E1_22336216

人工智能实验

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	DCS315	专业（方向）	计算机科学与技术（系统结构）
学号	22336216	姓名	陶宇卓

1 实验题目

实验一：最短路径算法

2 实验内容

一、算法原理

Dijkstra是一种求解非负权图上单源最短路径的算法。具体实现方法是将点集分为已处理的的点集和未处理的点集，从给定的点开始初始化（假定初始点为s），初始化dis[s]=0,其余点的dis均为 $+\infty$ 。

1. 从与s相连的节点中选取一个离s最近的点b加入已处理的点集，并计算两点之间的边权weight+dis[s]，如果比dis[b]小的话，就更新dis[b]。
2. 对新加入的点进行第一步的操作。

3. 直至所有点都被处理，算法结束。

> 可以使用优先队列priority_queue对该算法进行优化，每次操作后把{dis[s],s}压入优先队列pq。
>
> 也可以用二叉堆，线段树进行优化，但是我不太会（）

二、伪代码

```C++
void Dijkstra(G,d[],s)
    {
        初始化优先队列pq,数组vis;
        pq.push(d[s],s)
        while(还有点在pq里)
        {
            u=使d[u] 最小的还未被访问的顶点的标号;
            vis[u]=true;
            for(从u 出发能达到的所有顶点)
            {
                if(!vis[v] && d[s]+weight+d[u]<d[v])
                    更新d[v];
            }
        }
    }

```

三、关键代码展示

> 实验环境为conda环境，Python版本3.11.5

Dijkstra Python

```python
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
from queue import PriorityQueue
import sys
import time
import threading


# dijkstra函数
def dijkstra(graph, start, end):
    """
    迪杰斯特拉函数，求最短路径
    参数:字典graph,起点start,终点end
    返回:最短路长度dis[end],路径列表path

    """
    print(f"Start node: {start}, end_node: {end}\n")
    dis = {node: float('inf') for node in graph} # 字典(unordered_map)，初始化为{所有点：无穷}
    dis[start] = 0
    pq = PriorityQueue() # 优先队列，小顶堆
    pq.put((0, start)) # 压入一个元组(0,start)
    pre = {node: None for node in graph} # 一个字典提供给最短路径回溯，初始化为{所有点：空对象}

    # 关键部分，在算法原理讲完了
    while not pq.empty():
        cur_dis, cur_node = pq.get()
        for neighbor, weight in graph[cur_node]:
            distance = cur_dis + weight
            if distance < dis[neighbor]:
                dis[neighbor] = distance
                pre[neighbor] = cur_node
                pq.put((distance, neighbor))


    path = []
    cur = end

    # 不能用!=,判断None应该使用is/not
    while cur is not None:
        path.insert(0, cur)
```

```
        cur = pre[cur]

    return dis[end], path

# 读取输入数据
while True:
    input_line = input("请输入行数和列数:\n").split()
    m, n = int(input_line[0]), int(input_line[1])

    graph = {}

    # 读取边的信息
    for i in range(n):
        edge_line = input("请输入边"+str(i+1)+":\n").split()
        node1, node2, weight = edge_line[0], edge_line[1], int(edge_line[2])
        graph.setdefault(node1, []).append((node2, weight))
        graph.setdefault(node2, []).append((node1, weight))

    # 检查节点数是否相等
    if len(graph) != m:
        print("错误：图中的节点数量与指定的节点数量不匹配，重新输入。")
    else:
        break




# 转换为networkx图
G = nx.Graph()
for node, neighbors in graph.items():
    for neighbor, weight in neighbors:
        G.add_edge(node, neighbor, weight=weight)

# 绘制图形
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, font_weight='bold', node_size=700, node_color='skyblue', font_size=10, font_color='black')
edge_labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
plt.show(block=True)


# 处理循环输入
while True:

```

```
    input_line = input("请输入要查询的两个点:\n").split()
    start, end = input_line[0], input_line[1]
    if start == '0' and end == '0':
        break

    start_time = time.time()

    shortest_dis, shortest_path = dijkstra(graph, start, end)

    # 记录结束时间
    end_time = time.time()
    # 计算时间差
    elapsed_time = end_time - start_time
    print(f"程序运行时间: {elapsed_time} 秒\n")

    print(f"最短路径: {' - '.join(shortest_path)}, 长度: {shortest_dis
}")
```

