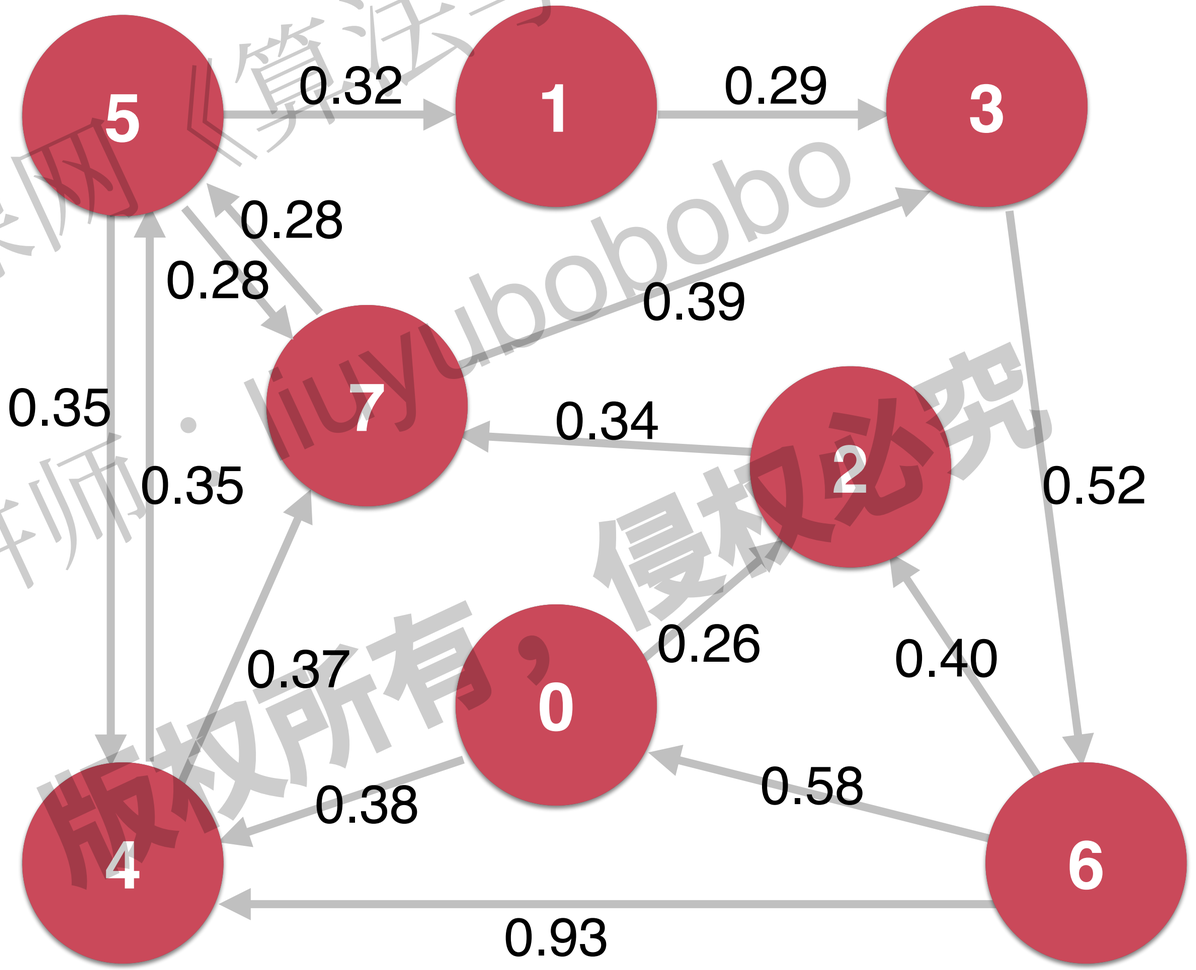
算法与数据结构

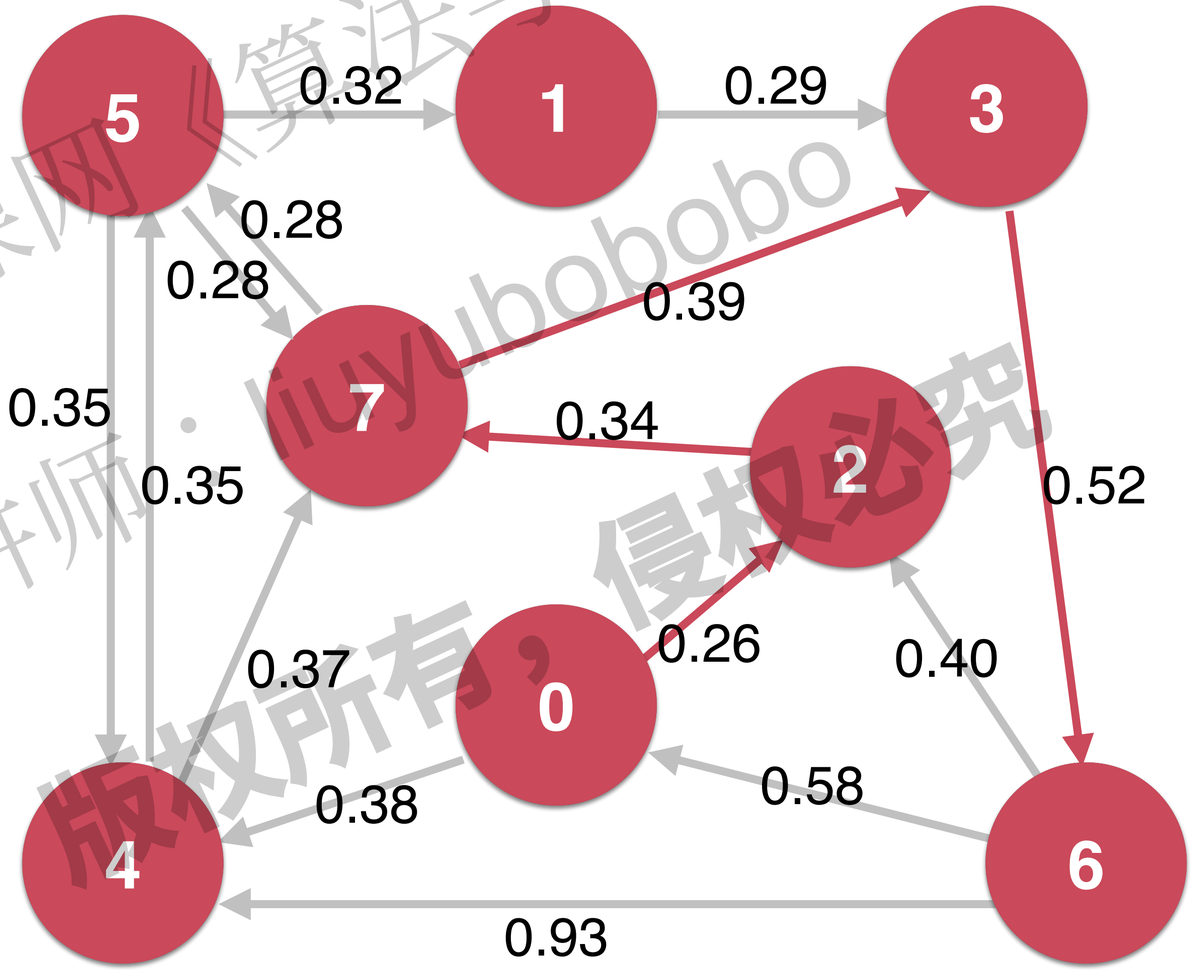
liuyubobobo

最路径问题 Shortest Path

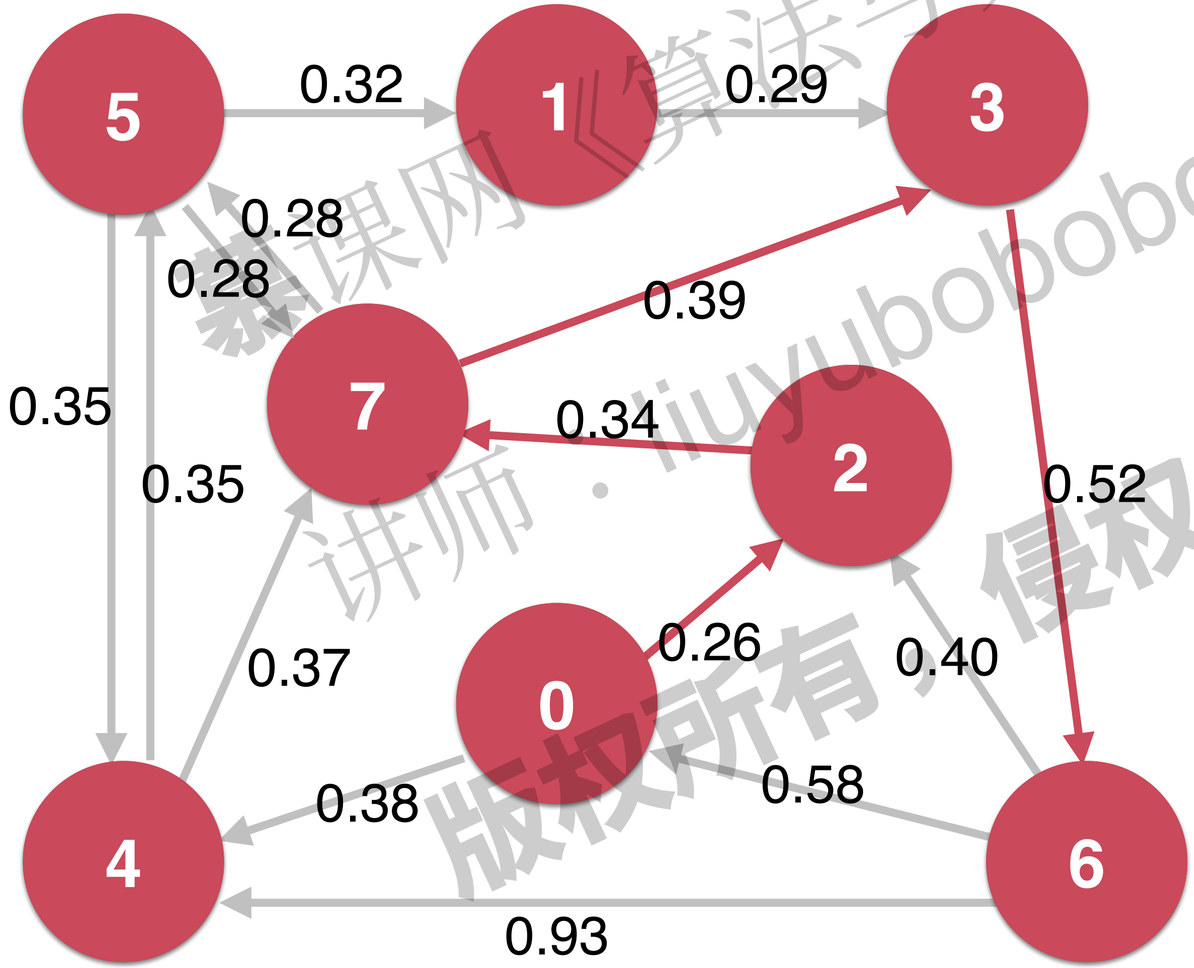
最短路径问题 Shortest Path



最短路径问题 Shortest Path



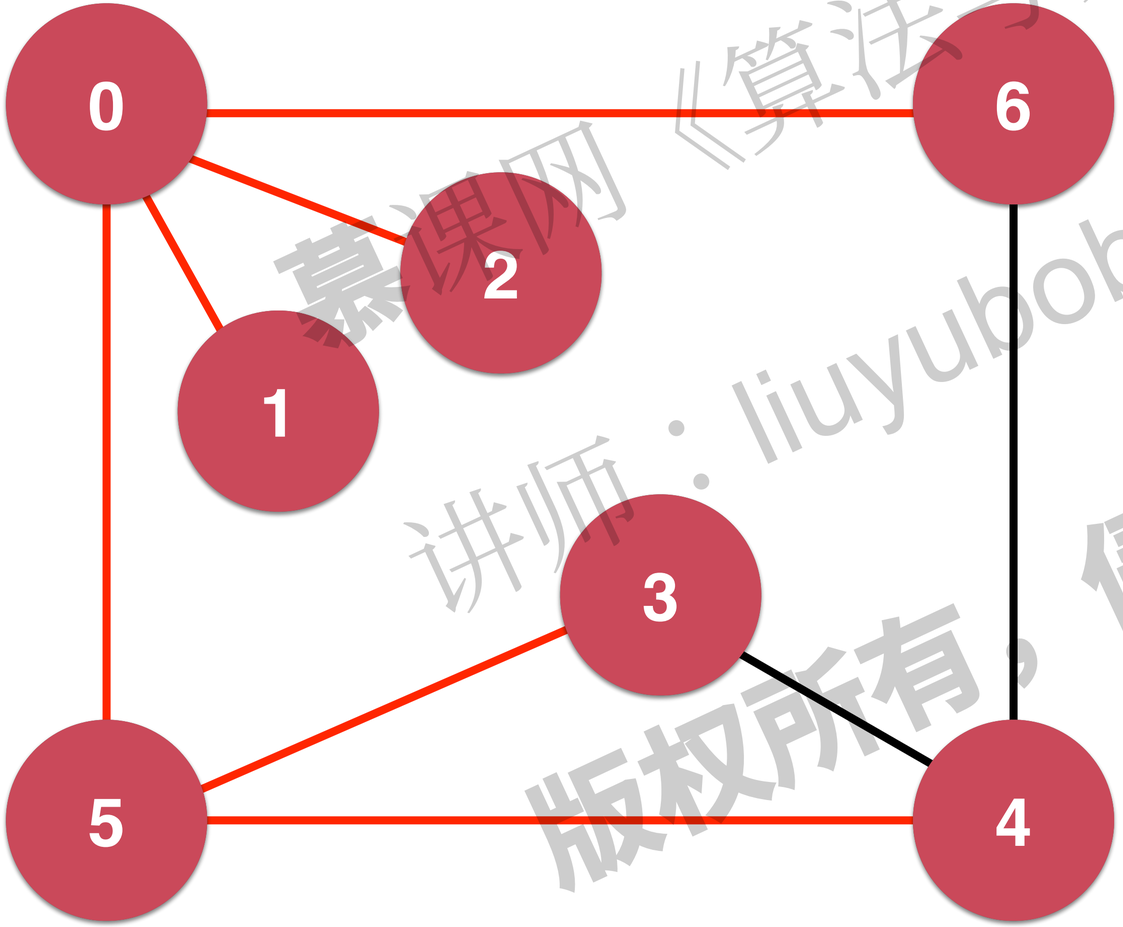
最短路径问题 Shortest Path



路径规划

工作任务规划

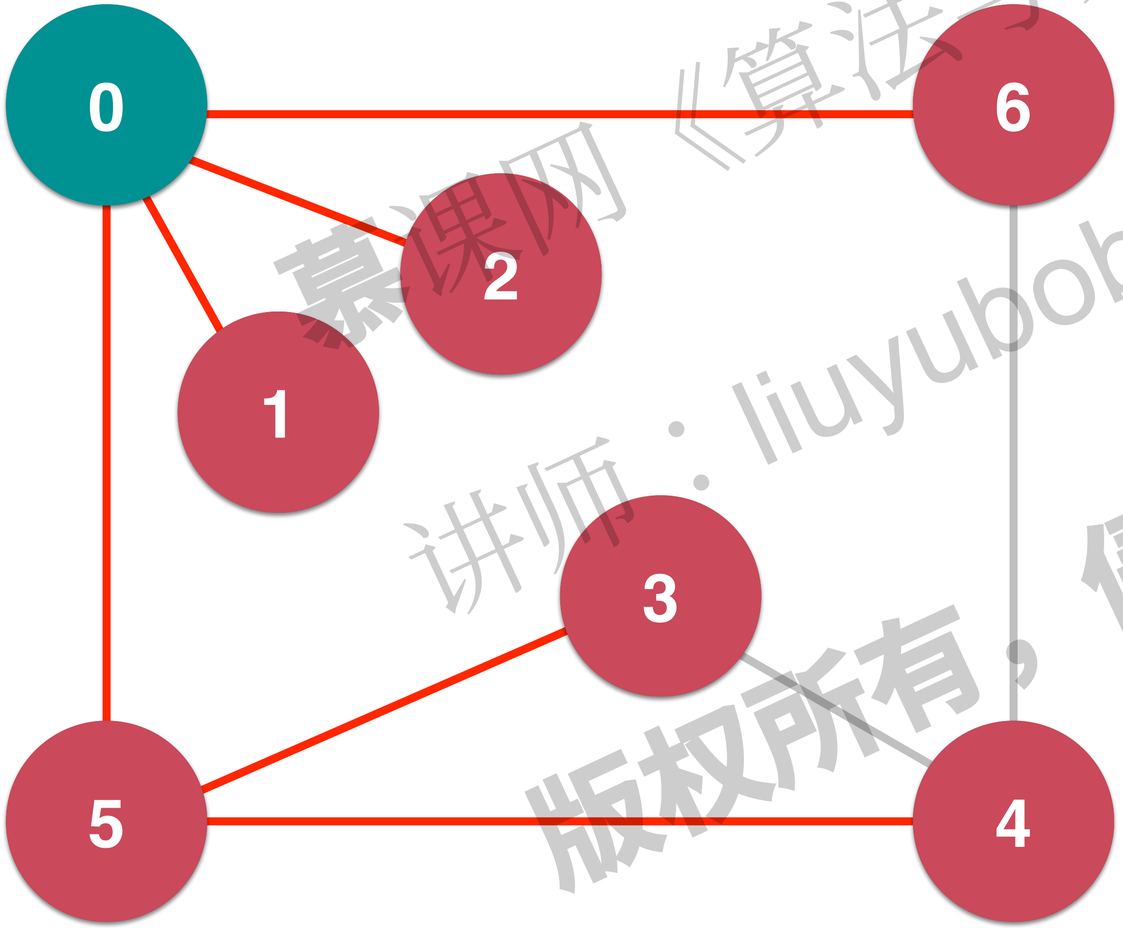
广度优先遍历



0	1	2	5	6
1	0			
2	0			
3	4	5		
4	3	5	6	
5	0	3	4	
6	0	4		

0 1 2 5 6 3 4

广度优先遍历



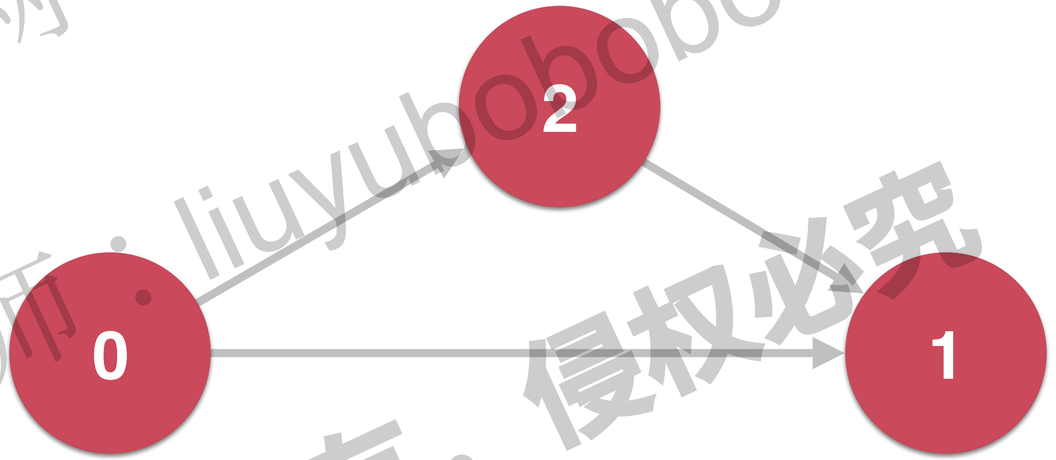
最短路径树

Shortest Path Tree

单源最短路径

