**实验四**

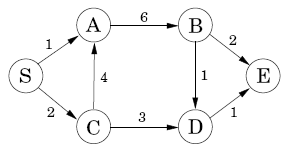
**实验目的与要求**：掌握动态规划方法的基本思想与设计策略。

**1．多段图中的最短路径问题**

建立一个从源点S到终点T的多段图，设计一个动态规划算法求出从S到T的最短路径值，并输出相应的最短路径。

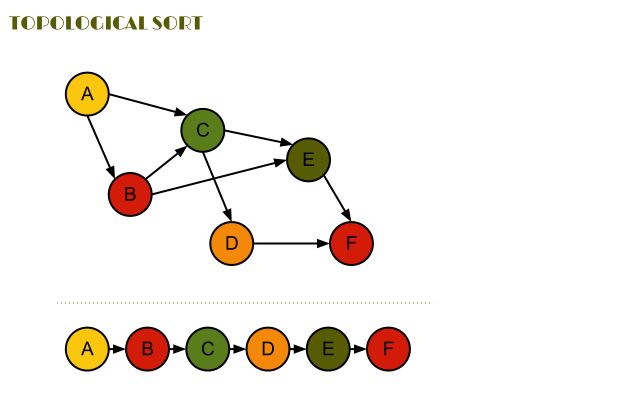
**2．有向无环图中的最短路径问题**

建立一个从源点S到终点E的有向无环图，设计一个动态规划算法求出从S到E的最短路径值，并输出相应的最短路径。



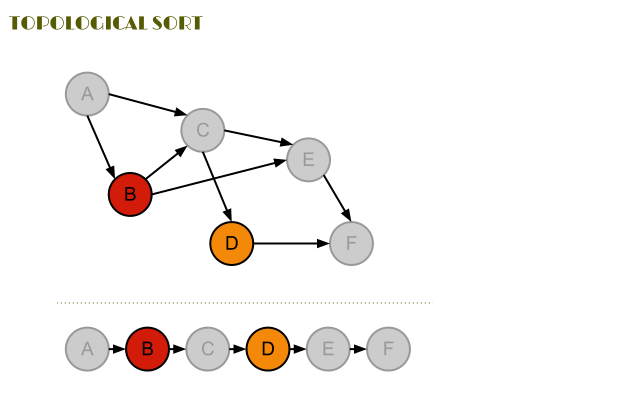
关于有向无环图（DAG）我们首先要知道，它们可以很容易地进行拓扑排序。拓扑排序应用范围非常之广，其中最常用的或许就是用于安排依赖任务（依赖任务是同属于一个工作中相同任务的实体，这些实体是保证互连的，它们解决共同的问题）。

下图即为对一个有向图的拓扑排序：



拓扑排序通常是用来“排序”依赖任务的！

经过拓扑排序，我们最终会得到一张DAG图的顶点列表，我们确信在拓扑排序列表中如果存在一条边，那么顶点会先于顶点进入列表中。



如果有一条边，那么顶点一定在顶点前面。这个结果通过这张图片变得更加通俗易懂。其中B和D之间没有边，但在列表中B在D前面！

此信息异常重要，我们唯一需要做的事情就是通过这个排序列表来计算距离最短的路径，这和Dijkstra算法比较相似。

好了，让我们来总结一下这个算法：

1.首先，我们必须对有向无环图进行拓扑排序；

2.其次，我们将到源点的距离初始化为0并将到其它所有顶点的距离设置为无穷大；

3.最后，对于列表中的每一个顶点，我们从它的所有邻节点中找到最短路径的那个顶点。

这很像Dijkstra算法，但是其主要区别是我们使用的是经过拓扑排序的列表。

**时间复杂度**

拓扑排序的时间复杂度是。找到拓扑顺序后，算法依次处理所有顶和其相邻顶点的顶点，总相邻顶点的个数是。因此，内循环运行。所以，这个算法的总体时间复杂度是。

