****

**J I A N G S U U N I V E R S I T Y**

《网络科学基础》

第四次平时作业



学院名称： 计算机学院

专业班级： 物联网2303班

学生姓名： 邱佳亮

学生学号： 3230611072

教师姓名： 熊书明

2024年11月

# 题一代码

1. import numpy as np *#导包*
2. import matplotlib.pyplot as plt
3. import heapq
4. *#%%*
5. def dijkstra\_path(G, start, end):
6. *# 初始化距离和前驱节点字典*
7. distances = {node: float('inf') for node in G}
8. previous\_nodes = {node: None for node in G}
9. distances[start] = 0
11. *# 优先队列，存储 (距离, 节点) 对*
12. priority\_queue = [(0, start)]
14. while priority\_queue:
15. *# 取出当前距离最小的节点*
16. current\_distance, current\_node = heapq.heappop(priority\_queue)
18. *# 如果到达终点，提前结束*
19. if current\_node == end:
20. break
22. *# 遍历当前节点的所有邻居*
23. for neighbor, weight in G[current\_node].items():
24. distance = current\_distance + weight
26. *# 如果找到更短的路径*
27. if distance < distances[neighbor]:
28. distances[neighbor] = distance
29. previous\_nodes[neighbor] = current\_node
30. heapq.heappush(priority\_queue, (distance, neighbor))
32. *# 构建最短路径*
33. path = []
34. current\_node = end
35. while previous\_nodes[current\_node] is not None:
36. path.insert(0, current\_node)
37. current\_node = previous\_nodes[current\_node]
38. if path:
39. path.insert(0, start)
41. return distances[end], path
42. *# 示例图*
43. G = {
44. 1: {2: 7, 3: 9, 6: 14},
45. 2: {1: 7, 3: 10, 4: 15},
46. 3: {1: 9, 2: 10, 4: 11, 6: 2},
47. 4: {2: 15, 3: 11, 5: 6},
48. 5: {4: 6, 6: 9},
49. 6: {1: 14, 3: 2, 5: 9}
50. }
51. *# 调用函数*
52. distance, path = dijkstra\_path(G, 1, 5)
53. print(f"最短距离: {distance}")
54. print(f"最短路径: {path}")
55. *#%%*
56. def bellman\_ford(G, start):
57. *# 初始化距离和前驱节点字典*
58. distances = {node: float('inf') for node in G}
59. previous\_nodes = {node: None for node in G}
60. distances[start] = 0
62. *# 获取所有边*
63. edges = []
64. for node in G:
65. for neighbor, weight in G[node]:
66. edges.append((node, neighbor, weight))
68. *# 进行 V-1 次松弛操作*
69. for \_ in range(len(G) - 1):
70. for u, v, weight in edges:
71. if distances[u] + weight < distances[v]:
72. distances[v] = distances[u] + weight
73. previous\_nodes[v] = u
75. *# 检查负权环*
76. for u, v, weight in edges:
77. if distances[u] + weight < distances[v]:
78. raise ValueError("图中存在负权环")
80. return distances, previous\_nodes
81. def reconstruct\_path(previous\_nodes, start, end):
82. path = []
83. current\_node = end
84. while current\_node is not None:
85. path.insert(0, current\_node)
86. current\_node = previous\_nodes[current\_node]
87. return path
88. *# 示例图*
89. G = {
90. 1: [(2, 7), (3, 9), (6, 14)],
91. 2: [(1, 7), (3, 10), (4, 15)],
92. 3: [(1, 9), (2, 10), (4, 11), (6, 2)],
93. 4: [(2, 15), (3, 11), (5, 6)],
94. 5: [(4, 6), (6, 9)],
95. 6: [(1, 14), (3, 2), (5, 9)]
96. }
97. *# 调用函数*
98. distances, previous\_nodes = bellman\_ford(G, 1)
99. print(f"最短距离: {distances}")
100. print(f"前驱节点: {previous\_nodes}")
101. *# 重建从起点到终点的最短路径*
102. end = 5
103. path = reconstruct\_path(previous\_nodes, 1, end)
104. print(f"从节点 1 到节点 {end} 的最短路径: {path}")*#Bellman-Ford算法*

