Note 13 Graph Algorithm II

Disjoint Set, MST, Single Source Shortest Path

I. Disjoint Set 并查集

1. Basic Operations:

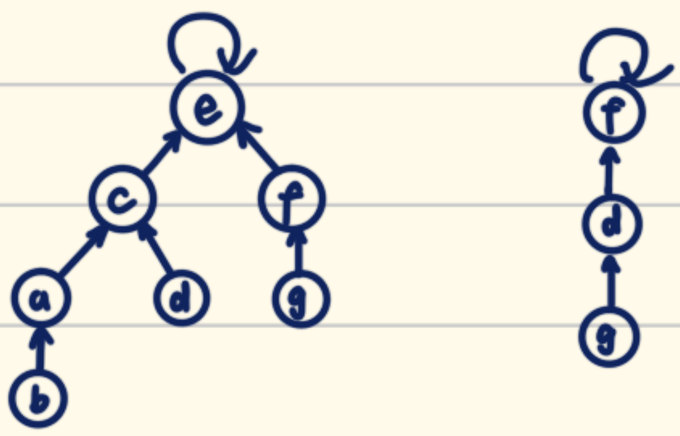
① MAKE-SET(x): build a new set, only contains x.

② UNION(x,y): union the sets that contain x and y (Sx and Sy) into a new set

③ FIND-SET(x): return the pointer that symbolizes the set that contains x.

2. Disjoint Set Forests:

① Representation of a set

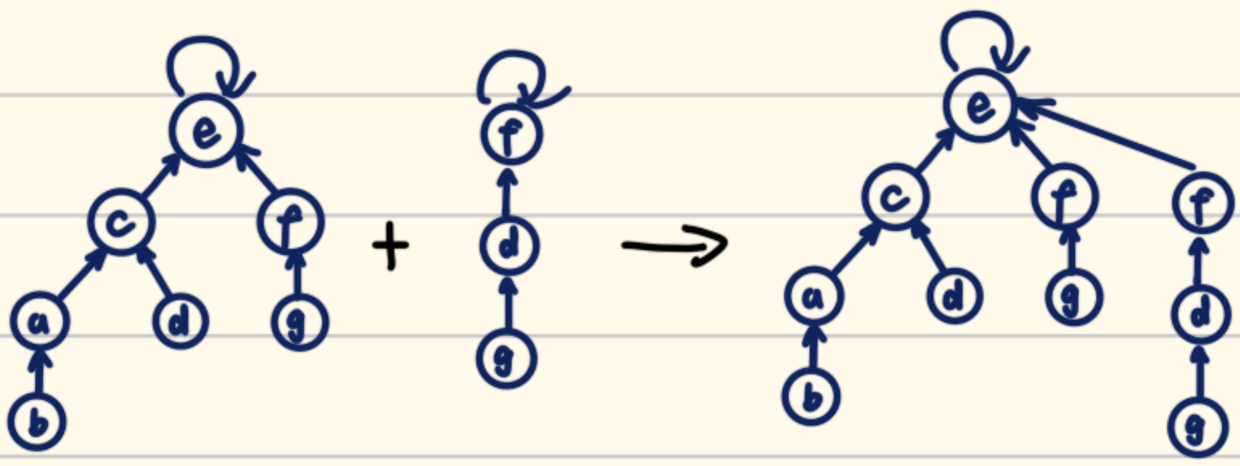


MAKE-SET(x)
1. $p[x] \leftarrow x$
2. $rank[x] \leftarrow 0$

- the root of each set represents the set. It's the father of itself.

② union by rank (in UNION(x,y))

- method 1: attach the tree with fewer nodes to the other with more nodes.
- method 2: attach the tree shorter (smaller rank) to the other higher.



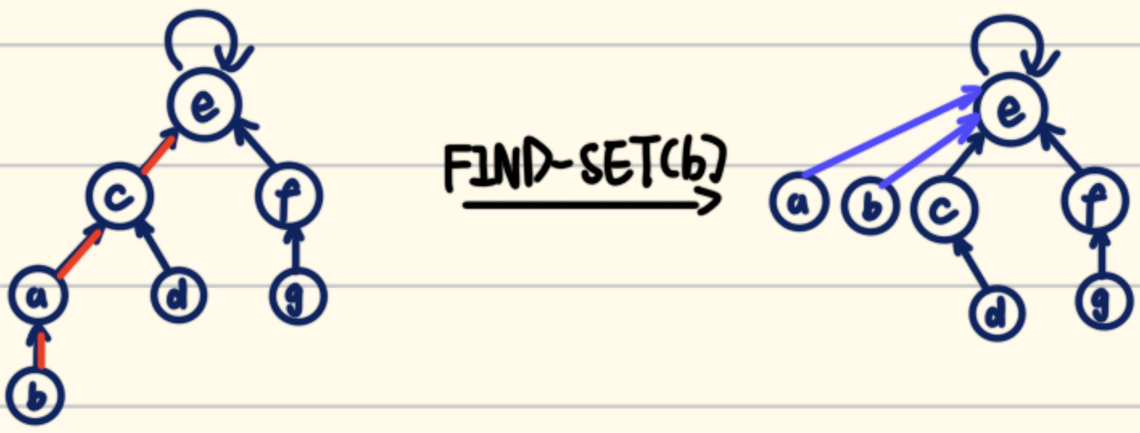
UNION(x, y)
1. LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
1. **if** $rank[x] > rank[y]$
2. **then** $p[y] \leftarrow x$
3. **else** $p[x] \leftarrow y$
4. **if** $rank[x] = rank[y]$
5. **then** $rank[y] \leftarrow rank[y] + 1$

when equally high, update the height

③ path compression (in FIND-SET(x))

When operating FIND-SET(x), directly connect all nodes on the path from the node to the root to the root.