文件读取版:

Dijkstra_v2 Python

```python
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
from queue import PriorityQueue
import sys
import time
import threading

def dijkstra(graph, start, end):
    """
    迪杰斯特拉函数，求最短路径
    参数:字典graph,起点start,终点end
    返回:最短路长度dis[end],路径列表path

    """
    print(f"Start node: {start}, end_node: {end}\n")
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    pq = PriorityQueue()
    pq.put((0, start))
    pre = {node: None for node in graph}

    while not pq.empty():
        cur_dis, cur_node = pq.get()
        for neighbor, weight in graph[cur_node]:
            distance = cur_dis + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                pre[neighbor] = cur_node
                pq.put((distance, neighbor))
    # 回溯最短路径
    path = []
    cur = end
    while cur is not None:
        path.insert(0, cur)
        cur = pre[cur]

    return distances[end], path

# 读取文本内容,字符串file_content
with open('text.txt', 'r') as file:
    file_content = file.read()

```

```python
# 以/n分割file_content,列表input_lines
input_lines = file_content.split('\n')

m, n = int(input_lines[0].split()[0]), int(input_lines[0].split()[1])

graph = {}

if len(set(node for edge_line in input_lines[1:n+1] for node in edge_line.split()[:2])) != m:
    print("错误：图中的节点数量与指定的节点数量不匹配。")
    sys.exit()

# 从第二行开始是点1,点2,边权
graph = {}
for i in range(1, n + 1):
    edge_line = input_lines[i].split()
    node1, node2, weight = edge_line[0], edge_line[1], int(edge_line[2])
    graph.setdefault(node1, []).append((node2, weight))
    graph.setdefault(node2, []).append((node1, weight))


# 转换为networkx图
G = nx.Graph()
for node, neighbors in graph.items():
    for neighbor, weight in neighbors:
        G.add_edge(node, neighbor, weight=weight)

# 绘制图形
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, font_weight='bold', node_size=700, node_color='skyblue', font_size=10, font_color='black')
edge_labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
plt.show(block=True)
# 从第n行开始是查询
for line in input_lines[n + 1:]:
    if not line:
        continue  # 如果是空的就跳过
    start, end = line.split()
    shortest_dis, shortest_path = dijkstra(graph, start, end)
    print(f"最短路径: {' - '.join(shortest_path)}, 长度: {shortest_dis}")
```

四、创新点&优化

使用了numpy库进行优化。由于numpy数组的索引只能是int类型的，所以我新建了一个字典node_to_index来实现字母到数字索引的映射。

同时，我利用networkx和matplotlib对无向图实现了可视化处理，具体解释如下：

> G: 要绘制的图对象。
>
> pos: 节点的布局，这里使用了 spring_layout 算法得到的布局。
>
> with_labels: 是否在节点上标注标签，设为 True 表示显示节点标签。
>
> font_weight: 节点标签的字体粗细。
>
> node_size: 节点的大小。
>
> node_color: 节点的颜色。
>
> font_size: 节点标签的字体大小。
>
> font_color: 节点标签的颜色。
>
> edge_labels：一个字典，键是元组(u,v)，值是边的属性也就是weight，将作为边的标签传入到 draw_networkx_edge_labels()里。

Dijkstra_v3 Python

```python
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
from queue import PriorityQueue
import sys
import time
import threading



def dijkstra(graph, start, end):
    """
    迪杰斯特拉函数，求最短路径
    参数:字典graph,起点start,终点end
    返回:最短路长度dis[end],路径列表path

    """
    dis = {node: float('inf') for node in range(len(graph))}
    dis[start] = 0
    pq = PriorityQueue()
    pq.put((0, start))
    pre = {node: None for node in range(len(graph))}

    while not pq.empty():
        cur_dis, cur_node = pq.get()

        # 同时获取索引和值
        for neighbor, weight in enumerate(graph[cur_node]):# 索引和值
            # 过滤掉没有连接的 其实也可以不过滤)
            if weight < np.inf:
                distance = cur_dis + weight
                if distance < dis[neighbor]:
                    dis[neighbor] = distance
                    pre[neighbor] = cur_node
                    pq.put((distance, neighbor))
    # 回溯最短路径
    path = []
    current = end
    while current is not None:
        path.insert(0, current)
        current = pre[current]

    return dis[end], path
```