## 运行结果

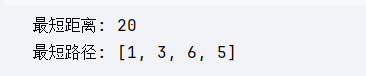


图 1 Dijkstra算法运行结果

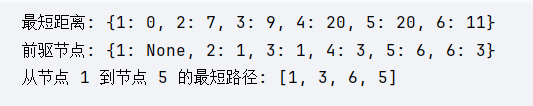


图 2 Bellman-Ford算法运行结果

## 算法比较

Dijkstra算法和Bellman-Ford算法都是用于求解图中单源最短路径问题的算法，它们都采用贪心策略来逼近最短路径。Dijkstra算法适用于边权全为非负的图，时间复杂度为O((V+E)logV)，实现相对复杂，不能处理负权边，也无法检测负权环。而Bellman-Ford算法可以处理含有负权边的图，能检测负权环，时间复杂度为O(VE)，实现简单，适用于可能包含负权边的图。所以如果图中含有负权边或需要检测负权环，Bellman-Ford算法效率更高；而当图的边权全为非负且图较为密集时，Dijkstra算法的效率更高。

# 题二代码

1. clc,clear;
2. a=zeros(6)
3. a(1,[2,3,5])=1;
4. a(2,[3,4])=1;
5. a(3,6)=1;
6. a(4,6)=1;
7. a=a+a*'*
8. [D,L,dist]=myAPL(a)
9. function[D,L,dist] =myAPL(a)
10. A=graph(a)
11. dist=distances(A);
12. D=max(max(dist));
13. Ldist=tril(dist);
14. he=sum(nonzeros(Ldist));
15. n=length(a);
16. L=he/nchoosek(n,2);
17. end

