COMP108
Data Structures and Algorithms

Greedy Algorithm (Part III Single-Source Shortest Paths)

Professor Prudence Wong

pwong@liverpool.ac.uk

2022-23

Single-source shortest-paths

Consider a (un)directed connected graph G

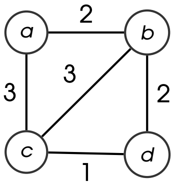
- The edges are labelled by weight

Given a particular vertex called the **source**

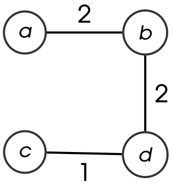
- Find **shortest** paths from the source to all other vertices (shortest path means the total weight of the path is the smallest)

Single-source shortest-paths - Examples

Graph G (edge label is weight)

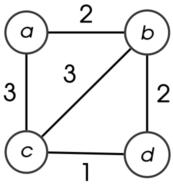


MST (global property)

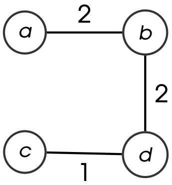


Single-source shortest-paths - Examples

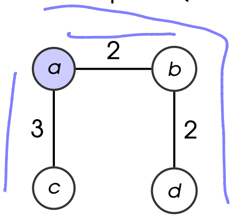
Graph G (edge label is weight)



MST (global property)



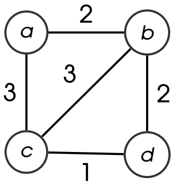
Shortest paths (local property):



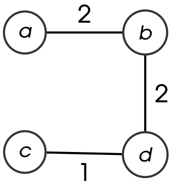
from a

Single-source shortest-paths - Examples

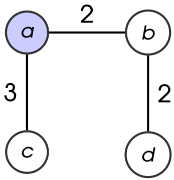
Graph G (edge label is weight)



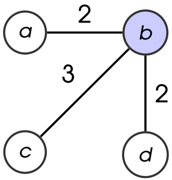
MST (global property)



Shortest paths (local property):



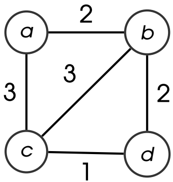
from a



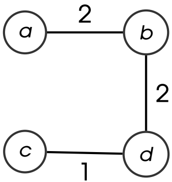
from b

Single-source shortest-paths - Examples

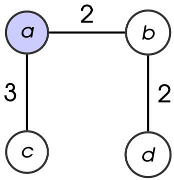
Graph G (edge label is weight)



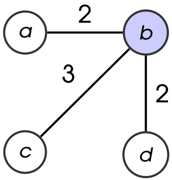
MST (global property)



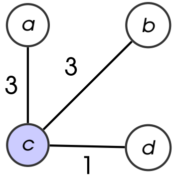
Shortest paths (local property):



from a



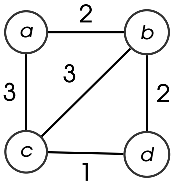
from b



from c

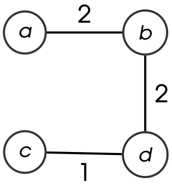
Single-source shortest-paths - Examples

Graph G (edge label is weight)

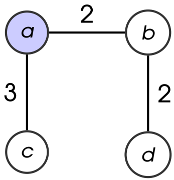


All pairs shortest paths

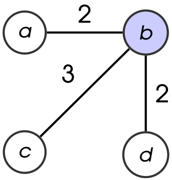
MST (global property)



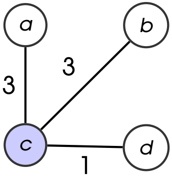
Shortest paths (local property):



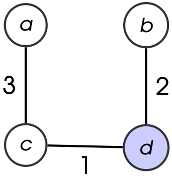
from a



from b



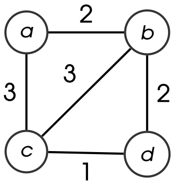
from c



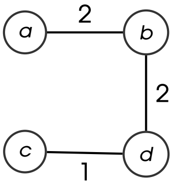
from d

Single-source shortest-paths - Examples

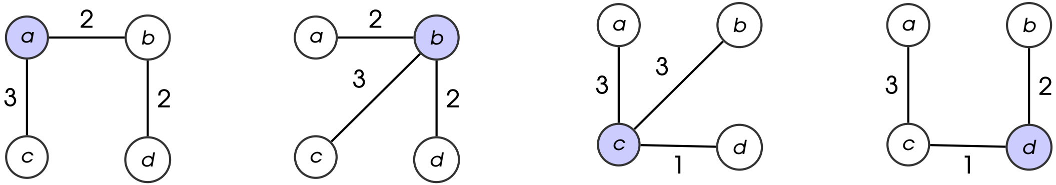
Graph G (edge label is weight)



MST (global property)



Shortest paths (local property):



from a from b from c from d

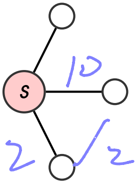
So it is source dependent.

