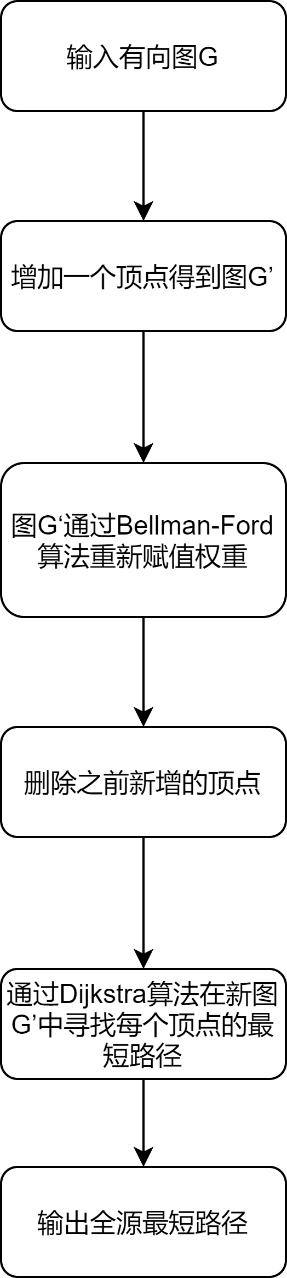
J



Dijkstra算法：

1） 创建一个动态数组，它记录最短路径树中包含的顶点，即计算并最终确定其与源点的最小距离。 初始化时这个数组是空的。

2） 为输入图中的所有顶点分配一个距离值。 将所有距离值初始化为 INFINITE。 将源顶点的距离值指定为 0，以便它首先被检索。

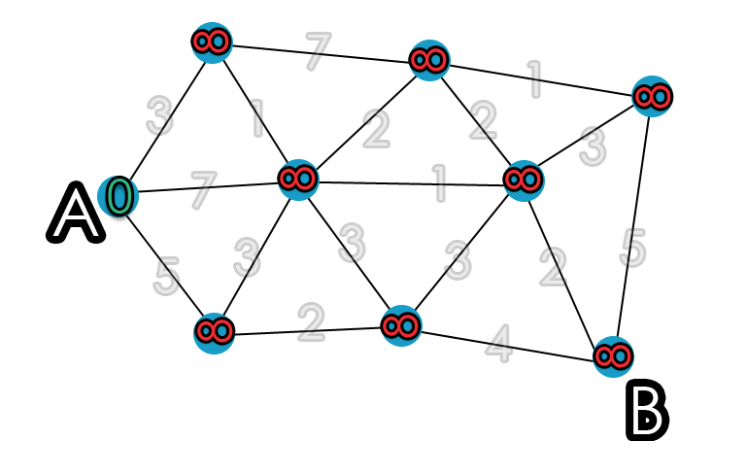
3) 动态数组不包括所有顶点

a) 选择一个不存于动态数组中的顶点，它具有与已在动态数组中的顶点的最短距离值。

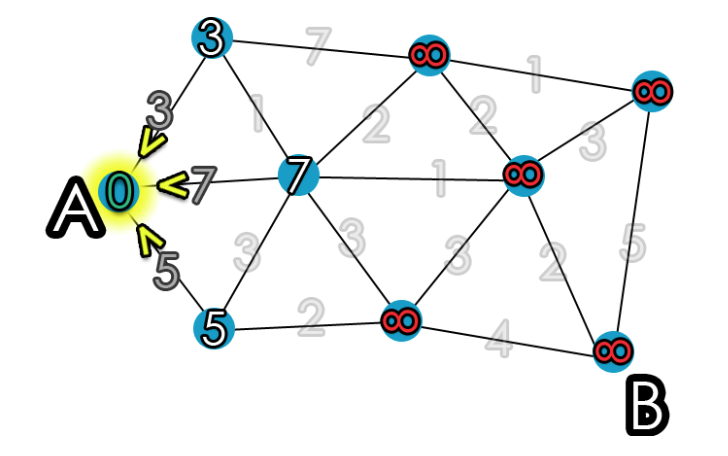
b) 将该顶点划入到动态数组中。

c) 更新与动态数组所有相邻顶点的距离值。要更新距离值，需要遍历所有相邻顶点。 对于每一个相邻的顶点v，如果u（距源点）的距离值和边uv的权重之和小于v的距离值，则更新顶点v的距离值。

