课时15

图的实现方式与核心算法

1. 图的实现方式
2. 图的拓扑排序
3. 拓扑排序的实现方式
4. 图的最短路径
5. 总结

图的实现方式

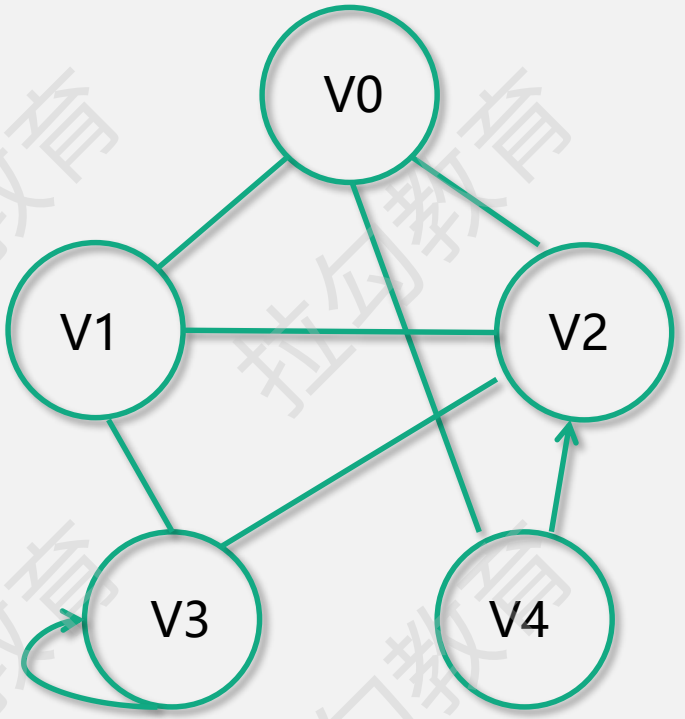
拉勾
LAGOU.COM

邻接矩阵法

邻接链表法

图的实现方式

邻接矩阵法 基本思想是开一个超大的数组



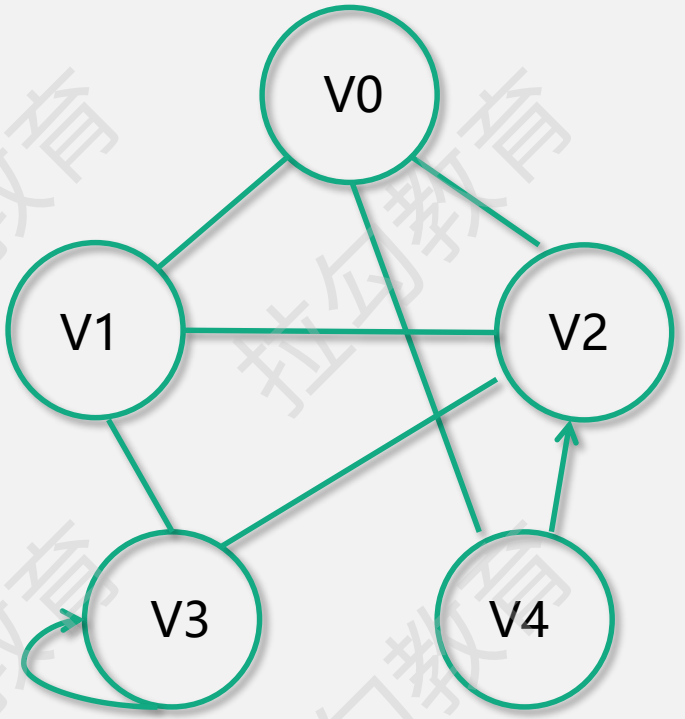
destination

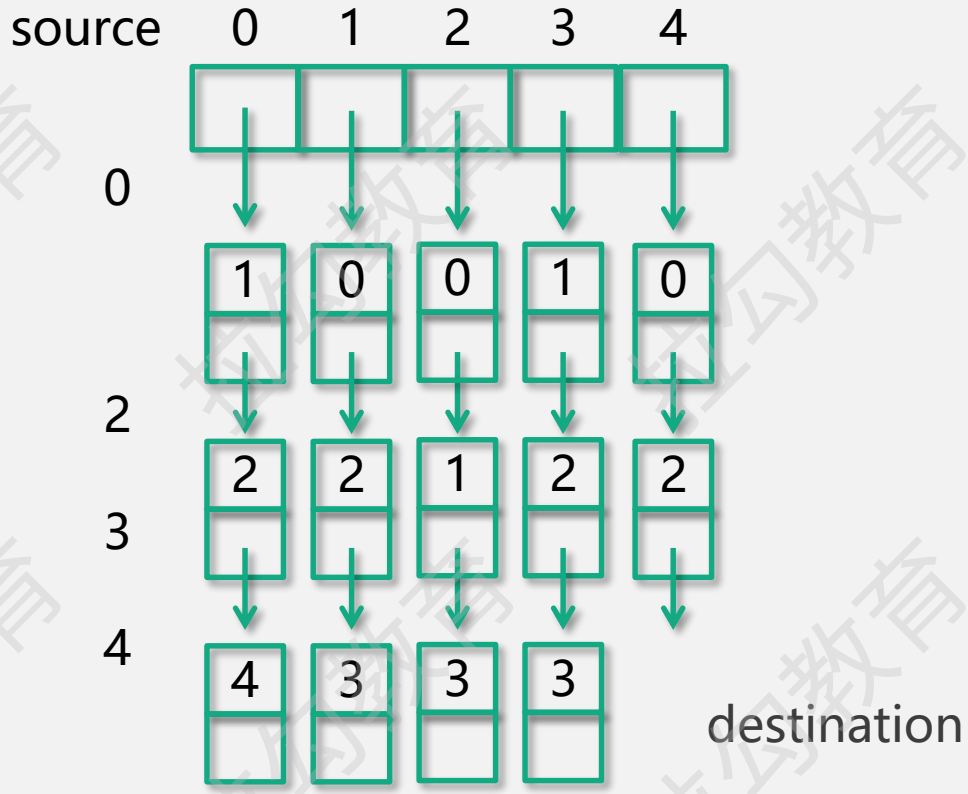
source

	0	1	2	3	4
0					
1					
2					
3					
4					

图的实现方式

邻接链表法 核心思想是把每一个节点所指向的点存储起来





