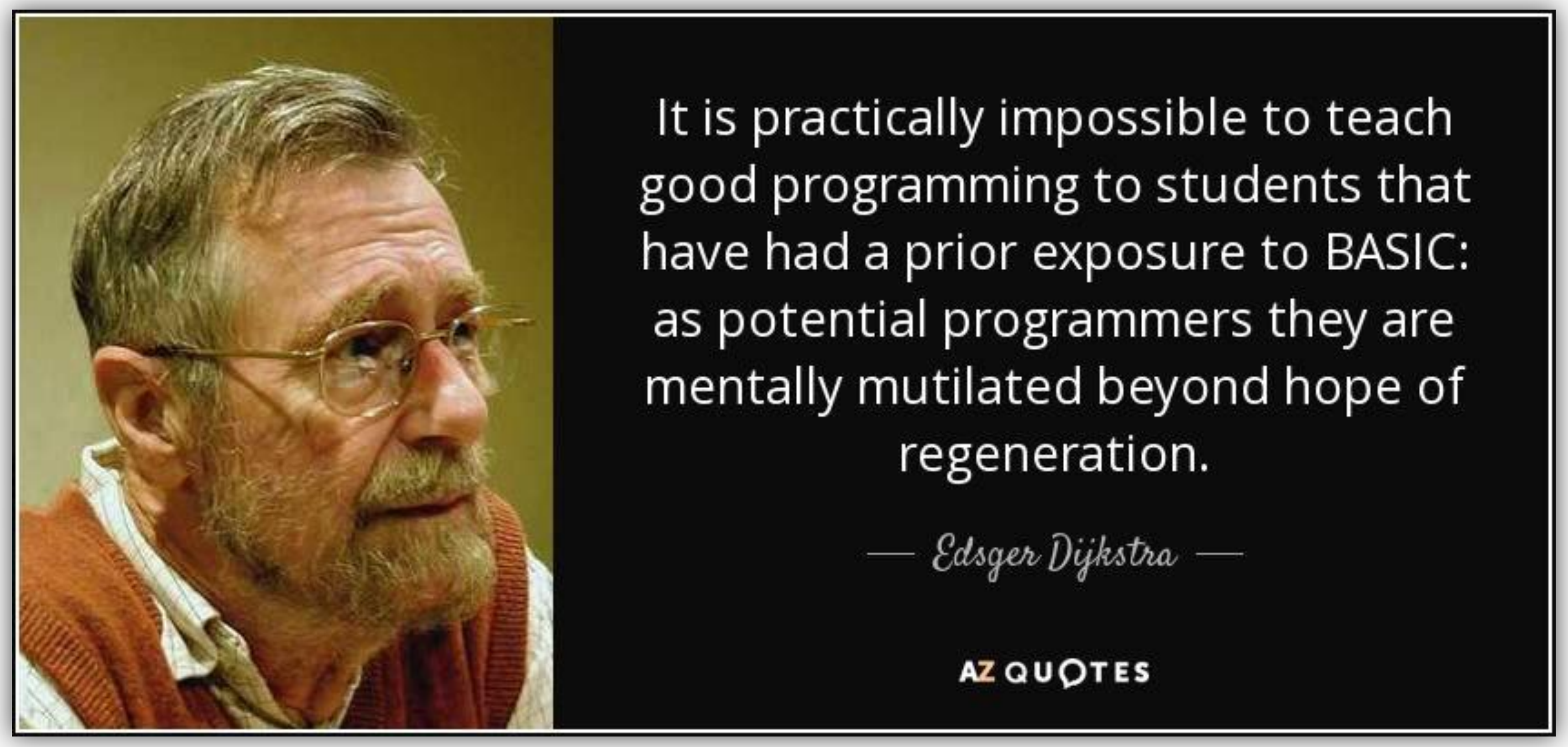
单源最短路径（ Dijkstra’s Algorithm ）



Edsger Wybe Dijkstra（1972年图灵奖）

单源最短路径算法：Dijkstra’s Algorithm

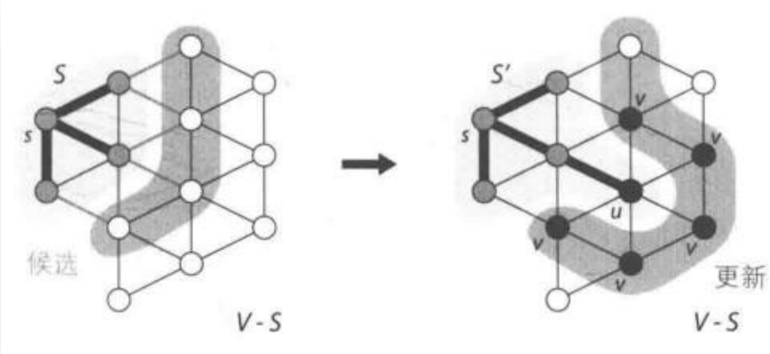
设图 $G = (V, E)$ 所有顶点的集合为 V，起点为 s，最短路径树中包含的顶点集合为 S。在各计算步骤中，我们将选出最短路径树的边和顶点并将其添加至 S。