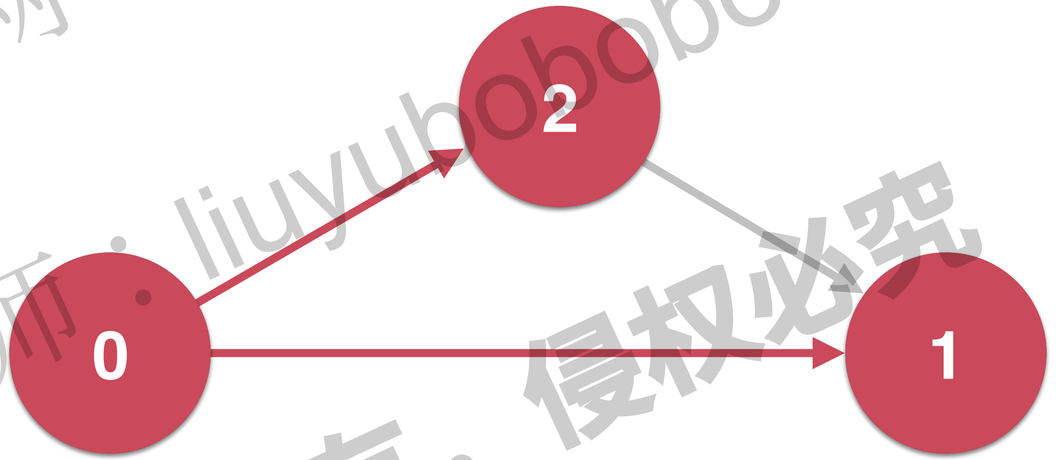
Single Source Shortest Path

无权图的最短路径



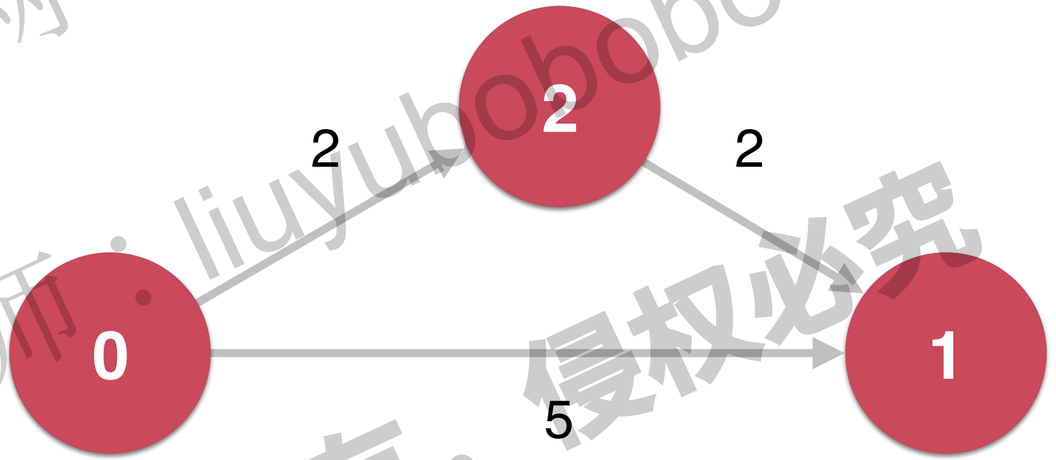
0	0
1	-
2	-

无权图的最短路径



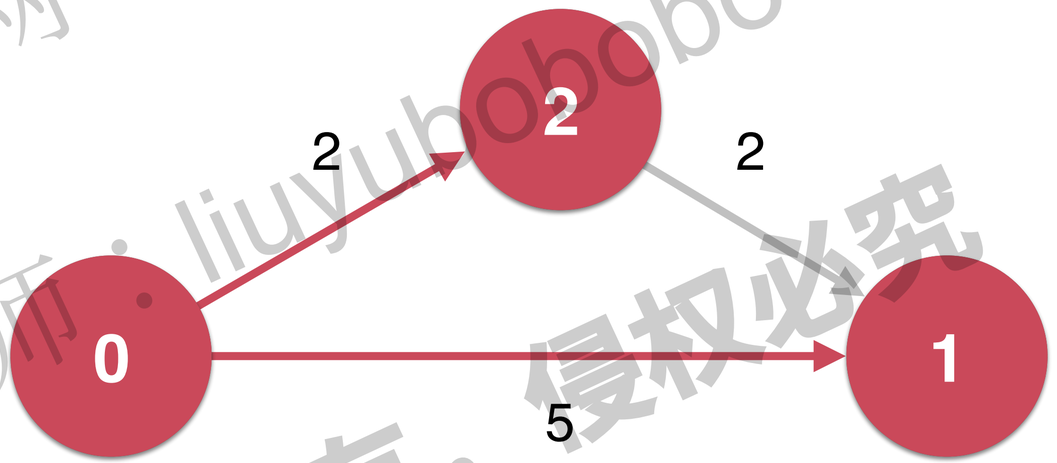
0	0
1	1
2	1

有权图的最短路径



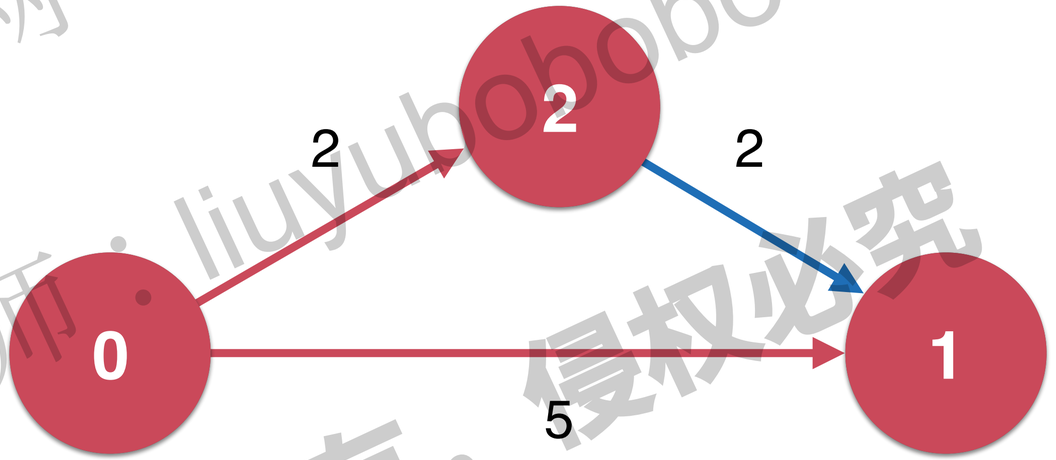
0	0
1	-
2	-

有权图的最短路径



0	0
1	5
2	2

有权图的最短路径



0	0
1	4
2	2

松弛操作 Relaxation

松弛操作是最短路径求解的核心

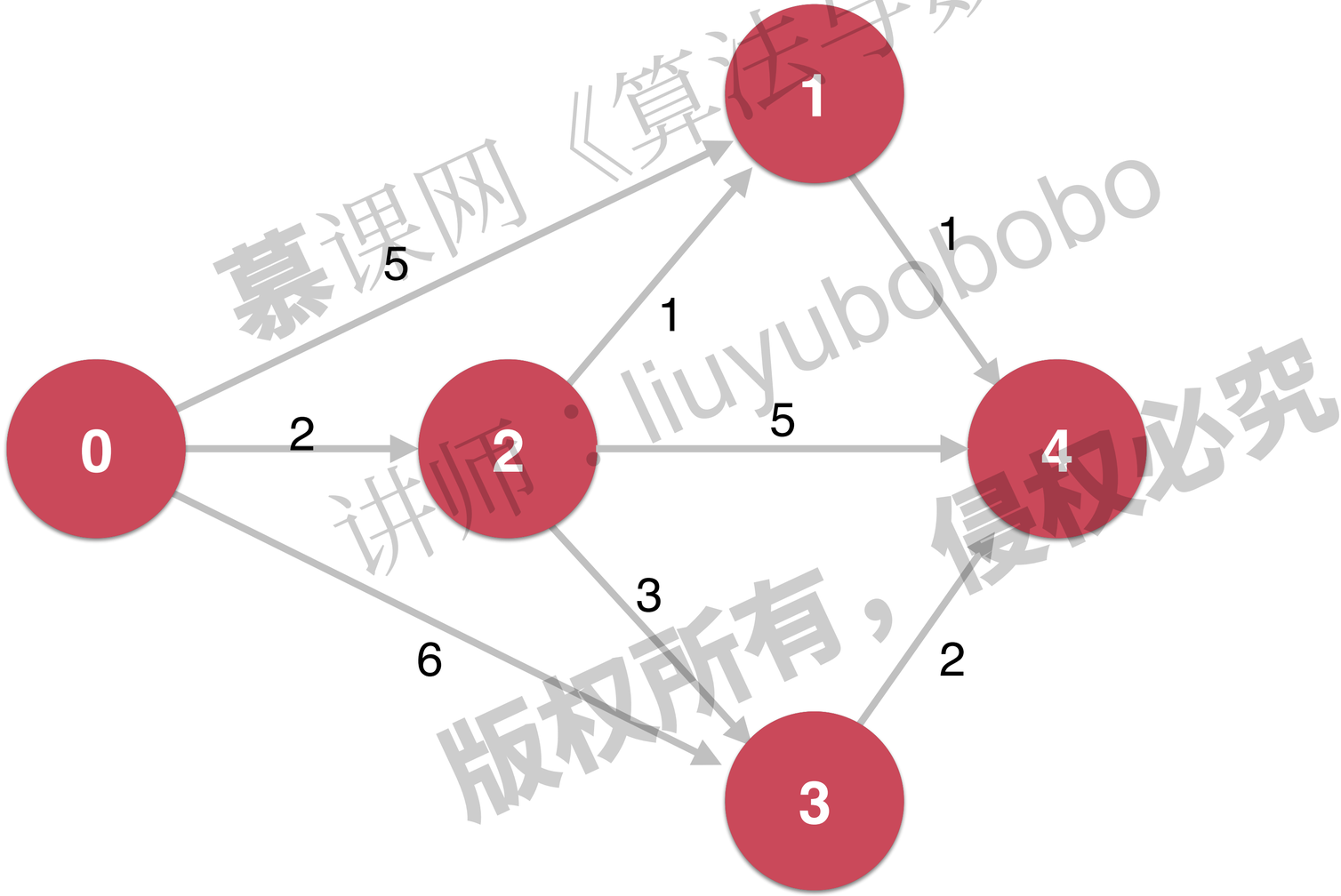
dijkstra 单源最短路径算法

dijkstra 单源最短路径算法

前提：图中不能有负权边

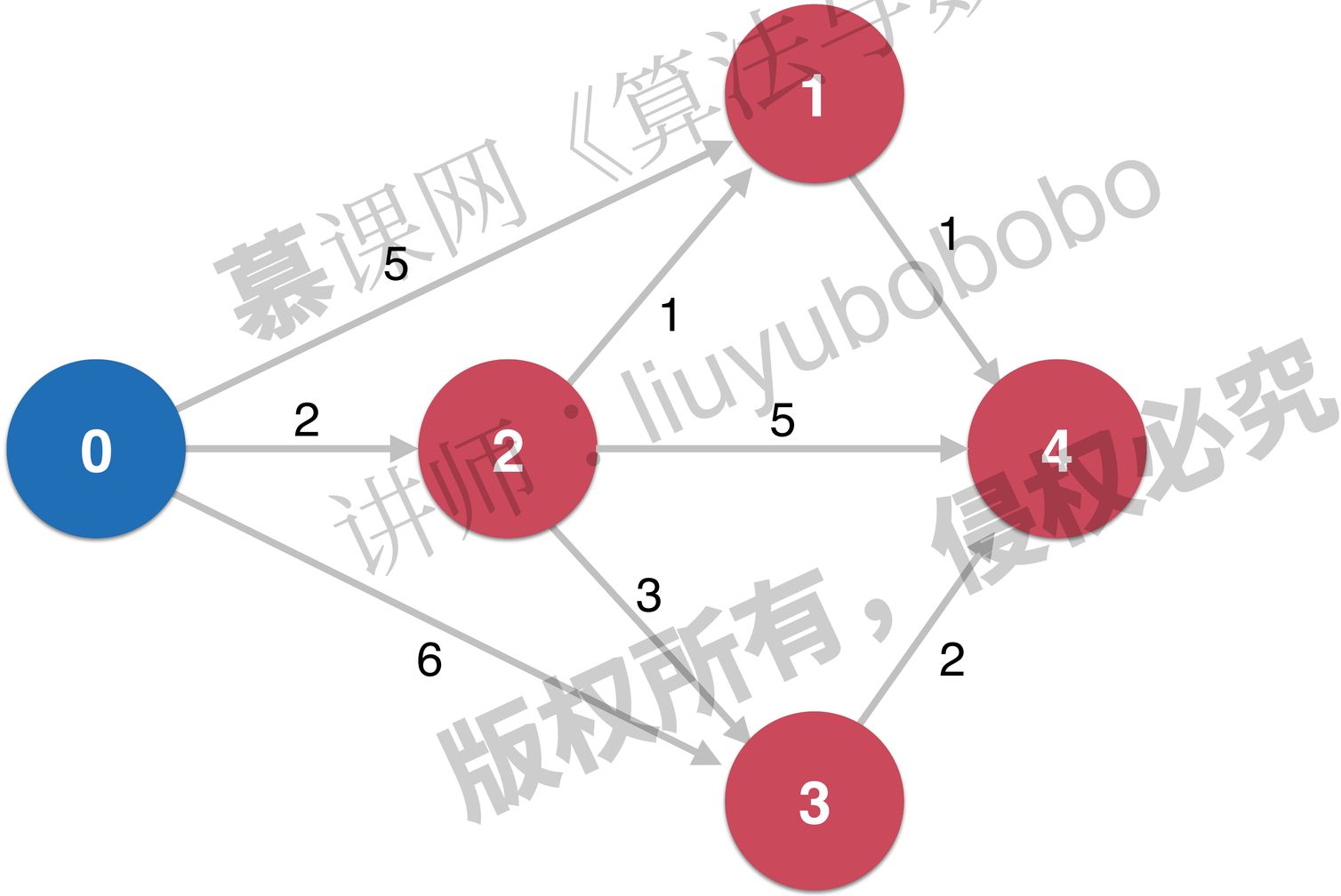
复杂度 O(E log(V))