图的拓扑排序

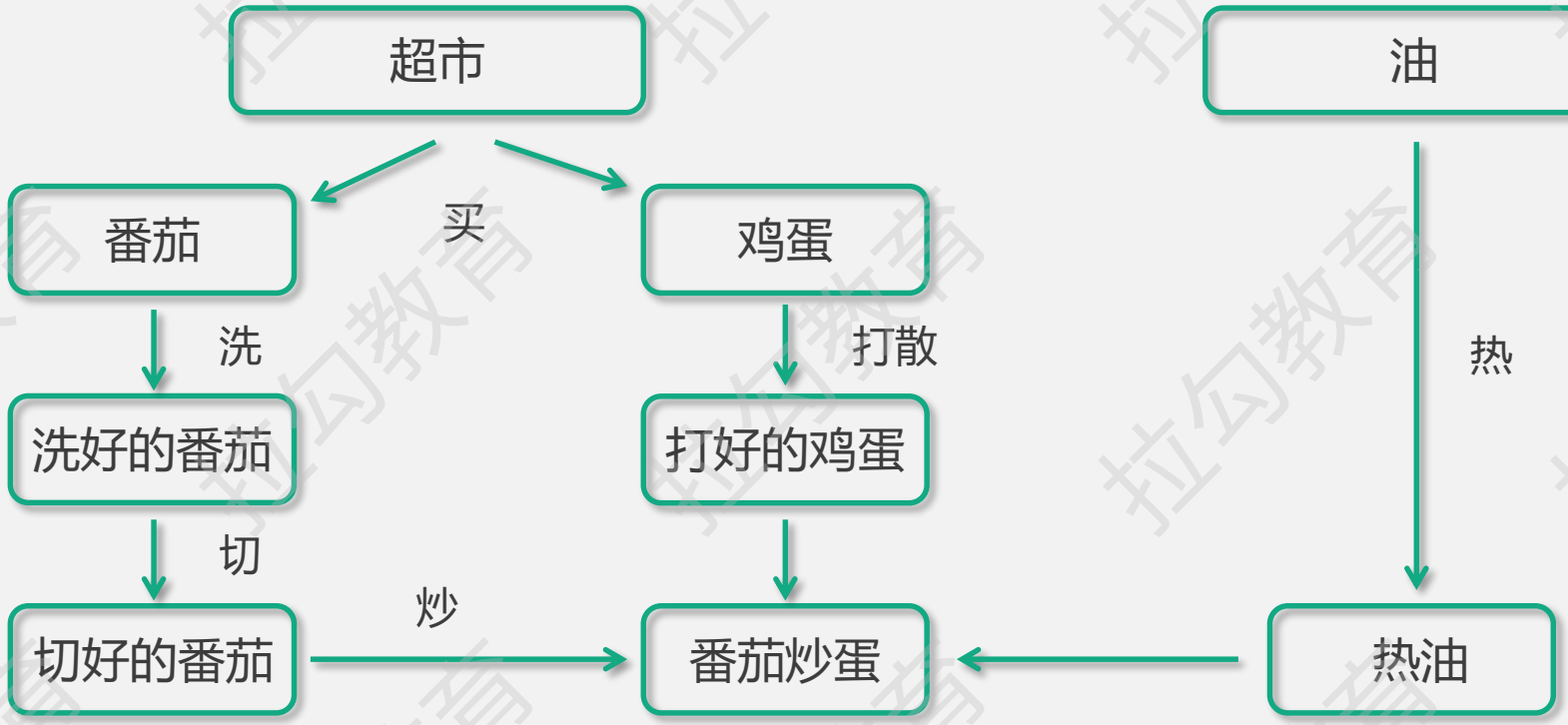
拉勾
LAGOU.COM

什么是拓扑排序呢

指的是对于一个有向无环图来说，排序所有的节点

使得对于从节点 u 到节点 v 的每个有向边 uv，u 在排序中都在 v 之前

图的拓扑排序

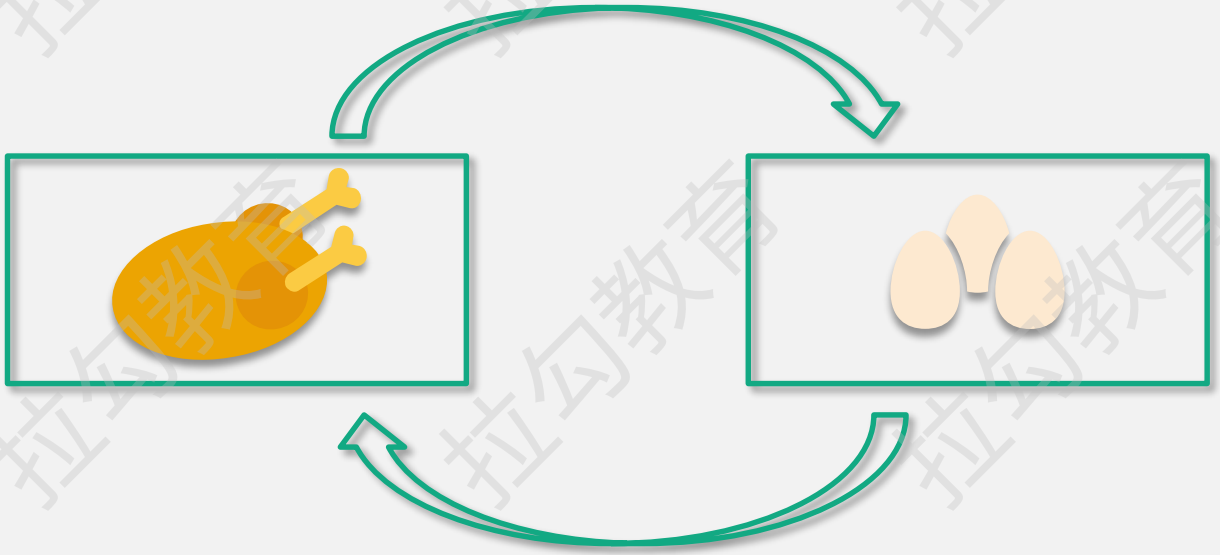


一个合法的拓扑排序必须使得被依赖的任务首先完成

图的拓扑排序

为什么拓扑排序只适用于有向无环图呢？

图的拓扑排序

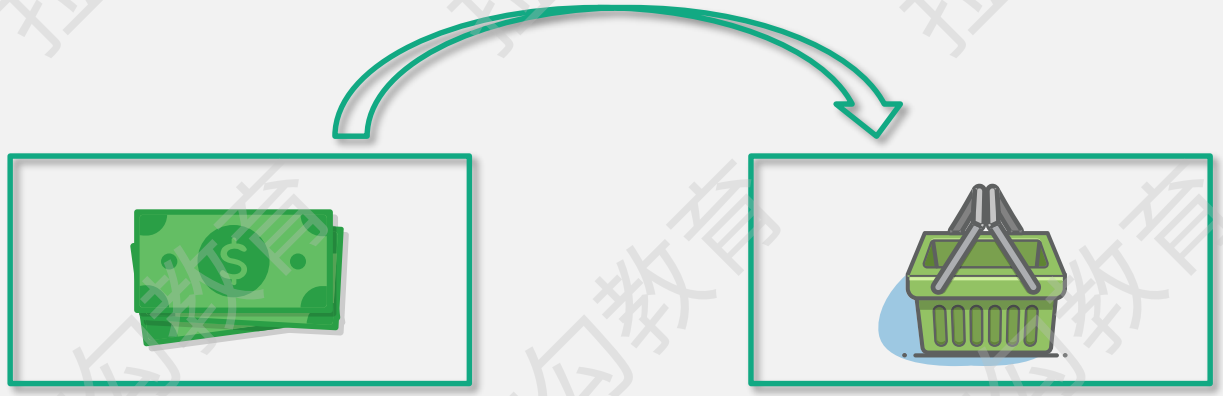


先有鸡还是先有蛋 （Chicken Egg Dilemma）是一个无法被拓扑排序的有环图

