

# Outline

## 1 Introduction

## 2 Minimum Spanning Trees

- Prim's Algorithm
- Kruskal's Algorithm

## 3 Shortest Paths

- Dijkstra's Algorithm

## 4 Network Flows

# Weighted Graph Algorithms

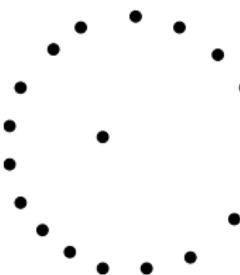
- Beyond DFS/BFS exists an alternate universe of algorithms for edge-weighted graphs.

# Weighted Graph Algorithms

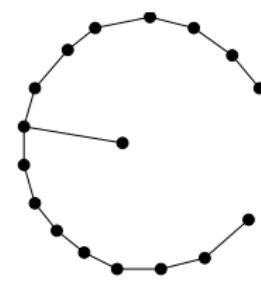
- Beyond DFS/BFS exists an alternate universe of algorithms for **edge-weighted graphs**.
- Our adjacency list representation quietly supported these graphs.  
(just add a **weight** field to each node).

# Definitions

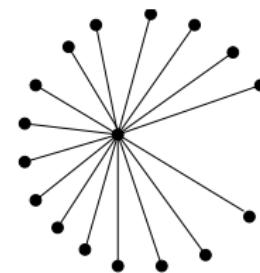
A **tree** is a connected graph with no cycles.



(a)



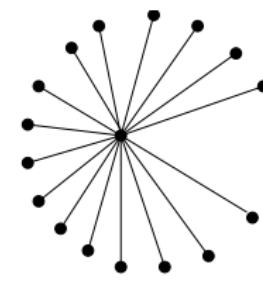
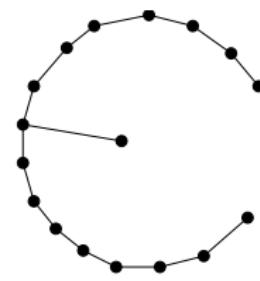
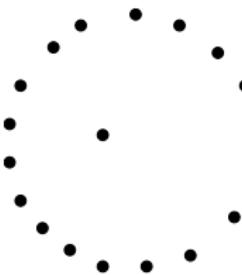
(b)



(c)

# Definitions

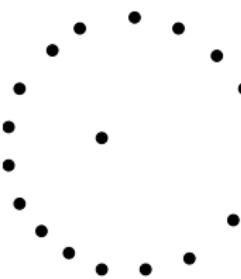
A **spanning tree** is a subgraph of  $G$  which has the same set of vertices of  $G$  and is a tree.



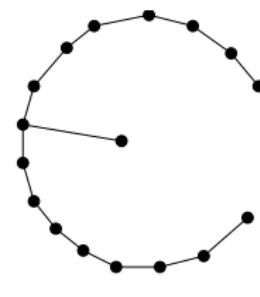
Describe an algorithm to find a spanning tree of a graph.

# Definitions

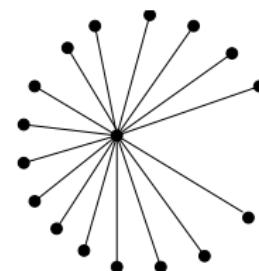
A **spanning tree** is a subgraph of  $G$  which has the same set of vertices of  $G$  and is a tree.



(a)



(b)



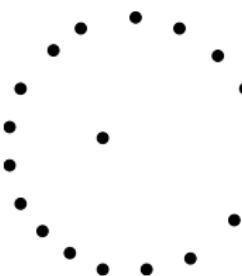
(c)

Describe an algorithm to find a spanning tree of a graph.

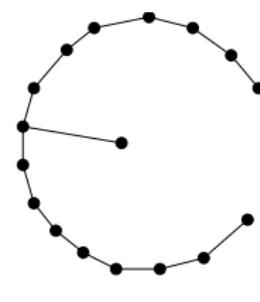
BFS or DFS - just take the tree of discovery or the tree edges

# Definitions

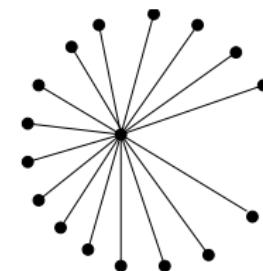
A **spanning tree** is a subgraph of  $G$  which has the same set of vertices of  $G$  and is a tree.



(a)



(b)

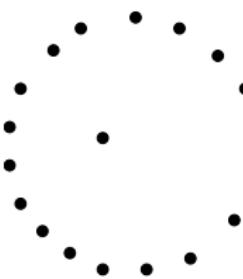


(c)

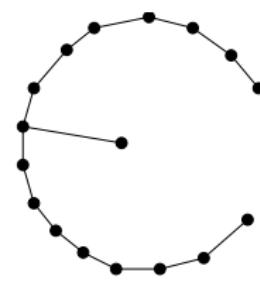
What kind of graph has only one spanning tree?

# Definitions

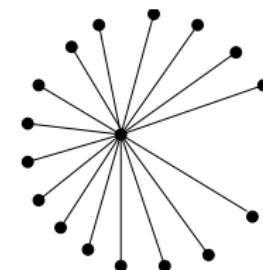
A **spanning tree** is a subgraph of  $G$  which has the same set of vertices of  $G$  and is a tree.



(a)



(b)



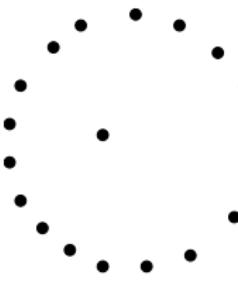
(c)

What kind of graph has only one spanning tree?

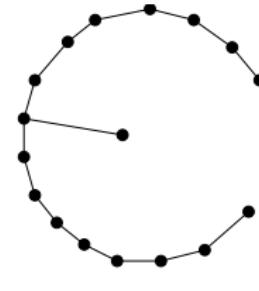
A tree

# Definitions

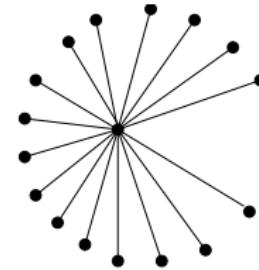
A **minimum spanning tree** of a weighted graph  $G$  is the spanning tree of  $G$  whose edges sum to minimum weight.



(a)



(b)



(c)

# Why study Minimum Spanning Trees?

The MST problem has a long history – the first algorithm dates back at least to 1926!

# Why study Minimum Spanning Trees?

The MST problem has a long history – the first algorithm dates back at least to 1926!

MST is taught in algorithm courses because

- ① it arises in many applications,
- ② it is a problem for which **greedy** algorithms give the optimal answer
- ③ Clever data structures are necessary to make it work well.

# Why study Minimum Spanning Trees?

The MST problem has a long history – the first algorithm dates back at least to 1926!

MST is taught in algorithm courses because

- ① it arises in many applications,
- ② it is a problem for which **greedy** algorithms give the optimal answer
- ③ Clever data structures are necessary to make it work well.

MSTs have many applications

- Constructing Networks
- Clustering points in space into natural groups

# Greedy MST Algorithms

---

**Algorithm:** genericMST( $G$ )

---

- 1  $A = \emptyset$
  - 2 **while**  $A$  does not form a spanning tree **do**
  - 3   | find an edge  $(u, v)$  that is safe for  $A$
  - 4   |  $A = A \cup \{(u, v)\}$
  - 5 **return**  $A$
-

# Greedy MST Algorithms

---

**Algorithm:** genericMST( $G$ )

---

- 1  $A = \emptyset$
  - 2 **while**  $A$  does not form a spanning tree **do**
  - 3   | find an edge  $(u, v)$  that is safe for  $A$
  - 4   |  $A = A \cup \{(u, v)\}$
  - 5 **return**  $A$
- 

## Two Algorithms

# Greedy MST Algorithms

---

**Algorithm:** genericMST( $G$ )

---

- 1  $A = \emptyset$
  - 2 **while**  $A$  does not form a spanning tree **do**
  - 3   | find an edge  $(u, v)$  that is safe for  $A$
  - 4   |  $A = A \cup \{(u, v)\}$
  - 5 **return**  $A$
- 

## Two Algorithms

### ① Prim's Algorithm

# Greedy MST Algorithms

---

**Algorithm:** genericMST( $G$ )

---

- 1  $A = \emptyset$
  - 2 **while**  $A$  does not form a spanning tree **do**
  - 3   | find an edge  $(u, v)$  that is safe for  $A$
  - 4   |  $A = A \cup \{(u, v)\}$
  - 5 **return**  $A$
- 

## Two Algorithms

### ① Prim's Algorithm

Starts from one vertex and grows the rest of the tree one edge at a time

# Greedy MST Algorithms

---

**Algorithm:** genericMST( $G$ )

---

- 1  $A = \emptyset$
  - 2 **while**  $A$  does not form a spanning tree **do**
  - 3   | find an edge  $(u, v)$  that is safe for  $A$
  - 4   |  $A = A \cup \{(u, v)\}$
  - 5 **return**  $A$
- 

## Two Algorithms

### ① Prim's Algorithm

Starts from one vertex and grows the rest of the tree one edge at a time

### ② Kruskal's Algorithm

# Greedy MST Algorithms

---

**Algorithm:** genericMST( $G$ )

---

- 1  $A = \emptyset$
  - 2 **while**  $A$  does not form a spanning tree **do**
  - 3   | find an edge  $(u, v)$  that is safe for  $A$
  - 4   |  $A = A \cup \{(u, v)\}$
  - 5 **return**  $A$
- 

## Two Algorithms

### ① Prim's Algorithm

Starts from one vertex and grows the rest of the tree one edge at a time

### ② Kruskal's Algorithm

Repeatedly add the smallest edge to the spanning tree that does not create a cycle

# Prim's Algorithm

Start with any vertex and grow the rest of the tree one edge at a time

## Definition

A vertex is in the *fringe* if it is not in the minimum spanning tree, but is connected to a node that is in the minimum spanning tree.

# Prim's Algorithm

Start with any vertex and grow the rest of the tree one edge at a time  
To grow the tree, we pick the **cheapest** edge from a tree vertex to a non-tree vertex.

## Definition

A vertex is in the *fringe* if it is not in the minimum spanning tree, but is connected to a node that is in the minimum spanning tree.

# Prim's Algorithm

Start with any vertex and grow the rest of the tree one edge at a time

To grow the tree, we pick the **cheapest** edge from a tree vertex to a non-tree vertex.

If  $G$  is connected, every vertex will appear in the minimum spanning tree

## Definition

A vertex is in the *fringe* if it is not in the minimum spanning tree, but is connected to a node that is in the minimum spanning tree.

# Pseudocode

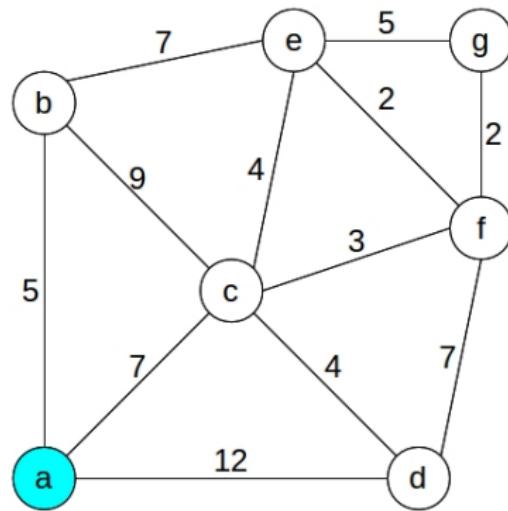
---

**Algorithm:** Prim – MST( $G$ )

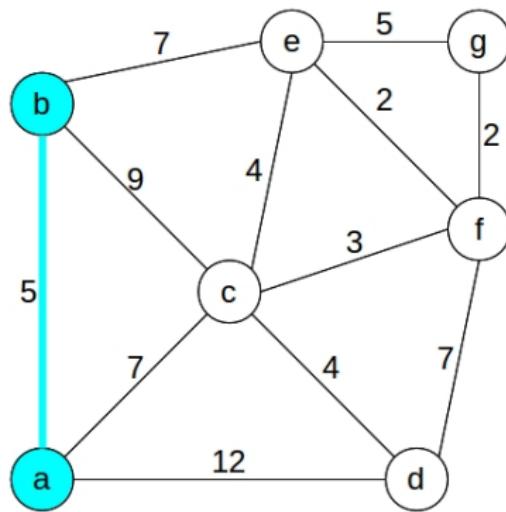
---

- 1 Select an arbitrary vertex  $s$  to start the tree from
  - 2 **while** there are still non-tree vertices **do**
    - 3     Select minimum weight edge between tree and non-tree vertex
    - 4     Add selected edge and vertex to the minimum spanning tree,  $T_{prim}$
  - 5 **return**  $T_{prim}$
-

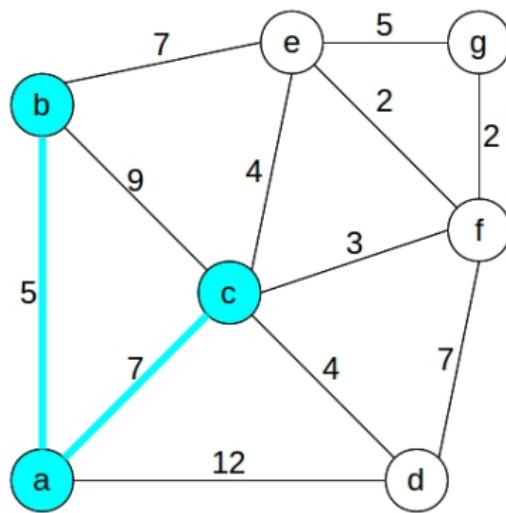
# Example



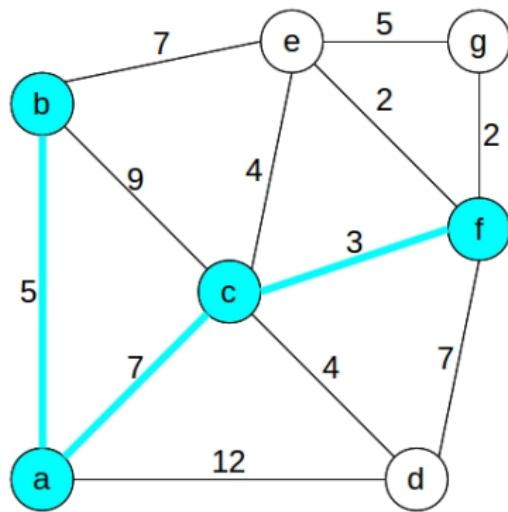
# Example



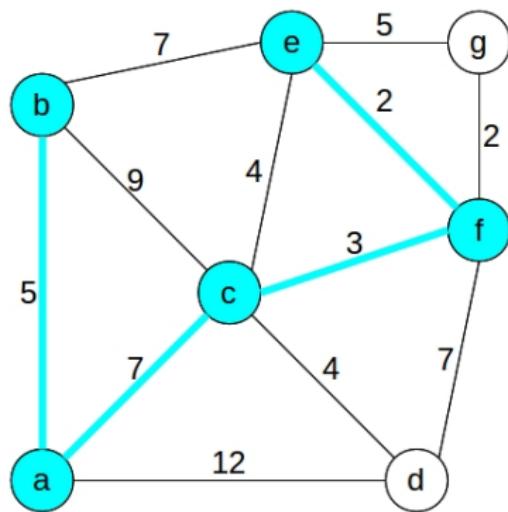
# Example



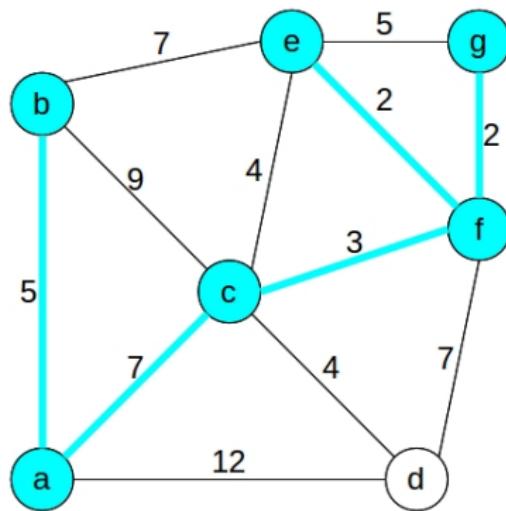
# Example



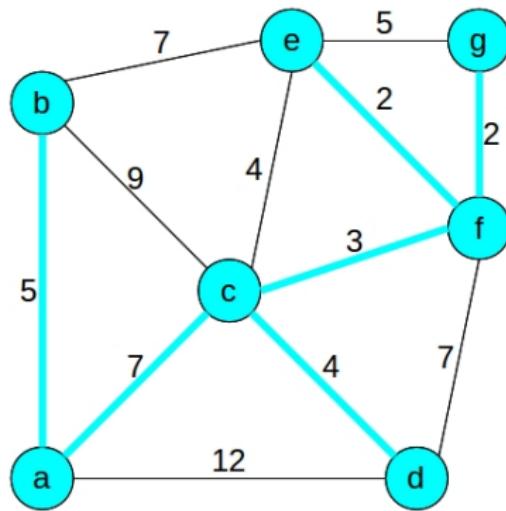
# Example



# Example



# Example



# Proof of Correctness

Prim's algorithm will create a minimum spanning tree.

Prim's algorithm will create a spanning tree since no cycle can be introduced. We must prove that this spanning tree is a tree of minimum cost.

# Proof of Correctness

Prim's algorithm will create a minimum spanning tree.

Prim's algorithm will create a spanning tree since no cycle can be introduced. We must prove that this spanning tree is a tree of minimum cost.

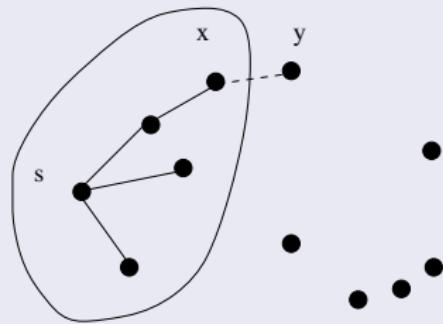
We will use proof by contradiction.

# Proof of Correctness

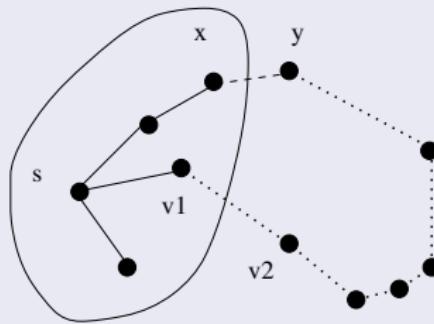
Prim's algorithm will create a minimum spanning tree.

Prim's algorithm will create a spanning tree since no cycle can be introduced. We must prove that this spanning tree is a tree of minimum cost.

We will use proof by contradiction.



(a)



(b)

# Proof of Correctness

Prim's algorithm will create a minimum spanning tree.

Prim's algorithm will create a spanning tree since no cycle can be introduced. We must prove that this spanning tree is a tree of minimum cost.

We will use proof by contradiction.

- Assume Prim's algorithm does not always give the minimum cost spanning tree on some graph. Therefore there exists a graph for which the algorithm fails.

# Proof of Correctness

## Prim's algorithm will create a minimum spanning tree.

Prim's algorithm will create a spanning tree since no cycle can be introduced. We must prove that this spanning tree is a tree of minimum cost.

We will use proof by contradiction.

- Assume Prim's algorithm does not always give the minimum cost spanning tree on some graph. Therefore there exists a graph for which the algorithm fails.
- Therefore there must be a first edge  $(x, y)$  which Prim's algorithm adds such that the partial minimum spanning tree cannot be extended into a minimum spanning tree

# Proof of Correctness

## Prim's algorithm will create a minimum spanning tree.

Prim's algorithm will create a spanning tree since no cycle can be introduced. We must prove that this spanning tree is a tree of minimum cost.

We will use proof by contradiction.

- Assume Prim's algorithm does not always give the minimum cost spanning tree on some graph. Therefore there exists a graph for which the algorithm fails.
- Therefore there must be a first edge  $(x, y)$  which Prim's algorithm adds such that the partial minimum spanning tree cannot be extended into a minimum spanning tree
- Therefore there is another minimum spanning tree,  $MST(G)$  without the edge  $(x, y)$  in it.

# Proof of Correctness

## Prim's algorithm will create a minimum spanning tree.

Prim's algorithm will create a spanning tree since no cycle can be introduced. We must prove that this spanning tree is a tree of minimum cost.

We will use proof by contradiction.

- Assume Prim's algorithm does not always give the minimum cost spanning tree on some graph. Therefore there exists a graph for which the algorithm fails.
- Therefore there must be a first edge  $(x, y)$  which Prim's algorithm adds such that the partial minimum spanning tree cannot be extended into a minimum spanning tree
- Therefore there is another minimum spanning tree,  $MST(G)$  without the edge  $(x, y)$  in it.
- In  $MST(G)$  there must be a path from  $x$  to  $y$ , without using the edge  $(x, y)$ , since the tree is connected

# Proof of Correctness

## Prim's algorithm will create a minimum spanning tree.

Prim's algorithm will create a spanning tree since no cycle can be introduced. We must prove that this spanning tree is a tree of minimum cost.

We will use proof by contradiction.

- Assume Prim's algorithm does not always give the minimum cost spanning tree on some graph. Therefore there exists a graph for which the algorithm fails.
- Therefore there must be a first edge  $(x, y)$  which Prim's algorithm adds such that the partial minimum spanning tree cannot be extended into a minimum spanning tree
- Therefore there is another minimum spanning tree,  $MST(G)$  without the edge  $(x, y)$  in it.
- In  $MST(G)$  there must be a path from  $x$  to  $y$ , without using the edge  $(x, y)$ , since the tree is connected
- Let  $(w, z)$  be the first edge on this path with one endpoint in the minimum spanning that we had before adding  $(x, y)$

# Proof of Correctness

## Prim's algorithm will create a minimum spanning tree.

Prim's algorithm will create a spanning tree since no cycle can be introduced. We must prove that this spanning tree is a tree of minimum cost.

We will use proof by contradiction.

- Assume Prim's algorithm does not always give the minimum cost spanning tree on some graph. Therefore there exists a graph for which the algorithm fails.
- Therefore there must be a first edge  $(x, y)$  which Prim's algorithm adds such that the partial minimum spanning tree cannot be extended into a minimum spanning tree
- Therefore there is another minimum spanning tree,  $MST(G)$  without the edge  $(x, y)$  in it.
- In  $MST(G)$  there must be a path from  $x$  to  $y$ , without using the edge  $(x, y)$ , since the tree is connected
- Let  $(w, z)$  be the first edge on this path with one endpoint in the minimum spanning that we had before adding  $(x, y)$
- Replace  $(w, z)$  with  $(x, y)$  to get a different spanning tree

# Proof of Correctness

## Prim's algorithm will create a minimum spanning tree.

Prim's algorithm will create a spanning tree since no cycle can be introduced. We must prove that this spanning tree is a tree of minimum cost.

We will use proof by contradiction.

- Assume Prim's algorithm does not always give the minimum cost spanning tree on some graph. Therefore there exists a graph for which the algorithm fails.
- Therefore there must be a first edge  $(x, y)$  which Prim's algorithm adds such that the partial minimum spanning tree cannot be extended into a minimum spanning tree
- Therefore there is another minimum spanning tree,  $MST(G)$  without the edge  $(x, y)$  in it.
- In  $MST(G)$  there must be a path from  $x$  to  $y$ , without using the edge  $(x, y)$ , since the tree is connected
- Let  $(w, z)$  be the first edge on this path with one endpoint in the minimum spanning that we had before adding  $(x, y)$
- Replace  $(w, z)$  with  $(x, y)$  to get a different spanning tree
- If Prim's algorithm chooses  $(x, y)$  before choosing  $(w, z)$  then  $W(x, y) < W(w, z)$ .

# Proof of Correctness

## Prim's algorithm will create a minimum spanning tree.

Prim's algorithm will create a spanning tree since no cycle can be introduced. We must prove that this spanning tree is a tree of minimum cost.

We will use proof by contradiction.

- Assume Prim's algorithm does not always give the minimum cost spanning tree on some graph. Therefore there exists a graph for which the algorithm fails.
- Therefore there must be a first edge  $(x, y)$  which Prim's algorithm adds such that the partial minimum spanning tree cannot be extended into a minimum spanning tree
- Therefore there is another minimum spanning tree,  $MST(G)$  without the edge  $(x, y)$  in it.
- In  $MST(G)$  there must be a path from  $x$  to  $y$ , without using the edge  $(x, y)$ , since the tree is connected
- Let  $(w, z)$  be the first edge on this path with one endpoint in the minimum spanning that we had before adding  $(x, y)$
- Replace  $(w, z)$  with  $(x, y)$  to get a different spanning tree
- If Prim's algorithm chooses  $(x, y)$  before choosing  $(w, z)$  then  $W(x, y) < W(w, z)$ .
- Therefore replacing  $(w, z)$  with  $(x, y)$  creates a smaller weighted minimum spanning tree

# Proof of Correctness

## Prim's algorithm will create a minimum spanning tree.

Prim's algorithm will create a spanning tree since no cycle can be introduced. We must prove that this spanning tree is a tree of minimum cost.

We will use proof by contradiction.

- Assume Prim's algorithm does not always give the minimum cost spanning tree on some graph. Therefore there exists a graph for which the algorithm fails.
- Therefore there must be a first edge  $(x, y)$  which Prim's algorithm adds such that the partial minimum spanning tree cannot be extended into a minimum spanning tree
- Therefore there is another minimum spanning tree,  $MST(G)$  without the edge  $(x, y)$  in it.
- In  $MST(G)$  there must be a path from  $x$  to  $y$ , without using the edge  $(x, y)$ , since the tree is connected
- Let  $(w, z)$  be the first edge on this path with one endpoint in the minimum spanning that we had before adding  $(x, y)$
- Replace  $(w, z)$  with  $(x, y)$  to get a different spanning tree
- If Prim's algorithm chooses  $(x, y)$  before choosing  $(w, z)$  then  $W(x, y) < W(w, z)$ .
- Therefore replacing  $(w, z)$  with  $(x, y)$  creates a smaller weighted minimum spanning tree
- Therefore  $MST(G)$  is NOT a minimum spanning tree, we have found a contradiction!

# How Fast is Prim's Algorithm?

---

**Algorithm:** Prim – MST( $G$ )

---

- 1 Select an arbitrary vertex  $s$  to start the tree from
  - 2 **while** there are still non-tree vertices **do**
  - 3     Select minimum weight edge between tree and non-tree vertex
  - 4     Add selected edge and vertex to the MST,  $T_{prim}$
  - 5 **return**  $T_{prim}$
-

# How Fast is Prim's Algorithm?

---

**Algorithm:** Prim – MST( $G$ )

---

- 1 Select an arbitrary vertex  $s$  to start the tree from /\*  $O(1)$  \*/
  - 2 **while** there are still non-tree vertices **do**
  - 3     Select minimum weight edge between tree and non-tree vertex
  - 4     Add selected edge and vertex to the MST,  $T_{prim}$
  - 5 **return**  $T_{prim}$
-

# How Fast is Prim's Algorithm?

---

**Algorithm:** Prim – MST( $G$ )

---

- 1 Select an arbitrary vertex  $s$  to start the tree from /\*  $O(1)$  \*/  
/\*  $O(n)$  iterations \*/
  - 2 **while** there are still non-tree vertices **do**
  - 3     Select minimum weight edge between tree and non-tree vertex
  - 4     Add selected edge and vertex to the MST,  $T_{prim}$
  - 5 **return**  $T_{prim}$
-

# How Fast is Prim's Algorithm?

---

**Algorithm:** Prim – MST( $G$ )

---

- 1 Select an arbitrary vertex  $s$  to start the tree from /\*  $O(1)$  \*/  
/\*  $O(n)$  iterations \*/
  - 2 **while** there are still non-tree vertices **do**
  - 3     Select minimum weight edge between tree and non-tree vertex
  - 4     Add selected edge and vertex to the MST,  $T_{prim}$  /\*  $O(1)$  \*/
  - 5 **return**  $T_{prim}$
-

# How Fast is Prim's Algorithm?

---

**Algorithm:** Prim – MST( $G$ )

---

- 1 Select an arbitrary vertex  $s$  to start the tree from /\*  $O(1)$  \*/  
/\*  $O(n)$  iterations \*/
  - 2 **while** there are still non-tree vertices **do**
  - 3     Select minimum weight edge between tree and non-tree vertex
  - 4     Add selected edge and vertex to the MST,  $T_{prim}$  /\*  $O(1)$  \*/
  - 5 **return**  $T_{prim}$
- 

Line 3 depends on the implementation and what data structure we use

# How Fast is Prim's Algorithm?

---

**Algorithm:** Prim – MST( $G$ )

---

- 1 Select an arbitrary vertex  $s$  to start the tree from /\*  $O(1)$  \*/  
/\*  $O(n)$  iterations \*/
  - 2 **while** there are still non-tree vertices **do**
  - 3     Select minimum weight edge between tree and non-tree vertex
  - 4     Add selected edge and vertex to the MST,  $T_{prim}$  /\*  $O(1)$  \*/
  - 5 **return**  $T_{prim}$
- 

## Simple

- Use BFS/DFS to look at all edges that connect a tree vertex to a non-tree vertex. Traversals takes  $O(n + m)$  time. In a connected graph,  $m \geq n - 1$
- You must do this  $n$  times (line 2). The total work is therefore  $O(nm)$
- How do we actually do this? How can we tell if an edge has one vertex in the tree and one vertex outside of the tree?

# How Fast is Prim's Algorithm?

---

**Algorithm:** Prim – MST( $G$ )

---

- 1 Select an arbitrary vertex  $s$  to start the tree from /\*  $O(1)$  \*/  
/\*  $O(n)$  iterations \*/
  - 2 **while** there are still non-tree vertices **do**
  - 3     Select minimum weight edge between tree and non-tree vertex
  - 4     Add selected edge and vertex to the MST,  $T_{prim}$  /\*  $O(1)$  \*/
  - 5 **return**  $T_{prim}$
- 

## Smarter

- Take  $O(n)$  time to find the smallest weighted edge with one end in the tree and one end outside the tree. Therefore  $O(n^2)$  total time.

# How Fast is Prim's Algorithm?

---

**Algorithm:** Prim – MST( $G$ )

---

- 1 Select an arbitrary vertex  $s$  to start the tree from /\*  $O(1)$  \*/  
/\*  $O(n)$  iterations \*/
  - 2 **while** there are still non-tree vertices **do**
  - 3     Select minimum weight edge between tree and non-tree vertex
  - 4     Add selected edge and vertex to the MST,  $T_{prim}$  /\*  $O(1)$  \*/
  - 5 **return**  $T_{prim}$
- 

## Smarter

- Take  $O(n)$  time to find the smallest weighted edge with one end in the tree and one end outside the tree. Therefore  $O(n^2)$  total time.
- Is  $O(n^2)$  better than  $O(nm)$ ?

# How Fast is Prim's Algorithm?

---

**Algorithm:** Prim – MST( $G$ )

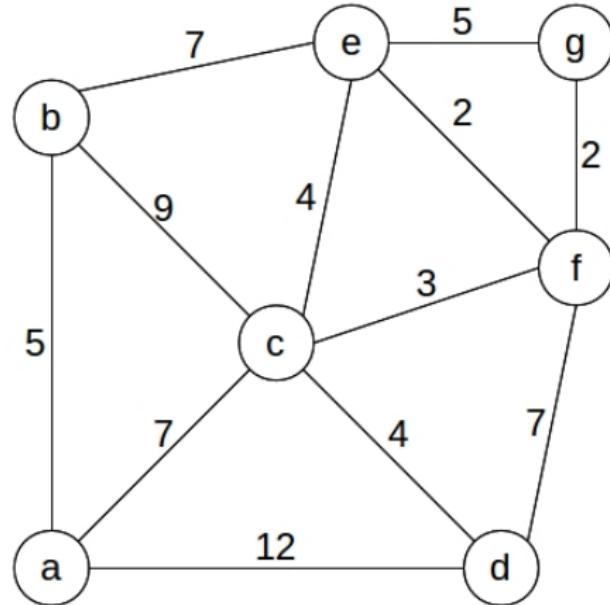
---

- 1 Select an arbitrary vertex  $s$  to start the tree from /\*  $O(1)$  \*/  
/\*  $O(n)$  iterations \*/
  - 2 **while** there are still non-tree vertices **do**
  - 3     Select minimum weight edge between tree and non-tree vertex
  - 4     Add selected edge and vertex to the MST,  $T_{prim}$  /\*  $O(1)$  \*/
  - 5 **return**  $T_{prim}$
- 

## Smarter

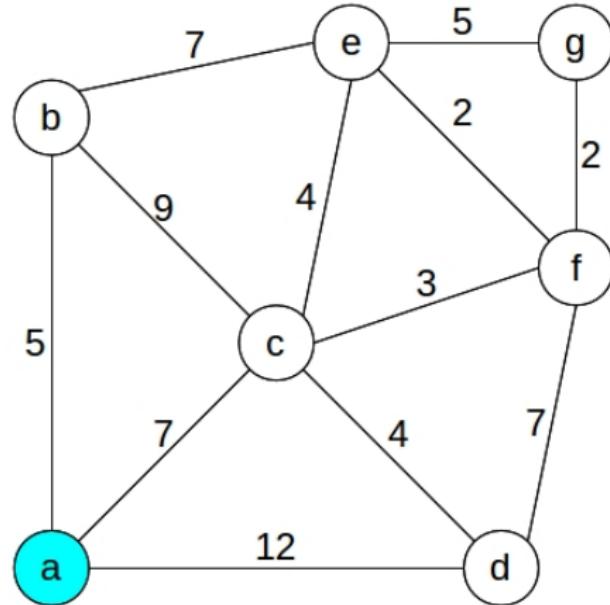
- Take  $O(n)$  time to find the smallest weighted edge with one end in the tree and one end outside the tree. Therefore  $O(n^2)$  total time.
- Is  $O(n^2)$  better than  $O(nm)$ ?
- How do we find the smallest weighted edge in  $O(n)$  time?