dijkstra 单源最短路径算法



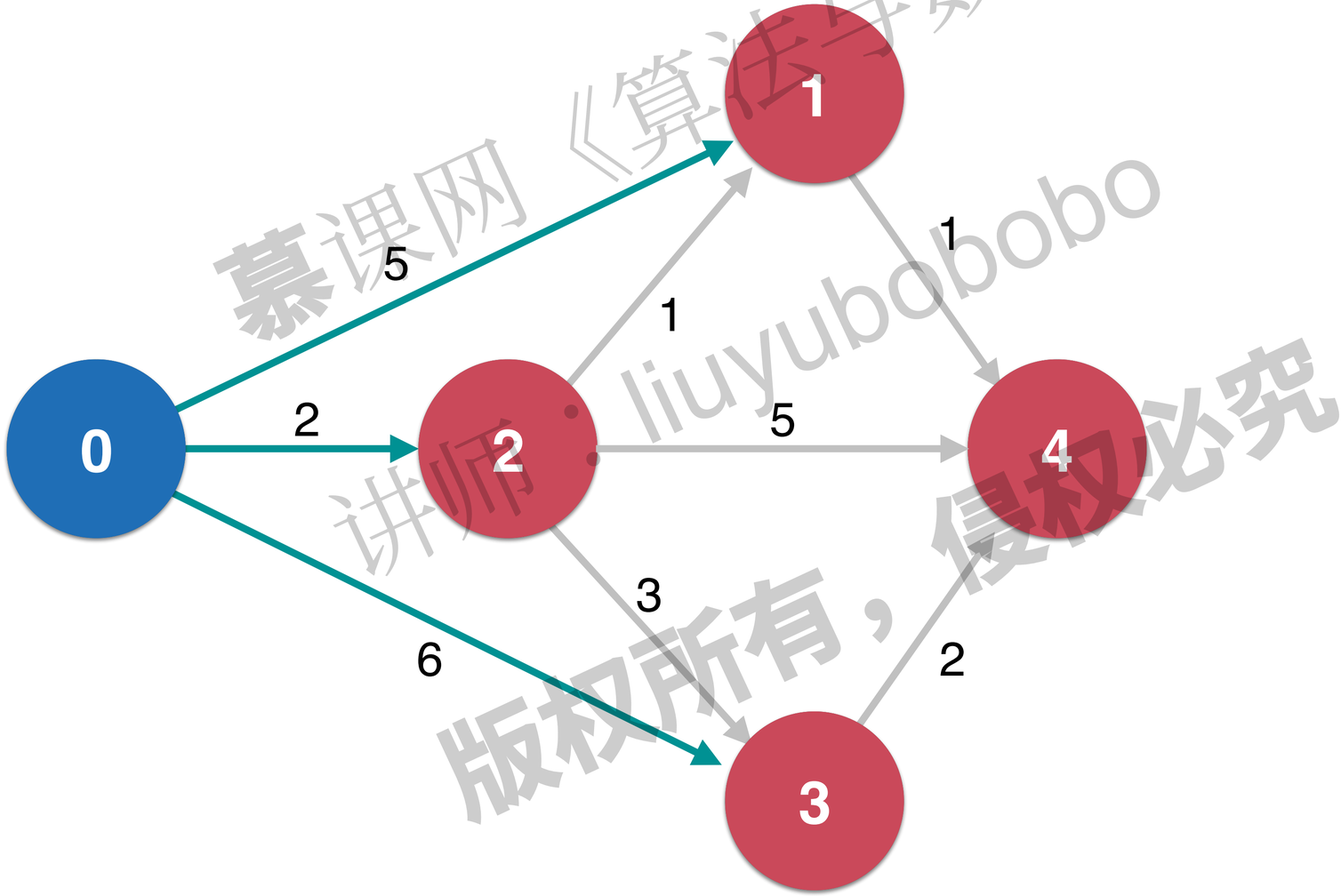
0	0
1	-
2	-
3	-
4	-

dijkstra 单源最短路径算法



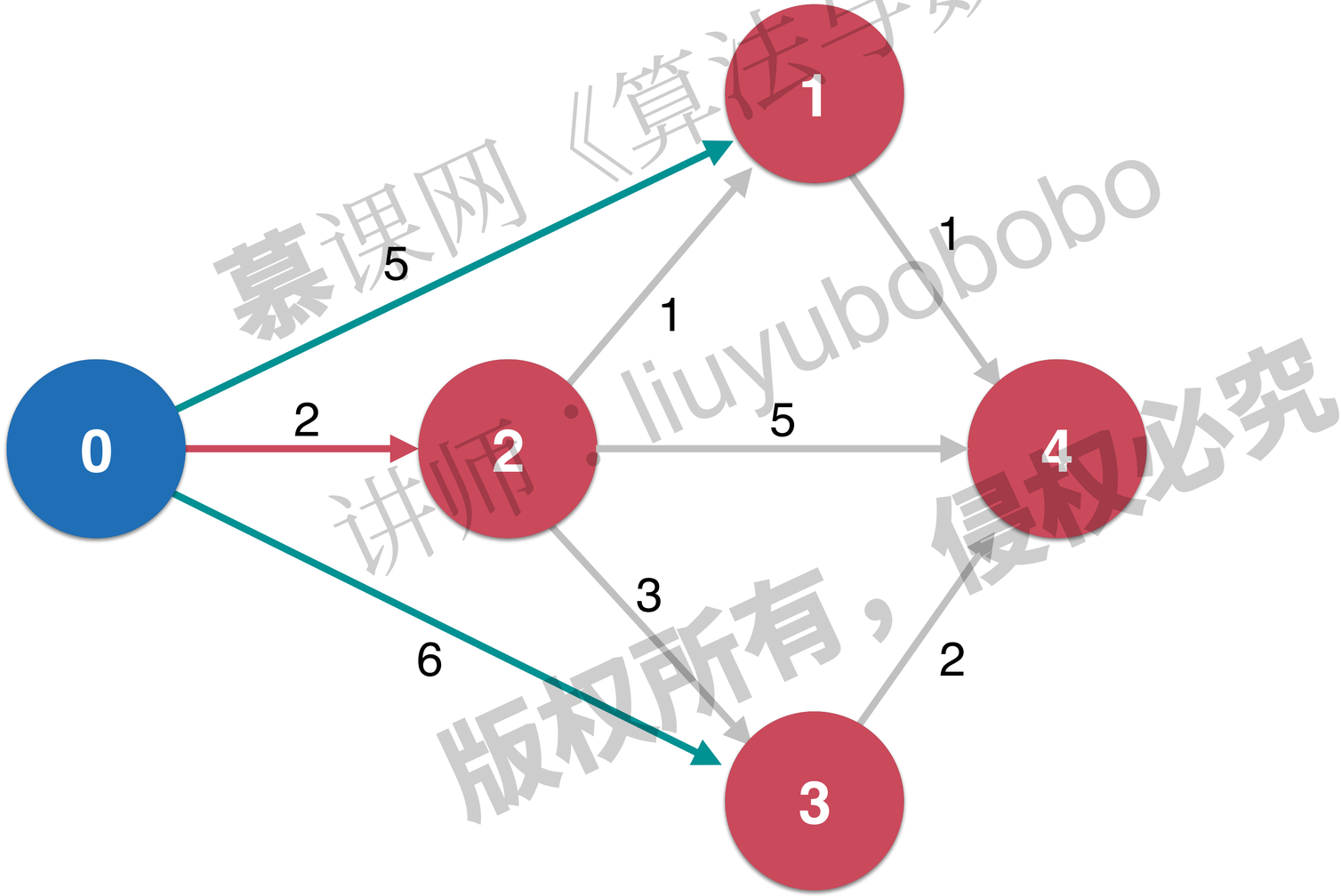
0	0
1	-
2	-
3	-
4	-

dijkstra 单源最短路径算法



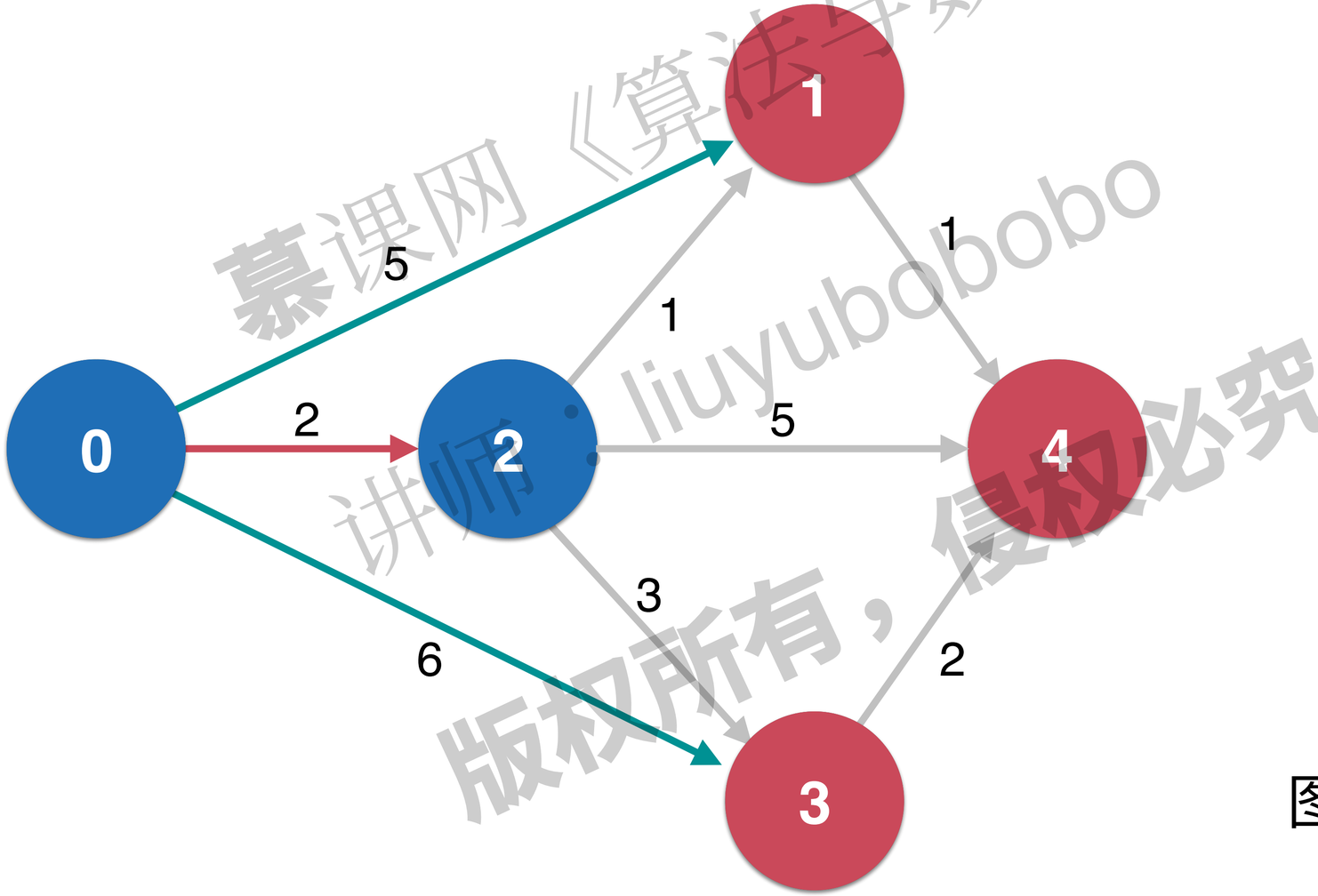
0	0
1	5
2	2
3	6
4	-

dijkstra 单源最短路径算法



0	0
1	5
2	2
3	6
4	-

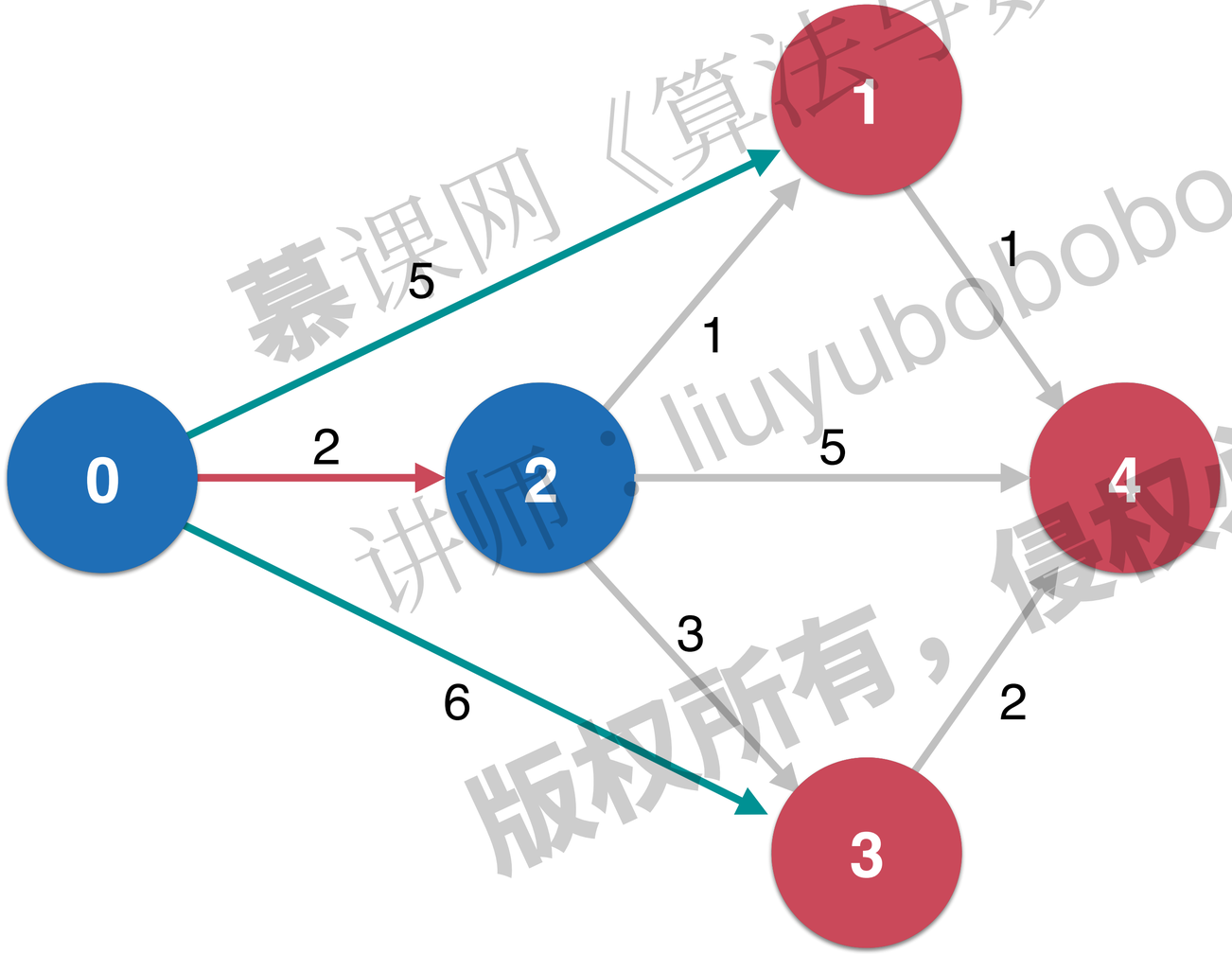
dijkstra 单源最短路径算法



0	0
1	5
2	2
3	6
4	-

图中不能有负权边

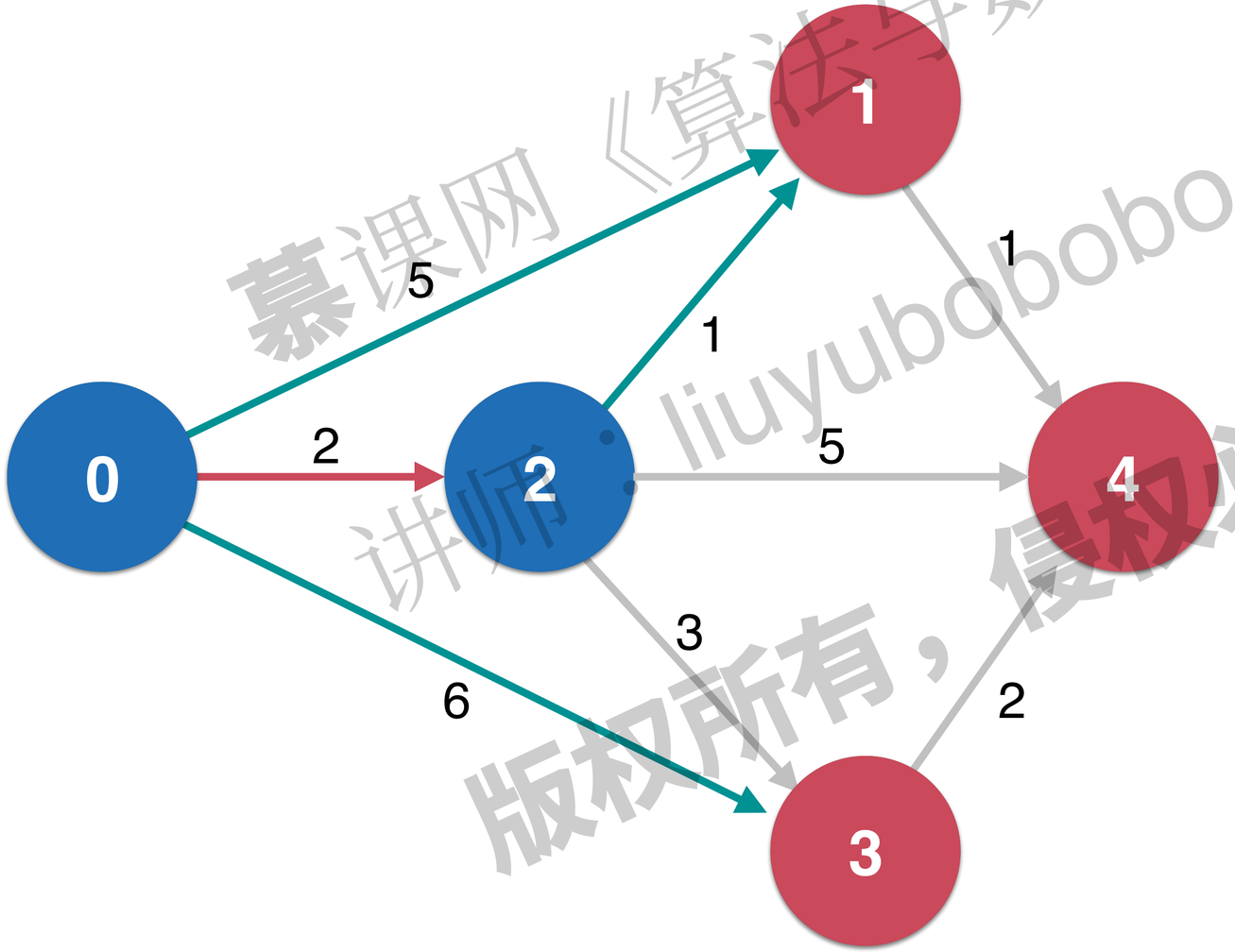
