

Dijkstra 算法

一、算法介绍

迪杰斯特拉算法为了解决有权图的单源最短路径问题，即从图中任意一点出发，到达图中其余各点的最短距离。这里的最短距离，此距离常指欧氏距离。也可以理解为最小代价。即从某一点出发，到达其余各点的最小代价。

二、算法思想

设 $G=(V,E)$ 是一个带权有向图，把图中顶点集合 V 分成两组，第一组为已求出最短路径的顶点集合（用 S 表示，初始时 S 中只有一个源点，以后每求得一条最短路径，就将加入到集合 S 中，直到全部顶点都加入到 S 中，算法就结束了），第二组为其余未确定最短路径的顶点集合（用 U 表示），按最短路径长度的递增次序依次把第二组的顶点加入 S 中。在加入的过程中，总保持从源点 v 到 S 中各顶点的最短路径长度不大于从源点 v 到 U 中任何顶点的最短路径长度。此外，每个顶点对应一个距离， S 中的顶点的距离就是从 v 到此顶点的最短路径长度， U 中的顶点的距离，是从 v 到此顶点只包括 S 中的顶点为中间顶点的当前最短路径长度。

三、算法步骤

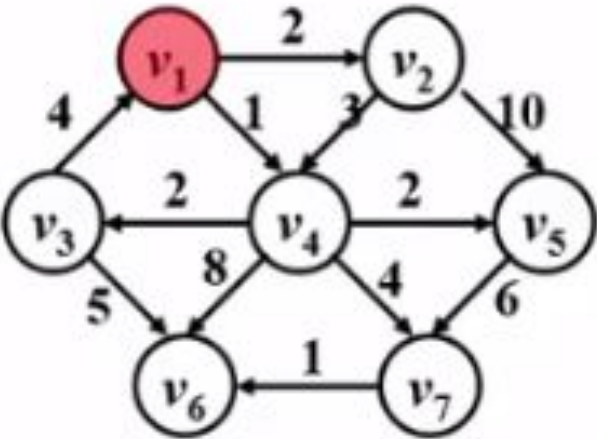
1、初始时， S 只包含源点，即 $S = \{v\}$ ， v 的距离为 0。 U 包含除 v 外的其他顶点，即： $U = \{\text{其余顶点}\}$ ，若 v 与 U 中顶点 u 有边，则 $\langle u,v \rangle$ 正常有权值，若 u 不是 v 的出边邻接点，则 $\langle u,v \rangle$ 权值为 ∞ 。

2、从 U 中选取一个距离 v 最小的顶点 k ，把 k ，加入 S 中（该选定的距离就是 v 到 k 的最短路径长度）。

3、以 k 为新考虑的中间点，修改 U 中各顶点的距离；若从源点 v 到顶点 u 的距离（经过顶点 k ）比原来距离（不经过顶点 k ）短，则修改顶点 u 的距离值，修改后的距离值的顶点 k 的距离加上边上的权。

4、重复步骤 b 和 c 直到所有顶点都包含在 S 中。

四、示例



问题叙述：如图所示，求以 v_1 为起始点，分别达到各点的最短距离和路径

问题解决：

1、设置数组 $dist[v]$ 表示从起点到达 v 点的最短距离，设置数组 $path[v]$ 表示到达 v 点的一个顶点。初始化设置所有 $dist = \infty$ ， $path = -1$ 。

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
dist	∞	∞	∞	∞	∞	∞	∞
path	-1	-1	-1	-1	-1	-1	-1

2、从起点 v_1 开始，因为 v_1 到自己的距离为 0，所以 $dist[v_1]=0$ 。 v_1 到 v_2 和 v_4 的距离分别为 2 和 1，此时， $dist[v_2]=2$ ， $dist[v_4]=1$ ， $path[v_2]=v_1$ ， $path[v_4]=v_1$ 。

	v1	v2	v3	v4	v5	v6	v7
dist	0	2	∞	1	∞	∞	∞
path	-1	V1	-1	V1	-1	-1	-1

3、下面进入 dijkstra 算法，因为 v1 已经在 S 中，所以下面选择未收录顶点中 dist 最小的 v4 进入 S，v4 有 v3,v5,v6,v7 四个邻接点，对于每个临界点进行如下判断：首先判断该点是否已经在 S 中，若已经在 S 中则跳过该点继续判断下个点；若该点 v_i (i 属于 3、5、6、7) 不在 S 中，则判断经由 v4 到达该点的距离+dist[v4]是否比 dist[v_i]更小，取其中更小的数值更新 dist[v_i]，并将 path[v_i]更新为 v4。

	v1	v2	v3	v4	v5	v6	v7
dist	0	2	3	1	3	9	5
path	-1	V1	V4	V1	V4	V4	V4

4、当 v4 的所有邻接点均判断完成后，再次选择新的顶点进入 S。观察发现 v2 是未收录的顶点中 dist 最小的，所以将 v2 加入 S。v2 有两个邻接点 v4 和 v5，其中 v4 已在 S 中，不与讨论，只看 v5,此时计算原点经过 v2 到 v5 的距离是 v2 到 v5 的距离+dist[v2]=12，比 v5 现在的 dist[v5]还要大，因此 dist[v5]不变，path[v5]也不变。

	v1	v2	v3	v4	v5	v6	v7
dist	0	2	3	1	3	9	5
path	-1	V1	V4	V1	V4	V4	V4

5、当 v2 的所有邻接点判断完后，从剩余的未被收录到 S 中的顶点里选择 dist 最小的 v3 (或者 v5) 加入 S。v3 的邻接点有 v1 和 v6，其中 v1 已经在 S 中，只需要判断 v6，从原点经过 v3 到 v6 的距离等于从 v3 到 v6 的距离+dist[v3]=8。比 v6 现在的 dist[v6]的值要小，所以更新 dist[v6]=8，并且更新 path[6]=v3。

	v1	v2	v3	v4	v5	v6	v7
dist	0	2	3	1	3	8	5
path	-1	V1	V4	V1	V4	V3	V4

6、当 v3 的所有邻接点判断完后，从剩余的未被收录到 S 中的顶点里选择 dist 最小的 v5 加入到 S。v5 的邻接点只有 v7，从原点经过 v5 到 v7 的距离等于 v5 到 v7 的距离 + dist[v5]=9。比现在 v7 的距离 dist[v7]还大，所以 dist[v7]不变，path[v7]不变。

	v1	v2	v3	v4	v5	v6	v7
dist	0	2	3	1	3	8	5
path	-1	V1	V4	V1	V4	V3	V4

7、当 v5 的所有邻接点判断完后，从剩余的未被收录到 S 中的顶点里选择 dist 最小的 v7 加入到 S，v7 的邻接点只有 v6，从原点经过 v7 到 v6 的距离等于 v7 到 v6 的距离 + dist[v7]=6，比现在的 dist[6]要小，所以更新 dist[v6]=6，path[v6]=v7。

	v1	v2	v3	v4	v5	v6	v7
dist	0	2	3	1	3	6	5
path	-1	V1	V4	V1	V4	V7	V4

8、当 v7 的所有邻接点判断完后，只剩下 v6 了，将 v6 加入 S 中。此时 v6 已经没有邻接点了。故此时算法结束。

	v1	v2	v3	v4	v5	v6	v7
dist	0	2	3	1	3	6	5
path	-1	V1	V4	V1	V4	V7	V4