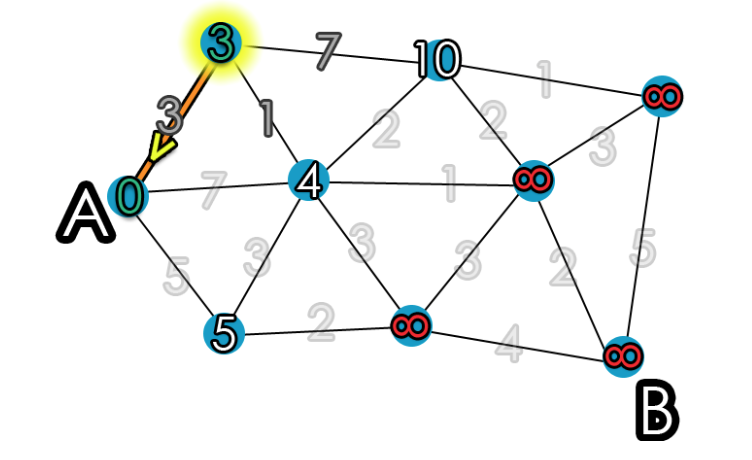
根据算法初始化距离。



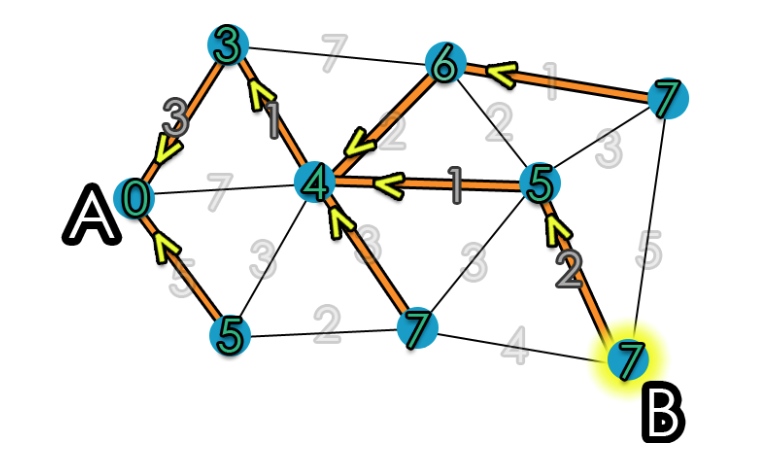
选择第一个节点并计算到相邻节点的距离。



选择距离最小的下一个节点； 重复相邻节点距离计算



最短路径树的最终结果



Bellman–Ford原理：

输入： 有向图和源顶点。

输出： 从源顶点到所有顶点的最短距离值。 如果存在负权重边，则不计算最短距离，返回存在负权重边。

1)将源顶点到所有顶点的距离初始化为无穷大，源顶点本身的距离初始化为0。创建一个大小为V的动态数组，除了其中源顶点值是0之外，其他所有值都为无穷大。

2)计算最短距离。 循环V-1次，其中V是给定图中的顶点数。

a) 对每条边uv，如果dist[v] > dist[u] +边uv的权重，则更新dist[v]，dist[v] = dist[u] +边 uv 的权重

3)循环中需要返回图中是否存在负权重边。对每条边 uv ，如果 dist[v] > dist[u] +边 uv 的权重，则返回“图包含负权重边”

第 3 步，如果图不包含负权重循环，则第 2 步保证得到了最短距离值。与其他动态规划问题一样，该算法以自底向上的方式计算最短路径。首先计算路径中最多有一条边的最短距离，然后，再计算最多有 2 条边的最短路径，依此类推。在外循环的第 i 次迭代之后，计算最多具有 i 条边的最短路径。假设没有负权重边的循环，如果我们计算了最多有i条边的最短路径，那么对所有边的迭代保证能得到最多有 (i+1) 条边的最短路径。

