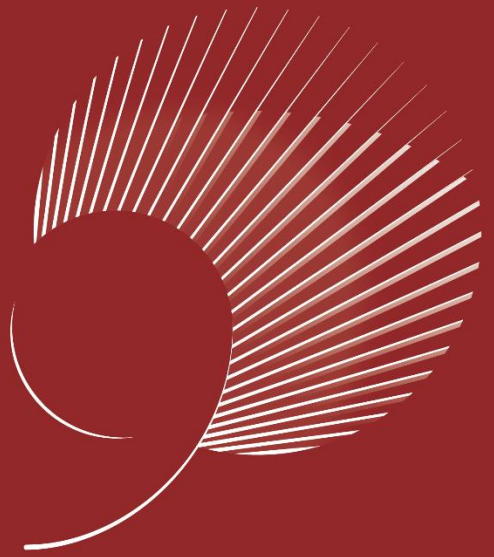
Chapter 24
Single-Source Shortest Paths

Chi-Yeh Chen

陳奇業

成功大學資訊工程學系



藏行顯光
成就共好

Achieve Securely
Prosper Mutually

國立成功大學 九十週年
90th Anniversary of NCKU

Shortest paths

- How to find the shortest route between two points on a map.
  - Input:
    - Directed graph $G = (V, E)$
    - Weight function w: $E \rightarrow R$
  - $Weight\ of\ path\ p = \langle v_0, v_1, \ldots, v_k \rangle$

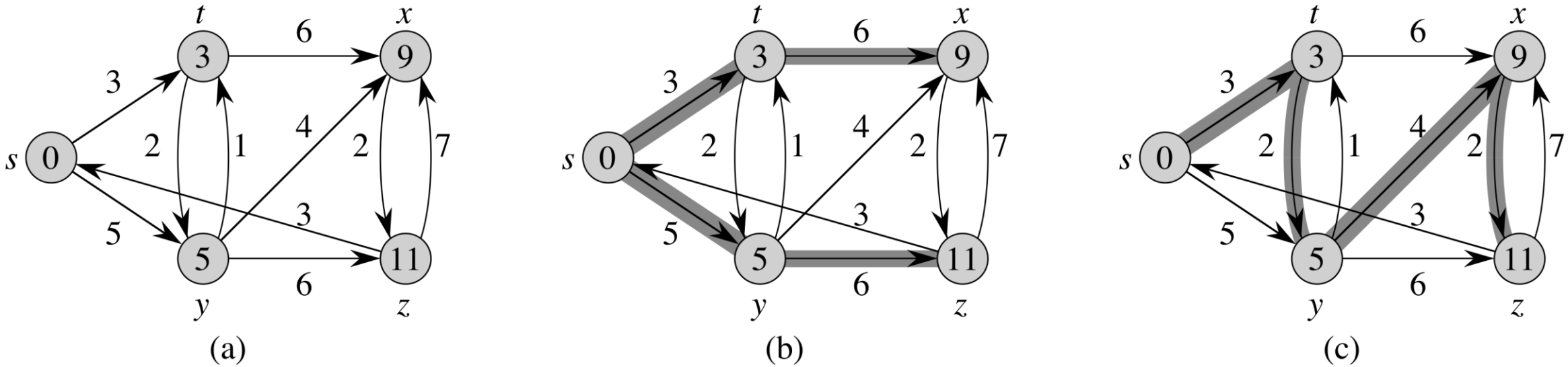
$$= \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

$$= \text{sum of edge weights on path } p.$$

- ***Shortest-path weight*** u to v :
  - $\delta(u,v) = \begin{cases} \min\{w(p): u \overset{p}{\rightsquigarrow} v\}, \text{if there exists a path } u \rightsquigarrow v \\ \infty, otherwise \end{cases}$
  - Shortest path u to v is any path p such that $w(p) = \delta(u,v)$.

- ***Example***: shortest paths from s

  [d *values appear inside vertices. Shaded edges show shortest paths.*]



- This example shows that the shortest path might not be unique.
- It also shows that when we look at shortest paths from one vertex to all other vertices, the shortest paths are organized as tree.

- Can think of weights as representing any measure that
  - accumulates linearly along a path.
  - we want to minimize.

- Example: time, cost, penalties, loss.
  - Generalization of breadth-first search to weighted graphs.

Variants

- ***Single-source***
  - Find shortest paths from a given source vertex $s \in V$ to every vertex $v \in V$.
- ***Single-destination***
  - Find shortest paths to a given destination vertex.
- ***Single-pair***
  - Find shortest path from u to v. No way known that's better in worst case than solving single-source.
- ***All-pairs***
  - Find shortest path from u to v for all u , $v \in V$.
  - We'll see algorithms for all-pairs in the next chapter.

Negative-weight edges

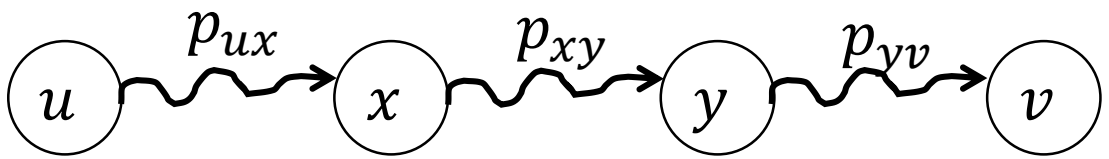
- OK, as long as no negative-weight cycles are reachable from the source.
  - If we have a negative-weight cycle, we can just keep going around it, and get $w(s, v) = -\infty$ for all v on the cycle.
  - But OK if the negative-weight cycle is not reachable from the source.
  - Some algorithms work only if there are no negative-weight edges in the graph. We'll be clear when they're allowed and not allowed.

Optimal substructure

Lemma 24.1 (Subpaths of shortest paths are shortest paths)

Given a weighted, directed graph $G = (V, E)$ with weight function $w: E \to \mathbb{R}$, let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from vertex v_0 to vertex v_k and, for any i and j such that $0 \le i \le j \le k$, let $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ be the subpath of p from vertex v_i to v_j. Then, p_{ij} is a shortest path from v_i to v_j.

Proof: Cut - and - paste.



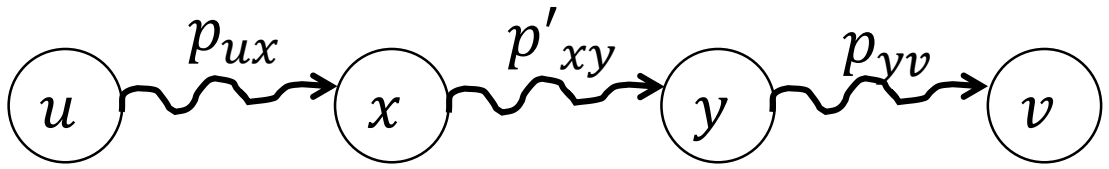
Suppose this path p is a shortest path from u to v.

Then $\delta(u, v) = w(p) = w(p_{ux}) + w(p_{xy}) + w(p_{yv})$.

Now suppose there exists a shorter path $x \overset{P'_{xy}}{\rightsquigarrow} y$

Then $w(p'_{xy}) < w(p_{xy})$

Construct p' :



Then $w(p') = w(p_{ux}) + w(p'_{xy}) + w(p_{yv})$

$< w(p_{ux}) + w(p_{xy}) + w(p_{yv})$

$= w(p)$

So p wasn't shortest path after all ! ■ (lemma)

Cycles

- Shortest paths can’t contain cycles :
  - Already ruled out negative-weight cycles.
  - Positive-weight ⇒ we can get a shorter path by omitting the cycle.
  - Zero-weight: no reason to use them ⇒ assume that our solutions won’t use them.

