2025 人工智能编程语言课程报告

肖伟文

(中山大学　智能工程学院)

摘　要

本报告聚焦于最短路算法的实现与应用，选取 Python 语言分别对 Dijkstra 算法和 Floyd 算法进行编程实践，以给定有向图为研究对象，深入探究其在求解最短路径问题中的表现。通过将图转化为二维数组并存储，在 Dijkstra 算法中，成功计算出以 v0 和 v1 为起点到其余顶点的最短路径，利用 numpy 和 pandas 库完成对最短路数据的统计分析，提取平均最短距离等关键指标，并借助 matplotlib 库实现路径长度分布的可视化展示；在附加挑战部分，实现了 Floyd 算法，能够准确计算出有向图中所有顶点对之间的最短路径，进一步拓展了对最短路问题的求解能力。

关键词：Dijkstra 算法、Floyd 算法、matplotlib 可视化

Abstract

In this report, Dijkstra's algorithm and Floyd's algorithm are practiced. In Dijkstra's algorithm, the shortest paths from v0 and v1 to the remaining vertices are calculated, and the numpy and pandas libraries are used to complete the statistical analysis of the data, extract the key indexes, and display them visually with the help of the matplotlib library; in the part of the additional challenge, Floyd's algorithm is implemented, which is capable of accurately calculating the shortest paths between all pairs of vertices in the directed graph. paths between all pairs of vertices in a directed graph.

1 算法理解与数据预处理

1.1 伪代码

以下是伪代码。

Algorithm 1 Dijkstra 算法伪代码

Require: 图的邻接矩阵 G，起点 s

Ensure: 各顶点到起点的最短距离 D

初始化距离数组 D，其中 $D(s)=0$，其他 $D(v)=\infty$
初始化集合 $S=\{s\}$，$U=V-\{s\}$（V 为图中所有顶点）
for 顶点 $v\in U$ do
    $D(v)=G[s][v]$ （若 s 到 v 无边，则 $G[s][v]=\infty$）
end for
while $U\neq\emptyset$ do
    从 U 中找出距离最小的顶点 k，即 $k=\arg\min_{v\in U} D(v)$
    $U=U-\{k\}$，$S=S\cup\{k\}$
    for 顶点 $v\in U$ do
        if $D(k)+G[k][v]<D(v)$ then
            更新距离：$D(v)=D(k)+G[k][v]$
        end if
    end for
end while
return D

