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	计算机科学与技术	专业（方向）	系统结构
学号	22336018	姓名	蔡可豪

1 实验题目

实现最短路径算法：给定无向图，以及图上的两个节点，求**最短路径**及其**长度**。使用**python**实现，至少实现**Dijkstra**算法。

2 实验内容

2.1 算法原理

Dijkstra据说是在一天早上喝咖啡时Wandering出这个著名算法的。算法解决的问题是如何在给定无向图的情况下，找到两点间的最短路径及其长度。
这其实是一个很“生活化”的问题，因为日常我们要去到一个地方，一般都是所谓的无向图，而且“条条大路通罗马”。
这个算法，本质是通过贪心策略解决了这个问题，核心思想其实就是一句话

> “为了找到到达下一个节点的最短路径，必须要找到可以到达下一个节点的前一个节点的最短路径。”

如此，用代码实现是比较简单的。无非就是动态更新状态然后不断比较前进。

2.2 伪代码

```
类 Graph:
    初始化(顶点数):
        设置顶点数 = 顶点数
        初始化邻接表为顶点数大小的列表，每个元素是空列表

    添加边(起点, 终点, 权重):
        在邻接表中起点的列表中添加(终点, 权重)
        在邻接表中终点的列表中添加(起点, 权重)

    Dijkstra算法(源点, 目标点):
        初始化距离列表，大小为顶点数，所有值为无穷大
        将源点的距离设置为0
```

```
初始化前驱列表，大小为顶点数，所有值为-1
初始化优先队列，并将(0, 源点)加入队列

当优先队列非空时:
    从优先队列中取出当前距离和当前节点
    如果当前节点是目标点，则中断循环
    如果当前距离大于当前节点在距离列表中的值，则继续下一次循环

    遍历当前节点的所有邻接节点及其权重:
        如果当前距离加上权重小于邻接节点在距离列表中的值:
            更新邻接节点在距离列表中的值为当前距离加上权重
            将(更新后的距离, 邻接节点)加入优先队列
            更新邻接节点的前驱为当前节点

初始化路径列表
从目标点开始回溯直到源点，构建路径列表
反转路径列表

返回路径列表和到目标点的距离
```

2.3 关键代码展示（含注释）

```python
import heapq

class Graph():
    def __init__(self, vertices):
        self.vertices = vertices  # 顶点数
        self.adj = [[] for _ in range(vertices)]  # 初始化邻接表

    def addEdge(self, nodeA, nodeB, weight):
        # 向图中添加一条边，包括起点、终点和权重
        self.adj[nodeA].append((nodeB, weight))
        self.adj[nodeB].append((nodeA, weight))

    def dijkstra(self, src, target):
        # 使用Dijkstra算法计算从源点到目标点的最短路径
        distances = [float('inf')] * self.vertices  # 所有顶点的距离初始化为无穷大
        distances[src] = 0  # 源点到自身的距离是0
        pre = [-1] * self.vertices  # 前驱节点初始化为-1
        pq = [(0, src)]  # 优先队列

        while pq:
            currentDistance, currentNode = heapq.heappop(pq)
            # 弹出当前距离最小的节点
            if currentNode == target:
            # 如果当前节点是目标节点，则退出循环
```

```
                break
            if currentDistance > distances[currentNode]:
            # 如果当前距离大于已知最短距离，则忽略
                continue

            for neighbor, weight in self.adj[currentNode]:
             # 遍历当前节点的所有邻接节点
                if weight + currentDistance < distances[neighbor]:  # 更新
距离
                    distances[neighbor] = weight + currentDistance
                    heapq.heappush(pq, (distances[neighbor], neighbor))
                    # 将新的距离和节点加入优先队列
                    pre[neighbor] = currentNode  # 更新前驱节点

        path, current = [], target  # 从目标节点回溯到源节点构建路径
        while current != -1:
            path.append(current)
            current = pre[current]
        path.reverse()  # 反转路径，因为我们是从目标回溯到源点的

        return path, distances[target]

def main():
    vertices, edges = map(int, input().split())  # 输入顶点和边的数量
    graph = Graph(vertices)

    for i in range(edges):  # 添加所有的边
        fromNode, toNode, weight = map(int, input().split())
        # 输入边的起点、终点和权重
        graph.addEdge(fromNode, toNode, weight)

    src, dest = map(int, input().split())  # 输入源点和目标点
    path, length = graph.dijkstra(src, dest)  # 计算最短路径和长度
    print(f"Minpath: {' -> '.join(map(str, path))}, length = {length}")
    # 输出最短路径和长度

if __name__ == "__main__":
    main()
```

2.4 创新点（优化）

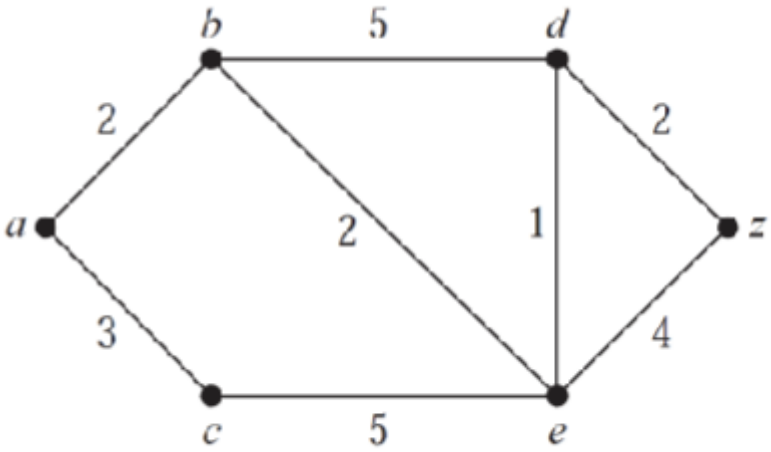
- **使用优先队列**：通过 `heapq` 库，利用优先队列优化了寻找当前最短距离节点的过程。这样就可以不用自己写轮子进行排序，而是直接利用了库中的高效数据结构。
- **路径重构**：利用前驱节点的方式记录并且输出最短路径，而不需要重新编码进行寻找输出。

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）

输入Example:

```
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
```




```
kehao@ALcohol-2:~/CodeSpace/SYSU-AI/lab1|main⚡  ⇒  /opt/homebrew/bin/python3 /Users/kehao/CodeSpace/SYSU-AI/lab1/main.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
Minpath: a -> b -> e -> d -> z, length = 7
kehao@ALcohol-2:~/CodeSpace/SYSU-AI/lab1|main⚡  ⇒
```

四、 思考题

没有布置思考题

五、 参考资料

无参考资料。