dijkstra 单源最短路径算法



确定新的节点的最短路径后，
进行Relaxation

0	0
1	5
2	2
3	6
4	-

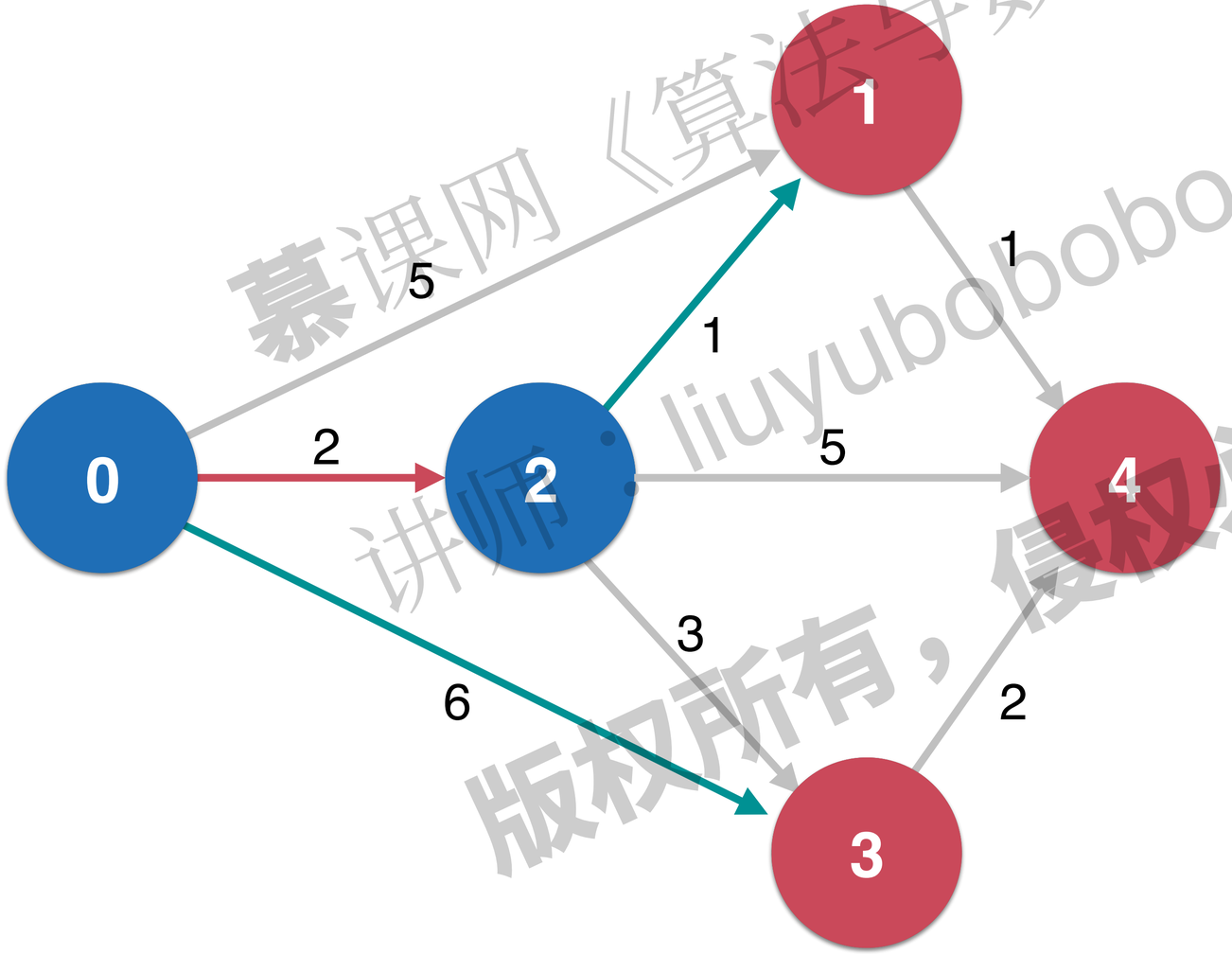
dijkstra 单源最短路径算法



确定新的节点的最短路径后，
进行Relaxation

0	0
1	5
2	2
3	6
4	-

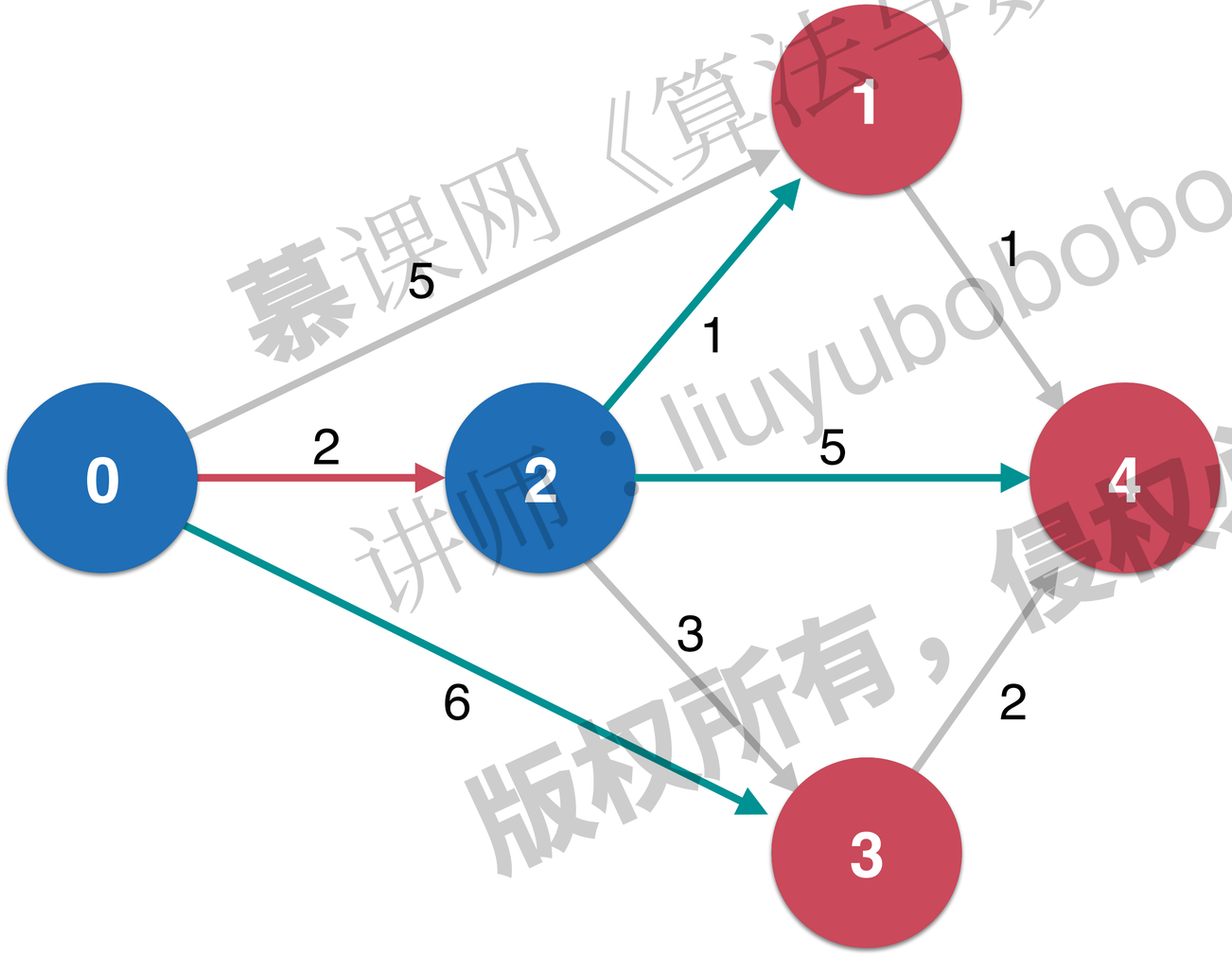
dijkstra 单源最短路径算法



确定新的节点的最短路径后，
进行Relaxation

0	0
1	3
2	2
3	6
4	-

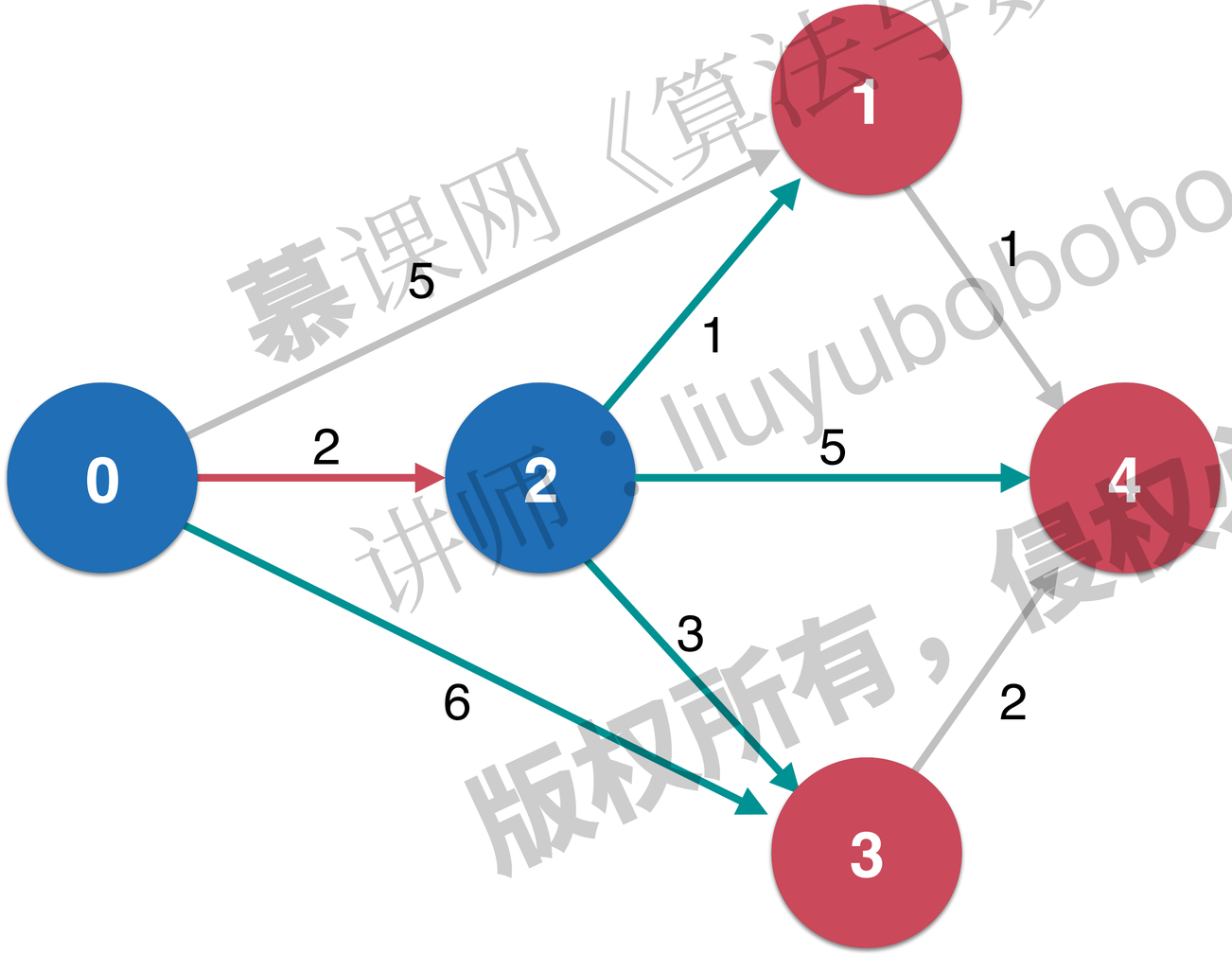
dijkstra 单源最短路径算法



确定新的节点的最短路径后，
进行Relaxation

0	0
1	3
2	2
3	6
4	7

