

# Floyd 算法说明文档

## 1 算法适用场景

Floyd 算法主要用于求解图中所有顶点对之间的最短路径。它适用于有向图和无向图，图的边权可正可负，但存在负权回路时，算法无法正确求出最短路径。

## 2 算法基本思想

Floyd 算法基于动态规划思想，通过引入中间顶点逐步更新图中各顶点之间的距离，寻找最短路径。

- **初始化距离矩阵:** 将图的邻接矩阵作为初始的距离矩阵, 其中  $distance[i][j]$  表示顶点  $i$  到顶点  $j$  的初始距离。
- **逐步引入中间顶点:** 依次尝试将顶点  $k$  作为中间顶点, 对于每一对顶点  $i$  和  $j$ , 检查从顶点  $i$  到顶点  $k$  再到顶点  $j$  的路径距离是否比当前已知的顶点  $i$  到顶点  $j$  的距离更小。如果是, 则更新  $distance[i][j]$ 。
- **重复上述过程:** 直到所有顶点都作为中间顶点被尝试过。最终, 距离矩阵  $distance$  中存储的是图中所有顶点对之间的最短距离。

## 3 算法具体步骤

1. **输入图的邻接矩阵:** 假设图有  $n$  个顶点, 输入邻接矩阵表示图中各顶点之间的边及其权值。
2. **初始化距离矩阵:** 将邻接矩阵赋值给  $distance$  矩阵。

3. **开始迭代：**对每个顶点  $k$  从 0 到  $n - 1$  进行循环：
  - 对每个顶点  $i$  从 0 到  $n - 1$  进行循环：
    - 对每个顶点  $j$  从 0 到  $n - 1$  进行循环：
      - \* 如果  $distance[i][k] + distance[k][j] < distance[i][j]$ ，则更新  $distance[i][j] = distance[i][k] + distance[k][j]$ 。
4. **输出结果：**最后得到的距离矩阵  $distance$  即为所有顶点对之间的最短距离。

## 4 示例说明

假设有图  $G$ ，包含顶点  $v_0$ 、 $v_1$ 、 $v_2$ 、 $v_3$ 、 $v_4$ 、 $v_5$ ，其邻接矩阵如下：

	$v_0$	$v_1$	$v_2$	$v_3$	$v_4$	$v_5$
$v_0$	$\infty$	$\infty$	10	$\infty$	30	100
$v_1$	$\infty$	$\infty$	5	$\infty$	$\infty$	$\infty$
$v_2$	$\infty$	$\infty$	$\infty$	50	$\infty$	$\infty$
$v_3$	$\infty$	$\infty$	$\infty$	50	$\infty$	10
$v_4$	$\infty$	$\infty$	$\infty$	20	$\infty$	60
$v_5$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$

表 1: 邻接矩阵

初始化时， $distance$  矩阵为邻接矩阵。经过一系列迭代更新后，最终得到的距离矩阵如下：

	$v_0$	$v_1$	$v_2$	$v_3$	$v_4$	$v_5$
$v_0$	-1	-1	10	50	30	60
$v_1$	-1	-1	5	55	-1	65
$v_2$	-1	-1	-1	50	-1	60
$v_3$	-1	-1	-1	-1	-1	10
$v_4$	-1	-1	-1	20	-1	30
$v_5$	-1	-1	-1	-1	-1	-1

表 2: 最终距离矩阵

## 5 算法优点

- 简单直观：算法结构简单，易于理解和实现。
- 易于编码：代码实现简洁，适合处理多源最短路径问题。
- 适用于稠密图：在处理稠密图时，性能相对较好。

## 6 算法缺点

- 时间复杂度高：时间复杂度为  $O(n^3)$ ，当图的顶点数较多时，计算效率较低。
- 空间复杂度高：需要存储距离矩阵，空间复杂度为  $O(n^2)$ ，对于大规模图可能占用大量内存。