FIND-SET(x)
1. **if** $x \neq p[x]$
2. **then** $p[x] \leftarrow$ FIND-SET$(p[x])$
3. **return** $p[x]$

compress the path when recursing back

3. Time Complexity Analysis

Using Amortized Analysis, we can get the time complexities of 3 operations above (When implementing set in "Disjoint Set Forest")

- MAKESET: O(1)
- FINDSET: O(α(N))
- UNION: O(α(N))

$$\alpha = \begin{cases} 0, & 0 \le n \le 2 \\ 1, & n = 3 \\ 2, & 4 \le n \le 7 \\ 3, & 8 \le n \le 2047 \\ 4, & 2047 \le n \le A_4(1) >> 10^{80} \end{cases}$$

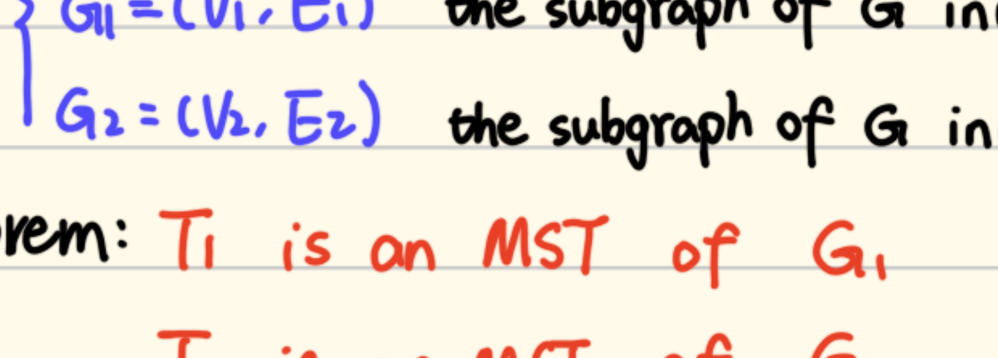
$$A_k(j) = \begin{cases} j+1 & \text{if } k=0 \\ A_{k-1}^{(j+1)}(j) & \text{if } k \ge 1 \end{cases}$$

II. Minimum Spanning Tree

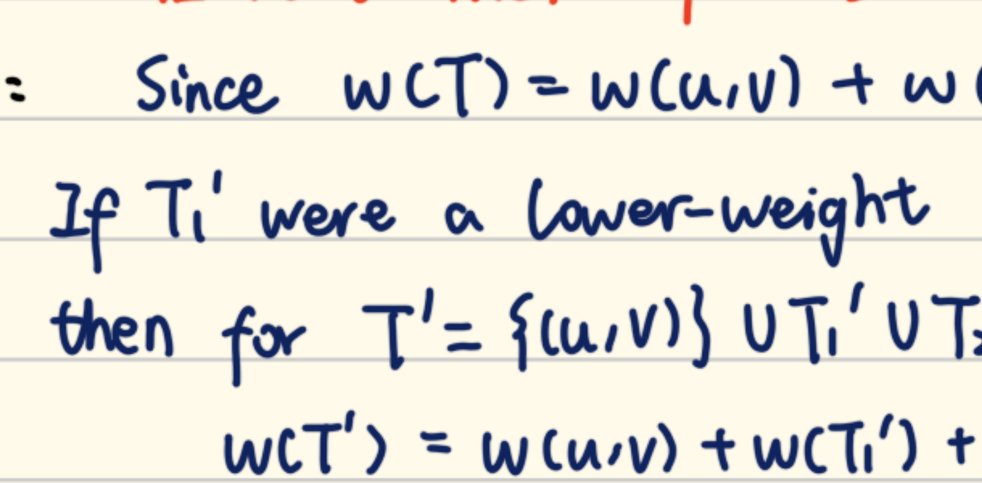
(I) Problem Analysis

1. Optimal Substructure (Idea of Dynamic Programming)

Minimum spanning tree T of G(V, E)



↓ Remove (u, v)



T is partitioned into subtrees T_1 & T_2

Let $\begin{cases} G_1 = (V_1, E_1) & \text{the subgraph of } G \text{ induced by the vertices of } T_1 \\ G_2 = (V_2, E_2) & \text{the subgraph of } G \text{ induced by the vertices of } T_2 \end{cases}$

- Theorem: T_1 is an MST of G_1

  T_2 is an MST of G_2

- Proof: Since $w(T) = w(u,v) + w(T_1) + w(T_2)$

  If T_1' were a lower-weight spanning tree than T_1 for G_1

  then for $T' = \{(u,v)\} \cup T_1' \cup T_2$:

  $$w(T') = w(u,v) + w(T_1') + w(T_2)$$

  $$\le w(u,v) + w(T_1) + w(T_2) = w(T)$$

  This is not right. ∴ T_1 is MST of G_1, T_2 is MST of G_2.

