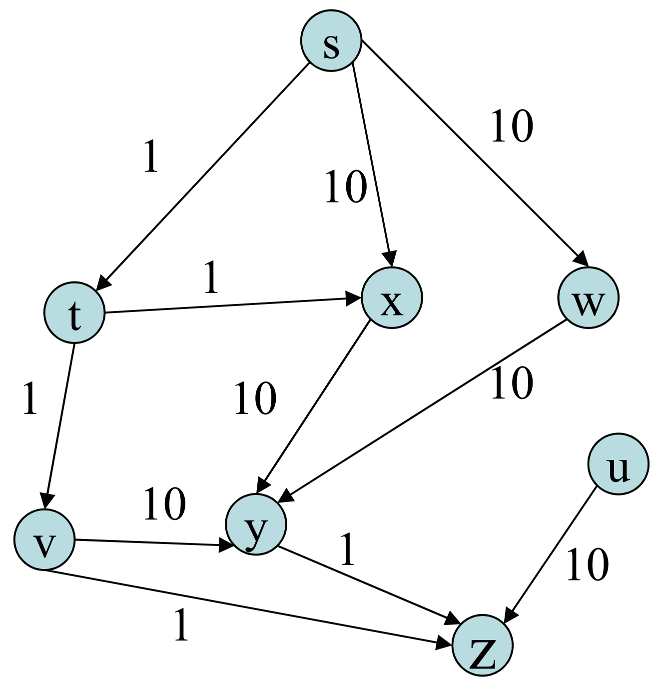
Single-Source Shortest Paths

Shortest-path problem

- 在一圖上找出兩點間最短路徑。
- G=(V, E)是一個Weighted Directed Graph (加權有向圖)
  - V:Vertex (node) 節點
  - E: edge (link) 連線 or arc (有向邊)
  - Weight function w: E➔R界定出每個邊的權重。
- 以p =(v_0,v_1,…,v_k)表一個自v_0到v_k的Path (路徑)。
- Shortest path 最短路徑

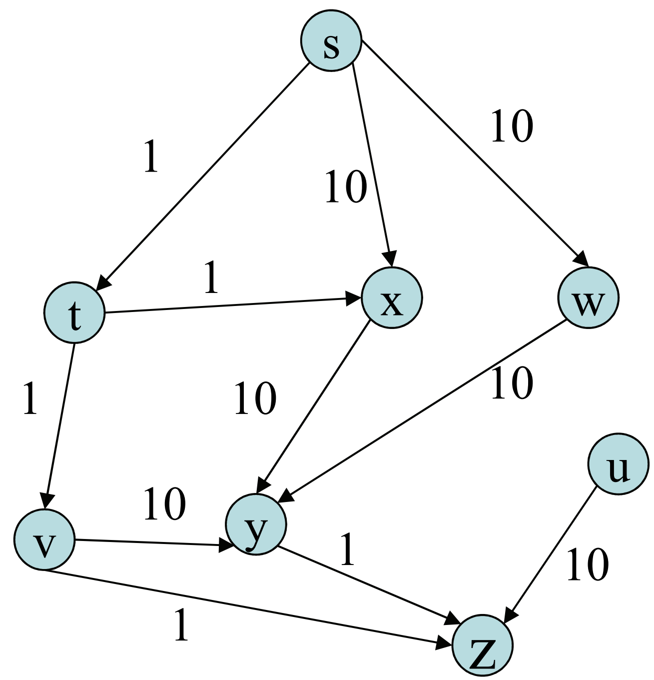


Shortest-path problem

- p =(v_0,v_1,…,v_k)
- 定義路徑長度 weight of path

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

- 定義自u到v的最短距離



$$\delta(u,v) = \begin{cases} \min\{w(p) : u \xrightarrow{p} v\}, \exists \text{a path from } u \text{ to } v. \\ \infty, \text{otherwise.} \end{cases}$$

Shortest-Path Variants

- Shortest-Path problems
  - Single-source shortest-paths problem: Find the shortest path from s to each vertex v. (e.g. BFS)
  - Single-destination shortest-paths problem: Find a shortest path to a given *destination* vertex t from each vertex v.
  - Single-pair shortest-path problem: Find a shortest path from u to v for given vertices u and v.
  - All-pairs shortest-paths problem: Find a shortest path from u to v for every pair of vertices u and v.

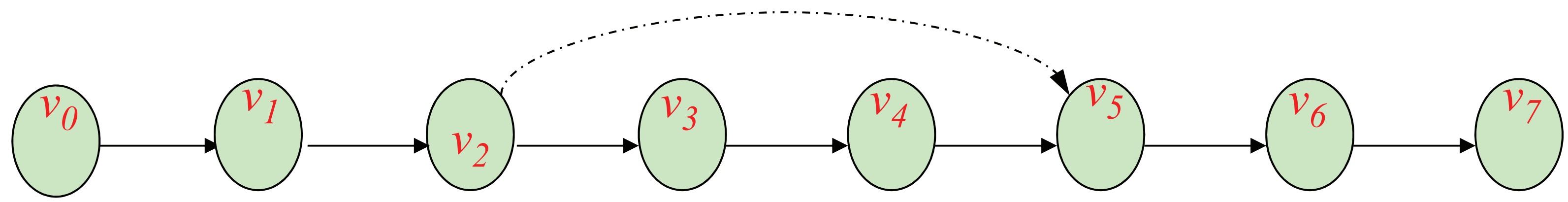
Optimal Substructure Property

Theorem: Subpaths of shortest paths are also shortest paths

- Let $P_{1k} = <v_1, \ldots, v_k>$ be a shortest path from v_1 to v_k
- Let $P_{ij} = <v_i, \ldots, v_j>$ be subpath of P_{1k} from v_i to v_j for any i, j
- Then P_{ij} is a shortest path from v_i to v_j

Optimal Substructure Property

Proof: By cut and paste

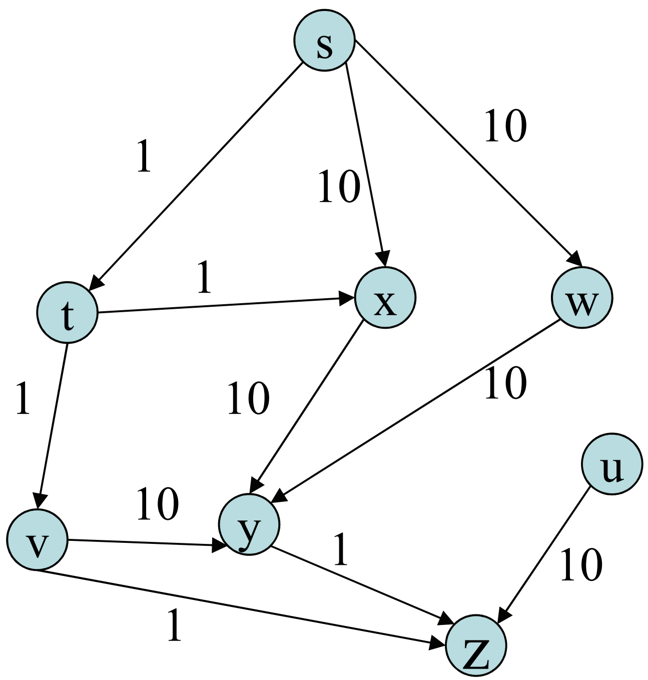


- If some subpath *were not* a shortest path
- We could substitute a shorter subpath to create a *shorter total path*
- Hence, the original path would not be shortest path

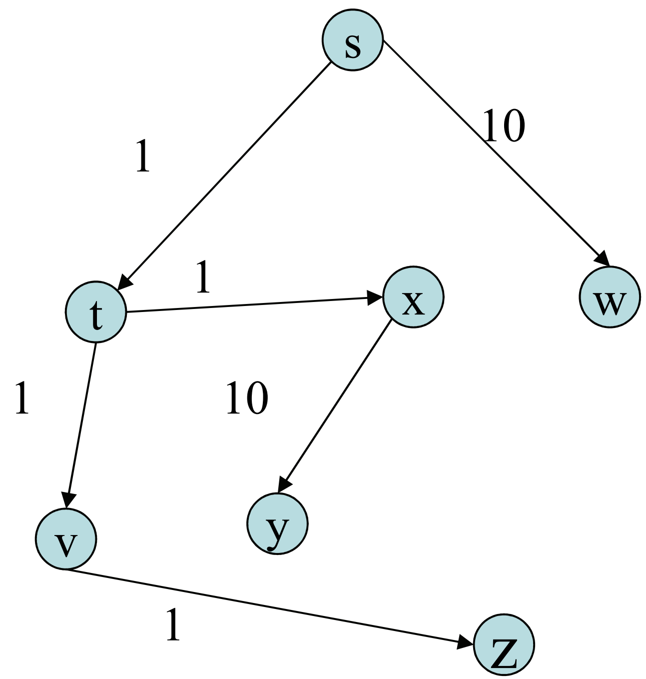
Shortest-path tree rooted at s

- 對應於圖G=(V, E) 的 Shortest-path tree rooted at s (根於s之最短路徑樹) G'=(V', E')，滿足下列三點：
  - V'是s可達的點集合 $V' \subseteq V$。
  - $E' \subseteq E$
  - G'是一個以s為根的 Rooted Tree。
  - 在G'中s到v的simple path即為G中s到v的最短路徑。

