第一次实验报告

实验1 最短路径算法

陳日康　信息与计算科学　**22336049**

一、实验要求

给定无向图以及图上两个节点，使用 python 实现 Dijkstra 算法，求出其最短路径以及长度，加深对 python 编程语言的认识，掌握图算法在实际问题中的应用，提升对数据结构与算法的综合运用能力。

二、算法原理

Dijkstra 算法是是典型最短路径算法，由荷兰计算机科学家艾兹赫尔·戴克斯特拉在 1956 年发现的算法，类似广度优先搜索的方法解决赋权图的单源最短路径问题。Dijkstra 算法是从一个顶点到其余各顶点的最短路径算法，采用的是贪心算法的策略，解决的是图上带权的单源最短路径问题。

Dijkstra 算法是从选定的起始点开始，首先把起点到所有点的距离存下来找个最短的，遍历该节点的所有邻居，计算从起始节点经过当前节点到邻居节点的距离会不会更近，如果距离更近了就更新距离，直到扩展到终点为止。这样把所有的点找遍之后就存下了起点到其他所有点的最短距离。

三、流程图

Dijkstra 算法是一种用于找到带权有向图中单源最短路径的贪婪算法。以下是算法的主要步骤：

1. **初始化：**
    - 创建一个优先级队列（通常使用最小堆）来存储待处理的节点，初始时将起始节点放入队列中。
    - 创建一个距离字典，记录从起始节点到每个节点的当前最短距离。初始时，将起始节点的距离设置为 0，其他节点的距离设置为无穷大。
2. **迭代：**
    - 从优先级队列中弹出当前距离最短的节点。
    - 遍历该节点的所有邻居，计算从起始节点经过当前节点到邻居节点的距离。
    - 如果通过当前节点到邻居节点的路径距离比当前记录的最短距离短，更新距离字典中的值，并将邻居节点加入优先级队列。
3. **重复：**
    - 重复迭代步骤，直到优先级队列为空。
4. **结果：**
    - 最终，距离字典中记录了从起始节点到每个节点的最短距离。
5. **路径重建：**
    - 若要获取最短路径，可以在算法执行的过程中记录每个节点的前驱节点。最后，通过回溯前驱节点，可以重建从起始节点到目标节点的最短路径。

Dijkstra 算法的关键思想是通过不断地选择距离最短的节点，逐步更新最短距离信息，最终找到起始节点到其他所有节点的最短路径。由于每次选择最短路径，该算法确保在处理过的节点中选择最短路径，因此不会错过最优解。

四、关键代码展示

1. `dijkstra` 函数

```
def dijkstra(graph, start):
    priority_queue = [(0, start)]
    distance = {vertex: float('infinity') for vertex in graph}
    distance[start] = 0
    predecessor = {vertex: None for vertex in graph}

    while priority_queue:
        current_distance, current_vertex = heapq.heappop(priority_queue)

        if current_distance > distance[current_vertex]:
            continue

        for neighbor, weight in graph[current_vertex].items():
            distance_to_neighbor = current_distance + weight

            if distance_to_neighbor < distance[neighbor]:
                distance[neighbor] = distance_to_neighbor
                predecessor[neighbor] = current_vertex
                heapq.heappush(priority_queue, (distance_to_neighbor, neighbor))

    return distance, predecessor
```

`dijkstra` 函数实现了 Dijkstra 算法，计算从指定起始节点到所有其他节点的最短路径和距离。使用优先级队列（最小堆）`priority_queue` 存储待处理的节点，其中元素为（当前距离，节点）。`distance` 字典用于记录从起始节点到每个节点的当前最短距离，初始值为无穷大，起始节点的距离为 0。`predecessor` 字典用于记录每个节点的前驱节点，即当前节点的最短路径中，该节点的前一个节点。

2. `get_shortest_path` 函数

```
def get_shortest_path(predecessor, start, target):
    path = []
    current_vertex = target

    while current_vertex is not None:
        path.insert(0, current_vertex)
        current_vertex = predecessor[current_vertex]

    return path
```

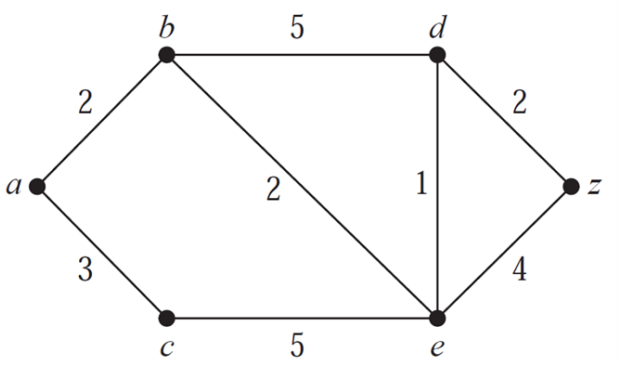
`get_shortest_path` 函数接受前驱节点字典、起始节点和目标节点作为输入，返回从起始节点到目标节点的最短路径。通过回溯前驱节点，从目标节点开始，逐步将路径中的节点插入 `path` 列表，直至回溯到起始节点。最终，`path` 中包含了从起始节点到目标节点的最短路径节点序列。

这两个函数是 Dijkstra 算法实现中的关键部分，`dijkstra` 函数执行算法的核心逻辑，`get_shortest_path` 用于路径的重建。程序的其余部分主要是输入处理和结果输出。

五、实验结果

- 样例

```
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
```



上图为实验课 ppt 中所提供的测试用例。

```
*IDLE Shell 3.11.2*
Python 3.11.2 (v3.11.2:878ead1ac1, Feb  7 2023, 10:02:41) [Clang 13.0.0 (clang-1
300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======== RESTART: /Users/carrieng/Desktop/python project/E1_22336049.py ========
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
Shortest distance from a to z: 7
Shortest path: a -> b -> e -> d -> z
```

上图为程序运行结果，结果正确。

六、参考资料

https://blog.csdn.net/qq_52905520/article/details/126312129

https://blog.csdn.net/m0_74939592/article/details/134021101

https://gitcode.csdn.net/65e6edf91a836825ed7887c4.html?dp_token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6NDI3Nzc3MCwiZXhwIjoxNzEwNzM2MzMwLCJpYXQiOjE3MTAxMzE1MzAsInVzZXJuYW1lIjoiMjMwMl83NjkwMzkxNiJ9.zJ4bJkOhcBVL4i1M5T2icMRuUay-xfq1K1SGtI3O338