子问题最优
子问题重叠
⇒ 可用动态规划

↪ A locally optimal choice is globally optimal.

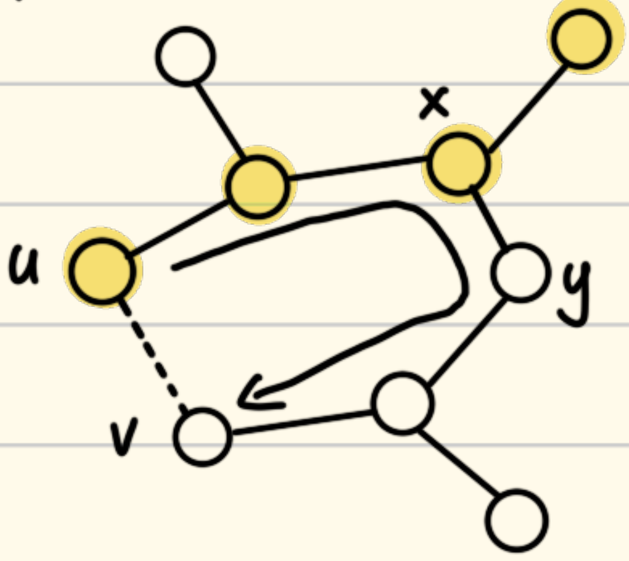
2. Greedy Analysis (Idea of Greedy Algorithm)

- Theorem : Let T be the MST of $G=(V,E)$, let $A \subseteq V$

  Suppose that $(u,v) \in E$ is the least-weight edge connecting A to $V-A$

  Then $(u,v) \in T$ (T is one of the MST if there 're several)

- Proof :



○ : $\in A$ ○ : $\in V-A$

(假定所有MST不含(u,v))

∵ u,v 分别在 A 与 $V-A$

∴ T 中必有另一边 (x,y) 横跨切割, $x \in A$, $y \in V-A$

$$w(x,y) > w(u,v)$$

将 (x,y) 删除, 增加 (u,v), 构造新的生成树 T'

$$T' = T - \{(x,y)\} \cup \{(u,v)\}$$

$$w(T') = w(T) - w(x,y) + w(u,v) < w(T)$$

∵ T' 是最小生成树 ∴ $w(T) \le w(T')$ ∴ 矛盾, 假设不成立

∴ $(u,v) \in$ one of the MST

(II). Kruskal Algorithm

1. 对所有边按照权重从小到大依次排序

   每次找到权重最小且连接两个同集合的边 (u,v), 放入生成树

```
MST-KRUSKAL(G, w)
1.  A ← ∅
2.  for each vertex v ∈ V[G]
3.      do MAKE-SET(v)
4.  sort the edges of E into nondecreasing order by weight w.
5.  for each edge (u, v) ∈ E, taken in nondecreasing order by
    weight.
6.      do if FIND-SET(u) ≠ FIND-SET(v)
7.          then A ← A ∪ {(u, v)}
8.               UNION(u, v)
9.  return A
```

2. Time Complexity :

- Sort Edges in Line 4 : $O(E \lg E)$
- Line 5-8 : $O(V)$ 个 MAKESET, $O(E)$ 个 FIND-SET 和 UNION

