**云南大学软件学院**

**实 验 报 告**

课程： 计算机网络实践 任课教师： 李海

姓名： 陈俊宏 学号： 20211060245 专业： 人工智能 成绩：



1．实验目的：

通过编程模拟实现LSA.

2．实验环境：

软件开发平台，可以使用任何编程语言。

3．实验要求

（1）求网络中任何两个结点之间的最短路径（网络中至少有4个节点）。

（2）得到任何一个节点上的转发表。

4.实验内容、拓扑结构

*7*

*1*

*8*

*1*

*2*

*2*

通过链路状态算法计算A点到其它各点的cost，最终输出A的路由表。（编程语言不限！）

算法提示：

***Initialization:***

2 N' = {u} /\*u is source node\*/

3 for all nodes j /\* j is dest node\*/

4 if j adjacent to u

5 then D(j) = c(u,j)

6 else D(j) = ∞

7

8 ***Loop***

9 find i not in N' such that D(i) is a minimum

10 add i to N'

11 update D(j) for all j adjacent to i and not in N' :

12 D(j) = min( D(j), D(i) + c(i,j) )

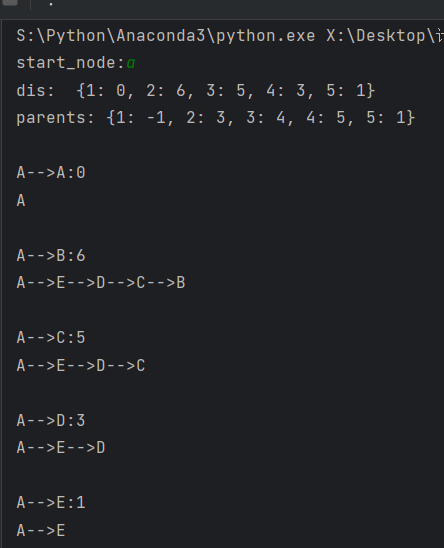
13 /\* new cost to j is either old cost to j or known

14 shortest path cost to i plus cost from i to j \*/

15 ***until all nodes in N'***

def Dijkstra(G, start):  
 # 输入是从 0 开始，所以起始点减 1  
 start = start - 1  
 inf = float('inf')  
 node\_num = len(G)  
 # visited 代表哪些顶点加入过  
 visited = [0] \* node\_num  
 # 初始顶点到其余顶点的距离  
 dis = {node: G[start][node] for node in range(node\_num)}  
 # parents 代表最终求出最短路径后，每个顶点的上一个顶点是谁，初始化为 -1，代表无上一个顶点  
 parents = {node: -1 for node in range(node\_num)}  
 # 起始点加入进 visited 数组  
 visited[start] = 1  
 # 最开始的上一个顶点为初始顶点  
 last\_point = start  
  
 for i in range(node\_num - 1):  
 # 求出 dis 中未加入 visited 数组的最短距离和顶点  
 min\_dis = inf  
 for j in range(node\_num):  
 if visited[j] == 0 and dis[j] < min\_dis:  
 min\_dis = dis[j]  
 # 把该顶点做为下次遍历的上一个顶点  
 last\_point = j  
 # 最短顶点假加入 visited 数组  
 visited[last\_point] = 1  
 # 对首次循环做特殊处理，不然在首次循环时会没法求出该点的上一个顶点  
 if i == 0:  
 parents[last\_point] = start + 1  
 for k in range(node\_num):  
 if G[last\_point][k] < inf and dis[k] > dis[last\_point] + G[last\_point][k]:  
 # 如果有更短的路径，更新 dis 和 记录 parents  
 dis[k] = dis[last\_point] + G[last\_point][k]  
 parents[k] = last\_point + 1  
  
 # 因为从 0 开始，最后把顶点都加 1  
 return {key + 1: values for key, values in dis.items()}, {key + 1: values for key, values in parents.items()}  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 inf = float('inf')  
 name = ['A', 'B', 'C', 'D', 'E']  
 G = [  
 [0, 7, inf, inf, 1],  
 [7, 0, 1, inf, 8],  
 [inf, 1, 0, 2, inf],  
 [inf, inf, 2, 0, 2],  
 [1, 8, inf, 2, 0]  
 ]  
 start\_node = str(input("start\_node:")).upper()  
 dis, parents = Dijkstra(G, (name.index(start\_node) + 1))  
 print("dis: ", dis)  
 print("parents:", parents, '\n')  
 for i in range(len(G)):  
 shortest\_path = ''  
 print(start\_node + '-->' + name[i] + ':' + str(dis[i + 1]))  
 if name[i] == start\_node:  
 print(start\_node + '\n')  
 continue  
 shortest\_path = f"-->{name[i]}" + shortest\_path  
 i = i + 1  
 while True:  
 shortest\_path = f"-->{name[parents[i] - 1]}" + shortest\_path  
 i = parents[i]  
 if parents[i] == -1:  
 shortest\_path = shortest\_path.lstrip('-->')  
 break  
 print(shortest\_path + '\n')

实验截图：



4．实验分析，回答下列问题

（1）给出LSA算法的主要思想。

答：

LSA算法的主要思想是将文本数据转化为向量空间模型，通过奇异值分解进行降维，得到一个新的向量空间模型，其中每个向量代表一个文档或一个单词，向量之间的距离反映了它们之间的语义相似度。

（2）通过图表算出任何两个节点之间的最短路径，并给出每个节点上的转发表。

转发表示例如下： A的转发表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | B | E | C | D |
| 最短路径 | （A,B) | (A,D,E) |  |  |
| 成本值 | 2 | 1 |  |  |

A的转发表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | B | C | D | E |
| 最短路径 | (A, E, D, C, B) | (A, E, D, C) | (A, E, D) | (A, E) |
| 成本值 | 6 | 5 | 3 | 1 |

B的转发表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | C | D | E |
| 最短路径 | (B, C, D, E, A) | (B, C) | (B, C, D) | (B, C, D, E) |
| 成本值 | 6 | 1 | 3 | 5 |

C的转发表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B | D | E |
| 最短路径 | (C, D, E, A) | (C, B) | (C, D) | (C, D, E) |
| 成本值 | 5 | 1 | 2 | 4 |

D的转发表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B | C | E |
| 最短路径 | (D, A) | (D, C, B) | (D, C) | (D, E) |
| 成本值 | 3 | 3 | 2 | 2 |

E的转发表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B | C | D |
| 最短路径 | (E, A) | (E, D, C, B) | (E, D, C) | (E, D) |
| 成本值 | 1 | 5 | 4 | 2 |