# Finding the smallest weighted edge in $O(n)$ time



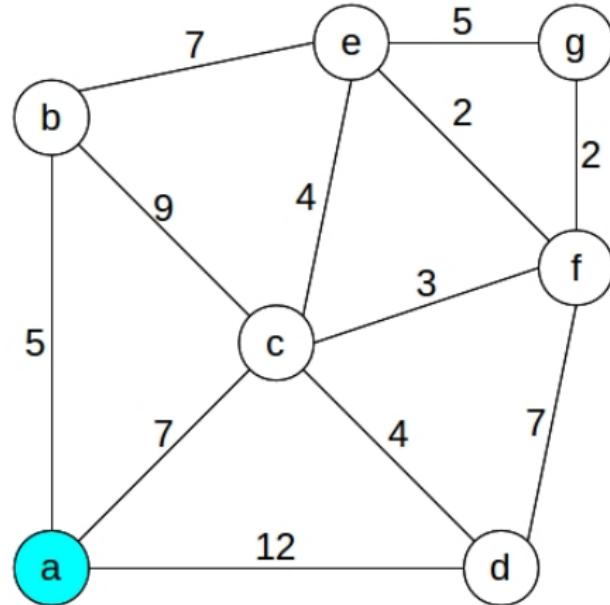
[a]	$\infty$
[b]	$\infty$
[c]	$\infty$
[d]	$\infty$
[e]	$\infty$
[f]	$\infty$
[g]	$\infty$

# Finding the smallest weighted edge in $O(n)$ time



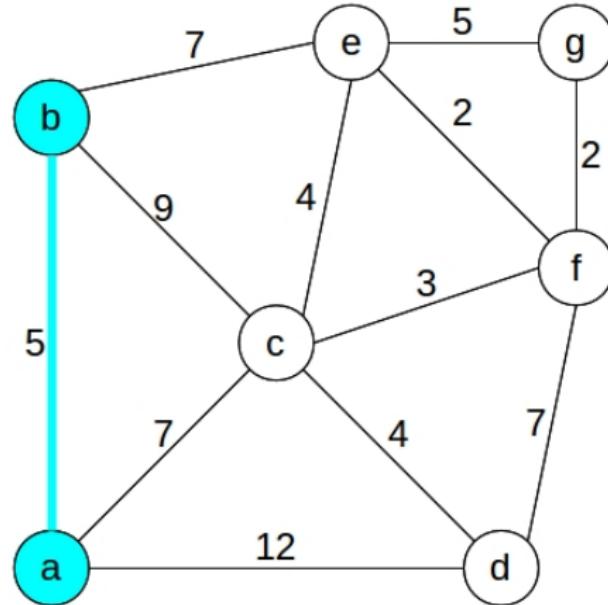
[a]	x
[b]	$\infty$
[c]	$\infty$
[d]	$\infty$
[e]	$\infty$
[f]	$\infty$
[g]	$\infty$

# Finding the smallest weighted edge in $O(n)$ time



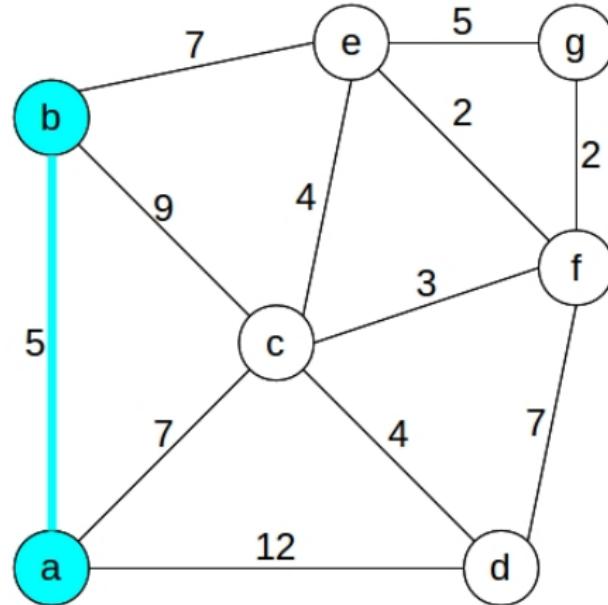
[a]	x
[b]	5
[c]	7
[d]	12
[e]	$\infty$
[f]	$\infty$
[g]	$\infty$

# Finding the smallest weighted edge in $O(n)$ time



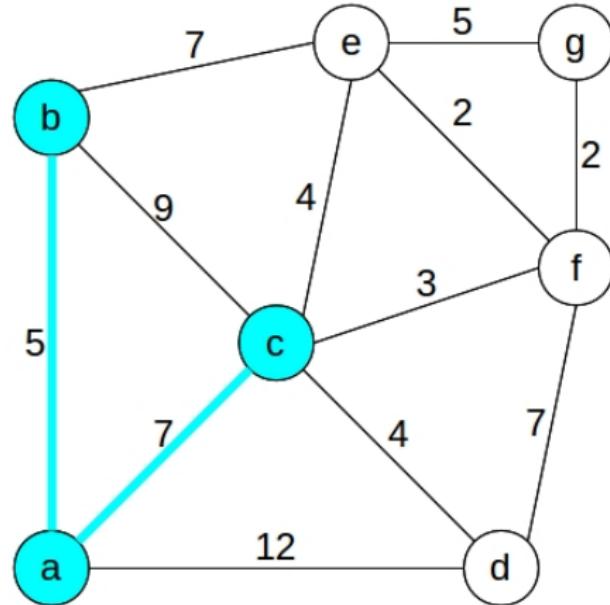
[a]	x
[b]	x
[c]	7
[d]	12
[e]	$\infty$
[f]	$\infty$
[g]	$\infty$

# Finding the smallest weighted edge in $O(n)$ time



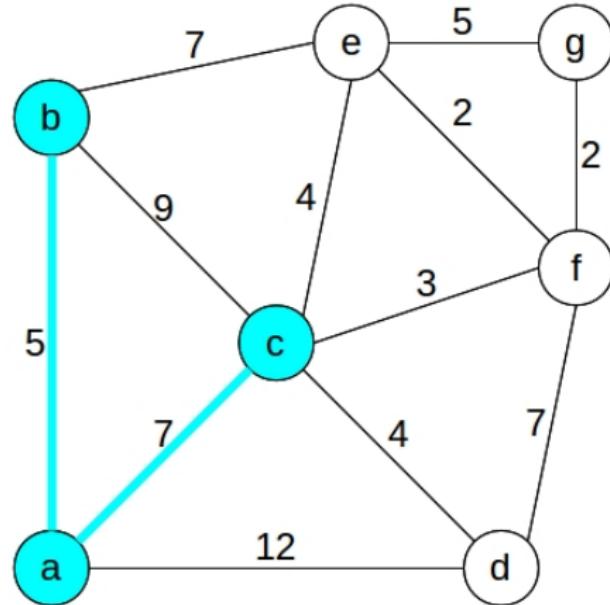
[a]	x
[b]	x
[c]	7
[d]	12
[e]	7
[f]	$\infty$
[g]	$\infty$

# Finding the smallest weighted edge in $O(n)$ time



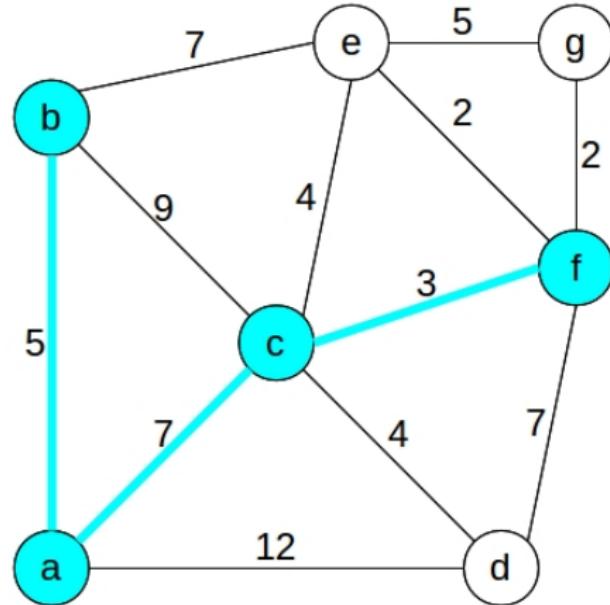
[a]	x
[b]	x
[c]	x
[d]	12
[e]	7
[f]	$\infty$
[g]	$\infty$

# Finding the smallest weighted edge in $O(n)$ time



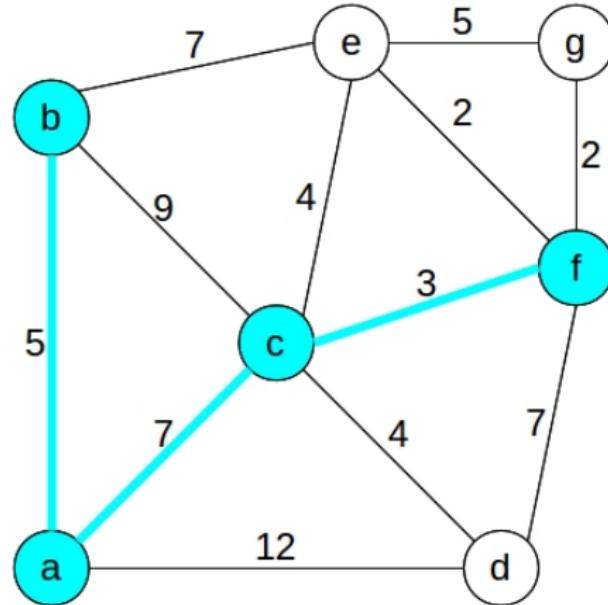
[a]	x
[b]	x
[c]	x
[d]	4
[e]	4
[f]	3
[g]	$\infty$

# Finding the smallest weighted edge in $O(n)$ time



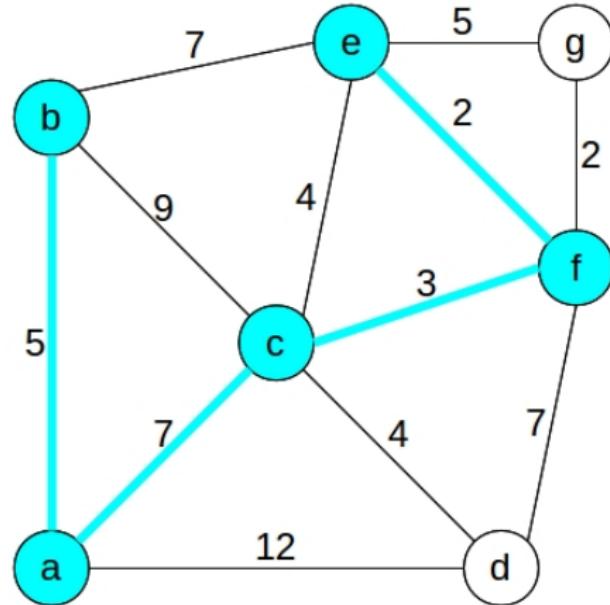
[a]	x
[b]	x
[c]	x
[d]	4
[e]	4
[f]	x
[g]	$\infty$

# Finding the smallest weighted edge in $O(n)$ time



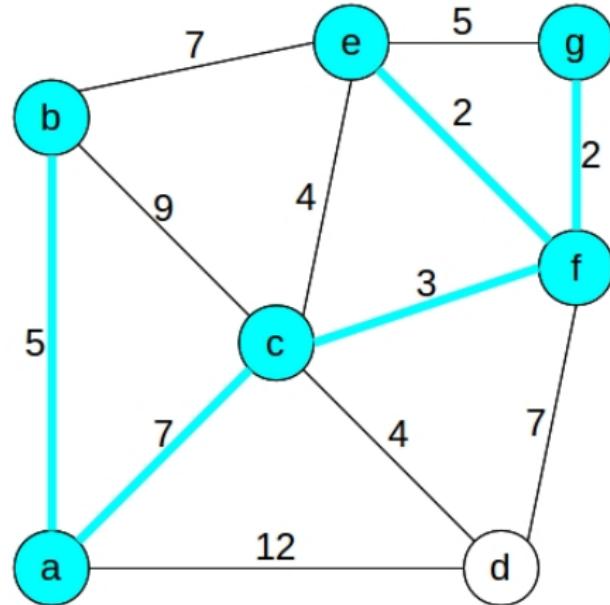
[a]	x
[b]	x
[c]	x
[d]	4
[e]	2
[f]	x
[g]	2

# Finding the smallest weighted edge in $O(n)$ time



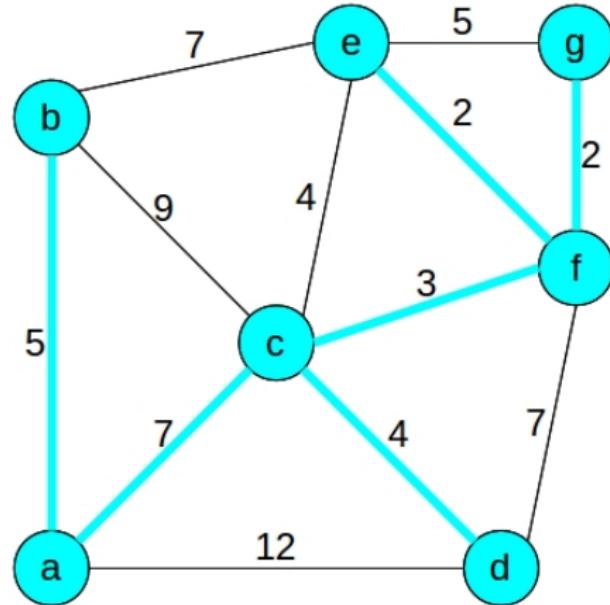
[a]	x
[b]	x
[c]	x
[d]	4
[e]	x
[f]	x
[g]	2

# Finding the smallest weighted edge in $O(n)$ time



[a]	x
[b]	x
[c]	x
[d]	4
[e]	x
[f]	x
[g]	x

# Finding the smallest weighted edge in $O(n)$ time



[a]	x
[b]	x
[c]	x
[d]	x
[e]	x
[f]	x
[g]	x

# How Fast is Prim's Algorithm?

---

**Algorithm:** Prim – MST( $G$ )

---

- 1 Select an arbitrary vertex  $s$  to start the tree from /\*  $O(1)$  \*/  
/\*  $O(n)$  iterations \*/
  - 2 **while** there are still non-tree vertices **do**
  - 3     Select minimum weight edge between tree and non-tree vertex
  - 4     Add selected edge and vertex to the MST,  $T_{prim}$  /\*  $O(1)$  \*/
  - 5 **return**  $T_{prim}$
- 

## Smartest

- Use Fibonacci Heaps (a sophisticated priority queue that allows us to extract the vertex that is the minimum distance from the tree in  $O(\lg n)$  time)
- Takes  $O(m + n \lg n)$  time

# Kruskal's Algorithm

Kruskal's algorithm is also greedy. It repeatedly adds the smallest edge to the spanning tree that does not create a cycle.

# Kruskal's Algorithm

Kruskal's algorithm is also greedy. It repeatedly adds the smallest edge to the spanning tree that does not create a cycle.

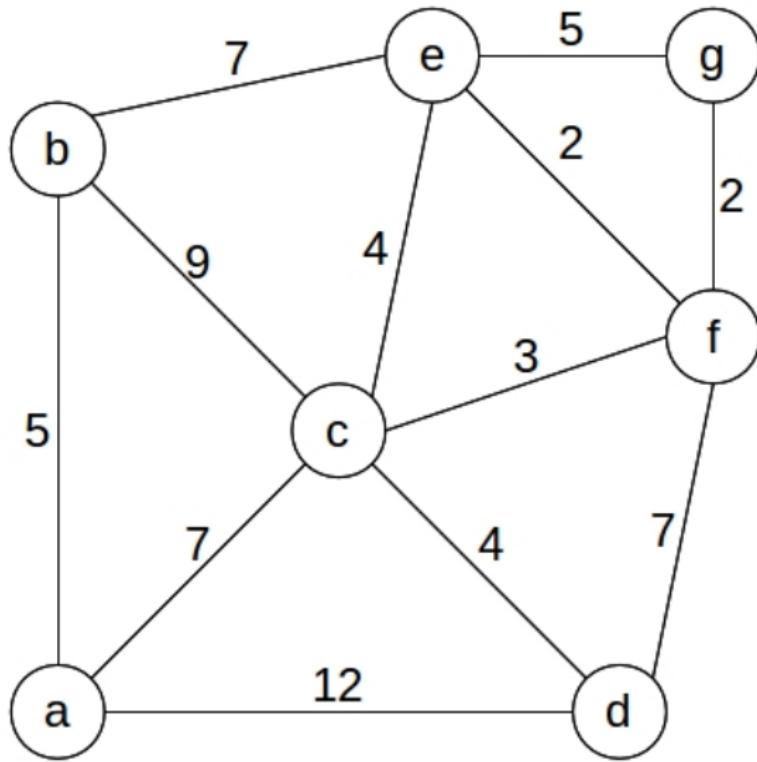
---

**Algorithm:** `kruskals( $G$ )`

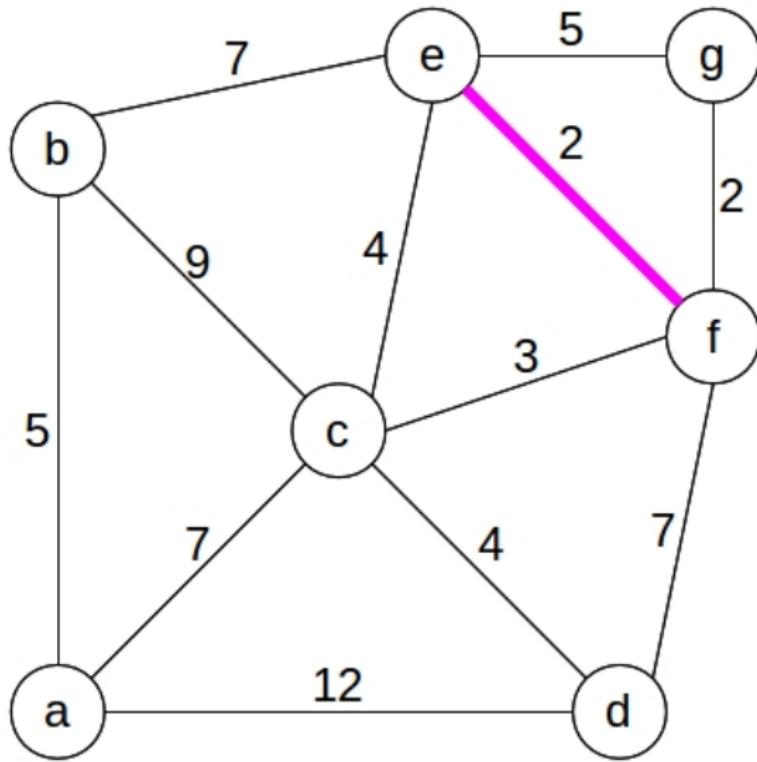
---

- 1 Sort the  $m$  edges
  - 2 **for** each edge in order **do**
  - 3     **if** the edge creates a cycle in the forest we have build thus far **then**
  - 4         Discard
  - 5     **else**
  - 6         Add to forest
-