  $O((V+E)\alpha(V))$

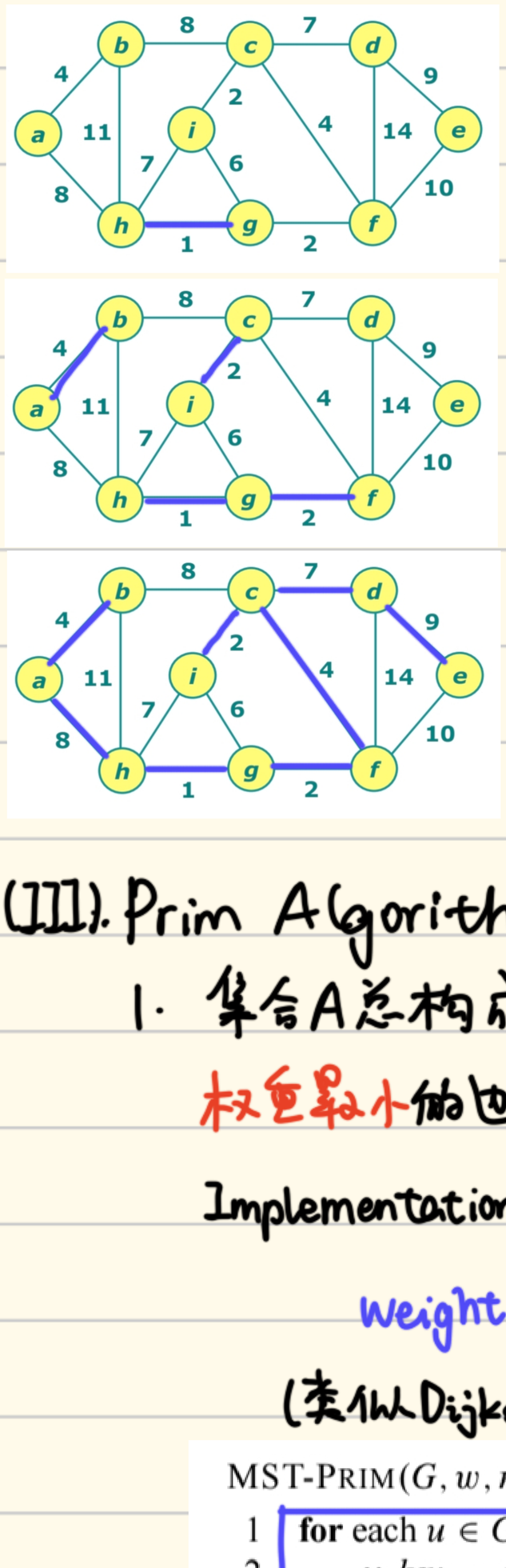
  由于连通 : $|V|-1 \le |E| \le |V|^2$

$\Rightarrow \quad O(\lg V) = O(\lg E),\ O(V) \le O(E)$

$O((V+E)\alpha(V)) \le O((V+E)\lg E) \le O(E\lg E) = O(E\lg V)$

$\Rightarrow$ Time Complexity: $O(E\lg V)$

3. Example



(III). Prim Algorithm

1. 集合A总构成一棵树，算法每次从连接 A 与 $\bar{A}$ 的所有边中，选择一条权重最小的边加入A中

Implementation Trick: Maintain V-A as a priority queue Q with the weight of the least weight edge connecting it to a vertex in A.

(类似Dijkstra实现的想法，在类似BFS的操作中不断行更新)

```
MST-PRIM(G, w, r)
1  for each u ∈ G.V                    → least weight edge connecting to A
2      u.key = ∞
3      u.π = NIL                       → 前继节点
4  r.key = 0
5  Q = G.V
6  while Q ≠ ∅
7      u = EXTRACT-MIN(Q)
8      for each v ∈ G.Adj[u]
9          if v ∈ Q and w(u, v) < v.key
10             v.π = u
11             v.key = w(u, v)         → update the weight and predecessor
```

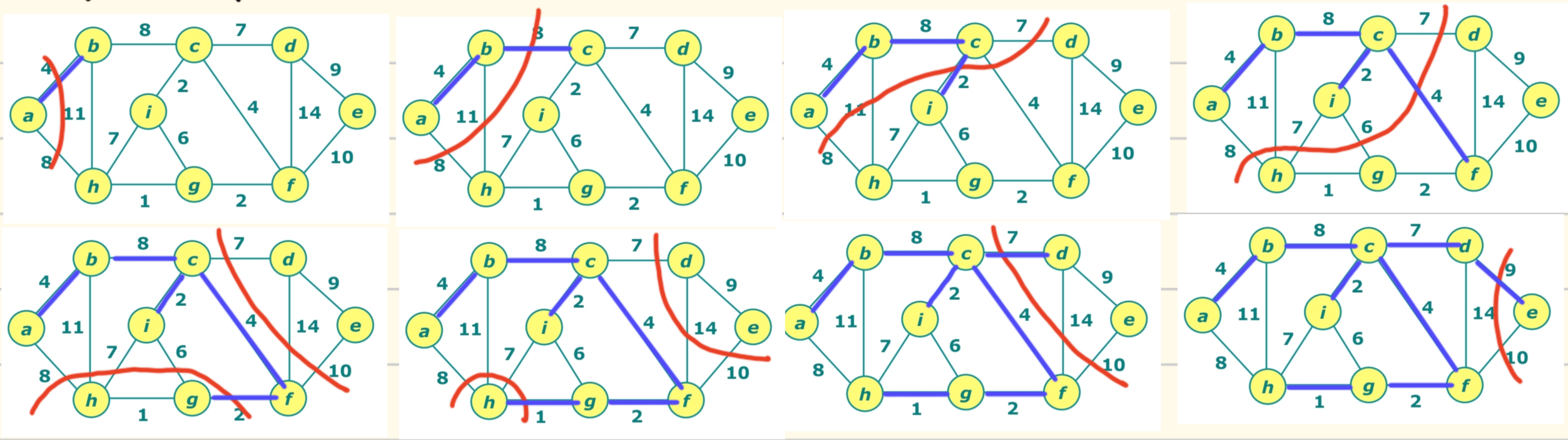
2. Time Complexity

Line 6-7: Extract-Min: $O(V\lg V)$ (each pop of a heap is $O(\lg V)$)

E次边，可能有 decrease-key 操作 ($O(E\lg V)$)

$\Rightarrow O(V\lg V + E\lg V) = O(E\lg V)$

3. Example



II. Single Source Shortest Paths

(I) Problem Analysis:

- Dynamic Programming:

  最短路径中的一部分也是从某出发点至终点的最短路

- Greedy:

  可以每次选 V-S 中最近/最轻的结点加入

(II) Dijkstra's algorithm

1. Algorithm

(1) Maintain a set S of vertices whose shortest path distances from s are unknown

(2) At each step add to S the vertex $v \in V-S$ whose distance estimate from s is minimal

(3) Update the distance estimates of vertices adjacent to v.