Output of single-source shortest-path algorithm

- For each vertex $v \in \boldsymbol{V}$:
  - $v.d = \delta(s, v)$.
    - Initially, $v.d = \infty$.
    - Reduces as algorithms progress. But always maintain $v.d \geq \delta(s, v)$.
    - Call $v.d$ a ***shortest-path estimate***.
  - $v.\pi =$ predecessor of v on a shortest path from s.
    - If no predecessor, $v.\pi = \text{NIL}$.
    - π induces a tree ------ ***shortest-path tree***.
    - We won't prove properties of π in lecture ------ see text.

Initialization

- All the shortest-paths algorithms start with INITIALIZE-SINGLE-SOURCE.

INITIALIZE-SINGLE-SOURCE(G, s)

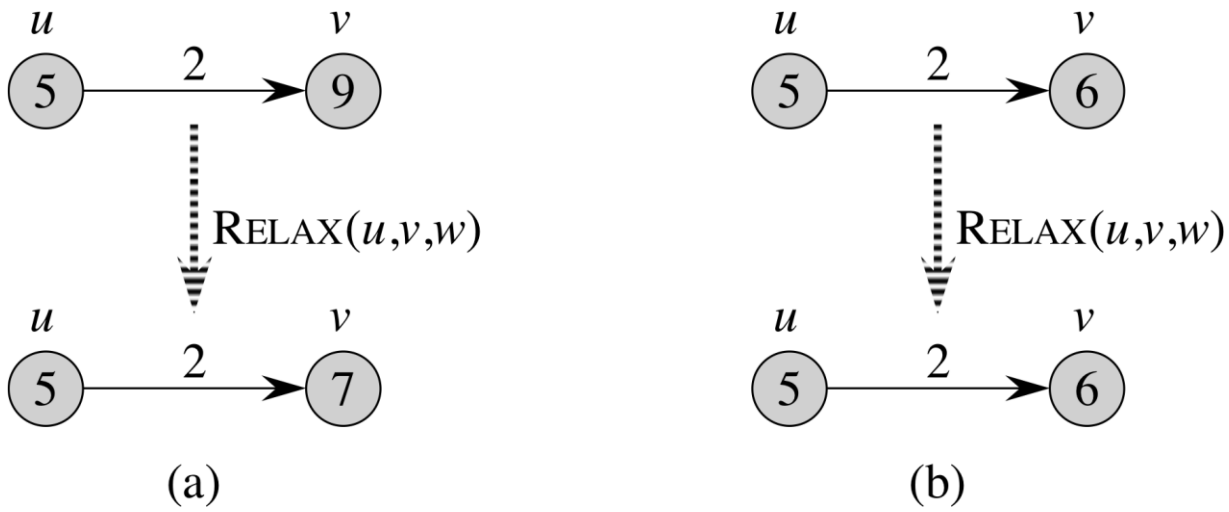
```
1 for each vertex v ∈ G.V do
2       v.d = ∞
3       v.π = NIL
4 s.d = 0
```

Relaxing an edge (u, v)

- Can we improve the shortest-path estimate for v by going through u and taking (u, v) ?

RELAX(u, v, w)

```
1 if v.d > u.d + w(u, v) then
2       v.d = u.d + w(u, v)
3       v.π = u
```



- For all the single-source shortest-paths algorithms we'll look at,
  - start by calling INITIALIZE-SINGLE-SOURCE,
  - then relax edges.

- The algorithms differ in the order and how many times they relax each edge.

Shortest-paths properties

- Based on calling INITIALIZE-SINGLE-SOURCE once and then calling RELAX zero or more times.

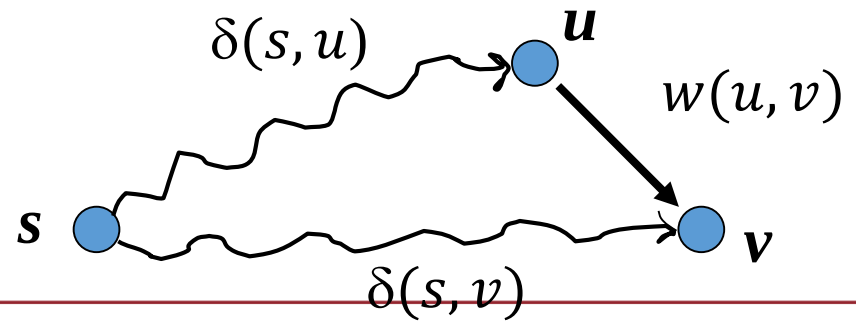
Lemma 24.10 (Triangle inequality)

For all $(u, v) \in E$, we have $\delta(s, v) \leq \delta(s, u) + w(u, v)$

***Proof*:**

Weight of shortest path $s \rightsquigarrow v$ is ≤ weight of any path $s \rightsquigarrow v$.

Path $s \rightsquigarrow u \rightarrow v$ is a path $s \rightsquigarrow v$, and if we use a shortest path $s \rightsquigarrow v$, its weight is $\delta(s, u) + w(u, v)$



Upper-bound property

Lemma 24.11 (Upper-bound property)

Let $G = (V, E)$ be a weighted, directed graph with weight function $w: E \to \mathbb{R}$. Let $s \in V$ be the source vertex, and let the graph be initialized by INITIALIZE-SINGLE-SOURCE(G, s). Then, $v.d \geq \delta(s, v)$ for all $v \in V$, and this invariant is maintained over any sequence of relaxation steps on the edges of G. Moreover, once $v.d$ achieves its lower bound $\delta(s, v)$, it never changes.

***Proof*:**

Initially true since $v.d = \infty$ implies $v.d \geq \delta(s, v)$ for all $v \in V - \{s\}$, and since $s.d = 0 \geq \delta(s, s)$.

Suppose there exists a vertex such that $v.d < \delta(s, v)$.

Without loss of generality, v is first vertex for which this happens.

Let u be the vertex that causes $v.d$ to change.

Then $v.d = u.d + w(u, v)$.

So, $v.d < \delta(s, v)$

$\leq \delta(s, u) + w(u, v)$ (triangle inequality)

$\leq u.d + w(u, v)$ (v is first violation)

$\Rightarrow v.d < u.d + w(u, v)$. (Contradicts $v.d = u.d + w(u, v)$)

Once $v.d$ reaches $\delta(s, v)$, it never goes lower. It never goes up, since relaxations only lower shortest-path estimates.

No-path property

Corollary 24.12 (No-path property)

Suppose that in a weighted, directed graph $G = (V, E)$ with weight function $w: E \to \mathbb{R}$, no path connects a source vertex $s \in V$ to a given vertex $v \in V$. Then, after the graph is initialized by INITIALIZE-SINGLE-SOURCE(G, s), we have $v.d = \delta(s, v) = \infty$, and this equality is maintained as an invariant over any sequence of relaxation steps on the edges of G.

Proof:

By the upper-bound property, we always have $\infty = \delta(s, v) \leq v.d$, and thus $v.d = \infty = \delta(s, v)$.

■

Lemma 24.13

Let $G = (V, E)$ be a weighted, directed graph with weight function $w: E \to \mathbb{R}$, and let $(u, v) \in E$. Then, immediately after relaxing edge (u, v) by executing RELAX(u, v, w), we have $v.d \leq u.d + w(u, v)$.

Proof:

If just prior to relaxing edge (u, v), we have $v.d > u.d + w(u, v)$, then $v.d = u.d + w(u, v)$ afterward.

If, instead, $v.d \leq u.d + w(u, v)$ just before the relaxation, then neither $u.d$ nor $v.d$ changes, and so $v.d \leq u.d + w(u, v)$ afterward.

Convergence property

Lemma 24.14 (Convergence property)

Let $G = (V, E)$ be a weighted, directed graph with weight function $w: E \to \mathbb{R}$, and let $s \in V$ be a source vertex, and let $s \rightsquigarrow u \to v$ be a shortest path in G for some vertices $u, v \in V$. Suppose that G is initialized by INITIALIZE-SINGLE-SOURCE(G, s), and then a sequence of relaxation steps that includes the call RELAX(u, v, w) is executed on the edges of G. If $u.d = \delta(s, u)$ at any time prior to the call, then $v.d = \delta(s, v)$ at all times after the call.

