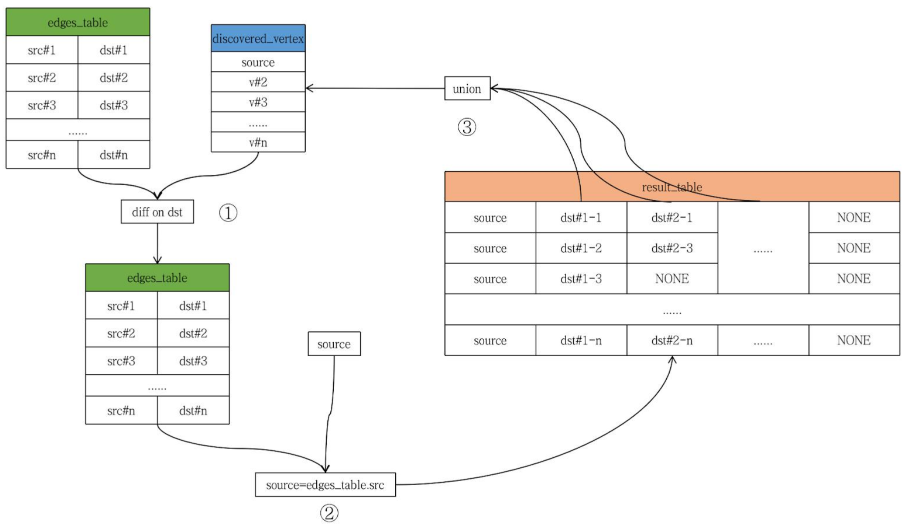
图算法设计说明

1、DFS & BFS



①剔除包含已发现顶点的边;

②迭代的关联 source 的邻居顶点以及邻居顶点的邻居顶点;

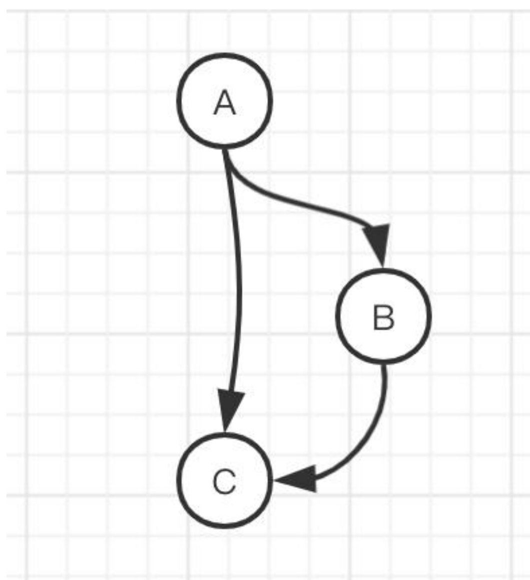
③新发现的顶点插入到已发现的顶点集中;

直到不再有新的邻居顶点。

1）DFS

result_table				
source	dst#1-1	dst#2-1	......	NONE
source	dst#1-2	dst#2-3		NONE
source	dst#1-3	NONE		NONE
......				
source	dst#1-n	dst#2-n	......	NONE

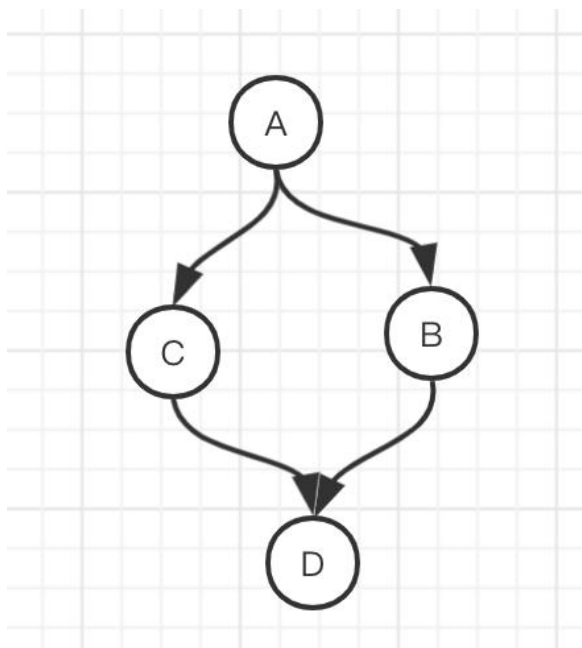
DFS 的结果不唯一，依赖于搜索进程先访问哪一个邻居顶点。在这种算法中，顶点总是出现在离 source 顶点最短的那一条路径中（如，下图中 DFS 得到的路径是 A-->B, A-->C，而不是 A-->B-->C）。