The idea of Dijkstra is similar to Prim, but kind of different

DIJKSTRA(G, w, s)

```
1.  for each vertex v ∈ V[G]
2.       do d[v] ← ∞
3.          π(v) ← NIL
4.  d[s] ← 0
5.  S ← ∅
6.  Q ← V[G]
7.  while Q ≠ ∅
8.        do u ← EXTRACT-MIN(Q)
9.           S ← S ∪ {u}
10.          for each vertex v ∈ Adj[u]
11.               do if d[v] > d[u] + w(u, v)
12.                  then d[v] ← d[u] + w(u, v)
13.                       π(v) ← u
```

Relaxation step.
Implicit DECREASE-KEY.

2. Time Complexity

从fringe里面 extract-min 了 V次，每次O(lgV) ⇒ O(VlgV)

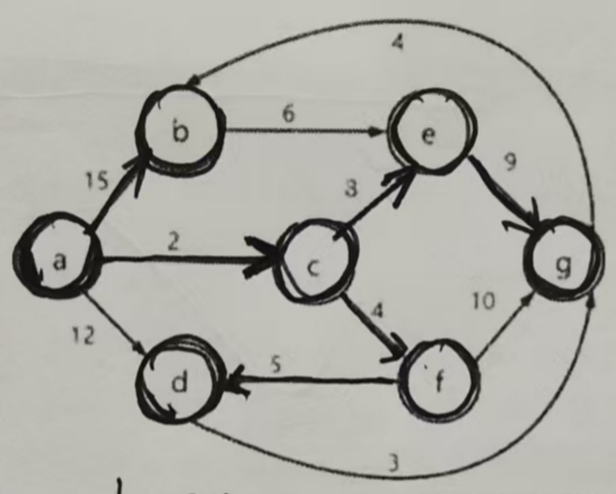
E条边，每次relax 时可能 decrease key，每次 O(lgV) ⇒ O(ElgV)

∵ O(V) ≤ O(E) (假设是连通图)，O(lgV) = O(lgE)

∴ Time complexity: O((V+E)lgV) = O(ElgV)

3. Example:

4、 写出用 Dijkstra 算法求的的下图 G 中顶点 a 到其他各顶点的最短路径，并给出执行过程表：



fringe: a:0

visited: {}

↓

fringe: c:2 | d:12 | b:15

visited: {a:0}

↓

fringe: e:5 | f:6 | d:12 | b:15

visited: {a:0, c:2}

↓

fringe: f:6 | d:12 | g:14 | b:15

visited: {a:0, c:2, e:5}

↓

fringe: d:11 | g:14 | b:15

visited: {a:0, c:2, e:5, f:6}

↓

fringe: g:14 | b:15

visited: {a:0, c:2, e:5, f:6, d:11}

↓

fringe: b:15

visited: {a:0, c:2, e:5, f:6, d:11, g:14}

↓

fringe: 空

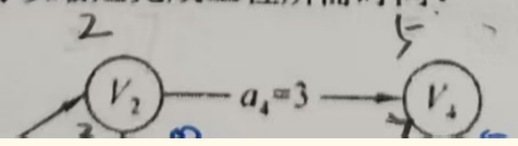
visited: {a:0, c:2, e:5, f:6, d:11, g:14, b:15}

5、 下图所示为一个用 AOE 网表示的工程，试回答：

Activity On Edges

(1) 完成此工程至少需要多少时间？

(2) 那些活动加速可以缩短完成工程所需时间？



4. Condition:

When using Dijkstra, there shouldn't be negative-weighted edges.