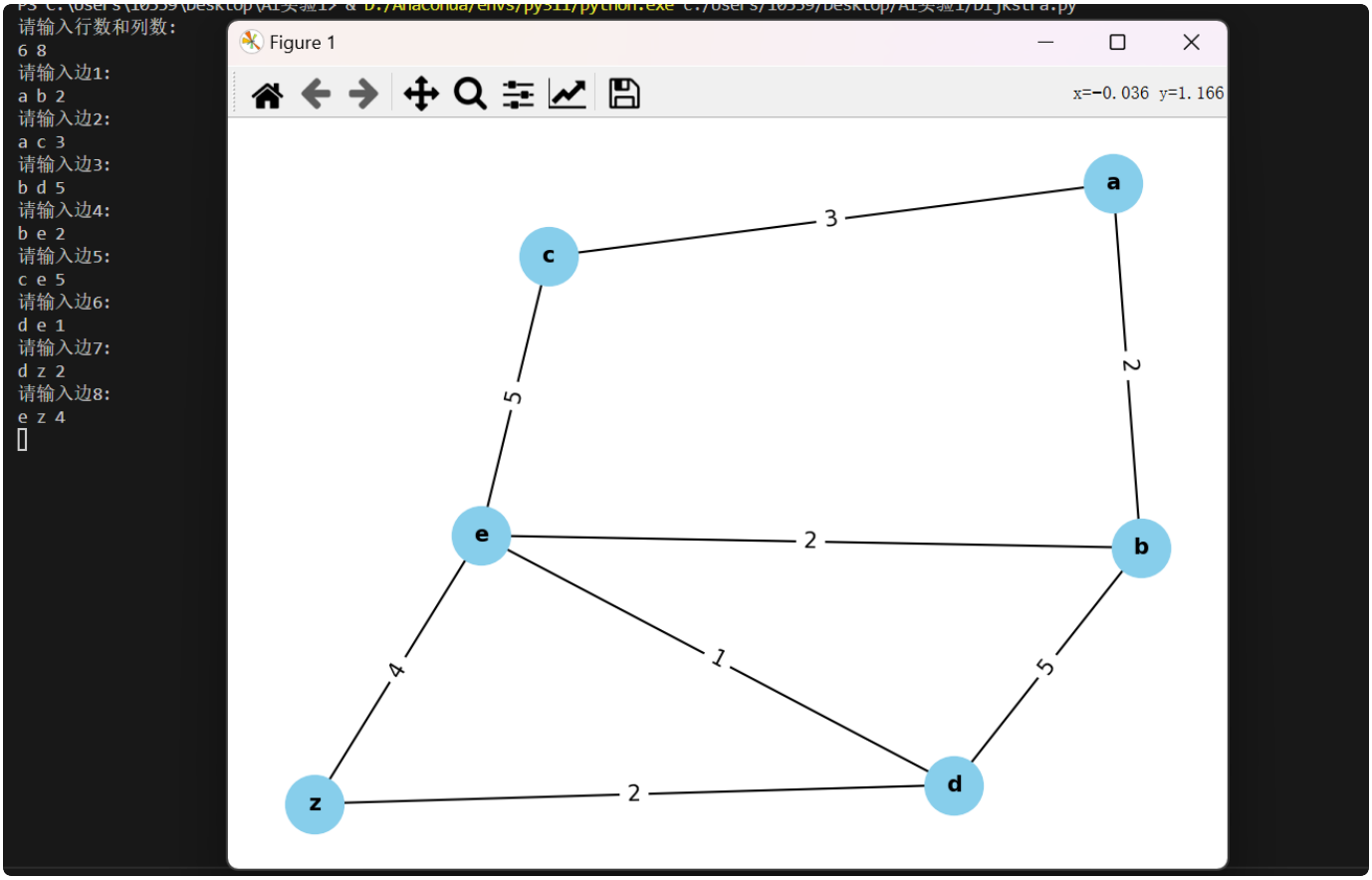
```
44
45  # 读取输入数据
46  while True:
47      input_line = input().split()
48      m, n = int(input_line[0]), int(input_line[1])
49
50      # 映射节点到整数索引
51      node_to_index = {}
52
53      # 创建一个m*m的二维数组，初始值无穷大
54      graph = np.full((m, m), np.inf)
55
56      # 读取边的信息
57      for _ in range(n):
58          edge_line = input().split()
59          node1, node2, weight = edge_line[0], edge_line[1], int(edge_
    line[2])
60
61          node_to_index.setdefault(node1, len(node_to_index))
62          node_to_index.setdefault(node2, len(node_to_index))
63          index1, index2 = node_to_index[node1], node_to_index[node2]
64
65          graph[index1][index2] = weight
66          graph[index2][index1] = weight
67
68      # 获取数组形状，graph.shape[0]:行数，graph.shape[1]:列数
69      if graph.shape[0] != m:
70          print("图中节点数与输入不相等，请重新输入")
71      else:
72          break
73
74
75  print()
76  print(graph)
77  print()
78
79  G = nx.Graph()
80  for i in range(m):
81      for j in range(i+1, m):
82          if graph[i][j] < np.inf:
83              G.add_edge(i, j, weight=graph[i][j])
84
85  # 绘制图形
86  pos = nx.spring_layout(G)
87
```