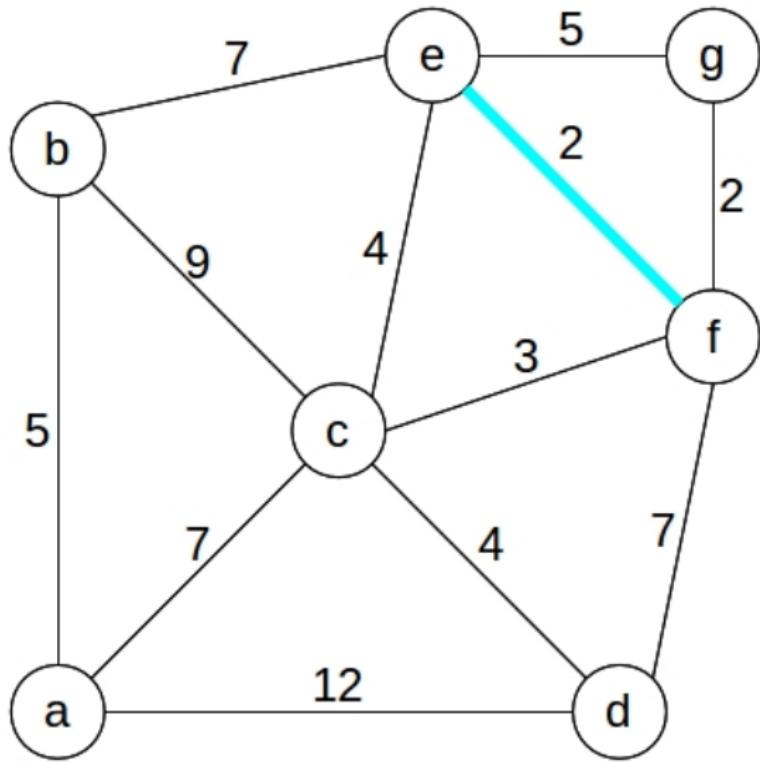
## Example



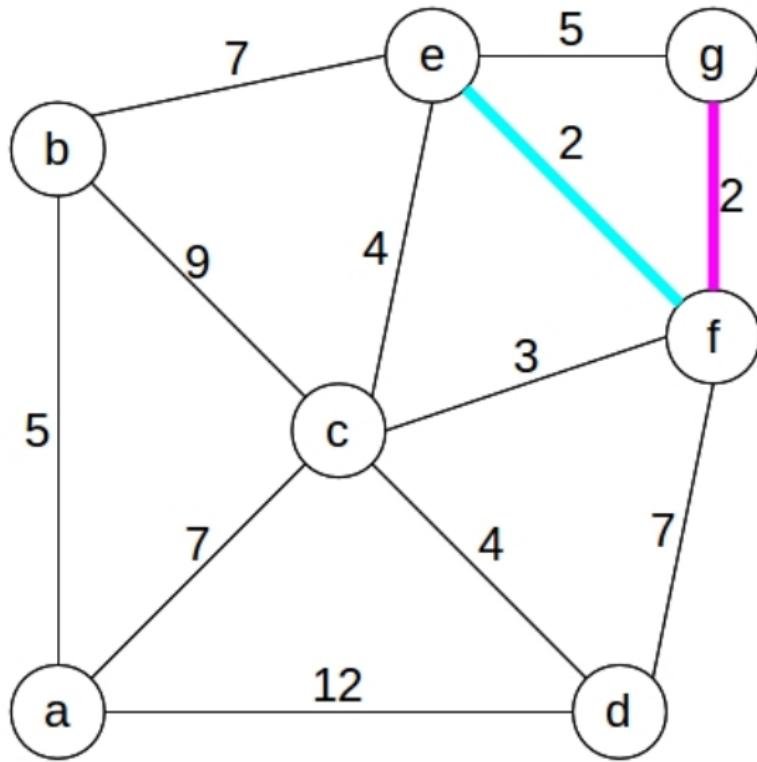
## Example



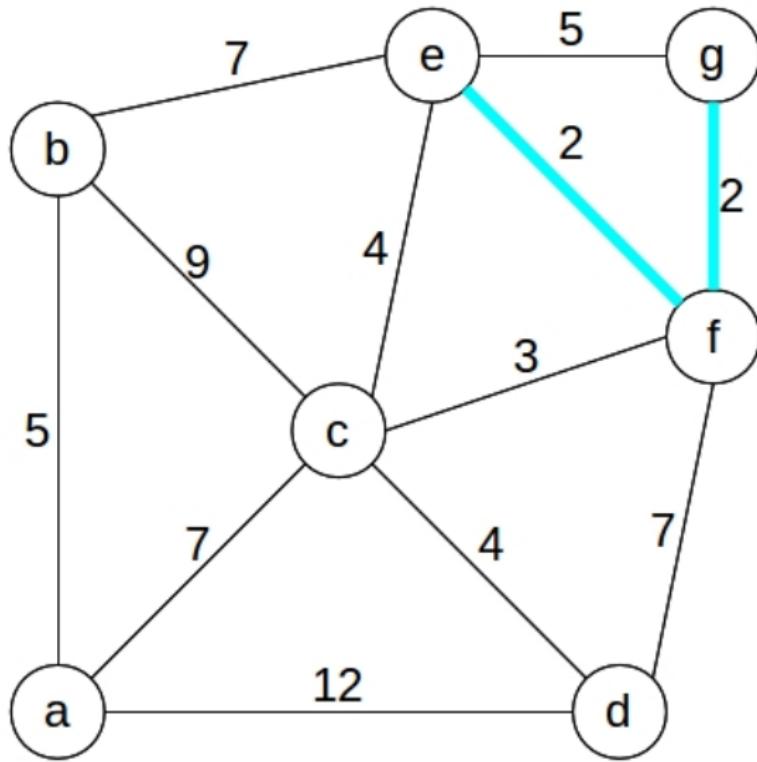
## Example



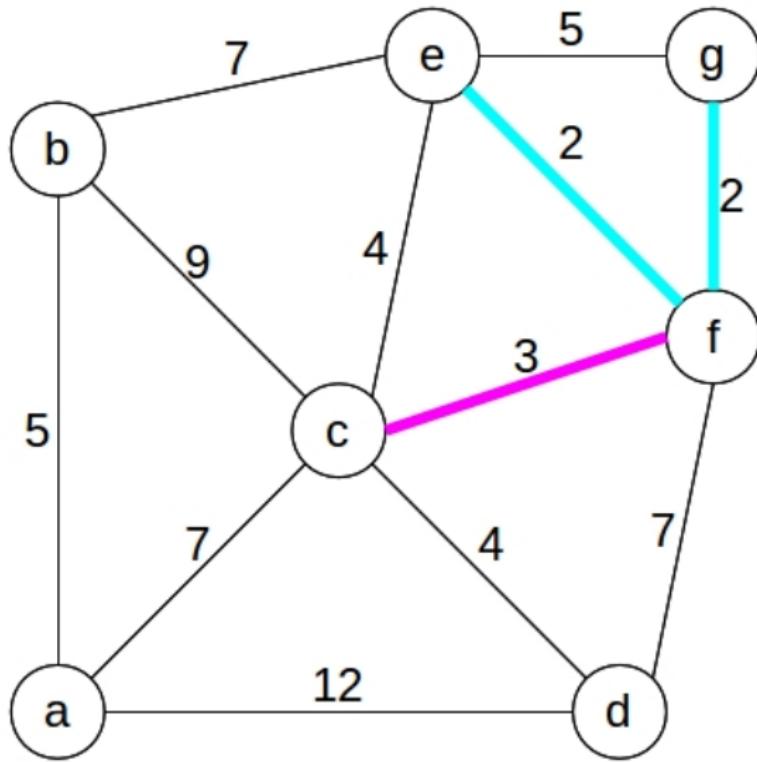
## Example



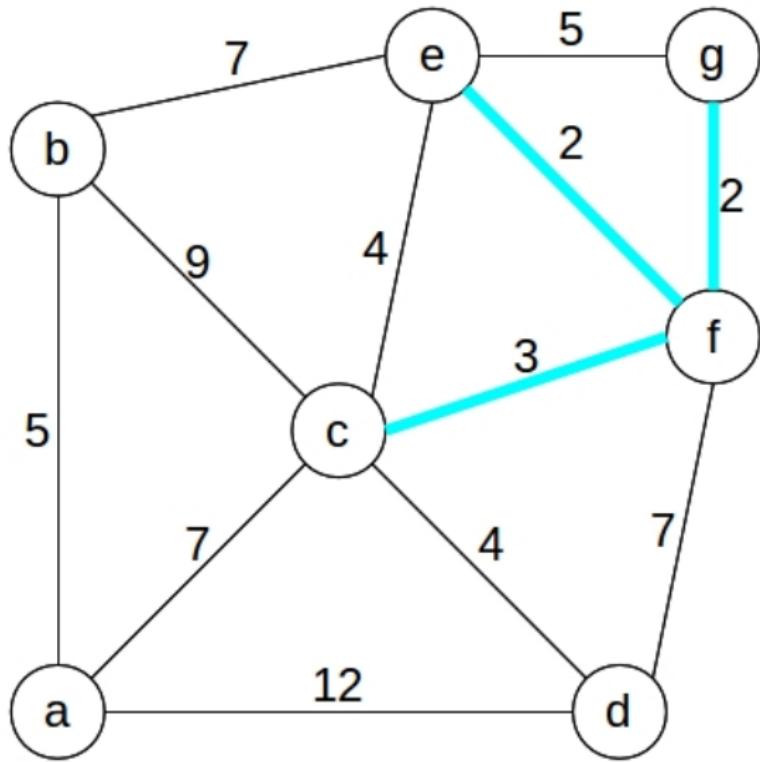
## Example



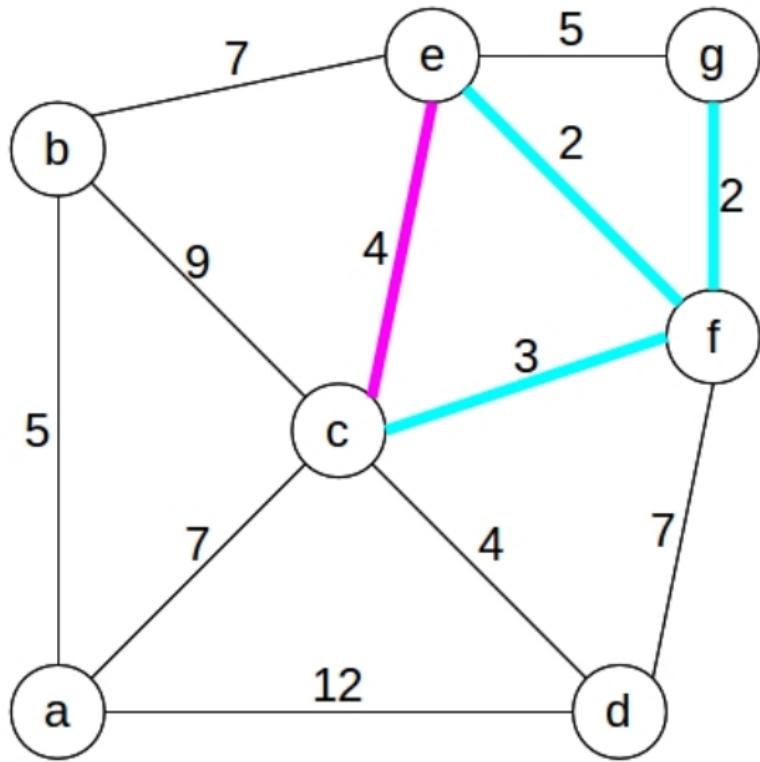
## Example



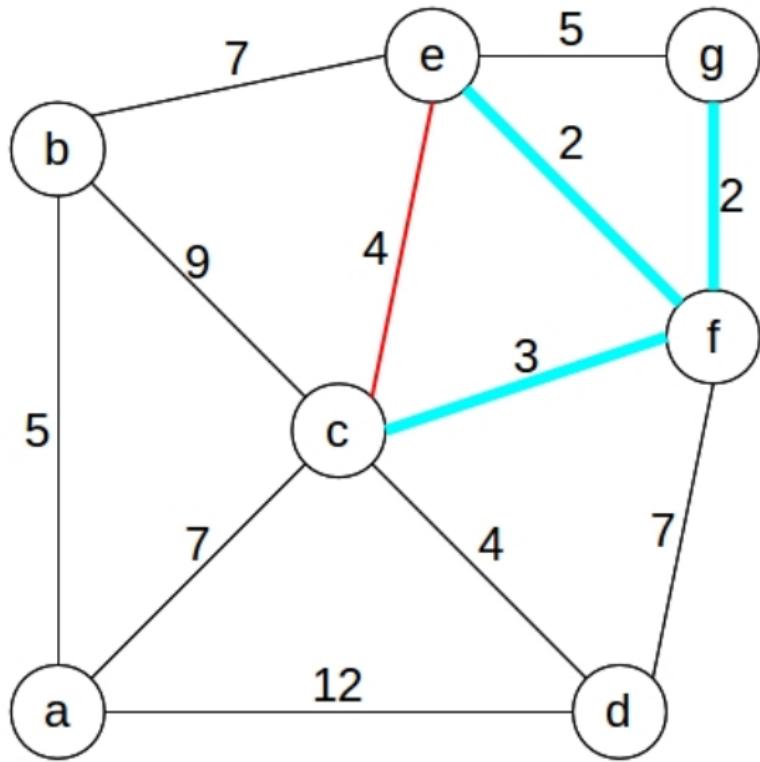
## Example



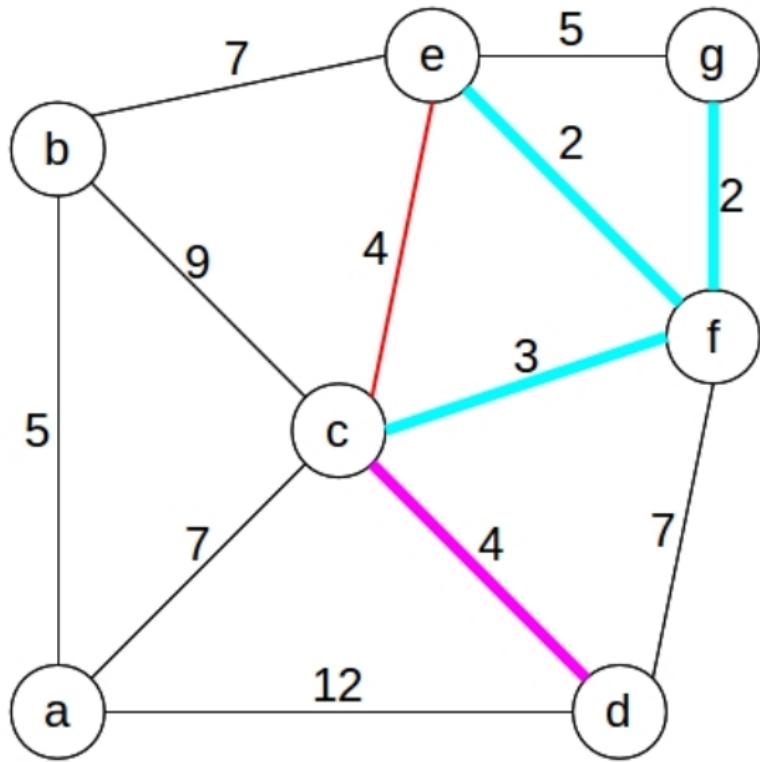
## Example



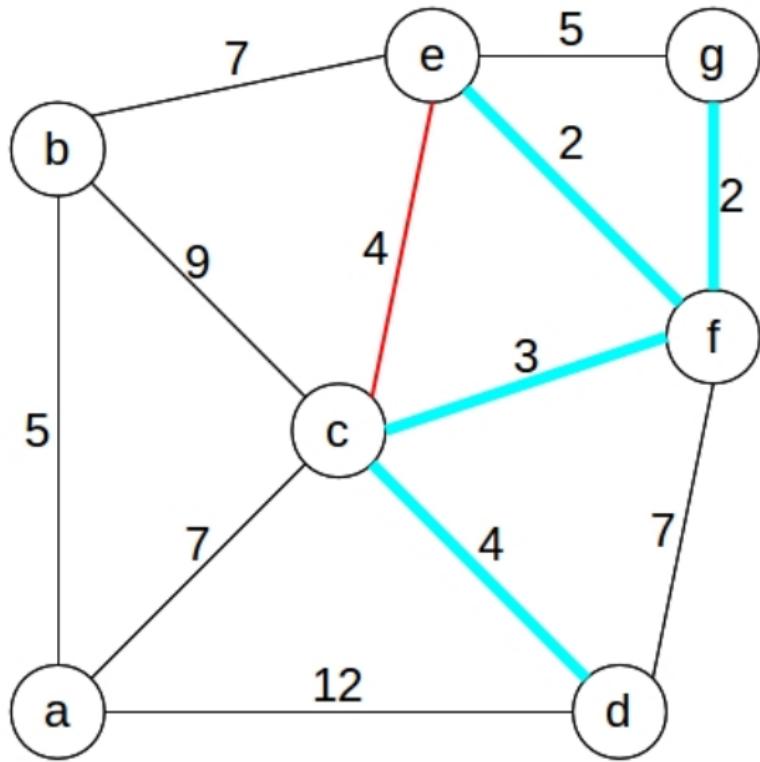
## Example



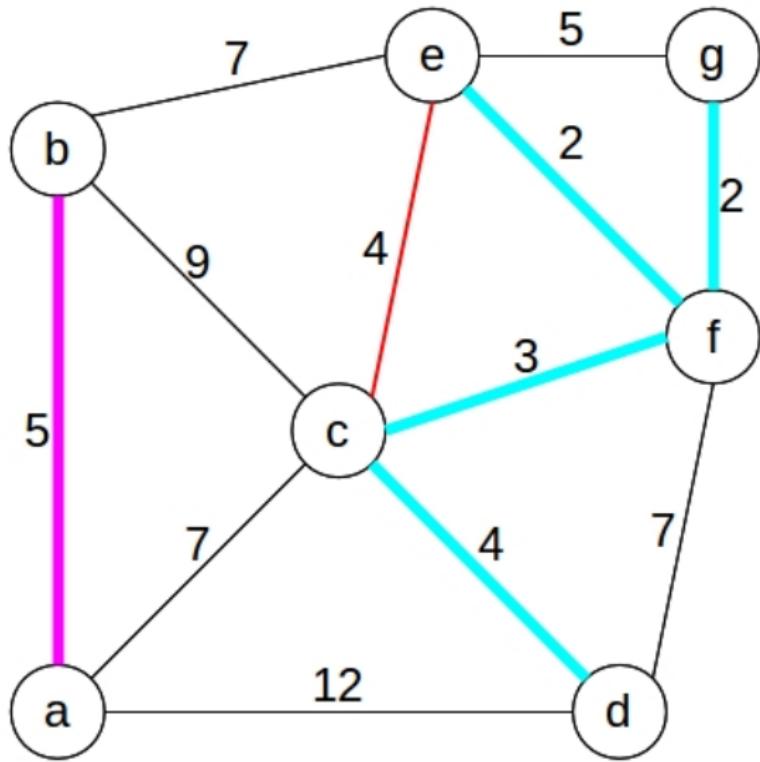
## Example



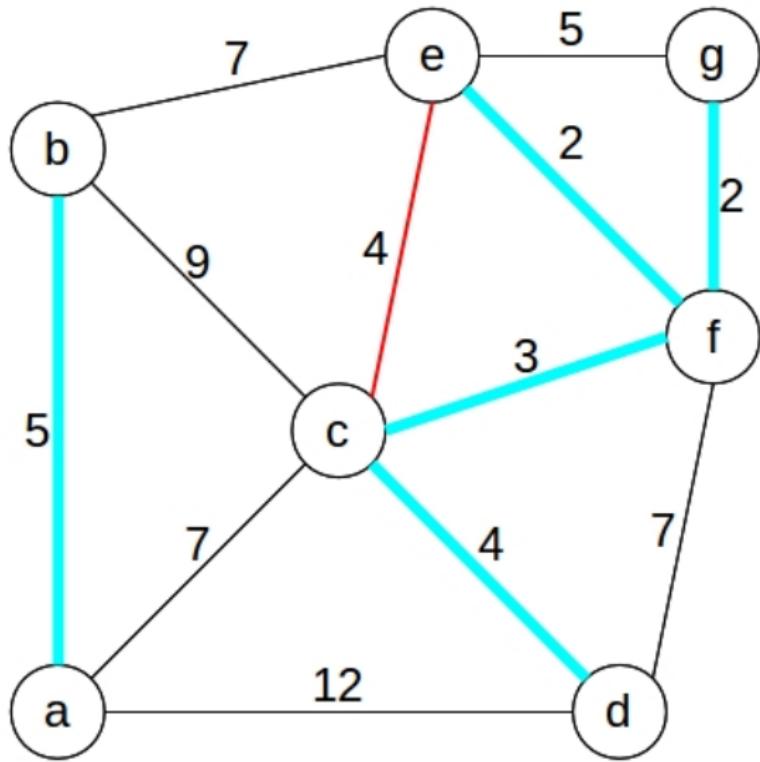
## Example



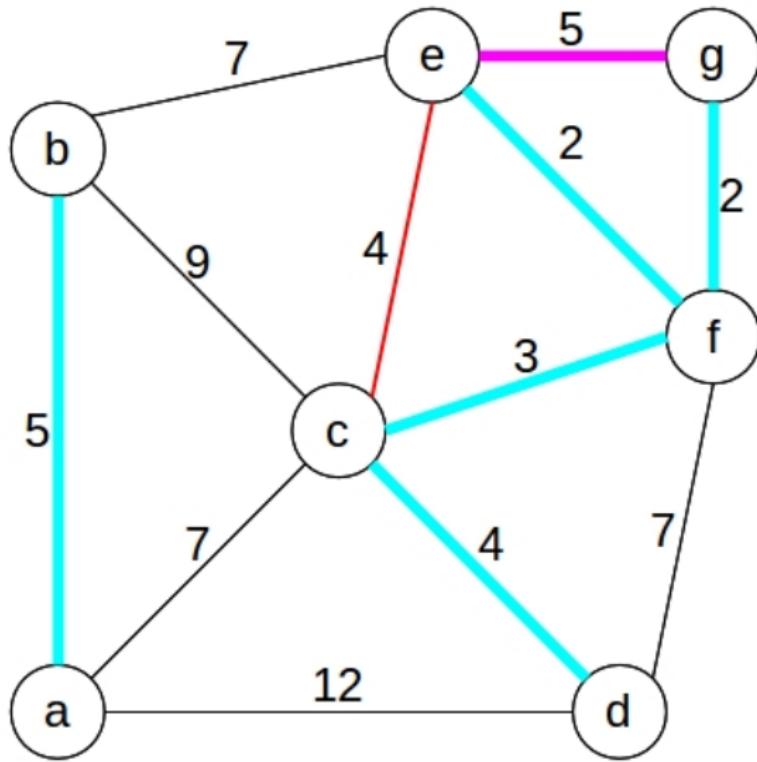
## Example



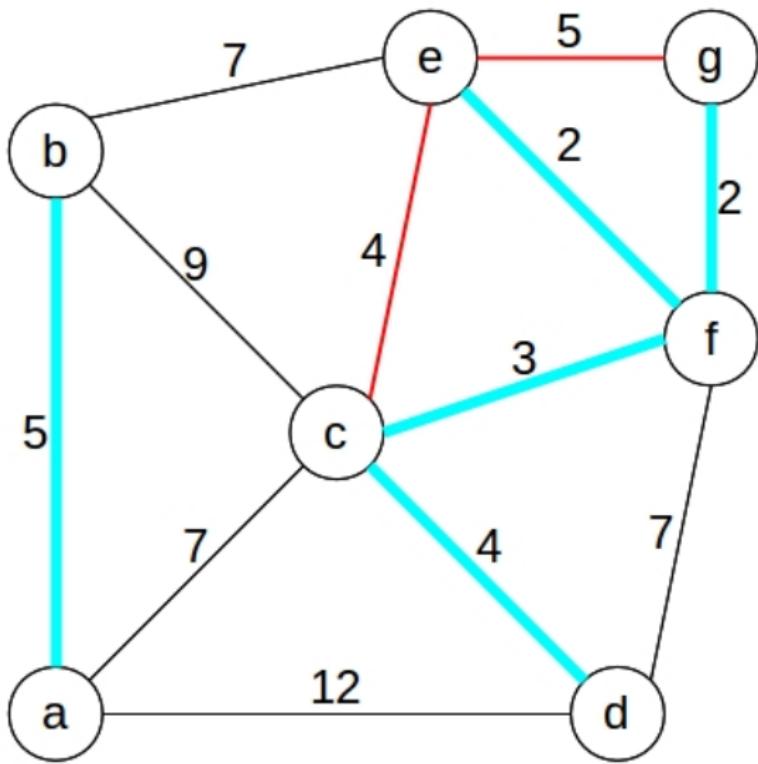
## Example



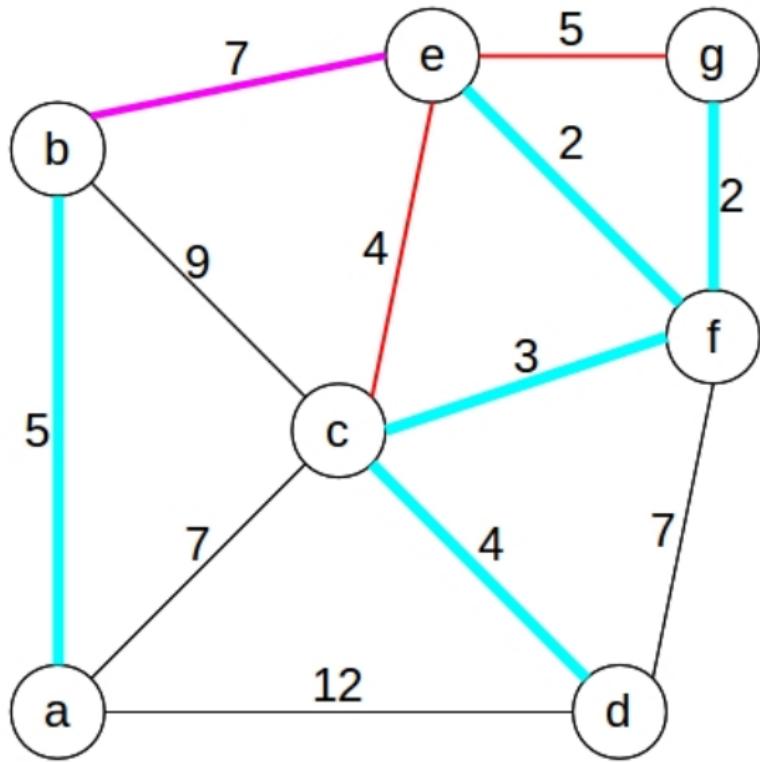
## Example



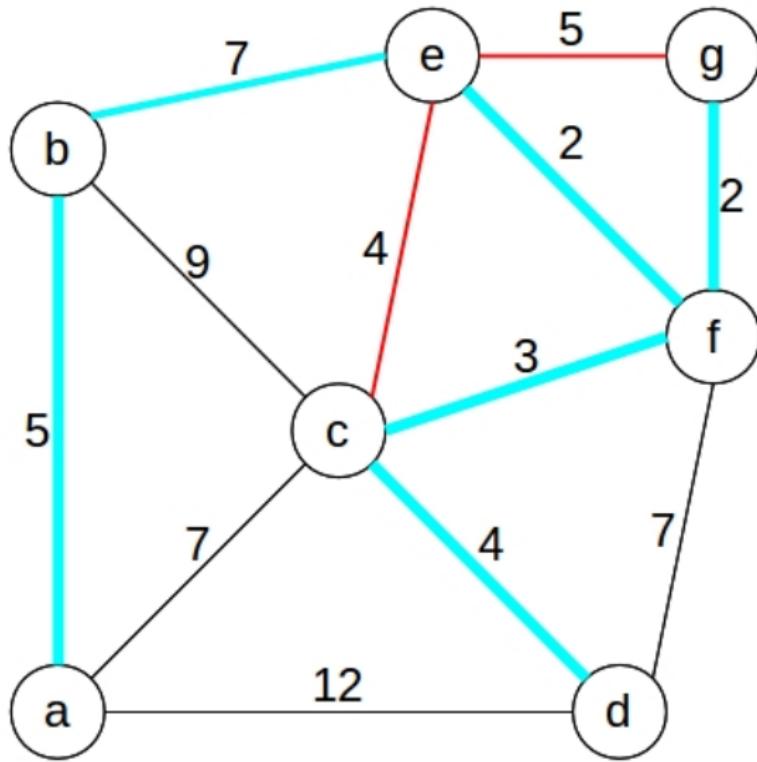
## Example



## Example



## Example



# Kruskal's Algorithm

Kruskal's algorithm is also greedy. It repeatedly adds the smallest edge to the spanning tree that does not create a cycle.

---

**Algorithm:** `kruskals( $G$ )`

---

- 1 Sort the  $m$  edges
  - 2 **for** each edge in order **do**
  - 3     **if** the edge creates a cycle in the forest we have build thus far **then**
  - 4         Discard
  - 5     **else**
  - 6         Add to forest
- 

## Proof of Correctness

# Kruskal's Algorithm

Kruskal's algorithm is also greedy. It repeatedly adds the smallest edge to the spanning tree that does not create a cycle.

---

**Algorithm:** `kruskals( $G$ )`

---

- 1 Sort the  $m$  edges
  - 2 **for** each edge in order **do**
  - 3     **if** the edge creates a cycle in the forest we have build thus far **then**
  - 4         Discard
  - 5     **else**
  - 6         Add to forest
- 

## Proof of Correctness

Again, use proof by contradiction. Similar to proof of Prims, so omit!

# How fast is Kruskal's algorithm?

# How fast is Kruskal's algorithm?

## Simple

---

**Algorithm:**  $\text{kruskals}(G)$

---

- 1 Sort the  $m$  edges
  - 2 **for** each edge in order **do**
  - 3     **if** the edge creates a cycle in the current forest
  - 4         **then**
  - 5             Discard
  - 6         **else**
  - 7             Add to forest
-

# How fast is Kruskal's algorithm?

## Simple

---

**Algorithm:** `kruskals( $G$ )`

---

- 1 Sort the  $m$  edges /\*  $O(m \lg m)$  \*/
  - 2 **for** each edge in order **do**
  - 3     **if** the edge creates a cycle in the current forest
  - 4         **then**
  - 5             Discard
  - 6         **else**
  - 7             Add to forest
-

# How fast is Kruskal's algorithm?

## Simple

---

**Algorithm:** `kruskals( $G$ )`

---

- 1 Sort the  $m$  edges /\*  $O(m \lg m)$  \*/  
/\*  $O(m)$  iterations \*/
  - 2 **for** each edge in order **do**
  - 3     **if** the edge creates a cycle in the current forest
  - 4         **then**
  - 5             Discard
  - 6         **else**
  - 7             Add to forest
-

# How fast is Kruskal's algorithm?

## Simple

---

**Algorithm:** `kruskals( $G$ )`

---

```
1 Sort the  $m$  edges /*  $O(m \lg m)$  */  
/*  $O(m)$  iterations */  
2 for each edge in order do  
3   if the edge creates a cycle in the current forest  
4     then  
5       Discard  
6     else  
7       Add to forest
```

---

- How do we detect cycles?

# How fast is Kruskal's algorithm?

## Simple

---

**Algorithm:** `kruskals( $G$ )`

---

```
1 Sort the  $m$  edges /*  $O(m \lg m)$  */  
/*  $O(m)$  iterations */  
2 for each edge in order do  
3   if the edge creates a cycle in the current forest /*  $O(n)$  */  
4     then  
5       Discard  
6     else  
7       Add to forest
```

---

- How do we detect cycles? BFS/DFS

# How fast is Kruskal's algorithm?

## Simple

---

**Algorithm:** `kruskals( $G$ )`

---

```
1 Sort the  $m$  edges /*  $O(m \lg m)$  */  
/*  $O(m)$  iterations */  
2 for each edge in order do  
3   if the edge creates a cycle in the current forest /*  $O(n)$  */  
4     then  
5       Discard /*  $O(1)$  */  
6     else  
7       Add to forest /*  $O(1)$  */
```

---

- How do we detect cycles? BFS/DFS

# How fast is Kruskal's algorithm?

## Simple

---

**Algorithm:** `kruskals( $G$ )`

---

```
1 Sort the  $m$  edges /*  $O(m \lg m)$  */  
/*  $O(m)$  iterations */  
2 for each edge in order do  
3   if the edge creates a cycle in the current forest /*  $O(n)$  */  
4     then  
5       Discard /*  $O(1)$  */  
6     else  
7       Add to forest /*  $O(1)$  */
```

---

- How do we detect cycles? BFS/DFS
- Total Time is therefore,  $T(n) = O(m \lg m + nm)$

# How fast is Kruskal's algorithm?

## Simple

---

**Algorithm:** `kruskals( $G$ )`

---

```
1 Sort the  $m$  edges /*  $O(m \lg m)$  */  
/*  $O(m)$  iterations */  
2 for each edge in order do  
3   if the edge creates a cycle in the current forest /*  $O(n)$  */  
4     then  
5       Discard /*  $O(1)$  */  
6     else  
7       Add to forest /*  $O(1)$  */
```

---

- How do we detect cycles? BFS/DFS
- Total Time is therefore,  $T(n) = O(m \lg m + nm) = O(nm)$

# How fast is Kruskal's algorithm?

## Smarter

What if we had the following two functions for component testing?

- $\text{sameComponent}(v, w)$  = Do  $v$  and  $w$  lie in the same connected component of the graph?
- $\text{mergeComponent}(C_1, C_2)$  = Merge connected components  $C_1$  and  $C_2$  into one component.

# How fast is Kruskal's algorithm?

## Smarter

What if we had the following two functions for component testing?

- $\text{sameComponent}(v, w)$  = Do  $v$  and  $w$  lie in the same connected component of the graph?
- $\text{mergeComponent}(C_1, C_2)$  = Merge connected components  $C_1$  and  $C_2$  into one component.

---

**Algorithm:** `kruskalMST( $G$ )`

---

```
1 Put the edges in a min-heap
2  $count = 0$ 

3 while  $count < n - 1$  do
4   | get min-weight edge  $(v, w)$  from heap
5   | if  $\text{!sameComponent}(v, w)$  then
6   |   | add to MST
7   |   |  $count = count + 1$ 
8   |   |  $\text{mergeComponent}(\text{component}(v), \text{component}(w))$ 
```

---

# How fast is Kruskal's algorithm?

## Smarter

What if we had the following two functions for component testing?

- $\text{sameComponent}(v, w) =$  Do  $v$  and  $w$  lie in the same connected component of the graph?
- $\text{mergeComponent}(C_1, C_2) =$  Merge connected components  $C_1$  and  $C_2$  into one component.

---

**Algorithm:** `kruskalMST( $G$ )`

---

```
1 Put the edges in a min-heap /*  $O(m)$  */  
2  $count = 0$   
  
3 while  $count < n - 1$  do  
4   | get min-weight edge  $(v, w)$  from heap  
5   | if  $\text{!sameComponent}(v, w)$  then  
6   |   | add to MST  
7   |   |  $count = count + 1$   
8   |   |  $\text{mergeComponent}(\text{component}(v), \text{component}(w))$ 
```

---

# How fast is Kruskal's algorithm?

## Smarter

What if we had the following two functions for component testing?

- $\text{sameComponent}(v, w) =$  Do  $v$  and  $w$  lie in the same connected component of the graph?
- $\text{mergeComponent}(C_1, C_2) =$  Merge connected components  $C_1$  and  $C_2$  into one component.

---

**Algorithm:** `kruskalMST( $G$ )`

---

```
1 Put the edges in a min-heap /*  $O(m)$  */  
2  $count = 0$  /*  $O(1)$  */  
  
3 while  $count < n - 1$  do  
4   | get min-weight edge  $(v, w)$  from heap  
5   | if  $\text{!sameComponent}(v, w)$  then  
6   |   | add to MST  
7   |   |  $count = count + 1$   
8   |   |  $\text{mergeComponent}(\text{component}(v), \text{component}(w))$ 
```

---

# How fast is Kruskal's algorithm?

## Smarter

What if we had the following two functions for component testing?

- $\text{sameComponent}(v, w)$  = Do  $v$  and  $w$  lie in the same connected component of the graph?
- $\text{mergeComponent}(C_1, C_2)$  = Merge connected components  $C_1$  and  $C_2$  into one component.

---

**Algorithm:** `kruskalMST( $G$ )`

---

```
1 Put the edges in a min-heap /*  $O(m)$  */
2  $count = 0$  /*  $O(1)$  */
   /*  $O(m)$  iterations */
3 while  $count < n - 1$  do
4   get min-weight edge  $(v, w)$  from heap
5   if  $\text{!sameComponent}(v, w)$  then
6     add to MST
7      $count = count + 1$ 
8      $\text{mergeComponent}(\text{component}(v), \text{component}(w))$ 
```

---

# How fast is Kruskal's algorithm?

## Smarter

What if we had the following two functions for component testing?

- $\text{sameComponent}(v, w)$  = Do  $v$  and  $w$  lie in the same connected component of the graph?
- $\text{mergeComponent}(C_1, C_2)$  = Merge connected components  $C_1$  and  $C_2$  into one component.

---

**Algorithm:**  $\text{kruskalMST}(G)$

---

```
1 Put the edges in a min-heap /* O(m) */
2 count = 0 /* O(1) */
   /* O(m) iterations */
3 while count < n - 1 do
4   get min-weight edge (v, w) from heap /* O(lg m) */
5   if !sameComponent(v, w) then
6     add to MST
7     count = count + 1
8     mergeComponent(component(v), component(w))
```

---

# How fast is Kruskal's algorithm?

## Smarter

What if we had the following two functions for component testing?

- $\text{sameComponent}(v, w)$  = Do  $v$  and  $w$  lie in the same connected component of the graph?
- $\text{mergeComponent}(C_1, C_2)$  = Merge connected components  $C_1$  and  $C_2$  into one component.

---

**Algorithm:**  $\text{kruskalMST}(G)$

---

```
1 Put the edges in a min-heap /* O(m) */
2 count = 0 /* O(1) */
   /* O(m) iterations */
3 while count < n - 1 do
4   get min-weight edge (v, w) from heap /* O(lg m) */
5   if !sameComponent(v, w) then
6     add to MST /* O(1) */
7     count = count + 1
8     mergeComponent(component(v), component(w))
```

---

# How fast is Kruskal's algorithm?

## Smarter

What if we had the following two functions for component testing?

- $\text{sameComponent}(v, w)$  = Do  $v$  and  $w$  lie in the same connected component of the graph?
- $\text{mergeComponent}(C_1, C_2)$  = Merge connected components  $C_1$  and  $C_2$  into one component.

---

**Algorithm:**  $\text{kruskalMST}(G)$

---

```
1 Put the edges in a min-heap /* O(m) */
2 count = 0 /* O(1) */
   /* O(m) iterations */
3 while count < n - 1 do
4   get min-weight edge (v, w) from heap /* O(lg m) */
5   if !sameComponent(v, w) then
6     add to MST /* O(1) */
7     count = count + 1 /* O(1) */
8     mergeComponent(component(v), component(w))
```

---

# How fast is Kruskal's algorithm?

## Smarter

What if we had the following two functions for component testing?

- $\text{sameComponent}(v, w) = \text{Do } v \text{ and } w \text{ lie in the same connected component of the graph?}$
- $\text{mergeComponent}(C_1, C_2) = \text{Merge connected components } C_1 \text{ and } C_2 \text{ into one component.}$

---

**Algorithm:**  $\text{kruskalMST}(G)$

---

```
1 Put the edges in a min-heap /* O(m) */
2 count = 0 /* O(1) */
   /* O(m) iterations */
3 while count < n - 1 do
4   get min-weight edge (v, w) from heap /* O(lg m) */
5   if !sameComponent(v, w) then
6     add to MST /* O(1) */
7     count = count + 1 /* O(1) */
8     mergeComponent(component(v), component(w))
```

---

How do we implement these functions?

# How fast is Kruskal's algorithm?

## Smarter

What if we had the following two functions for component testing?

- $\text{sameComponent}(v, w) = \text{Do } v \text{ and } w \text{ lie in the same connected component of the graph?}$
- $\text{mergeComponent}(C_1, C_2) = \text{Merge connected components } C_1 \text{ and } C_2 \text{ into one component.}$

---

**Algorithm:**  $\text{kruskalMST}(G)$

---

```
1 Put the edges in a min-heap /* O(m) */
2 count = 0 /* O(1) */
   /* O(m) iterations */
3 while count < n - 1 do
4   get min-weight edge (v, w) from heap /* O(lg m) */
5   if !sameComponent(v, w) then
6     add to MST /* O(1) */
7     count = count + 1 /* O(1) */
8     mergeComponent(component(v), component(w))
```

---

How do we implement these functions?

- BFS/DFS?

# How fast is Kruskal's algorithm?

## Smarter

What if we had the following two functions for component testing?

- $\text{sameComponent}(v, w)$  = Do  $v$  and  $w$  lie in the same connected component of the graph?
- $\text{mergeComponent}(C_1, C_2)$  = Merge connected components  $C_1$  and  $C_2$  into one component.

---

**Algorithm:**  $\text{kruskalMST}(G)$ 

---

```
1 Put the edges in a min-heap /* O(m) */  
2 count = 0 /* O(1) */  
/* O(m) iterations */  
3 while count < n - 1 do  
4     get min-weight edge (v, w) from heap /* O(lg m) */  
5     if !sameComponent(v, w) /* O(n) */  
        then  
7         add to MST /* O(1) */  
8         count = count + 1 /* O(1) */  
9         mergeComponent(component(v), component(w))
```

---

How do we implement these functions?

- BFS/DFS?

# How fast is Kruskal's algorithm?

## Smarter

What if we had the following two functions for component testing?

- $\text{sameComponent}(v, w)$  = Do  $v$  and  $w$  lie in the same connected component of the graph?
- $\text{mergeComponent}(C_1, C_2)$  = Merge connected components  $C_1$  and  $C_2$  into one component.

---

**Algorithm:**  $\text{kruskalMST}(G)$ 

---

```
1 Put the edges in a min-heap /* O(m) */
2 count = 0 /* O(1) */
   /* O(m) iterations */
3 while count < n - 1 do
4   get min-weight edge (v, w) from heap /* O(lg m) */
5   if !sameComponent(v, w) /* O(n) */
6     then
7       add to MST /* O(1) */
8       count = count + 1 /* O(1) */
9       mergeComponent(component(v), component(w)) /* O(1) */
```

---

How do we implement these functions?

- BFS/DFS?

# How fast is Kruskal's algorithm?

## Smarter

What if we had the following two functions for component testing?

- $\text{sameComponent}(v, w)$  = Do  $v$  and  $w$  lie in the same connected component of the graph?
- $\text{mergeComponent}(C_1, C_2)$  = Merge connected components  $C_1$  and  $C_2$  into one component.

---

**Algorithm:**  $\text{kruskalMST}(G)$

---

```
1 Put the edges in a min-heap /*  $O(m)$  */  
2  $count = 0$  /*  $O(1)$  */  
   /*  $O(m)$  iterations */  
3 while  $count < n - 1$  do  
4   get min-weight edge  $(v, w)$  from heap /*  $O(\lg m)$  */  
5   if  $\text{!sameComponent}(v, w)$  /*  $O(n)$  */  
6     then  
7       add to MST /*  $O(1)$  */  
8        $count = count + 1$  /*  $O(1)$  */  
9        $\text{mergeComponent}(\text{component}(v), \text{component}(w))$  /*  $O(1)$  */
```

---

How do we implement these functions?

- BFS/DFS?  $T(n) = O(m + m(\lg m + n)) = O(mn)$

# How fast is Kruskal's algorithm?

## Smarter

What if we had the following two functions for component testing?

- $\text{sameComponent}(v, w) = \text{Do } v \text{ and } w \text{ lie in the same connected component of the graph?}$
- $\text{mergeComponent}(C_1, C_2) = \text{Merge connected components } C_1 \text{ and } C_2 \text{ into one component.}$

---

**Algorithm:** `kruskalMST( $G$ )`

---

```
1 Put the edges in a min-heap /*  $O(m)$  */
2 count = 0 /*  $O(1)$  */
   /*  $O(m)$  iterations */
3 while count <  $n - 1$  do
4   get min-weight edge  $(v, w)$  from heap /*  $O(\lg m)$  */
5   if ! $\text{sameComponent}(v, w)$  then
6     add to MST /*  $O(1)$  */
7     count = count + 1 /*  $O(1)$  */
8      $\text{mergeComponent}(\text{component}(v), \text{component}(w))$ 
```

---

How do we implement these functions?

- BFS/DFS?  $T(n) = O(m + m(\lg m + n)) = O(mn)$
- What if both functions take  $O(\lg n)$  time?

# How fast is Kruskal's algorithm?

## Smarter

What if we had the following two functions for component testing?

- $\text{sameComponent}(v, w)$  = Do  $v$  and  $w$  lie in the same connected component of the graph?
- $\text{mergeComponent}(C_1, C_2)$  = Merge connected components  $C_1$  and  $C_2$  into one component.

---

**Algorithm:**  $\text{kruskalMST}(G)$

---

```
1 Put the edges in a min-heap /*  $O(m)$  */  
2  $count = 0$  /*  $O(1)$  */  
   /*  $O(m)$  iterations */  
3 while  $count < n - 1$  do  
4   get min-weight edge  $(v, w)$  from heap /*  $O(\lg m)$  */  
5   if  $\text{!sameComponent}(v, w)$  /*  $O(\lg n)$  */  
6     then  
7       add to MST /*  $O(1)$  */  
8        $count = count + 1$  /*  $O(1)$  */  
9        $\text{mergeComponent}(\text{component}(v), \text{component}(w))$  /*  $O(\lg n)$  */
```

---

How do we implement these functions?

- BFS/DFS?  $T(n) = O(m + m(\lg m + n)) = O(mn)$
- What if both functions take  $O(\lg n)$  time?

# How fast is Kruskal's algorithm?

## Smarter

What if we had the following two functions for component testing?

- $\text{sameComponent}(v, w)$  = Do  $v$  and  $w$  lie in the same connected component of the graph?
- $\text{mergeComponent}(C_1, C_2)$  = Merge connected components  $C_1$  and  $C_2$  into one component.

---

**Algorithm:**  $\text{kruskalMST}(G)$

---

```
1 Put the edges in a min-heap /*  $O(m)$  */  
2  $count = 0$  /*  $O(1)$  */  
   /*  $O(m)$  iterations */  
3 while  $count < n - 1$  do  
4   get min-weight edge  $(v, w)$  from heap /*  $O(\lg m)$  */  
5   if  $\text{!sameComponent}(v, w)$  /*  $O(\lg n)$  */  
6     then  
7       add to MST /*  $O(1)$  */  
8        $count = count + 1$  /*  $O(1)$  */  
9        $\text{mergeComponent}(\text{component}(v), \text{component}(w))$  /*  $O(\lg n)$  */
```

---

How do we implement these functions?

- BFS/DFS?  $T(n) = O(m + m(\lg m + n)) = O(mn)$
- What if both functions take  $O(\lg n)$  time?  $T(n) = O(m + m(\lg m + \lg n)) = O(m \lg m)$

# Another Strategy for Testing for Cycles

- Kruskal's algorithm builds up connected components.

## Another Strategy for Testing for Cycles

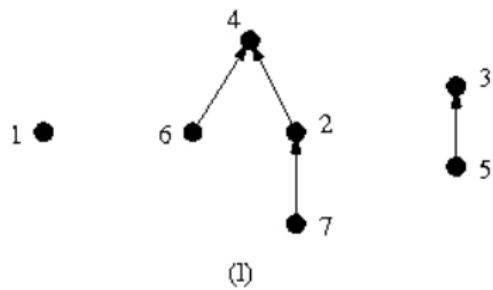
- Kruskal's algorithm builds up connected components.
- If the two end-vertices of an edge that is being considered are in the same connected component, then it creates a cycle.

## Another Strategy for Testing for Cycles

- Kruskal's algorithm builds up connected components.
- If the two end-vertices of an edge that is being considered are in the same connected component, then it creates a cycle.
- Thus if we can maintain which vertices are in which component fast, we do not have to test for cycles!

# The Union-Find Data Structure

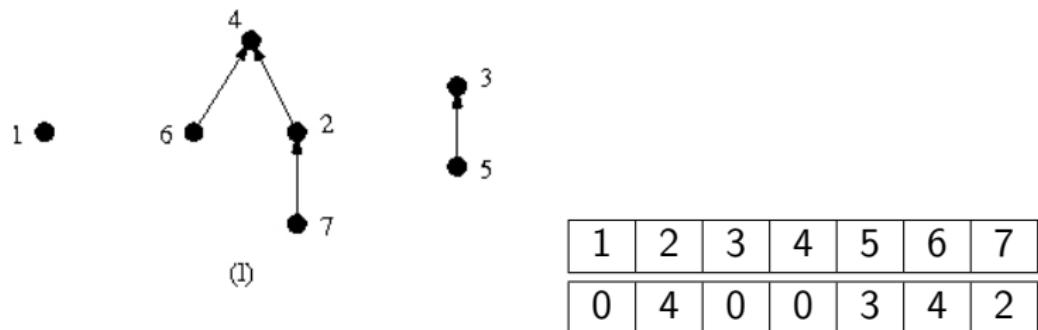
Represent each set as a tree with pointers from a node to its parent. Each element is contained in a node, and the *name* of the set is the key at the root



1	2	3	4	5	6	7
0	4	0	0	3	4	2

# The Union-Find Data Structure

Represent each set as a tree with pointers from a node to its parent. Each element is contained in a node, and the *name* of the set is the key at the root



## Union and Find Operations

- $\text{find}(i)$  = Returns the label of the root of the tree containing element  $i$ .
- $\text{union}(i,j)$  = Make the root of one of the trees the parent of the root of the other tree.

