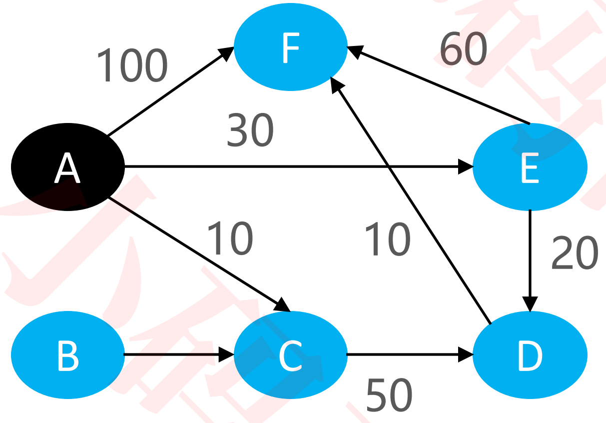
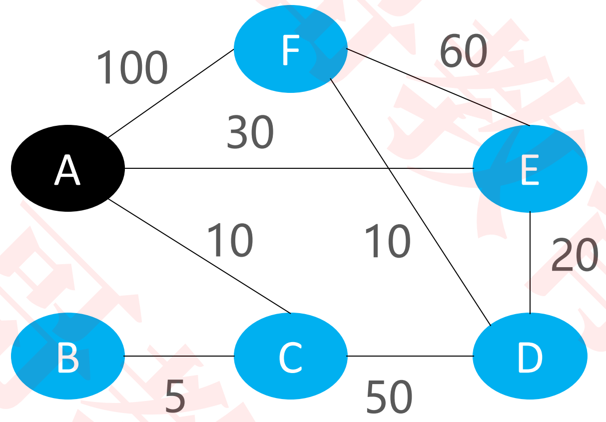
最短路径（Shortest Path）

■ 最短路径是指两顶点之间权值之和最小的路径（有向图、无向图均适用，不能有负权环）



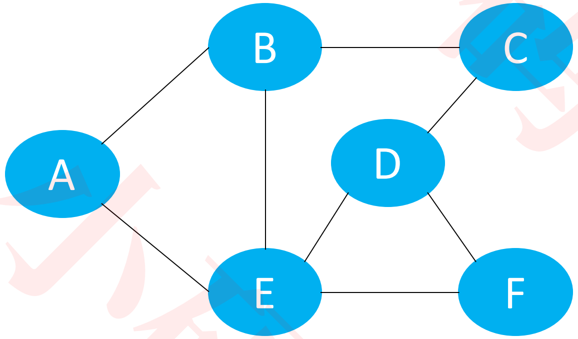
源点	终点	最短路径	路径长度
A	B		∞
	C	A → C	10
	D	A → E → D	50
	E	A → E	30
	F	A → E → D → F	60

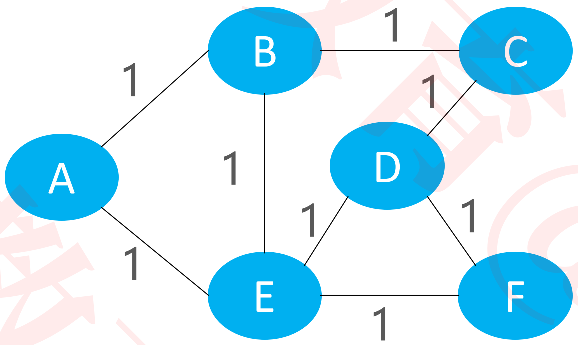


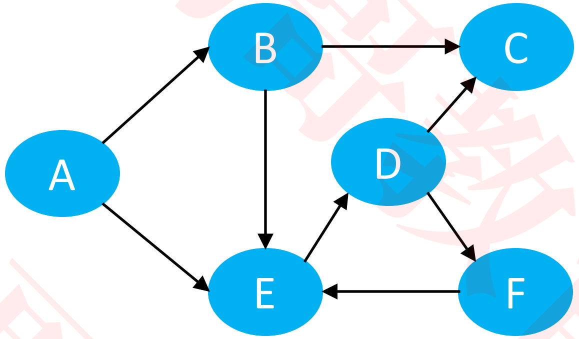
源点	终点	最短路径	路径长度
A	B	A → C → B	15
	C	A → C	10
	D	A → E → D	50
	E	A → E	30
	F	A → E → D → F	60