Proof:

After relaxation:

$$v.d \leq u.d + w(u, v) \text{ (Lemma 24.13)}$$
$$= \delta(s, u) + w(u, v)$$
$$= \delta(s, v) \text{ (lemma 24.1----- optimal substructure)}$$

Since $v.d \geq \delta(s, v)$, must have $v.d = \delta(s, v)$. ■

Path relaxation property

Lemma 24.15 (Path-relaxation property)

Let $G = (V, E)$ be a weighted, directed graph with weight function $w: E \to \mathbb{R}$, and let $s \in V$ be a source vertex. Consider any shortest path $p = \langle v_0, v_1, \ldots, v_k \rangle$ from $s = v_0$ to v_k. If G is initialized by INITIALIZE-SINGLE-SOURCE(G, s) and then a sequence of relaxation steps occurs that includes, in order, relaxing the edges $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, then $v_k.d = \delta(s, v_k)$ after these relaxations and at all time afterward. This property holds no mater what other edge relaxations occur, including relaxations that are intermixed with relaxations of the edges of p.

Proof:

Induction to show that$v_i.d = \delta(s, v_i)$ after (v_{i-1}, v_i) is relaxed.

Basis:

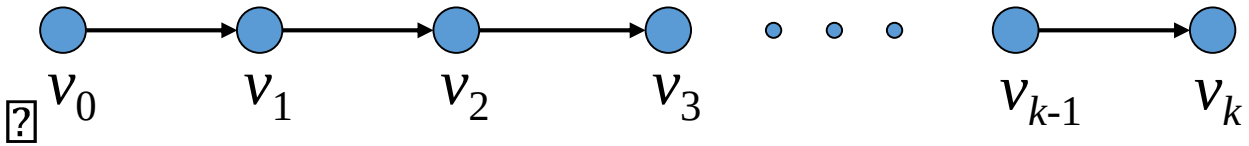
$i = 0$. Initially, $v_0.d = 0 = \delta(s, v_0) = \delta(s, s)$.

Inductive step:

Assume $v_{i-1}.d = \delta(s, v_{i-1})$.

Relax (v_{i-1}, v_i).

By Convergence Property, $v_i.d = \delta(s, v_i)$ afterward and $v_i.d$ never changes.



- $v_1.d = (s, v_0) + w(v_0, v_1) = \delta(s, v_1)$
- $v_2.d = (s, v_1) + w(v_1, v_2) = \delta(s, v_2)$
- $v_3.d = (s, v_2) + w(v_2, v_3) = \delta(s, v_3)$

.
.
.

- $v_k.d = (s, v_{k-1}) + w(v_{k-1}, v_k) = \delta(s, v_k)$

The Bellman-Ford algorithm

NCKU
National Cheng Kung University

藏行顯光
成就共好
Achieve Securely
Prosper Mutually
國立成功大學 九十週年
90th Anniversary of NCKU

The Bellman-Ford algorithm

- Allows negative-weight edges.
- Computes $v.d$ and $v.\pi$ for all $v \in \boldsymbol{V}$.
- Returns TRUE if no negative-weight cycles reachable from s, FALSE otherwise.

```
BELLMAN-FORD(G, w, s)
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 for i = 1 to |G.V| − 1 do
3     for each edge (u, v) ∈ G.E do
4         RELAX(u, v, w)
5 for each edge (u, v) ∈ G.E do
6     if v.d > u.d + w(u, v) then
7         return FALSE
8 return TRUE
```

Core: The first **for** loop relaxes all edges $|\boldsymbol{V}| - 1$ times.

Time: $\Theta(\boldsymbol{VE})$.

- $i = 1$

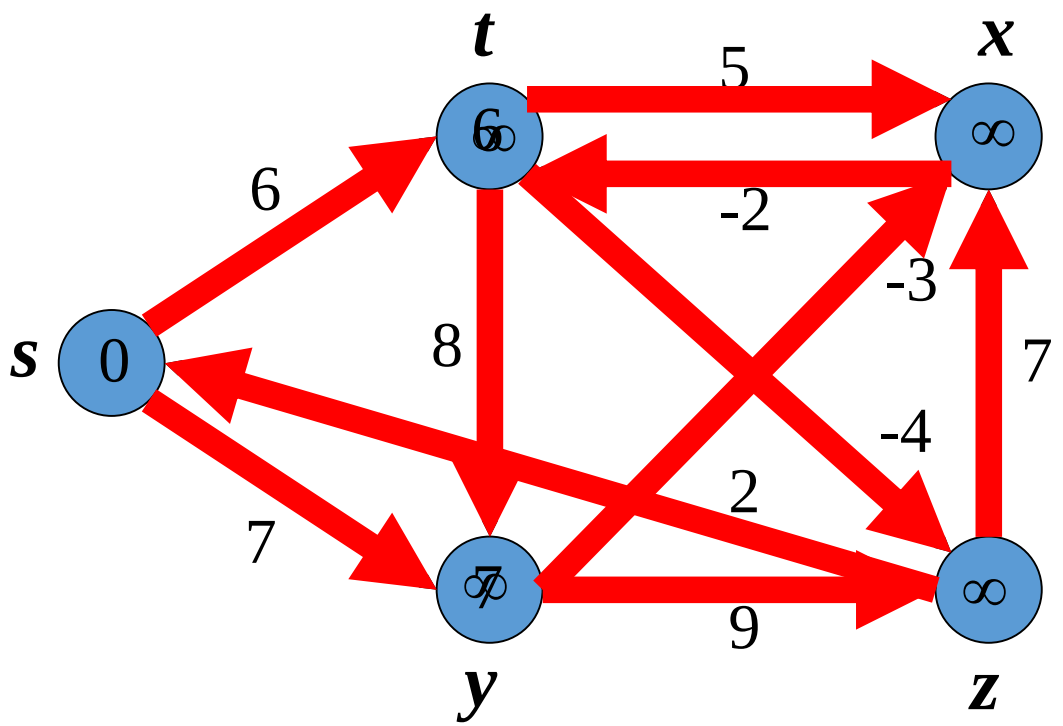


Figure 24.4 The execution of the Bellman-Ford algorithm. The source is vertex s. The d values are shown within the vertices, and shaded edges indicate predecessor values: if edge (u, v) is shaded, then $\pi[v] = u$. In this particular example, each pass relaxes the edges in the order $(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$. **(a)** The situation just before the first pass over the edges. **(b)–(e)** The situation after each successive pass over the edges. The d and π values in part (e) are the final values. The Bellman-Ford algorithm returns TRUE in this example.