2）BFS

result_table				
source	dst#1-1	dst#2-1	......	NONE
source	dst#1-2	dst#2-3		NONE
source	dst#1-3	NONE		NONE
		......		
source	dst#1-n	dst#2-n	......	NONE

当从 source 顶点到某个顶点存在两条相同长度的路径时（下图中 A-->B-->D, A-->C-->D），此算法的表现与 networkx 不完全一致，此算法将返回 B-->D 与 C-->D 两条边，而 networkx 只会返回其中一条边，因为 D 顶点已经在 B/C 的邻居顶点中出现。



2、拓扑排序

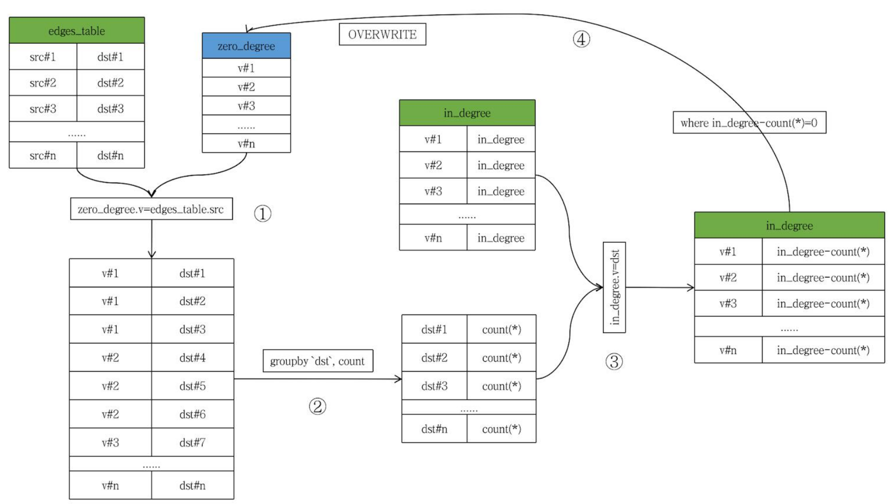
判断图中是否有环，对于无环图，同时得到图的拓扑排序。

1）排列完全孤立的顶点；

2）创建顶点入度的信息；

3）找出图中所有入度为 0 的顶点集 zero_degree，并排列这些顶点；

4）迭代入度为 0 的顶点



①关联入度为 0 的顶点的邻居；

②基于关联到的这些边，统计邻居顶点的入度；

③更新 in_degree 表中，邻居顶点的入度；

④将 in_degree 中入度变为 0 的顶点排列，并覆盖到 zero_degree 表中，开始下一轮的迭代。

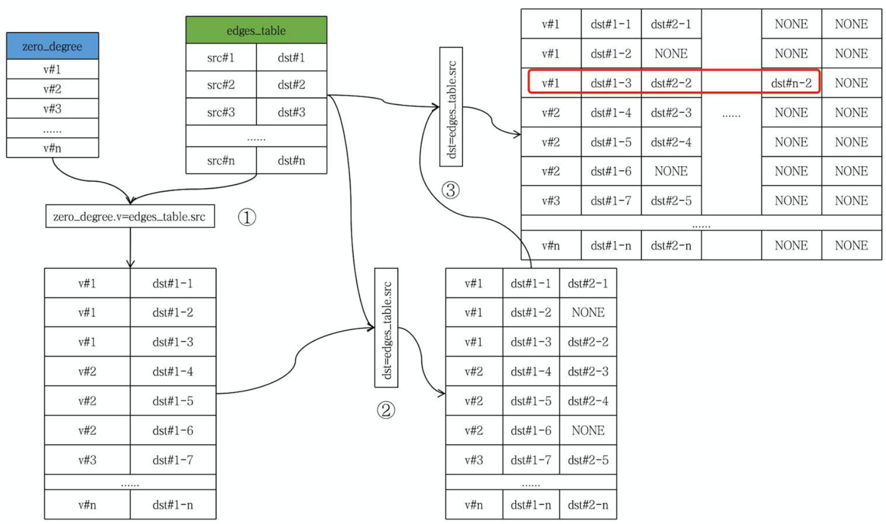
直到，不再有新的入度为 0 的顶点产生，如果此时，in_degree 表中不存在入度为大于 0 的顶点，则图中无环，否则，图中有环。