因为鸡依赖于蛋，蛋又依赖于鸡

你无法把鸡排在蛋前面，也不能把蛋排在鸡的前面

图的拓扑排序



这里面隐含的一个有向无环图就是钱指向了你想做的事情

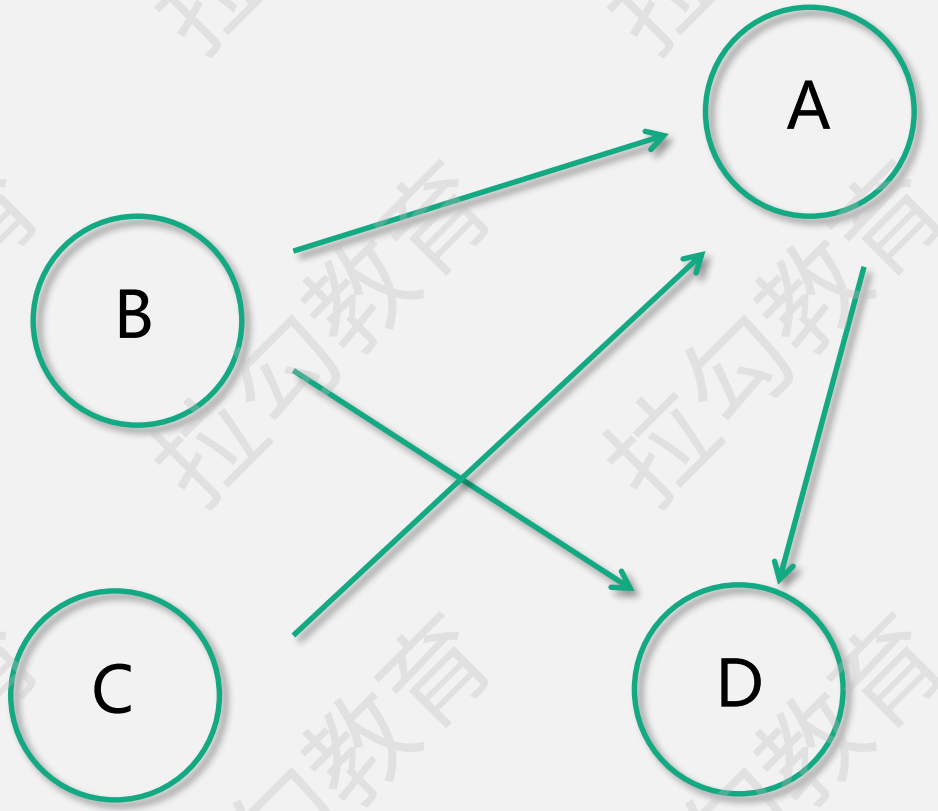
那么我们就很容易的得出一个合理的拓扑排序，要把钱排在你想做的事情之前

拓扑排序的实现方式

拉勾 LAGOU.COM

有向图的入度 指的是终止于一个节点的边的数量

有向图的出度 指的是始于一个节点的边的数量



拓扑排序的实现方式

拉勾 LAGOU.COM

卡恩于 1962 年提出的算法，其实是贪婪算法的一种形式

简单来说就是：

- 假设 L 是存放结果的列表，我们先找到那些入度为零的节点
  把这些节点放到 L 中，因为这些节点没有任何的父节点
