ord('a')
    path=print_path(res[1],p)
    if(len(path)==0):
        print("The path is not exist!")
    else :
        print("The path is:",path)
```

Floyd 算法：

```
import numpy
from numpy import Infinity
map_to_chr={}# 创建字典将整数映射到字符
map_to_int={}# 创建字典将字符映射到整数
def create_graph(f,vertex_num):
    """创建无向图"""
    graph=[[Infinity]*vertex_num for _ in range(vertex_num)] #用列表解析式初始化图
    #graph=numpy.full((vertex_num,vertex_num),Infinity) #调用 numpy 创建二维数组
    k=0
    for line in f.readlines():
        """逐行读取文件，创建无向图"""
        line=line.strip()
        print(line)
        s=line.split( ) #分割空格
        weight=int(s[2])
        if(s[0] not in map_to_int):
            map_to_int[s[0]]=k
            map_to_chr[k]=s[0]
            k+=1
        if(s[1] not in map_to_int):
            map_to_int[s[1]]=k
            map_to_chr[k]=s[1]
```

```
            k+=1
        graph[map_to_int[s[0]]][map_to_int[s[1]]]=weight
        graph[map_to_int[s[1]]][map_to_int[s[0]]]=weight
    return graph

def print_path(path,start,end):
    print("The path is:",end="")#避免换行
    while(start!=end): #判断是否到达终点
        print(map_to_chr[start],end="->")
        start=path[start][end]
    print(map_to_chr[end])

def Floyd(graph,path):
    for i in range (vertex_num):
        for j in range (vertex_num):
            for k in range (vertex_num):
                if (graph[j][k]>graph[j][i]+graph[i][k]):
                    graph[j][k]=graph[j][i]+graph[i][k] #更新中间节点的距离
                    path[j][k]=path[j][i]              #更新路径到中间节点

f=open('test.txt')
s=f.readline()  #读取文件的第一行，获取点数和边数
s=s.split()
vertex_num=int(s[0])
edge_num=int(s[1])
graph=create_graph(f,vertex_num)
path = [[Infinity]*vertex_num for _ in range(vertex_num)]
for i in range (vertex_num):
    for j in range (vertex_num):
        path[i][j]=j    #初始化 path 二维数组
Floyd(graph,path)       #调用 Floyd 算法求最短路径
while True:             #循环输入，当输入'#'时终止
    s =input("Please input the start and end:(input '#' exit)\n")
    if(s=='#'):break
    s=s.split()
    start = map_to_int[s[0]]
    end = map_to_int[s[1]]
    print("The distance is:",graph[start][end]) #打印最短距离
    print_path(path,start,end)
```

4. 创新点&优化（如果有）

1、在 dijkstra 算法与 Floyd 算法中通过动态规划的方式进行 dis 数组的更新与求解，减少了代码比较的次数和代码量，使其变得精简化。

2、创建双射字典，不必遍历 26 个字母，减少了建立矩阵时的空间复杂度。

3、在 dijkstra 算法中设置 parent 数组初始化为-1，并通过回溯（事实上是不断迭代）求得路径，即类似于数的存储，再从叶子节点遍历到根节点，再通过反序即可得到。

三、 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）

Input：

```
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
```

Output：

```
Please input the start and end:(input '#' exit) #显示输出的提示
a z                                             #输入起点与终点
The distance is: 7.0                            #输出距离
The path is: ['a', 'b', 'e', 'd', 'z']          #输出起点到终点的路径
Please input the start and end:(input '#' exit) #循环输入
a c
The distance is: 3.0
The path is: ['a', 'c']
Please input the start and end:(input '#' exit)
c z
The distance is: 8.0
The path is: ['c', 'e', 'd', 'z']
Please input the start and end:(input '#' exit) #输入’#’结束/
#
```

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）

经过验证，实验结果与预期一致，dijkstra 算法的时间复杂度为 O(m*n^2)，n 为点数，m 为输入的样例个数，空间复杂度为 O(n^2)。

而 Floyd 算法时间复杂度为 O(n^3)，n 为点数，空间复杂度为 O(n^2)

因此，当输入样例 m>n 时，最好选用 Floyd 算法以减少时间复杂度