算法介绍 Topological sorting

3、（DAG 中的）最长路径

1）找出图中所有入度为 0 的顶点集 zero_degree；

2）迭代 zero_degree，关联邻居顶点



①关联入度为 0 的顶点的邻居顶点；

②关联邻居顶点的邻居顶点；

③直到再无新的邻居顶点，则最后一个邻居顶点所在的路径，即为最长路径。

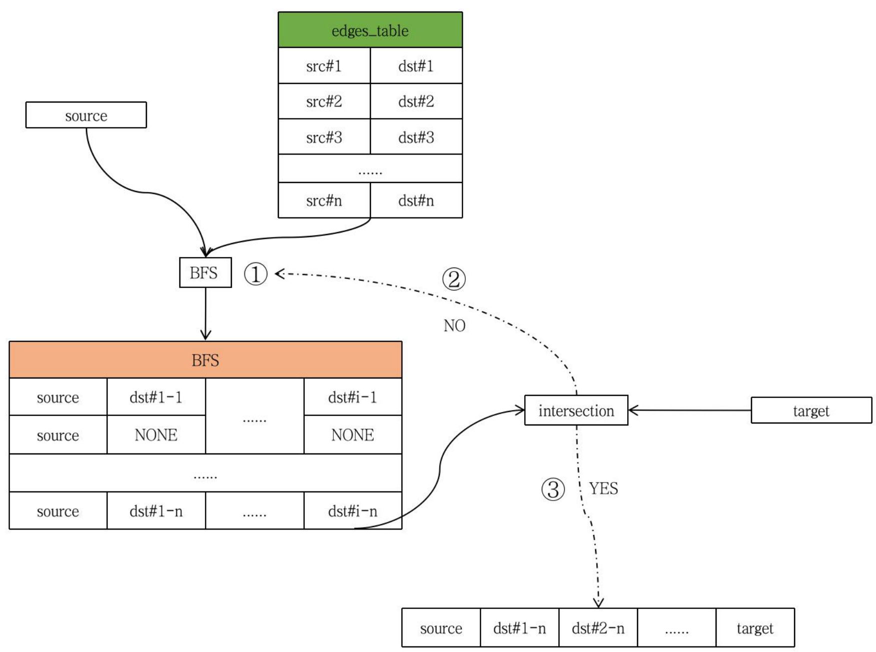
如果考虑边的权重，则在迭代时，添加另外一列，维护当前路径的加权长度。

4、两个顶点之间是否可达以及最短路径

4.1 无权重边（BFS 方法）

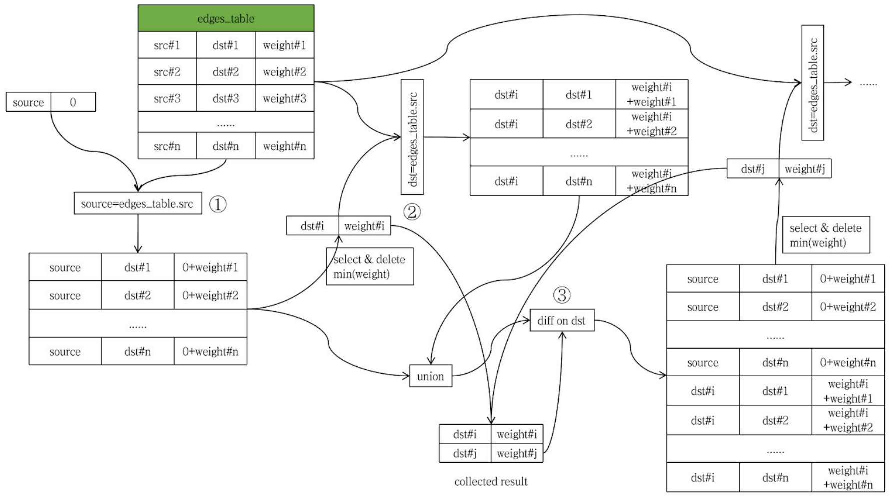
1）从 source 顶点开始，执行一次 BFS，检查 target 顶点是否在 source 的邻居顶点中，如果在，则得到最短路径；