1.2 流程图

以下是流程图1。

2 Python 编程

2.1 代码

以下是展示代码：

Listing 1: show.py

```
1 import pandas as pd
```

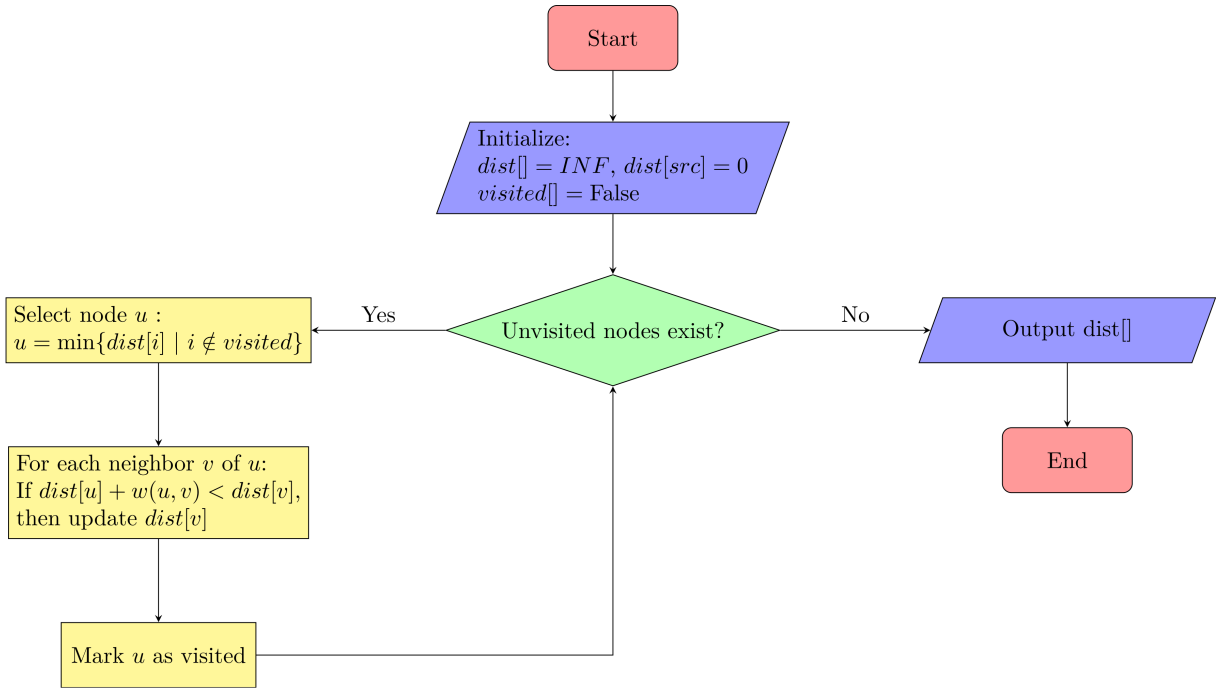

Figure 1: Dijkstra 算法流程图（Tikz 绘制）

```
import numpy as np
from matplotlib.animation import FuncAnimation
import networkx as nx
import os

# Load the edge data
filename = "edge.csv"  # The CSV file containing the edge data for the graph
edge_input = pd.read_csv(filename)  # Read the CSV file into a pandas DataFrame
edge_input = edge_input.iloc[:, 1:]  # Remove the first column (if it's an index or irrelevant)
print("Edge Input:")  # Print the loaded edge data for verification
print(edge_input)

# Initialize the graph
n = len(edge_input)  # Number of nodes in the graph (assumes square adjacency matrix)
INF = 1e9  # Representation of infinity (a very large number)
edge = np.zeros((n, n))  # Initialize an n x n adjacency matrix with zeros

# Fill the adjacency matrix
for i in range(n):  # Iterate over rows
    for j in range(n):  # Iterate over columns
        value = edge_input.iloc[i, j]  # Get the value from the DataFrame
```

```
23          if value.isnumeric():  # Check if the value is numeric (valid edge weight)
24              edge[i][j] = int(value)  # Convert to integer and store in the adjacency matrix
25          else:
26              edge[i][j] = INF  # If not numeric, treat it as no edge (infinity)
27  print("Adjacency Matrix:")  # Print the adjacency matrix for verification
28  print(edge)
29
30  # Dijkstra's Algorithm
31  def dijkstra(graph, src):
32      """
33      Implementation of Dijkstra's algorithm to find the shortest paths
34      from a source node to all other nodes in a weighted graph.
35
36      Parameters:
37      - graph: 2D list or numpy array representing the adjacency matrix of the graph
38      - src: The source node (0-indexed)
39
40      Returns:
41      - dist: List of shortest distances from the source node to each node
42      """
43      n = len(graph)  # Number of nodes in the graph
44      dist = [INF] * n  # Initialize distances to all nodes as infinity
45      dist[src] = 0  # Distance to the source node itself is 0
46      visited = [False] * n  # Track whether each node has been visited
47
48      for _ in range(n):  # Iterate n times (once for each node)
49          # Find the unvisited node with the smallest distance
50          min_dist = INF  # Initialize the minimum distance as infinity
51          u = -1  # Initialize the node index
52          for i in range(n):  # Iterate over all nodes