# Analysis of Union-Find Trees

- What is the worst-case time to execute a find or a union?

# Analysis of Union-Find Trees

- What is the worst-case time to execute a find or a union? The height of the union find tree.

# Analysis of Union-Find Trees

- What is the worst-case time to execute a `find` or a `union`? The height of the union find tree.
- What is the worst-case height of a union-find tree?

# Analysis of Union-Find Trees

- What is the worst-case time to execute a find or a union? The height of the union find tree.
- What is the worst-case height of a union-find tree?  $O(n)$

# Analysis of Union-Find Trees

- What is the worst-case time to execute a find or a union? The height of the union find tree.
- What is the worst-case height of a union-find tree?  $O(n)$
- When we union two trees, which tree is the parent?

# Analysis of Union-Find Trees

- What is the worst-case time to execute a find or a union? The height of the union find tree.
- What is the worst-case height of a union-find tree?  $O(n)$
- When we union two trees, which tree is the parent? The tree with the larger height than the other one.

# Analysis of Union-Find Trees

- What is the worst-case time to execute a find or a union? The height of the union find tree.
- What is the worst-case height of a union-find tree?  $O(n)$
- When we union two trees, which tree is the parent? The tree with the larger height than the other one.  
Why?

# Analysis of Union-Find Trees

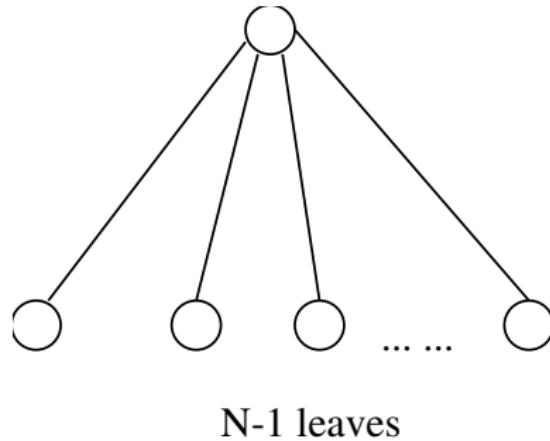
- What is the worst-case time to execute a find or a union? The height of the union find tree.
- What is the worst-case height of a union-find tree?  $O(n)$
- When we union two trees, which tree is the parent? The tree with the larger height than the other one.

Why? The height of the final tree will increase only if both subtrees are of equal height. This means that each operation takes  $O(\lg n)$  time!

# Can we do better?

# Can we do better?

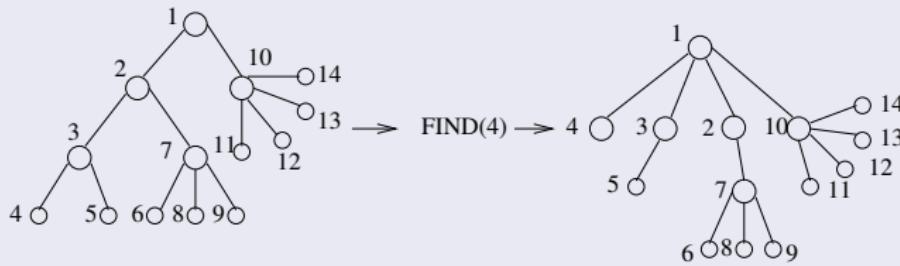
The ideal *Union-Find* tree has depth 1:



# Can we do better?

## Path Compression

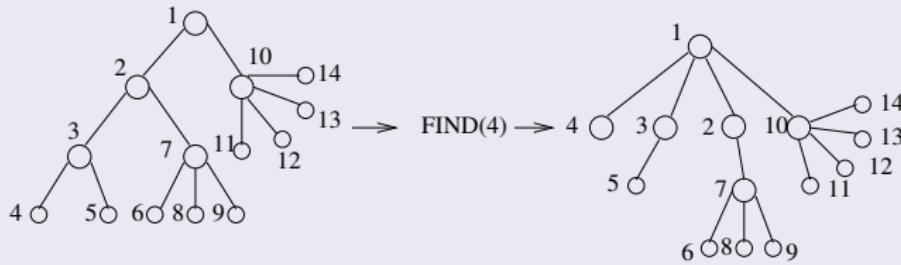
On a find, if we are going to walk along a path anyway, why not change the pointers to point to the root?



# Can we do better?

## Path Compression

On a find, if we are going to walk along a path anyway, why not change the pointers to point to the root?



Difficult analysis shows that it takes  $O(n\alpha(n))$  time for a sequence of  $n$  union-finds, where  $\alpha(n)$  is the **inverse Ackerman function**

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
$i$	1					
	2					
	3					
	:					

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
i	1					
	2					
	3					
	:					

$i = 1, j = 1$

$A(1, 1) =$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
i	1					
	2					
	3					
	:					

$$i = 1, j = 1$$

$$A(1, 1) = 2^1 = 2$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
i	1	$2^1 = 2$				
	2					
	3					
	:					

$$i = 1, j = 1$$

$$A(1, 1) = 2^1 = 2$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
i	1	$2^1 = 2$				
	2					
	3					
	:					

$i = 1, j = 2$

$A(1, 2) =$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
i	1	$2^1 = 2$				
	2					
	3					
	:					

$$i = 1, j = 2$$

$$A(1, 2) = 2^2 = 4$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		j				
		1	2	3	4	...
i	1	$2^1 = 2$	$2^2 = 4$			
	2					
	3					
	:					

$$i = 1, j = 2$$

$$A(1, 2) = 2^2 = 4$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		j				
		1	2	3	4	...
i	1	$2^1 = 2$	$2^2 = 4$			
	2					
	3					
	:					

$i = 1, j = 3$

$A(1, 3) =$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		j				
		1	2	3	4	...
i	1	$2^1 = 2$	$2^2 = 4$			
	2					
	3					
	:					

$$i = 1, j = 3$$

$$A(1, 3) = 2^3 = 8$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$		
	2					
	3					
	:					

$$i = 1, j = 3$$

$$A(1, 3) = 2^3 = 8$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$		
	2					
	3					
	:					

$i = 1, j = 4$

$A(1, 4) =$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$		
	2					
	3					
	:					

$$i = 1, j = 4$$

$$A(1, 4) = 2^4 = 16$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
	2				
	3				
	:				

$$i = 1, j = 4$$

$$A(1, 4) = 2^4 = 16$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
	2				
	3				
	:				

$i = 2, j = 1$

$A(2, 1) =$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
	2				
	3				
	:				

$i = 2, j = 1$

$$A(2, 1) = A(2 - 1, 2)$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
	2				
	3				
	:				

$i = 2, j = 1$

$$A(2, 1) = A(2 - 1, 2) = A(1, 2)$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
	2				
	3				
	:				

$$i = 2, j = 1$$

$$A(2, 1) = A(2 - 1, 2) = A(1, 2) = 2^2 = 4$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
1		$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
$i$	2	$2^2 = 4$			
	3				
	:				

$i = 2, j = 1$

$$A(2, 1) = A(2 - 1, 2) = A(1, 2) = 2^2 = 4$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
1		$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
$i$	2	$2^2 = 4$			
	3				
	:				

$i = 2, j = 2$

$A(2, 2) =$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
1		$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
$i$	2	$2^2 = 4$			
	3				
	:				

$i = 2, j = 2$

$$A(2, 2) = A(2 - 1, A(2, 2 - 1))$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
1		$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
$i$	2	$2^2 = 4$			
	3				
	:				

$i = 2, j = 2$

$$A(2, 2) = A(2 - 1, A(2, 2 - 1)) = A(1, A(2, 1))$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
1		$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
$i$	2	$2^2 = 4$			
	3				
	:				

$i = 2, j = 2$

$$A(2, 2) = A(2 - 1, A(2, 2 - 1)) = A(1, A(2, 1)) = A(1, 2^2) = 2^{2^2} = 16$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
1		$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
i	2	$2^2 = 4$	$2^{2^2} = 16$		
	3				
	:				

$i = 2, j = 2$

$$A(2, 2) = A(2 - 1, A(2, 2 - 1)) = A(1, A(2, 1)) = A(1, 2^2) = 2^{2^2} = 16$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
1		$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
i	2	$2^2 = 4$	$2^{2^2} = 16$		
	3				
	:				

$i = 2, j = 3$

$A(2, 3) =$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
1		$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
i	2	$2^2 = 4$	$2^{2^2} = 16$		
	3				
	:				

$i = 2, j = 3$

$$A(2, 3) = A(2 - 1, A(2, 3 - 1))$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
1		$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
$i$	2	$2^2 = 4$	$2^{2^2} = 16$		
	3				
	:				

$i = 2, j = 3$

$$A(2, 3) = A(2 - 1, A(2, 3 - 1)) = A(1, A(2, 2))$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
1		$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
i	2	$2^2 = 4$	$2^{2^2} = 16$		
	3				
	:				

$i = 2, j = 3$

$$A(2, 3) = A(2 - 1, A(2, 3 - 1)) = A(1, A(2, 2)) = A(1, 2^{2^2})$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
1		$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
i	2	$2^2 = 4$	$2^{2^2} = 16$		
	3				
	:				

$i = 2, j = 3$

$$A(2, 3) = A(2 - 1, A(2, 3 - 1)) = A(1, A(2, 2)) = A(1, 2^{2^2}) = 2^{2^{2^2}} = 65536$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	
	3				
	$\vdots$				

$i = 2, j = 3$

$$A(2, 3) = A(2 - 1, A(2, 3 - 1)) = A(1, A(2, 2)) = A(1, 2^{2^2}) = 2^{2^{2^2}} = 65536$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	
	3				
	$\vdots$				

$i = 2, j = 4$

$$A(2, 4) =$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	
	3				
	$\vdots$				

$i = 2, j = 4$

$$A(2, 4) = A(2 - 1, A(2, 4 - 1))$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	
	3				
	$\vdots$				

$i = 2, j = 4$

$$A(2, 4) = A(2 - 1, A(2, 4 - 1)) = A(1, A(2, 3))$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	
	3				
	$\vdots$				

$i = 2, j = 4$

$$A(2, 4) = A(2-1, A(2, 4-1)) = A(1, A(2, 3)) = A(1, 2^{2^{2^2}})$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	
	3				
	$\vdots$				

$i = 2, j = 4$

$$A(2, 4) = A(2-1, A(2, 4-1)) = A(1, A(2, 3)) = A(1, 2^{2^{2^2}}) = 2^{2^{2^{2^2}}} = 2^{65536}$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$
	3				
	$\vdots$				

$i = 2, j = 4$

$$A(2, 4) = A(2-1, A(2, 4-1)) = A(1, A(2, 3)) = A(1, 2^{2^{2^2}}) = 2^{2^{2^{2^2}}} = 2^{65536}$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3					
	:					

$i = 3, j = 1$

$A(3, 1) =$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3					
	:					

$i = 3, j = 1$

$$A(3, 1) = A(3 - 1, 2)$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3					
	:					

$i = 3, j = 1$

$$A(3, 1) = A(3 - 1, 2) = A(2, 2)$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		j				
		1	2	3	4	...
i	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3					
	:					

$i = 3, j = 1$

$$A(3, 1) = A(3 - 1, 2) = A(2, 2) = 2^{2^2} = 16$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3	$2^{2^2} = 16$				
	$\vdots$					

$i = 3, j = 1$

$$A(3, 1) = A(3 - 1, 2) = A(2, 2) = 2^{2^2} = 16$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3	$2^{2^2} = 16$				
	$\vdots$					

$i = 3, j = 2$

$$A(3, 2) =$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3	$2^{2^2} = 16$				
	$\vdots$					

$i = 3, j = 2$

$$A(3, 2) = A(3 - 1, A(3, 2 - 1))$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3	$2^{2^2} = 16$				
	$\vdots$					

$i = 3, j = 2$

$$A(3, 2) = A(3 - 1, A(3, 2 - 1)) = A(2, A(3, 1))$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3	$2^{2^2} = 16$				
	$\vdots$					

$i = 3, j = 2$

$$A(3, 2) = A(3 - 1, A(3, 2 - 1)) = A(2, A(3, 1)) = A(2, 2^{2^2})$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3	$2^{2^2} = 16$				
	$\vdots$					

$i = 3, j = 2$

$$A(3, 2) = A(3 - 1, A(3, 2 - 1)) = A(2, A(3, 1)) = A(2, 2^{2^2}) = A(2, 16)$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3	$2^{2^2} = 16$				
	$\vdots$					

$i = 3, j = 2$

$$A(3, 2) = A(3 - 1, A(3, 2 - 1)) = A(2, A(3, 1)) = A(2, 2^{2^2}) = A(2, 16) = 2^{2^{2^{2^2}}} \Big\}^{16}$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$
	3	$2^{2^2} = 16$	$2^{2^{2^{2^2}}} \}^{16}$		
	$\vdots$				

$i = 3, j = 2$

$$A(3, 2) = A(3 - 1, A(3, 2 - 1)) = A(2, A(3, 1)) = A(2, 2^{2^2}) = A(2, 16) = 2^{2^{2^{2^2}}} \}^{16}$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$
	3	$2^{2^2} = 16$	$2^{2^{2^{2^2}}} \}^{16}$		
	$\vdots$				

$i = 3, j = 3$

$A(3, 3) =$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$
	3	$2^{2^2} = 16$	$2^{2^{2^{2^2}}} \}^{16}$		
	$\vdots$				

$i = 3, j = 3$

$$A(3, 3) = A(3 - 1, A(3, 3 - 1))$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$			
		1	2	3	4
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$
	3	$2^{2^2} = 16$	$2^{2^{2^{2^2}}} \}^{16}$		
	$\vdots$				

$i = 3, j = 3$

$$A(3, 3) = A(3 - 1, A(3, 3 - 1)) = A(2, A(3, 2))$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
i	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3	$2^{2^2} = 16$	$2^{2^{2^{...2}}} \brace 16$			
	$\vdots$					

$i = 3, j = 3$

$$A(3, 3) = A(3 - 1, A(3, 3 - 1)) = A(2, A(3, 2)) = A(2, 2^{2^{2^{...2}}} \brace 16)$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
i	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3	$2^{2^2} = 16$	$2^{2^{2^{...2}}} \brace 16$			
	:					

$i = 3, j = 3$

$$A(3, 3) = A(3 - 1, A(3, 3 - 1)) = A(2, A(3, 2)) = A(2, 2^{2^{2^{...2}}} \brace 16) = 2^{2^{2^{2^{2^{2^{...2}}} \brace 2^{2^{2^{...2}}} \brace 16}}$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
i	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3	$2^{2^2} = 16$	$2^{2^{2^2}} \brace{16}$	$2^{2^{2^{2^2}}} \brace{2^{2^2}} \brace{16}$		
	:					

$i = 3, j = 3$

$$A(3, 3) = A(3 - 1, A(3, 3 - 1)) = A(2, A(3, 2)) = A(2, 2^{2^{2^2}} \brace{16}) = 2^{2^{2^{2^2}} \brace{2^{2^2}} \brace{16}}$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3	$2^{2^2} = 16$	$2^{2^{2^2}} \brace{16}$	$2^{2^{2^{2^2}}} \brace{2^{2^2}} \brace{16}$		
	:					

$i = 3, j = 4$

$$A(3, 4) =$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
i	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3	$2^{2^2} = 16$	$2^{2^{2^2}} \brace{16}$	$2^{2^{2^{2^2}}} \brace{2^{2^2}} \brace{16}$		
	:					

$i = 3, j = 4$

$$A(3, 4) = A(3 - 1, A(3, 4 - 1))$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3	$2^{2^2} = 16$	$2^{2^{2^{2^2}}} \brace{16}$	$2^{2^{2^{2^{2^2}}}} \brace{2^{2^{2^2}}} \brace{16}$		
	$\vdots$					

$i = 3, j = 4$

$$A(3, 4) = A(3 - 1, A(3, 4 - 1)) = A(2, A(3, 3))$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
i	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3	$2^{2^2} = 16$	$2^{2^{2^2}} \brace{16}$	$2^{2^{2^{2^2}}} \brace{2^{2^2}} \brace{16}$		
	$\vdots$					

$i = 3, j = 4$

$$A(3, 4) = A(3 - 1, A(3, 4 - 1)) = A(2, A(3, 3)) = A(2, 2^{2^{2^2}} \brace{2^{2^2}} \brace{16})$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
i	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3	$2^{2^2} = 16$	$2^{2^{2^2}} \brace{16}$	$2^{2^{2^{2^2}}} \brace{2^{2^2}} \brace{16}$		
	$\vdots$					

$i = 3, j = 4$

$$A(3, 4) = A(3 - 1, A(3, 4 - 1)) = A(2, A(3, 3)) = A(2, 2^{2^{2^2}} \brace{2^{2^2}} \brace{16}) = 2^{2^{2^{2^2}} \brace{2^{2^2}} \brace{2^{2^2}} \brace{16}}$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
i	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3	$2^{2^2} = 16$	$2^{2^{2^2}} \brace{16}$	$2^{2^{2^{2^2}}} \brace{2^{2^{2^2}}} \brace{16}$	$2^{2^{2^{2^2}}} \brace{2^{2^{2^2}}} \brace{2^{2^{2^2}}} \brace{16}$	
	:					

$i = 3, j = 4$

$$A(3, 4) = A(3 - 1, A(3, 4 - 1)) = A(2, A(3, 3)) = A(2, 2^{2^{2^2}} \brace{2^{2^2}} \brace{16}) = 2^{2^{2^{2^2}} \brace{2^{2^2}} \brace{2^{2^2}} \brace{16}}$$

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
i	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3	$2^{2^2} = 16$	$2^{2^{2^2}} \brace{16}$	$2^{2^{2^{2^2}}} \brace{2^{2^{2^2}}} \brace{16}$	$2^{2^{2^{2^2}}} \brace{2^{2^{2^2}}} \brace{2^{2^{2^2}}} \brace{16}$	
	:					

What is  $\alpha(\text{number of atoms in the universe})$ ?

# Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

		$j$				
		1	2	3	4	...
$i$	1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	
	2	$2^2 = 4$	$2^{2^2} = 16$	$2^{2^{2^2}} = 65536$	$2^{2^{2^{2^2}}} = 2^{65536}$	
	3	$2^{2^2} = 16$	$2^{2^{2^2}} \brace{16}$	$2^{2^{2^{2^2}}} \brace{2^{2^2}} \brace{16}$	$2^{2^{2^{2^{2^2}}}} \brace{2^{2^{2^2}}} \brace{2^{2^2}} \brace{16}$	
	:					

What is  $\alpha(\text{number of atoms in the universe})$ ?

# Analysis of Kruskal's Algorithm using Union-Find

---

**Algorithm:** kruskalMST( $G$ )

---

```
1 Put the edges in a min-heap
2 count = 0

3 while count < n - 1 do
4   | get min-weight edge (v, w) from heap
5   | if !sameComponent(v, w)
6   | then
7     |   | add to MST
8     |   | count = count + 1
9     |   | mergeComponent(component(v), component(w))
```

---

$$T(n) =$$

# Analysis of Kruskal's Algorithm using Union-Find

---

**Algorithm:** kruskalMST( $G$ )

---

```
1 Put the edges in a min-heap /*  $O(m)$  */  
2  $count = 0$   
  
3 while  $count < n - 1$  do  
4   | get min-weight edge  $(v, w)$  from heap  
5   | if !sameComponent( $v, w$ )  
6   |   then  
7   |     | add to MST  
8   |     |  $count = count + 1$   
9   |     | mergeComponent(component( $v$ ), component( $w$ ))
```

---

$$T(n) =$$

# Analysis of Kruskal's Algorithm using Union-Find

---

**Algorithm:** kruskalMST( $G$ )

---

```
1 Put the edges in a min-heap /*  $O(m)$  */  
2 count = 0 /*  $O(1)$  */  
  
3 while count <  $n - 1$  do  
4   | get min-weight edge  $(v, w)$  from heap  
5   | if !sameComponent( $v, w$ )  
6   | then  
7     |   | add to MST  
8     |   | count = count + 1  
9     |   | mergeComponent(component( $v$ ), component( $w$ ))
```

---

$$T(n) =$$

# Analysis of Kruskal's Algorithm using Union-Find

---

**Algorithm:** kruskalMST( $G$ )

---

```
1 Put the edges in a min-heap /*  $O(m)$  */
2 count = 0 /*  $O(1)$  */
   /*  $O(m)$  iterations */
3 while count <  $n - 1$  do
4   | get min-weight edge  $(v, w)$  from heap
5   | if !sameComponent( $v, w$ )
6   | then
7   |   | add to MST
8   |   | count = count + 1
9   |   | mergeComponent(component( $v$ ), component( $w$ ))
```

---

$$T(n) =$$

# Analysis of Kruskal's Algorithm using Union-Find

---

**Algorithm:** kruskalMST( $G$ )

---

```
1 Put the edges in a min-heap /*  $O(m)$  */
2 count = 0 /*  $O(1)$  */
   /*  $O(m)$  iterations */
3 while count <  $n - 1$  do
4   | get min-weight edge  $(v, w)$  from heap /*  $O(\lg m)$  */
5   | if !sameComponent( $v, w$ )
6   | then
7     |   | add to MST
8     |   | count = count + 1
9     |   | mergeComponent(component( $v$ ), component( $w$ ))
```

---

$$T(n) =$$

# Analysis of Kruskal's Algorithm using Union-Find

---

**Algorithm:** kruskalMST( $G$ )

---

```
1 Put the edges in a min-heap /*  $O(m)$  */
2 count = 0 /*  $O(1)$  */
   /*  $O(m)$  iterations */
3 while count <  $n - 1$  do
4   | get min-weight edge  $(v, w)$  from heap /*  $O(\lg m)$  */
5   | if !sameComponent( $v, w$ ) /*  $O(\alpha(n))$  */
6   |   then
7   |     | add to MST
8   |     | count = count + 1
9   |     | mergeComponent(component( $v$ ), component( $w$ ))
```

---

$$T(n) =$$

# Analysis of Kruskal's Algorithm using Union-Find

---

**Algorithm:** kruskalMST( $G$ )

---

```
1 Put the edges in a min-heap /*  $O(m)$  */
2 count = 0 /*  $O(1)$  */
   /*  $O(m)$  iterations */
3 while count <  $n - 1$  do
4   | get min-weight edge  $(v, w)$  from heap /*  $O(\lg m)$  */
5   | if !sameComponent( $v, w$ ) /*  $O(\alpha(n))$  */
6   |   then
7   |     | add to MST /*  $O(1)$  */
8   |     | count = count + 1
9   |     | mergeComponent(component( $v$ ), component( $w$ ))
```

---

$$T(n) =$$

# Analysis of Kruskal's Algorithm using Union-Find

---

**Algorithm:** kruskalMST( $G$ )

---

```
1 Put the edges in a min-heap /*  $O(m)$  */
2 count = 0 /*  $O(1)$  */
   /*  $O(m)$  iterations */
3 while count <  $n - 1$  do
4   | get min-weight edge  $(v, w)$  from heap /*  $O(\lg m)$  */
5   | if !sameComponent( $v, w$ ) /*  $O(\alpha(n))$  */
6   |   then
7   |     | add to MST /*  $O(1)$  */
8   |     | count = count + 1 /*  $O(1)$  */
9   |     | mergeComponent(component( $v$ ), component( $w$ ))
```

---

$$T(n) =$$

# Analysis of Kruskal's Algorithm using Union-Find

---

**Algorithm:** kruskalMST( $G$ )

---

```
1 Put the edges in a min-heap /*  $O(m)$  */
2 count = 0 /*  $O(1)$  */
   /*  $O(m)$  iterations */
3 while count <  $n - 1$  do
4   | get min-weight edge  $(v, w)$  from heap /*  $O(\lg m)$  */
5   | if !sameComponent( $v, w$ ) /*  $O(\alpha(n))$  */
6   |   then
7   |     | add to MST /*  $O(1)$  */
8   |     | count = count + 1 /*  $O(1)$  */
9   |     | mergeComponent(component( $v$ ), component( $w$ )) /*  $O(\alpha(n))$  */
```

---

$$T(n) =$$

# Analysis of Kruskal's Algorithm using Union-Find

---

**Algorithm:** kruskalMST( $G$ )

---

```
1 Put the edges in a min-heap /*  $O(m)$  */
2 count = 0 /*  $O(1)$  */
   /*  $O(m)$  iterations */
3 while count <  $n - 1$  do
4   | get min-weight edge  $(v, w)$  from heap /*  $O(\lg m)$  */
5   | if !sameComponent( $v, w$ ) /*  $O(\alpha(n))$  */
6   |   then
7   |     | add to MST /*  $O(1)$  */
8   |     | count = count + 1 /*  $O(1)$  */
9   |     | mergeComponent(component( $v$ ), component( $w$ )) /*  $O(\alpha(n))$  */
```

---

$$T(n) = O(m + m(\lg m + \alpha(n)))$$

# Analysis of Kruskal's Algorithm using Union-Find

---

**Algorithm:** kruskalMST( $G$ )

---

```
1 Put the edges in a min-heap /*  $O(m)$  */
2 count = 0 /*  $O(1)$  */
   /*  $O(m)$  iterations */
3 while count <  $n - 1$  do
4   | get min-weight edge  $(v, w)$  from heap /*  $O(\lg m)$  */
5   | if !sameComponent( $v, w$ ) /*  $O(\alpha(n))$  */
6   |   then
7   |     | add to MST /*  $O(1)$  */
8   |     | count = count + 1 /*  $O(1)$  */
9   |     | mergeComponent(component( $v$ ), component( $w$ )) /*  $O(\alpha(n))$  */
```

---

$$\begin{aligned}T(n) &= O(m + m(\lg m + \alpha(n))) \\&= O(m \lg m)\end{aligned}$$

# Prim's Algorithm vs. Kruskal's Algorithm

Difference between the algorithms

# Prim's Algorithm vs. Kruskal's Algorithm

## Difference between the algorithms

- Prim's minimum spanning is growing outwards. You always have a single partially constructed tree.
- Kruskal's minimum spanning tree has a forest of smaller trees - it is not always connected.

# Prim's Algorithm vs. Kruskal's Algorithm

## Analysis

Algorithm	Worst-Case Time Complexity		
	All Graphs	Dense Graphs	Sparse Graphs
Prim's			
Kruskal's			

# Prim's Algorithm vs. Kruskal's Algorithm

## Analysis

Algorithm	Worst-Case Time Complexity		
	All Graphs	Dense Graphs	Sparse Graphs
Prim's	$O(m + n \lg n)$		
Kruskal's			

# Prim's Algorithm vs. Kruskal's Algorithm

## Analysis

Algorithm	Worst-Case Time Complexity		
	All Graphs	Dense Graphs	Sparse Graphs
Prim's	$O(m + n \lg n)$		
Kruskal's	$O(m \lg m)$		

# Prim's Algorithm vs. Kruskal's Algorithm

## Analysis

Algorithm	Worst-Case Time Complexity		
	All Graphs	Dense Graphs	Sparse Graphs
Prim's	$O(m + n \lg n)$	$O(n^2)$	
Kruskal's	$O(m \lg m)$		

# Prim's Algorithm vs. Kruskal's Algorithm

## Analysis

Algorithm	Worst-Case Time Complexity		
	All Graphs	Dense Graphs	Sparse Graphs
Prim's	$O(m + n \lg n)$	$O(n^2)$	
Kruskal's	$O(m \lg m)$	$O(n^2 \lg n^2)$	

# Prim's Algorithm vs. Kruskal's Algorithm

## Analysis

Algorithm	Worst-Case Time Complexity		
	All Graphs	Dense Graphs	Sparse Graphs
Prim's	$O(m + n \lg n)$	$O(n^2)$	
Kruskal's	$O(m \lg m)$	$O(n^2 \lg n^2)$	

# Prim's Algorithm vs. Kruskal's Algorithm

## Analysis

Algorithm	Worst-Case Time Complexity		
	All Graphs	Dense Graphs	Sparse Graphs
Prim's	$O(m + n \lg n)$	$O(n^2)$	$O(n \lg n)$
Kruskal's	$O(m \lg m)$	$O(n^2 \lg n^2)$	

# Prim's Algorithm vs. Kruskal's Algorithm

## Analysis

Algorithm	Worst-Case Time Complexity		
	All Graphs	Dense Graphs	Sparse Graphs
Prim's	$O(m + n \lg n)$	$O(n^2)$	$O(n \lg n)$
Kruskal's	$O(m \lg m)$	$O(n^2 \lg n^2)$	$O(n \lg n)$

# Prim's Algorithm vs. Kruskal's Algorithm

## Analysis

Algorithm	Worst-Case Time Complexity		
	All Graphs	Dense Graphs	Sparse Graphs
Prim's	$O(m + n \lg n)$	$O(n^2)$	$O(n \lg n)$
Kruskal's	$O(m \lg m)$	$O(n^2 \lg n^2)$	$O(n \lg n)$

# Shortest Paths

Finding the shortest path between two nodes in a graph arises in many different applications:

# Shortest Paths

Finding the shortest path between two nodes in a graph arises in many different applications:

- Transportation problems – finding the cheapest way to travel between two locations.

# Shortest Paths

Finding the shortest path between two nodes in a graph arises in many different applications:

- Transportation problems – finding the cheapest way to travel between two locations.
- Motion planning – what is the most natural way for a cartoon character to move about a simulated environment.

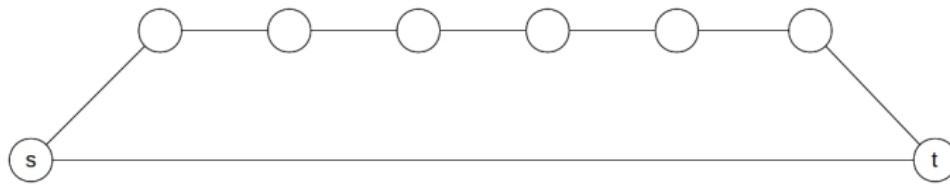
# Shortest Paths

Finding the shortest path between two nodes in a graph arises in many different applications:

- Transportation problems – finding the cheapest way to travel between two locations.
- Motion planning – what is the most natural way for a cartoon character to move about a simulated environment.
- Communications problems – how long will it take for a message to get from one place to another?

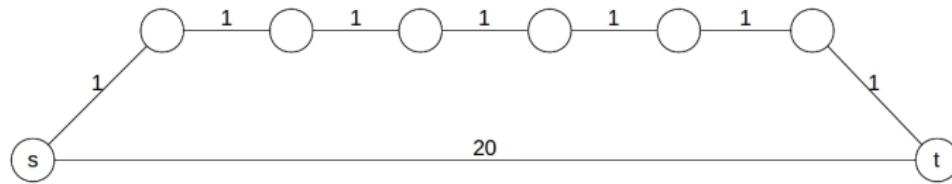
# Weighted vs. Unweighted Graphs

What is the cost of the shortest path from  $s$  to  $t$  in the unweighted graph below?



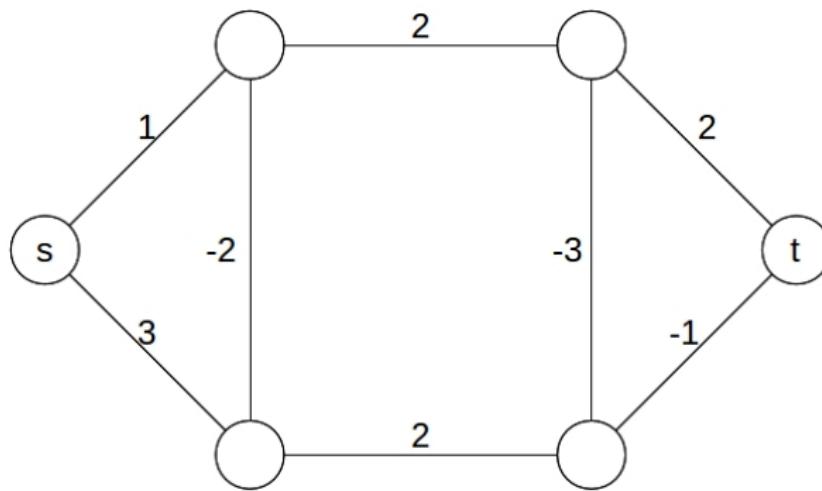
# Weighted vs. Unweighted Graphs

What is the cost of the shortest path from  $s$  to  $t$  in the weighted graph below?



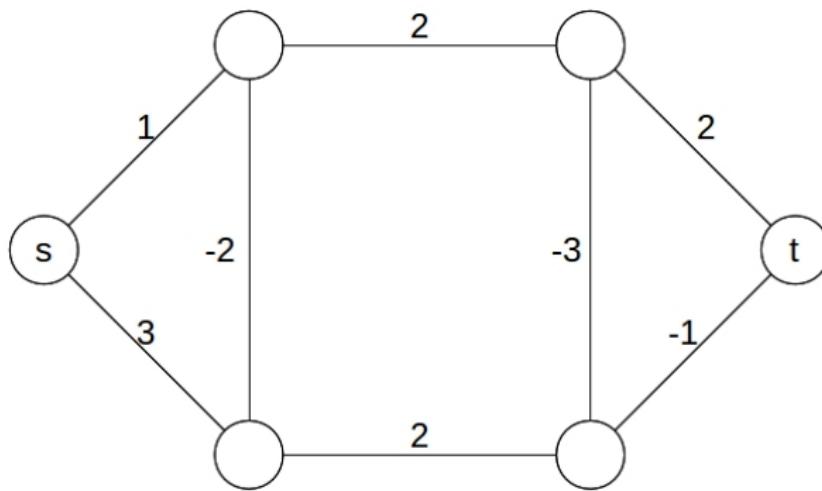
# Negative Edge Weights

What is the cost of the shortest path from  $s$  to  $t$  in the weighted graph below?