(III) Shortest Paths in Weighted DAG

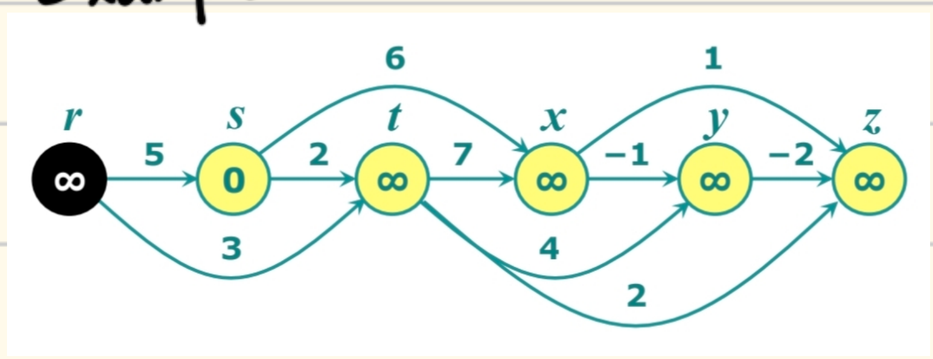
1. Algorithm

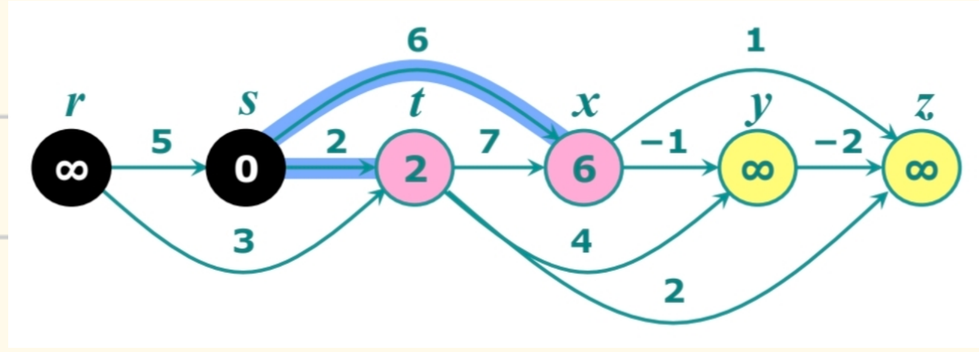
DAG-SHORTEST-PATH(G, w, s)

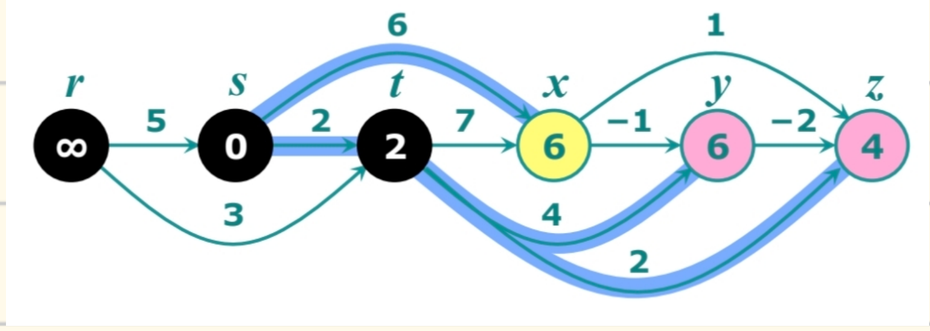
1. Topologically sort the vertices of G
2. **for** each vertex $v \in V[G]$
3. **do** $d[v] \leftarrow \infty$
4. $\pi(v) \leftarrow$ NIL
5. $d[s] \leftarrow 0$
6. **for** each vertex u, taken in topologically sorted order
7. **do for** each vertex $v \in Adj[u]$
8. **do if** $d[v] > d[u] + w(u, v)$
9. **then** $d[v] \leftarrow d[u] + w(u, v)$
10. $\pi(v) \leftarrow u$

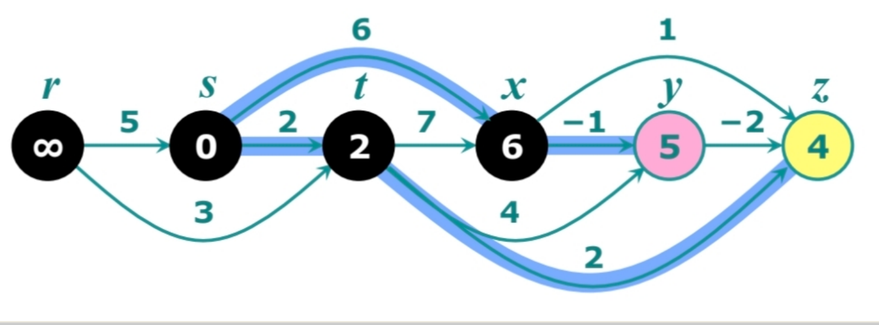
按拓扑排序顺序进行relax，无需再维护一个fringe

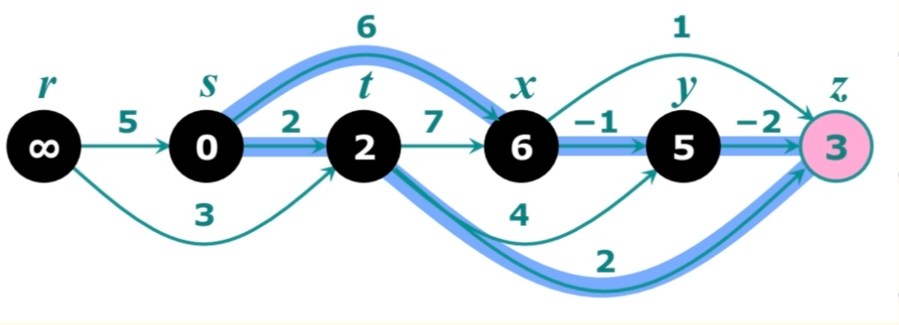
2. Example:

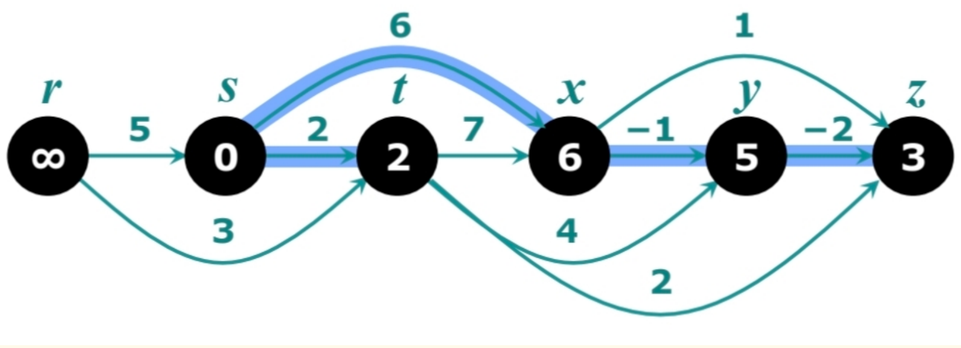












(IV.) Bellman - Ford Algorithm

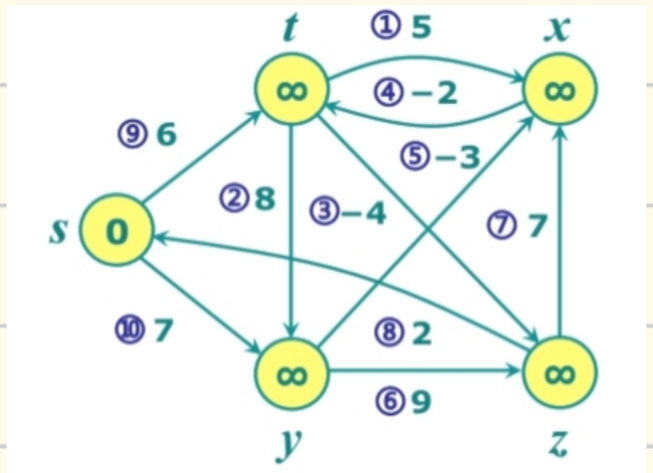
1. Algorithm

```
BELLMAN-FORD(G, w, s)
1.  for each vertex v ∈ V[G]
2.       do d[v] ← ∞
3.          π(v) ← NIL
4.  d[s] ← 0
5.  for i ← 1 to |V[G]| − 1
6.       do for each edge (u, v) ∈ E[G]
7.                do if d[v] > d[u] + w(u, v)
8.                   then d[v] ← d[u] + w(u, v)
9.                        π(v) ← u
10. for each edge (u, v) ∈ E[G]
11.      do if d[v] > d[u] + w(u, v)
12.               then return FALSE
13. return TURE
```

由于最短路一定包括不多于 |V|-1 条边，故只要对每条边做 |V|-1 次 relaxation 就一定能找到最短路（每次都对所有边松弛）

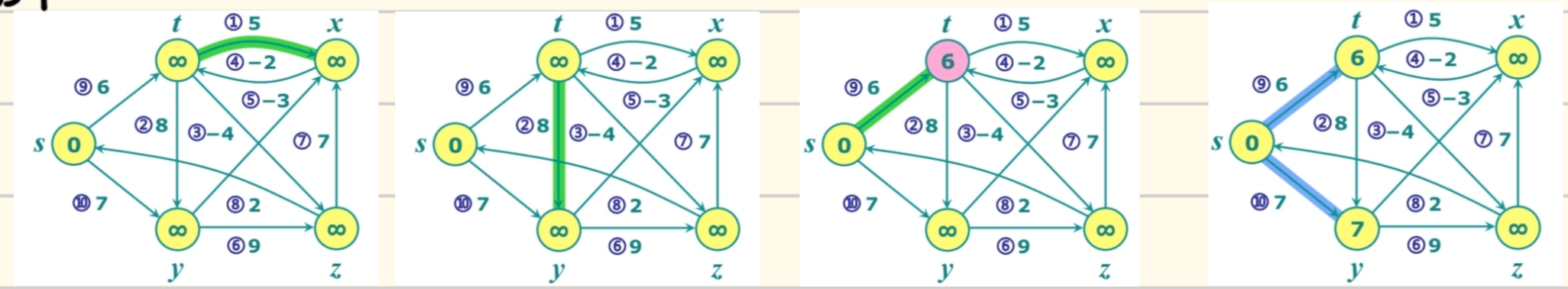
2. Time Complexity: O(VE)