- $i = 2$

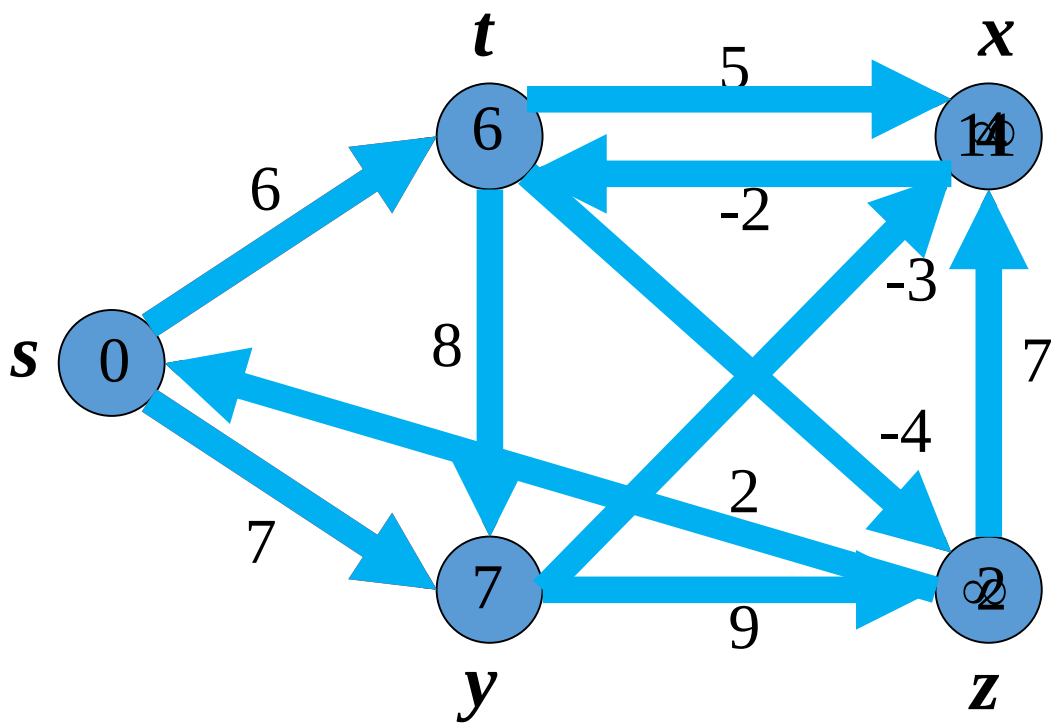


Figure 24.4 The execution of the Bellman-Ford algorithm. The source is vertex s. The d values are shown within the vertices, and shaded edges indicate predecessor values: if edge (u, v) is shaded, then $\pi[v] = u$. In this particular example, each pass relaxes the edges in the order $(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$. **(a)** The situation just before the first pass over the edges. **(b)–(e)** The situation after each successive pass over the edges. The d and π values in part (e) are the final values. The Bellman-Ford algorithm returns TRUE in this example.

- $i = 3$

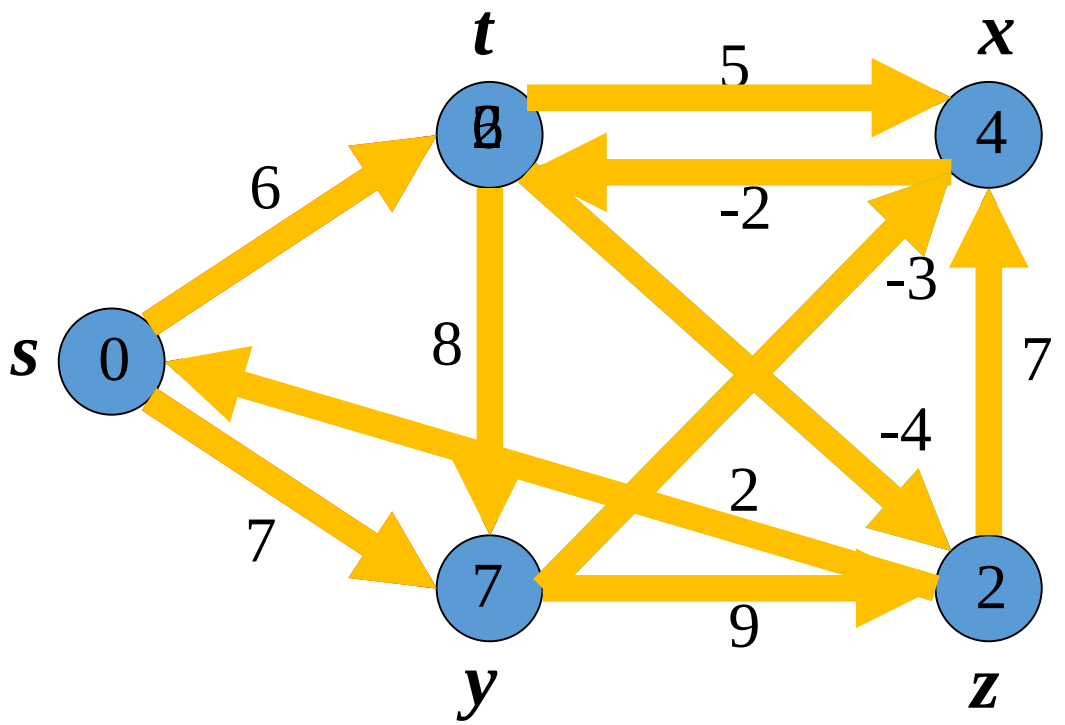


Figure 24.4 The execution of the Bellman-Ford algorithm. The source is vertex s. The d values are shown within the vertices, and shaded edges indicate predecessor values: if edge (u, v) is shaded, then $\pi[v] = u$. In this particular example, each pass relaxes the edges in the order $(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$. **(a)** The situation just before the first pass over the edges. **(b)–(e)** The situation after each successive pass over the edges. The d and π values in part (e) are the final values. The Bellman-Ford algorithm returns TRUE in this example.

- $i = 4$

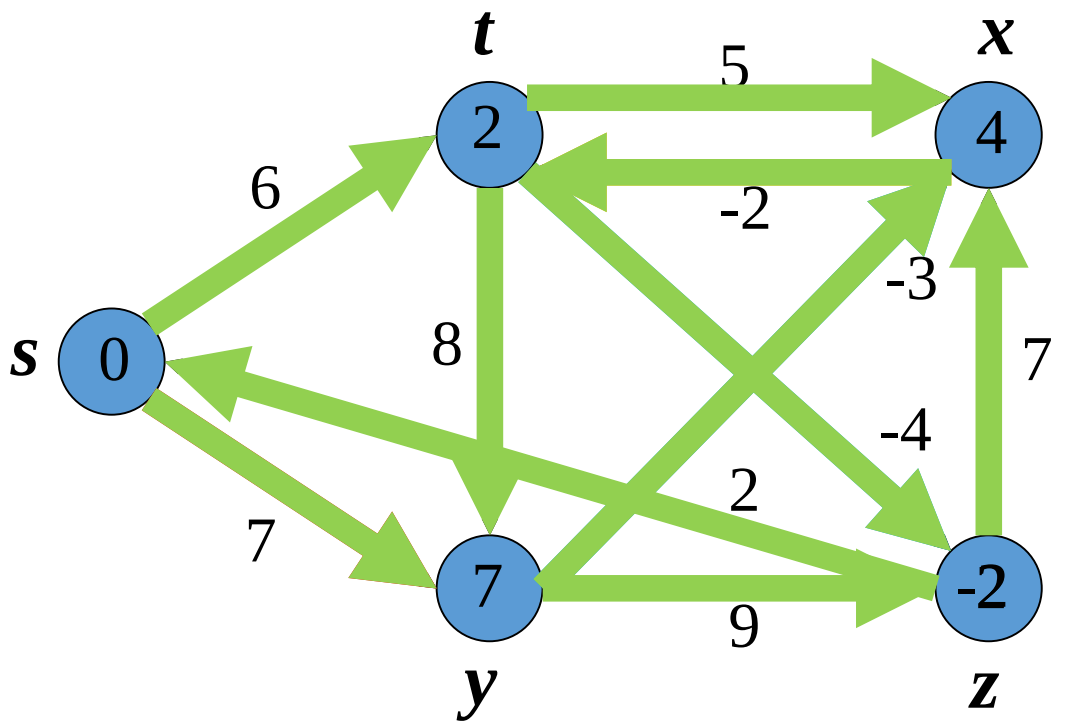
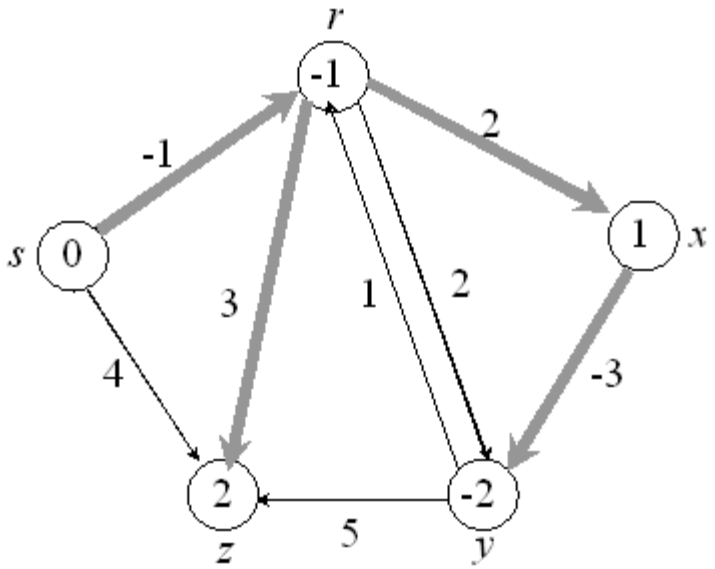


Figure 24.4 The execution of the Bellman-Ford algorithm. The source is vertex s. The d values are shown within the vertices, and shaded edges indicate predecessor values: if edge (u, v) is shaded, then $\pi[v] = u$. In this particular example, each pass relaxes the edges in the order $(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$. **(a)** The situation just before the first pass over the edges. **(b)–(e)** The situation after each successive pass over the edges. The d and π values in part (e) are the final values. The Bellman-Ford algorithm returns TRUE in this example.

