中山大學

SUN YAT-SEN UNIVERSITY

《人工智能编程语言》- 期末大作业

使用 Python 语言实现最短路算法

姓　　名：　胡嘉睿

学　　号：　24311019

教学班号：　智慧交通班

专　　业：　智慧交通

院　　系：　智能工程学院

2024~2025 学年第二学期

一. 算法理解与数据预处理

1. 在计算机中储存图

图在数据结构中的表示

图在数据结构中可以使用**邻接矩阵**和**邻接表**两种方式来表示。
邻接矩阵是一个 $n \times n$ 的矩阵，其中 n 是图中顶点的数量，矩阵中的每个元素表示顶点之间的边的权重。邻接表则是一个数组，每个元素是一个线性表，表中存储与该顶点相连的所有顶点及其对应的边的权重。

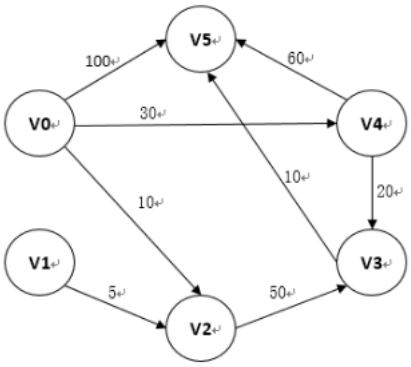

图 1　原始数据

设 图 1 中有顶点 $V_0, V_1, V_2, V_3, V_4, V_5$ 。构建邻接矩阵 A ，其中 A_{ij} 表示从顶点 i 到顶点 j 的边的权重，若不存在边则权重为 0 。

统计边数

V_0 的出边:
- $V_0 \rightarrow V_2$: 权重为 10
- $V_0 \rightarrow V_4$: 权重为 30
- $V_0 \rightarrow V_5$: 权重为 100

V_1 的出边:
- $V_1 \rightarrow V_2$: 权重为 5

V_2 的出边:
- $V_2 \rightarrow V_3$: 权重为 50
- $V_2 \rightarrow V_4$: 权重为 10

V_3 的出边:
- $V_3 \rightarrow V_4$: 权重为 20
- $V_3 \rightarrow V_5$: 权重为 60

V_4 的出边:
- 无出边

V_5 的出边:
- 无出边

对应的邻接矩阵如下：

$$A = \begin{bmatrix} 0 & 0 & 10 & 0 & 30 & 100 \\ 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 50 & 10 & 0 \\ 0 & 0 & 0 & 0 & 20 & 60 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

对应的邻接表如下：

AdjList :

$V_0 : (V_2, 10), (V_4, 30), (V_5, 100);$

$V_1 : (V_2, 5);$

$V_2 : (V_3, 50), (V_4, 10);$

$V_3 : (V_4, 20), (V_5, 60);$

$V_4 : ();$

$V_5 : ();$

2. 模拟 Dijkstra 算法

在 Dijkstra 算法中，我们将邻接矩阵中的权重视为边的长度。对于无直接相连的顶点，权重为 infinity (无穷大)。算法维护一个从源点到各顶点的当前最短距离估计数组 `dist`，以及一个已找到最短路径的顶点集合 `S`。

	V_0	V_1	V_2	V_3	V_4	V_5
V_0	0	0	10	0	30	100
V_1	0	0	5	0	0	0
V_2	0	0	0	50	10	0
V_3	0	0	0	0	20	60
V_4	0	0	0	0	0	0
V_5	0	0	0	0	0	0

表 1　原始邻接矩阵

将邻接矩阵转化为 Dijkstra 算法的输入图（0 权重边转化为 ∞，对角线为 0）：

	V_0	V_1	V_2	V_3	V_4	V_5
V_0	0	∞	10	∞	30	100
V_1	∞	0	5	∞	∞	∞
V_2	∞	∞	0	50	10	∞
V_3	∞	∞	∞	0	20	60
V_4	∞	∞	∞	∞	0	0
V_5	∞	∞	∞	∞	0	0

表 2　Dijkstra 算法输入图

2.1. 以 V_0 为源点

首先我们指定源点 V_0，并初始化 `dist` 数组为 [∞], [10], [∞], [30], [100]，`S` 集合为空。

第一次迭代

1. 从 `dist` 数组中找到最小值 10，对应的顶点为 V_2，将其加入集合 `S`。

$$S = \{V_2\}$$

第二次迭代

1. 从 `dist` 数组中找到最小值 20，对应的顶点为 V_4，将其加入集合 `S`。

$$S = \{V_2, V_4\}$$

2. 检查 V_4 的出边（无出边），无需更新 `dist` 数组。

第三次迭代

1. 从 `dist` 数组中找到最小值 60，对应的顶点为 V_3，将其加入集合 `S`。

$$S = \{V_2, V_4, V_3\}$$

2. 检查 V_3 的出边：
   - $V_3 \to V_4$: $60 + 20 = 80 \geqslant 20$（不更新）
   - $V_3 \to V_5$: $60 + 60 = 120 \geqslant 100$（不更新）