53              if not visited[i] and dist[i] < min_dist:  # Check if node is unvisited and has a smaller
                    distance
54                  min_dist = dist[i]  # Update the minimum distance
55                  u = i  # Update the node index
56
57          if u == -1:  # If no unvisited node is reachable, break the loop
58              break
59
60          # Mark the node as visited
```

```
61          visited[u] = True
62
63          # Update distances to neighboring nodes
64          for v in range(n):  # Iterate over all nodes
65              if not visited[v] and graph[u][v] != INF:  # Check if the neighbor is unvisited and has an
                    edge
66                  # Update the distance to the neighbor if a shorter path is found
67                  dist[v] = min(dist[v], dist[u] + graph[u][v])
68
69      # Replace distances that are still infinity with -1 (indicating unreachable nodes)
70      dist = [x if x < INF else -1 for x in dist]
71      return dist
72
73  # Task 1: Run Dijkstra's algorithm from a source node (e.g., node 0)
74
75  source_node = 0  # Define the source node (0-indexed)
76  shortest_distances = dijkstra(edge, source_node)  # Compute shortest distances
77  print(f"Shortest distances from node {source_node}:")  # Print the results
78  print(shortest_distances)
79
80  # Task 2: Run Dijkstra's algorithm from a different source node (e.g., node 1)
81
82  source_node = 1  # Define the source node (0-indexed)
83  shortest_distances = dijkstra(edge, source_node)  # Compute shortest distances
84  print(f"Shortest distances from node {source_node}:")  # Print the results
85  print(shortest_distances)
86
87  # Task 3 : Analyze the shortest path statistics for all nodes in the graph
88
89  def analyze_shortest_paths(graph): {}
90
91  # Perform analysis and print results
92  shortest_path_stats = analyze_shortest_paths(edge)
93  print("Shortest Path Statistics:")
94  print(shortest_path_stats)
95
96  # Task 4: Visualize the shortest path between two nodes
97
98  import matplotlib.pyplot as plt
```

```
99
100 def visualize_shortest_path_between_two_nodes(graph, start, end): {}
101
102 # Example usage: Visualize the shortest path between node 0 and node 2
103 def save_shortest_path_images(graph, output_folder="image"): {}
104
105 # Generate and save images for all reachable node pairs
106 save_shortest_path_images(edge)
107
108 def visualize_dijkstra_execution(graph, src): {}
109
110 # Visualize Dijkstra's algorithm execution from node 0
111 visualize_dijkstra_execution(edge, src=0)
112 plt.show()  # Ensure the plot is displayed properly
```