- ***Example***:



- Values you get on each pass and how quickly it converges depends on order of relaxation.
- But guaranteed to converge after $|V| - 1$ passes, assuming no negative-weight cycles.

- ***Proof***: Use path-relaxation property.

  Let v be reachable from s and let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from s to v, where $v_0 = s$ and $v_k = v$.

  Since p is acyclic, it has $\leq |\boldsymbol{V}| - 1$ edges, so $k \leq |\boldsymbol{V}| - 1$.

  Each iteration of the **for** loop relaxes all edges:

  - First iteration relaxes (v_0, v_1).
  - Second iteration relaxes (v_1, v_2).
  - kth iteration relaxes (v_{k-1}, v_k).

  By the path-relaxation property, $v.d = v_k.d = \delta(s, v_k) = \delta(s, v)$.

- How about the TRUE/FALSE return value?
  - Suppose there is no negative-weight cycle reachable from s.

    At termination, for all $(u, v) \in \boldsymbol{E}$,

    $v.d = \delta(s, v)$

    $\leq \delta(s, u) + w(u, v)$ (triangle inequality)

    $= u.d + w(u, v)$

    So BELLMAN-FORD returns TRUE.

- Now suppose there exists negative-weight cycle

  $c = \langle v_0, v_1, \ldots, v_k \rangle$ where $v_0 = v_k$, reachable from s.

  Then $\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$

  Suppose (for contradiction) that BELLMAN-FORD returns TRUE.

  Then $v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_1)$ for $i = 1,2, \ldots, k$

  sum around c:

$$\sum_{i=1}^{k} v_i.d \leq \sum_{i=1}^{k} (v_{i-1}.d + w(v_{i-1}, v_i))$$

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

Each vertex appears once in each summation $\sum_{i=1}^{k} v_i.d$ and $\sum_{i=1}^{k} v_{i-1}.d$.

$\Rightarrow 0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i)$.

This contradicts C being a negative-weight cycle!

■

Single-source shortest paths in a directed acyclic graph

NCKU
National Cheng Kung University

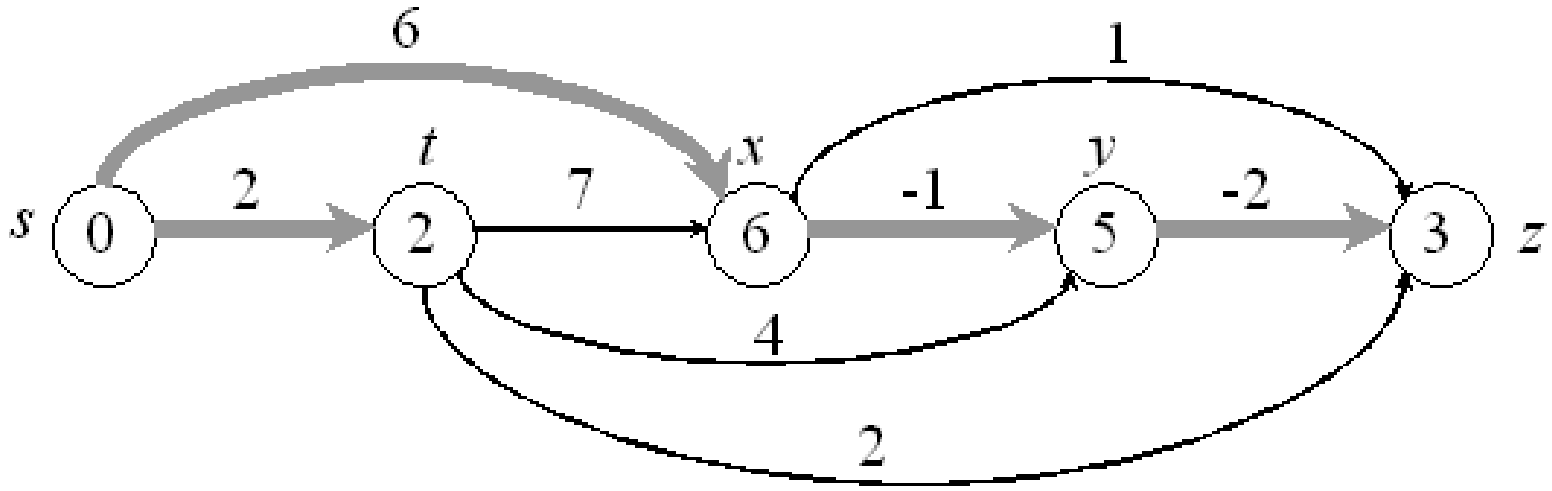
藏行顯光
成就共好
Achieve Securely
Prosper Mutually
國立成功大學 九十週年
90th Anniversary of NCKU

Single-source shortest paths in a directed acyclic graph

- Since a dag, we're guaranteed no negative-weight cycles.

```
DAG-SHORTEST-PATHS(G, w, s)
1 topologically sort the vertices of G
2 INITIALIZE-SINGLE-SOURCE(G, s)
3 for each vertex u, taken in topologically sorted order do
4     for each edge v ∈ G.Adj[u] do
5         RELAX(u, v, w)
```

- ***Example***:



- ***Time***:$\Theta(V + E)$.

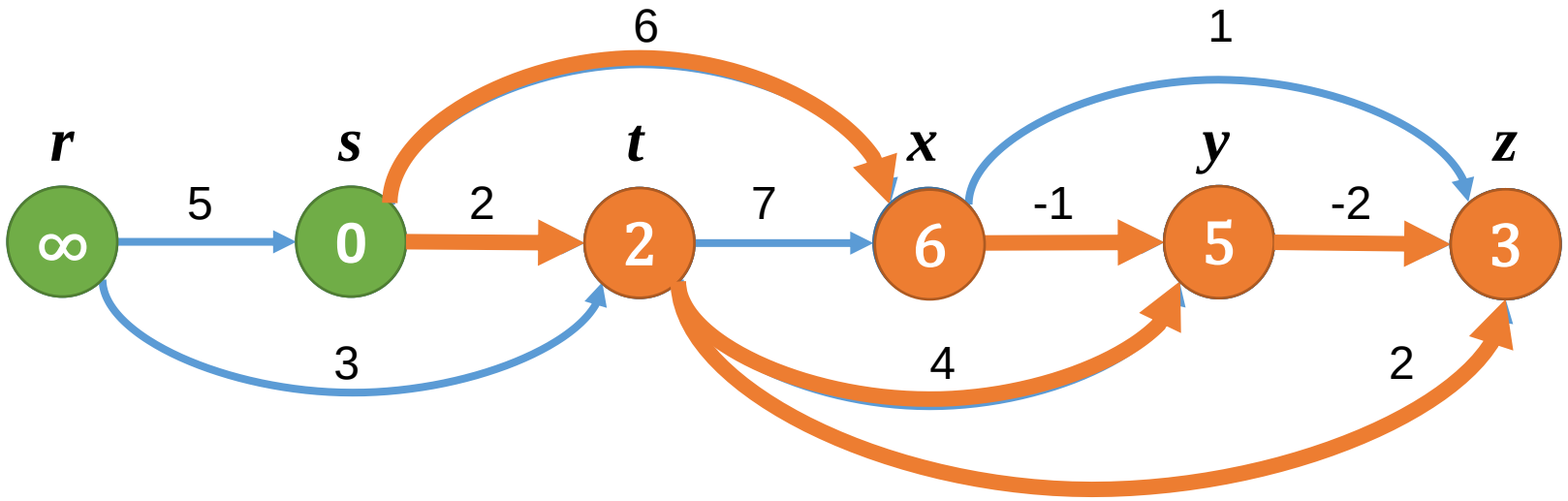


Figure 24.5 The execution of the algorithm for shortest paths in a directed acyclic graph. The vertices are topologically sorted from left to right. The source vertex is *s*. The *d* values are shown within the vertices, and shaded edges indicate the π values.