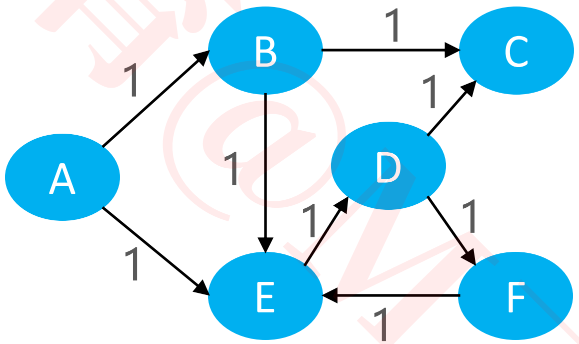
最短路径 – 无权图

- 无权图相当于是全部边权值为1的有权图



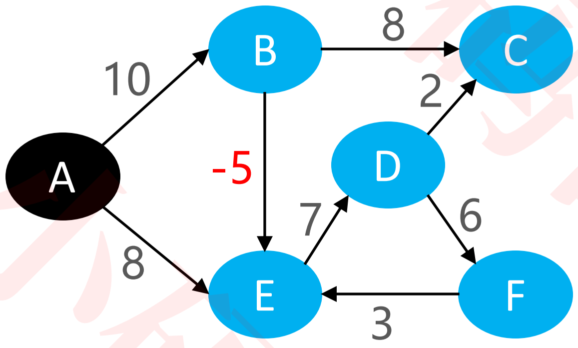






最短路径 – 负权边

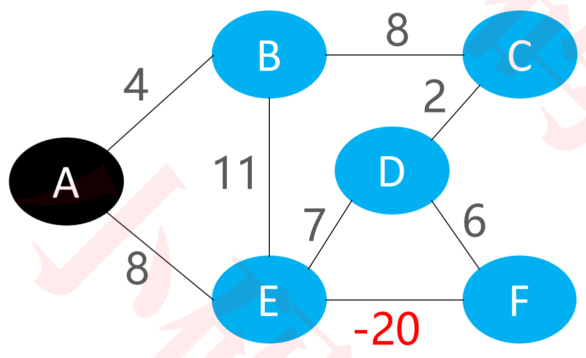
■ 有负权边，但没有负权环时，存在最短路径

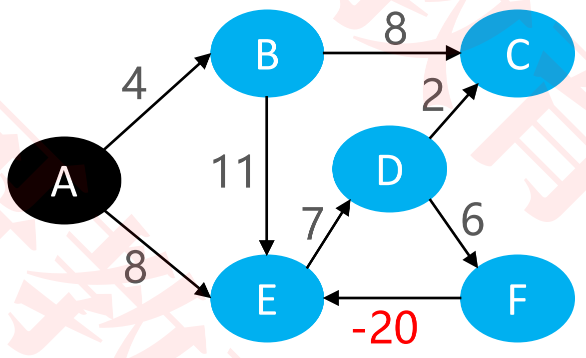


■ A到E的最短路径是：A → B → E

最短路径 – 负权环

- 有负权环时，不存在最短路径





- 通过负权环， A到E的路径可以无限短
  - A → E → D → F → E → D → F → E → D → F → E → D → F → E → ......