# Negative Edge Weights

What is the cost of the shortest path from  $s$  to  $t$  in the weighted graph below?



How do negative edge weights affect minimum spanning trees?

There is no effect.

# Edsger Wybe Dijkstra: 1930-2002

# Edsger Wybe Dijkstra: 1930-2002

- Received the Turing Award in 1972 for fundamental contributions to developing programming languages.

# Edsger Wybe Dijkstra: 1930-2002

- Received the Turing Award in 1972 for fundamental contributions to developing programming languages.
- He made fundamental contributions in the following domains:
  - Algorithm Design
  - Programming Languages
  - Program Design
  - Operating Systems
  - Distributed Processing
  - Formal specification and verification
  - Design of mathematical arguments

# Edsger Wybe Dijkstra: 1930-2002

“Computer Science is no more about computers than astronomy is about telescopes”

# Edsger Wybe Dijkstra: 1930-2002

“The competent programmer is fully aware of the strictly limited size of his own skull; therefore he approaches the programming task in full humility, and among other things he avoids clever tricks like the plague”  
(from 1972 Turing Award Lecture)

# Edsger Wybe Dijkstra: 1930-2002

“Program testing can best show the presence of errors, but never their absence”

# Edsger Wybe Dijkstra: 1930-2002

“If you don't know what your program is supposed to do, you'd better not start writing it”

# Edsger Wybe Dijkstra: 1930-2002

“The price of reliability is the pursuit of the utmost simplicity”

# Dijkstra's Algorithm

- The principle behind Dijkstra's algorithm is that if  $s, \dots, x, \dots, t$  is the shortest path from  $s$  to  $t$

# Dijkstra's Algorithm

- The principle behind Dijkstra's algorithm is that if  $s, \dots, x, \dots, t$  is the shortest path from  $s$  to  $t$ , then  $s, \dots, x$  had better be the shortest path from  $s$  to  $x$ .

# Dijkstra's Algorithm

- The principle behind Dijkstra's algorithm is that if  $s, \dots, x, \dots, t$  is the shortest path from  $s$  to  $t$ , then  $s, \dots, x$  had better be the shortest path from  $s$  to  $x$ .
- This suggests a dynamic programming-like strategy, where we store the distance from  $s$  to all nearby nodes, and use them to find the shortest path to more distant nodes.

# Dijkstra Basics

## *Output*

- Shortest Path Tree rooted at  $s$  (similar to BF tree)
- $\text{parent}[v]$
- distance (from  $s$ )  $d[v]$

## *Initialization*

- set each  $d[v]$  to  $\infty$  (except  $d[s] = 0$ )
- set each  $\text{parent}[v]$  to nil.

# Pseudocode

---

**Algorithm:** `dijkstra( $G, s$ )`

---

# Pseudocode

Set Initial Conditions

---

**Algorithm:** `dijkstra( $G, s$ )`

---

# Pseudocode

## Set Initial Conditions

- For each vertex

---

**Algorithm:**  $\text{dijkstra}(G, s)$

---

1 **for** each vertex  $v \in V$  **do**

|

# Pseudocode

## Set Initial Conditions

- For each vertex
  - The shortest “known” distance is  $\infty$

---

**Algorithm:** `dijkstra( $G, s$ )`

---

1 **for** each vertex  $v \in V$  **do**  
2      $d[v] = \infty$

---

# Pseudocode

## Set Initial Conditions

- For each vertex
  - The shortest “known” distance is  $\infty$
  - There is no parent

---

**Algorithm:** `dijkstra( $G, s$ )`

---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
```

# Pseudocode

## Set Initial Conditions

- For each vertex
  - The shortest “known” distance is  $\infty$
  - There is no parent
- The shortest path to the start vertex has distance 0.

---

**Algorithm:** dijkstra( $G, s$ )

---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4  $d[s] = 0$ 
```

# Pseudocode

## Set Initial Conditions

- For each vertex
  - The shortest “known” distance is  $\infty$
  - There is no parent
- The shortest path to the start vertex has distance 0.
- Create a set to hold the vertices we know the shortest path to.

---

**Algorithm:** dijkstra( $G, s$ )

---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4  $d[s] = 0$ 
5  $S = \emptyset$ 
```

# Pseudocode

## Set Initial Conditions

- For each vertex
  - The shortest “known” distance is  $\infty$
  - There is no parent
- The shortest path to the start vertex has distance 0.
- Create a set to hold the vertices we know the shortest path to.
- Create a priority queue on  $d$

---

**Algorithm:** dijkstra( $G, s$ )

---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
```

# Pseudocode

Look at each vertex in the priority queue

---

**Algorithm:**  $\text{dijkstra}(G, s)$

---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
```



# Pseudocode

Look at each vertex in the priority queue

- Extract the vertex,  $u$ , with the shortest path from  $s$

---

**Algorithm:**  $\text{dijkstra}(G, s)$

---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4  $d[s] = 0$ 
5  $S = \emptyset$ 
6  $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
```

---

# Pseudocode

Look at each vertex in the priority queue

- Extract the vertex,  $u$ , with the shortest path from  $s$
- Add  $u$  to the set  $S$  (we now know this must be the shortest path to  $u$ )

---

**Algorithm:** dijkstra( $G, s$ )

---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4  $d[s] = 0$ 
5  $S = \emptyset$ 
6  $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
```

---

# Pseudocode

Look at each vertex in the priority queue

- Extract the vertex,  $u$ , with the shortest path from  $s$
- Add  $u$  to the set  $S$  (we now know this must be the shortest path to  $u$ )
- Look at each neighbor,  $v$ , of  $u$

---

**Algorithm:** `dijkstra( $G, s$ )`

---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4  $d[s] = 0$ 
5  $S = \emptyset$ 
6  $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
do
```



# Pseudocode

Look at each vertex in the priority queue

- Extract the vertex,  $u$ , with the shortest path from  $s$
- Add  $u$  to the set  $S$  (we now know this must be the shortest path to  $u$ )
- Look at each neighbor,  $v$ , of  $u$ 
  - Check if the distance to  $v$  through  $u$  is less than the currently known distance to  $v$  ("relaxation step")

---

**Algorithm:** dijkstra( $G, s$ )

---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4  $d[s] = 0$ 
5  $S = \emptyset$ 
6  $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$  do
11    if  $d[v] > d[u] + w(u, v)$ 
        then
```

---

# Pseudocode

Look at each vertex in the priority queue

- Extract the vertex,  $u$ , with the shortest path from  $s$
- Add  $u$  to the set  $S$  (we now know this must be the shortest path to  $u$ )
- Look at each neighbor,  $v$ , of  $u$ 
  - Check if the distance to  $v$  through  $u$  is less than the currently known distance to  $v$  ("relaxation step")
  - Update the distance to  $v$

---

**Algorithm:** dijkstra( $G, s$ )

---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4  $d[s] = 0$ 
5  $S = \emptyset$ 
6  $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
```

---

# Pseudocode

Look at each vertex in the priority queue

- Extract the vertex,  $u$ , with the shortest path from  $s$
- Add  $u$  to the set  $S$  (we now know this must be the shortest path to  $u$ )
- Look at each neighbor,  $v$ , of  $u$ 
  - Check if the distance to  $v$  through  $u$  is less than the currently known distance to  $v$  ("relaxation step")
  - Update the distance to  $v$
  - Set the parent of  $v$

---

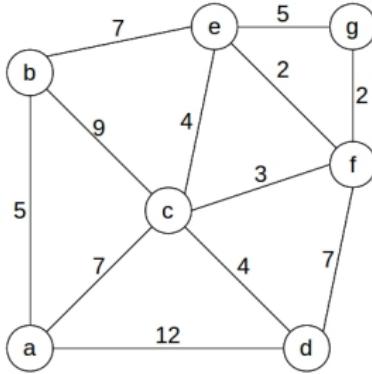
**Algorithm:** dijkstra( $G, s$ )

---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4  $d[s] = 0$ 
5  $S = \emptyset$ 
6  $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$  do
11    if  $d[v] > d[u] + w(u, v)$ 
12    then
13       $d[v] = d[u] + w(u, v)$ 
       $p[v] = u$ 
```

---

# Example



---

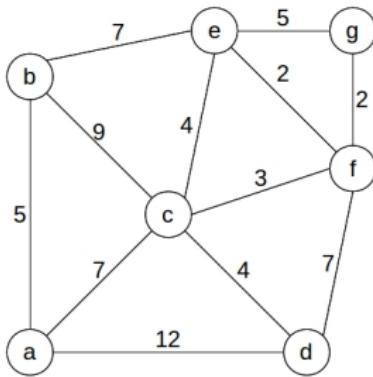
**Algorithm:** dijkstra( $G, s)$

---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

# Example

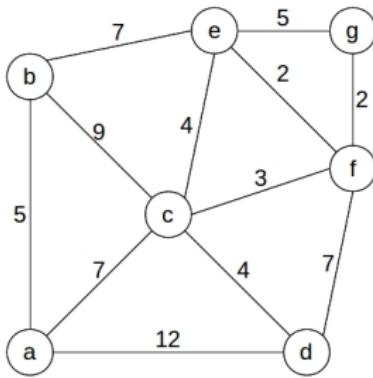


**Algorithm:** dijkstra( $G, s$ )

```
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$ 
3   |    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14          |    $d[v] = d[u] + w(u, v)$ 
15          |    $p[v] = u$ 
```

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	NULL	NULL	NULL	NULL	NULL	NULL
$S$							
$Q$	a, b, c, d, e, f, g						

# Example

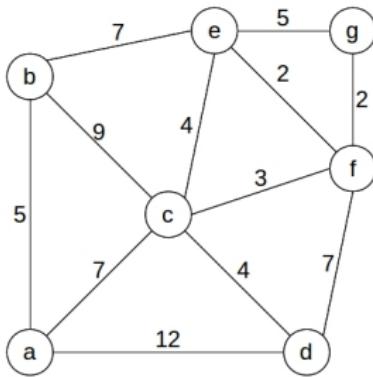


**Algorithm:** dijkstra( $G, s$ )

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	NULL	NULL	NULL	NULL	NULL	NULL
$S$							
$Q$	a, b, c, d, e, f, g						

# Example

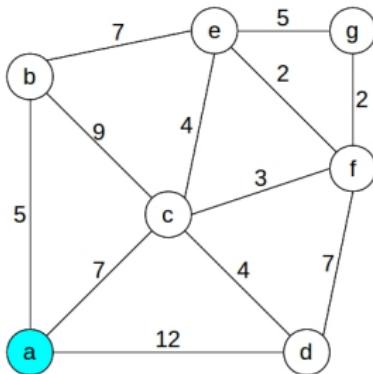


**Algorithm:** dijkstra( $G, s$ )

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	NULL	NULL	NULL	NULL	NULL	NULL
$S$							
$Q$	a, b, c, d, e, f, g						

# Example



$u = a$

---

**Algorithm:** dijkstra( $G, s$ )

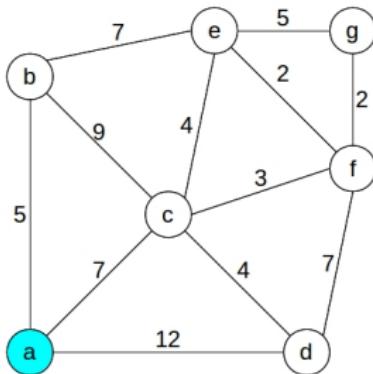
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	NULL	NULL	NULL	NULL	NULL	NULL
$S$							
$Q$	b, c, d, e, f, g						

# Example



$u = a$

---

**Algorithm:** dijkstra( $G, s$ )

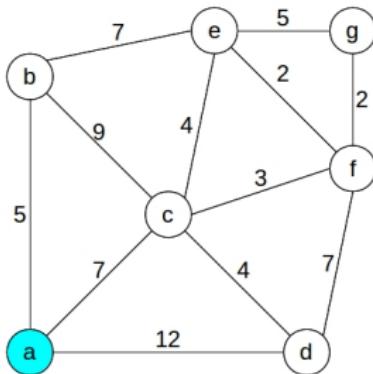
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	NULL	NULL	NULL	NULL	NULL	NULL
$S$	a						
$Q$	b, c, d, e, f, g						

# Example



$u = a$

---

**Algorithm:** dijkstra( $G, s$ )

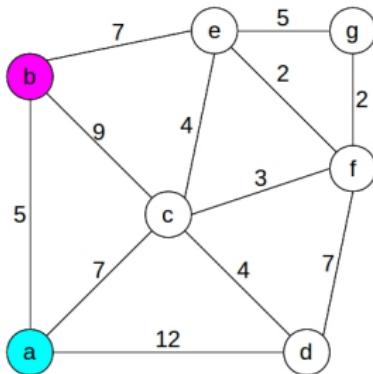
---

```
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$ 
3   |    $p[v] = \text{NULL}$ 
4   |    $d[s] = 0$ 
5   |    $S = \emptyset$ 
6   |    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$ 
9   |    $S = S \cup \{u\}$ 
10  |   for each vertex  $v$  adjacent to  $u$ 
11    |   |   do
12    |   |   |   if  $d[v] > d[u] + w(u, v)$ 
13    |   |   |   |   then
14    |   |   |   |   |    $d[v] = d[u] + w(u, v)$ 
15    |   |   |   |   |    $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	NULL	NULL	NULL	NULL	NULL	NULL
$S$	a						
$Q$	b, c, d, e, f, g						

# Example



$u = a$

$v = b$

---

**Algorithm:** dijkstra( $G, s$ )

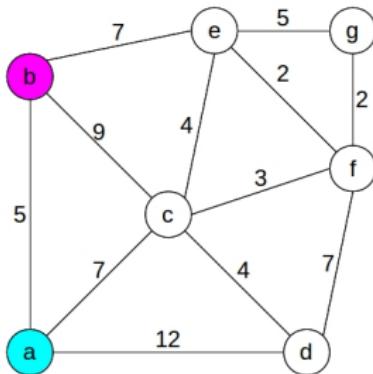
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	NULL	NULL	NULL	NULL	NULL	NULL
$S$	a						
$Q$	b, c, d, e, f, g						

# Example



$u = a$

$v = b$

---

**Algorithm:** dijkstra( $G, s$ )

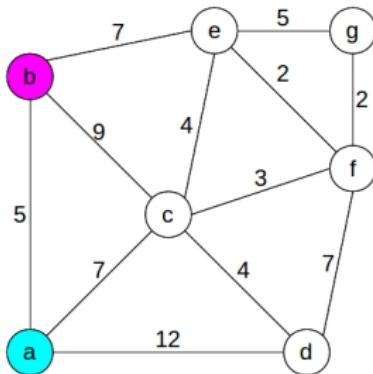
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	NULL	NULL	NULL	NULL	NULL	NULL
$S$	a						
$Q$	b, c, d, e, f, g						

# Example



$u = a$

$v = b$

---

**Algorithm:** dijkstra( $G, s$ )

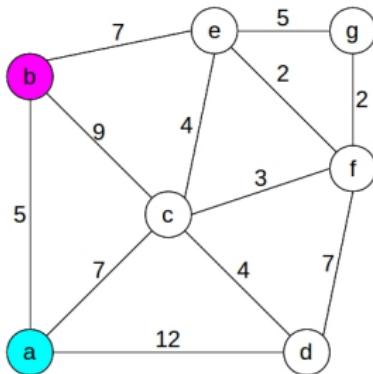
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	NULL	NULL	NULL	NULL	NULL	NULL
$S$	a						
$Q$	b, c, d, e, f, g						

# Example



$u = a$

$v = b$

---

**Algorithm:** dijkstra( $G, s$ )

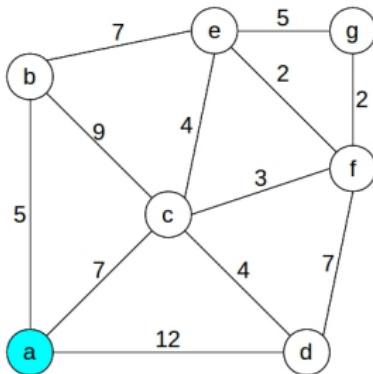
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	NULL	NULL	NULL	NULL	NULL
$S$	a						
$Q$	b, c, d, e, f, g						

# Example



$u = a$

---

**Algorithm:** dijkstra( $G, s$ )

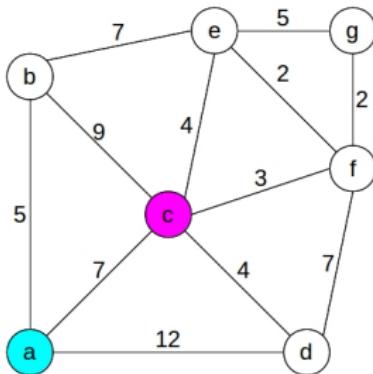
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	NULL	NULL	NULL	NULL	NULL
$S$	a						
$Q$	b, c, d, e, f, g						

# Example



$u = a$

$v = c$

---

**Algorithm:** dijkstra( $G, s$ )

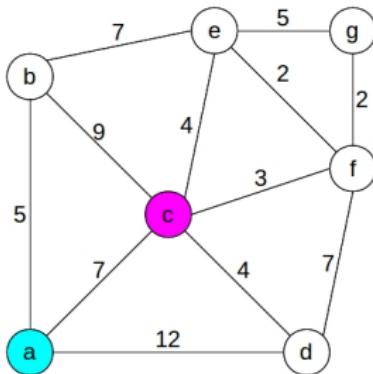
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	NULL	NULL	NULL	NULL	NULL
$S$	a						
$Q$	b, c, d, e, f, g						

# Example



$u = a$

$v = c$

---

**Algorithm:** dijkstra( $G, s$ )

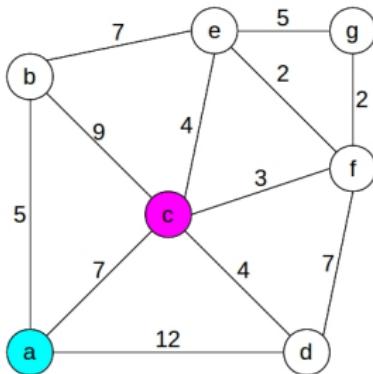
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	NULL	NULL	NULL	NULL	NULL
$S$	a						
$Q$	b, c, d, e, f, g						

# Example



$u = a$

$v = c$

---

**Algorithm:** dijkstra( $G, s$ )

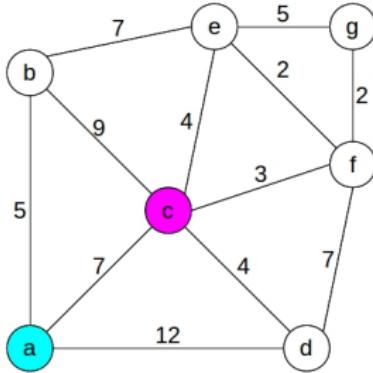
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	$\infty$	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	NULL	NULL	NULL	NULL	NULL
$S$	a						
$Q$	b, c, d, e, f, g						

# Example



$u = a$

$v = c$

---

**Algorithm:** dijkstra( $G, s$ )

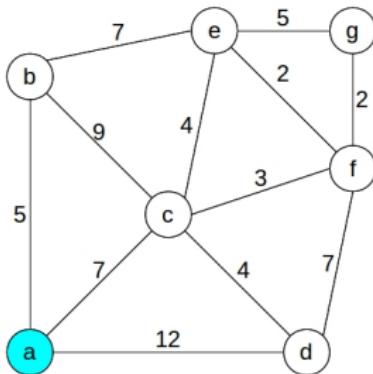
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	$\infty$	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	a	NULL	NULL	NULL	NULL
$S$	a						
$Q$	b, c, d, e, f, g						

# Example



$u = a$

---

**Algorithm:** dijkstra( $G, s$ )

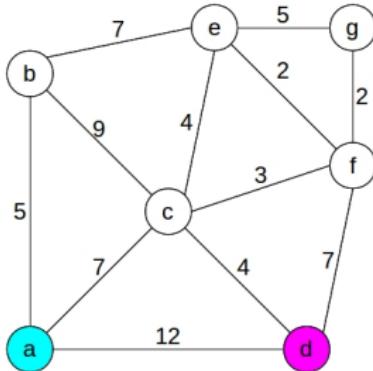
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	$\infty$	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	a	NULL	NULL	NULL	NULL
$S$	a						
$Q$	b, c, d, e, f, g						

# Example



$u = a$

$v = d$

---

**Algorithm:** dijkstra( $G, s$ )

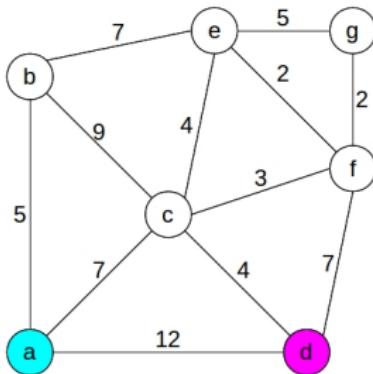
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	$\infty$	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	a	NULL	NULL	NULL	NULL
$S$	a						
$Q$	b, c, d, e, f, g						

# Example



$u = a$

$v = d$

---

**Algorithm:** dijkstra( $G, s$ )

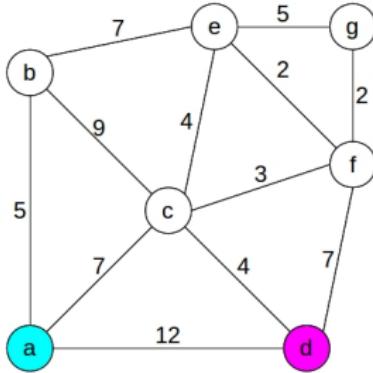
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	$\infty$	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	a	NULL	NULL	NULL	NULL
$S$	a						
$Q$	b, c, d, e, f, g						

# Example



$u = a$

$v = d$

---

**Algorithm:** dijkstra( $G, s$ )

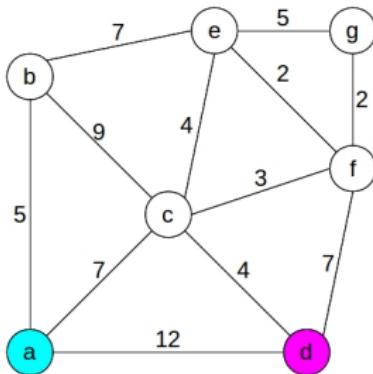
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	a	NULL	NULL	NULL	NULL
$S$	a						
$Q$	b, c, d, e, f, g						

# Example



$u = a$

$v = d$

---

**Algorithm:** dijkstra( $G, s$ )

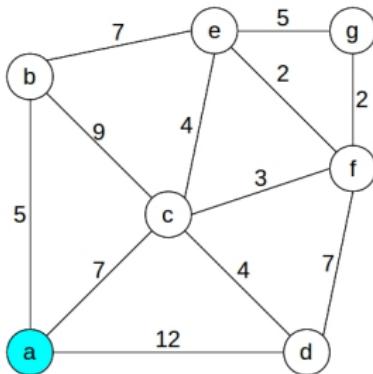
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	NULL	NULL	NULL
$S$	a						
$Q$	b, c, d, e, f, g						

# Example



$u = a$

---

**Algorithm:** dijkstra( $G, s$ )

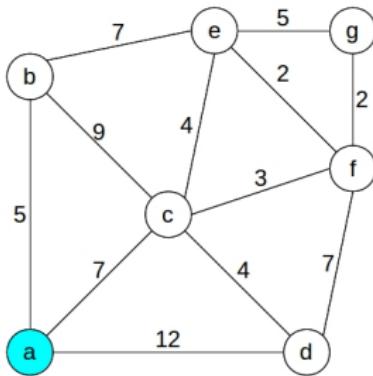
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	NULL	NULL	NULL
$S$	a						
$Q$	b, c, d, e, f, g						

# Example

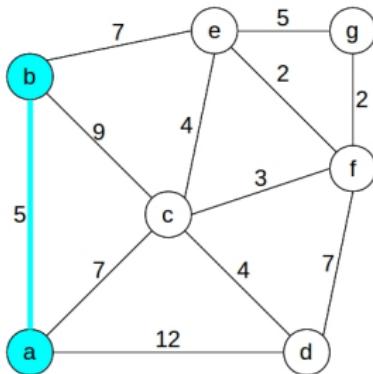


**Algorithm:** dijkstra( $G, s)$

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	NULL	NULL	NULL
$S$	a						
$Q$	b, c, d, e, f, g						

# Example



$u = b$

---

**Algorithm:** dijkstra( $G, s$ )

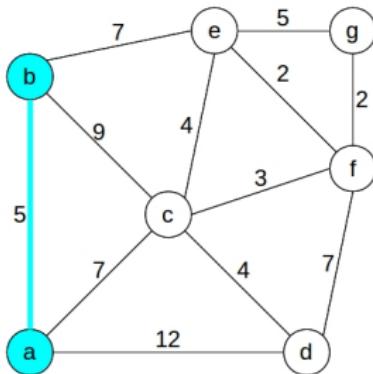
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	NULL	NULL	NULL
$S$	a						
$Q$	c, d, e, f, g						

# Example



$u = b$

---

**Algorithm:** dijkstra( $G, s$ )

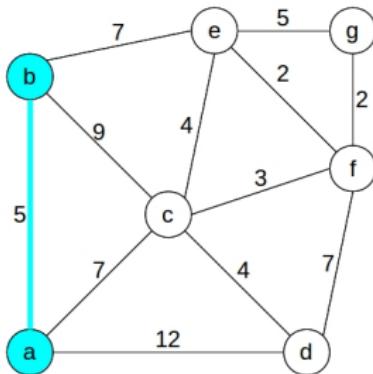
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	NULL	NULL	NULL
$S$	a, b						
$Q$	c, d, e, f, g						

# Example



$u = b$

---

**Algorithm:** dijkstra( $G, s$ )

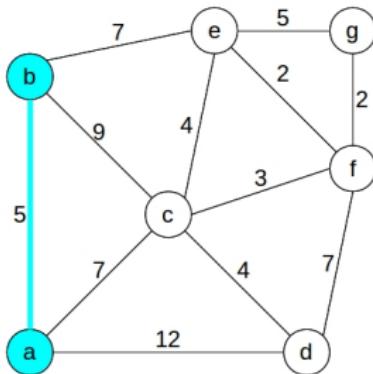
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	NULL	NULL	NULL
$S$	a, b						
$Q$	c, d, e, f, g						

# Example



$$u = b  
v = a$$

---

**Algorithm:** dijkstra( $G, s$ )

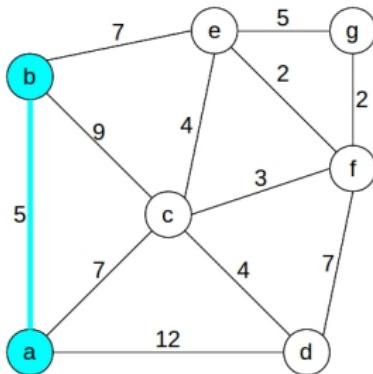
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	NULL	NULL	NULL
$S$	a, b						
$Q$	c, d, e, f, g						

# Example



$$u = b  
v = a$$

---

**Algorithm:** dijkstra( $G, s$ )

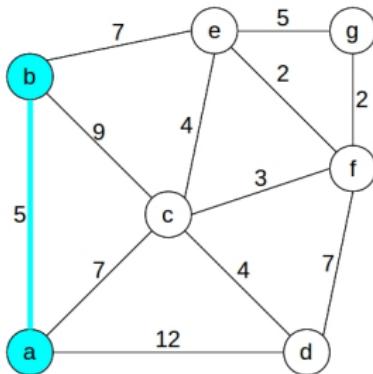
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	NULL	NULL	NULL
$S$	a, b						
$Q$	c, d, e, f, g						

# Example



$u = b$

---

**Algorithm:** dijkstra( $G, s$ )

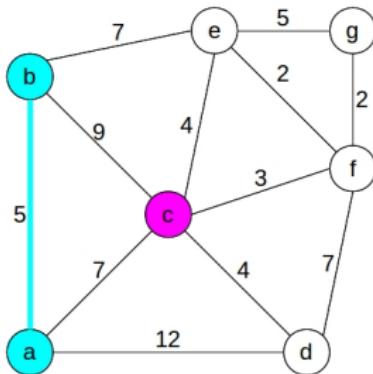
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	NULL	NULL	NULL
$S$	a, b						
$Q$	c, d, e, f, g						

# Example



$$u = b  
v = c$$

---

**Algorithm:** dijkstra( $G, s$ )

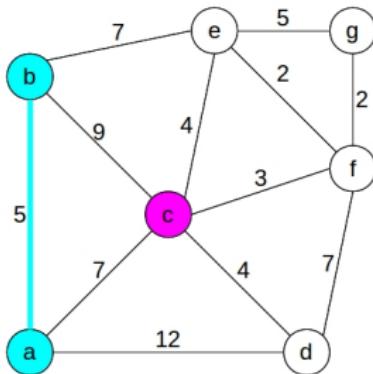
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	NULL	NULL	NULL
$S$	a, b						
$Q$	c, d, e, f, g						

# Example



$$u = b$$
$$v = c$$

---

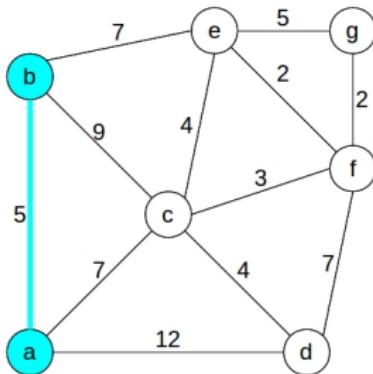
**Algorithm:** dijkstra( $G, s$ )

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	NULL	NULL	NULL
$S$	a, b						
$Q$	c, d, e, f, g						

# Example



$u = b$

---

**Algorithm:** dijkstra( $G, s$ )

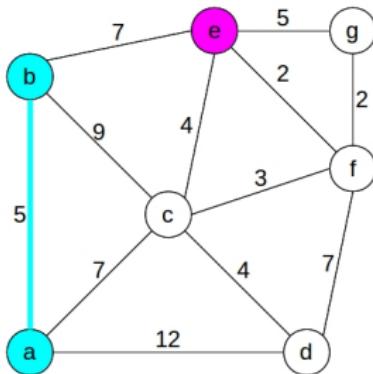
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	NULL	NULL	NULL
$S$	a, b						
$Q$	c, d, e, f, g						

# Example



$$u = b  
v = e$$

---

**Algorithm:** dijkstra( $G, s$ )

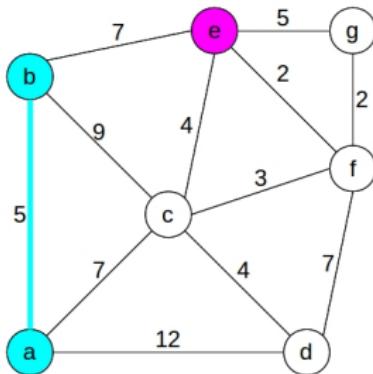
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	NULL	NULL	NULL
$S$	a, b						
$Q$	c, d, e, f, g						

# Example



$$u = b  
v = e$$

---

**Algorithm:** dijkstra( $G, s$ )

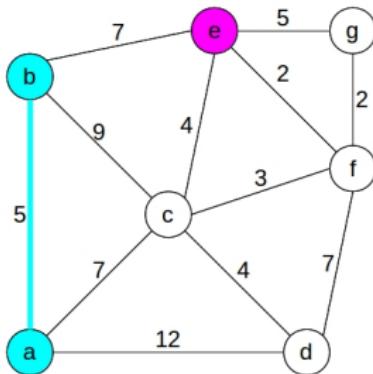
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	$\infty$	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	NULL	NULL	NULL
$S$	a, b						
$Q$	c, d, e, f, g						

# Example



$u = b$   
 $v = e$

---

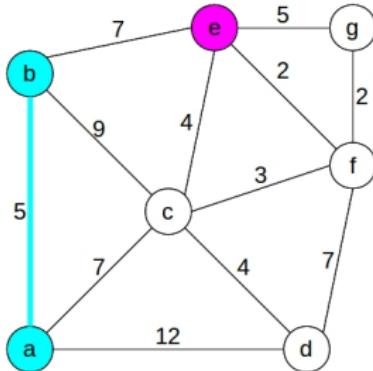
**Algorithm:** dijkstra( $G, s$ )

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	12	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	NULL	NULL	NULL
$S$	a, b						
$Q$	c, d, e, f, g						

# Example



$$u = b  
v = e$$

---

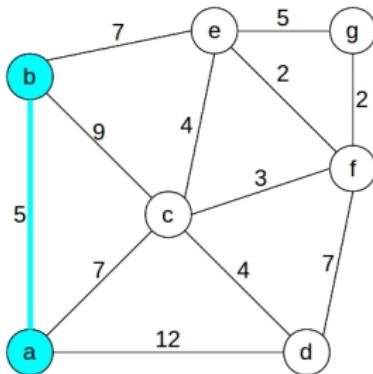
**Algorithm:** dijkstra( $G, s$ )

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	12	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	b	NULL	NULL
$S$	a, b						
$Q$	c, d, e, f, g						

# Example



$u = b$

---

**Algorithm:** dijkstra( $G, s$ )

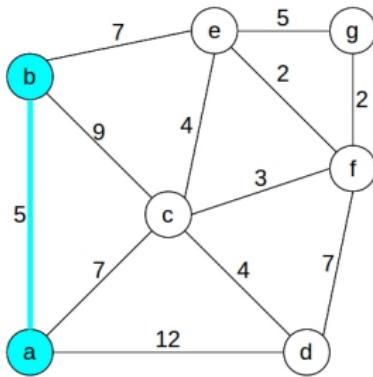
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	12	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	b	NULL	NULL
$S$	a, b						
$Q$	c, d, e, f, g						

# Example



---

**Algorithm:** dijkstra( $G, s)$

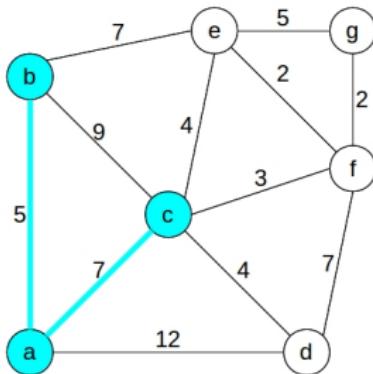
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	12	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	b	NULL	NULL
$S$	a, b						
$Q$	c, d, e, f, g						