Shortest-path tree rooted at s範例



Original Graph G

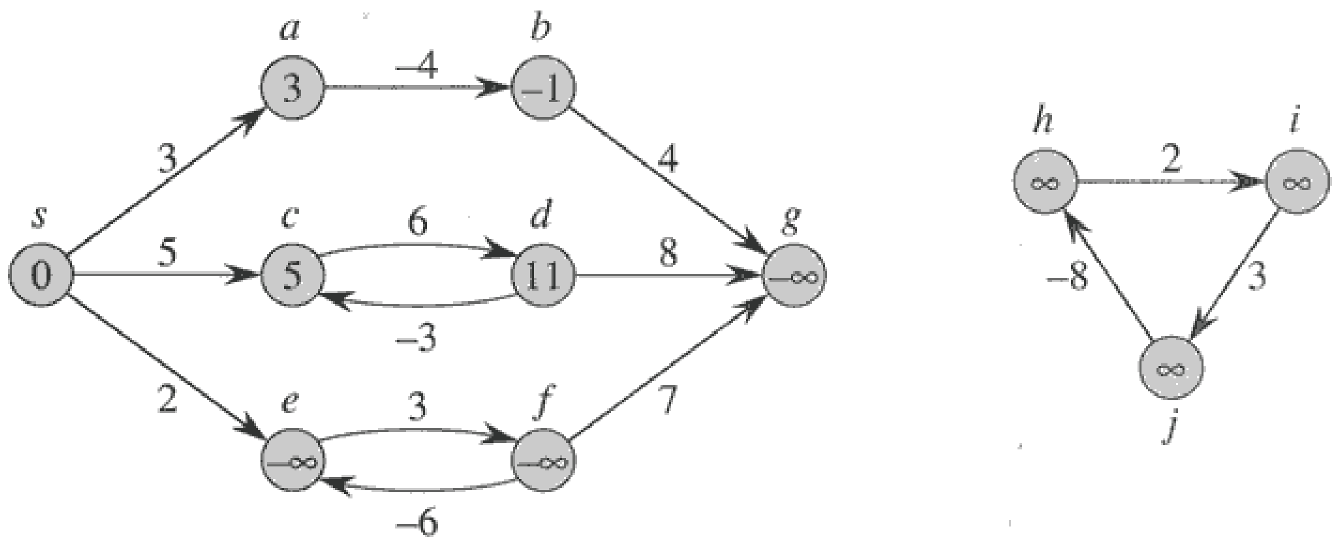


Shortest-path tree rooted at s

Negative-weight edges

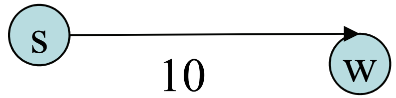
- No problem, as long as no negative-weight cycles are reachable from the source
- Otherwise, we can just keep going around it, and

get $w(s, v) = -\infty$ for all v on the cycle.



Predecessor graph

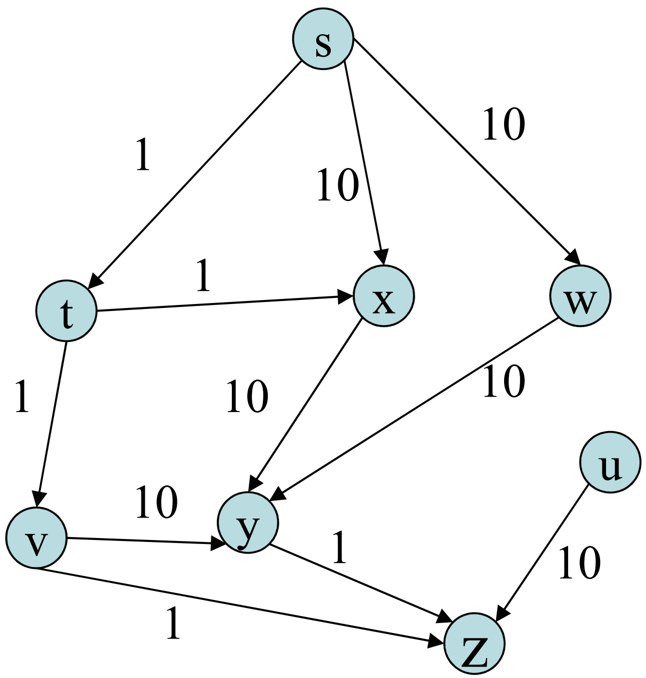
- 指向 v 的 vertex (vertices), 稱為 v 的 *predecessor(s)* 記成 $\pi[v]$。 e.g. $\pi[w]=s$.



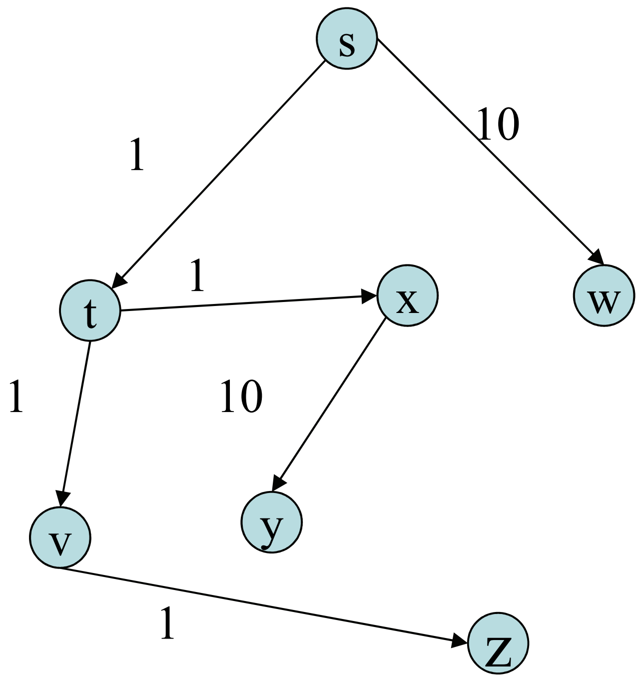
- 對一圖 $G=(V, E)$，根據一個表 π 來建構的子圖 (subgraph) $G_\pi=(V_\pi, E_\pi)$，滿足以下條件：
  - $\pi[s]=NIL$，且 $s \in V_\pi$。設定起始節點
  - 若 $\pi[v]\neq NIL$ 則 $(\pi[v], v) \in E_\pi$ 且 $v \in V_\pi$ 。
- Shortest-path tree rooted at s 是 Predecessor graph 的特例。

Predecessor graph 範例

$\pi[s]$	$\pi[t]$	$\pi[u]$	$\pi[v]$	$\pi[w]$	$\pi[x]$	$\pi[y]$	$\pi[z]$
NIL	s	NIL	t	s	t	x	v



Original Graph G



Shortest-path tree rooted at s

Initialize-Single-Source演算法

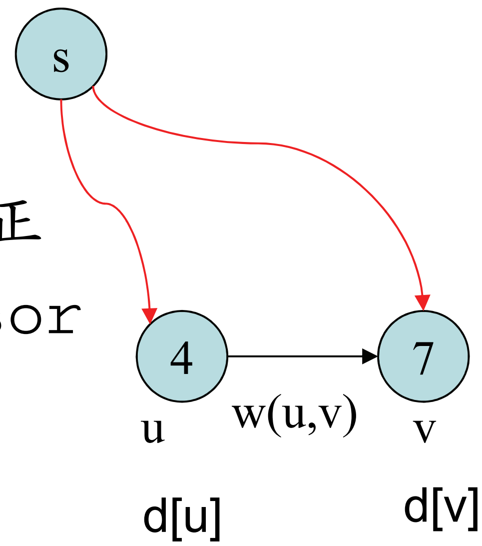
- 定義變數d[v]代表目前已知之自s至v的最短距離。
- 定義變數 π[v]代表目前已知自s至v的最短路徑上，v之前的那一點(predecessor)。
- 初始時，d[v]=∞，π[v]=NIL，d[s]=0。
- 即除自s到s的最短路徑已知之外，其餘均設為未知。

```
Initialize-Single-
  Source(G,s)
{
for each vertex v∈V[G]
      do d[v]←∞
         π[v]←NIL
  d[s]←0
}
```

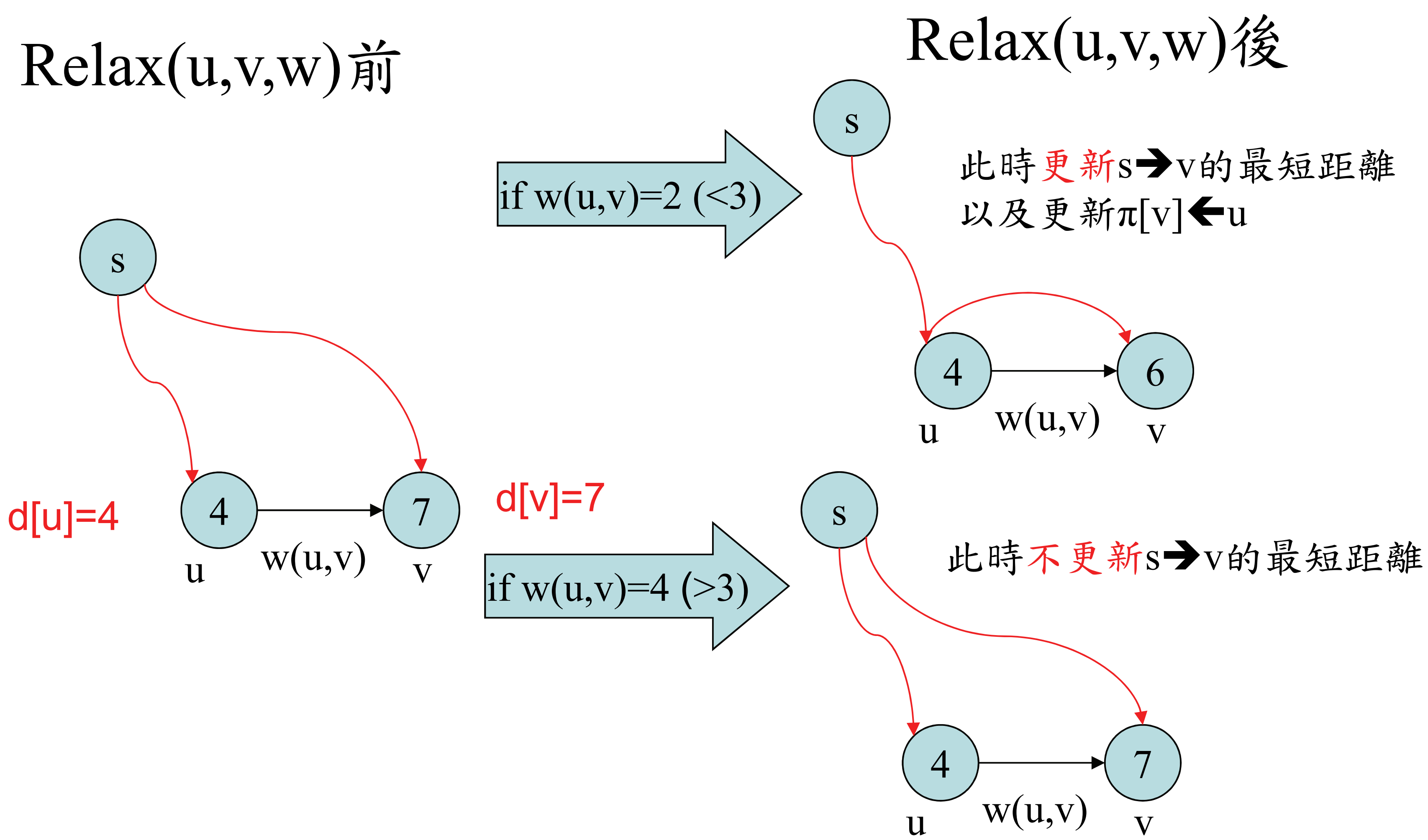
Relaxation 演算法

- 主要的目的在於利用有向邊(u, v)的資訊來更新目前所知的最短路徑。
- 目前最近距離d[v]
- 考慮新的邊(u, v)的新距離 d[u]+w(u,v)

```
Relax(u,v,w)
{ if d[v]>d[u]+w(u,v)
      then d[v]←d[u]+w(u,v)  修正
           π[v]←u  設定predecessor
}
```