Algorithms for shortest paths

- There are many algorithms to solve this problem
- One of them is **Dijkstra's algorithm**, which assumes the weights of edges are non-negative

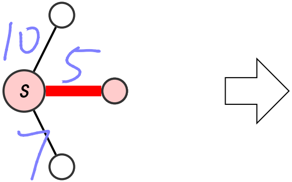
Idea of Dijkstra's algorithm

Choose the edge adjacent to any chosen vertices such that cost of path to source is minimum



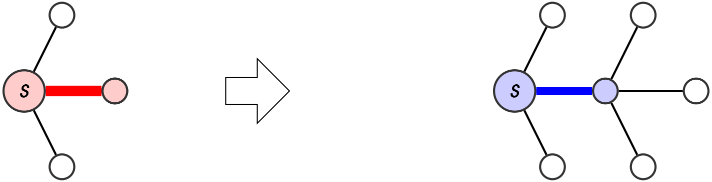
Idea of Dijkstra's algorithm

Choose the edge adjacent to any chosen vertices such that cost of path to source is minimum



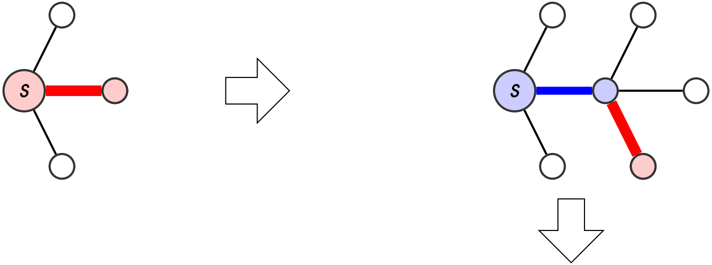
Idea of Dijkstra's algorithm

Choose the edge adjacent to any chosen vertices such that cost of path to source is minimum



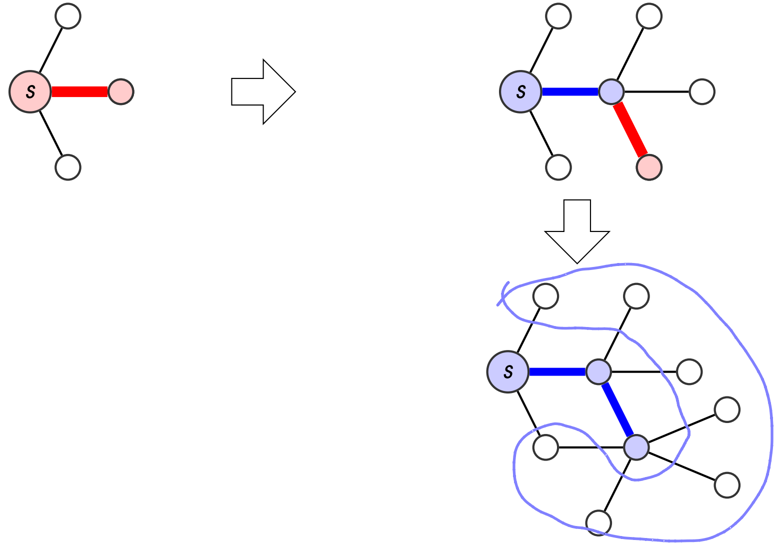
Idea of Dijkstra's algorithm

Choose the edge adjacent to any chosen vertices such that cost of path to source is minimum



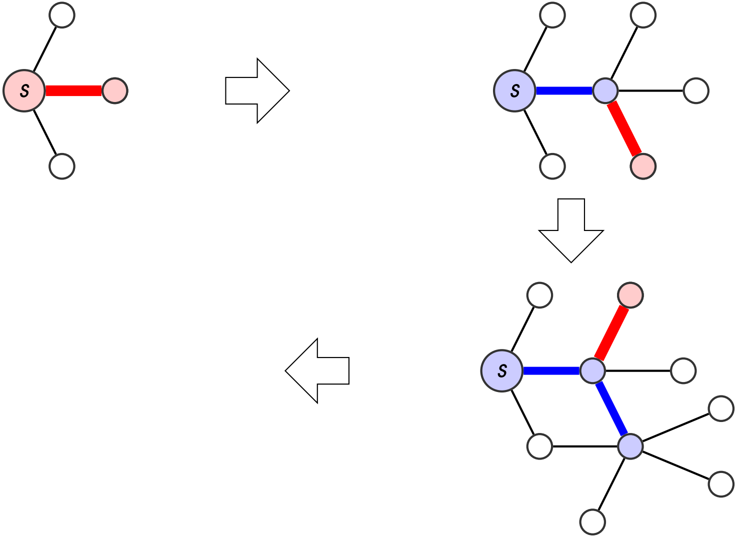
Idea of Dijkstra's algorithm

Choose the edge adjacent to any chosen vertices such that cost of path to source is minimum



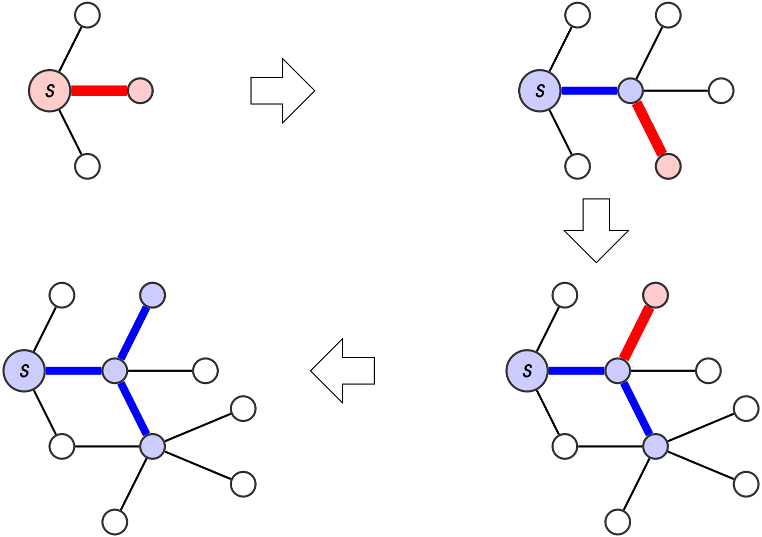
Idea of Dijkstra's algorithm

Choose the edge adjacent to any chosen vertices such that cost of path to source is minimum