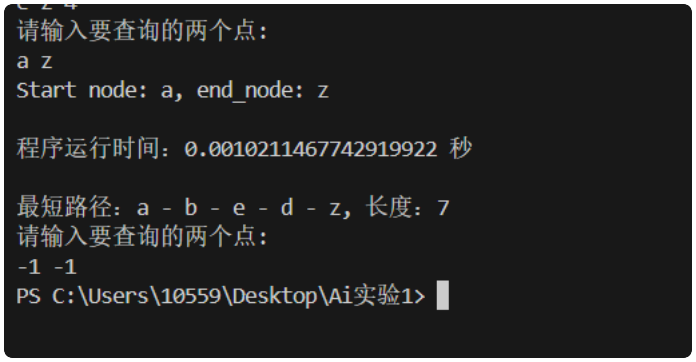
```
88  nx.draw(G, pos, with_labels=True, font_weight='bold', node_size=700,
89   node_color='skyblue', font_size=10, font_color='black')
90  edge_labels = nx.get_edge_attributes(G, 'weight')
91  nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
92  plt.show()
93
94  # 处理循环输入
95  while True:
96      input_line = input().split()
97      start, end = input_line[0], input_line[1]
98      if start == "-1" and end == "-1":
            break
99      index_start, index_end = node_to_index[start], node_to_index[end
100 ]
101
102     start_time = time.time()

103     shortest_distance, shortest_path = dijkstra(graph, index_start,
104 index_end)
105
106     # 记录结束时间
107     end_time = time.time()
108     # 计算时间差
109     elapsed_time = end_time - start_time
110     print(f"程序运行时间: {elapsed_time} 秒\n")
111
        # shortest_path里面是数字，要将数字索引转换回字符
112     shortest_path_chars = [node for node in node_to_index.keys() if
    node_to_index[node] in shortest_path]
113     print(f"最短路径: {'-'.join(shortest_path_chars)}, 长度: {shortest
114 _distance}")
```