Relaxation 範例



Triangle Inequality

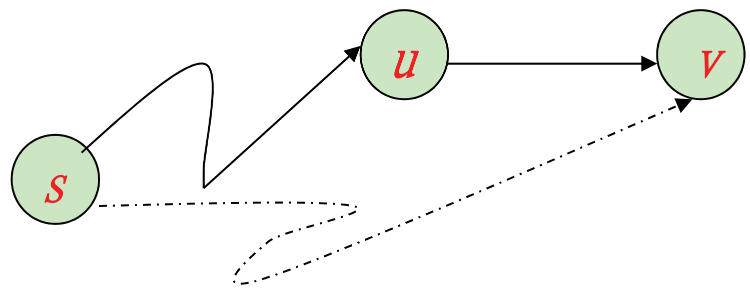
Lemma 1: for a given vertex *s* Î *V and for every edge (u,v)* Î *E,*

- $\delta(s,v) \le \delta(s,u) + w(u,v)$
- 三角不等式(Triangle inequality)：對所有的邊(u, v)，δ(s, v) ≤ δ(s, u)+w(u, v)。

Proof: shortest path $s \rightsquigarrow v$ *is no longer than any other path.*

- in particular the path that takes the shortest path $s \rightsquigarrow v$ *and*

then takes cycle (u,v)



最短路徑與 Relaxation的性質

- 上限性質(Upper-Bound property)：
  $\delta(s, v) \le d[v]$，即$d[v]$總是$s \rightarrow v$的最短距離上限。
  一旦$d[v]=\delta(s, v)$，則Relaxation不會更改$d[v]$值。

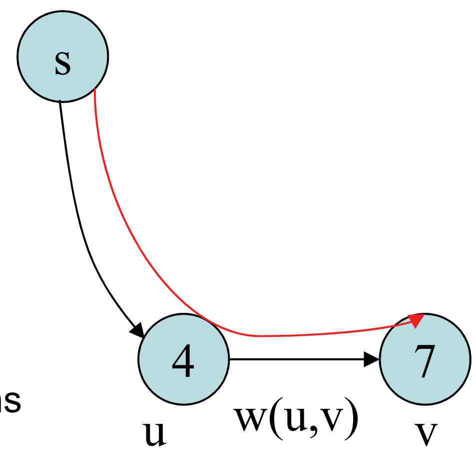
$\delta(s, v)$:s 到v 的最短距離

$d[v]$: 目前算得 s 到v 的最短距離

最短路徑與Relaxation的性質

- 無路徑性質：
  如果s到v並無路徑，則 $d[v] = \delta(s, v) = \infty$。

- 收斂性質(convergence property)：
  若s➔v的最短路徑包含邊(u, v)且$d[u]=\delta(s, u)$ 最短，
  則此時執行Relax(u, v, w)會使得$d[v]=\delta(s, v)$。

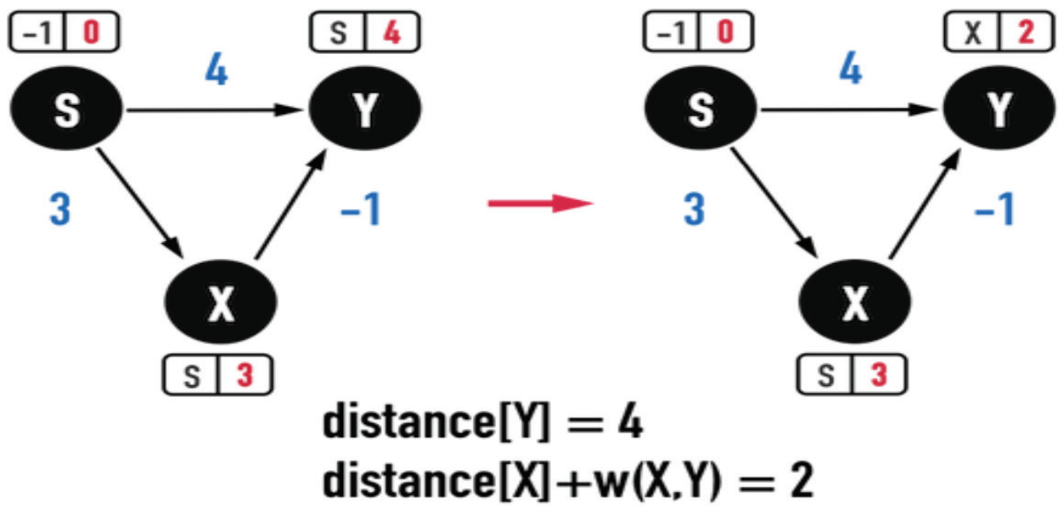


收斂性質(convergence property)

假定Graph上存在從vertex(X)指向vertex(Y)之edge(X,Y)，並且從起點vertex走到vertex(Y)之最短路徑包含這條edge。

若在對edge(X,Y)進行`Relax()`之前，從vertex(S)到達vertex(X)的path就已經滿足最短路徑，`distance[X]` $= \delta(S, X)$，那麼在對edge(X,Y)進行`Relax()`後，必定得到從vertex(S)走到vertex(Y)之最短路徑，並更新`distance[Y]` $= \delta(S, Y)$，而且至此之後，`distance[Y]`將不會再被更新。

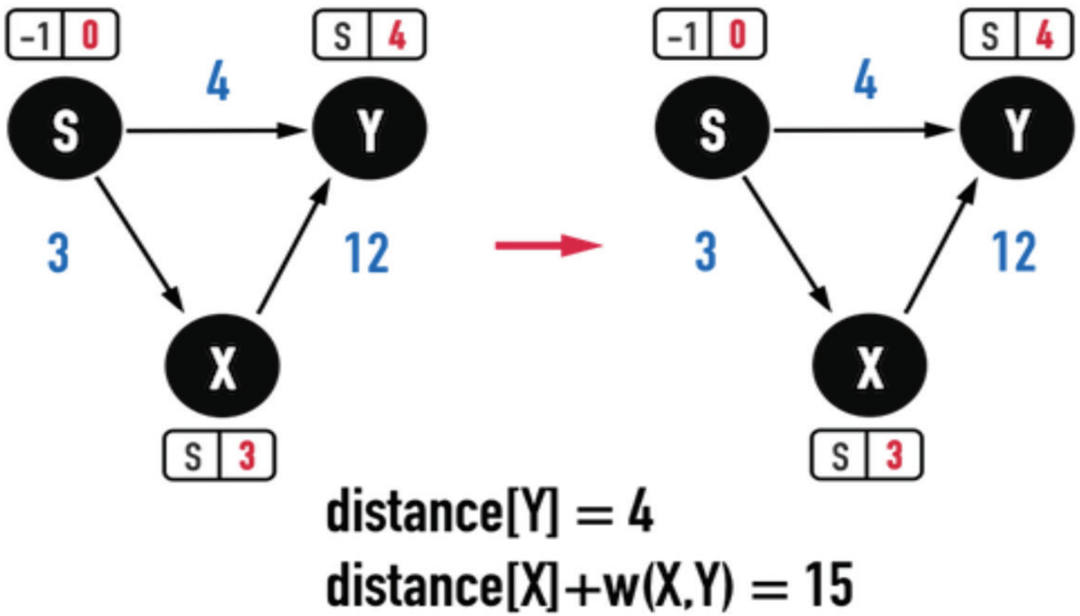
如圖五(a)，$Path = S - X - Y$是從vertex(S)走到vertex(Y)的最短路徑，並且在對edge(X,Y)進行`Relax()`之前，從vertex(S)走到vertex(X)之path已經是最短路徑，`distance[X]` $= \delta(S, X)$，那麼，此時對edge(X,Y)進行`Relax()`，必定能得到從vertex(S)走到vertex(Y)之最短路徑，`distance[Y]` $= \delta(S, Y)$。



收斂性質(convergence property)

但是若從vertex(S)走到vertex(Y)之最短路徑不包含edge(X,Y)，那麼即使在對edge(X,Y)做`Relax()`之前，`distance[X]`已經等於$\delta(S,X)$，`distance[Y]`仍然不更新。

這表示，`distance[Y]`已經等於$\delta(S,Y)$，如圖五(b)。或者，從vertex(S)走到vertex(Y)之最短路徑會從其他vertex走到vertex(Y)。



最短路徑與Relaxation的性質

- Path-relaxation性質：
  如$p=(v_0,v_1,\ldots,v_k)$是一個自$s=v_0 \rightarrow v_k$的最短路徑，則依序執行$Relax(v_0,v_1,w)$， $Relax(v_1,v_2,w), \ldots$，$Relax(v_{k-1},v_k,w)$會使得$d[v_k]=\delta(s, v_k)$。

- Predecessor graph性質：
  當經過一連串的Relaxation後，對所有的點v，$d[v]=\delta(s, v)$時，此時對應的Predecessor graph G_π即是一個 Shortest-path tree rooted at s。

Path-relaxation property

考慮一條從vertex(0)到vertex(K)之路徑$P: v_0 - v_1 - \ldots - v_K$，如果在對path之edge進行`Relax()`的順序中，**曾經出現**edge(v_0,v_1)、edge(v_1,v_2)、...、edge(v_{K-1},v_K)的順序，那麼這條path一定是最短路徑，滿足`distance[K]`$= \delta(v_0, v_K)$。

- 在對edge(v_1,v_2)進行`Relax()`之前，只要已經對edge(v_0,v_1)進行過`Relax()`，那麼，不管還有對其餘哪一條edge進行`Relax()`，`distance[2]`必定會等於$\delta(0,2)$，因為**Convergence property**。