## 运行结果

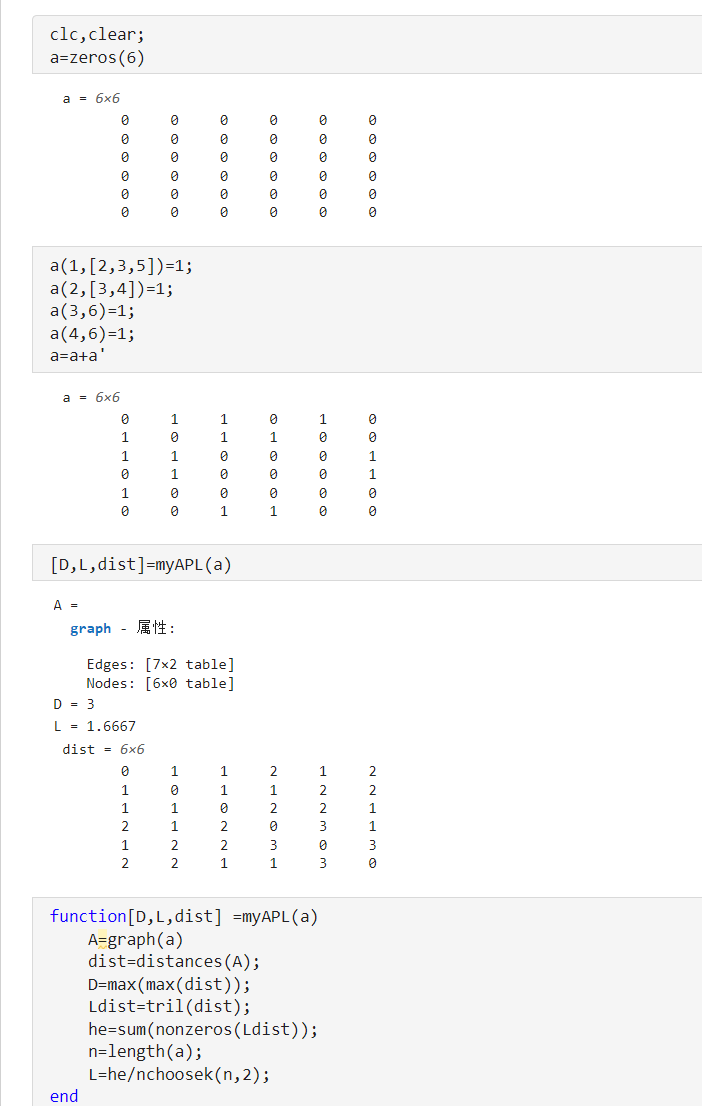


图 3 运行结果

## 查找帮助文档

### tril()

函数功能：返回矩阵 A 的下三角部分

实例：B提取了A的下三角部分，C仅提取了A的主对角线下的部分：



图 4 函数示例

### sparse()

函数功能：将矩阵转换为稀疏矩阵以节省内存；生成m×n的全零稀疏矩阵。

实例：通过sparse将A矩阵转换为稀疏矩阵，节约了内存；也可以建立一个10000\*5000的稀疏矩阵：

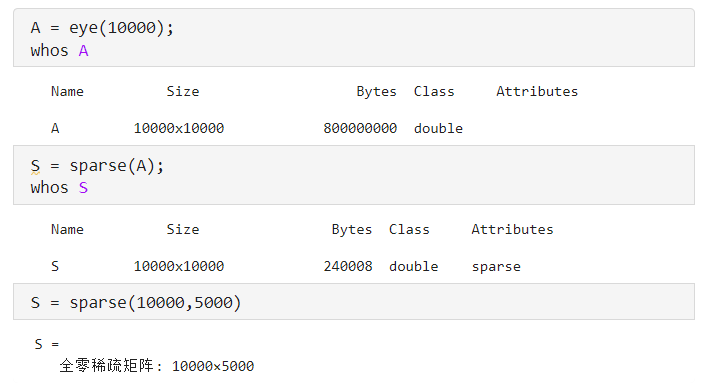


图 5 函数示例

### graphallshortestpaths()

函数功能：在图中找到所有最短路径。

实例：建立了一个稀疏矩阵G，并找到了其最短路径；graphallshortestpaths已在R2022b版本被删除，用distances代替。

1. G = sparse([6 1 2 2 3 4 4 5 5 6 1],[2 6 3 5 4 1 6 3 4 3 5],[41 99 51 32 15 45 38 32 36 29 21])
2. graphallshortestpaths(G)

### max()

函数功能：返回数组的最大元素。

实例：使用max函数返回了数组A的最大元素52：

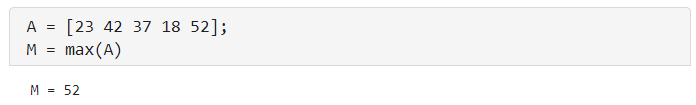


图 6 函数示例

### sum()

函数功能：返回沿数组一维度的元素之和。

实例：使用sum函数返回向量或矩阵A的元素之和，可以指定沿列方向或行方向计算：

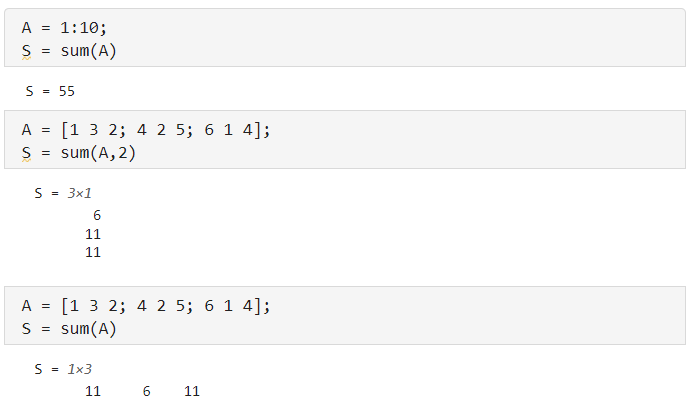


图 7 函数示例

### nonzeros()

函数功能：返回矩阵中非零元素的列向量，v 中的元素按列排序。

实例：建立了10\*10的稀疏矩阵A，用nonzeros函数返回了权不为0的边的权重，并按由小到大排序：

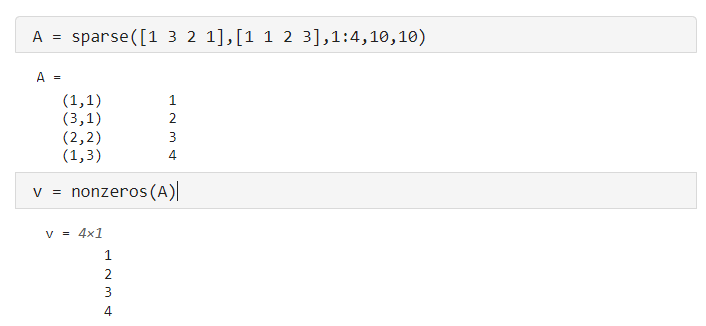


图 8 函数示例

### nchoosek()

函数功能：返回二项式系数。

实例：计算了二项式系数：

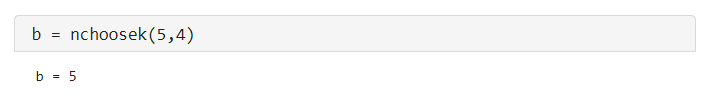


图 9 函数示例