3 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）

    a. Dijkstra.py

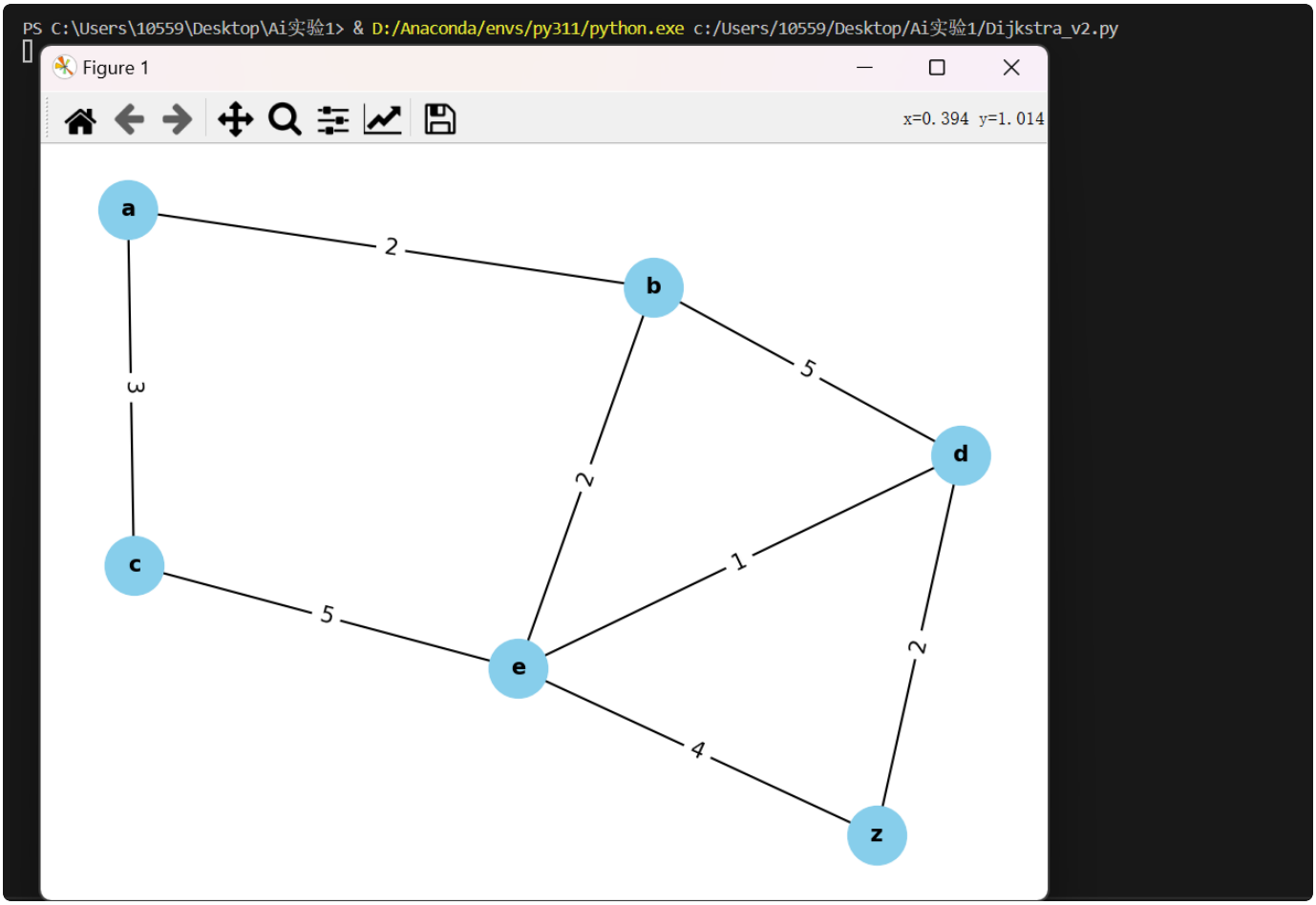




b. Dijkstra_v2.py

text.txt Plain Text

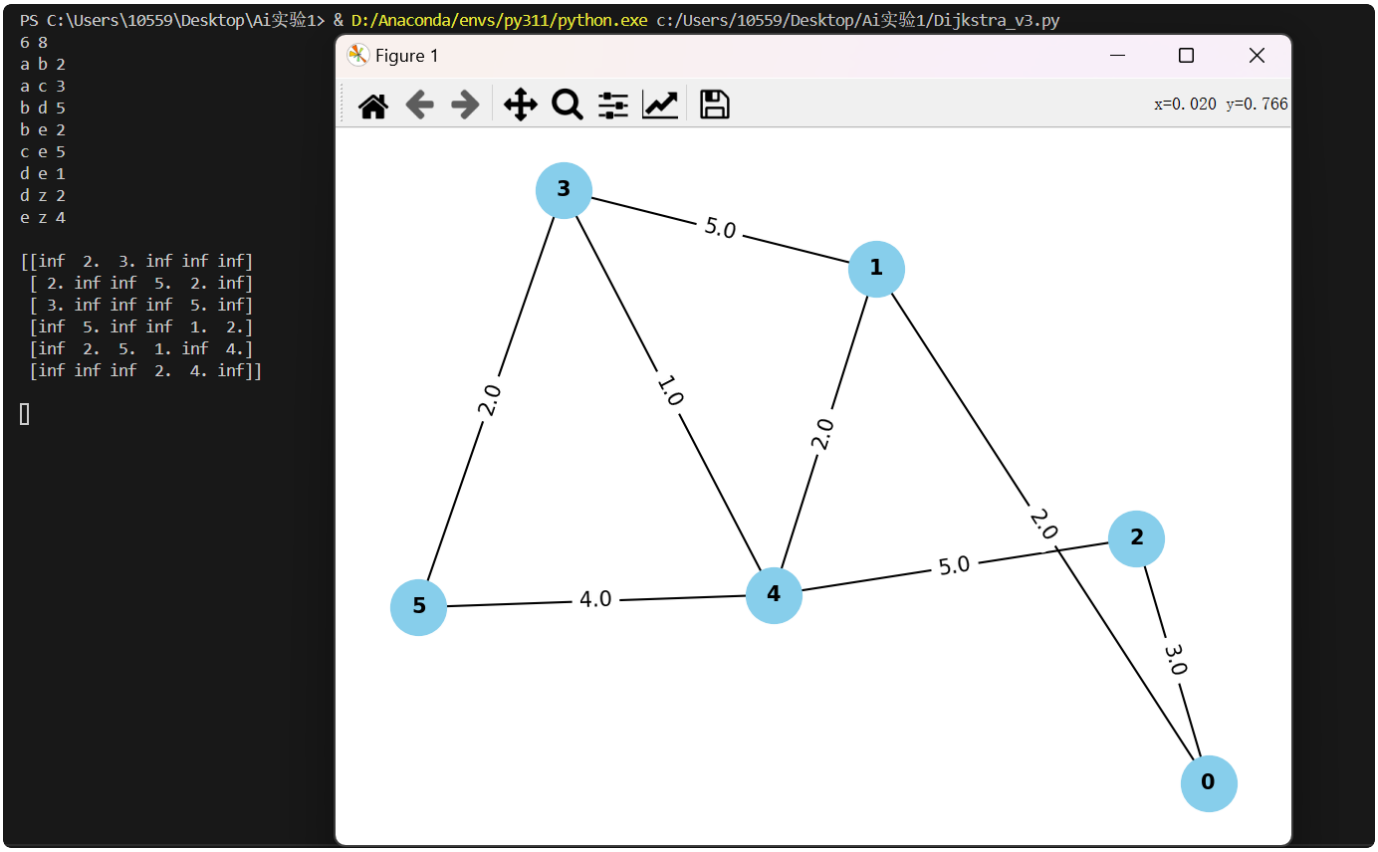
```
6 8
a b 2
a c 3
b d 5
b e 2
c e 5
d e 1
d z 2
e z 4
a z
```




```
PS C:\Users\10559\Desktop\Ai实验1> & D:/Anaconda/envs/py311/python.exe c:/Users/10559/Desktop/Ai实验1/Dijkstra_v2.py
Start node: a, end_node: z

最短路径: a - b - e - d - z, 长度: 7
PS C:\Users\10559\Desktop\Ai实验1>
```

|----------numpy优化后----------|

1. 实验结果展示示例




```
a z
程序运行时间：0.0 秒

最短路径：a-b-d-e-z，长度：7.0
b z
程序运行时间：0.0 秒

最短路径：b-d-e-z，长度：5.0
c z
程序运行时间：0.0 秒

最短路径：c-d-e-z，长度：8.0
d z
程序运行时间：0.0009541511535644531 秒

最短路径：d-z，长度：2.0
-1 -1
PS C:\Users\10559\Desktop\Ai实验1>
```

4 思考题

无

5 参考资料

最短路 – OI Wiki

此处为语雀内容卡片，点击链接查看：https://www.yuque.com/taoyzh/mip82z/eqft9oqpg09n0gsa?singleDoc=&view=doc_embed

《krusual,prim,dijkstra》

Python中的None_python none–CSDN博客

Python字典使用教程：Python字典常用操作方法_python字典的基本操作–CSDN博客

GitHub – jackfrued/Python–100–Days: Python – 100天从新手到大师

Python之Numpy详细教程_python numpy–CSDN博客

Python 使用 NetworkX_python networkx–CSDN博客

networkx画图时显示节点和边的属性_draw_networkx_edge_labels–CSDN博客

Python获取程序运行时间的三种方法_python 运行时间–CSDN博客