例如，現有一條從vertex(0)走到vertex(3)之最短路徑為$Path: 0-1-2-3$，根據**Convergence property**，只要在對edge(1,2)進行`Relax()`之前，已經對edge(0,1)進行`Relax()`，如此便保證$Path: 0-1-2$一定是最短路徑，此時再對edge(2,3)進行`Relax()`，便能找到$Path: 0-1-2-3$。
換言之，只要確保`Relax()`的過程曾經出現「edge(0,1)->edge(1,2)->edge(2,3)」的順序，不需理會中間是否有其他edge進行`Relax()`，即使有也不影響最後結果。

- 例如，`Relax()`順序：「edge(2,3)->**edge(0,1)**->edge(2,3)->**edge(1,2)**->**edge(2,3)**」仍可以得到最短路徑，`distance[3]`$= \delta(0,3)$。

Bellman-Ford 演算法

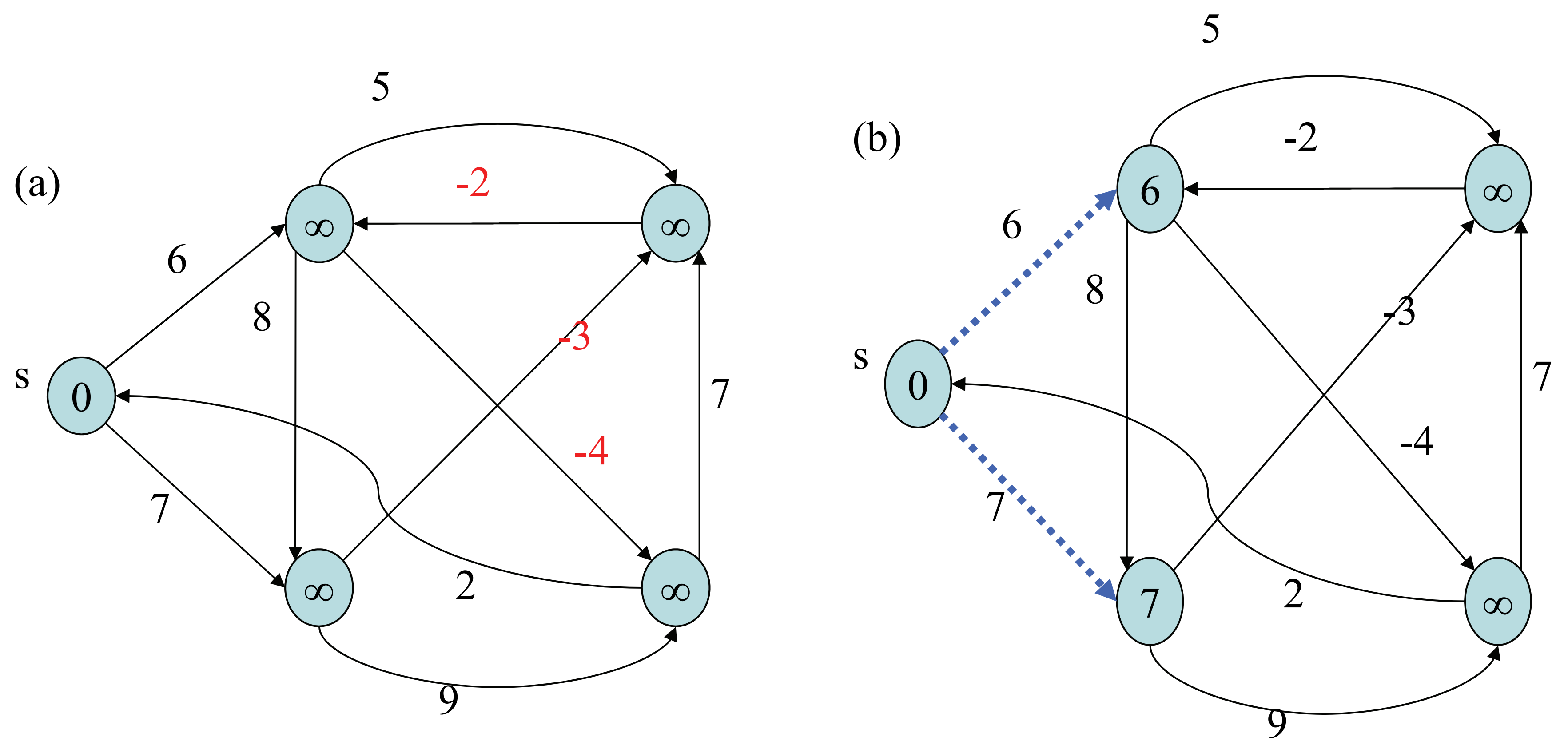
- 可以計算出沒有負迴圈的圖之最短路徑。

```
Bellman-Ford(G,w,s)
{ Initialize-Single-Source(G,s)
  for i = 1 to |V-1|
      do for each edge (u,v)∈E
           do Relex(u,v,w)
  for each edge (u,v)∈ E
      do if d[v]>d[u]+w(u,v)
           then return false     //代表有負迴圈
  return true                    //代表計算成功
}
```

```
Initialize-Single-
   Source(G,s)
{
for each vertex
   v∈V[G]
        do d[v]←∞
           π[v]←NIL
   d[s]←0
}
```

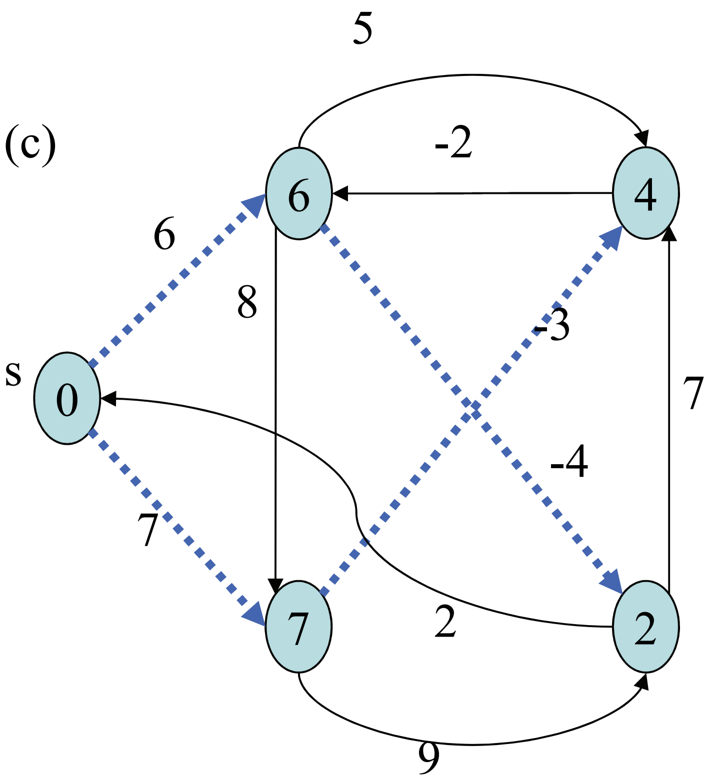
Bellman-Ford演算法運作範例

Initialize-Single-Source(G,s)

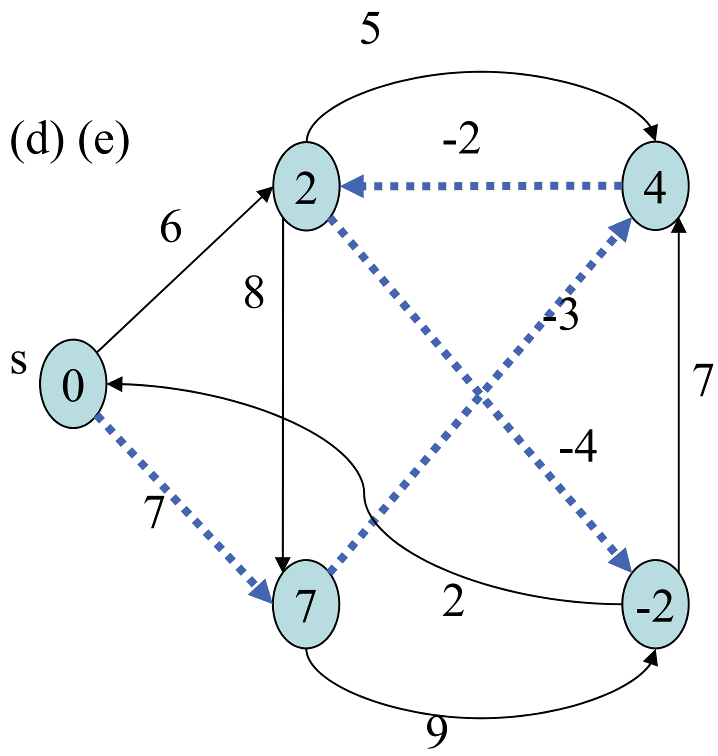


Bellman-Ford演算法運作範例

(c)



(d) (e)



Bellman-Ford演算法分析

- 正確性：
  - 對所有的邊作一次Relaxation則可將在Shortest-path tree rooted at s中一步可及的path正確計算出。
  - 由path-relaxation性質，做完|V|-1次之後，所有的Shortest simple path終點v，$d[v]=\delta(s,v)$。