- 然后把与这些节点相连的边从图中去掉，再寻找图中入度为零的节点
- 对于新找到的这些入度为零的节点来说
  他们的父节点都已经在 L 中了，所以也可以放入 L

重复上述操作，直到找不到入度为零的节点

- 如果此时 L 中的元素个数和节点总数相同，则说明排序完成
- 如果 L 中的元素个数和节点总数不同，则说明原图中存在环，无法进行拓扑排序

拓扑排序的实现方式

拉勾 LAGOU.COM

卡恩算法

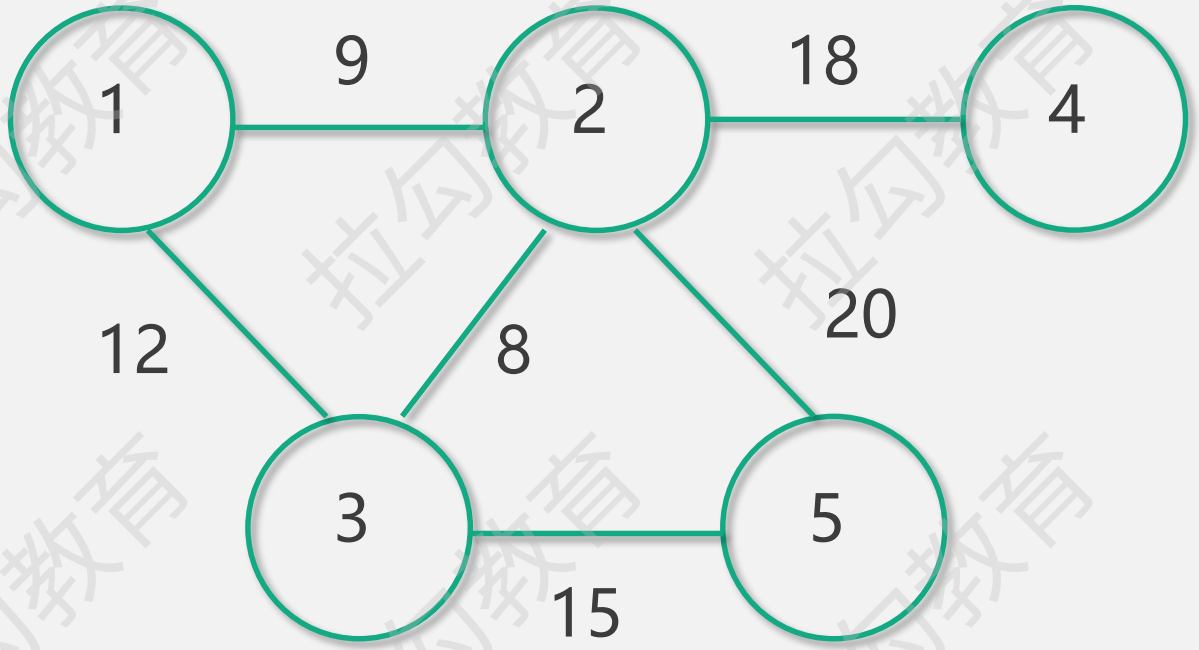
```
L ← Empty list that will contain the sorted
elements
S ← Set of all nodes with no incoming edge

while S is non-empty do
    remove a node n from S
    add n to tail of L
    for each node m with an edge e from n to m do
        remove edge e from the graph
        if m has no other incoming edges then
            insert m into S

if graph has edges then
    return error   (graph has at least one cycle)
else
    return L   (a topologically sorted order)
```