2）继续执行 BFS，检查 target 顶点是否出现在 BFS 结果的最后一层中，若有，则得到最短路径；若无，则继续执行 BFS。



直到 BFS 无新的顶点，此时，顶点之间无可达路径。

4.2 有权重边（Dijkstra's Method）



①关联 source 顶点的邻居顶点以及边的权重（图中，从 source 开始的最短路径的 target 顶点一定是其邻居顶点）；

②松弛操作：提取并删除最短路径顶点 v 及其加权距离，关联 v 的邻居顶点并更新邻居顶点的加权距离；

③提取最短路径顶点：将顶点 v 的路径顶点的加权距离表合并到现有的表中，并剔除已获得最短路径的顶点；

从合并后的表中提取并删除最短路径顶点 u 及其加权距离，对顶点 u 的邻居顶点进行松弛操作，……，直到合并的加权距离表为空。

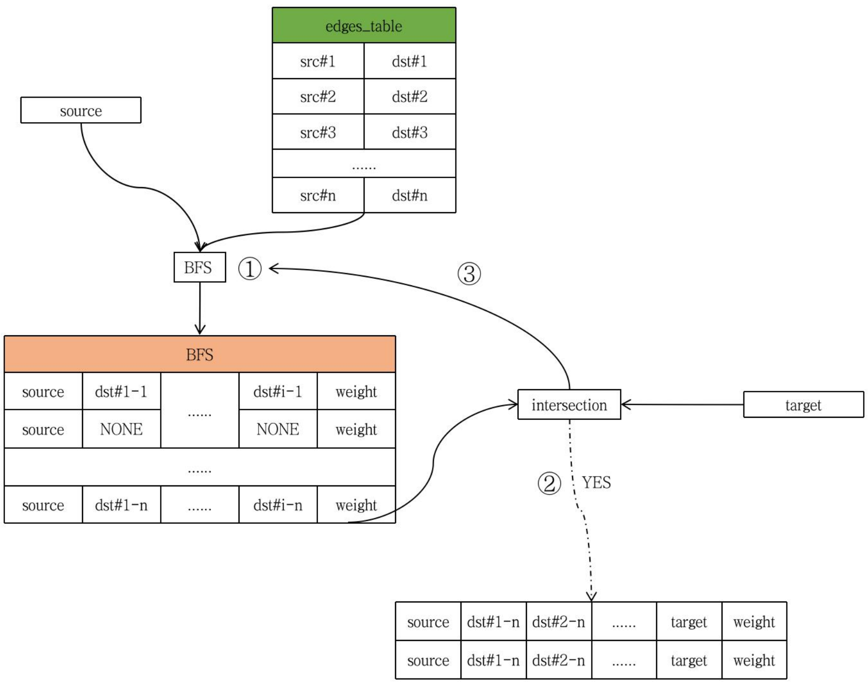
如果②中关联到的顶点的邻居为空，则直接进入到下一轮的“提取最短路径顶点-松弛操作”循环。

如果要计算 source 顶点到某个特定目标顶点 target 的最短距离，那么，在提取最短路径顶点获得 target 顶点时，即可定制迭代。

4.3 有权重边（BFS 方法）

Dijkstra 中要进行 O(|V|)次松弛操作，每次松弛操作查询邻居顶点时，都要进行一次查询操作，当图的规模较大时，会产生大量的查询操作。

1）类似无权重边，不同的是，在搜索到 target 顶点后，该路径停止搜索，其他路径继续搜索，直到所有路径均完成。



2）对所有 source 到 target 的路径，取其中权重最小的路径。

5、（包含某个顶点的）强连通分量（SCC）

1）如果该顶点是孤立顶点，则该顶点自成一个 SCC；

2）如果该顶点入度为 0 而出度不为 0，那么该顶点不在任何一个 SCC 中；

3）从该顶点开始（在有环图中）执行 BFS，得到其后代顶点集；

4）反转边的方向，从 source 开始执行 BFS，得到其（反向）后代顶点集；

5）两个后代顶点集的交集即为包含 source 的 SCC。

算法的讨论与解释

- 从 source 开始搜索，一定可以访问到 SCC 中所有的顶点，因为 SCC 中，任意两个顶点可达；
- BFS 得到的后代顶点集与 DFS 是一致的，BFS 的效率优于 DFS，因此，使用 BFS 替代 DFS；
- source 顶点作为 DFS 数的根节点，完成时间一定是 SCC 中最晚的，因此，反转边的方向后，从 source 顶点开始执行 DFS；
- 反转之后的 DFS 访问到的顶点，并不全在 SCC 中，可能会访问到 source 的父辈顶点，而在全图的 DFS 中，父辈顶点的完成时间晚于 source，只有交集中的顶点的完成时间早于 source，因此，交集构成了 SCC；
- 对图中所有尚未出现在已发现的 SCC 中的顶点，调用以上操作，即可得到图中所有的 SCC。