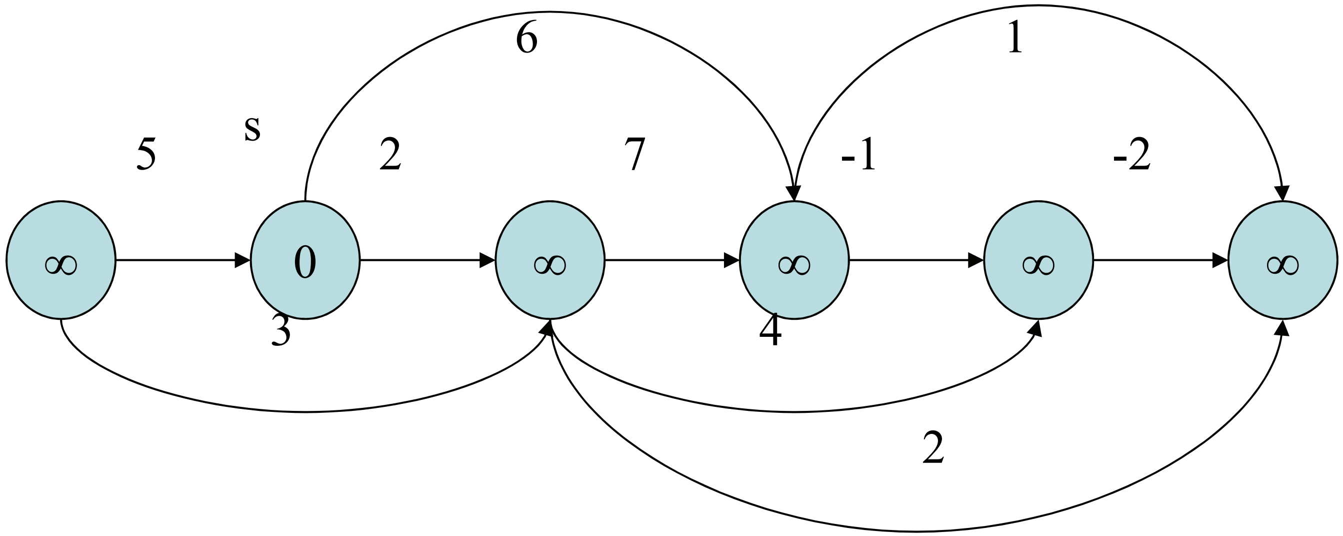
Bellman-Ford演算法分析

- 就時間複雜度的分析如下：
  - Initialize-Single-Source花去$O(|V|)$的時間。
  - 對所有的邊做了$O(|V|)$次的Relaxation，花去$O(|V||E|)$的時間。
  - 最後花去$O(|E|)$的時間檢查是否有負迴圈。
- 故總共花去$\mathbf{O(|V||E|)}$的時間。

Single-source shortest paths in DAGs

- DAG (directed access graph)
  - 就是**不存在cycle**的directed graph

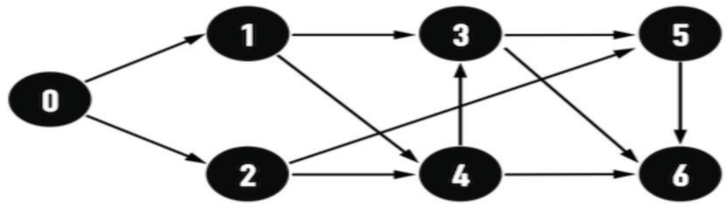
(a)



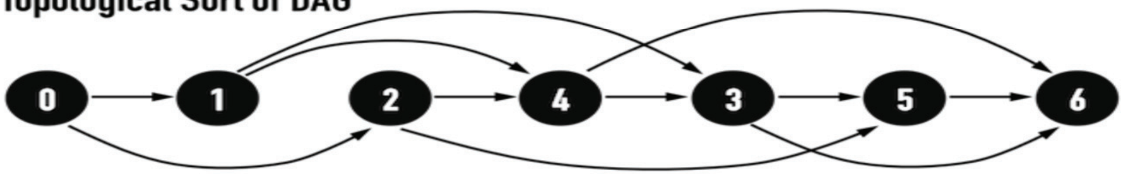
Topologically sort 拓樸排序法

- 若在**DAG**上，存在一條從vertex(X)指向vertex(Y)的edge(X,Y)，那麼在**Topological Sort**中，vertex(X)一定出現在vertex(Y)之前。
- Topological Sort可能不唯一。

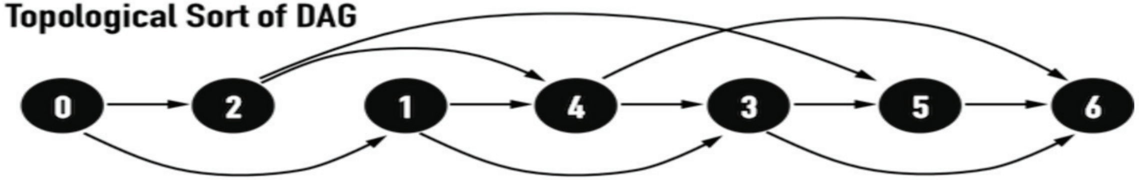
Directed Acyclic Graph



Topological Sort of DAG



Topological Sort of DAG



Single-source shortest paths in DAGs

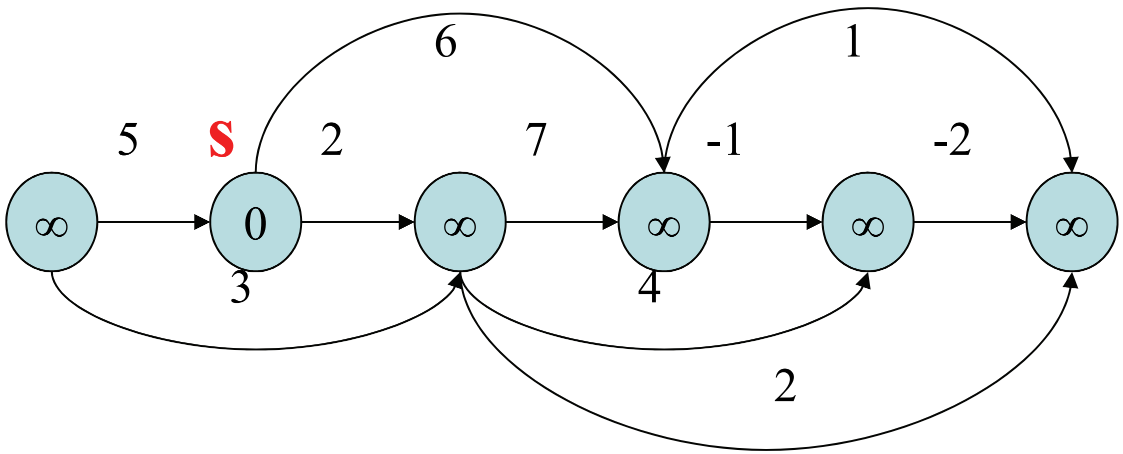
- DAG (directed access graph)
- 跟Bellman-Ford不同的是，按照特定的順序作Relaxation，可較短的時間內計算出最短路徑。

```
DAG-Shortest-Path(G,w,s)
{ Topologically sort V[G]
  Initialize-Single-Source(G,s)
  for each u taken in topological order
      do for each v∈adj[u]
            do Relax(u,v,w)
}
```

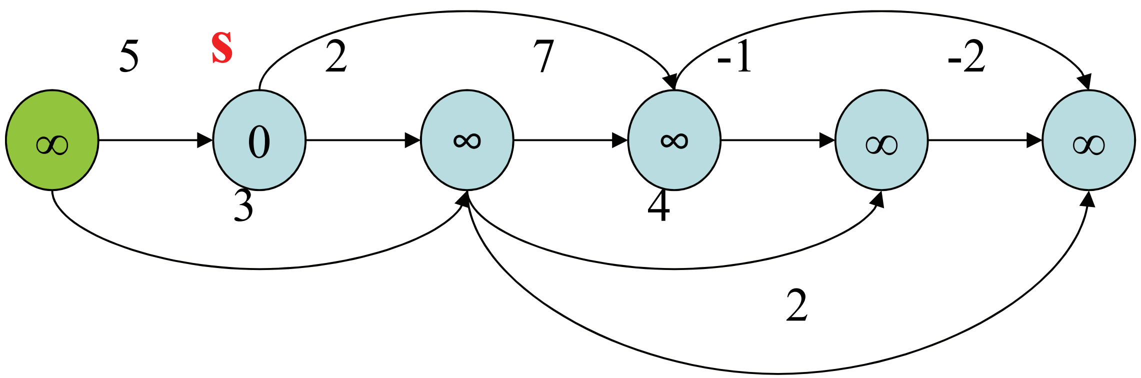
- 以上演算法僅需O(|V|+|E|)的時間。

DAG-Shortest-Path操作範例

(a)

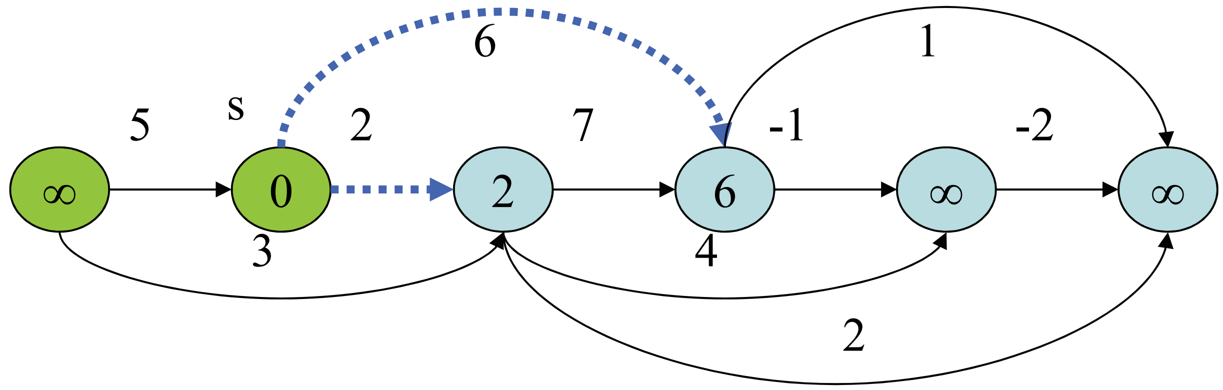


(b)

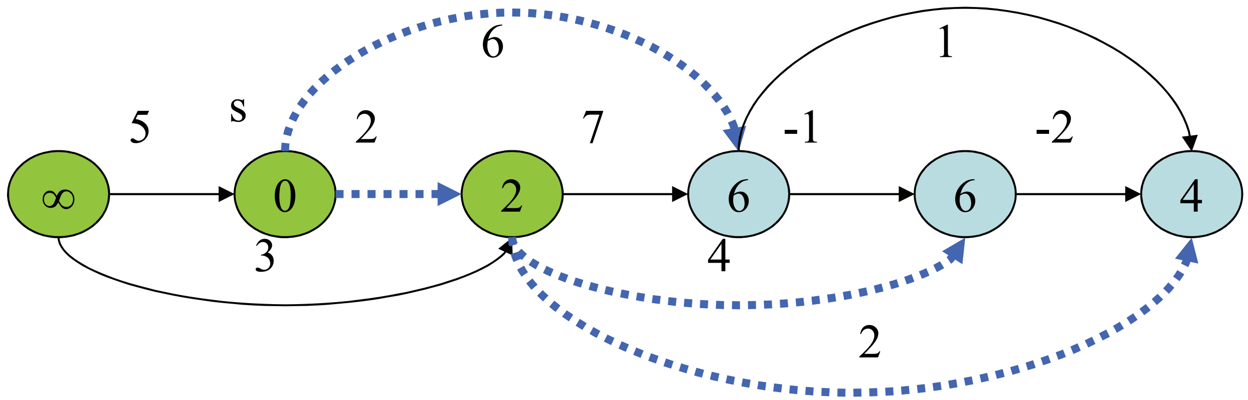


DAG-Shortest-Path操作範例

(c)