最短路径

■ 最短路径的典型应用之一：路径规划问题

■ 求解最短路径的3个经典算法

□ 单源最短路径算法

✓ Dijkstra（迪杰斯特拉算法）

✓ Bellman-Ford（贝尔曼-福特算法）

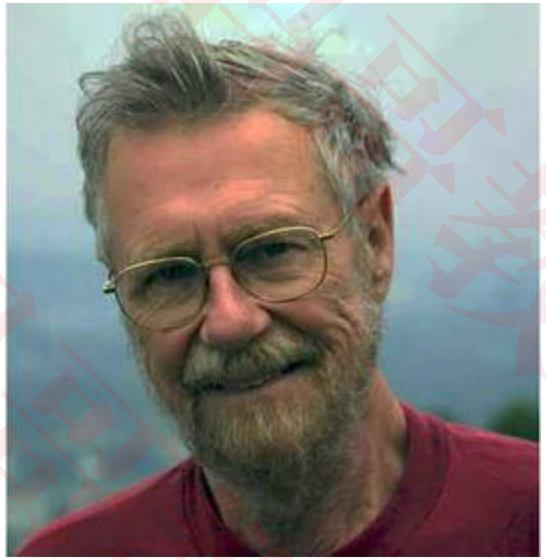
□ 多源最短路径算法

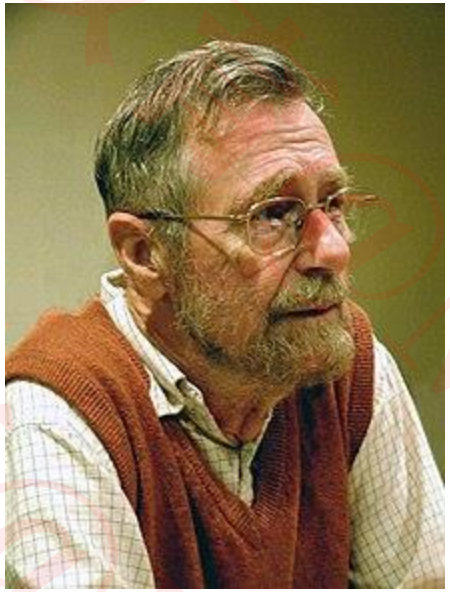
✓ Floyd（弗洛伊德算法）

Dijkstra

- Dijkstra 属于单源最短路径算法，用于计算一个顶点到其他所有顶点的最短路径
- 使用前提：不能有负权边
- 时间复杂度：可优化至 O(ElogV)，E 是边数量，V 是节点数量