3. Example:

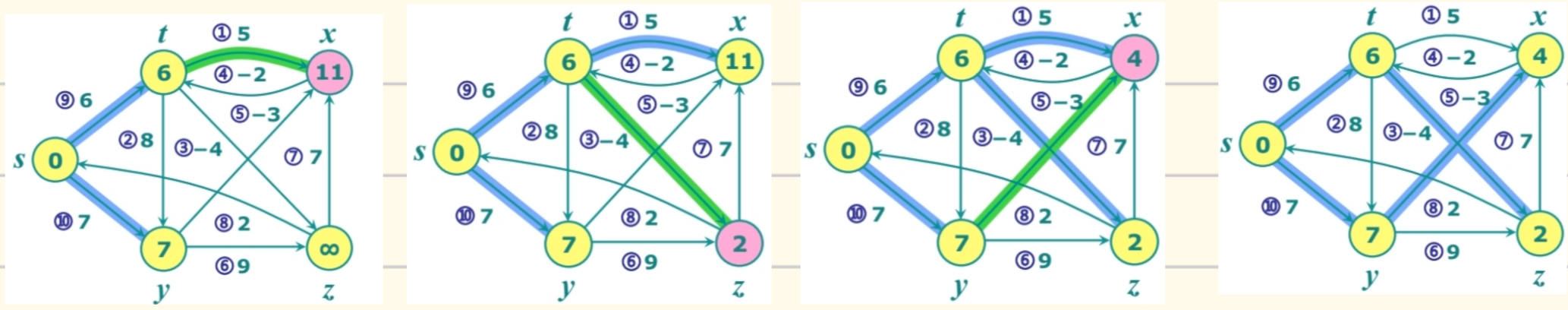


Order of edge relaxation.

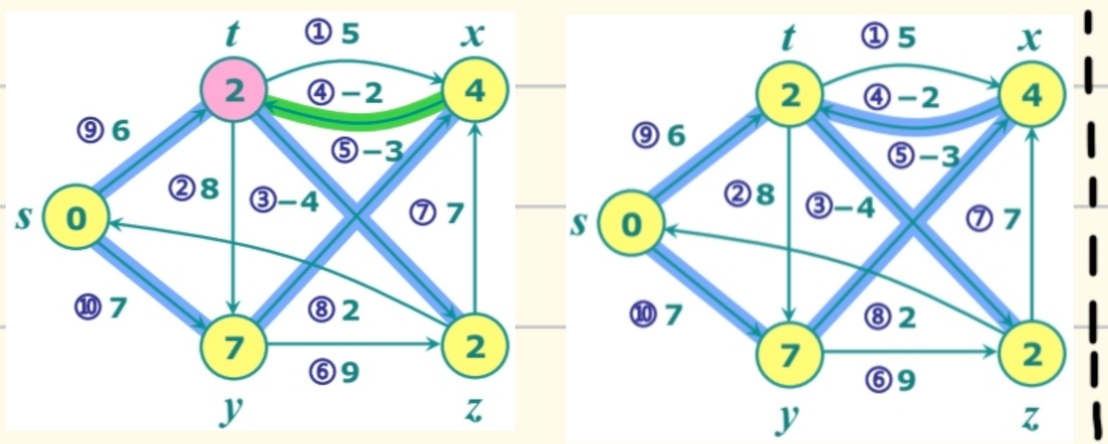
Pass 1:



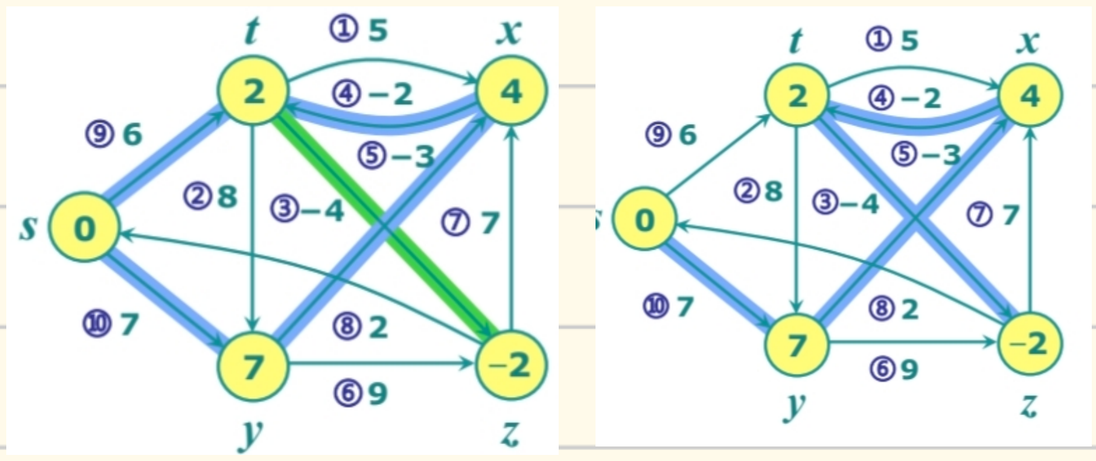
Pass 2:



Pass 3:



Pass 4:



Algorithms	Negative-Weighted Edges	Cycle	Negative-Weighted Cycle	Time Complexity
Dijkstra	X	V	X	O(E lg V)
A*	X	V	X	O(E lg V)
Dag Shortest Path	V	X	X	O(V+E)
Bellman-Ford	V	V	X	O(VE)