- ***Correctness***: Because we process vertices in topologically sorted order, edges of any path must be relaxed in order of appearance in the path.

  ⇒ Edges on any shortest path are relaxed in order.

  ⇒ By path-relaxation property, correct.

■

Dijkstra’s algorithm

NCKU
National Cheng Kung University

藏行顯光
成就共好
Achieve Securely
Prosper Mutually
國立成功大學 九十週年
90th Anniversary of NCKU

Dijkstra's algorithm

- No negative-weight *edges*.

- Essentially a weighted version of breadth-first search.
  - Instead of a **FIFO** queue, uses a priority queue.
  - Keys are shortest-path weights ($v.d$).

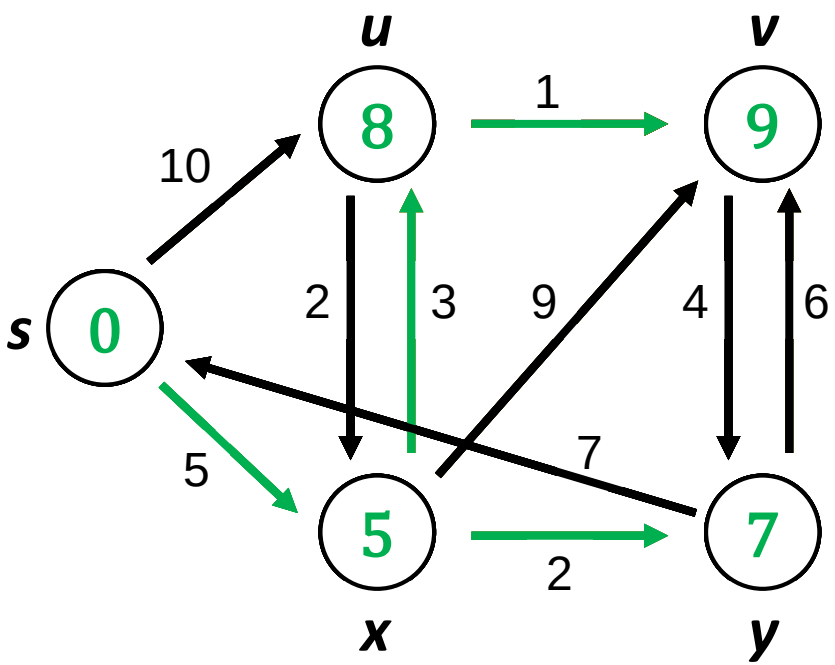
- Have two sets of vertices:
  - S = vertices whose final shortest-path weights are determined.
  - Q= priority queue = $V - S$.

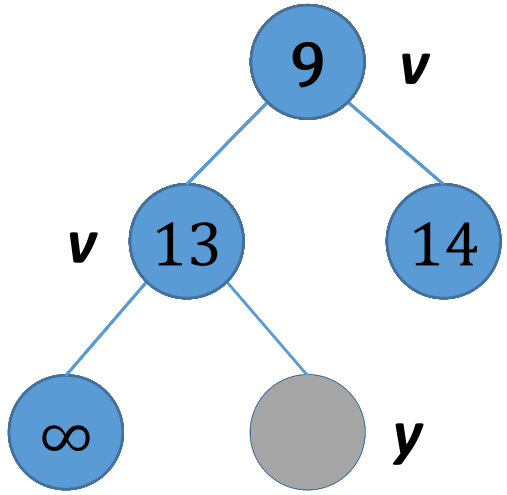
DIJKSTRA(G, w, r)

```
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 S = ∅
3 Q = G.V
4 while Q ≠ ∅ do
5     u =EXTRACT-MIN(Q)
6     S = S ∪ {u}
7     for each vertex v ∈ G.Adj[u] do
8         RELAX(u, v, w)
```

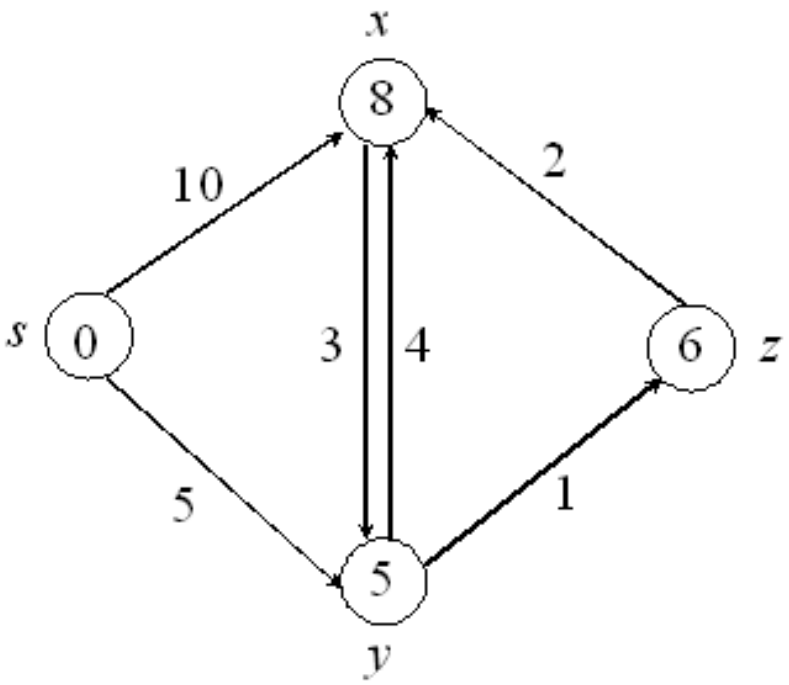
Like Prim's algorithm, but computing $v.d$, and using shortest-path weights as keys.

Dijkstra's algorithm can be viewed as greedy, since it always chooses the "lightest" ("closest"?) vertex in $V - S$ to add to S.


u
v
8
1
9
10
s
0
2
3
9
4
6
7
5
5
2
7
x
y


9
v
v
13
14
∞
y

- ***Example:***



Order of adding to S: s, y, z, x.

- ***Correctness***:
  - **Loop invariant** : At the start of each iteration of the **while** loop, $v.d = \delta(s, v)$ for all $v \in S$.
  - **Initialization**: Initially, $S = \emptyset$, so trivially true.
  - **Termination**:
    At end, $Q = \emptyset \Rightarrow S = V \Rightarrow v.d = \delta(s, v)$ for all $v \in V$.

- ***Maintenance***:

  Need to show that $u.d = \delta(s,u)$ when u is added to S in each iteration.

  Suppose there exists u such that $u.d \neq \delta(s,u)$. Without loss of generality, let u be the first vertex for which $u.d \neq \delta(s,u)$ when u is added to S.