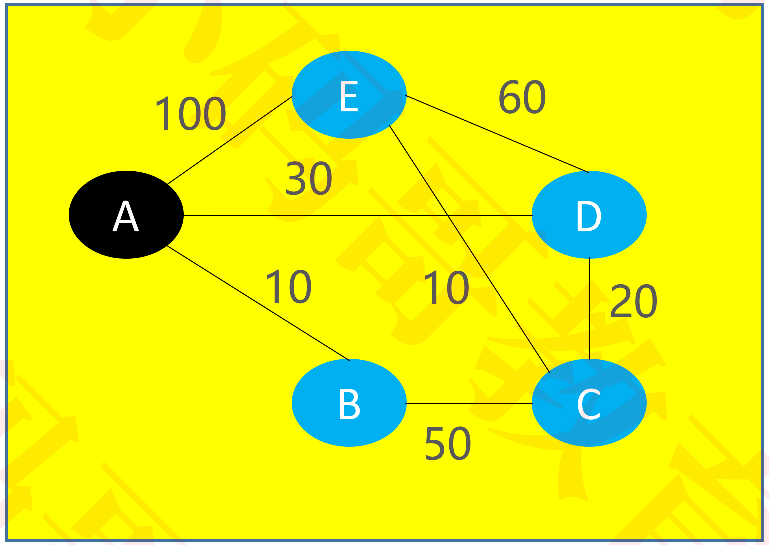
- 由荷兰的科学家 Edsger Wybe Dijkstra 发明，曾在1972年获得图灵奖



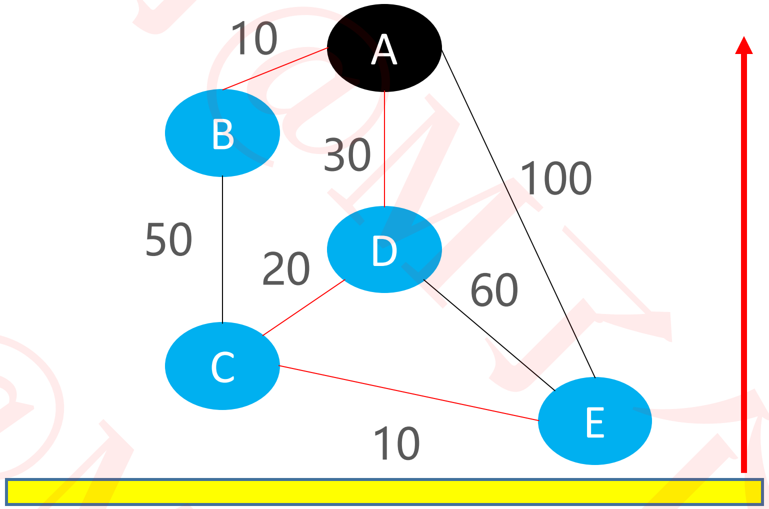


Dijkstra – 等价思考

- ■ Dijkstra 的原理其实跟生活中的一些自然现象完全一样
- □ 把每1个顶点想象成是1块小石头
- □ 每1条边想象成是1条绳子，每一条绳子都连接着2块小石头，边的权值就是绳子的长度
- □ 将小石头和绳子平放在一张桌子上（下图是一张俯视图，图中黄颜色的是桌子）