# Example



$u = c$

---

**Algorithm:** dijkstra( $G, s$ )

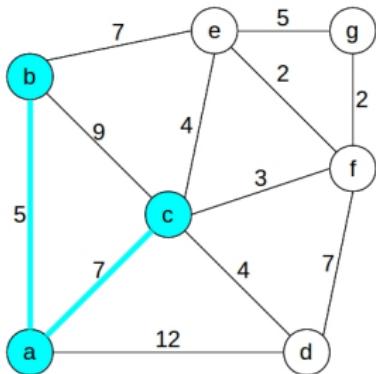
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	12	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	b	NULL	NULL
$S$	a, b						
$Q$	c, d, e, f, g						

# Example



$u = c$

---

**Algorithm:** dijkstra( $G, s)$

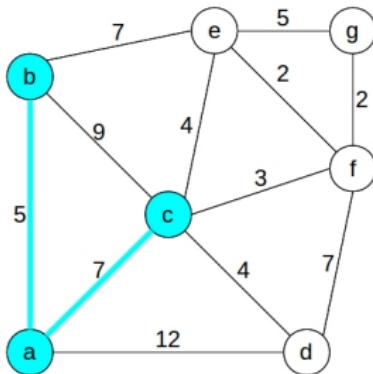
---

```
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$ 
3   |    $p[v] = \text{NULL}$ 
4   |    $d[s] = 0$ 
5   |    $S = \emptyset$ 
6   |    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$ 
9   |    $S = S \cup \{u\}$ 
10  |   for each vertex  $v$  adjacent to  $u$ 
11    |   |   do
12    |   |   |   if  $d[v] > d[u] + w(u, v)$ 
13    |   |   |   |   then
14    |   |   |   |   |    $d[v] = d[u] + w(u, v)$ 
15    |   |   |   |   |    $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	12	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	b	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

---

**Algorithm:** dijkstra( $G, s$ )

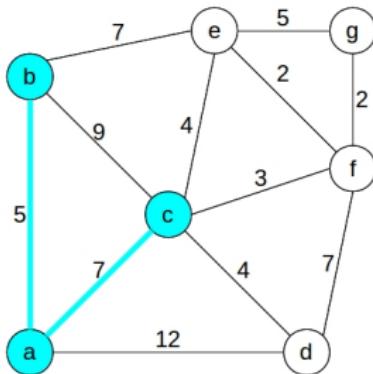
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	12	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	b	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

$v = a$

---

**Algorithm:** dijkstra( $G, s$ )

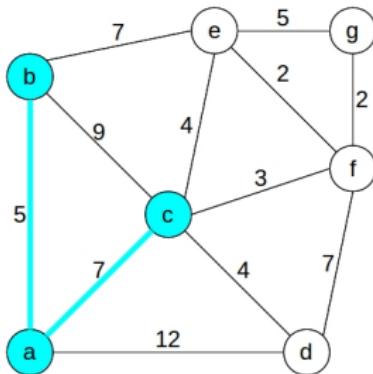
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	12	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	b	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

$v = a$

---

**Algorithm:** dijkstra( $G, s$ )

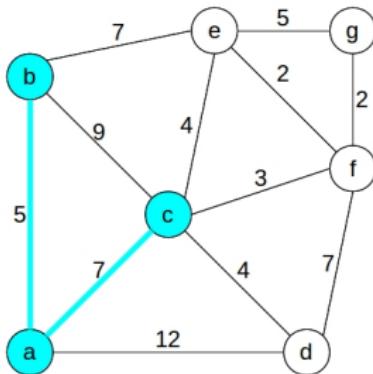
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	12	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	b	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

---

**Algorithm:** dijkstra( $G, s$ )

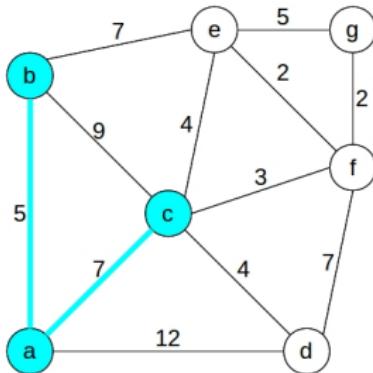
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	12	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	b	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

$v = b$

---

**Algorithm:** dijkstra( $G, s$ )

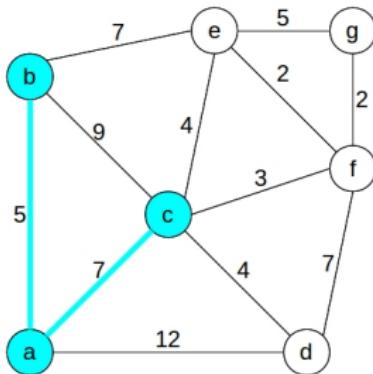
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	12	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	b	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

$v = b$

---

**Algorithm:** dijkstra( $G, s$ )

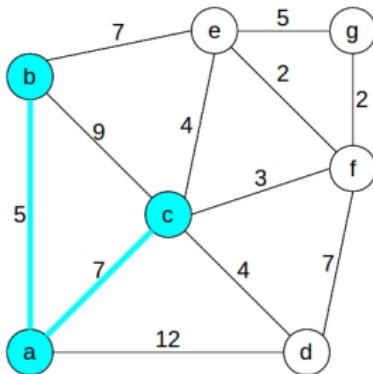
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	12	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	b	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

---

**Algorithm:** dijkstra( $G, s$ )

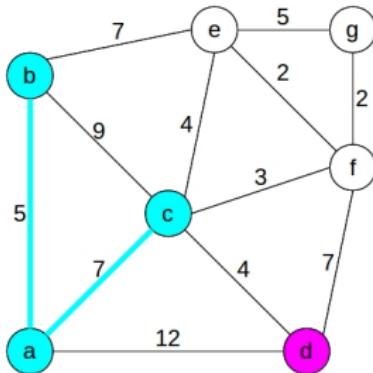
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	12	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	b	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

$v = d$

---

**Algorithm:** dijkstra( $G, s$ )

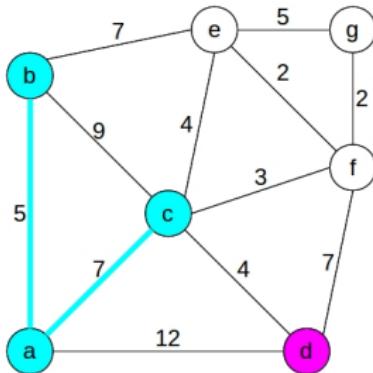
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	12	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	b	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

$v = d$

---

**Algorithm:** dijkstra( $G, s$ )

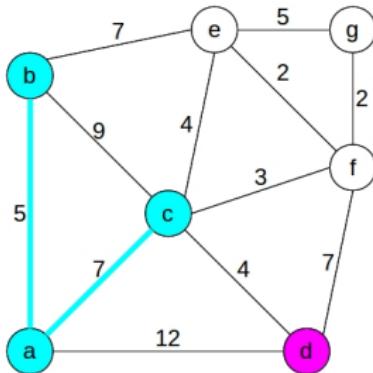
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	12	12	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	b	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

$v = d$

---

**Algorithm:** dijkstra( $G, s$ )

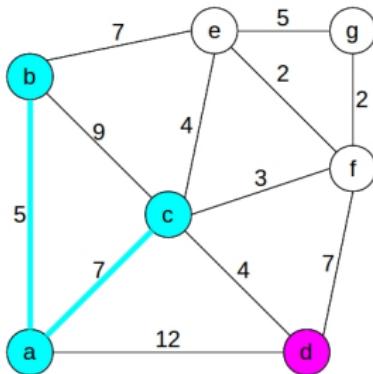
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	12	$\infty$	$\infty$
$p[u]$	NULL	a	a	a	b	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

$v = d$

---

**Algorithm:** dijkstra( $G, s$ )

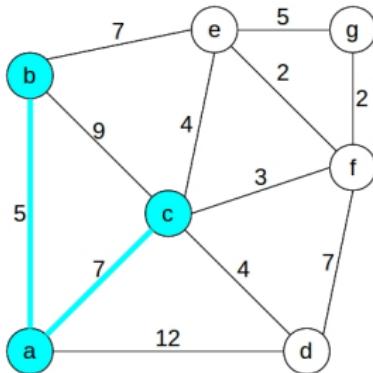
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	12	$\infty$	$\infty$
$p[u]$	NULL	a	a	c	b	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

---

**Algorithm:** dijkstra( $G, s$ )

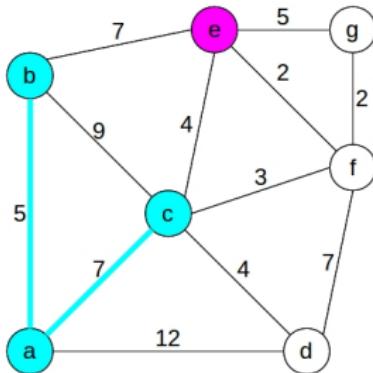
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	12	$\infty$	$\infty$
$p[u]$	NULL	a	a	c	b	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

$v = e$

---

**Algorithm:** dijkstra( $G, s$ )

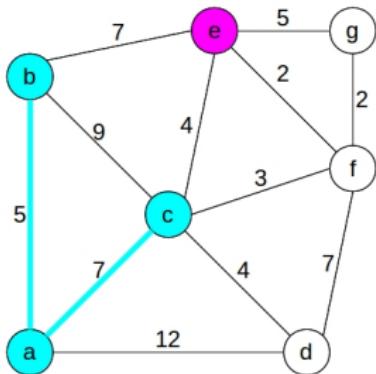
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	12	$\infty$	$\infty$
$p[u]$	NULL	a	a	c	b	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

$v = e$

---

**Algorithm:** dijkstra( $G, s$ )

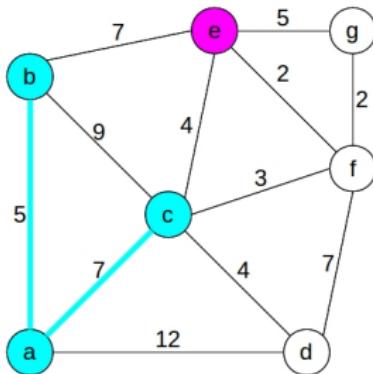
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	12	$\infty$	$\infty$
$p[u]$	NULL	a	a	c	b	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

$v = e$

---

**Algorithm:** dijkstra( $G, s$ )

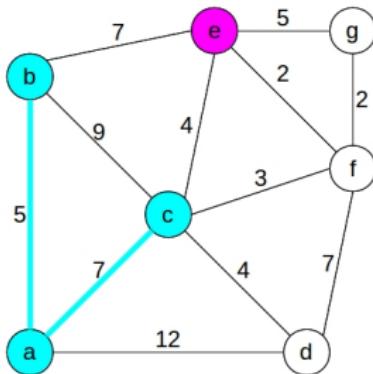
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	$\infty$	$\infty$
$p[u]$	NULL	a	a	c	b	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

$v = e$

---

**Algorithm:** dijkstra( $G, s$ )

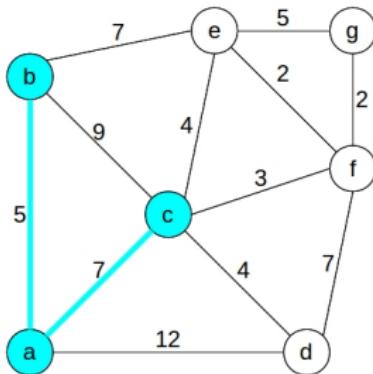
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	$\infty$	$\infty$
$p[u]$	NULL	a	a	c	c	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

---

**Algorithm:** dijkstra( $G, s$ )

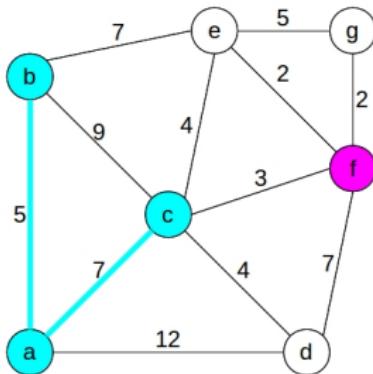
---

```
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$ 
3   |    $p[v] = \text{NULL}$ 
4   |    $d[s] = 0$ 
5   |    $S = \emptyset$ 
6   |    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$ 
9   |    $S = S \cup \{u\}$ 
10  |   for each vertex  $v$  adjacent to  $u$ 
11    |   |   do
12    |   |   |   if  $d[v] > d[u] + w(u, v)$ 
13    |   |   |   |   then
14    |   |   |   |   |    $d[v] = d[u] + w(u, v)$ 
15    |   |   |   |   |    $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	$\infty$	$\infty$
$p[u]$	NULL	a	a	c	c	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

$v = f$

---

**Algorithm:** dijkstra( $G, s$ )

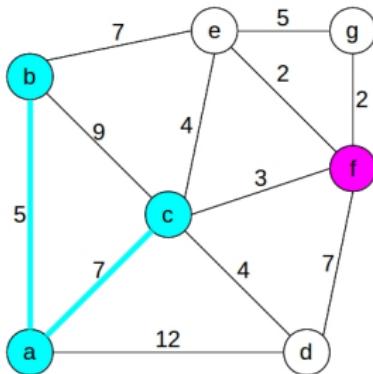
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	$\infty$	$\infty$
$p[u]$	NULL	a	a	c	c	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

$v = f$

---

**Algorithm:** dijkstra( $G, s$ )

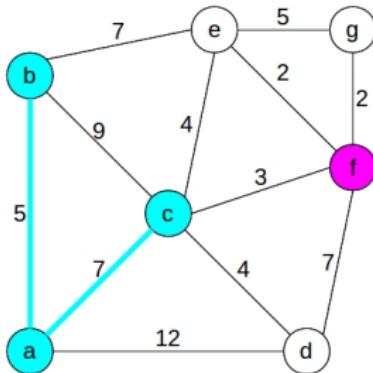
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	$\infty$	$\infty$
$p[u]$	NULL	a	a	c	c	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

$v = f$

---

**Algorithm:** dijkstra( $G, s$ )

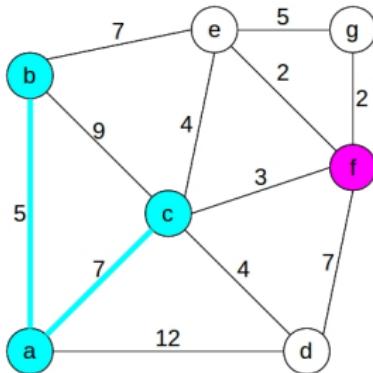
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	$\infty$
$p[u]$	NULL	a	a	c	c	NULL	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

$v = f$

---

**Algorithm:** dijkstra( $G, s$ )

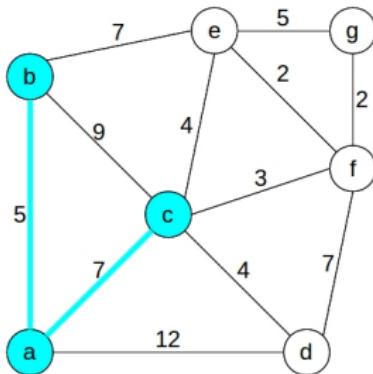
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	$\infty$
$p[u]$	NULL	a	a	c	c	c	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example



$u = c$

---

**Algorithm:** dijkstra( $G, s$ )

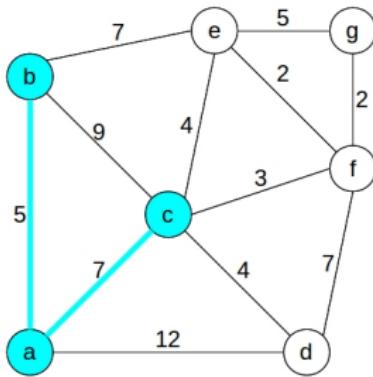
---

```
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$ 
3   |    $p[v] = \text{NULL}$ 
4   |    $d[s] = 0$ 
5   |    $S = \emptyset$ 
6   |    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$ 
9   |    $S = S \cup \{u\}$ 
10  |   for each vertex  $v$  adjacent to  $u$ 
11    |   |   do
12    |   |   |   if  $d[v] > d[u] + w(u, v)$ 
13    |   |   |   |   then
14    |   |   |   |   |    $d[v] = d[u] + w(u, v)$ 
15    |   |   |   |   |    $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	$\infty$
$p[u]$	NULL	a	a	c	c	c	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example

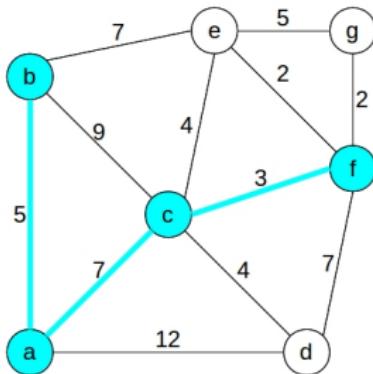


**Algorithm:** dijkstra( $G, s)$

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	$\infty$
$p[u]$	NULL	a	a	c	c	c	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example

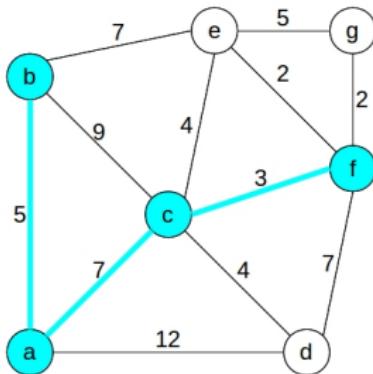


**Algorithm:** dijkstra( $G, s$ )

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	$\infty$
$p[u]$	NULL	a	a	c	c	c	NULL
$S$	a, b, c						
$Q$	d, e, f, g						

# Example

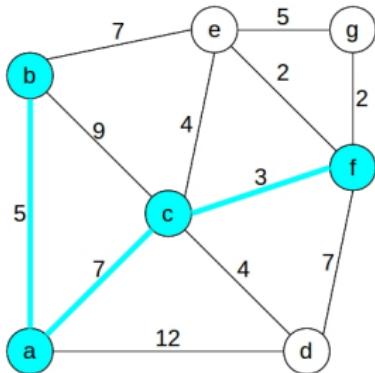


**Algorithm:** dijkstra( $G, s$ )

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	$\infty$
$p[u]$	NULL	a	a	c	c	c	NULL
$S$	a, b, c, f						
$Q$	d, e, g						

# Example

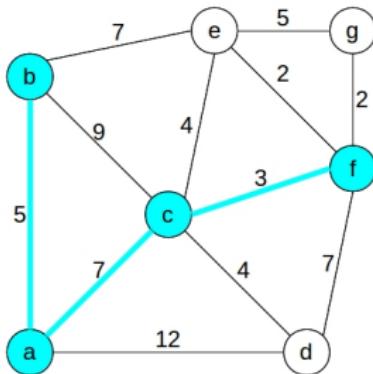


**Algorithm:** dijkstra( $G, s$ )

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	$\infty$
$p[u]$	NULL	a	a	c	c	c	NULL
$S$	a, b, c, f						
$Q$	d, e, g						

# Example



$$u = f  
v = c$$

---

**Algorithm:** dijkstra( $G, s$ )

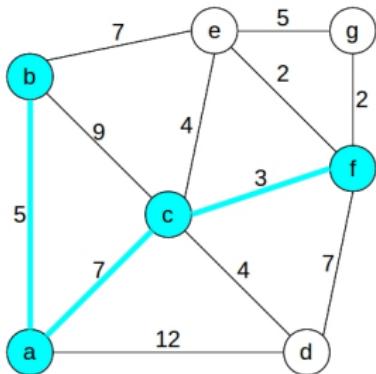
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	$\infty$
$p[u]$	NULL	a	a	c	c	c	NULL
$S$	a, b, c, f						
$Q$	d, e, g						

# Example



$$u = f  
v = c$$

---

**Algorithm:** dijkstra( $G, s$ )

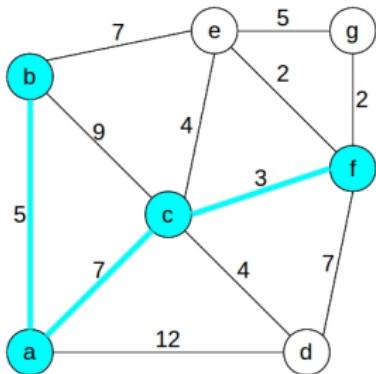
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	$\infty$
$p[u]$	NULL	a	a	c	c	c	NULL
$S$	a, b, c, f						
$Q$	d, e, g						

# Example

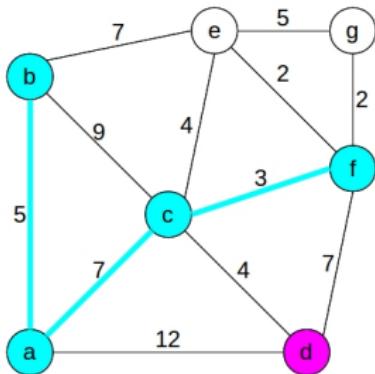


**Algorithm:** dijkstra( $G, s$ )

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	$\infty$
$p[u]$	NULL	a	a	c	c	c	NULL
$S$	a, b, c, f						
$Q$	d, e, g						

# Example



$u = f$

$v = d$

---

**Algorithm:** dijkstra( $G, s$ )

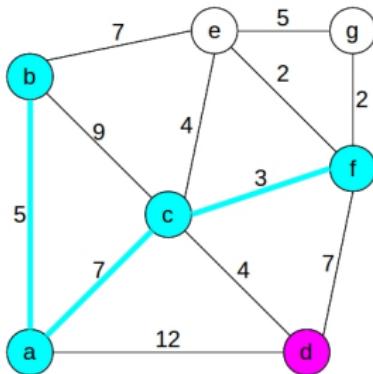
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	$\infty$
$p[u]$	NULL	a	a	c	c	c	NULL
$S$	a, b, c, f						
$Q$	d, e, g						

# Example



$u = f$

$v = d$

---

**Algorithm:** dijkstra( $G, s)$

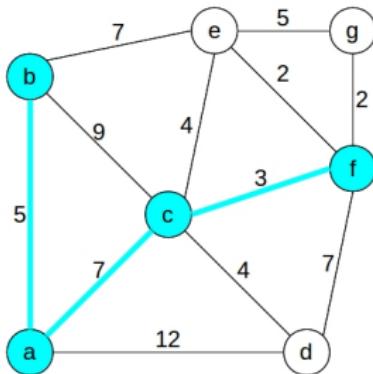
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	$\infty$
$p[u]$	NULL	a	a	c	c	c	NULL
$S$	a, b, c, f						
$Q$	d, e, g						

# Example



$u = f$

---

**Algorithm:** dijkstra( $G, s$ )

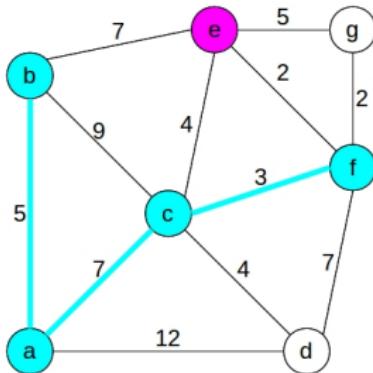
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	$\infty$
$p[u]$	NULL	a	a	c	c	c	NULL
$S$	a, b, c, f						
$Q$	d, e, g						

# Example



$$u = f  
v = e$$

---

**Algorithm:** dijkstra( $G, s$ )

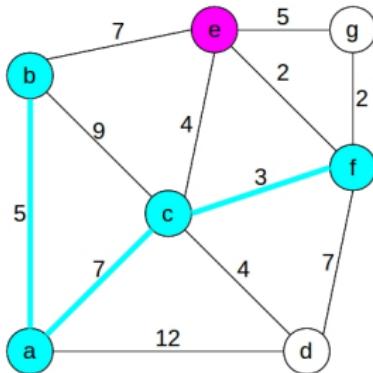
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	$\infty$
$p[u]$	NULL	a	a	c	c	c	NULL
$S$	a, b, c, f						
$Q$	d, e, g						

# Example



$$u = f  
v = e$$

---

**Algorithm:** dijkstra( $G, s$ )

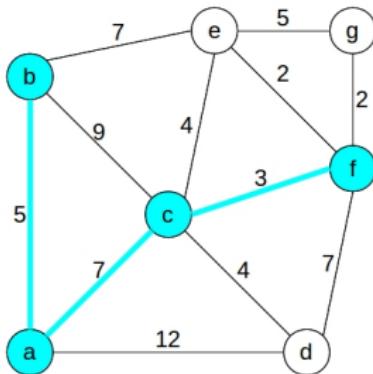
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	$\infty$
$p[u]$	NULL	a	a	c	c	c	NULL
$S$	a, b, c, f						
$Q$	d, e, g						

# Example

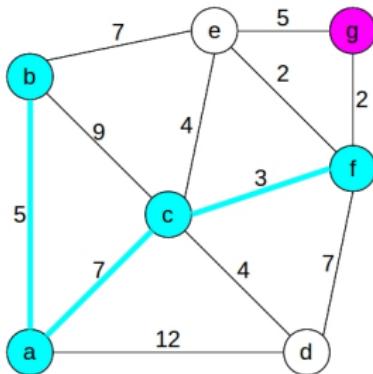


**Algorithm:** dijkstra( $G, s$ )

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	$\infty$
$p[u]$	NULL	a	a	c	c	c	NULL
$S$	a, b, c, f						
$Q$	d, e, g						

# Example



$u = f$

$v = g$

---

**Algorithm:** dijkstra( $G, s$ )

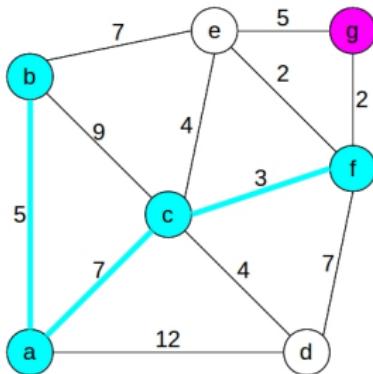
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	$\infty$
$p[u]$	NULL	a	a	c	c	c	NULL
$S$	a, b, c, f						
$Q$	d, e, g						

# Example



$u = f$

$v = g$

---

**Algorithm:** dijkstra( $G, s$ )

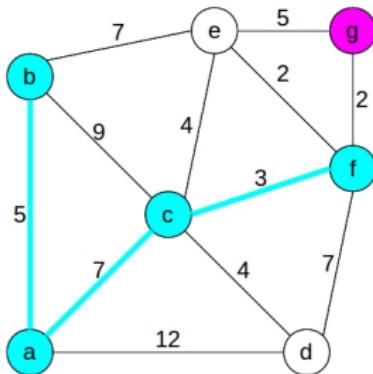
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	$\infty$
$p[u]$	NULL	a	a	c	c	c	NULL
$S$	a, b, c, f						
$Q$	d, e, g						

# Example



$u = f$

$v = g$

---

**Algorithm:** dijkstra( $G, s$ )

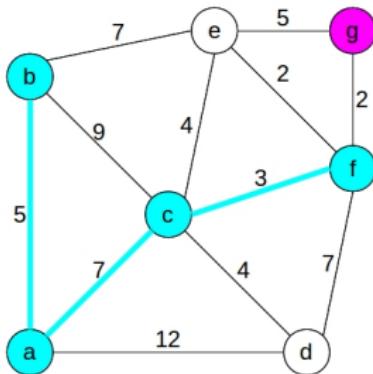
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	NULL
$S$	a, b, c, f						
$Q$	d, e, g						

# Example



$u = f$

$v = g$

---

**Algorithm:** dijkstra( $G, s$ )

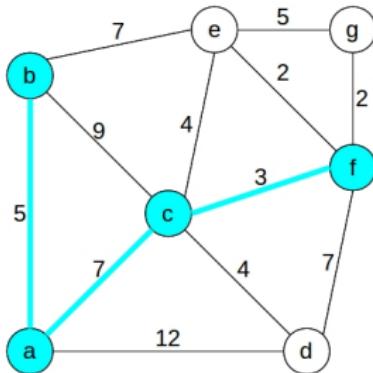
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f						
$Q$	d, e, g						

# Example



$u = f$

---

**Algorithm:** dijkstra( $G, s$ )

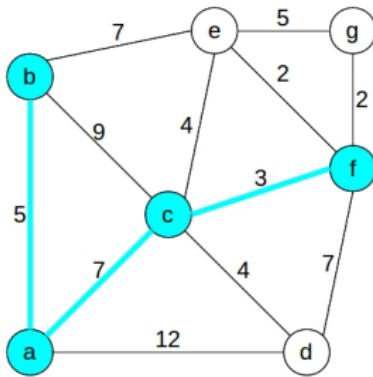
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f						
$Q$	d, e, g						

# Example

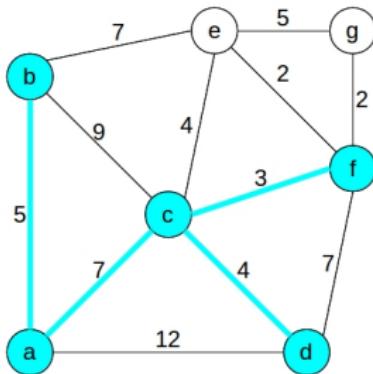


**Algorithm:** dijkstra( $G, s)$

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f						
$Q$	d, e, g						

# Example



$u = d$

---

**Algorithm:** dijkstra( $G, s$ )

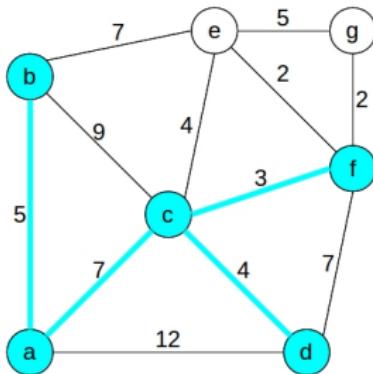
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f						
$Q$	e, g						

# Example



$u = d$

---

**Algorithm:** dijkstra( $G, s$ )

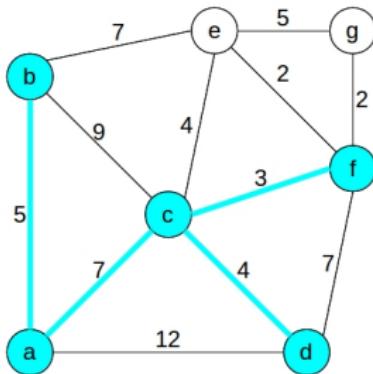
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, <b>d</b>						
$Q$	e, g						

# Example



$u = d$

---

**Algorithm:** dijkstra( $G, s$ )

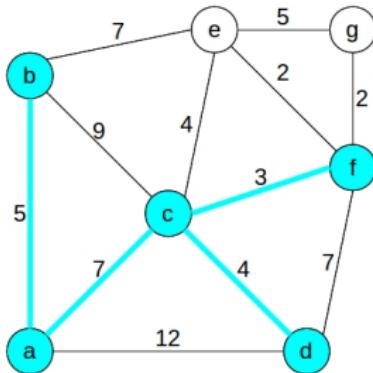
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d						
$Q$	e, g						

# Example



$$u = d  
v = a$$

---

**Algorithm:** dijkstra( $G, s$ )

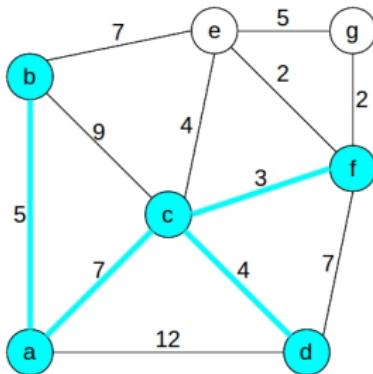
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d						
$Q$	e, g						

# Example



$$u = d  
v = a$$

---

**Algorithm:** dijkstra( $G, s$ )

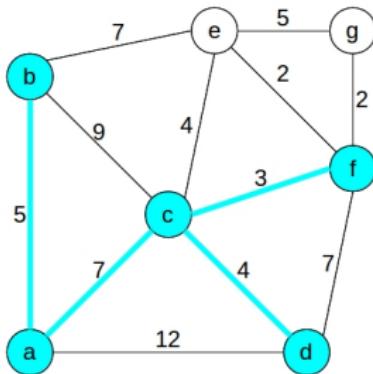
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d						
$Q$	e, g						

# Example



$u = d$

---

**Algorithm:** dijkstra( $G, s$ )

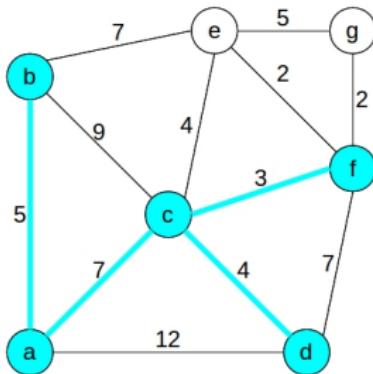
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d						
$Q$	e, g						

# Example



$$u = d  
v = c$$

---

**Algorithm:** dijkstra( $G, s$ )

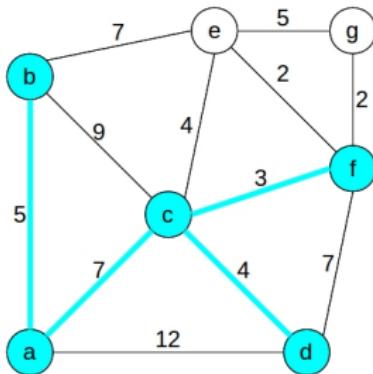
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d						
$Q$	e, g						

# Example



$$u = d  
v = c$$

---

**Algorithm:** dijkstra( $G, s$ )

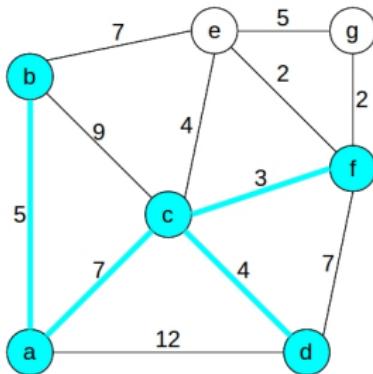
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d						
$Q$	e, g						

# Example



$u = d$

---

**Algorithm:** dijkstra( $G, s$ )

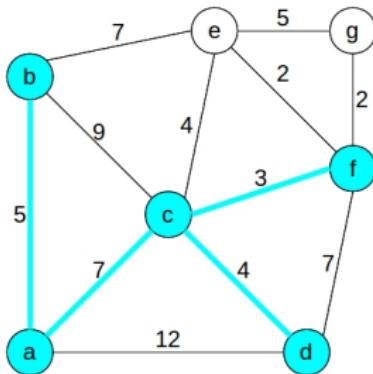
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d						
$Q$	e, g						

# Example



$$u = d$$

$$v = f$$

---

**Algorithm:** dijkstra( $G, s$ )

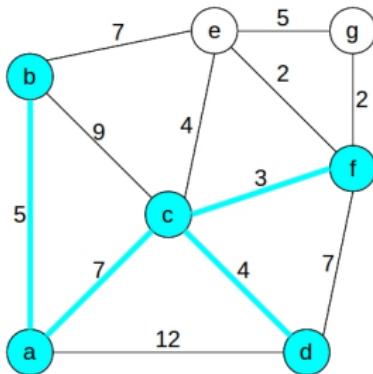
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d						
$Q$	e, g						

# Example



$u = d$

$v = f$

---

**Algorithm:** dijkstra( $G, s)$

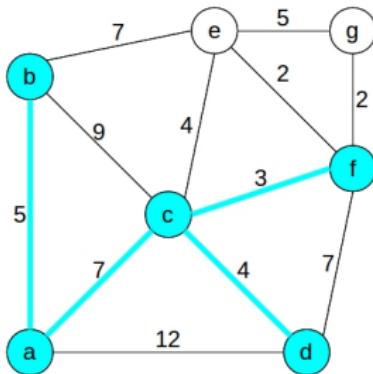
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d						
$Q$	e, g						

# Example



$u = d$

---

**Algorithm:** dijkstra( $G, s$ )

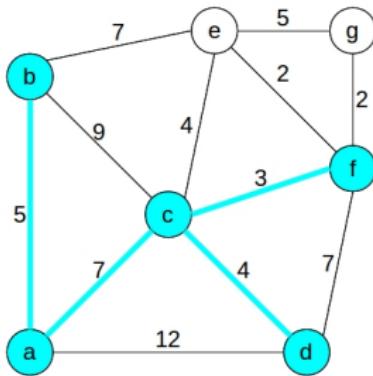
---