第四次迭代

1. 从 `dist` 数组中找到最小值 100，对应的顶点为 V_5，将其加入集合 `S`。

$$S = \{V_2, V_4, V_3, V_5\}$$

2. 检查 V_5 的出边（无出边），无需更新 `dist` 数组。

最终结果

此时未处理顶点仅剩 V_1，其距离值为 ∞，说明从 V_0 无法到达 V_1。最终结果：

	顶点	V_0	V_1	V_2	V_3	V_4	V_5
最短距离	0	∞	10	60	20	100	

表 3　最终结果

2.2. 以 V_1 为源点

我们以 V_1 为源点，初始化 dist 数组为 [∞], [0], [∞], [∞], [∞]， S 集合为空。

第一次迭代

1. 从 dist 数组中找到最小值 0，对应的顶点为 V_1，将其加入集合 S。

$$S = \{V_1\}$$

第二次迭代

1. 从 dist 数组中找到最小值 5，对应的顶点为 V_2，将其加入集合 S。

$$S = \{V_1, V_2\}$$

2. 检查 V_2 的出边：
   - $V_2 \to V_3$: $5 + 50 = 55 \geqslant \infty$（更新）
   - $V_2 \to V_4$: $5 + 10 = 15 \geqslant \infty$（更新）

第三次迭代

1. 从 dist 数组中找到最小值 15，对应的顶点为 V_4，将其加入集合 S。

$$S = \{V_1, V_2, V_4\}$$

2. 检查 V_4 的出边（无出边），无需更新 dist 数组。

第四次迭代

1. 从 dist 数组中找到最小值 55，对应的顶点为 V_3，将其加入集合 S。

$$S = \{V_1, V_2, V_4, V_3\}$$

2. 检查 V_3 的出边：
   - $V_3 \to V_4$: $55 + 20 = 75 \geqslant 15$（不更新）
   - $V_3 \to V_5$: $55 + 60 = 115 \geqslant \infty$（更新）

第五次迭代

1. 从 dist 数组中找到最小值 100，对应的顶点为 V_5，将其加入集合 S。

$$S = \{V_1, V_2, V_4, V_3, V_5\}$$

2. 检查 V_5 的出边（无出边），无需更新 dist 数组。

最终结果

此时未处理顶点仅剩 V_0，其距离值为 ∞，说明从 V_1 无法到达 V_0。最终结果：

	顶点	V_0	V_1	V_2	V_3	V_4	V_5
最短距离	∞	0	5	55	15	100	115

表 4　最终结果

二. Python 编程

1. 代码实现

将原始矩阵保存在 `data.txt` 文件中，读取数据并构建邻接矩阵。然后实现 Dijkstra 算法，计算从源点到各顶点的最短路径。
具体代码以及注释见附件的 `work.py` 文件。

2. 运行结果

```
candlest@yl15pro ~/文/作/P/Final> python ./work.py
①从v0出发的最短路径: [0, -1, 10, 60, 20, 100]
②从v1出发的最短路径: [-1, 0, 5, 55, 15, 115]

③统计分析:
平均最短距离: 33.16
最大最短距离: 115
最小最短距离: 0
非连通节点比例: 47.22%
kf.kio.filewidgets.kfilefiltercombo: KFileFilterCombo::setCurrentFilter: Could not find file filt
er KFileFilter(MIME patterns: QList() File patterns: QList("*.png") label: "Portable Network Grap
hics")
kf.kio.filewidgets.kfilefiltercombo: KFileFilterCombo::setCurrentFilter: Could not find file filt
er KFileFilter(MIME patterns: QList() File patterns: QList("*.png") label: "Portable Network Grap
hics")
kf.kio.filewidgets.kfilefiltercombo: KFileFilterCombo::setCurrentFilter: Could not find file filt
er KFileFilter(MIME patterns: QList() File patterns: QList("*.png") label: "Portable Network Grap
hics")
candlest@yl15pro ~/文/作/P/Final>
```