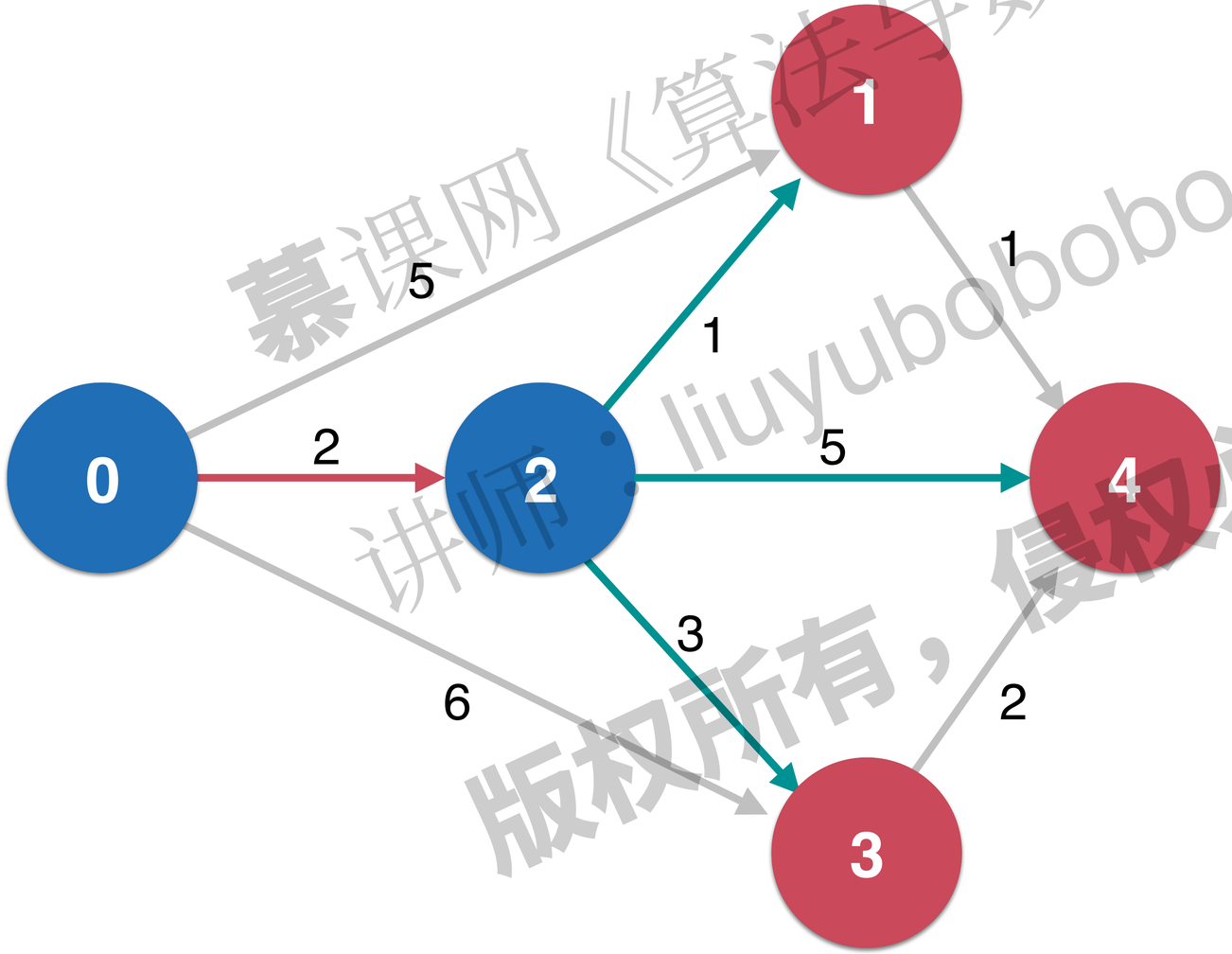
dijkstra 单源最短路径算法



确定新的节点的最短路径后，
进行Relaxation

0	0
1	3
2	2
3	6
4	7

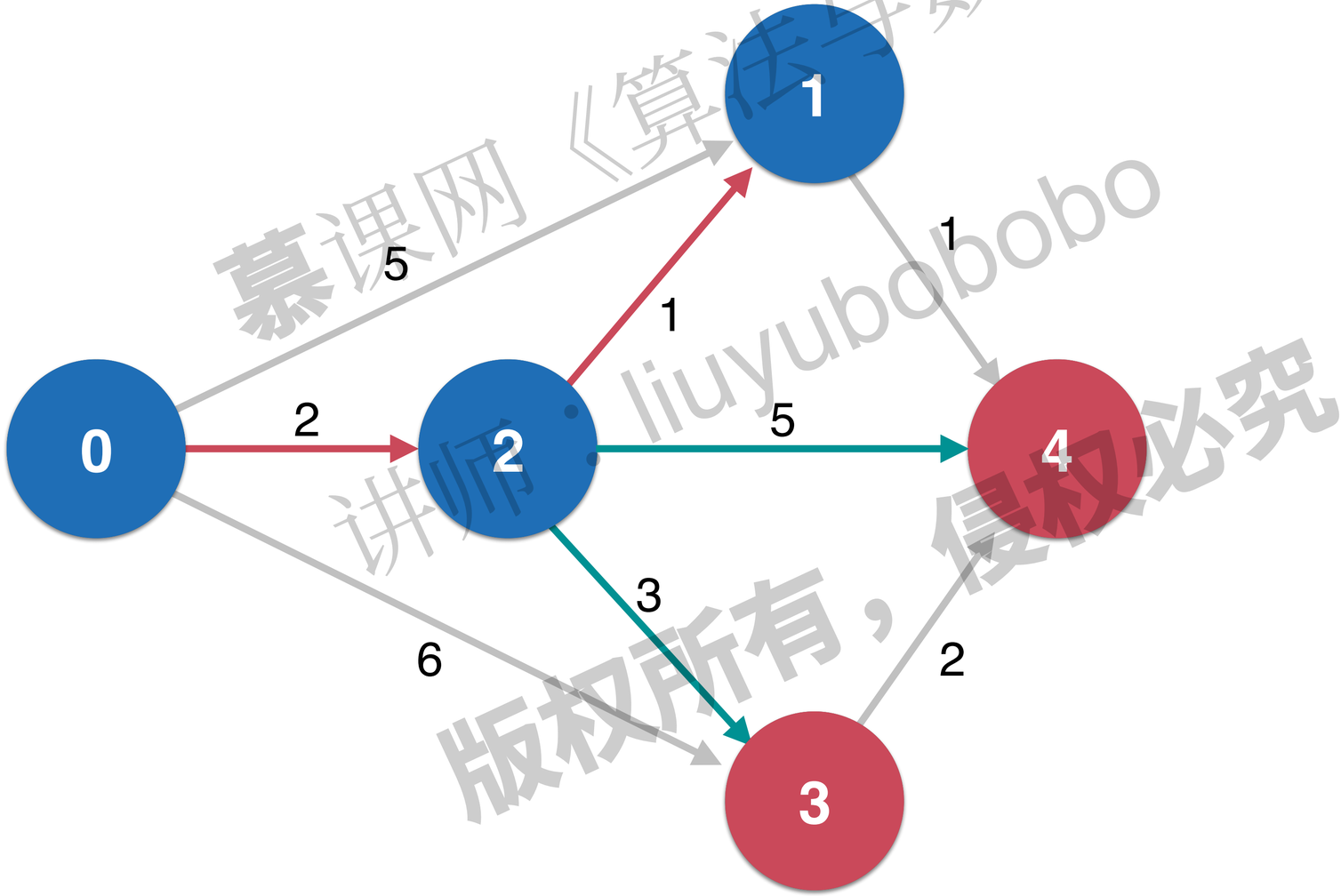
dijkstra 单源最短路径算法



确定新的节点的最短路径后，
进行Relaxation

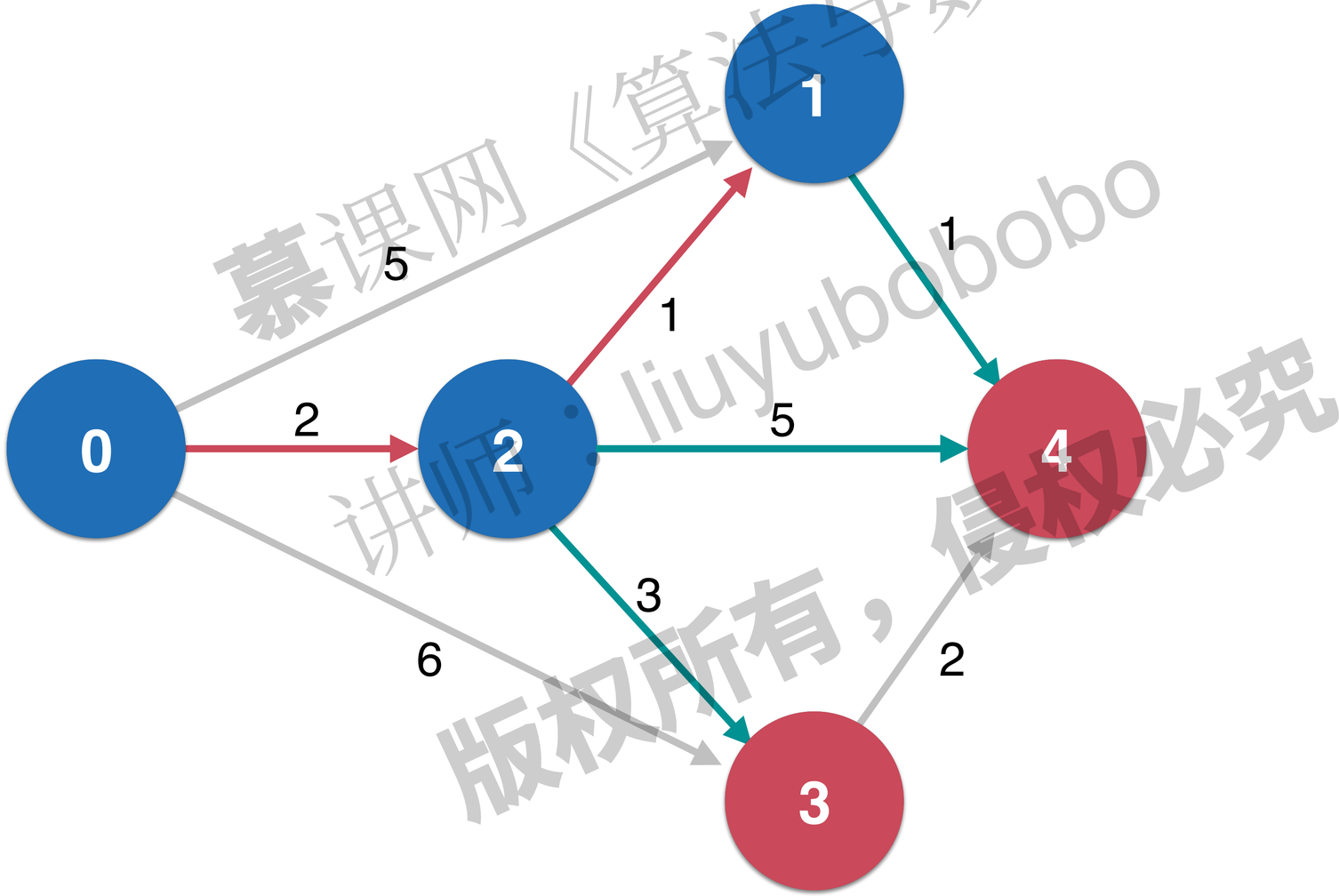
0	0
1	3
2	2
3	5
4	7

dijkstra 单源最短路径算法



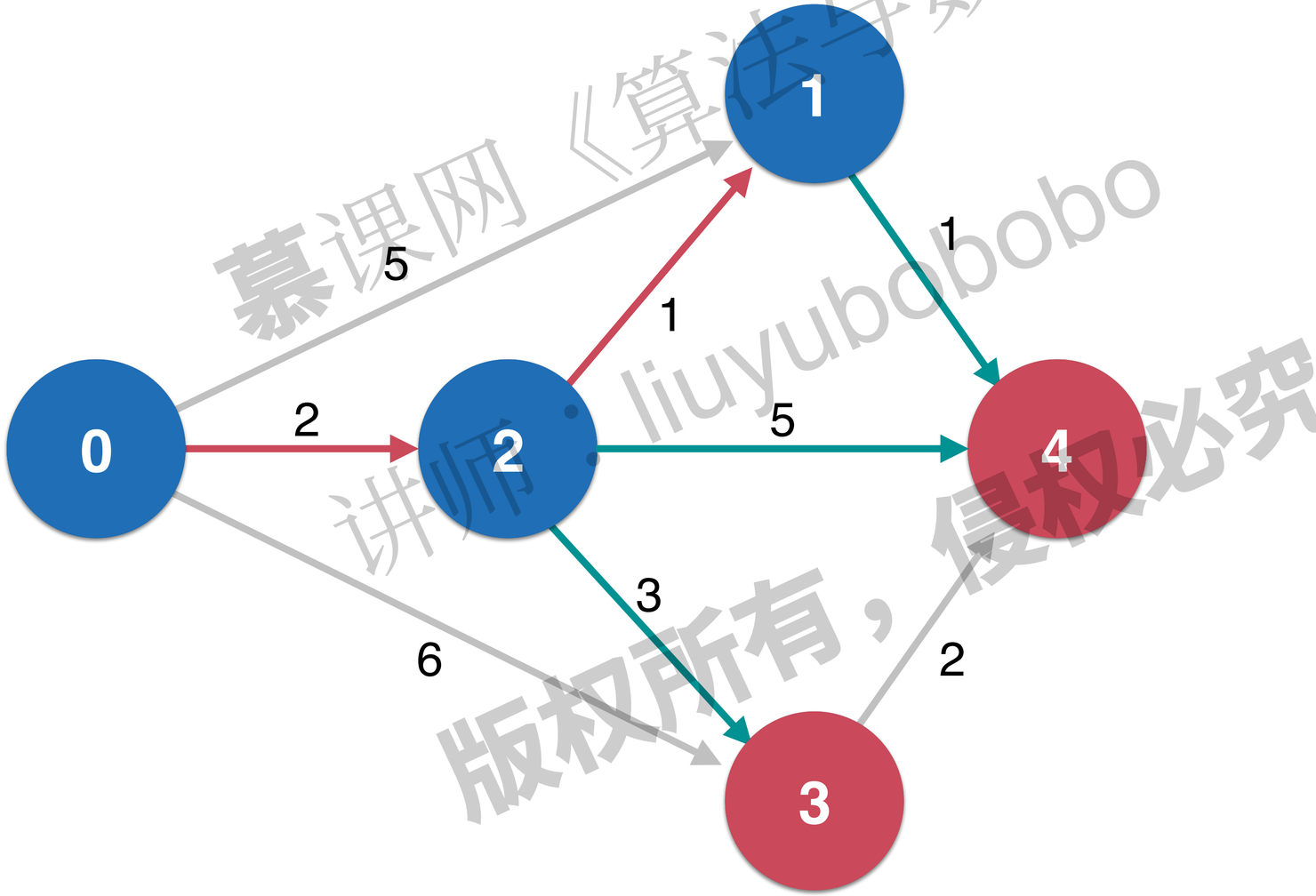
0	0
1	3
2	2
3	5
4	7

dijkstra 单源最短路径算法



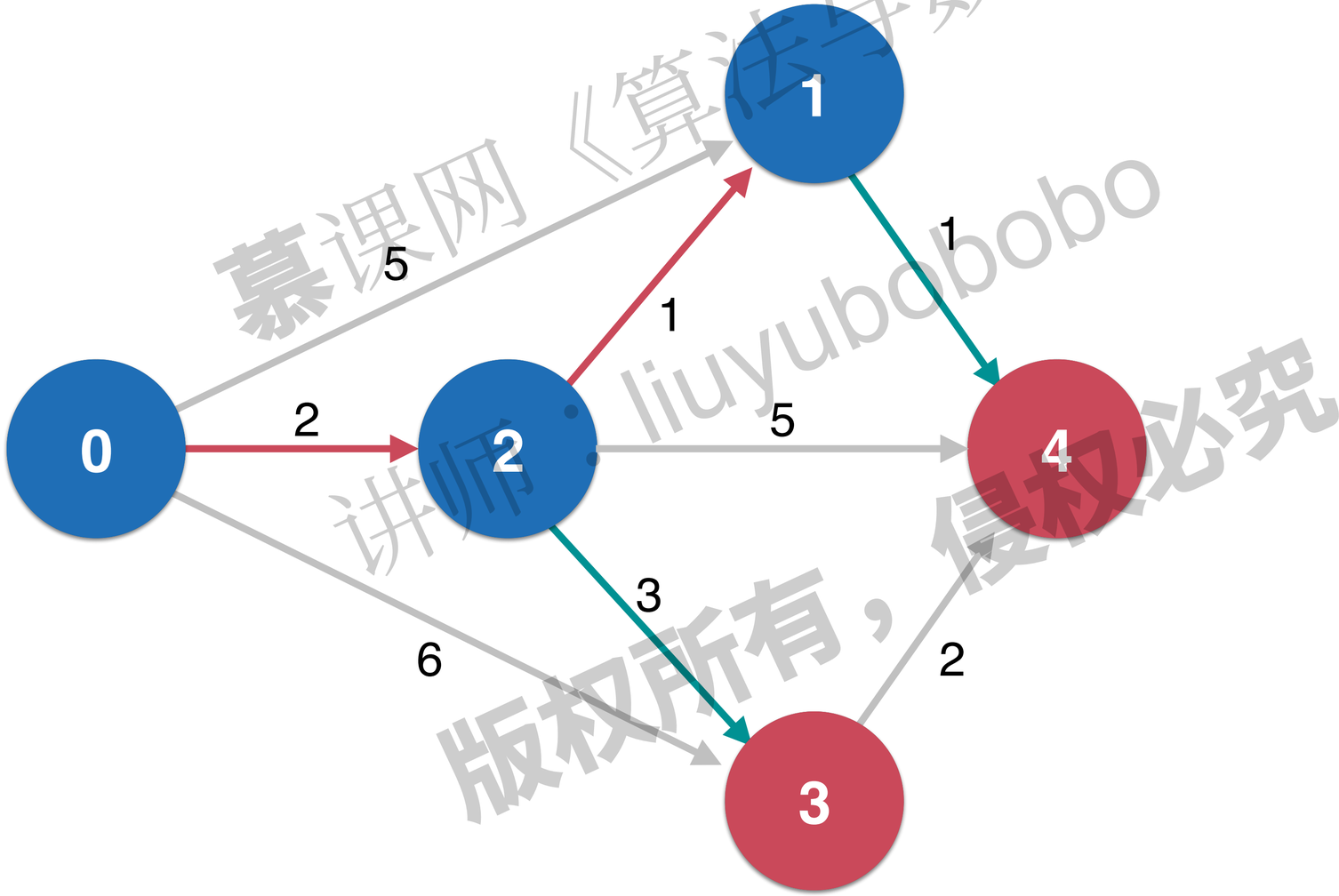
0	0
1	3
2	2
3	5
4	7

dijkstra 单源最短路径算法