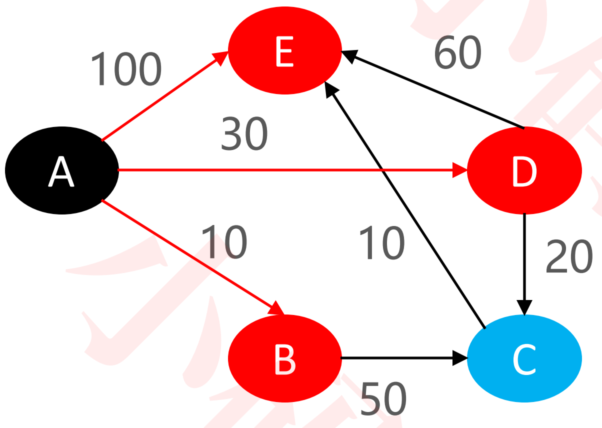


- ■ 接下来想象一下，手拽着小石头A，慢慢地向上提起来，远离桌面
- □ B、D、C、E会依次离开桌面
- □ 最后绷直的绳子就是A到其他小石头的最短路径

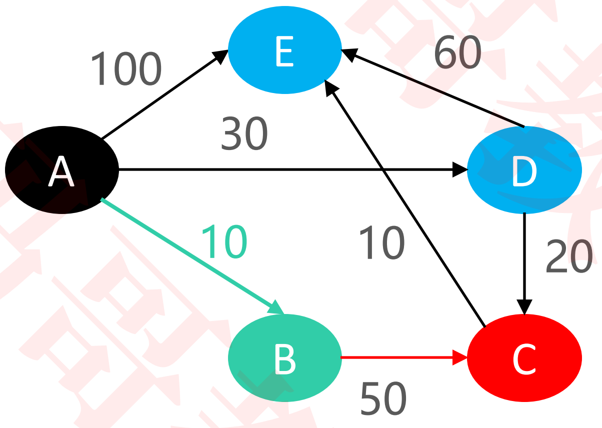


- ■ 有一个很关键的信息
- □ 后离开桌面的小石头
- ✓ 都是被先离开桌面的小石头拉起来的

Dijkstra – 执行过程



源点	终点	最短路径	路径长度
A	B	A → B	10
	C		∞
	D	A → D	30
	E	A → E	100

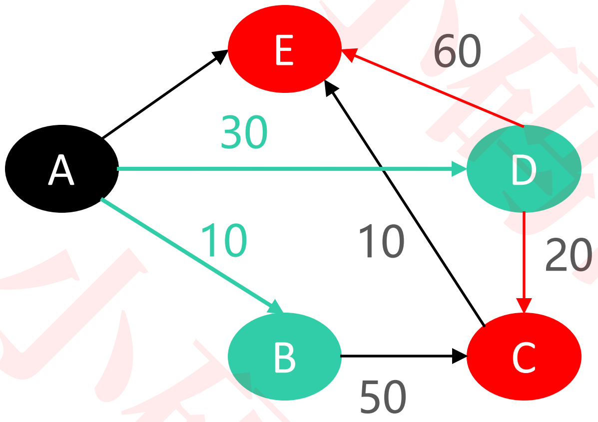


源点	终点	最短路径	路径长度
A	B	A → B	10
	C	A → B → C	60
	D	A → D	30
	E	A → E	100

- 绿色
  - 已经“离开桌面”
  - 已经确定了最终的最短路径
- 红色：更新了最短路径信息

小码哥教育 SEEMYGO

Dijkstra – 执行过程

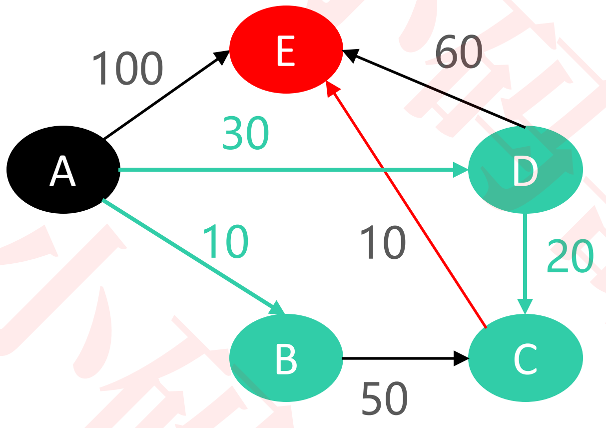


源点	终点	最短路径	路径长度
A	B	A → B	10
	C	A → D → C	50
	D	A → D	30
	E	A → D → E	90

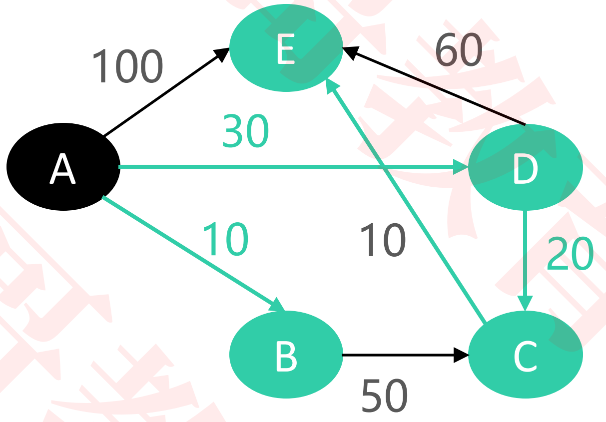
- 松弛操作（Relaxation）：更新2个顶点之间的最短路径
  - 这里一般是指：更新源点到另一个点的最短路径
  - 松弛操作的意义：尝试找出更短的最短路径