图的最短路径

在一个有权重的图中，找到两个点之间权重之和最短的路径



图的最短路径

拉勾
LAGOU.COM

怎样让计算机找到最短路径呢

便是大名鼎鼎的 Dijkstra 算法

图的最短路径

拉勾
LAGOU.COM

最经典的 Dijkstra 算法原始版本仅适用于找到两个固定节点之间的最短路径

后来更常见的变体固定了一个节点作为源节点

然后找到该节点到图中所有其他节点的最短路径从而产生一个最短路径树

图的最短路径

这个算法是通过为每个节点 v 保留当前为止所找到的从 s 到 v 的最短路径来工作的

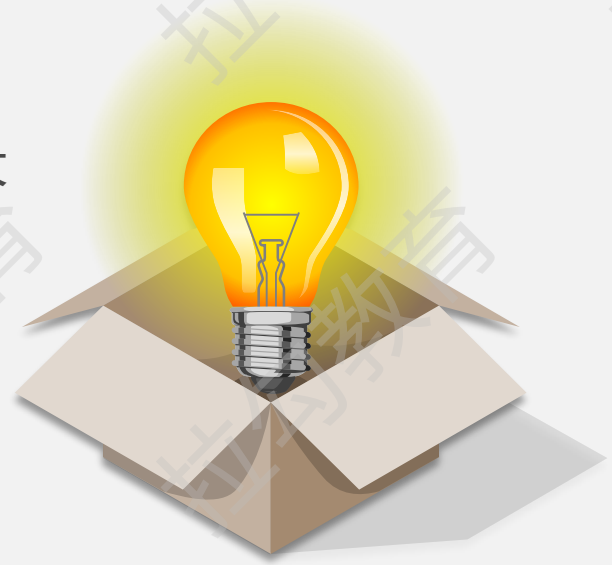
初始时：

原点 s 的路径权重被赋为 0（即原点的实际最短路径 = 0）

同时把所有其他节点的路径长度设为无穷大，即表示我们不知道任何通向这些节点的路径

结束时：

d[v] 中存储的便是从 s 到 v 的最短路径，或者如果路径不存在的话，则是无穷大



图的最短路径

拉勾 LAGOU.COM

Dijkstra 算法

```
function Dijkstra(Graph, source):

    create vertex set Q

    for each vertex v in Graph:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist[source] ? 0

    while Q is not empty:
        u ← vertex in Q with min dist[u]

        remove u from Q
```

图的最短路径

Dijkstra 算法

```
    add v to Q
dist[source] ? 0

while Q is not empty:
    u ← vertex in Q with min dist[u]

    remove u from Q

    for each neighbor v of u:          // only v that are still in Q
        alt ? dist[u] + length(u, v)
        if alt < dist[v]:
            dist[v] ← alt
            prev[v] ← u

return dist[], prev[]
```

总结

拉勾
LAGOU.COM

图的拓扑排序

揭开了 Spark 最核心的机密算法

图的最短路径算法