图 2　运行结果

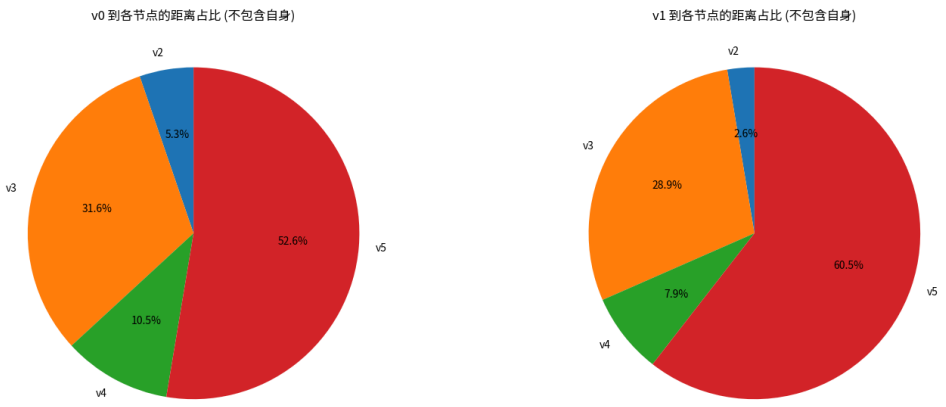

图 3　v_1, v_2 到各节点的最短路径占总的比

	v0	v1	v2	v3	v4	v5
步骤	0	∞	∞	∞	∞	∞
步骤0	0	∞	10	∞	30	100
步骤1	0	∞	10	60	20	100
步骤2	0	∞	10	60	20	100
步骤3	0	∞	10	60	20	100
步骤4	0	∞	10	60	20	100
步骤5	0	-1	10	60	20	100
步骤6						

图 4 v_0到各节点的最短路径动态图（完整动态演示请运行程序查看）

Dijkstra算法过程可视化 (起点: v1, 当前步骤: 6)

	v0	v1	v2	v3	v4	v5
初始化	∞	0	∞	∞	∞	∞
步骤0	∞	0	5	∞	∞	∞
步骤1	∞	0	5	55	15	∞
步骤2	∞	0	5	55	15	∞
步骤3	∞	0	5	55	15	115
步骤4	∞	0	5	55	15	115
步骤5	-1	0	5	55	15	115
步骤6						

图 5 v_1到各节点的最短路径动态图（完整动态演示请运行程序查看）

三. 附加挑战：寻找一种新的最短路算法

Dijkstra 算法已经被证明为**普遍最优** [1]，然而在某些情况下，Dijkstra 算法的性能可能不如其他算法。

比如在稀疏图中，Dijkstra 算法的时间复杂度为 $O(E + V \log V)$，而 Bellman-Ford 算法的时间复杂度为 $O(VE)$。

对于交通领域而言，Dijkstra 算法的性能可能会受到交通流量、路况等因素的影响。比如在高峰期，某些路段的通行能力可能会降低，从而导致 Dijkstra 算法计算出的最短路径不再是最优路径。

同时，交通流量是快速变化的，Dijkstra 算法无法实时更新最短路径。因此，在交通领域，Dijkstra 算法可能不是最优选择。

所以，我尝试使用 A*算法来寻找最短路径。A*算法是一种启发式搜索算法，它结合了 Dijkstra 算法和贪心算法的优点。A* 算法使用启发式函数来估计从当前节点到目标节点的距离，从而更快地找到最短路径。

A*算法的时间复杂度为 $O(E + V \log V)$，在稀疏图中性能优于 Dijkstra 算法。

A*算法的实现与 Dijkstra 算法类似，只需在 Dijkstra 算法的基础上添加**启发式函数**即可。具体实现见附件的 `A_star.py` 和 `work_modified.py` 文件。

启发式函数

启发式函数是一个估计当前节点到目标节点距离的函数，用于指导 A* 算法的搜索方向，从而更快找到最短路径。其选择对算法性能影响显著，常用的启发式函数包括曼哈顿距离、欧几里得距离和切比雪夫距离等。在交通领域，启发式函数可根据交通流量和路况动态调整，以提升算法效率。

在 `A_star.py` 中，我们实现了一个 简单的 A* 算法。可以通过修改启发式函数来适应不同的场景。默认提供了曼哈顿距离和无（退化为 Dijkstra 算法）两种启发式函数。我们在 `work_modified.py` 中调用它。

运行结果如下：

```
candlest@y115pro ~/文/作/P/Final> python ./work_modified.py
从v0出发的最短路径: [0, -1, 10, 60, 20, 100]
从v1出发的最短路径: [-1, 0, 5, 55, 15, 115]

统计分析:
平均最短距离: 33.16
最大最短距离: 115
最小最短距离: 0
非连通节点比例: 47.22%
```

图 6　A*算法运行结果，使用曼哈顿距离

3 参考文献

[1] B. Haeupler, R. Hladík, V. Rozhoň, R. E. Tarjan, 和 J. Tětek, 《Universal Optimality of Dijkstra via Beyond-Worst-Case Heaps》. [在线]. 载于: https://arxiv.org/abs/2311.11793