05.1 - DAGs and Topological Ordering

Directed Acyclic Graphs（有向无环图）

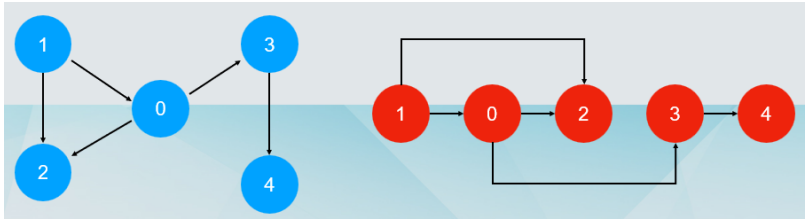
有向无环图就是一个有向但没有环的图

- 如果我们从一个 DAG G 中移除一个节点 u 以及其所有相关的边（移除所有 u 的入边与出边），得到的图 G' 仍然是一个 DAG
- 如果 G' 中存在一个环，那么同样的环也会存在于 G 中

这说明了DAG的一个重要性质，即移除一个节点及其关联边不会破坏图的无环性。这对于拓扑排序等算法的实现非常重要，因为这些算法通常通过逐步移除源节点来构建拓扑序列

Topological Ordering（拓扑排序）

给定一个有向图 G，拓扑排序是 G 的一种节点排列 $u_1, \ldots, u_n$，使得对于每个边 $e = (u_i, u_j)$，都有 $i < j$
换句话说，拓扑排序按某种顺序排列节点，使得所有边都“指向前方”



拓扑排序与 DAG

如果图 G 有一个拓扑排序，那么 G 是一个 DAG

证明:
假设 G 有一个拓扑排序 $u_1, \ldots, u_n$，且 G 也包含一个环 C
设 u_i 是环 C 中按拓扑排序最小的元素
设 u_j 是 u_i 在环中的前驱，即存在边 $e = (u_j, u_i)$
由于边 e 的存在，u_j 必须在拓扑排序中出现在 u_i 之前
这就意味着 u_i 不是 C 中按拓扑排序最小的元素，这与假设相矛盾
因此定理得证

如果 G 是一个DAG，那么 G 有一个拓扑排序

要证明 DAG 有一个拓扑排序，我们需要找到一个没有入度的节点
定义：一个 source node 是没有入度的节点
Lemma：每个 DAG 至少有一个 source node

反证法证明 lemma
假设 DAG 没有 source node
则每个节点都至少有一条入边
我们在 G^{rev} 中移动到 u 的一个邻居节点
至少在 $n+1$ 步后，我们将访问一个节点两次
这意味着图中至少存在一个环，因此这个图不可能是 DAG
这与假设矛盾

归纳法证明：如果 G 是一个DAG，那么 G 有一个拓扑排序
如果 DAG 只有两个节点，它显然有拓扑排序
我们假设一个有 k 个节点的 DAG 有拓扑排序
现在证明一个有 $k+1$ 个节点的 DAG 也有拓扑排序
根据 lemma，G 中至少有一个 source node，不妨设 u 为 source node
将 u 放在拓扑排序的第一位，移除 u 以及其相关边得到图 G'
G' 是一个拥有 k 个节点的 DAG，则 G' 有一个拓扑排序
那么我们将 G 复原，显然 G 有一个拓扑排序

上面的归纳法证明其实就是通过算法证明，因为我们可以将归纳法正面转换为一个算法：

```
Algorithm TopologicalSort(G)
    Find a source vertex u and put it first in the order.
    Let G' = G - {u}
    TopologicalSort(G')
    Append this order after u
```

进行拓扑排序的运行时间

朴素的算法

我们需要找到 source node
这需要在每次迭代中检查所有的节点：

- 第一次：检查 n 个节点
- 第二次：检查 $n-1$ 个节点
- ……

因此，运行时间是 $O(n^2)$

一个更快的算法

如果一个节点没有被算法选中（并因此被移除）作为 source node，我们就可以说该节点处于 active
我们需要维护：

- （a）对于每个节点 w，记录从其他活跃节点来的入边数
- （b）集合 S 包含所有没有来自其他活跃节点的入边的活跃节点

算法如下：

1. 一开始，所有节点都是活跃的。通过遍历图初始化上面的 a、b 步骤，复杂度为 $O(n+m)$
2. 每次迭代：
    1. 从 S 中选择一个节点 u
    2. 从图中删除节点 u
    3. 遍历节点 u 的所有邻居节点 w，将 (a) 中的值减少 1
    4. 如果某个节点 w 的 (a) 值变为 0，则将 w 添加到集合 S 中

这个算法相比于原始算法的优化之处在于它避免了在每次迭代中重新遍历整个图来寻找 source node
S 是一个用来维护的集合，因此找到下一个 source node 的时间是 $O(1)$

除每个节点时只处理其直接邻居，因此减少 (a) 的时间是 $O(1)$
因此总体上只需要 $O(n+m)$