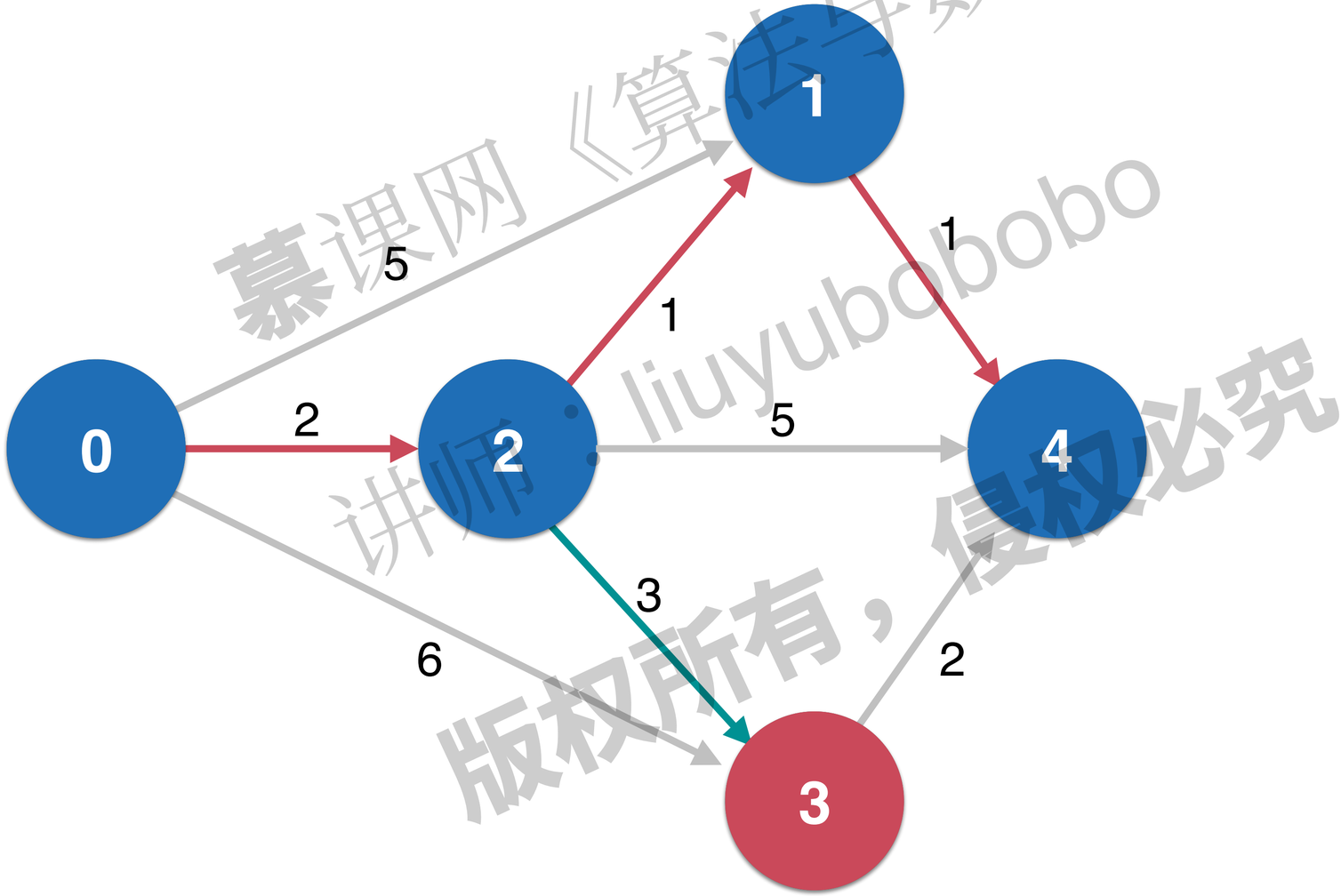
0	0
1	3
2	2
3	5
4	4

dijkstra 单源最短路径算法



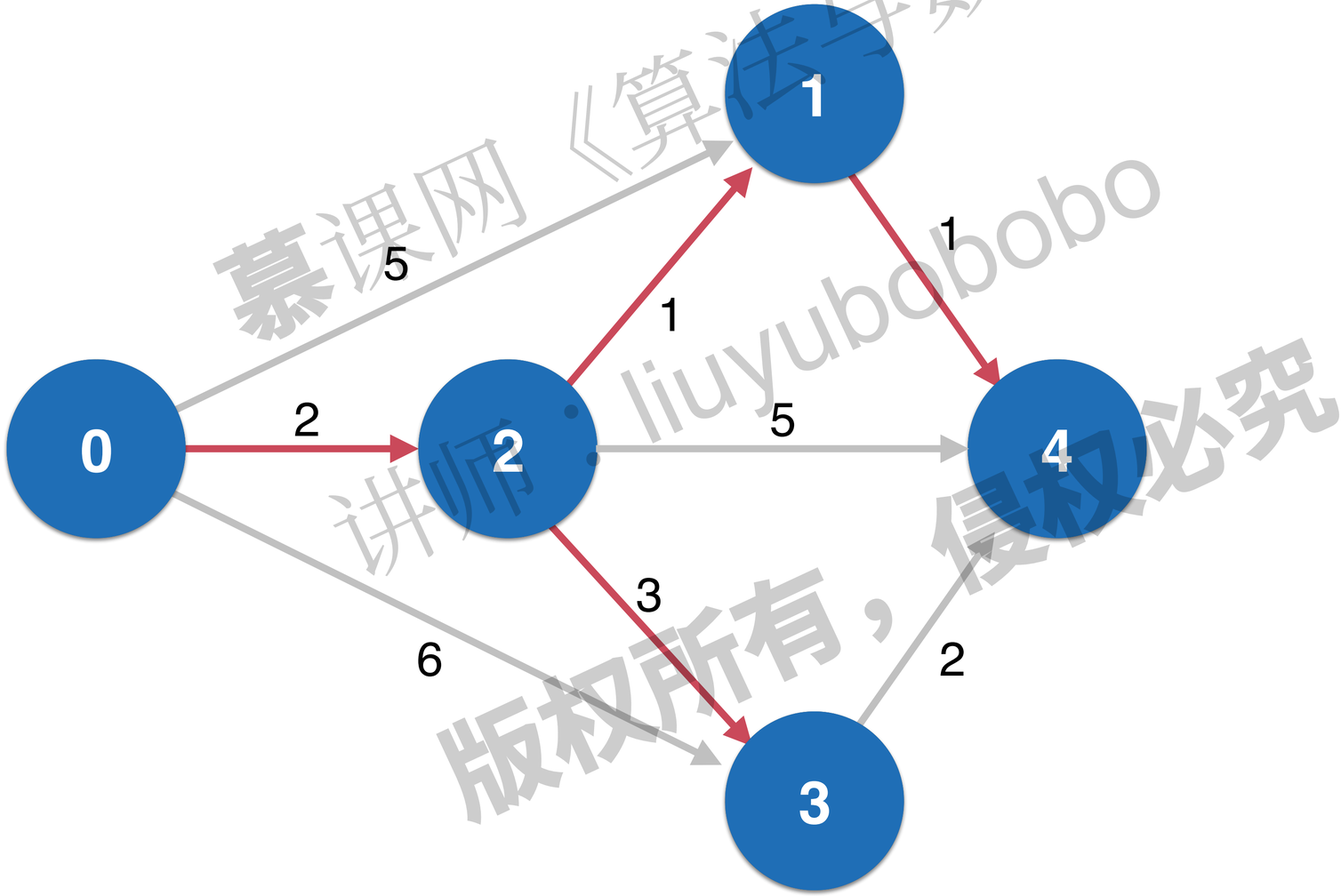
0	0
1	3
2	2
3	5
4	4

dijkstra 单源最短路径算法



0	0
1	3
2	2
3	5
4	4

dijkstra 单源最短路径算法



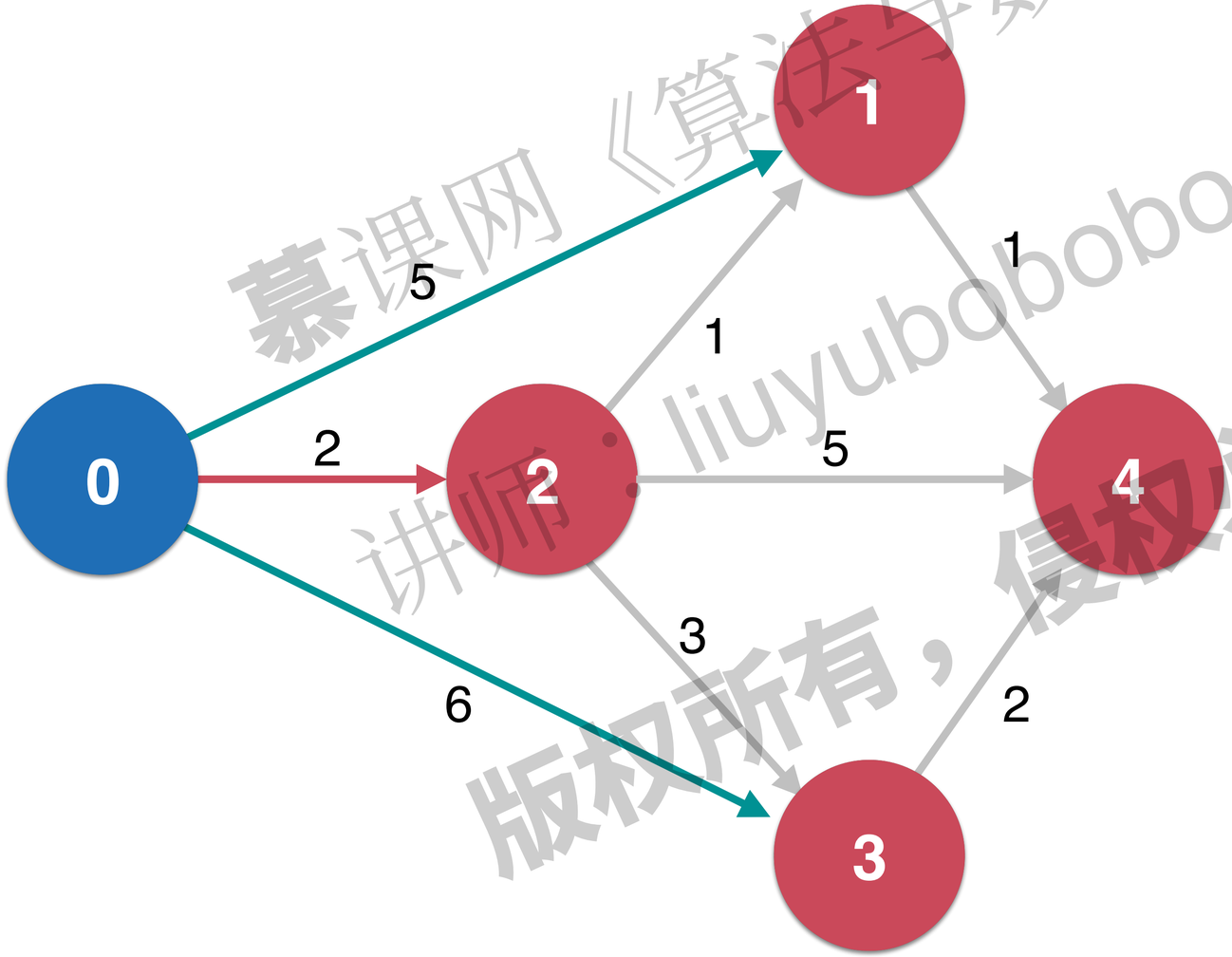
IndexMinHeap

0	0
1	3
2	2
3	5
4	4

操作：实现 dijkstra

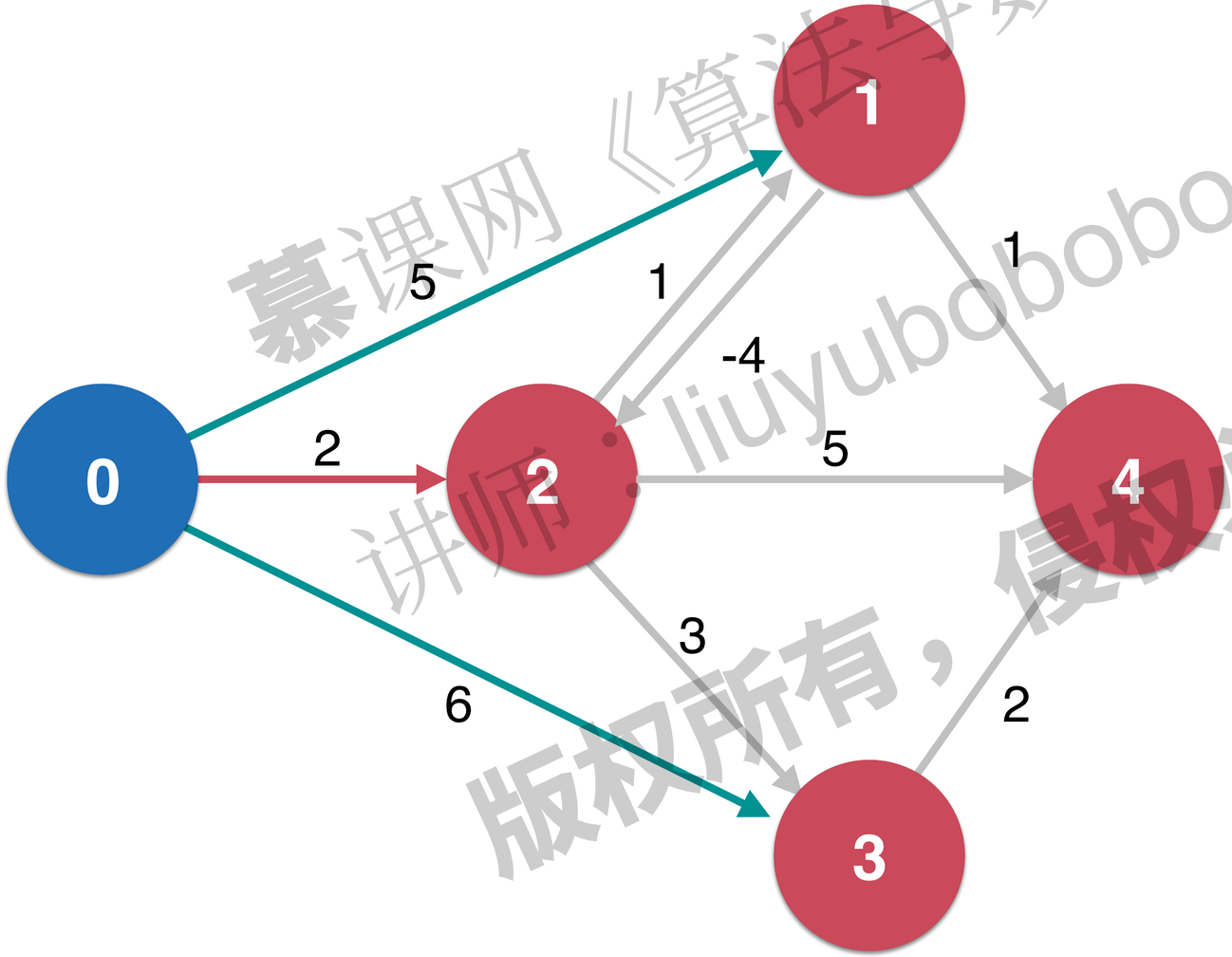
处理负权边

处理负权边



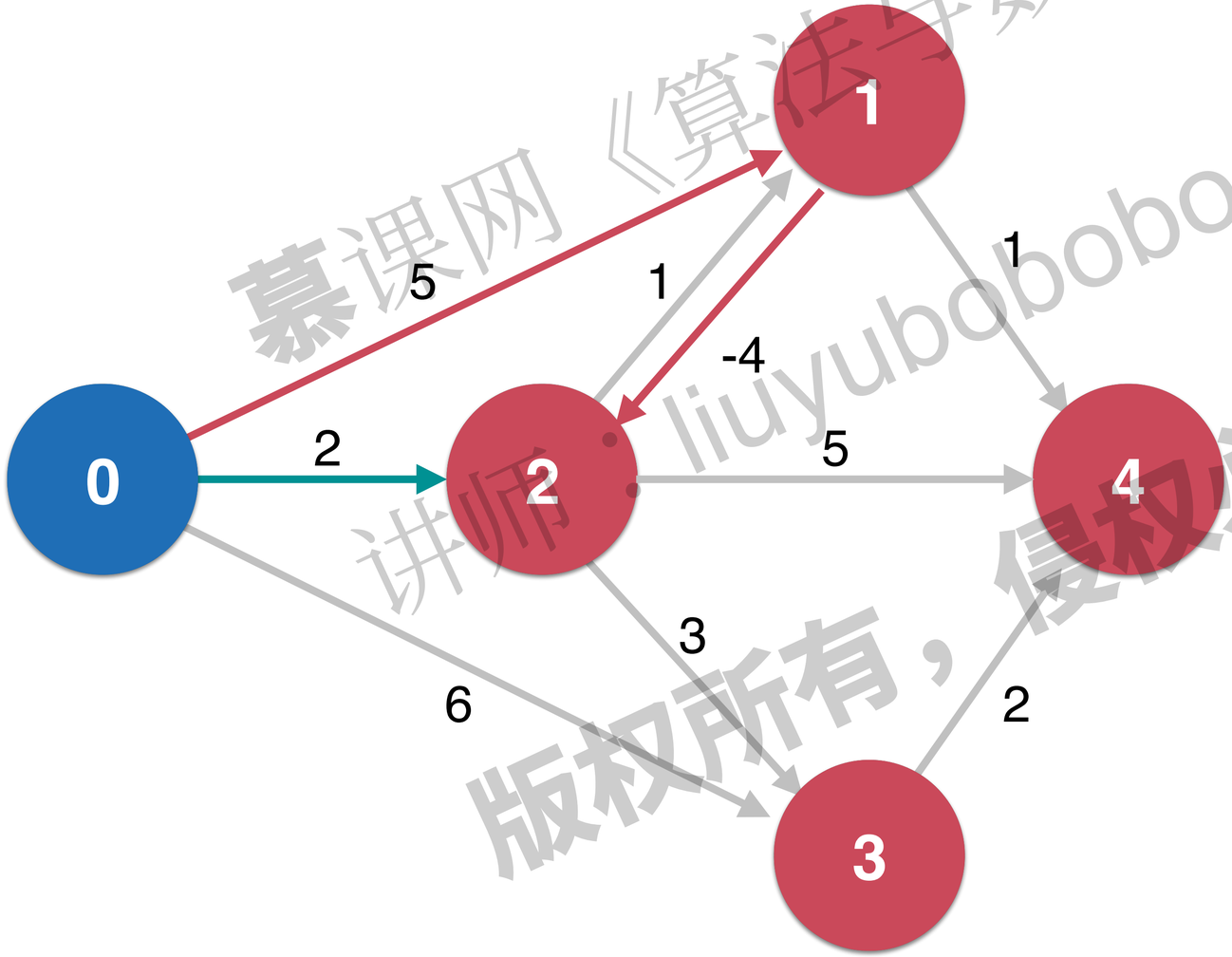
0	0
1	5
2	2
3	6