对于各顶点 i，设仅经由 S 内顶点的 s 到 i 的最短路径成本为 $d[i]$，i 在最短路径树中的父结点为 $p[i]$。

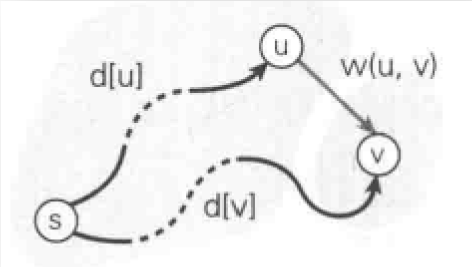
1. 初始状态下将 S 置空。
   初始化 s 的 $d[s] = 0$；除 s 以外，所有属于 V 的顶点 i 的 $d[i] = \infty$。
2. 循环进行下述处理，直至 $S = V$ 为止。
   - ▶ 从 $V - S$ 中选出 $d[u]$ 最小的顶点 u
   - ▶ 将 u 添加至 S，同时将与 u 相邻且属于 $V - S$ 的所有顶点 v 的值按照下述方式更新

$$
\begin{aligned}
&\text{if } d[u] + w(u, v) < d[v] \\
&\quad d[v] = d[u] + w(u, v) \\
&\quad p[v] = u
\end{aligned}
$$

在步骤 2 的各处理执行结束后（即选择下一个 u 之前），$d[v]$ 中记录着从 s 出发，仅经由 S 内顶点抵达 v 的最短路径成本。也就是说，当所有处理进行完毕后，V 中所有顶点的 $d[v]$ 都记录着 s 到 v 的最短路径成本（距离）。



最短路径树的生成



最短路径成本的更新

单源最短路径算法：Dijkstra’s Algorithm

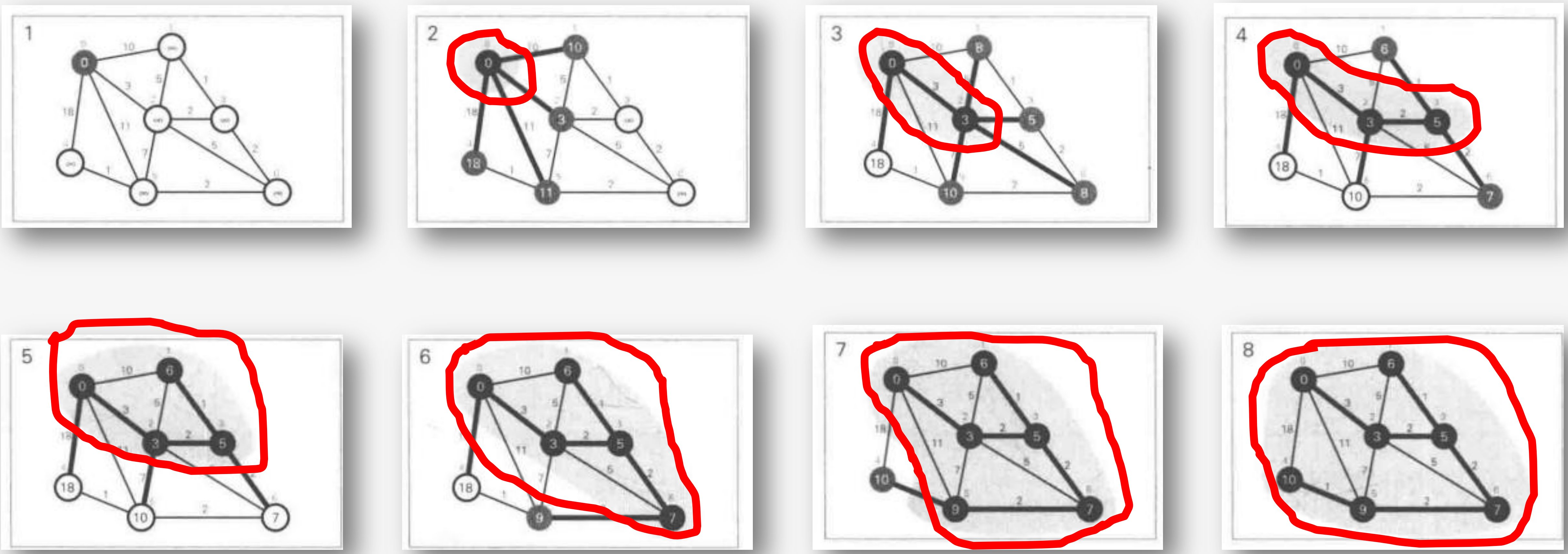
用邻接矩阵实现狄克斯特拉算法时，需要用到下列变量。这里的 $n = |V|$。

color[n]	color[v] 用于记录 *v* 的访问状态 WHITE、GRAY、BLACK
M[n][n]	邻接矩阵，M[u][v] 中记录 *u* 到 *v* 的边的权值
d[n]	d[v] 用于记录起点 *s* 到 *v* 的最短路径成本
p[n]	p[v] 用于记录顶点 *v* 在最短路径树中的父结点