  Observations:

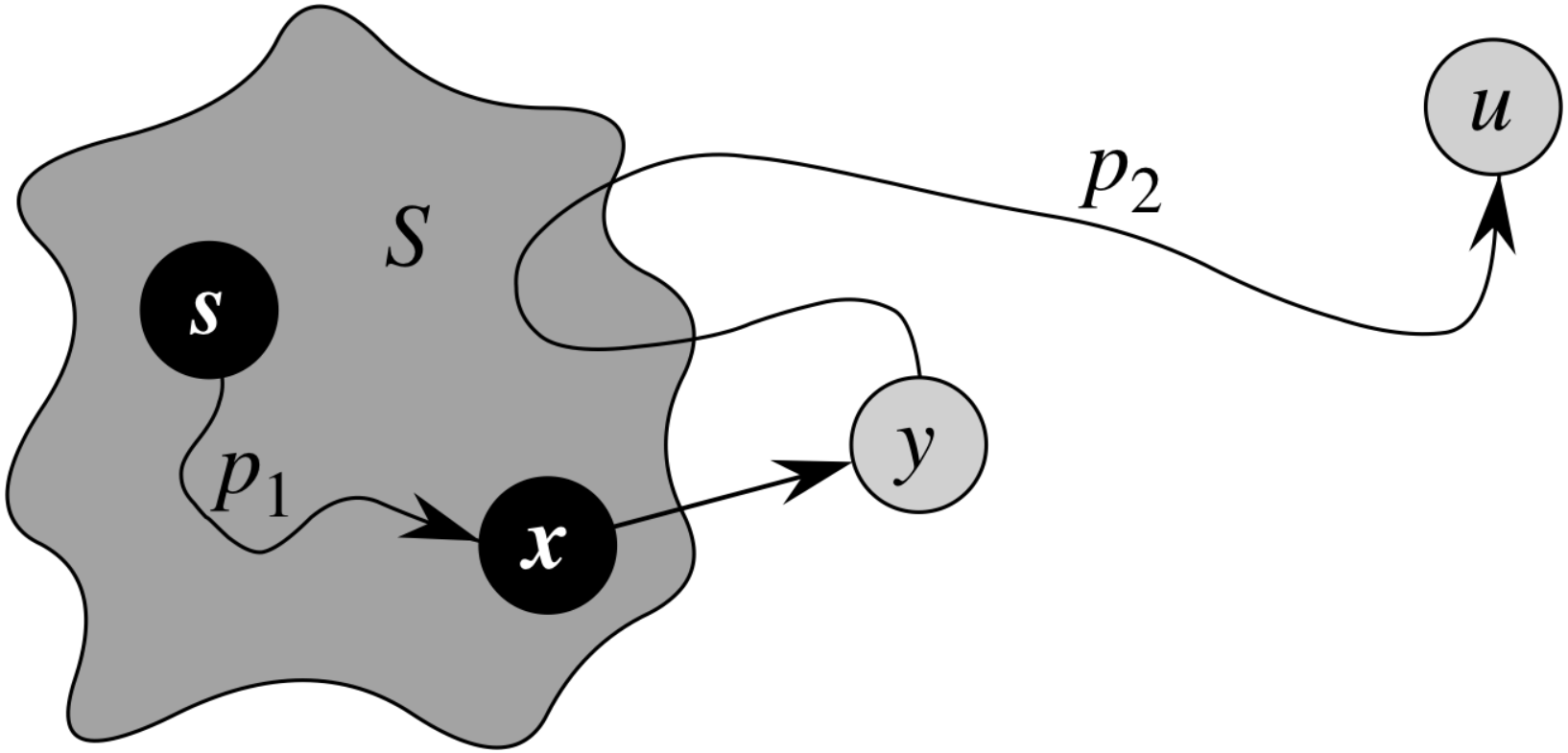
  - $u \neq s$, since $s.d = \delta(s,s) = 0$.
  - Therefore, $s \in S$, so $S \neq \emptyset$.
  - There must be some path $s \rightsquigarrow u$,
    since otherwise $u.d = \delta(s,u) = \infty$ by no-path property.

So, there's a path $s \rightsquigarrow u$.

This means there's a shortest path $s \overset{p}{\rightsquigarrow} u$.

Just before u is added to S, path p connects a vertex in S (i.e., s) to a vertex in $V - S$ (i.e., u).

Let y be first vertex along p that's in $V - S$, and let $x \in S$ be y's predecessor.



Decompose p into $s \overset{p_1}{\rightsquigarrow} x \rightarrow y \overset{p_2}{\rightsquigarrow} u$.

(Could have $x = s$ or $y = u$, so that p_1 or p_2 may have no edges.)

- ***Claim***

  $y.d = \delta(s, y)$ when u is added to S.

  Proof

  $x \in S$ and u is the first vertex such that $u.d \neq \delta(s, u)$ when u is added to $S \Rightarrow x.d = \delta(s, x)$ when x is added to S.

  Relaxed (x, y) at that time, so by the convergence property, $y.d = \delta(s, y)$.

  ■ (claim)

Now can get a contradiction to $u.d \neq \delta(s,u)$:

y is on shortest path $s \rightsquigarrow u$, and all edge weights are nonnegative

$\Rightarrow \ \delta(s,y) \leq \delta(s,u)$

$$\Rightarrow y.d = \delta(s,y)$$
$$\leq \delta(s,u)$$
$$\leq u.d \qquad \text{(upper-bound property).}$$

Also, both y and u were in Q when we chose u,
so $u.d \leq y.d \ \Rightarrow u.d = y.d.$

Therefore, $y.d = \delta(s,y) \leq \delta(s,u) \leq u.d$ becomes $y.d = \delta(s,y) = \delta(s,u) = u.d$.

Contradicts assumption that $u.d \neq \delta(s,u)$.

Hence, Dijkstra's algorithm is correct. ■

- ***Analysis***: Like Prim's algorithm, depends on implementation of priority queue.
  - If binary heap,

    Each EXTRACT-MIN takes $O(\lg V)$.

    DECREASE-KEY takes $O(\lg V)$

    Therefore, time is $O\big((V+E)\lg V\big) \Rightarrow O(E \lg V)$.
  - If Fibonacci heap,

    Each EXTRACT-MIN takes $O(\lg V)$.

    DECREASE-KEY takes $O(1)$

    Therefore, time is $O(V \lg V + E)$.

Difference constraints

Difference constraints

- Given a set of inequalities of the form $x_j - x_i \leq b_k$.
  - x's are variables, $1 \leq i,\ j \leq n$,
  - b's are constants, $1 \leq k \leq m$.

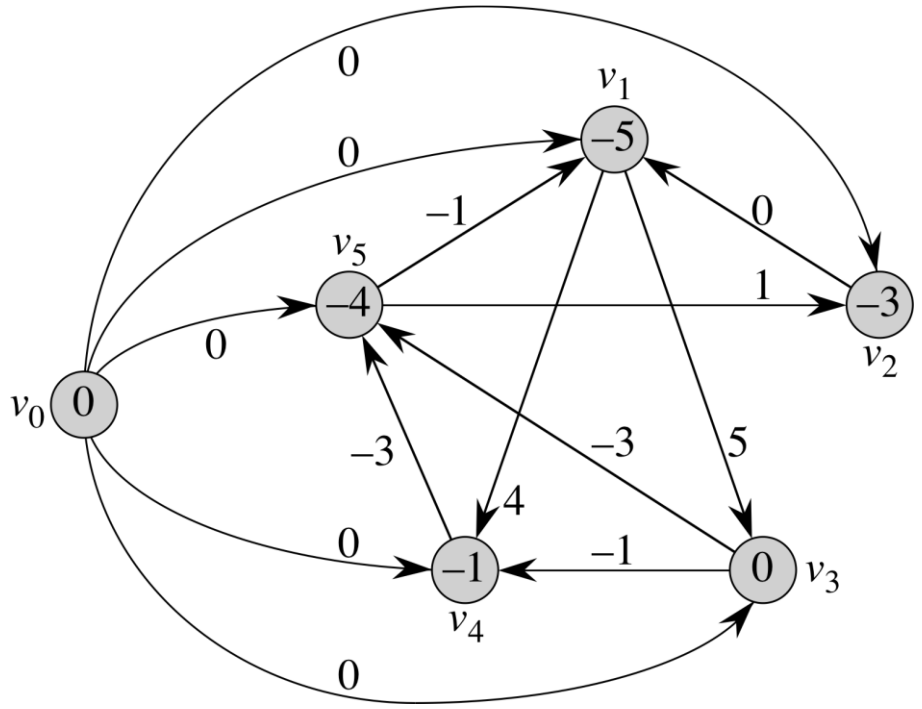
  Want to find a set of values for the x's that satisfy all m inequalities, or determine that no such values exist.