4	-

处理负权边



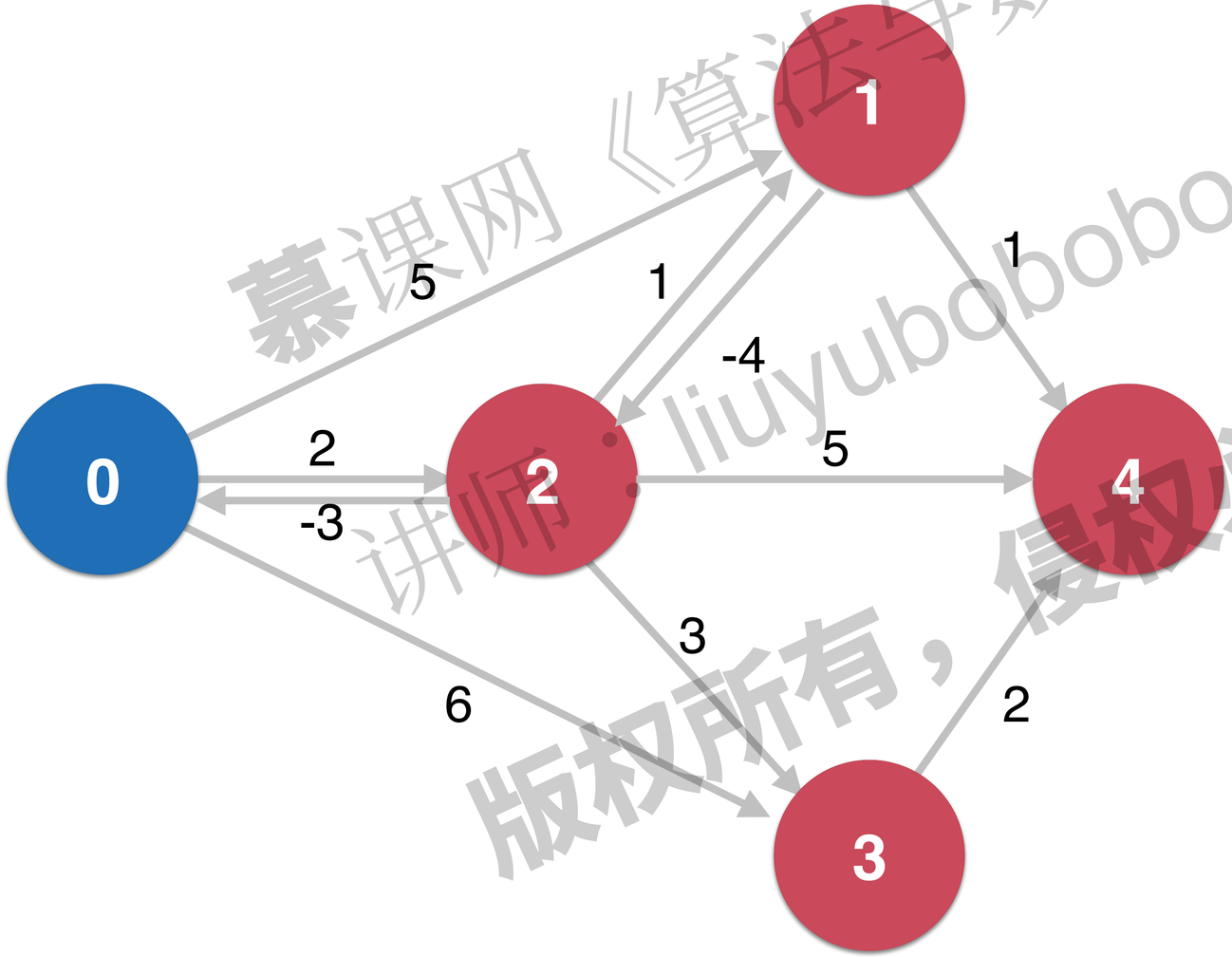
0	0
1	5
2	2
3	6
4	-

处理负权边



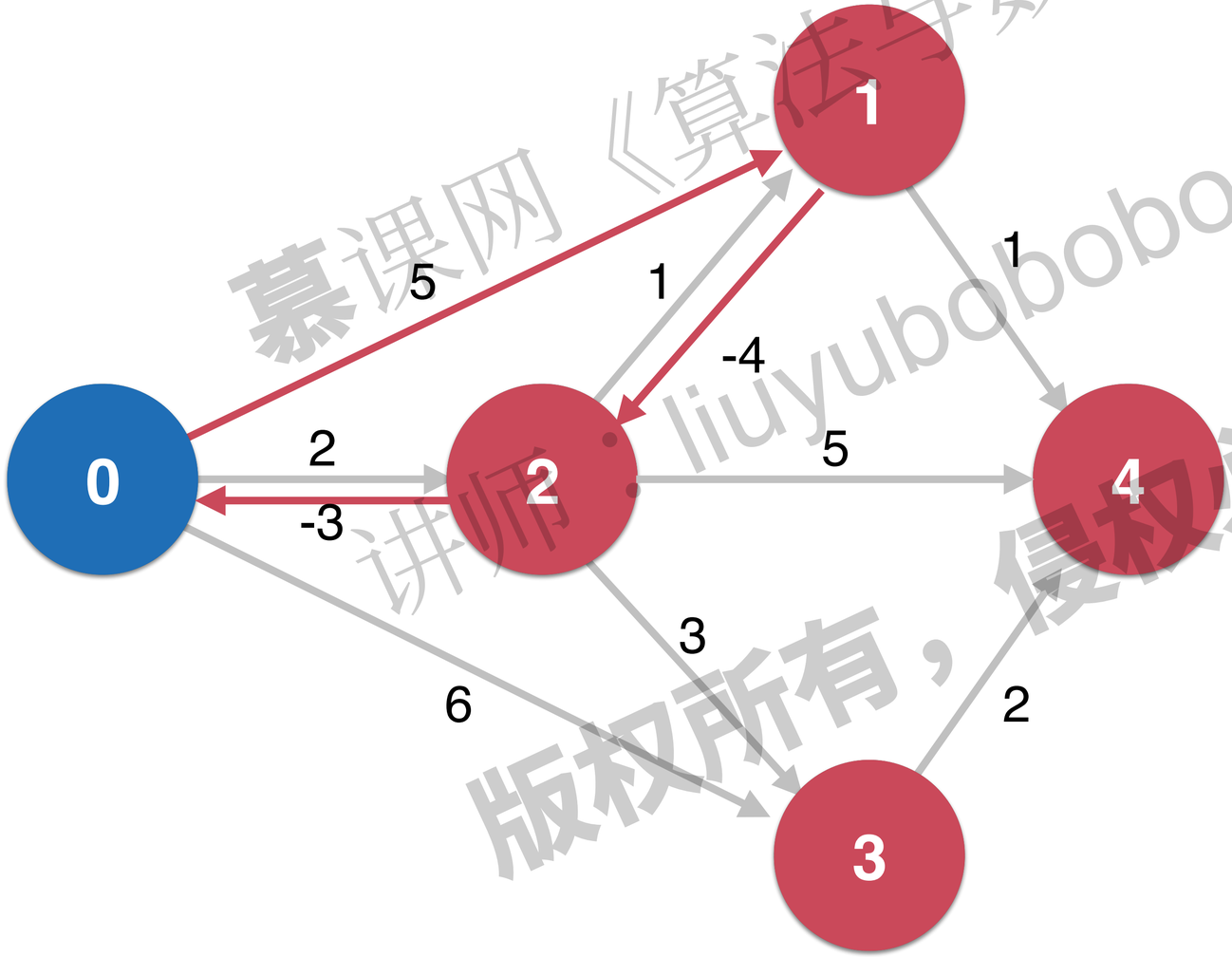
0	0
1	5
2	1
3	6
4	-

处理负权边



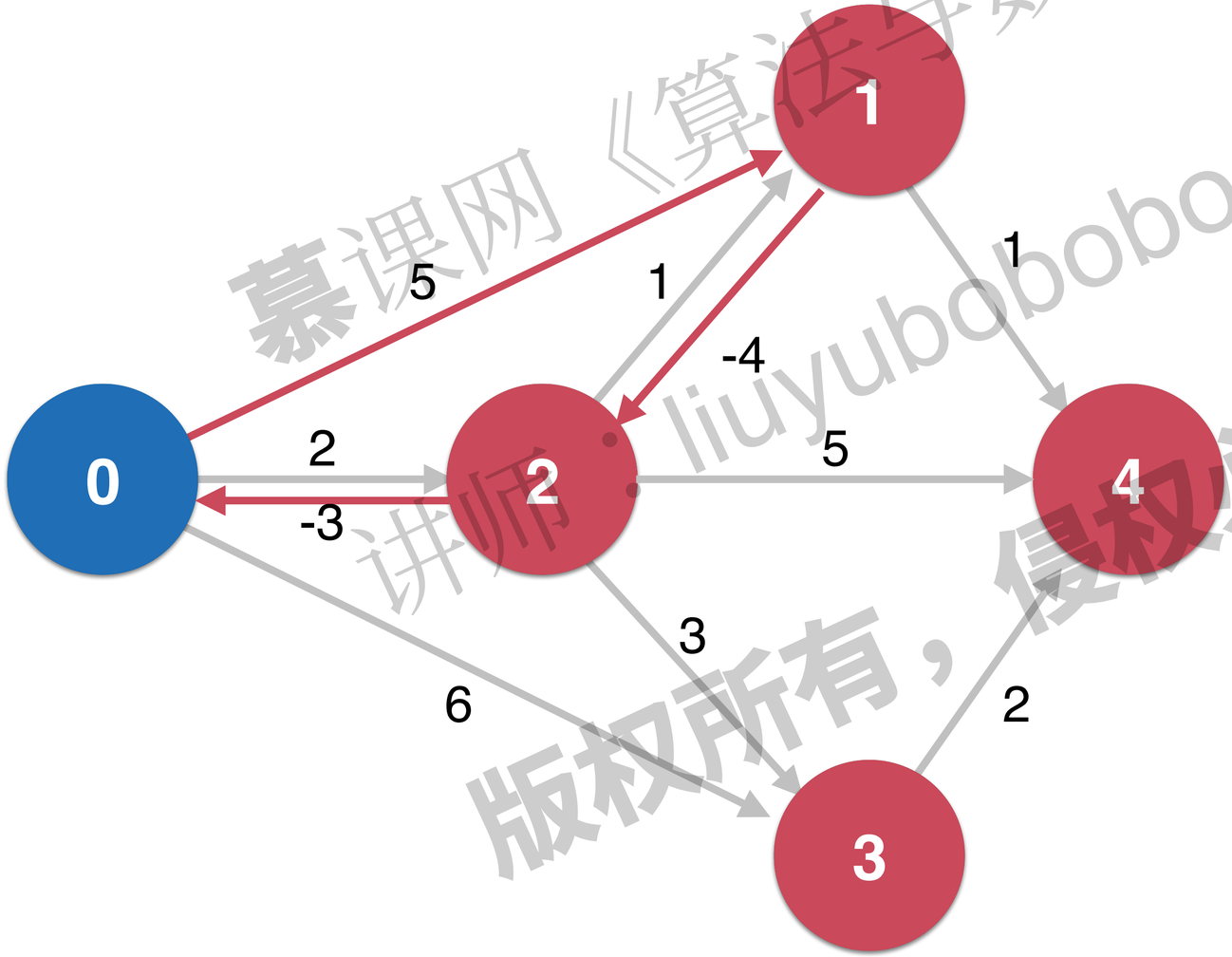
0	0
1	5
2	1
3	6
4	-

处理负权边



0	0
1	5
2	1
3	6
4	-

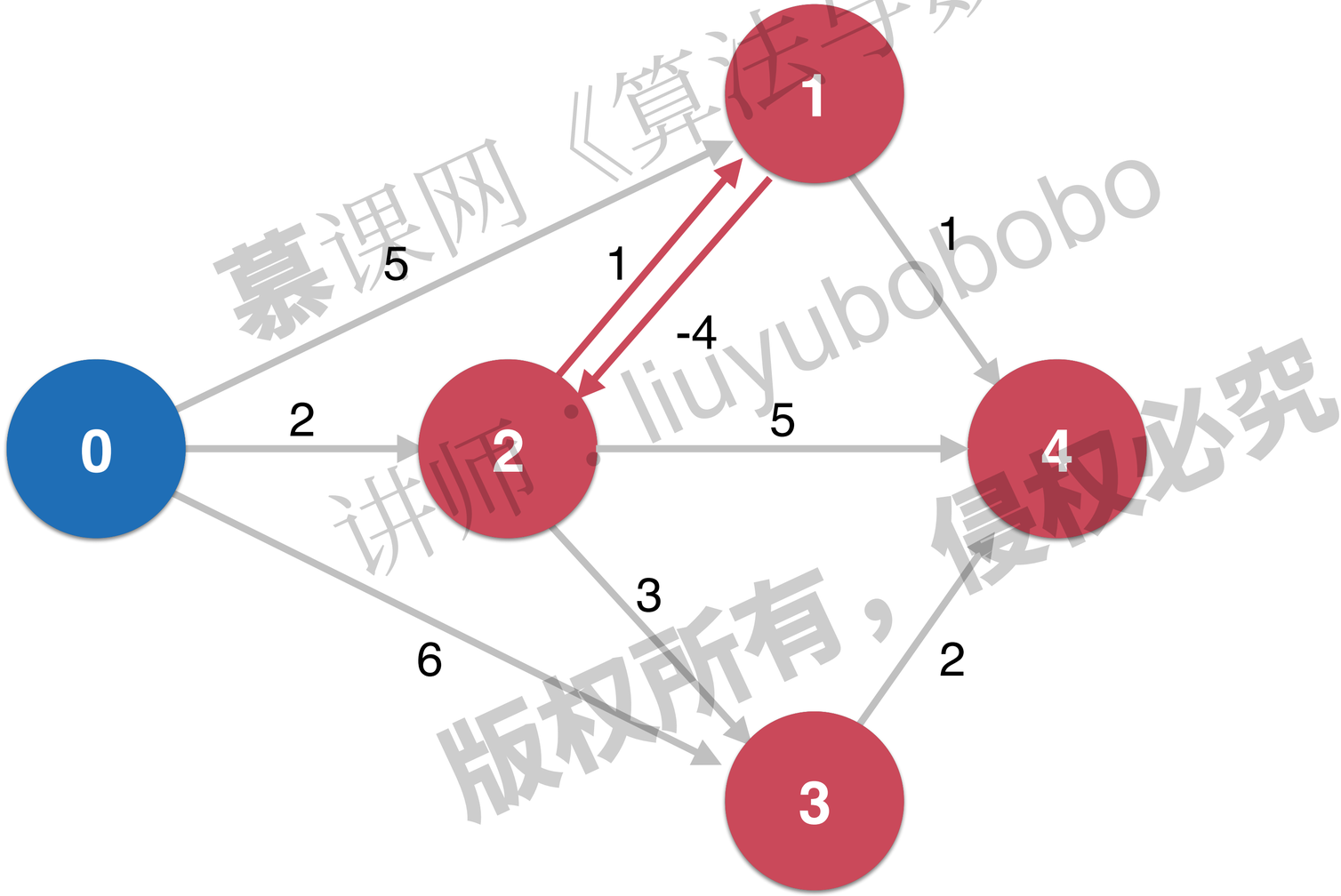
处理负权边



拥有负权环的图，

没有最短路径

处理负权边



Bellman-Ford 单源最短路径算法

Bellman-Ford 单源最短路径算法

前提：图中不能有负权环

Bellman-Ford可以判断图中是否有负权环

复杂度 O(EV)