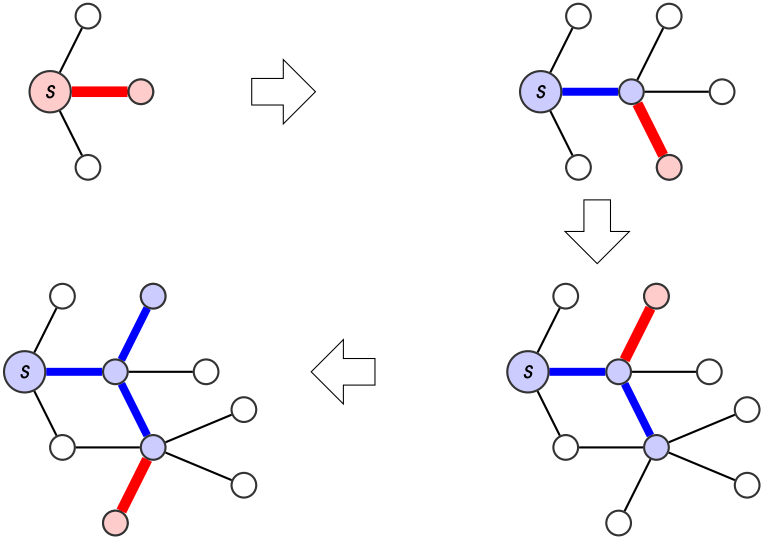
Idea of Dijkstra's algorithm

Choose the edge adjacent to any chosen vertices such that cost of path to source is minimum

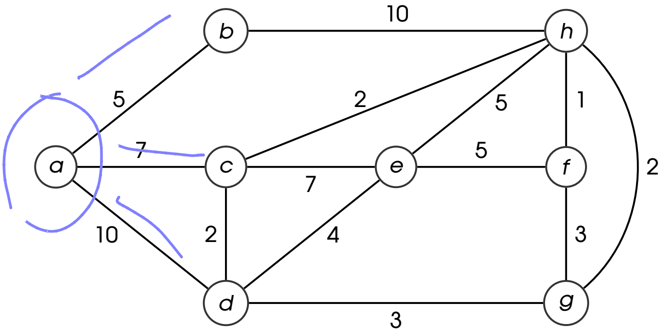


Idea of Dijkstra's algorithm

Choose the edge adjacent to any chosen vertices such that cost of path to source is minimum

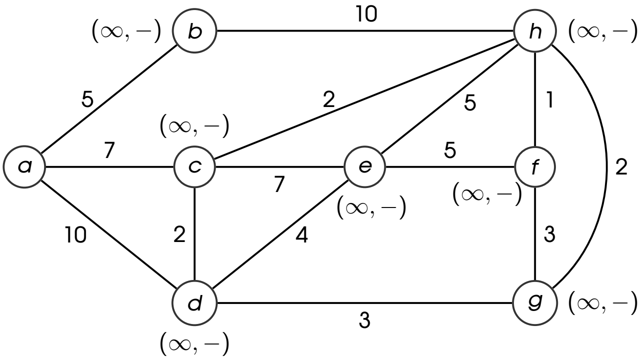


Dijkstra's algorithm - suppose *a* is the source



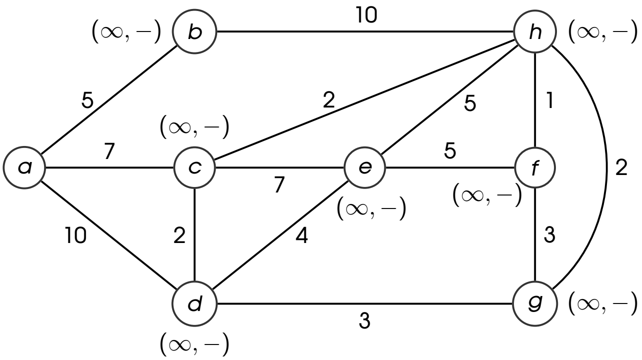
Dijkstra's algorithm - suppose a is the source

Every vertex v keeps 2 labels, initially as $(\infty, -)$: (1) weight of current shortest path from a, (2) the vertex preceding v on that path



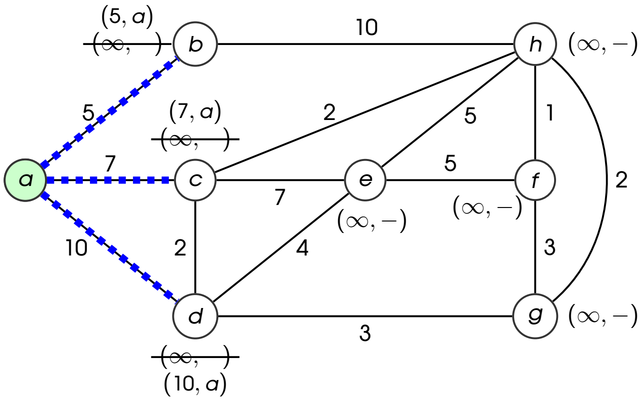
Dijkstra's algorithm - suppose a is the source

Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round



Dijkstra's algorithm - suppose *a* is the source

Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round

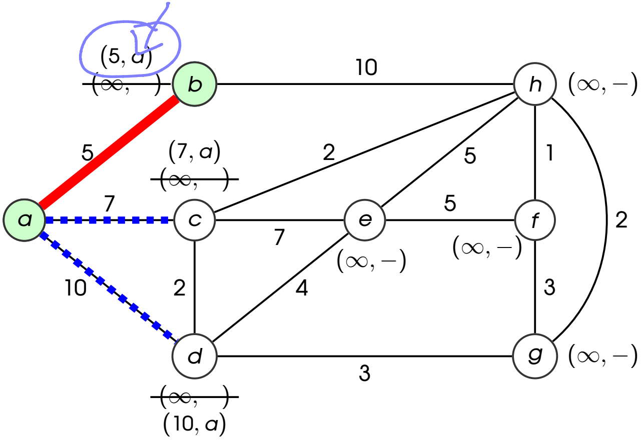


Update *b*, *c*, *d*

Choose from *b*, *c*, *d*:

Dijkstra's algorithm - suppose *a* is the source

Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round

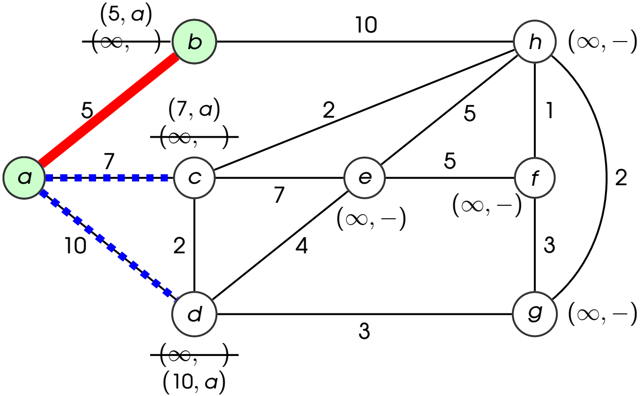


Update *b*, *c*, *d*

Choose from *b*, *c*, *d*: **(*a*, *b*) is chosen**

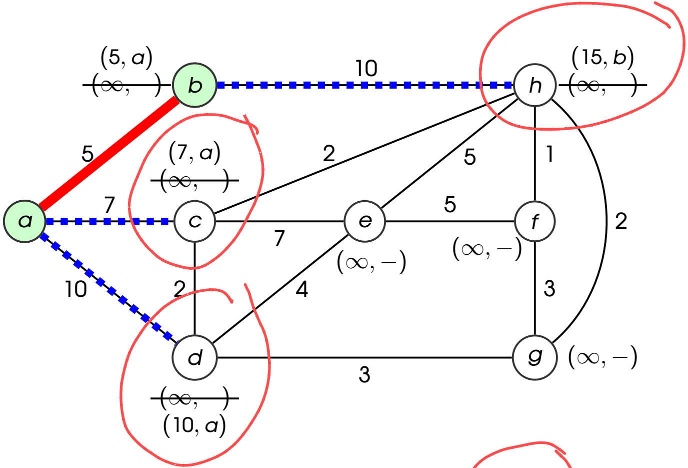
Dijkstra's algorithm - suppose *a* is the source

Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round



Dijkstra's algorithm - suppose *a* is the source

Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round

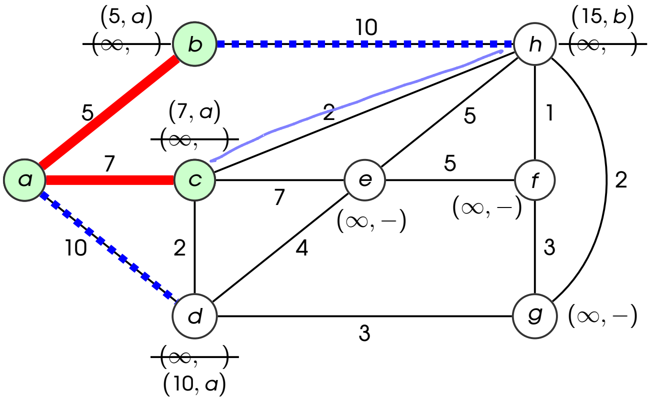


Update *h*

Choose from *c*, *d*, *h*:

Dijkstra's algorithm - suppose a is the source

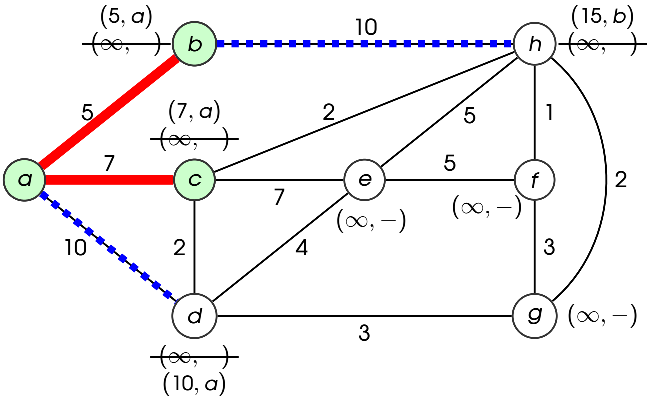
Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round



Update h Choose from c, d, h: **(a, c) is chosen**

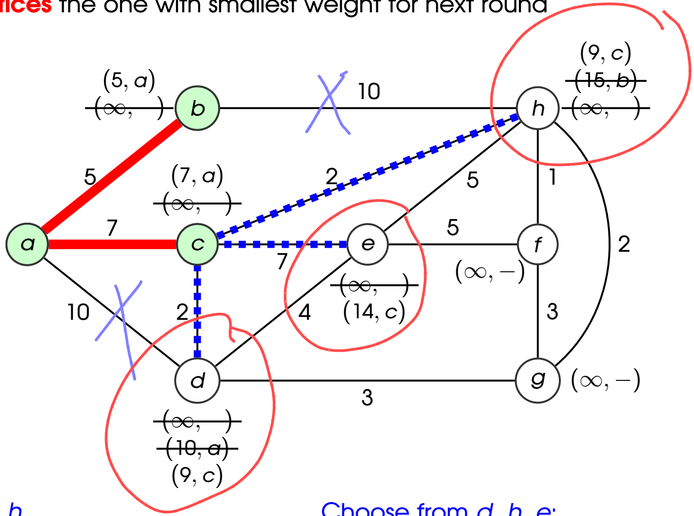
Dijkstra's algorithm - suppose *a* is the source

Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round



Dijkstra's algorithm - suppose a is the source

Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round

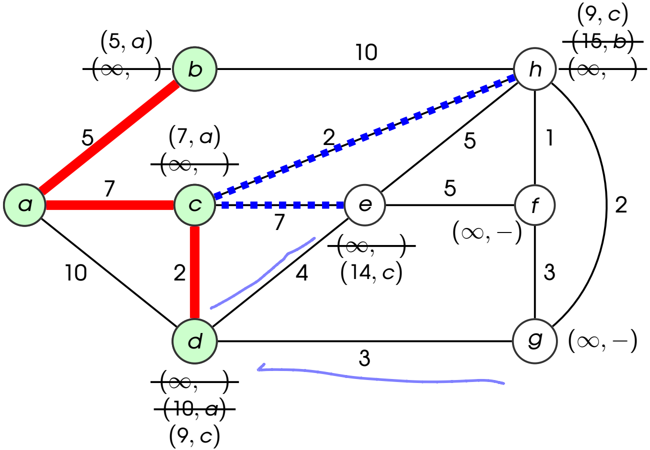


Update d, e, h

Choose from d, h, e:

Dijkstra's algorithm - suppose a is the source

Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round

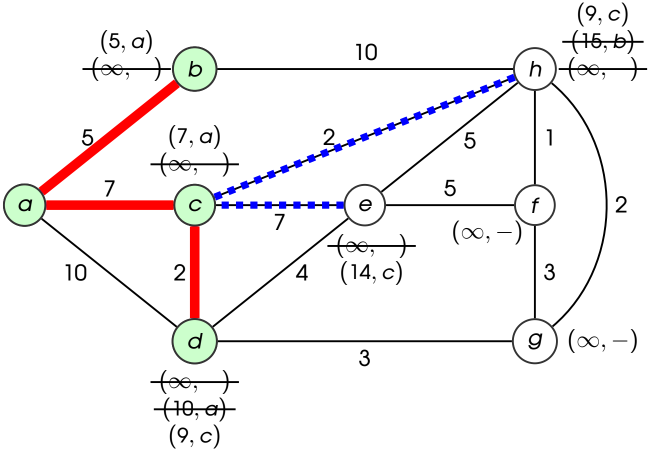


Update d, e, h

Choose from d, h, e: **(c, d) is chosen**

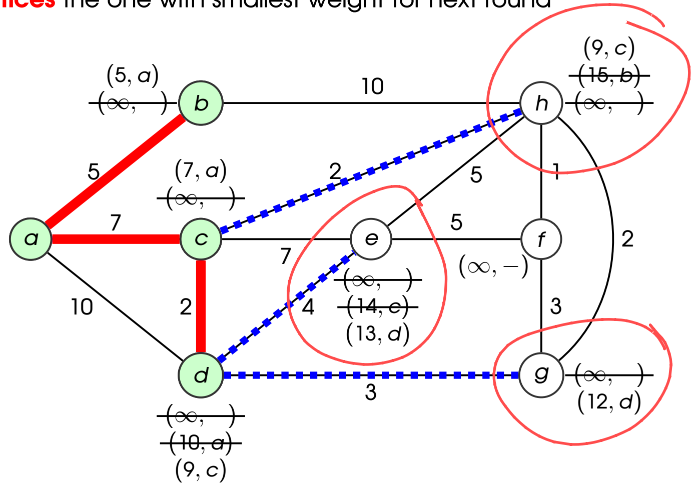
Dijkstra's algorithm - suppose *a* is the source

Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round



Dijkstra's algorithm - suppose *a* is the source

Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round

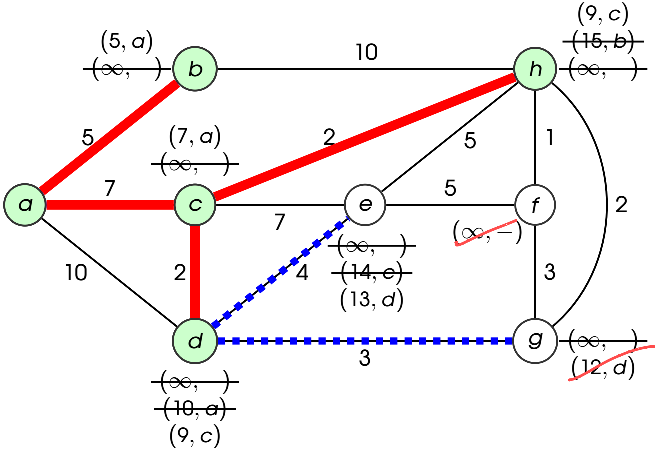


Update *e*, *g*

Choose from *h*, *e*, *g*:

Dijkstra's algorithm - suppose *a* is the source

Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round

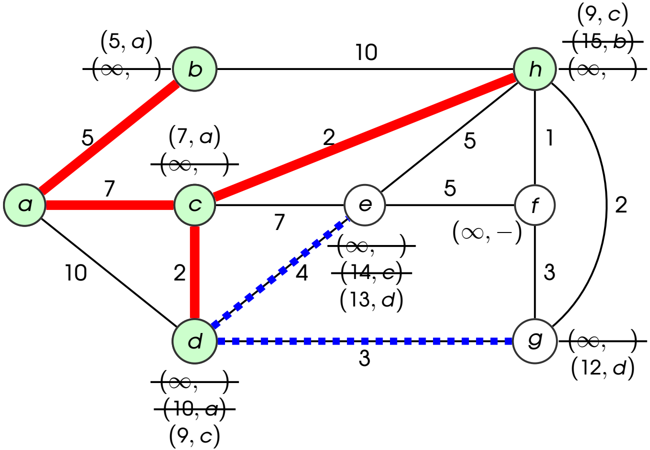


Update *e*, *g*

Choose from *h*, *e*, *g*: **(*c*, *h*) is chosen**

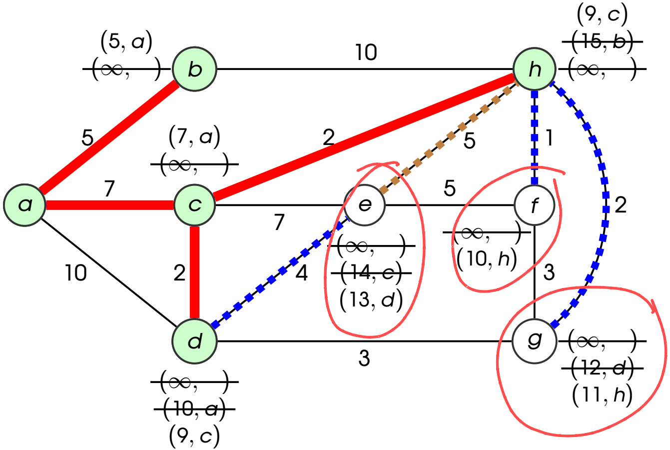
Dijkstra's algorithm - suppose *a* is the source

Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round



Dijkstra's algorithm - suppose *a* is the source

Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round

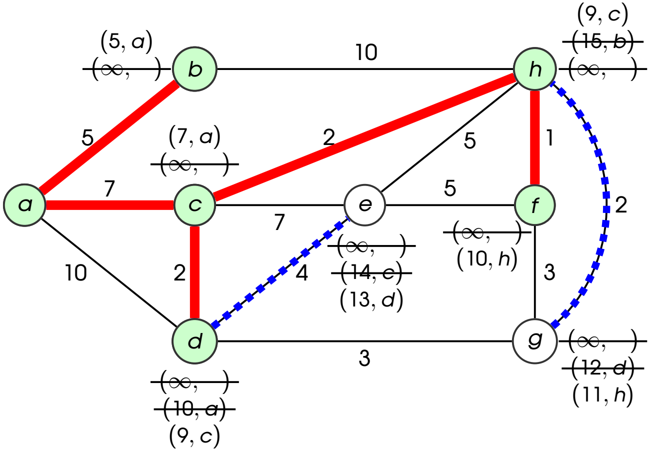


Update *g*, *f*; Unchanged: *e*

Choose from *e*, *g*, *f*:

Dijkstra's algorithm - suppose a is the source

Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round

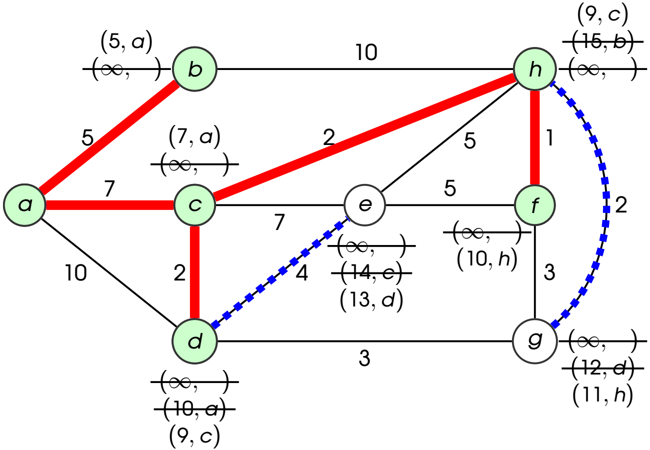


Update g, f; Unchanged: e

Choose from e, g, f: **(h, f) is chosen**

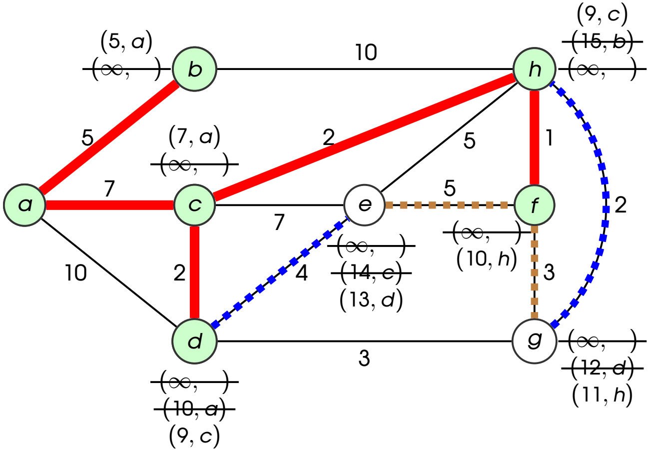
Dijkstra's algorithm - suppose *a* is the source

Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round



Dijkstra's algorithm - suppose *a* is the source

Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round

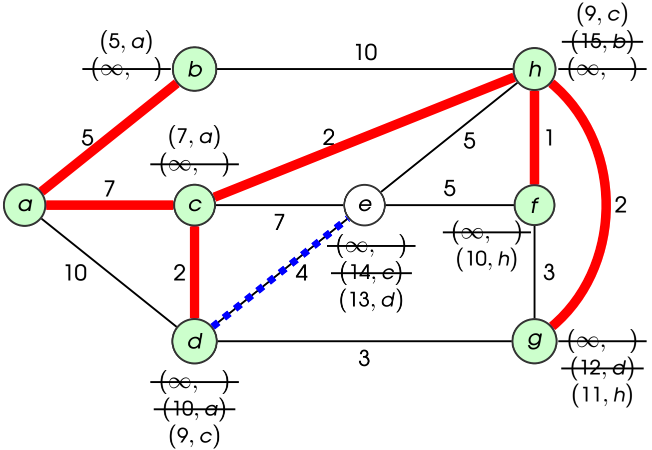


Unchanged: *e*, *g*

Choose from *e*, *g*:

Dijkstra's algorithm - suppose *a* is the source

Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round

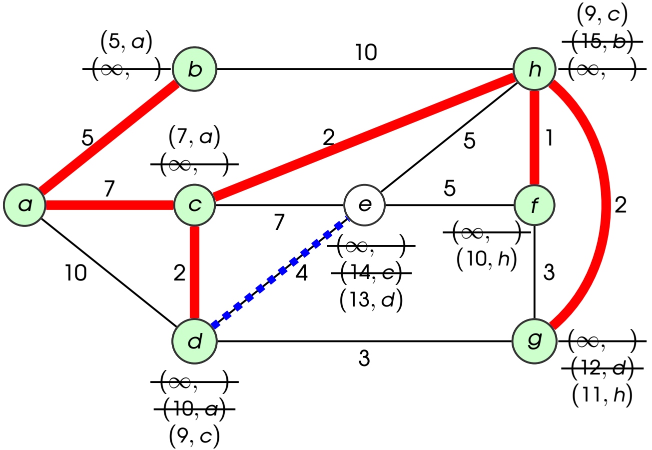


Unchanged: *e*, *g*

Choose from *e*, *g*: **(*h*, *g*) is chosen**

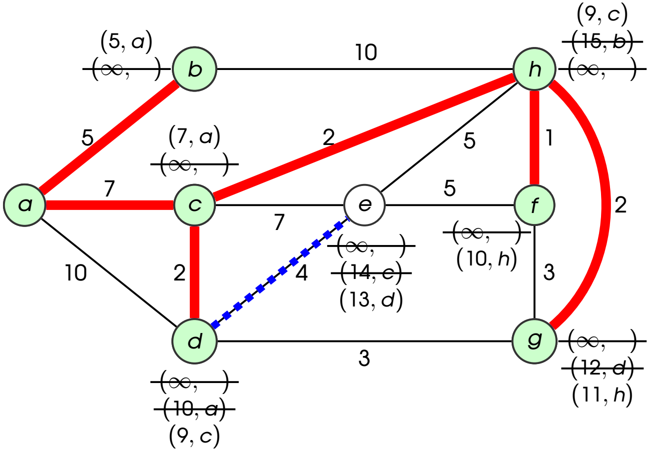
Dijkstra's algorithm - suppose *a* is the source

Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round



Dijkstra's algorithm - suppose *a* is the source

Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round

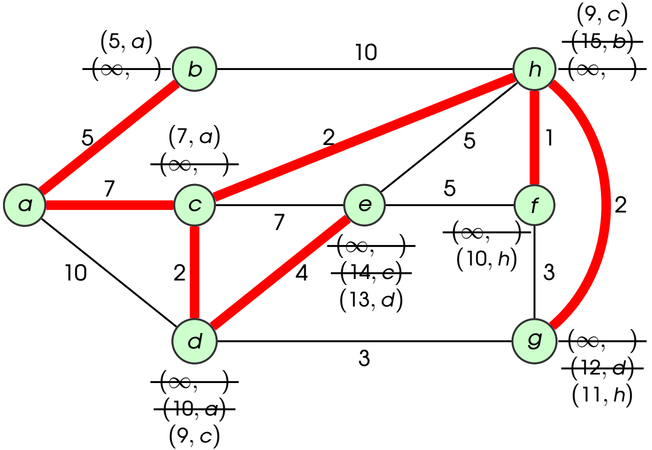


g has no unchosen neighbour $\Longrightarrow$ No change

Choose from *e*:

Dijkstra's algorithm - suppose *a* is the source

Each round: (1) a vertex is picked, neighbours' labels updated, (2) pick from **ALL unchosen vertices** the one with smallest weight for next round



g has no unchosen neighbour $\Longrightarrow$ No change

Choose from *e*: **(*d*, *e*) is chosen**

Alternative presentation

b	c	d	e	f	g	h	chosen
$(\infty, -)$	$(\infty, -)$	$(\infty, -)$	$(\infty, -)$	$(\infty, -)$	$(\infty, -)$	$(\infty, -)$	$(\infty, -)$
$(5, a)$	$(7, a)$	$(10, a)$	$(\infty, -)$	$(\infty, -)$	$(\infty, -)$	$(\infty, -)$	(a, b)
	$(7, a)$	$(10, a)$	$(\infty, -)$	$(\infty, -)$	$(\infty, -)$	$(15, b)$	(a, c)
		$(9, c)$	$(14, c)$	$(\infty, -)$	$(\infty, -)$	$(9, c)$	(c, d)
			$(13, d)$	$(\infty, -)$	$(12, d)$	$(9, c)$	(c, h)
			$(13, d)$	$(10, h)$	$(11, h)$		(h, f)
			$(13, d)$		$(11, h)$		(h, g)
			$(13, d)$				(d, e)

Dijkstra's algorithm

To describe the algorithm using pseudo code, we give some notations

Each vertex v is labelled with two labels:

- a **numeric label** $d(v)$ indicates the length of the shortest path from the source to v found so far
- another label $p(v)$ indicates next-to-last vertex on such path, i.e., the vertex immediately preceding v on that shortest path