此处省略可视化部分的完整实现，完整代码请见 Python 编程文件夹下 dijkstra.py

2.2 运行效果

以下是运行截图2。

2.3 可视化

图15中的每张子图标记了两点间的最短路边集。

动画可视化在 visual 文件夹下。

2.4 附加题

见附加题文件夹。

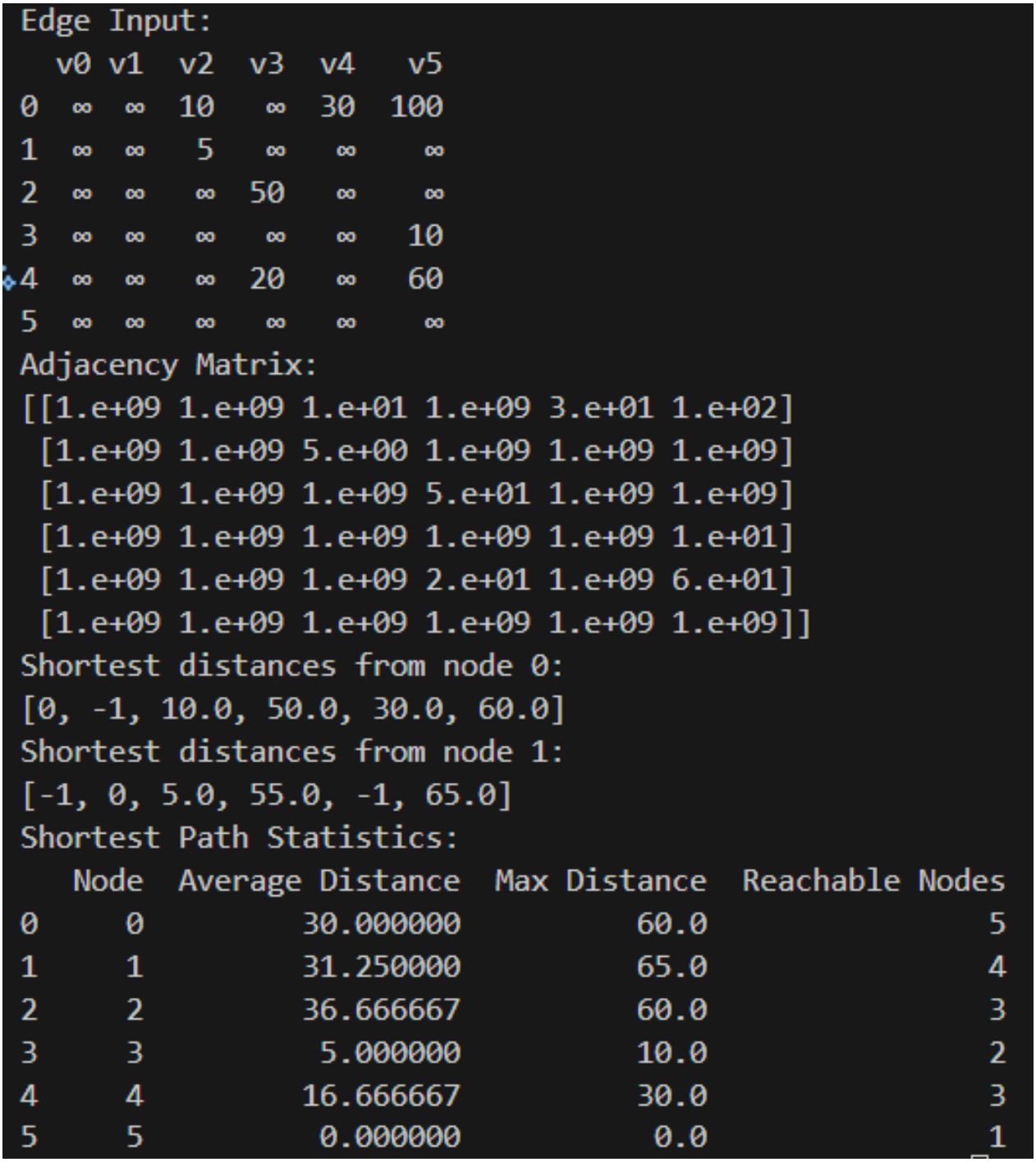

Figure 2: 运行截图

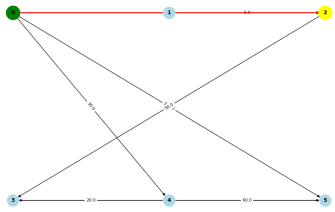

Figure 3: 0 to 2

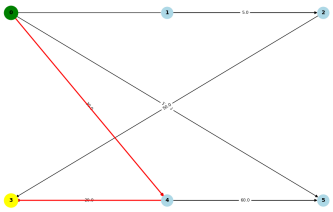

Figure 4: 0 to 3

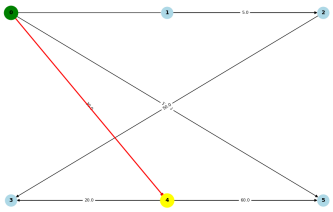

Figure 5: 0 to 4

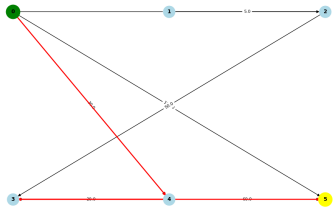

Figure 6: 0 to 5

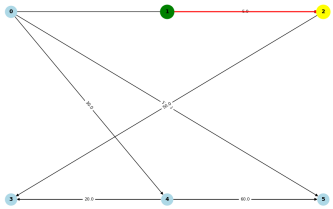

Figure 7: 1 to 2

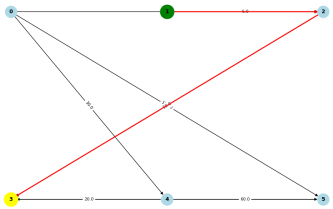

Figure 8: 1 to 3

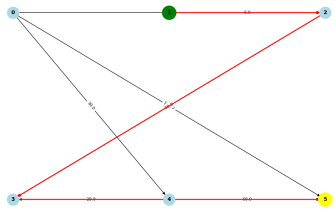

Figure 9: 1 to 5

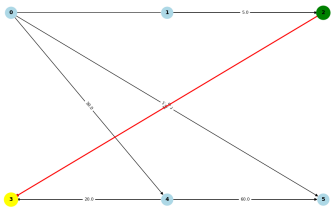

Figure 10: 2 to 3

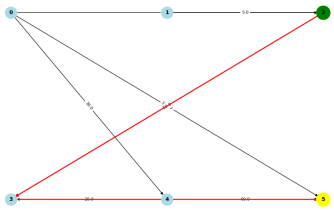

Figure 11: 2 to 5

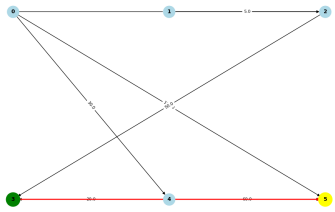

Figure 12: 3 to 5

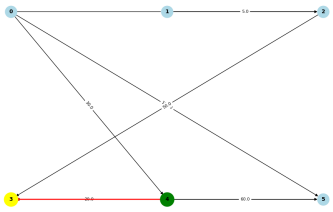

Figure 13: 4 to 3

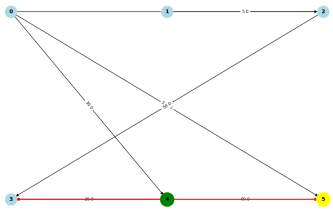

Figure 14: 4 to 5

Figure 15: 两两间最短路