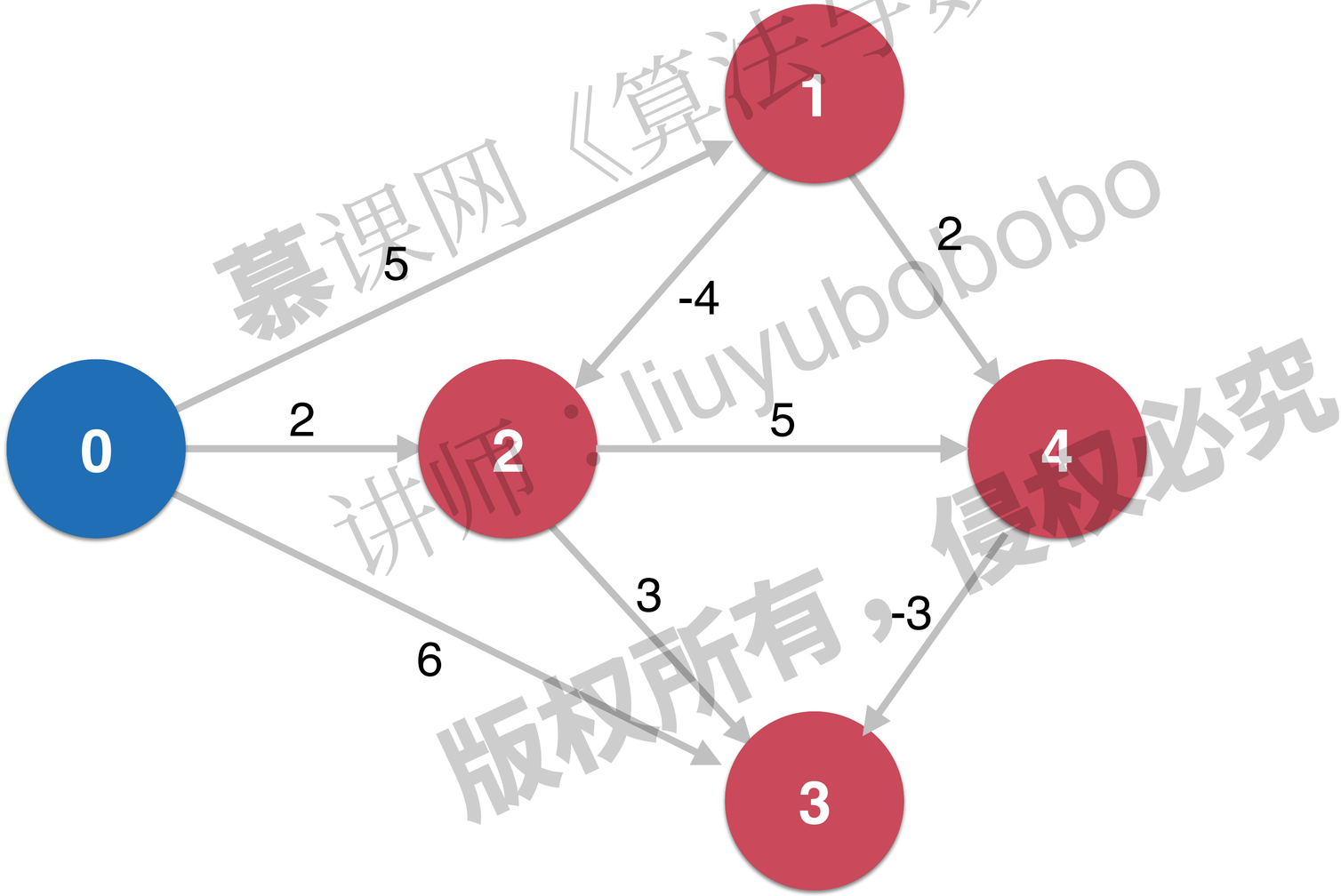
Bellman-Ford 单源最短路径算法

如果一个图没有负权环，

从一点到另外一点的最短路径，最多经过所有的V个顶线，有V-1条边

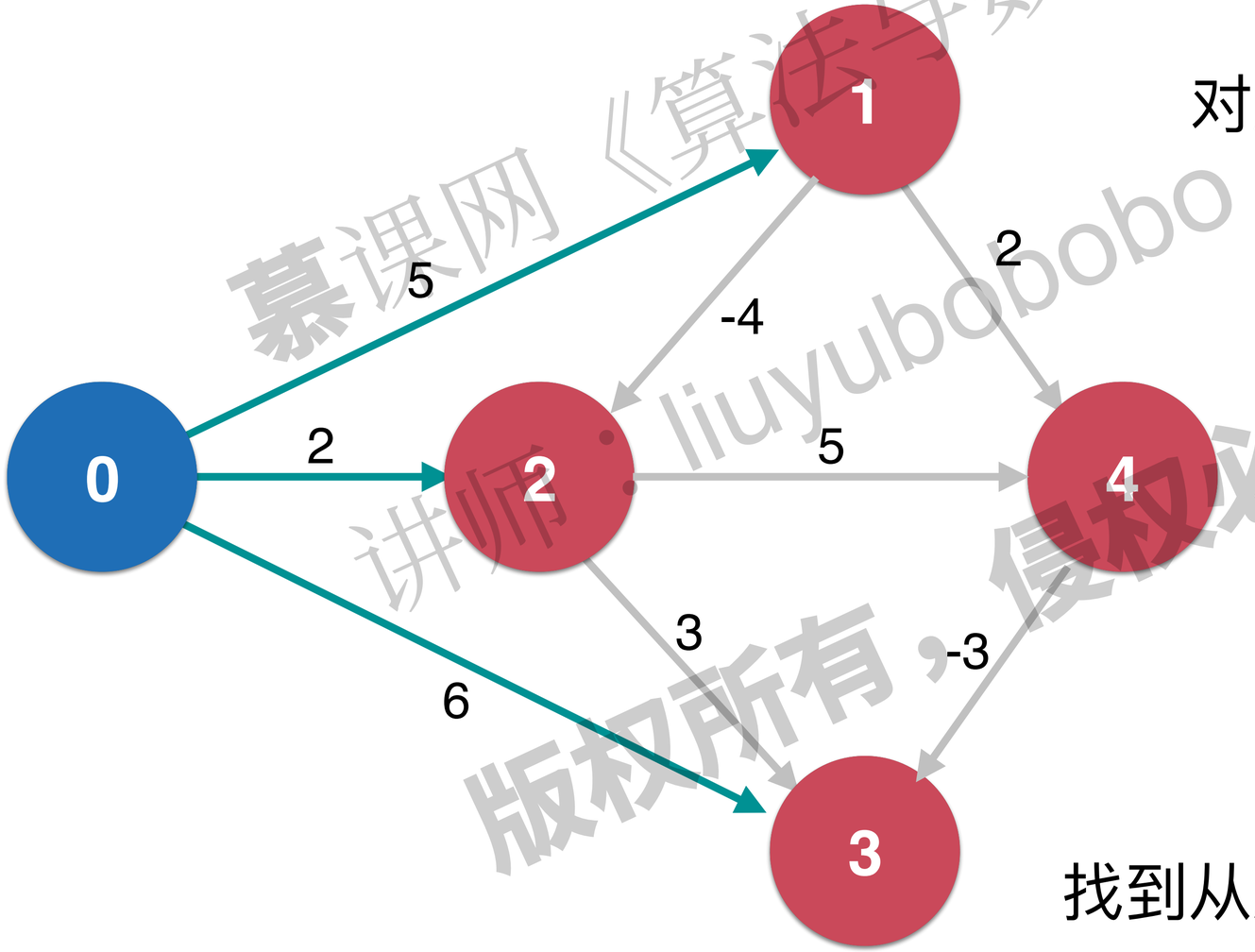
否则，存在顶点经过了两次，既存在负权环

Bellman-Ford 单源最短路径算法



0	0
1	-
2	-
3	-
4	-

Bellman-Ford 单源最短路径算法

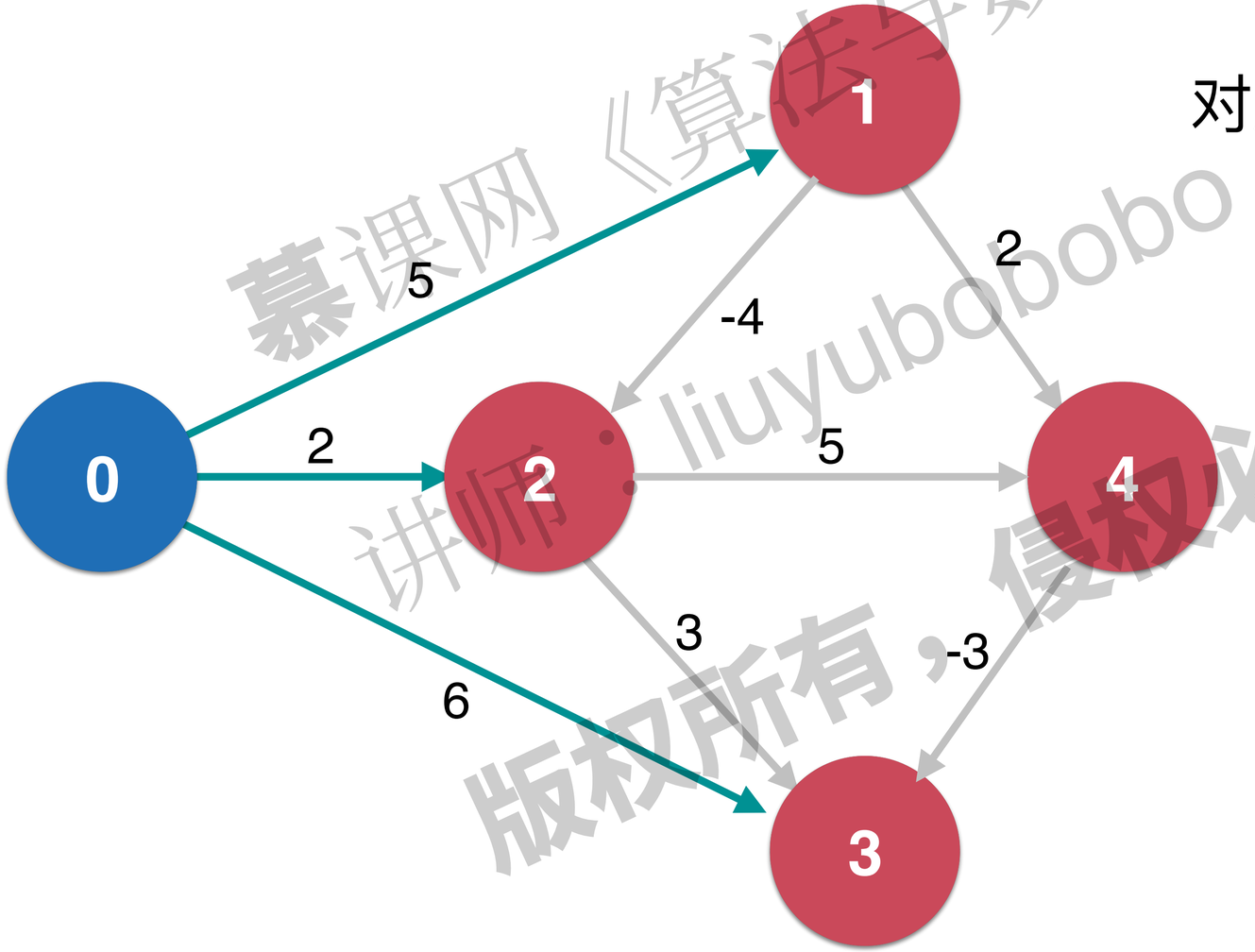


对所有的点进行第1次松弛操作

0	0	
1	5	0->1
2	2	0->2
3	6	0->3
4	-	

找到从原点开始，经过1条边的最短路径

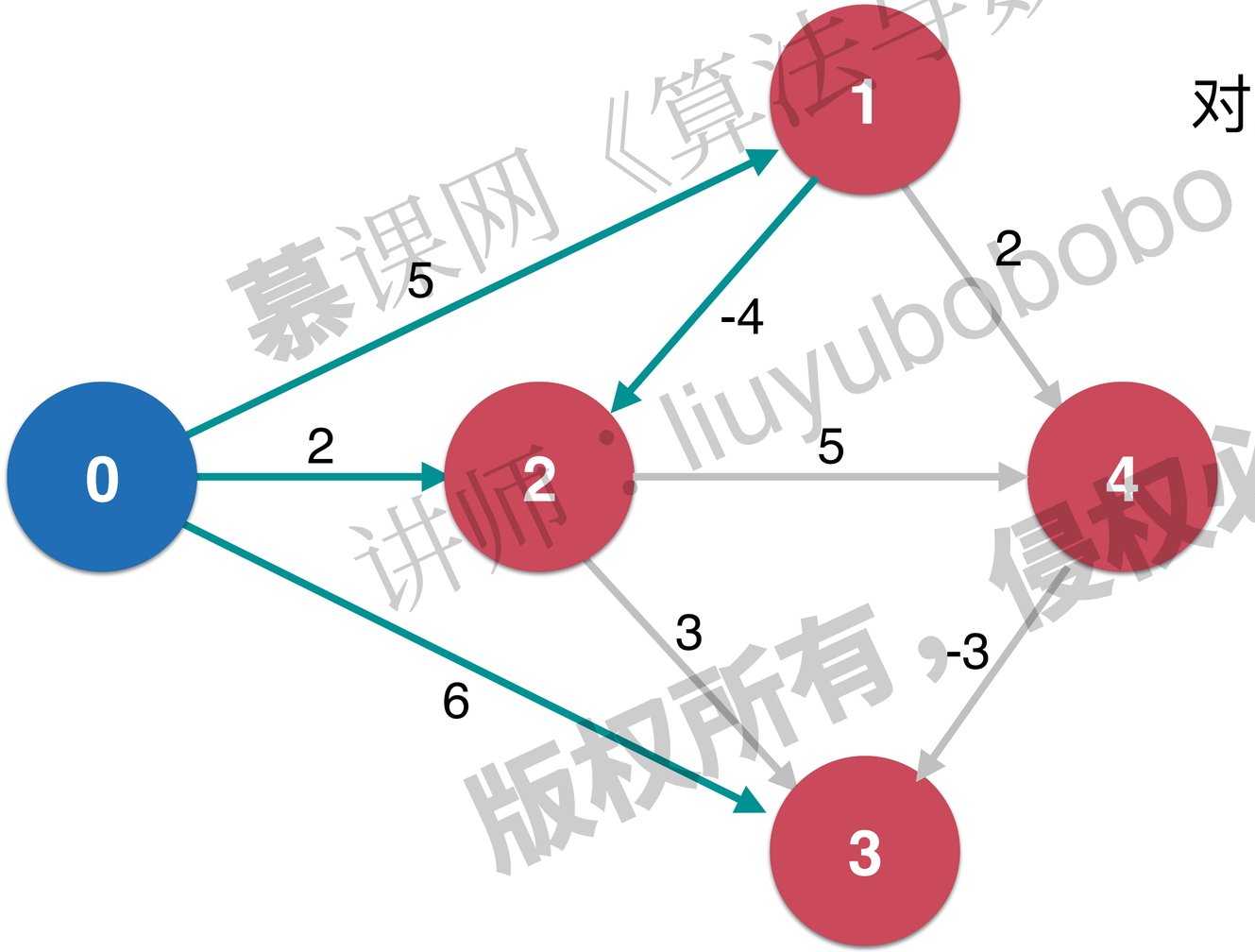
Bellman-Ford 单源最短路径算法



对所有的点进行第2次松弛操作

0	0	
1	5	0->1
2	2	0->2
3	6	0->3
4	-	

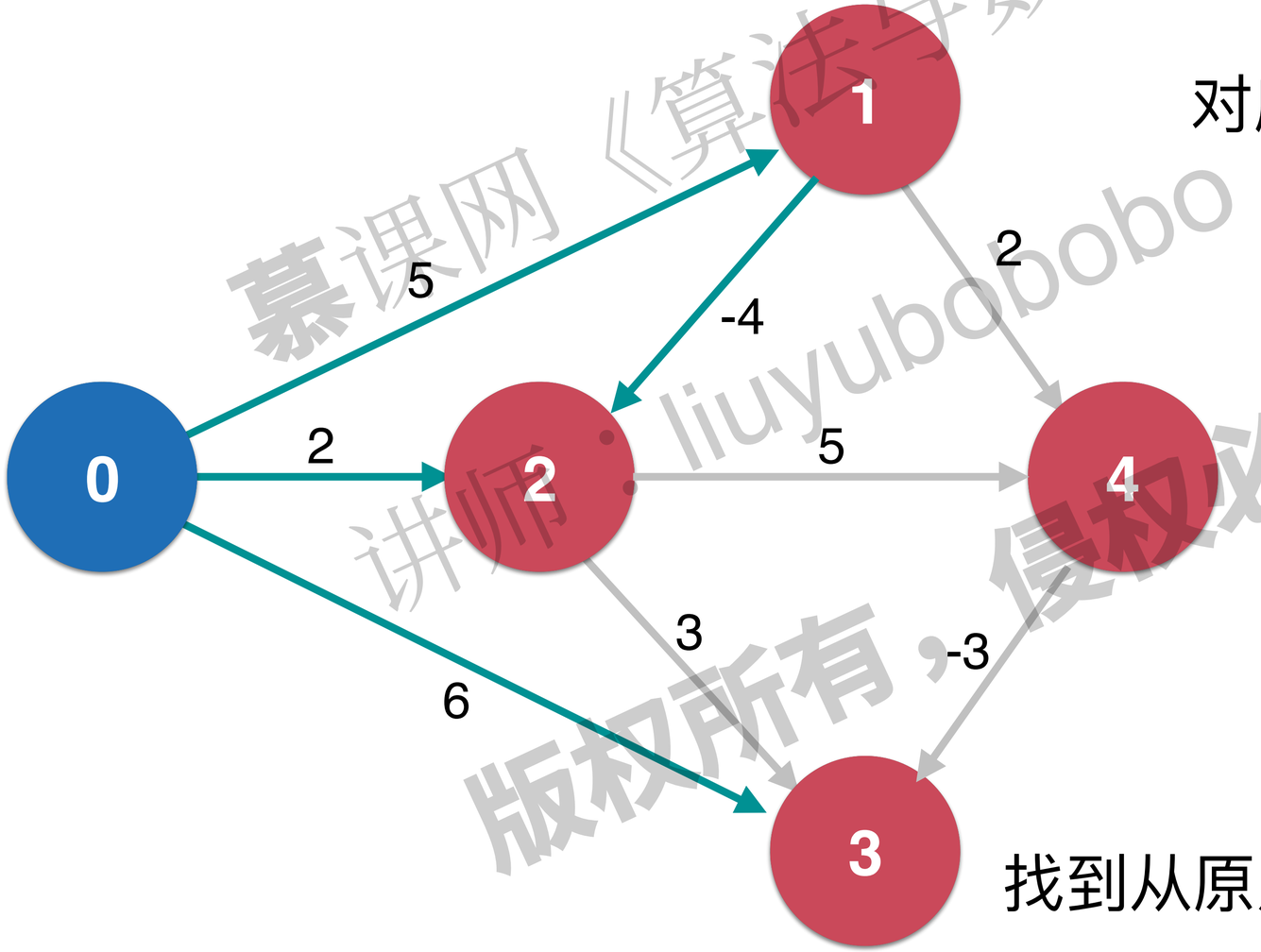
Bellman-Ford 单源最短路径算法



对所有的点进行第2次松弛操作

0	0	
1	5	0->1
2	2	0->2
3	6	0->3
4	-	

Bellman-Ford 单源最短路径算法



对所有的点进行第2次松弛操作

0	0	
1	5	0->1
2	1	0->1->2
3	6	0->3
4	-	

找到从原点开始，经过2条边的最短路径

Bellman-Ford 单源最短路径算法

对一个点的一次松弛操作，就是找到经过这个点的另外一条路径，多一条边，权值更小。

如果一个图没有负权环，从一点到另外一点的最短路径，最多经过所有的V个顶线，有V-1条边

对所有的点进行V-1次松弛操作

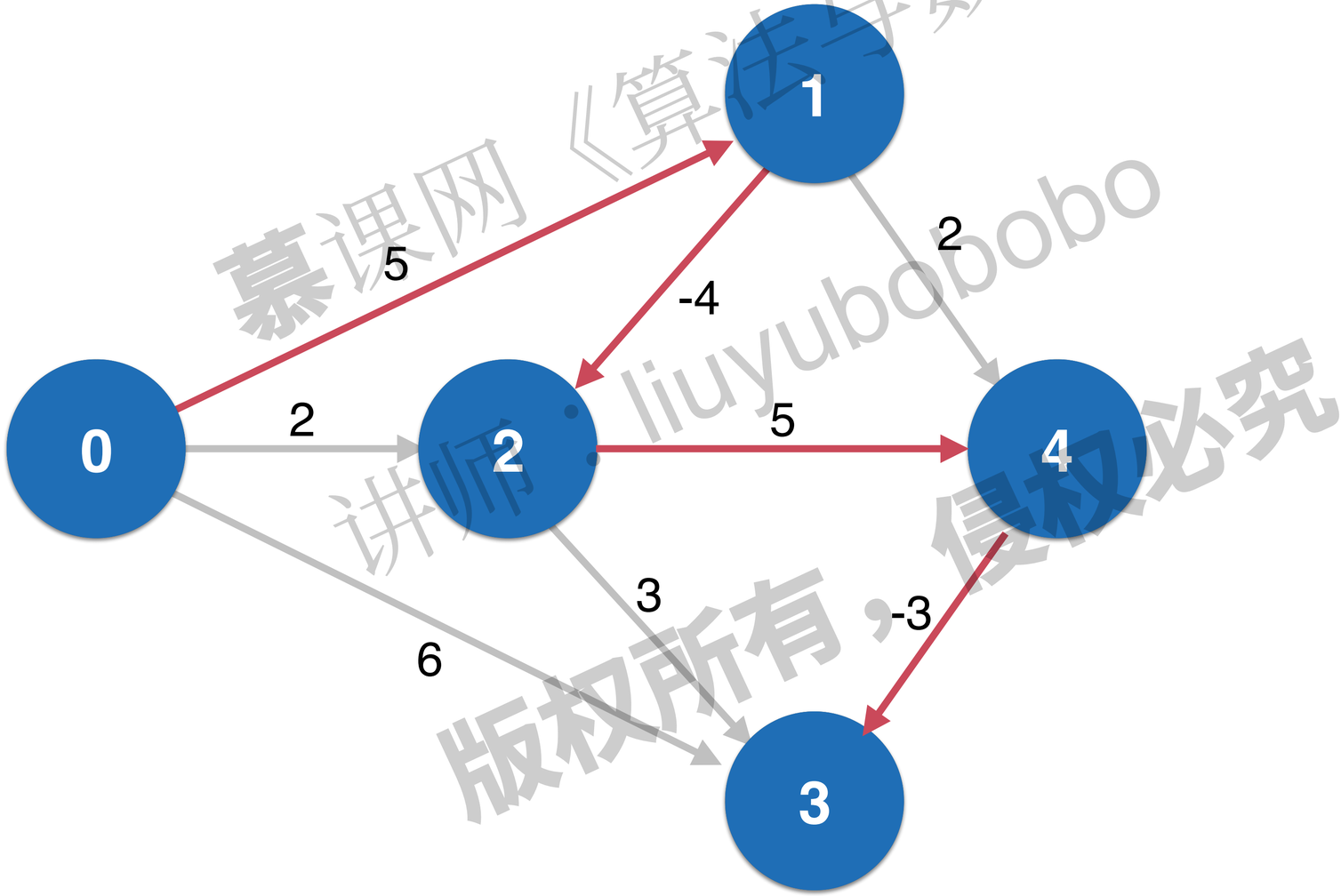
Bellman-Ford 单源最短路径算法

对所有的点进行V-1次松弛操作，理论上就找到了从源点到其他所有点的最短路径。

如果还可以继续松弛，所说原图中有负权环。

操作：Bellman-Ford 单源最短路径

Bellman-Ford 单源最短路径算法



更多和最短路径相关的问题

单源最短路径算法

具体实现，distTo[i] 初始化为“正无穷”

Bellman-Ford算法的优化

利用队列数据结构

queue-based bellman-ford算法

单源最短路径算法

dijkstra	无负权边	有向无向图均可	O(ElogV)
Bellman-Ford	无负权环	有向图	O(VE)
利用拓扑排序	有向无环图 DAG	有向图	O(V + E)

所有对最短路径算法

Floyed算法，处理无负权环的图

O(V^3)

最长路径算法

最长路径问题不能有正权环。

无权图的最长路径问题是指数级难度的。

对于有权图，不能使用Dijkstra求最长路径问题。

可以使用 Bellman-Ford算法。

其他

欢迎大家关注我的个人公众号：是不是很酷

算法与数据结构

liuyubobobo