- 确定A到D的最短路径后，对DC、DE边进行松弛操作，更新了A到C、A到E的最短路径

Dijkstra – 执行过程



源点	终点	最短路径	路径长度
A	B	A → B	10
	C	A → D → C	50
	D	A → D	30
	E	A → D → C → E	60

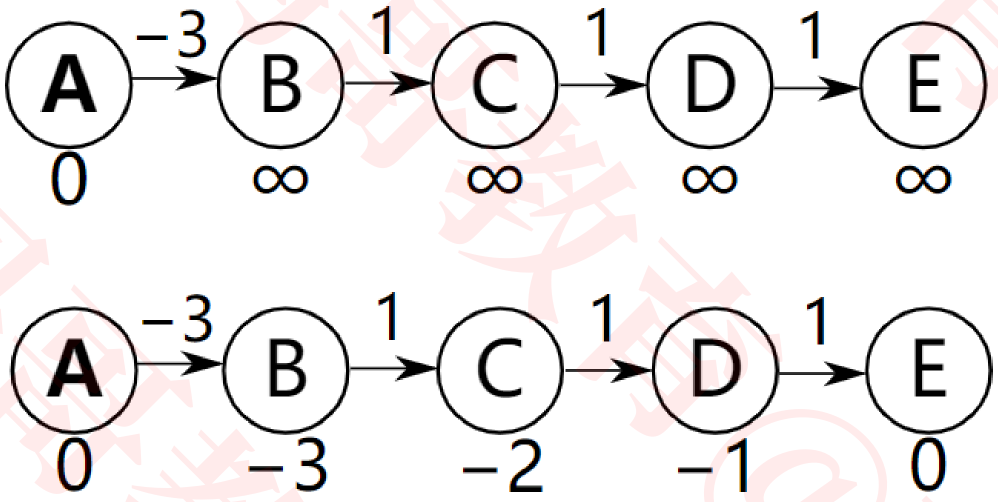


源点	终点	最短路径	路径长度
A	B	A → B	10
	C	A → D → C	50
	D	A → D	30
	E	A → D → C → E	60

Bellman-Ford

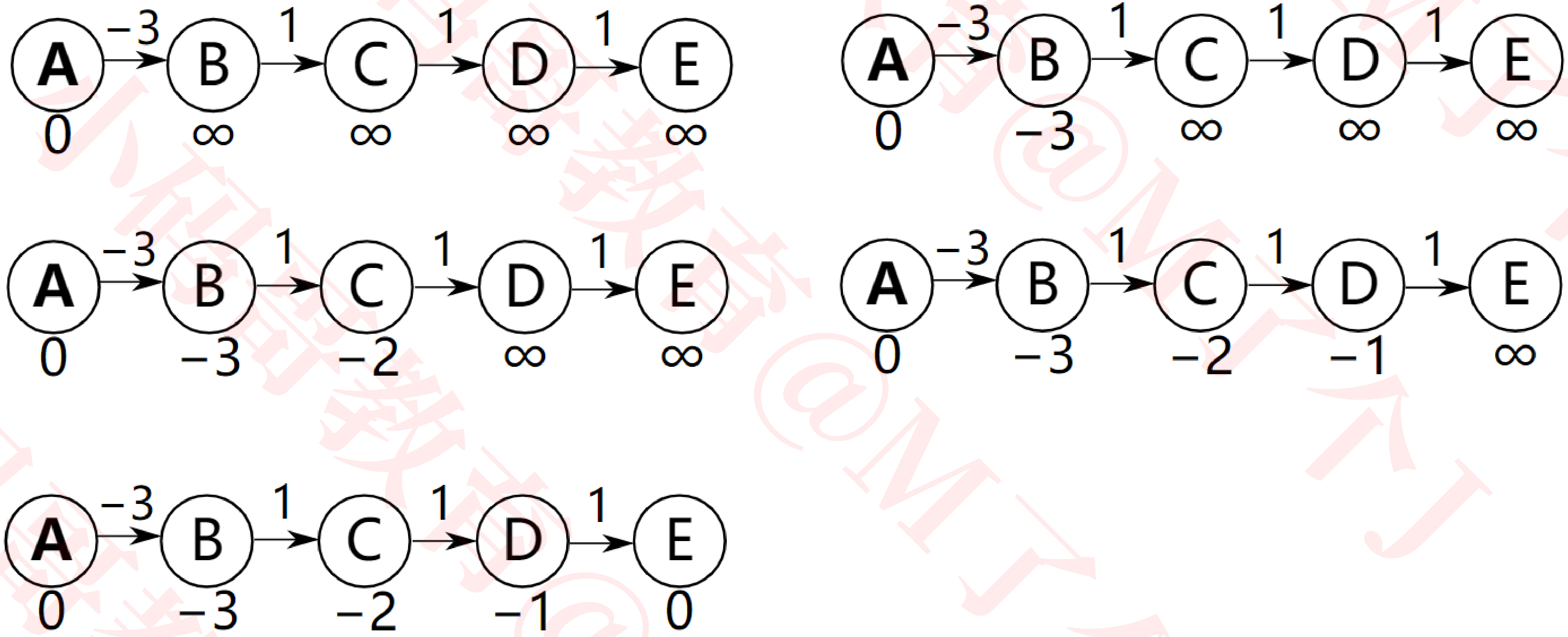
- Bellman-Ford 也属于单源最短路径算法，支持负权边，还能检测出是否有负权环
  - 算法原理：对所有的边进行 V – 1 次松弛操作（ V 是节点数量），得到所有可能的最短路径
  - 时间复杂度：O(EV)，E 是边数量，V 是节点数量