```
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$ 
3   |    $p[v] = \text{NULL}$ 
4   |    $d[s] = 0$ 
5   |    $S = \emptyset$ 
6   |    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$ 
9   |    $S = S \cup \{u\}$ 
10  |   for each vertex  $v$  adjacent to  $u$ 
11    |   |   do
12    |   |   |   if  $d[v] > d[u] + w(u, v)$ 
13    |   |   |   |   then
14    |   |   |   |   |    $d[v] = d[u] + w(u, v)$ 
15    |   |   |   |   |    $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d						
$Q$	e, g						

# Example

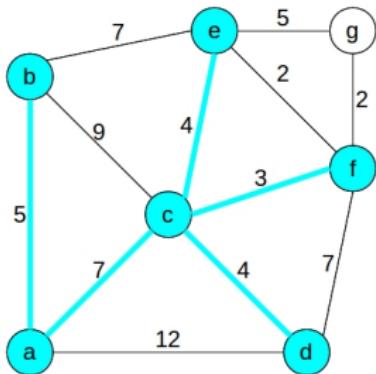


**Algorithm:** dijkstra( $G, s)$

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d						
$Q$	e, g						

# Example



$u = e$

---

**Algorithm:** dijkstra( $G, s)$

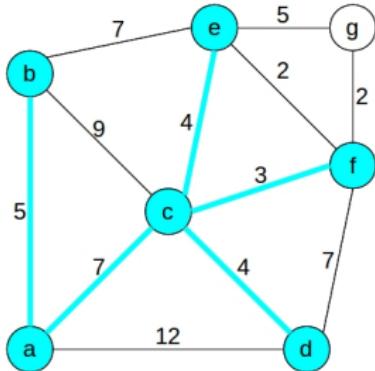
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d						
$Q$	g						

# Example

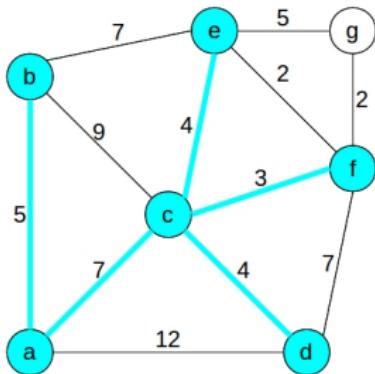


**Algorithm:** dijkstra( $G, s$ )

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, <b>e</b>						
$Q$	g						

# Example



$u = e$

---

**Algorithm:** dijkstra( $G, s$ )

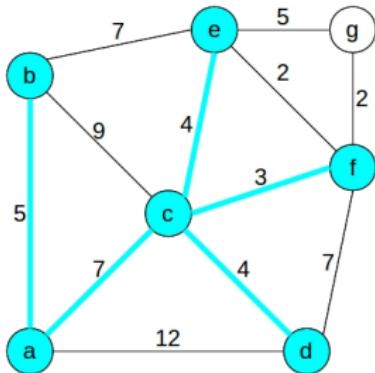
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e						
$Q$	g						

# Example



$u = e$

$v = b$

---

**Algorithm:** dijkstra( $G, s$ )

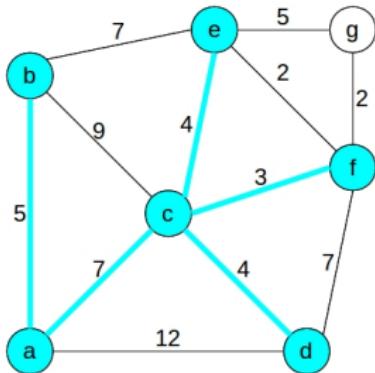
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e						
$Q$	g						

# Example



$u = e$

$v = b$

---

**Algorithm:** dijkstra( $G, s)$

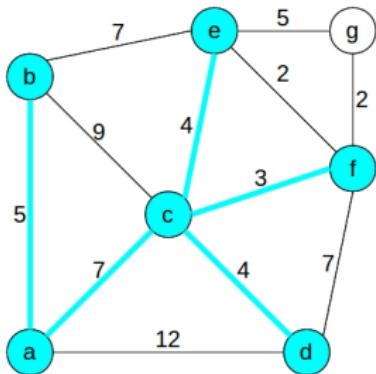
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e						
$Q$	g						

# Example



---

**Algorithm:** dijkstra( $G, s$ )

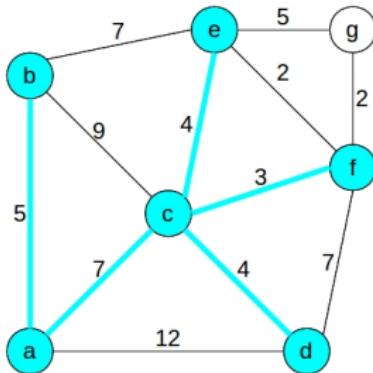
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e						
$Q$	g						

# Example



$u = e$

$v = c$

---

**Algorithm:** dijkstra( $G, s$ )

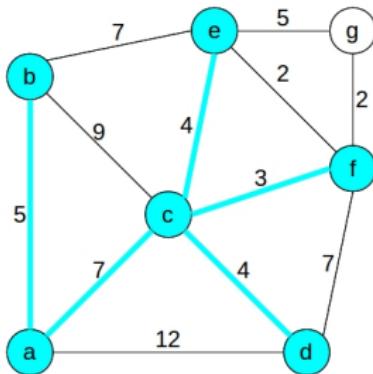
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e						
$Q$	g						

# Example



$u = e$

$v = c$

---

**Algorithm:** dijkstra( $G, s$ )

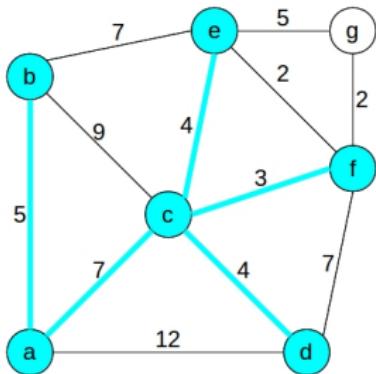
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e						
$Q$	g						

# Example



---

**Algorithm:** dijkstra( $G, s$ )

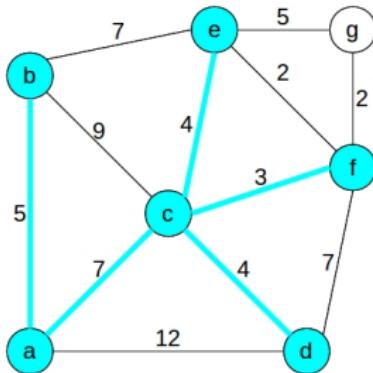
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e						
$Q$	g						

# Example



$u = e$

$v = f$

---

**Algorithm:** dijkstra( $G, s$ )

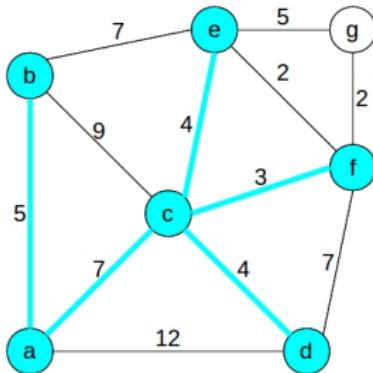
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e						
$Q$	g						

# Example



$u = e$

$v = f$

---

**Algorithm:** dijkstra( $G, s$ )

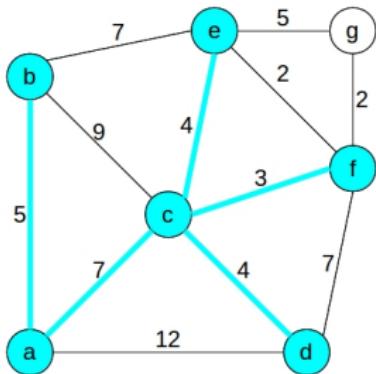
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e						
$Q$	g						

# Example



---

**Algorithm:** dijkstra( $G, s$ )

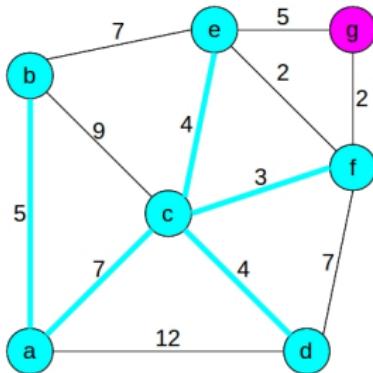
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e						
$Q$	g						

# Example



$u = e$

$v = g$

---

**Algorithm:** dijkstra( $G, s$ )

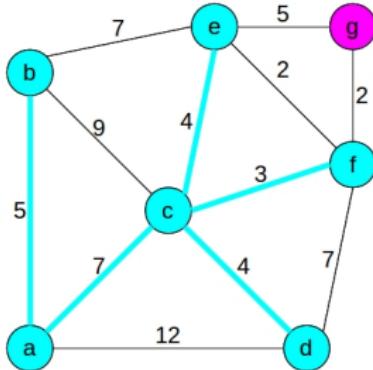
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e						
$Q$	g						

# Example



$u = e$

$v = g$

---

**Algorithm:** dijkstra( $G, s$ )

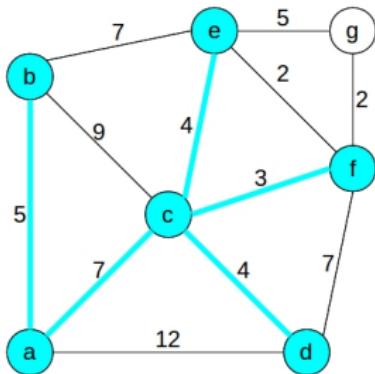
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e						
$Q$	g						

# Example



---

**Algorithm:** dijkstra( $G, s$ )

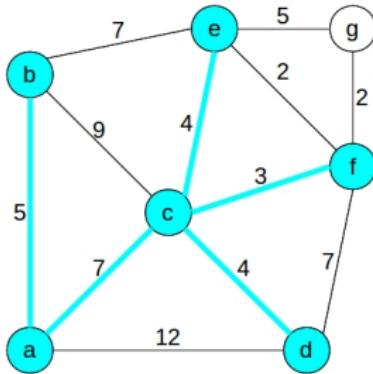
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e						
$Q$	g						

# Example

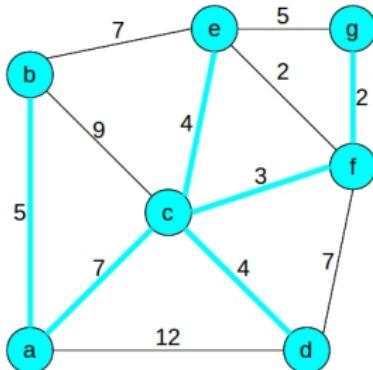


**Algorithm:** dijkstra( $G, s$ )

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e						
$Q$	g						

# Example



$u = g$

---

**Algorithm:** dijkstra( $G, s)$

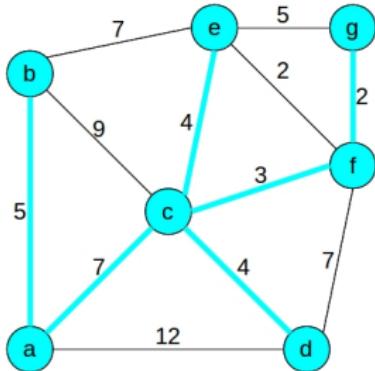
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e						
$Q$							

# Example



$u = g$

---

**Algorithm:** dijkstra( $G, s$ )

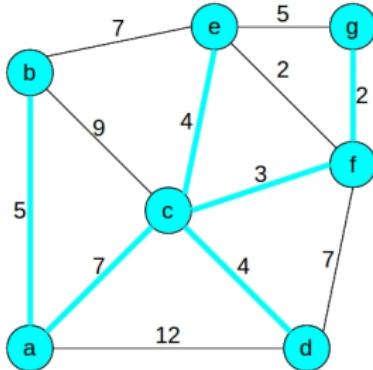
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e, <b>g</b>						
$Q$							

# Example



$u = g$

---

**Algorithm:** dijkstra( $G, s$ )

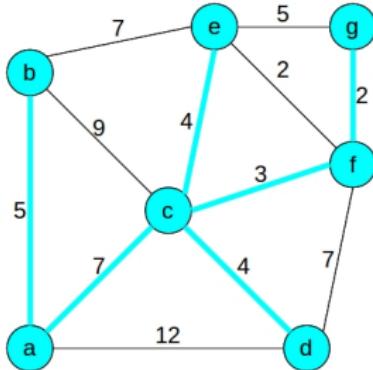
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e, g						
$Q$							

# Example



$u = g$

$v = e$

---

**Algorithm:** dijkstra( $G, s$ )

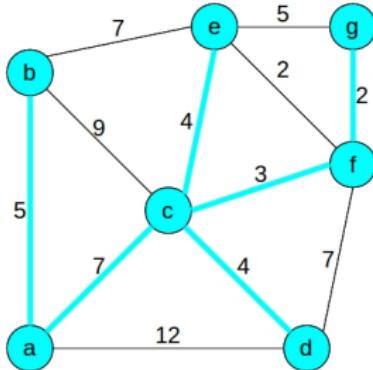
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e, g						
$Q$							

# Example



$u = g$

$v = e$

---

**Algorithm:** dijkstra( $G, s$ )

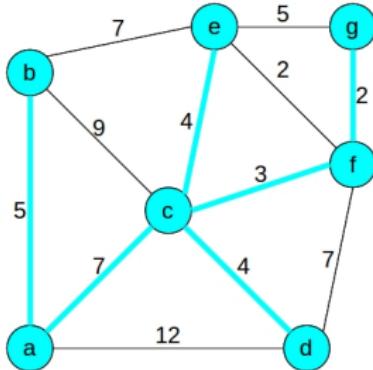
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e, g						
$Q$							

# Example



$u = g$

---

**Algorithm:** dijkstra( $G, s$ )

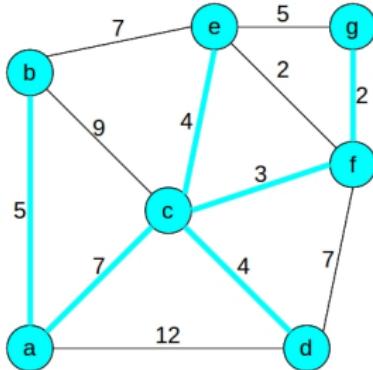
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e, g						
$Q$							

# Example



$u = g$

$v = f$

---

**Algorithm:** dijkstra( $G, s$ )

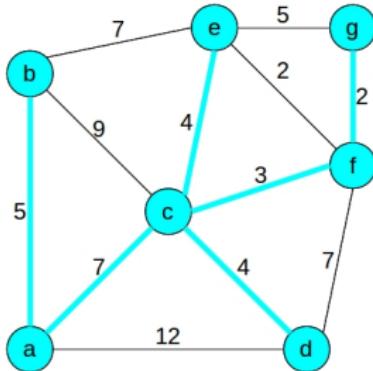
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e, g						
$Q$							

# Example



$u = g$

$v = f$

---

**Algorithm:** dijkstra( $G, s$ )

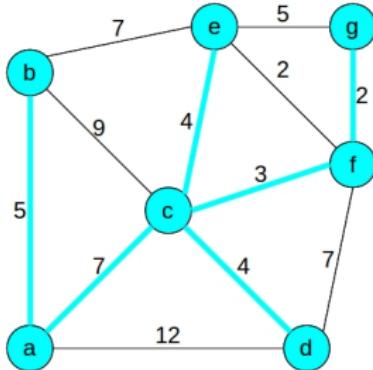
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e, g						
$Q$							

# Example



$u = g$

---

**Algorithm:** dijkstra( $G, s$ )

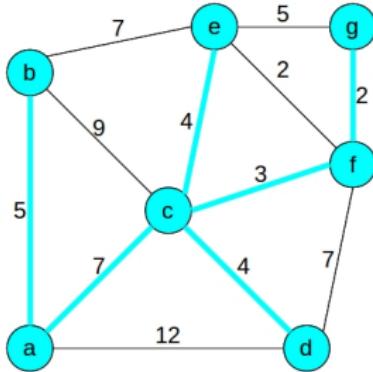
---

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

---

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e, g						
$Q$							

# Example



**Algorithm:** dijkstra( $G, s$ )

```
1 for each vertex  $v \in V$  do
2    $d[v] = \infty$ 
3    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 
7 while  $Q \neq \emptyset$  do
8    $u = \text{extractMin}(Q)$ 
9    $S = S \cup \{u\}$ 
10  for each vertex  $v$  adjacent to  $u$ 
11    do
12      if  $d[v] > d[u] + w(u, v)$ 
13        then
14           $d[v] = d[u] + w(u, v)$ 
15           $p[v] = u$ 
```

	[a]	[b]	[c]	[d]	[e]	[f]	[g]
$d[u]$	0	5	7	11	11	10	12
$p[u]$	NULL	a	a	c	c	c	f
$S$	a, b, c, f, d, e, g						
$Q$							

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$ 
3   |    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 

7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$ 
9   |    $S = S \cup \{u\}$ 

10  |   for each vertex  $v$  adjacent to  $u$  do
11    |     |   if  $d[v] > d[u] + w(u, v)$ 
12    |     |   then
13    |       |    $d[v] = d[u] + w(u, v)$ 
14    |       |    $p[v] = u$ 
```

---

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$ 
3   |    $p[v] = \text{NULL}$ 
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 

7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$ 
9   |    $S = S \cup \{u\}$ 

10  |   for each vertex  $v$  adjacent to  $u$  do
11    |   |   if  $d[v] > d[u] + w(u, v)$ 
12    |   |   then
13    |   |   |    $d[v] = d[u] + w(u, v)$ 
14    |   |   |    $p[v] = u$ 
```

---

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$  /* O(1) */
3   |    $p[v] = \text{NULL}$  /* O(1) */
4    $d[s] = 0$ 
5    $S = \emptyset$ 
6    $Q = V[G]$ 

7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$ 
9   |    $S = S \cup \{u\}$ 

10  |   for each vertex  $v$  adjacent to  $u$  do
11    |   |   if  $d[v] > d[u] + w(u, v)$ 
12    |   |   then
13    |   |   |    $d[v] = d[u] + w(u, v)$ 
14    |   |   |    $p[v] = u$ 
```

---

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$  /* O(1) */
3   |    $p[v] = \text{NULL}$  /* O(1) */
4    $d[s] = 0$  /* O(1) */
5    $S = \emptyset$  /* O(1) */
6    $Q = V[G]$ 

7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$ 
9   |    $S = S \cup \{u\}$ 

10  |   for each vertex  $v$  adjacent to  $u$  do
11    |   |   if  $d[v] > d[u] + w(u, v)$ 
12    |   |   then
13    |   |   |    $d[v] = d[u] + w(u, v)$ 
14    |   |   |    $p[v] = u$ 
```

---

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$  /* O(1) */
3   |    $p[v] = \text{NULL}$  /* O(1) */
4    $d[s] = 0$  /* O(1) */
5    $S = \emptyset$  /* O(1) */
6    $Q = V[G]$  /* O(n) */

7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$ 
9   |    $S = S \cup \{u\}$ 

10  for each vertex  $v$  adjacent to  $u$  do
11    |   if  $d[v] > d[u] + w(u, v)$ 
12    |   then
13    |     |    $d[v] = d[u] + w(u, v)$ 
14    |     |    $p[v] = u$ 
```

---

Recall the Build Heap algorithm from Chapter 4.

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$  /* O(1) */
3   |    $p[v] = \text{NULL}$  /* O(1) */
4    $d[s] = 0$  /* O(1) */
5    $S = \emptyset$  /* O(1) */
6    $Q = V[G]$  /* O(n) */
   /* O(n) iterations */
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$ 
9   |    $S = S \cup \{u\}$ 

10  |   for each vertex  $v$  adjacent to  $u$  do
11    |     |   if  $d[v] > d[u] + w(u, v)$ 
12    |     |   then
13    |       |    $d[v] = d[u] + w(u, v)$ 
14    |       |    $p[v] = u$ 
```

---

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$  /* O(1) */
3   |    $p[v] = \text{NULL}$  /* O(1) */
4    $d[s] = 0$  /* O(1) */
5    $S = \emptyset$  /* O(1) */
6    $Q = V[G]$  /* O(n) */
   /* O(n) iterations */
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$  /* O(?) (EXTRACT-MIN) */
9   |    $S = S \cup \{u\}$ 

10  for each vertex  $v$  adjacent to  $u$  do
11    |   if  $d[v] > d[u] + w(u, v)$ 
12    |   then
13    |     |    $d[v] = d[u] + w(u, v)$ 
14    |     |    $p[v] = u$ 
```

---

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$  /* O(1) */
3   |    $p[v] = \text{NULL}$  /* O(1) */
4    $d[s] = 0$  /* O(1) */
5    $S = \emptyset$  /* O(1) */
6    $Q = V[G]$  /* O(n) */
   /* O(n) iterations */
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$  /* O(?) (EXTRACT-MIN) */
9   |    $S = S \cup \{u\}$  /* O(1) */

10  for each vertex  $v$  adjacent to  $u$  do
11    |   if  $d[v] > d[u] + w(u, v)$ 
12    |   then
13    |     |    $d[v] = d[u] + w(u, v)$ 
14    |     |    $p[v] = u$ 
```

---

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$  /* O(1) */
3   |    $p[v] = \text{NULL}$  /* O(1) */
4    $d[s] = 0$  /* O(1) */
5    $S = \emptyset$  /* O(1) */
6    $Q = V[G]$  /* O(n) */
   /* O(n) iterations */
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$  /* O(?) (EXTRACT-MIN) */
9   |    $S = S \cup \{u\}$  /* O(1) */
   |   /* O(m) iterations total for the entire algorithm */
10  for each vertex  $v$  adjacent to  $u$  do
11    |   if  $d[v] > d[u] + w(u, v)$ 
12    |   then
13      |       |    $d[v] = d[u] + w(u, v)$ 
14      |       |    $p[v] = u$ 
```

---

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$  /* O(1) */
3   |    $p[v] = \text{NULL}$  /* O(1) */
4    $d[s] = 0$  /* O(1) */
5    $S = \emptyset$  /* O(1) */
6    $Q = V[G]$  /* O( $n$ ) */
   /* O(n) iterations */
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$  /* O(?) (EXTRACT-MIN) */
9   |    $S = S \cup \{u\}$  /* O(1) */
   |   /* O( $m$ ) iterations total for the entire algorithm */
10  for each vertex  $v$  adjacent to  $u$  do
11    |     if  $d[v] > d[u] + w(u, v)$  /* O(1) */
12    |       then
13      |         |    $d[v] = d[u] + w(u, v)$ 
14      |         |    $p[v] = u$ 
```

---

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$  /* O(1) */
3   |    $p[v] = \text{NULL}$  /* O(1) */
4    $d[s] = 0$  /* O(1) */
5    $S = \emptyset$  /* O(1) */
6    $Q = V[G]$  /* O(n) */
   /* O(n) iterations */
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$  /* O(?) (EXTRACT-MIN) */
9   |    $S = S \cup \{u\}$  /* O(1) */
   |   /* O(m) iterations total for the entire algorithm */
10  for each vertex  $v$  adjacent to  $u$  do
11    |   if  $d[v] > d[u] + w(u, v)$  /* O(1) */
12    |   then
13      |     |    $d[v] = d[u] + w(u, v)$  /* O(?) (DECREASE-KEY) */
14      |     |    $p[v] = u$ 
```

---

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$  /* O(1) */
3   |    $p[v] = \text{NULL}$  /* O(1) */
4    $d[s] = 0$  /* O(1) */
5    $S = \emptyset$  /* O(1) */
6    $Q = V[G]$  /* O( $n$ ) */
   /* O(n) iterations */
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$  /* O(?) (EXTRACT-MIN) */
9   |    $S = S \cup \{u\}$  /* O(1) */
   |   /* O( $m$ ) iterations total for the entire algorithm */
10  for each vertex  $v$  adjacent to  $u$  do
11    |     if  $d[v] > d[u] + w(u, v)$  /* O(1) */
12    |       then
13      |         |    $d[v] = d[u] + w(u, v)$  /* O(?) (DECREASE-KEY) */
14      |         |    $p[v] = u$  /* O(1) */
```

---

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$  /* O(1) */
3   |    $p[v] = \text{NULL}$  /* O(1) */
4    $d[s] = 0$  /* O(1) */
5    $S = \emptyset$  /* O(1) */
6    $Q = V[G]$  /* O(n) */
   /* O(n) iterations */
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$  /* O(?) (EXTRACT-MIN) */
9   |    $S = S \cup \{u\}$  /* O(1) */
   |   /* O(m) iterations total for the entire algorithm */
10  for each vertex  $v$  adjacent to  $u$  do
11    |   if  $d[v] > d[u] + w(u, v)$  /* O(1) */
12    |   then
13      |     |    $d[v] = d[u] + w(u, v)$  /* O(?) (DECREASE-KEY) */
14      |     |    $p[v] = u$  /* O(1) */
```

---

## Min-Heap

- EXTRACT-MIN is
- DECREASE-KEY is

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$  /* O(1) */
3   |    $p[v] = \text{NULL}$  /* O(1) */
4    $d[s] = 0$  /* O(1) */
5    $S = \emptyset$  /* O(1) */
6    $Q = V[G]$  /* O(n) */
   /* O(n) iterations */
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$  /* O(?) (EXTRACT-MIN) */
9   |    $S = S \cup \{u\}$  /* O(1) */
   |   /* O(m) iterations total for the entire algorithm */
10  for each vertex  $v$  adjacent to  $u$  do
11    |   if  $d[v] > d[u] + w(u, v)$  /* O(1) */
12    |   then
13      |     |    $d[v] = d[u] + w(u, v)$  /* O(?) (DECREASE-KEY) */
14      |     |    $p[v] = u$  /* O(1) */
```

---

## Min-Heap

- EXTRACT-MIN is  $O(\lg n)$
- DECREASE-KEY is

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$  /* O(1) */
3   |    $p[v] = \text{NULL}$  /* O(1) */
4    $d[s] = 0$  /* O(1) */
5    $S = \emptyset$  /* O(1) */
6    $Q = V[G]$  /* O(n) */
   /* O(n) iterations */
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$  /* O(?) (EXTRACT-MIN) */
9   |    $S = S \cup \{u\}$  /* O(1) */
   |   /* O(m) iterations total for the entire algorithm */
10  for each vertex  $v$  adjacent to  $u$  do
11    |   if  $d[v] > d[u] + w(u, v)$  /* O(1) */
12    |   then
13      |     |    $d[v] = d[u] + w(u, v)$  /* O(?) (DECREASE-KEY) */
14      |     |    $p[v] = u$  /* O(1) */
```

---

## Min-Heap

- EXTRACT-MIN is  $O(\lg n)$
- DECREASE-KEY is  $O(\lg n)$

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$  /* O(1) */
3   |    $p[v] = \text{NULL}$  /* O(1) */
4    $d[s] = 0$  /* O(1) */
5    $S = \emptyset$  /* O(1) */
6    $Q = V[G]$  /* O(n) */
   /* O(n) iterations */
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$  /* O(?) (EXTRACT-MIN) */
9   |    $S = S \cup \{u\}$  /* O(1) */
   |   /* O(m) iterations total for the entire algorithm */
10  for each vertex  $v$  adjacent to  $u$  do
11    |   if  $d[v] > d[u] + w(u, v)$  /* O(1) */
12    |   then
13      |     |    $d[v] = d[u] + w(u, v)$  /* O(?) (DECREASE-KEY) */
14      |     |    $p[v] = u$  /* O(1) */
```

---

## Min-Heap

- EXTRACT-MIN is  $O(\lg n)$
- DECREASE-KEY is  $O(\lg n)$

$$T(n) = O(n + n + n(\lg n) + m(\lg n))$$

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$  /* O(1) */
3   |    $p[v] = \text{NULL}$  /* O(1) */
4    $d[s] = 0$  /* O(1) */
5    $S = \emptyset$  /* O(1) */
6    $Q = V[G]$  /* O(n) */
   /* O(n) iterations */
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$  /* O(?) (EXTRACT-MIN) */
9   |    $S = S \cup \{u\}$  /* O(1) */
   |   /* O(m) iterations total for the entire algorithm */
10  |   for each vertex  $v$  adjacent to  $u$  do
11    |     |   if  $d[v] > d[u] + w(u, v)$  /* O(1) */
12    |     |   then
13    |       |   |    $d[v] = d[u] + w(u, v)$  /* O(?) (DECREASE-KEY) */
14    |       |   |    $p[v] = u$  /* O(1) */
```

---

## Min-Heap

- EXTRACT-MIN is  $O(\lg n)$
- DECREASE-KEY is  $O(\lg n)$

$$T(n) = O(n + n + n(\lg n) + m(\lg n)) = O(m \lg n)$$

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$  /* O(1) */
3   |    $p[v] = \text{NULL}$  /* O(1) */
4    $d[s] = 0$  /* O(1) */
5    $S = \emptyset$  /* O(1) */
6    $Q = V[G]$  /* O(n) */
   /* O(n) iterations */
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$  /* O(?) (EXTRACT-MIN) */
9   |    $S = S \cup \{u\}$  /* O(1) */
   |   /* O(m) iterations total for the entire algorithm */
10  for each vertex  $v$  adjacent to  $u$  do
11    |   if  $d[v] > d[u] + w(u, v)$  /* O(1) */
12    |   then
13      |     |    $d[v] = d[u] + w(u, v)$  /* O(?) (DECREASE-KEY) */
14      |     |    $p[v] = u$  /* O(1) */
```

---

## Fibonacci-Heap

- EXTRACT-MIN is
- DECREASE-KEY is

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$  /* O(1) */
3   |    $p[v] = \text{NULL}$  /* O(1) */
4    $d[s] = 0$  /* O(1) */
5    $S = \emptyset$  /* O(1) */
6    $Q = V[G]$  /* O(n) */
   /* O(n) iterations */
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$  /* O(?) (EXTRACT-MIN) */
9   |    $S = S \cup \{u\}$  /* O(1) */
   |   /* O(m) iterations total for the entire algorithm */
10  for each vertex  $v$  adjacent to  $u$  do
11    |   if  $d[v] > d[u] + w(u, v)$  /* O(1) */
12    |   then
13      |     |    $d[v] = d[u] + w(u, v)$  /* O(?) (DECREASE-KEY) */
14      |     |    $p[v] = u$  /* O(1) */
```

---

## Fibonacci-Heap

- EXTRACT-MIN is  $O(\lg n)$
- DECREASE-KEY is

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$  /* O(1) */
3   |    $p[v] = \text{NULL}$  /* O(1) */
4    $d[s] = 0$  /* O(1) */
5    $S = \emptyset$  /* O(1) */
6    $Q = V[G]$  /* O(n) */
   /* O(n) iterations */
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$  /* O(?) (EXTRACT-MIN) */
9   |    $S = S \cup \{u\}$  /* O(1) */
   |   /* O(m) iterations total for the entire algorithm */
10  for each vertex  $v$  adjacent to  $u$  do
11    |   if  $d[v] > d[u] + w(u, v)$  /* O(1) */
12    |   then
13      |     |    $d[v] = d[u] + w(u, v)$  /* O(?) (DECREASE-KEY) */
14      |     |    $p[v] = u$  /* O(1) */
```

---

## Fibonacci-Heap

- EXTRACT-MIN is  $O(\lg n)$
- DECREASE-KEY is  $O(1)$  amortized

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$  /* O(1) */
3   |    $p[v] = \text{NULL}$  /* O(1) */
4    $d[s] = 0$  /* O(1) */
5    $S = \emptyset$  /* O(1) */
6    $Q = V[G]$  /* O(n) */
   /* O(n) iterations */
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$  /* O(?) (EXTRACT-MIN) */
9   |    $S = S \cup \{u\}$  /* O(1) */
   |   /* O(m) iterations total for the entire algorithm */
10  |   for each vertex  $v$  adjacent to  $u$  do
11    |     |   if  $d[v] > d[u] + w(u, v)$  /* O(1) */
12    |     |   then
13    |       |   |    $d[v] = d[u] + w(u, v)$  /* O(?) (DECREASE-KEY) */
14    |       |   |    $p[v] = u$  /* O(1) */
```

---

## Fibonacci-Heap

- EXTRACT-MIN is  $O(\lg n)$
- DECREASE-KEY is  $O(1)$  amortized

$$T(n) = O(n + n + n(\lg n) + m(1))$$

# Analysis

---

**Algorithm:** dijkstra( $G, s$ )

---