单源最短路径算法：Dijkstra’s Algorithm

```
dijkstra(s)
  将所有顶点 u 的 color[u] 设为 WHITE, d[u] 初始化为 INFTY
  d[s] = 0
  p[s] = -1

  while true
    mincost = INFTY
    for i 从 0 至 n-1
      if color[i] != BLACK && d[i] < mincost
        mincost = d[i]
        u = i

    if mincost == INFTY
      break

    color[u] = BLACK

    for v 从 0 至 n-1
      if color[v] != BLACK 且 u 和 v 之间存在边
        if d[u] + M[u][v] < d[v]
          d[v] = d[u] + M[u][v]
          p[v] = u
          color[v] = GRAY
```

单源最短路径算法：Dijkstra’s Algorithm



单源最短路径算法：Dijkstra’s Algorithm

求单源点最短路径**Dijkstra算法**是一种基于**贪心策略**的算法。 每次新扩展一个路程最短的点，更新与其相邻的点的路程。当所有边权为正时，由于不会存在一个路程更短的没扩展过的点，所以这个点的路程永远不会再改变，因而保证了算法的正确性。不过根据这个原理，用本算法求最短路径的图是**不能有负权边**的，因为扩展到负权边的时候会产生更短的路程，有可能就破坏了已经更新的点路程不会改变的性质。

单源最短路径算法：Dijkstra’s Algorithm

邻接矩阵存图——算法复杂度为 $O(|V|^2)$

注意——不可以应用于包含负权值的图!

单源最短路径算法（优化）：Dijkstra’s Algorithm

优化思路：采用邻接表存图 + 二叉堆（ 优先级队列 ）

单源最短路径算法（优化）：Dijkstra’s Algorithm

狄克斯特拉算法（优先级队列）

设图 $G=(V, E)$ 所有顶点的集合为 V，起点为 s，最短路径树中包含的顶点集合为 S。在各计算步骤中，我们将选出最短路径树的边和顶点并将其添加至 S。

对于各顶点 i，设仅经由 S 内顶点的 s 到 i 的最短路径成本为 $d[i]$，i 在最短路径树中的父结点为 $p[i]$。

1. 初始状态下将 S 置空。
   初始化 s 的 $d[s]=0$；除 s 以外，所有属于 V 的顶点 i 的 $d[i]=\infty$。
   以 $d[i]$ 为键值，将 V 的顶点构建成最小堆 H。
2. 循环进行下述处理，直至 $S=V$ 为止。
   - ▶ 从 H 中取出 $d[u]$ 最小的顶点 u
   - ▶ 将 u 添加至 S，同时将与 u 相邻且属于 $V-S$ 的所有顶点 v 的值按照下述方式更新

     if $d[u]+w(u, v)<d[v]$
       $d[v]=d[u]+w(u, v)$
       $p[v]=u$
       以 v 为起点更新堆 H。

单源最短路径算法（优化）：Dijkstra’s Algorithm

```
dijkstra(s)
  将所有顶点 u 的 color[u] 设为 WHITE, d[u] 初始化为 INFTY
  d[s] = 0

  Heap heap = Heap( n, d )
  heap.construct()

  while heap.size >= 1
    u = heap.extractMin()

    color[u] = BLACK

    // 如果仍存在与 u 相邻的顶点 v
    while ( v = next( u ) ) != NIL
      if color[v] != BLACK
        if d[u] + M[u][v] < d[v]
          d[v] = d[u] + M[u][v]
          color[v] = GRAY
          heap.update( v )
```

用堆实现狄克斯特拉算法

单源最短路径算法（优化）：Dijkstra’s Algorithm

```
dijkstra(s)
  将所有顶点 u 的 color[u] 设为 WHITE，d[u] 初始化为 INFTY
  d[s] = 0

  PQ.push( Node( s, 0 ) ) // 将起点插入优先级队列
  // 选 s 作为最开始的 u

  while PQ 不为空
    u = PQ.extractMin()

    color[u] = BLACK

    if d[u] < u 的成本 // 取出最小值，如果不是最短路径则忽略
      continue

    // 如果仍存在与 u 相邻的顶点 v
    while ( v = next( u ) ) != NIL
      if color[v] != BLACK
        if d[u] + M[u][v] < d[v]
          d[v] = d[u] + M[u][v]
          color[v] = GRAY
          PQ.push( Node( v, d[v] ) )
```

用优先级队列实现狄克斯特拉算法

单源最短路径算法（优化）：Dijkstra’s Algorithm

在使用**邻接表和二叉堆**实现的狄克斯特拉算法中， 从二叉堆取出顶点 u 需要消耗 **O(|V|log|V|)**，更新 d[v] 又需要消耗 **O(|E|log|V|)**，因此整体算法复杂度为 **O((|V| + |E|)log|V|)**。

另外，用**优先级队列替代二叉堆**的实现方法中，需要从队列中取出 |V| 次顶点，向队列执行|E|次插入操作，所以算法复杂度同为 **O((|V| + |E|)log|V|)**。