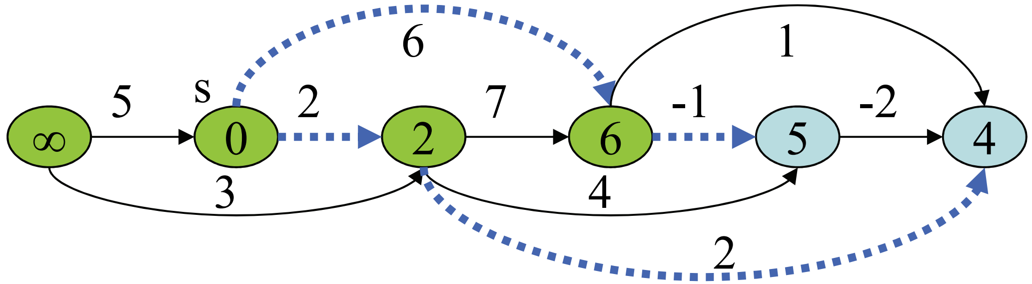


(d)

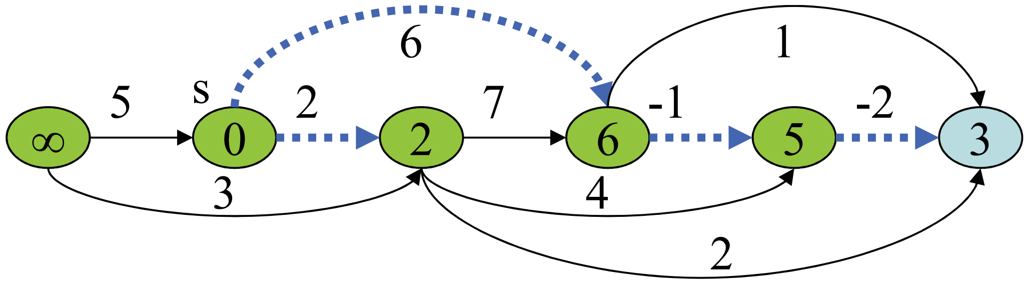


DAG-Shortest-Path 操作範例

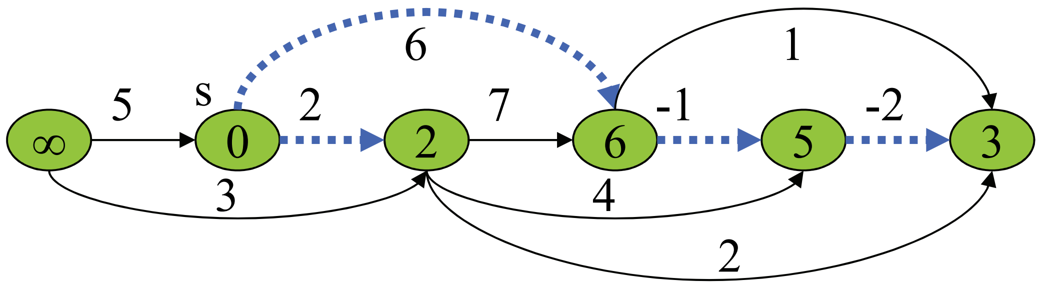
(e)



(f)



(g)

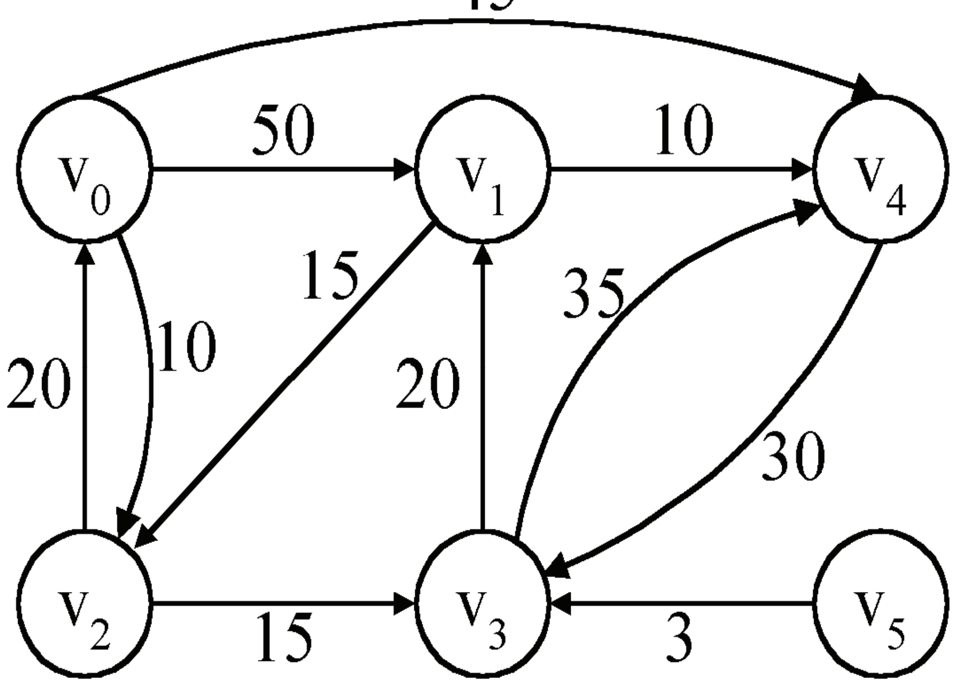


Dijkstra 演算法

- 僅能處理無負邊(no negative weight arc)的圖。
- 比Bellman-ford演算法快，也是藉由挑選特定順序來做Relaxation來達成目的。
- 利用Priority queue來實做 (Greedy Method)。
- 主要關鍵想法：利用**收斂性質**。

The single-source shortest path problem

- shortest paths from v_0 to all destinations



(a)

	Path	Length
1)	v_0v_2	10
2)	$v_0v_2v_3$	25
3)	$v_0v_2v_3v_1$	45
4)	v_0v_4	45

(b)

Dijkstra演算法

```
Q: Priority queue with d as the key
Dijkstra(G,w,s)
{
Initialize-Single-Source(G,s)
  Q=V[G]
  while Q is not empty
      do u=Extract-Min(Q)
         for each v∈adj[u]
           do Relax(u,v,w)
}
```

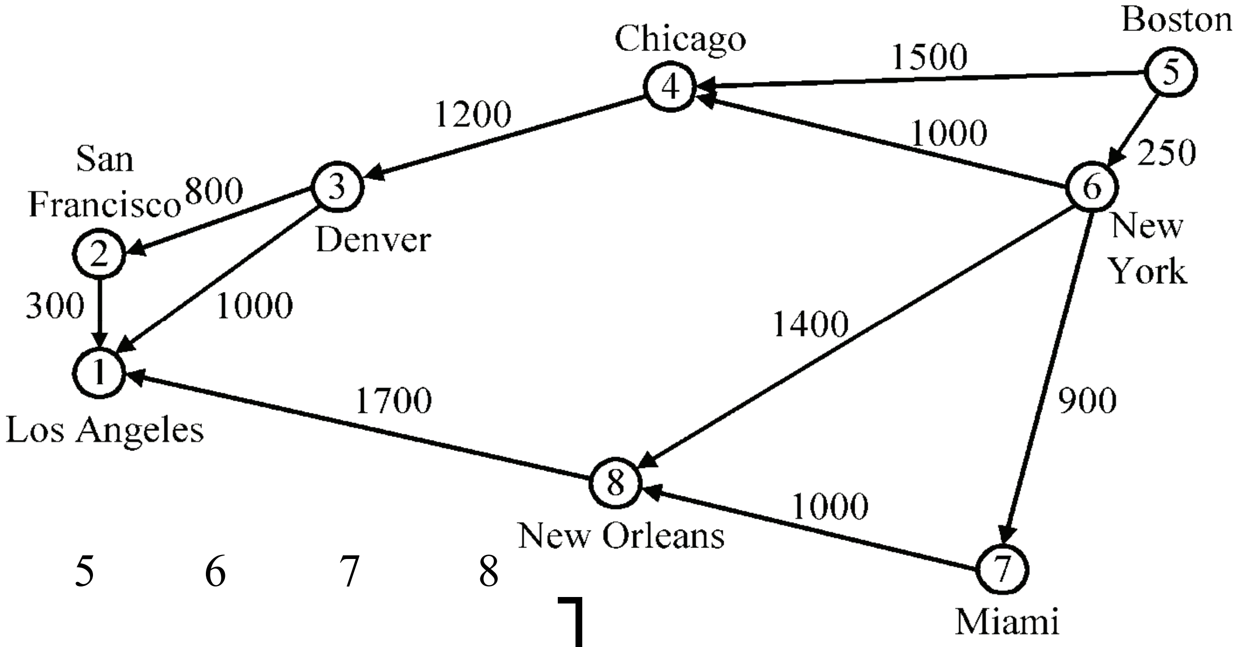
Algorithm 3–4 □ Dijkstra's algorithm to generate single-source shortest paths

Input: A directed graph $G = (V, E)$ and a source vertex v_0. For each edge $(u, v) \in E$, there is a non-negative number $c(u, v)$ associated with it. $|V| = n + 1$.

Output: For each $v \in V$, the length of a shortest path from v_0 to v.