  Call such a set of values a ***feasible solution***.

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{pmatrix}$$

$$\begin{aligned} x_1 - x_2 &\leq 0 \\ x_1 - x_5 &\leq -1 \\ x_2 - x_5 &\leq 1 \\ x_3 - x_1 &\leq 5 \\ x_4 - x_1 &\leq 4 \\ x_4 - x_3 &\leq -1 \\ x_5 - x_3 &\leq -3 \\ x_5 - x_4 &\leq -3 \end{aligned}$$

- ***Example*:**



Solution: $x = (-5, -3, 0, -1, -4)$

Also: $x = (0, 2, 5, 4, 1) = [\text{above solution}] + 5$

- ***Lemma 24.8***

  If x is a feasible solution, then so is $x + d$ for any constant d.

  Proof

  x is a feasible solution

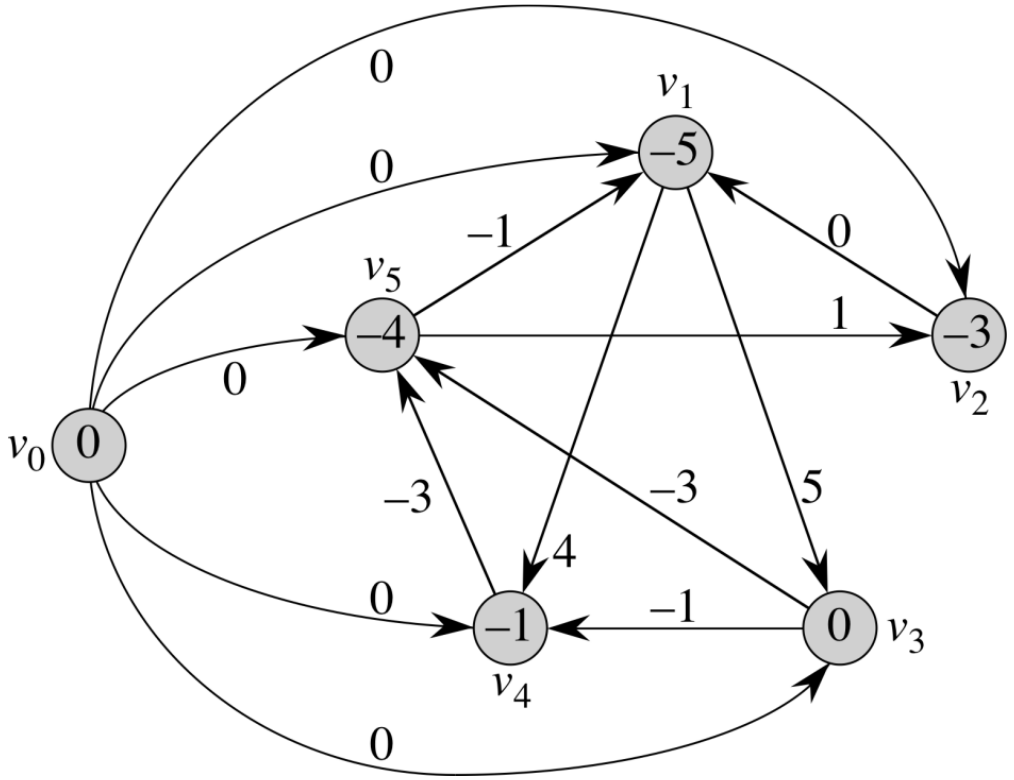
  $\Rightarrow x_j - x_i \leq b_k$ for all i, j, k.

  $\Rightarrow (x_j + d) - (x_i + d) \leq b_k$. ■ (lemma)

- **Constraint graph**

  $G = (V, E)$, weighted, directed.

  - $V = \langle v_0, v_1, \ldots, v_k \rangle$: one vertex per variable $+v_0$
  - $E = \{(v_i, v_j) : x_j - x_i \leq b_k \text{ is a constraint}\} \cup \{(v_0, v_1), (v_0, v_2), \ldots, (v_0, v_n)\}$
  - $w(v_0, v_i) = 0$ for all j
  - $w(v_i, v_j) = b_k$ if $x_j - x_i \leq b_k$



- ***Theorem***

  Given a system of difference constraints, let $G = (V, E)$ be the corresponding Constraint graph.

  1. If G has no negative-weight cycles, then
     $x = (\delta(v_0, v_1), \delta(v_0, v_2), \ldots, \delta(v_0, v_n))$ is a feasible solution.
  2. If G has a negative-weight cycle, then there is no feasible solution.

- ***Proof***

1. Show no negative-weight cycles ⇒ feasible solution.

   Need to show that $x_j - x_i \leq b_k$ for all constraints. Use
   $x_j = \delta(v_0, v_j)$
   $x_i = \delta(v_0, v_i)$
   $b_k = w(v_i, v_j)$

   By the triangle inequality,
   $\delta(v_0, v_j) \leq \delta(v_0, v_i) + w(v_i, v_j)$
   $x_j \leq x_i + b_k$
   $x_j - x_i \leq b_k$

   Therefore, feasible.

2. Show negative-weight cycles ⇒ no feasible solution.

Without loss of generality, let a negative-weight cycle be

$C = \langle v_0, v_1, \ldots, v_k \rangle$ where $v_1 = v_k$ (v_0 can't be on C, since v_0 has no entering edges.)

C corresponds to the constraints

$$x_2 - x_1 \leq w(v_1, v_2)$$

$$x_3 - x_2 \leq w(v_2, v_3)$$

...

$$x_{k-1} - x_{k-2} \leq w(v_{k-2}, v_{k-1})$$

$$x_k - x_{k-1} \leq w(v_{k-1}, v_k)$$

(The last two inequalities above are incorrect in the first three printings of the book. They were corrected in the fourth printing.)

If x is a solution satisfying these inequalities, it must satisfy their sum.

So add them up.

Each x_i is added once and subtracted once. $(v_1 = v_k \Rightarrow x_1 = x_k)$

We get $0 \leq w(C)$.

But $w(C) < 0$, since C is a negative-weight cycle.

Contradiction ⇒ no such feasible solution x exists.

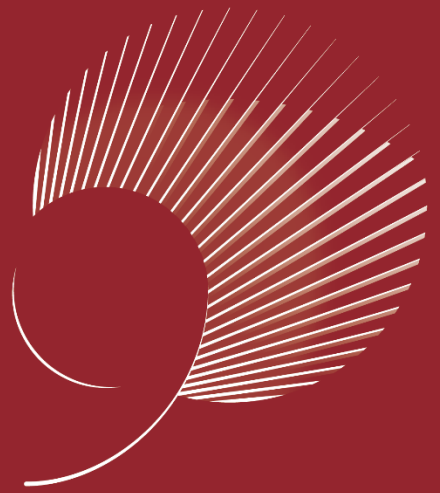
■ (theorem)

- How to find a feasible solution

1. Form constraint graph.
   - $n + 1$ vertices.
   - $m + n$ edges.
   - $\Theta(m + n)$ time.
2. Run BELLMAN-FORD from v_0 .
   - $O(VE) = O((n + 1)(m + n)) = O(n^2 + nm)$ time.
3. If BELLMAN-FORD returns FALSE ⇒ no feasible solution.

   If BELLMAN-FORD returns TURE

   ⇒ set $x_i = \delta(v_0, v_i)$ for all i.



藏行顯光
成就共好

Achieve Securely
Prosper Mutually

國立成功大學 九十週年
90th Anniversary of NCKU

國立成功大學
National Cheng Kung University
1931