Dijkstra's algorithm - Pseudo code

// Given a graph $G = (V, E)$ and a source vertex s

Dijkstra's algorithm - Pseudo code

// Given a graph $G = (V, E)$ and a source vertex s

for every vertex v in the graph do

set $d(v) \leftarrow \infty$ and $p(v) \leftarrow null$

Dijkstra's algorithm - Pseudo code

// Given a graph $G = (V, E)$ and a source vertex s

for every vertex v in the graph do

set $d(v) \leftarrow \infty$ and $p(v) \leftarrow null$

set $d(s) \leftarrow 0$ and $V_T \leftarrow \emptyset$ and $E_T \leftarrow \emptyset$

Dijkstra's algorithm - Pseudo code

// Given a graph $G = (V, E)$ and a source vertex s

for every vertex v in the graph do

  set $d(v) \leftarrow \infty$ and $p(v) \leftarrow null$

set $d(s) \leftarrow 0$ and $V_T \leftarrow \emptyset$ and $E_T \leftarrow \emptyset$

while $V \setminus V_T \neq \emptyset$ do   // there is still some vertex left

begin

end

Dijkstra's algorithm - Pseudo code

// Given a graph $G = (V, E)$ and a source vertex s

for every vertex v in the graph do

  set $d(v) \leftarrow \infty$ and $p(v) \leftarrow null$

set $d(s) \leftarrow 0$ and $V_T \leftarrow \emptyset$ and $E_T \leftarrow \emptyset$

while $V \setminus V_T \neq \emptyset$ do   // there is still some vertex left

begin

  choose the vertex u in $V \setminus V_T$ with minimum $d(u)$

end

Dijkstra's algorithm - Pseudo code

// Given a graph $G = (V, E)$ and a source vertex s

for every vertex v in the graph do

    set $d(v) \leftarrow \infty$ and $p(v) \leftarrow null$

set $d(s) \leftarrow 0$ and $V_T \leftarrow \emptyset$ and $E_T \leftarrow \emptyset$

while $V \setminus V_T \neq \emptyset$ do   // there is still some vertex left

begin

    choose the vertex u in $V \setminus V_T$ with minimum $d(u)$

    set $V_T \leftarrow V_T \cup \{u\}$ and $E_T \leftarrow E_T \cup \{(p(u), u)\}$

end

Dijkstra's algorithm - Pseudo code

// Given a graph $G = (V, E)$ and a source vertex s

for every vertex v in the graph do

    set $d(v) \leftarrow \infty$ and $p(v) \leftarrow null$

set $d(s) \leftarrow 0$ and $V_T \leftarrow \emptyset$ and $E_T \leftarrow \emptyset$

while $V \setminus V_T \neq \emptyset$ do   // there is still some vertex left

begin

    choose the vertex u in $V \setminus V_T$ with minimum $d(u)$

    set $V_T \leftarrow V_T \cup \{u\}$ and $E_T \leftarrow E_T \cup \{(p(u), u)\}$

    for every vertex v in $V \setminus V_T$ that is a neighbour of u do

end

Dijkstra's algorithm - Pseudo code

// Given a graph $G = (V, E)$ and a source vertex s

for every vertex v in the graph do

    set $d(v) \leftarrow \infty$ and $p(v) \leftarrow null$

set $d(s) \leftarrow 0$ and $V_T \leftarrow \emptyset$ and $E_T \leftarrow \emptyset$

while $V \setminus V_T \neq \emptyset$ do   // there is still some vertex left

begin

    choose the vertex u in $V \setminus V_T$ with minimum $d(u)$

    set $V_T \leftarrow V_T \cup \{u\}$ and $E_T \leftarrow E_T \cup \{(p(u), u)\}$

    for every vertex v in $V \setminus V_T$ that is a neighbour of u do

        if $d(u) + w(u, v) < d(v)$ then   // a shorter path is found

end

Dijkstra's algorithm - Pseudo code

// Given a graph $G = (V, E)$ and a source vertex s

for every vertex v in the graph do

  set $d(v) \leftarrow \infty$ and $p(v) \leftarrow null$

set $d(s) \leftarrow 0$ and $V_T \leftarrow \emptyset$ and $E_T \leftarrow \emptyset$

while $V \setminus V_T \neq \emptyset$ do   // there is still some vertex left

begin

  choose the vertex u in $V \setminus V_T$ with minimum $d(u)$

  set $V_T \leftarrow V_T \cup \{u\}$ and $E_T \leftarrow E_T \cup \{(p(u), u)\}$

  for every vertex v in $V \setminus V_T$ that is a neighbour of u do

    if $d(u) + w(u, v) < d(v)$ then   // a shorter path is found

      set $d(v) \leftarrow d(u) + w(u, v)$ and $p(v) \leftarrow u$

end

Dijkstra's algorithm - Pseudo code

$|V| = n \quad |E| = m$

// Given a graph $G = (V, E)$ and a source vertex s

for every vertex v in the graph do

    set $d(v) \leftarrow \infty$ and $p(v) \leftarrow null$

set $d(s) \leftarrow 0$ and $V_T \leftarrow \emptyset$ and $E_T \leftarrow \emptyset$

while $V \setminus V_T \neq \emptyset$ do   // there is still some vertex left $O(n)$ iterations

begin

    choose the vertex u in $V \setminus V_T$ with minimum $d(u)$ — $O(n)$

    set $V_T \leftarrow V_T \cup \{u\}$ and $E_T \leftarrow E_T \cup \{(p(u), u)\}$ — $O(1)$

    for every vertex v in $V \setminus V_T$ that is a neighbour of u do $O(n)$

        if $d(u) + w(u, v) < d(v)$ then   // a shorter path is found

            set $d(v) \leftarrow d(u) + w(u, v)$ and $p(v) \leftarrow u$ $\}\ O(1)$

end

Time complexity?

$O(n^2)$

Summary

Summary: Dijkstra's algorithm for Single-Source Shortest-Paths

Next week: Divide and Conquer algorithms

For note taking