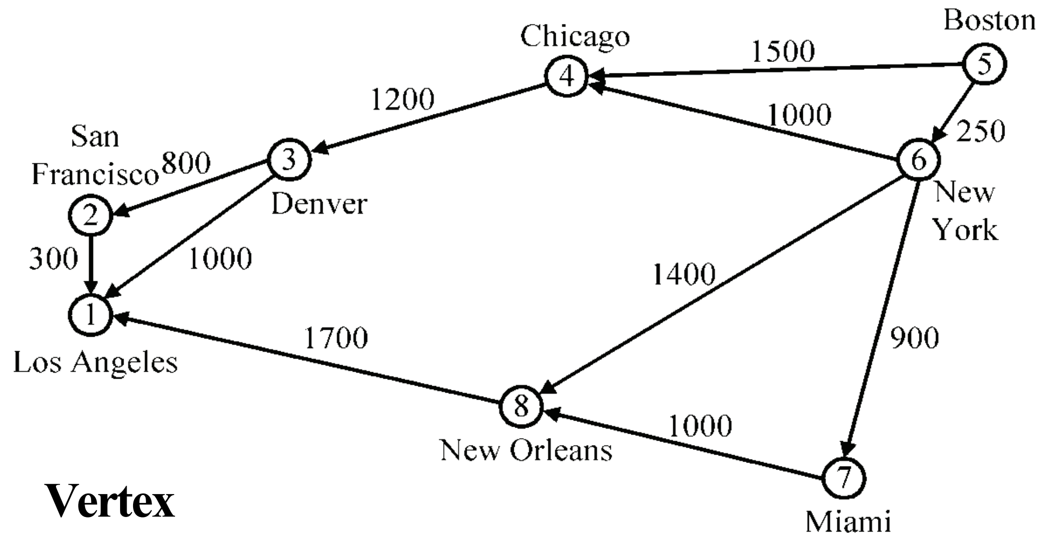
```
S := {v0}
For i := 1 to n do
Begin
    If (v0, vi) ∈ E then
        L(vi) := c(v0, vi)
    else
        L(vi) := ∞
End
For i := 1 to n do
Begin
    Choose u from V − S such that L(u) is the smallest
    S := S ∪ {u} (* Put u into S*)
    For all w in V − S do
        L(w) := min(L(w), L(u) + c(u, w))
End
```

Dijkstra's algorithm



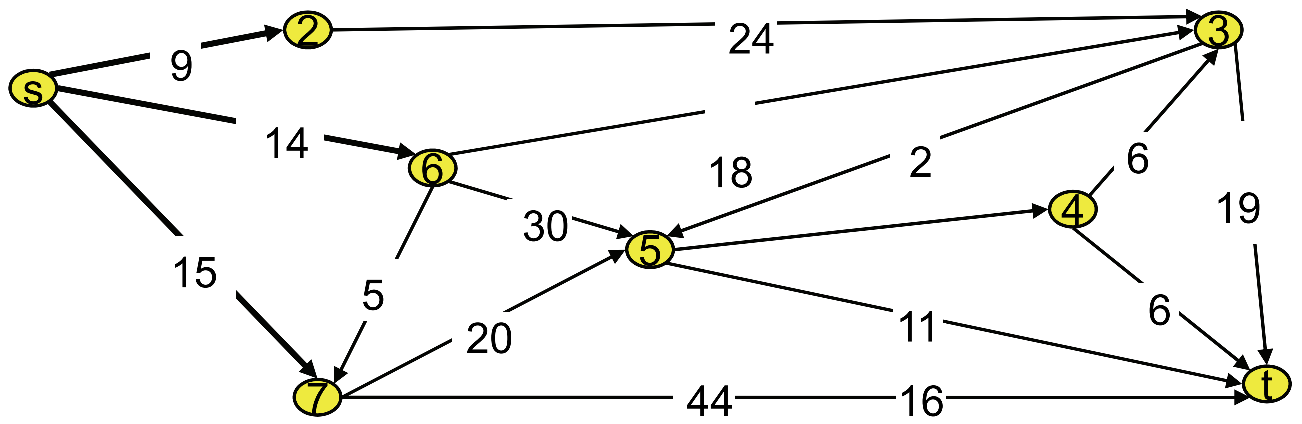
Cost adjacency matrix. All entries not shown are $+\infty$.

$$
\begin{array}{c}
 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8
\end{array}
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\left[\begin{array}{cccccccc}
0 & & & & & & & \\
300 & 0 & & & & & & \\
1000 & 800 & 0 & & & & & \\
 & & 1200 & 0 & & & & \\
 & & & 1500 & 0 & 250 & & \\
 & & & 1000 & & 0 & 900 & 1400 \\
 & & & & & & 0 & 1000 \\
1700 & & & & & & & 0
\end{array}\right]
\end{array}
$$



Iteration	S	Vertex Selected	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Initial		----								
1	5	6	$+\infty$	$+\infty$	$+\infty$	1500	0	250	$+\infty$	$+\infty$
2	5,6	7	$+\infty$	$+\infty$	$+\infty$	1250	0	250	1150	1650
3	5,6,7	4	$+\infty$	$+\infty$	$+\infty$	1250	0	250	1150	1650
4	5,6,7,4	8	$+\infty$	$+\infty$	2450	1250	0	250	1150	1650
5	5,6,7,4,8	3	3350	$+\infty$	2450	1250	0	250	1150	1650
6	5,6,7,4,8,3	2	3350	3250	2450	1250	0	250	1150	1650
	5,6,7,4,8,3,2		3350	3250	2450	1250	0	250	1150	1650

- Time complexity : $O(n^2)$



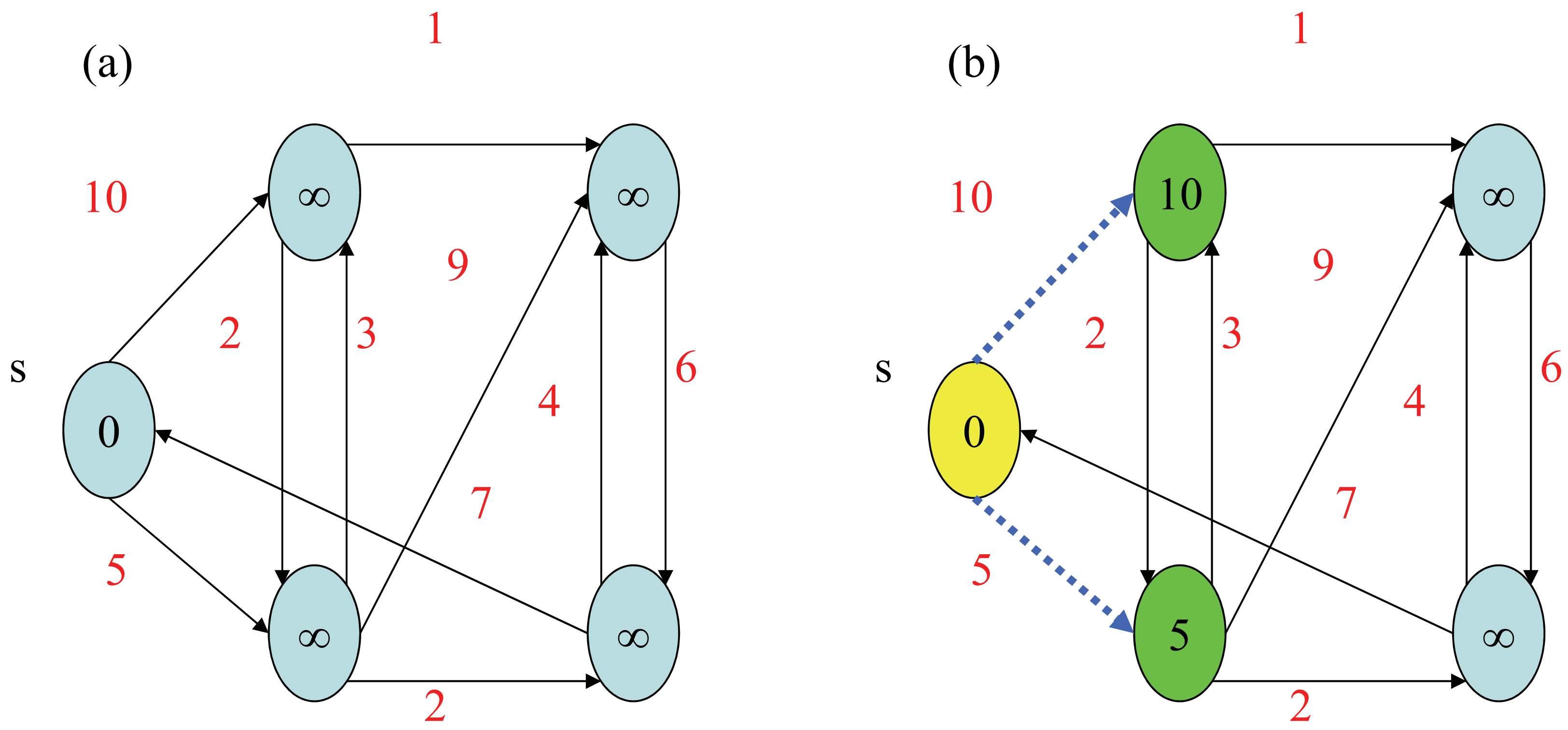
1	2	3	4	5	6	7	8	集合
0	∞	∞	∞	∞	∞	∞	∞	**C={}**
0	**9**	∞	∞	∞	14	15	∞	C={1}
0	9	33	∞	∞	**14**	15	∞	C={1, 2}
0	9	32	∞	44	14	**15**	∞	C={1, 2, 6}
0	9	**32**	∞	35	14	15	59	C={1, 2, 4,7}
0	9	32	∞	**34**	14	15	51	C={1, 2, 3, 6, 7}
0	9	32	**45**	34	14	15	50	C={1, 2, 3, 5, 6, 7}
0	9	32	45	34	14	15	**50**	C={1, 2, 3, 4, 5, 6, 7}