- 下图的最好情况是恰好从左到右的顺序对边进行松弛操作
  - 对所有边仅需进行 1 次松弛操作就能计算出A到达其他所有顶点的最短路径

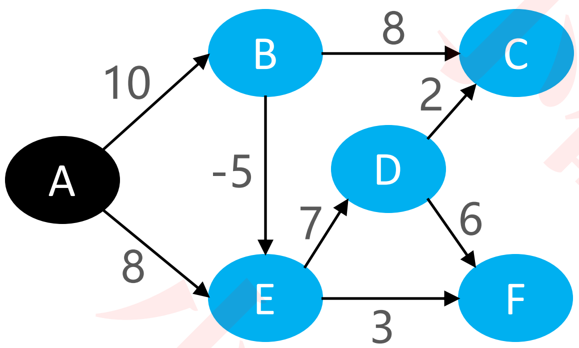


Bellman-Ford

- 最坏情况是恰好每次都从右到左的顺序对边进行松弛操作
  - 对所有边需进行 V – 1 次松弛操作才能计算出A到达其他所有顶点的最短路径



Bellman-Ford – 实例

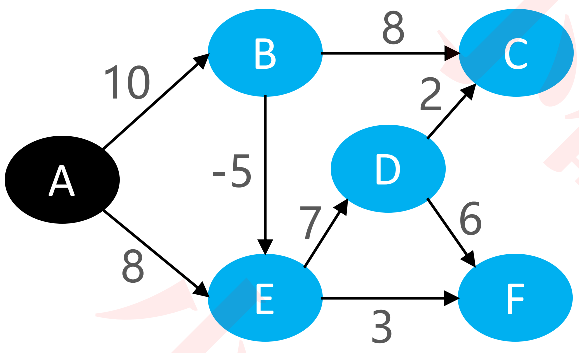


- 一共8条边
- 假设每次松弛操作的顺序是：DC、DF、BC、ED、EF、BE、AE、AB

第1次松弛操作			
源点	终点	最短路径	路径长度
A	B	A → B	10
	C		∞
	D		∞
	E	A → E	8
	F		∞

第2次松弛操作			
源点	终点	最短路径	路径长度
A	B	A → B	10
	C	A → B → C	18
	D	A → E → D	15
	E	A → B → E	5
	F	A → E → F	11

Bellman-Ford – 实例

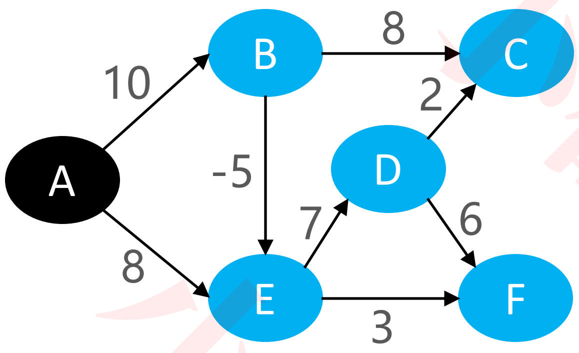


- 每次松弛操作的顺序是：DC、DF、BC、ED、EF、BE、AE、AB

第2次松弛操作			
源点	终点	最短路径	路径长度
A	B	A → B	10
	C	A → B → C	18
	D	A → E → D	15
	E	A → B → E	5
	F	A → E → F	11

第3次松弛操作			
源点	终点	最短路径	路径长度
A	B	A → B	10
	C	A → E → D→ C	17
	D	A → B → E → D	12
	E	A → B → E	5
	F	A → B → E → F	8

Bellman-Ford – 实例



■ 每次松弛操作的顺序是：DC、DF、BC、ED、EF、BE、AE、AB

第3次松弛操作			
源点	终点	最短路径	路径长度
A	B	A → B	10
	C	A → E → D→ C	17
	D	A → B → E → D	12
	E	A → B → E	5
	F	A → B → E → F	8

第4次松弛操作			
源点	终点	最短路径	路径长度
A	B	A → B	10
	C	A → B → E → D → C	14
	D	A → B → E → D	12
	E	A → B → E	5
	F	A → B → E → F	8

■ 不难分析出，经过4次松弛操作之后，已经计算出了A到其他所有顶点的最短路径

Floyd

- ■ Floyd 属于多源最短路径算法，能够求出任意2个顶点之间的最短路径，支持负权边
- □ 时间复杂度：$O(V^3)$，效率比执行 V 次 Dijkstra 算法要好（ V 是顶点数量）

```
for (int k = 0; k < V; k++) {
  for (int i = 0; i < V; i++) {
    for (int j = 0; j < V; j++) {
      if (dist(i, k) + dist(k, j) < dist(i, j)) {
        dist(i, j) = dist(i, k) + dist(k, j);
      }
    }
  }
}
```

- ■ 算法原理
- □ 从任意顶点 i 到任意顶点 j 的最短路径不外乎两种可能

① 直接从 i 到 j

② 从 i 经过若干个顶点到 j

- □ 假设 dist(i，j) 为顶点 i 到顶点 j 的最短路径的距离
- □ 对于每一个顶点 k，检查 dist(i，k) + dist(k，j) < dist(i，j) 是否成立

✓ 如果成立，证明从 i 到 k 再到 j 的路径比 i 直接到 j 的路径短，设置 dist(i，j) = dist(i，k) + dist(k，j)

✓ 当我们遍历完所有结点 k，dist(i，j) 中记录的便是 i 到 j 的最短路径的距离