1	2	3	4	5	6	7	8
∞	1	6	5	3	1	1	5

到點8的最短路徑：

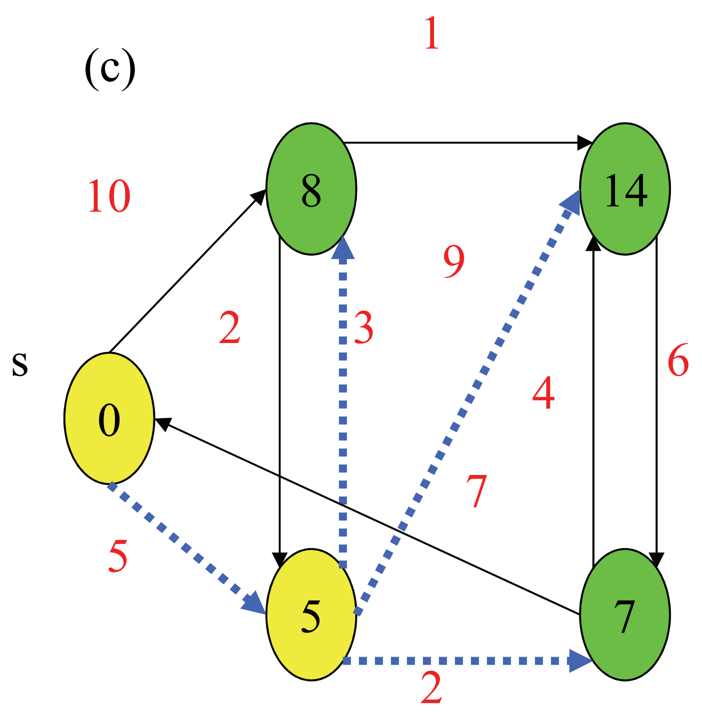
由陣列找尋順序為 8→5→3→6→1，所以最短路徑為 1→6→3→5→8

Dijkstra演算法操作範例

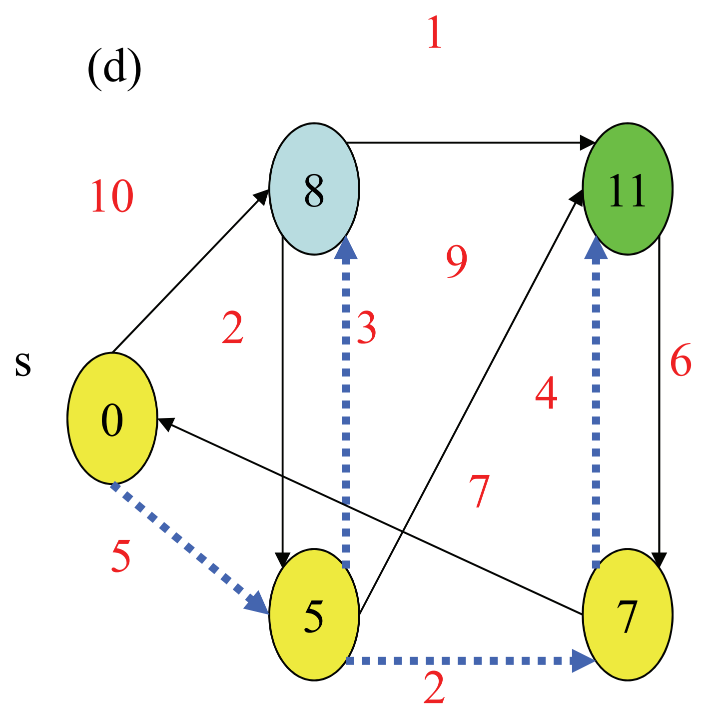


Dijkstra演算法操作範例

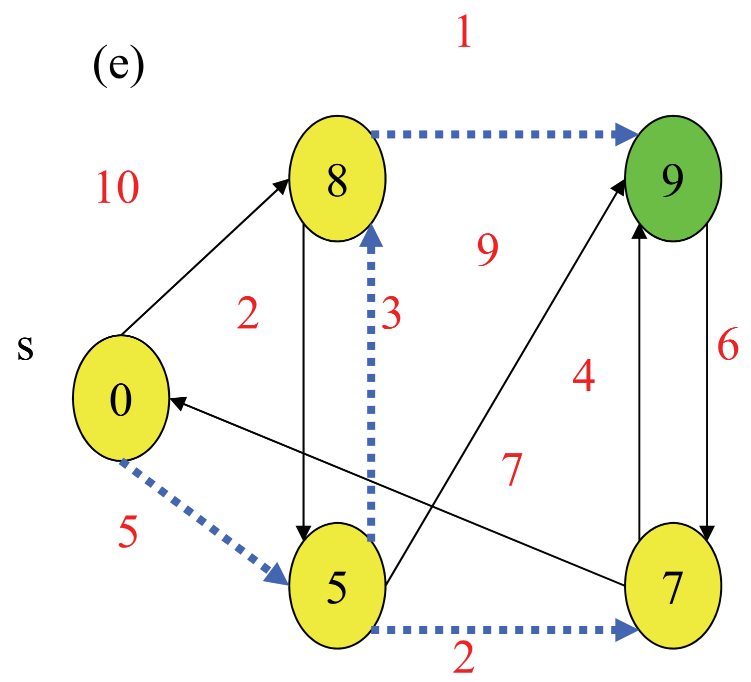
(c)

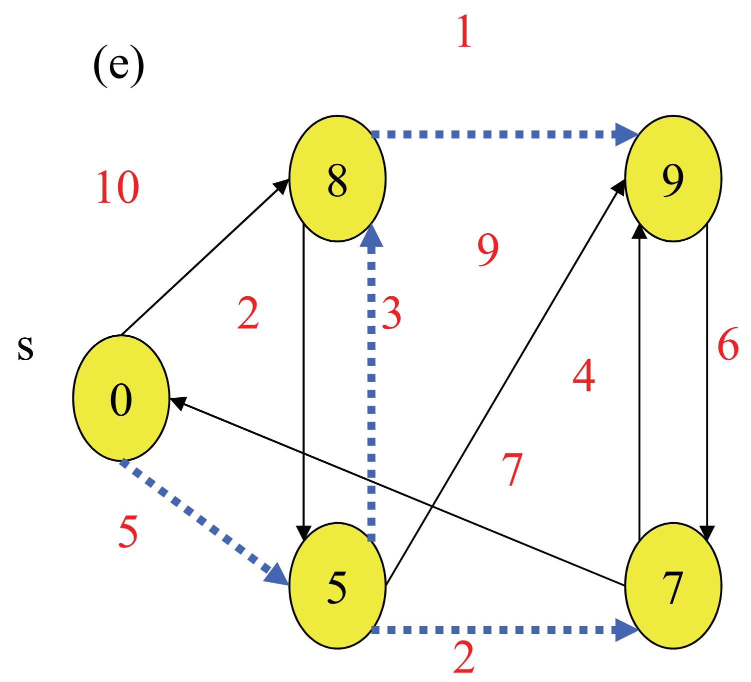


(d)



Dijkstra演算法操作範例



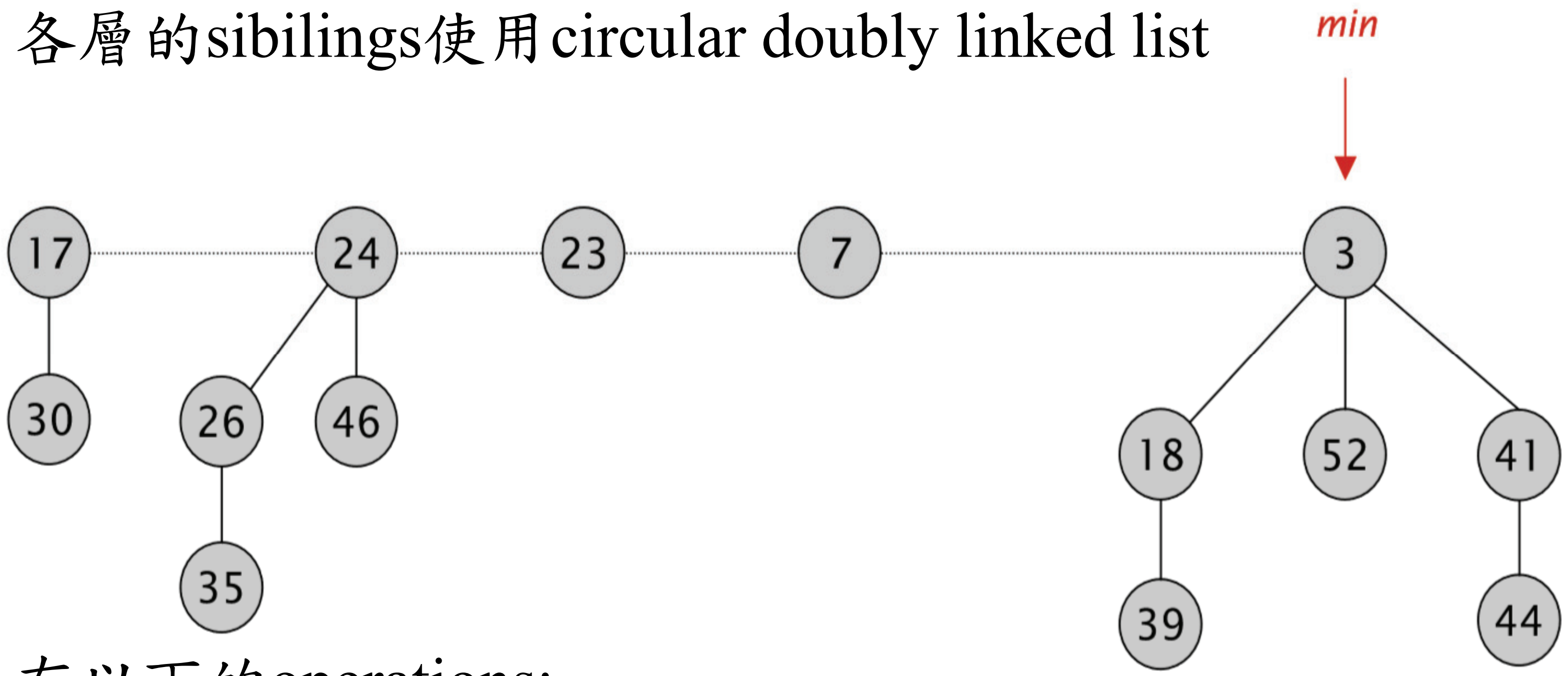


Dijkstra演算法分析

- 依照使用的Priority queue實做方式不同，會有不同的執行時間。
- 使用Linear array來實做，消耗$O(|V|^2)$的時間。
- 使用Binary heap來實做，消耗$O(|E|\log|V|)$的時間。
- 使用Fibonacci heap來實做，消耗$O(|E|+|V|\log|V|)$的時間。

Fibonacci heaps

- 也是forest (min tree or max tree組成)
- 各層的sibilings使用circular doubly linked list



- 有以下的operations:
  - Insert O(1), find min O(1), delete min O(log n), decrease O(1), merge O(1)
  - 除了delete min外, 其他都可以"平均"達到O(1)!