```
/* O(n) iterations */
1 for each vertex  $v \in V$  do
2   |    $d[v] = \infty$  /* O(1) */
3   |    $p[v] = \text{NULL}$  /* O(1) */
4    $d[s] = 0$  /* O(1) */
5    $S = \emptyset$  /* O(1) */
6    $Q = V[G]$  /* O(n) */
   /* O(n) iterations */
7 while  $Q \neq \emptyset$  do
8   |    $u = \text{extractMin}(Q)$  /* O(?) (EXTRACT-MIN) */
9   |    $S = S \cup \{u\}$  /* O(1) */
   |   /* O(m) iterations total for the entire algorithm */
10  |   for each vertex  $v$  adjacent to  $u$  do
11    |     |   if  $d[v] > d[u] + w(u, v)$  /* O(1) */
12    |     |   then
13    |       |   |    $d[v] = d[u] + w(u, v)$  /* O(?) (DECREASE-KEY) */
14    |       |   |    $p[v] = u$  /* O(1) */
```

---

## Fibonacci-Heap

- EXTRACT-MIN is  $O(\lg n)$
- DECREASE-KEY is  $O(1)$  amortized

$$T(n) = O(n + n + n(\lg n) + m(1)) = O(n \lg n + m)$$

# All-Pairs Shortest Paths

- Many applications, such as finding the **diameter** of a graph, require finding the shortest path between all pairs of vertices.

# All-Pairs Shortest Paths

- Many applications, such as finding the **diameter** of a graph, require finding the shortest path between all pairs of vertices.
- We can run Dijkstra's algorithm  $n$  times (once from each possible start vertex) to solve all-pairs shortest path problem in  $O(n(n \log n + m))$ .

# All-Pairs Shortest Paths

- Many applications, such as finding the **diameter** of a graph, require finding the shortest path between all pairs of vertices.
- We can run Dijkstra's algorithm  $n$  times (once from each possible start vertex) to solve all-pairs shortest path problem in  $O(n(n \log n + m))$ .
- Can we do better?

# All-Pairs Shortest Paths

- Many applications, such as finding the **diameter** of a graph, require finding the shortest path between all pairs of vertices.
- We can run Dijkstra's algorithm  $n$  times (once from each possible start vertex) to solve all-pairs shortest path problem in  $O(n(n \log n + m))$ .
- Can we do better?

Not today :-).

# The Floyd-Warshall Algorithm

- Develop recurrence that yields a dynamic programming formulation.  
Number the vertices from 1 to  $n$ .

# The Floyd-Warshall Algorithm

- Develop recurrence that yields a dynamic programming formulation.  
Number the vertices from 1 to  $n$ .
- Let  $d[i,j]^k$  be the shortest path from  $i$  to  $j$  using only vertices from  $1, 2, \dots, k$  as possible intermediate vertices.

# The Floyd-Warshall Algorithm

- Develop recurrence that yields a dynamic programming formulation.  
Number the vertices from 1 to  $n$ .
- Let  $d[i, j]^k$  be the shortest path from  $i$  to  $j$  using only vertices from  $1, 2, \dots, k$  as possible intermediate vertices.

$$d[i, j]^k = \begin{cases} \min(d[i, j]^{k-1}, d[i, k]^{k-1} + d[k, j]^{k-1}) & k \geq 1 \\ w(i, j) & k = 0 \end{cases}$$

# The Floyd-Warshall Algorithm

- Develop recurrence that yields a dynamic programming formulation.  
Number the vertices from 1 to  $n$ .
- Let  $d[i, j]^k$  be the shortest path from  $i$  to  $j$  using only vertices from  $1, 2, \dots, k$  as possible intermediate vertices.

---

**Algorithm:** floydWarshall

---

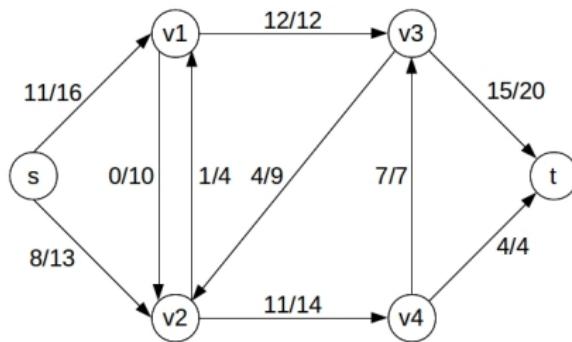
```
1  $d[i, j]^0 = w(i, j)$ 
2 for  $k = 1$  to  $n$  do
3   for  $i = 1$  to  $n$  do
4     for  $j = 1$  to  $n$  do
5        $d[i, j]^k = \min(d[i, j]^{k-1}, d[i, k]^{k-1} + d[k, j]^{k-1})$ 
```

---

# Network Flows

## Definition

A *Flow Network* is a directed graph,  $G = (V, E)$ , where the weight of each edge,  $(u, v) \in E$ , has a nonnegative capacity. One vertex is distinguished as the **source** vertex,  $s$ , and another vertex is distinguished as the **target** vertex,  $t$ .



## Definition

A *Flow Network* is a directed graph,  $G = (V, E)$ , where the weight of each edge,  $(u, v) \in E$ , has a nonnegative capacity. One vertex is distinguished as the **source** vertex,  $s$ , and another vertex is distinguished as the **target** vertex,  $t$ .

## Used to model

- Flow of current in an electrical circuit
- liquid flowing through pipes
- traffic flow through a network of roads
- packets in a computer network

# Network Flows

## Definition

A *Flow Network* is a directed graph,  $G = (V, E)$ , where the weight of each edge,  $(u, v) \in E$ , has a nonnegative capacity. One vertex is distinguished as the **source** vertex,  $s$ , and another vertex is distinguished as the **target** vertex,  $t$ .

Flow function  $f : V \times V \longrightarrow R$

## Definition

A *Flow Network* is a directed graph,  $G = (V, E)$ , where the weight of each edge,  $(u, v) \in E$ , has a nonnegative capacity. One vertex is distinguished as the **source** vertex,  $s$ , and another vertex is distinguished as the **target** vertex,  $t$ .

Flow function  $f : V \times V \longrightarrow R$

$f$  satisfies three (intuitive) properties

## Definition

A *Flow Network* is a directed graph,  $G = (V, E)$ , where the weight of each edge,  $(u, v) \in E$ , has a nonnegative capacity. One vertex is distinguished as the **source** vertex,  $s$ , and another vertex is distinguished as the **target** vertex,  $t$ .

Flow function  $f : V \times V \longrightarrow R$

$f$  satisfies three (intuitive) properties

- ① Capacity constraint:  $f(u, v) \leq c(u, v)$

## Definition

A *Flow Network* is a directed graph,  $G = (V, E)$ , where the weight of each edge,  $(u, v) \in E$ , has a nonnegative capacity. One vertex is distinguished as the **source** vertex,  $s$ , and another vertex is distinguished as the **target** vertex,  $t$ .

Flow function  $f : V \times V \longrightarrow R$

$f$  satisfies three (intuitive) properties

- ① Capacity constraint:  $f(u, v) \leq c(u, v)$
- ② Skew Symmetry:  $f(u, v) = -f(v, u)$

## Definition

A *Flow Network* is a directed graph,  $G = (V, E)$ , where the weight of each edge,  $(u, v) \in E$ , has a nonnegative capacity. One vertex is distinguished as the **source** vertex,  $s$ , and another vertex is distinguished as the **target** vertex,  $t$ .

Flow function  $f : V \times V \longrightarrow R$

$f$  satisfies three (intuitive) properties

- ① Capacity constraint:  $f(u, v) \leq c(u, v)$
- ② Skew Symmetry:  $f(u, v) = -f(v, u)$
- ③ Flow conservation:  
for any vertex  $u$  (other than  $s$  or  $t$ )  $\sum_v f(u, v) = 0$

# Network Flows

## Definition

A *Flow Network* is a directed graph,  $G = (V, E)$ , where the weight of each edge,  $(u, v) \in E$ , has a nonnegative capacity. One vertex is distinguished as the **source** vertex,  $s$ , and another vertex is distinguished as the **target** vertex,  $t$ .

## Maximum Flow Problem Definition

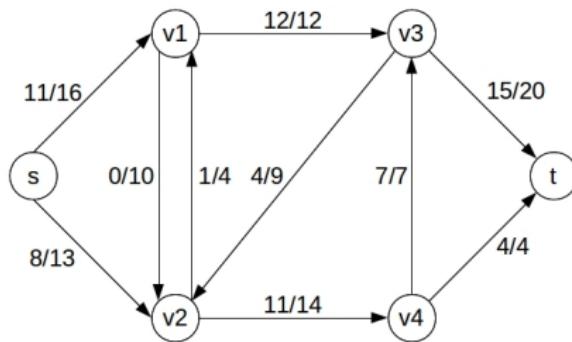
Input: A flow network,  $G$

Output: The flow function,  $f$ , of maximum value.

# Network Flows

## Definition

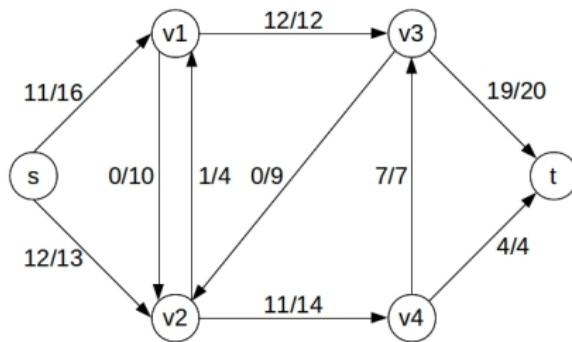
A *Flow Network* is a directed graph,  $G = (V, E)$ , where the weight of each edge,  $(u, v) \in E$ , has a nonnegative capacity. One vertex is distinguished as the **source** vertex,  $s$ , and another vertex is distinguished as the **target** vertex,  $t$ .



# Network Flows

## Definition

A *Flow Network* is a directed graph,  $G = (V, E)$ , where the weight of each edge,  $(u, v) \in E$ , has a nonnegative capacity. One vertex is distinguished as the **source** vertex,  $s$ , and another vertex is distinguished as the **target** vertex,  $t$ .



# Augmenting Paths

## Definition

An *augmenting path* is a sequence of edges from  $s$  to  $t$  along which flow can be increased

# Augmenting Paths

## Definition

An *augmenting path* is a sequence of edges from  $s$  to  $t$  along which flow can be increased

The flow through a network is optimal (maximum) if and only if it contains no augmenting path.

# Augmenting Paths

## Definition

An *augmenting path* is a sequence of edges from  $s$  to  $t$  along which flow can be increased

The flow through a network is optimal (maximum) if and only if it contains no augmenting path.

- Sometimes an augmenting path will traverse edges in **reverse direction!**

# Augmenting Paths

## Definition

An *augmenting path* is a sequence of edges from  $s$  to  $t$  along which flow can be increased

The flow through a network is optimal (maximum) if and only if it contains no augmenting path.

- Sometimes an augmenting path will traverse edges in **reverse direction!**
- BFS/DFS on flow graph will not find an augmenting path.

# Residual Graph/Network

## Definition

The *residual capacity* of edge  $(u, v)$  is defined as

$$r(u, v) = c(u, v) - f(u, v).$$

# Residual Graph/Network

## Definition

The *residual capacity* of edge  $(u, v)$  is defined as

$$r(u, v) = c(u, v) - f(u, v).$$

Set up **residual graph** as follows:

- Same vertices as in original graph
- Include edge  $(u, v)$  if  $r(u, v) > 0$

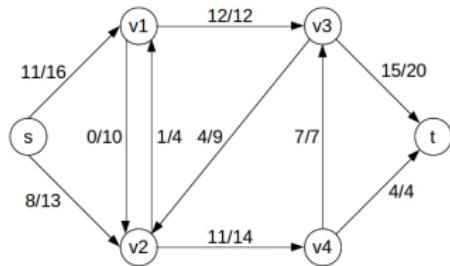
# Residual Graph/Network

## Definition

The *residual capacity* of edge  $(u, v)$  is defined as  
 $r(u, v) = c(u, v) - f(u, v)$ .

Set up **residual graph** as follows:

- Same vertices as in original graph
- Include edge  $(u, v)$  if  $r(u, v) > 0$



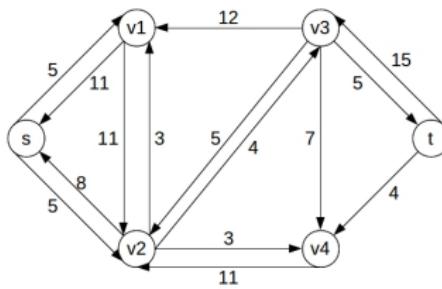
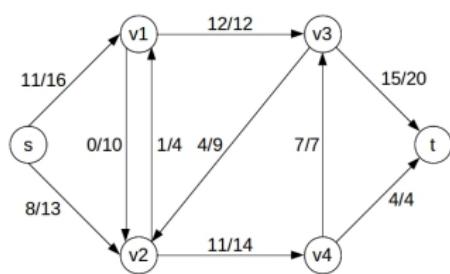
# Residual Graph/Network

## Definition

The *residual capacity* of edge  $(u, v)$  is defined as  
 $r(u, v) = c(u, v) - f(u, v)$ .

Set up **residual graph** as follows:

- Same vertices as in original graph
- Include edge  $(u, v)$  if  $r(u, v) > 0$



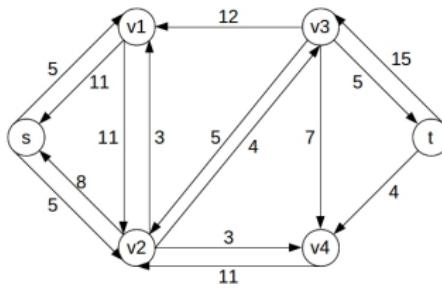
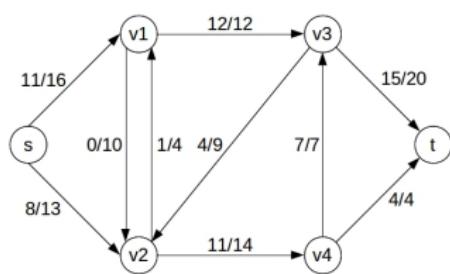
# Residual Graph/Network

## Definition

The *residual capacity* of edge  $(u, v)$  is defined as  
 $r(u, v) = c(u, v) - f(u, v)$ .

Set up **residual graph** as follows:

- Same vertices as in original graph
- Include edge  $(u, v)$  if  $r(u, v) > 0$

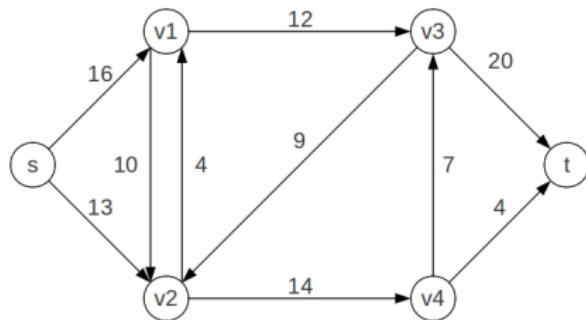


**NOW** use BFS/DFS on residual graph to find augmenting path!



# Ford-Fulkerson Algorithm

## The Network Flow



---

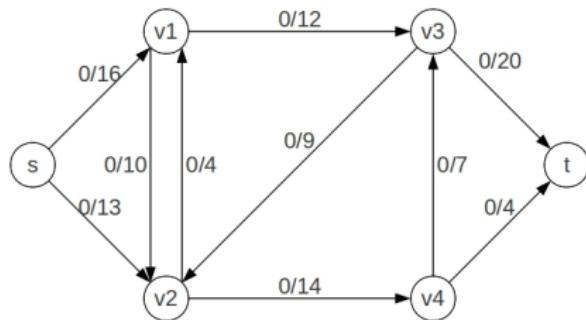
### Algorithm: fordFulkerson

---

- 1 Initialize  $f$  to 0 for each edge
  - 2 **while** there exists an augmenting path  $p$  **do**
  - 3   | augment flow,  $f$ , along  $p$
-

# Ford-Fulkerson Algorithm

## The Network Flow



---

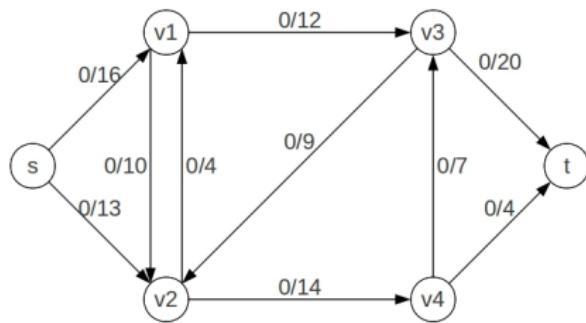
### Algorithm: fordFulkerson

---

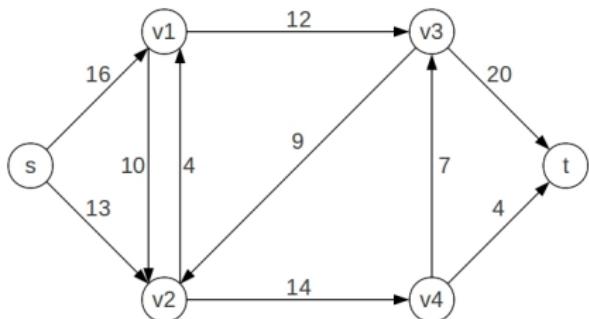
- 1 Initialize  $f$  to 0 for each edge
  - 2 while there exists an augmenting path  $p$  do
  - 3   | augment flow,  $f$ , along  $p$
-

# Ford-Fulkerson Algorithm

The Network Flow



The Residual Graph



---

**Algorithm: fordFulkerson**

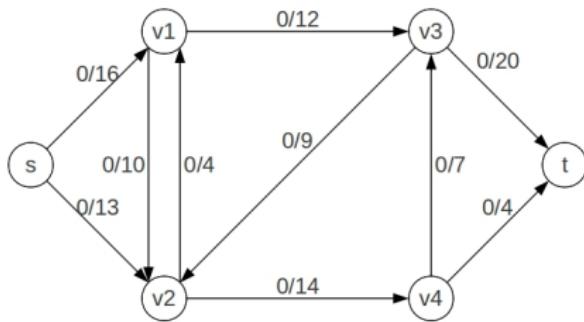
---

- 1 Initialize  $f$  to 0 for each edge
  - 2 **while there exists an augmenting path  $p$  do**
  - 3   | augment flow,  $f$ , along  $p$
- 

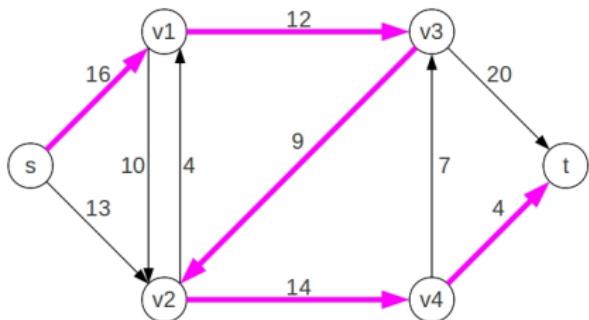
*Build the residual graph*

# Ford-Fulkerson Algorithm

The Network Flow



The Residual Graph



---

**Algorithm: fordFulkerson**

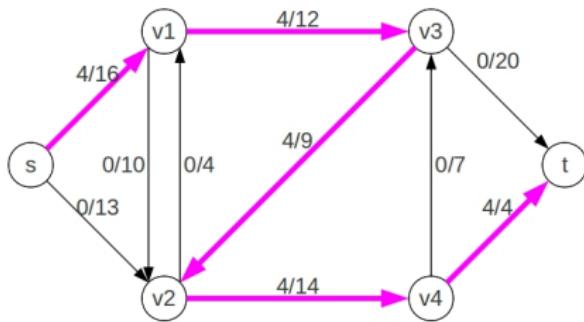
---

- 1 Initialize  $f$  to 0 for each edge
  - 2 **while** there exists an augmenting path  $p$  **do**
  - 3   | augment flow,  $f$ , along  $p$
- 

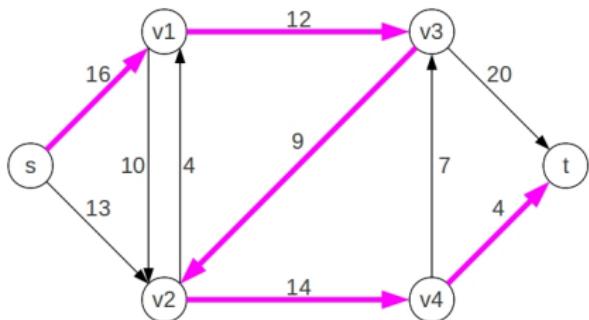
*Use a traversal algorithm to look for an augmenting path on the residual graph*

# Ford-Fulkerson Algorithm

The Network Flow



The Residual Graph



---

**Algorithm: fordFulkerson**

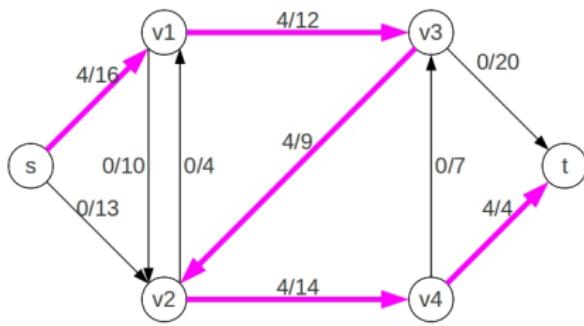
---

- 1 Initialize  $f$  to 0 for each edge
  - 2 **while** there exists an augmenting path  $p$  **do**
  - 3   | augment flow,  $f$ , along  $p$
- 

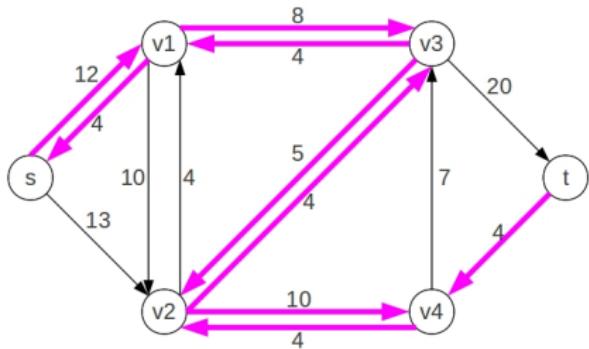
*Update the network flow graph*

# Ford-Fulkerson Algorithm

## The Network Flow



## The Residual Graph



---

### Algorithm: fordFulkerson

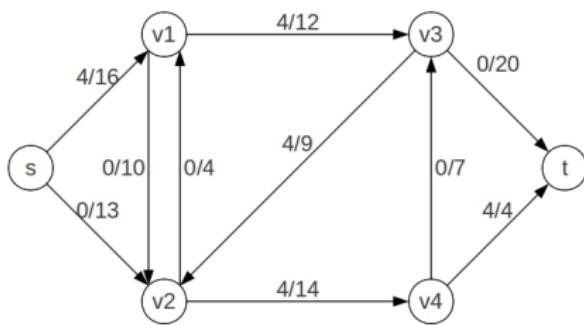
---

- 1 Initialize  $f$  to 0 for each edge
  - 2 **while** there exists an augmenting path  $p$  **do**
  - 3   | augment flow,  $f$ , along  $p$
- 

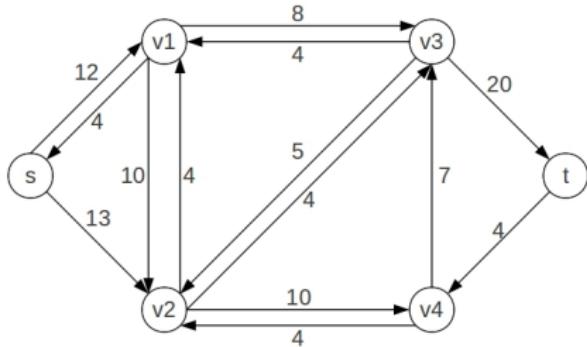
*Update the residual graph*

# Ford-Fulkerson Algorithm

The Network Flow



The Residual Graph



---

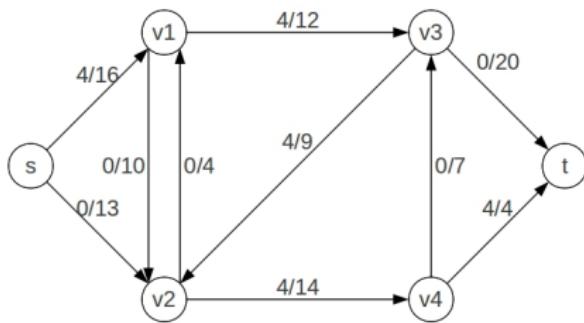
**Algorithm: fordFulkerson**

---

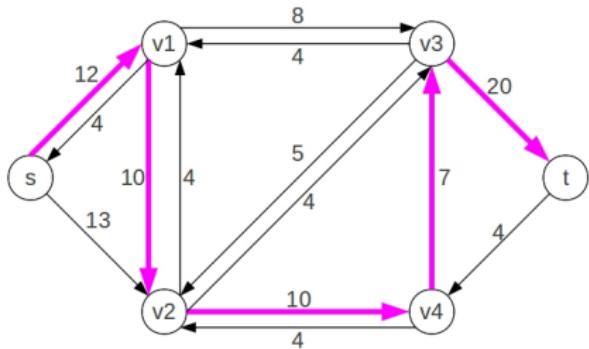
- 1 Initialize  $f$  to 0 for each edge
  - 2 **while there exists an augmenting path  $p$  do**
  - 3   | augment flow,  $f$ , along  $p$
-

# Ford-Fulkerson Algorithm

## The Network Flow



## The Residual Graph



---

### Algorithm: fordFulkerson

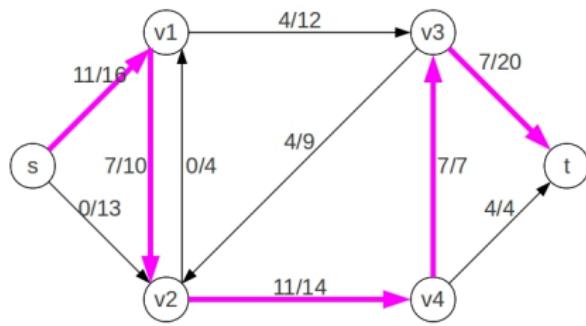
---

- 1 Initialize  $f$  to 0 for each edge
  - 2 **while** there exists an augmenting path  $p$  **do**
  - 3   | augment flow,  $f$ , along  $p$
- 

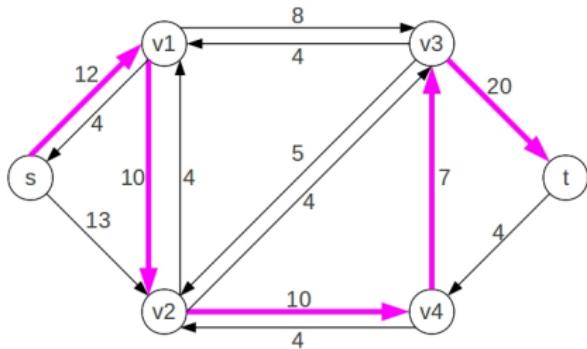
*Use a traversal algorithm to look for an augmenting path on the residual graph*

# Ford-Fulkerson Algorithm

The Network Flow



The Residual Graph



---

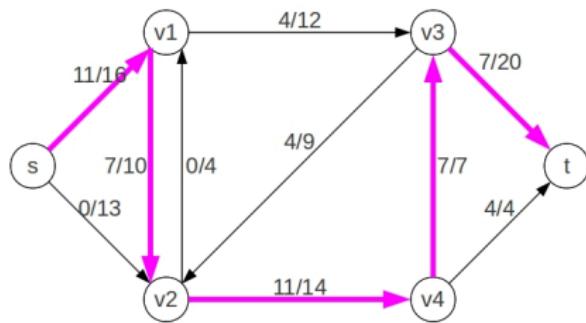
**Algorithm:** `fordFulkerson`

- 1 Initialize  $f$  to 0 for each edge
  - 2 **while** there exists an augmenting path  $p$  **do**
  - 3   | augment flow,  $f$ , along  $p$
- 

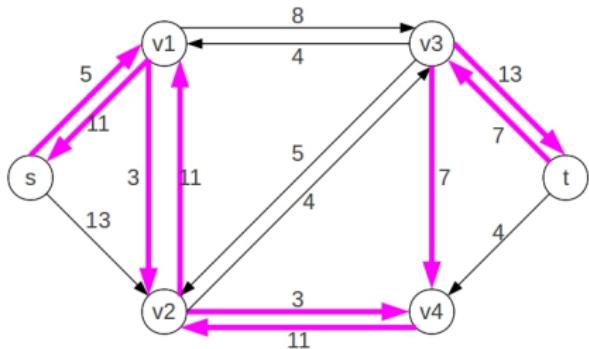
*Update the network flow graph*

# Ford-Fulkerson Algorithm

The Network Flow



The Residual Graph



---

**Algorithm: fordFulkerson**

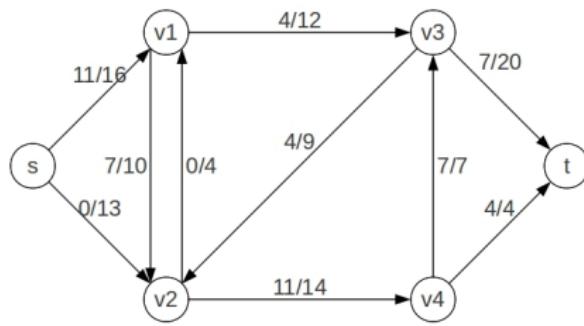
---

- 1 Initialize  $f$  to 0 for each edge
  - 2 **while** there exists an augmenting path  $p$  **do**
  - 3   | augment flow,  $f$ , along  $p$
- 

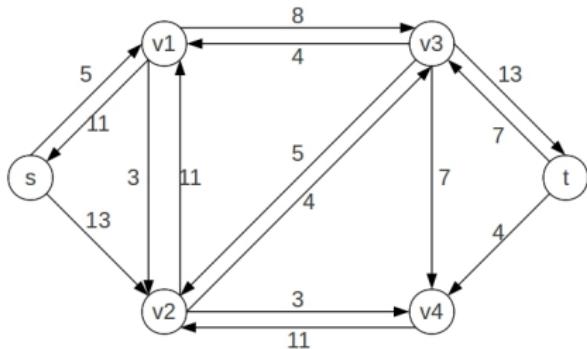
*Update the residual graph*

# Ford-Fulkerson Algorithm

The Network Flow



The Residual Graph



---

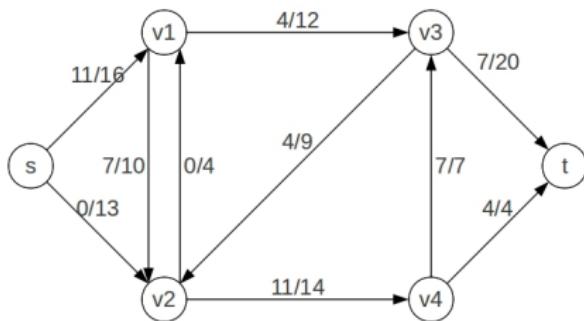
**Algorithm: fordFulkerson**

---

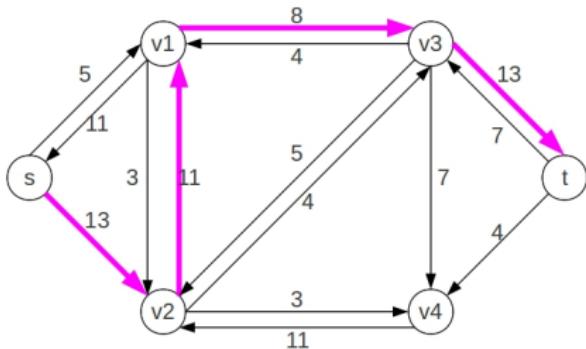
- 1 Initialize  $f$  to 0 for each edge
  - 2 **while there exists an augmenting path  $p$  do**
  - 3   | augment flow,  $f$ , along  $p$
-

# Ford-Fulkerson Algorithm

## The Network Flow



## The Residual Graph



---

### Algorithm: fordFulkerson

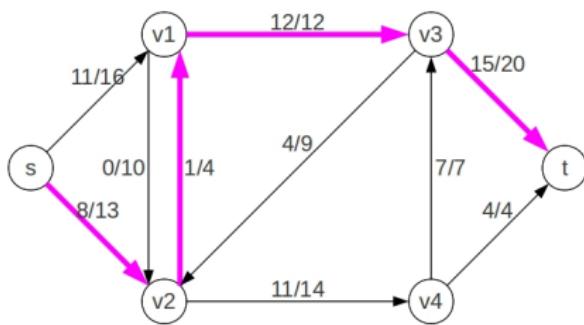
---

- 1 Initialize  $f$  to 0 for each edge
  - 2 **while** there exists an augmenting path  $p$  **do**
  - 3   | augment flow,  $f$ , along  $p$
- 

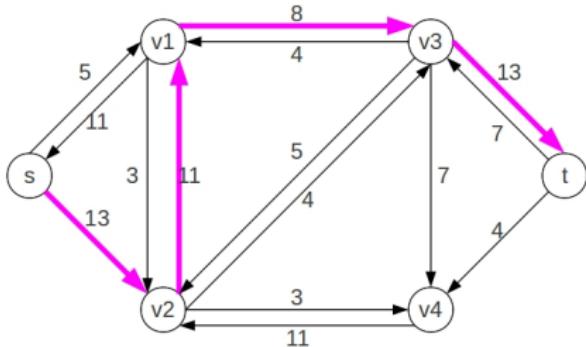
*Use a traversal algorithm to look for an augmenting path on the residual graph*

# Ford-Fulkerson Algorithm

The Network Flow



The Residual Graph



---

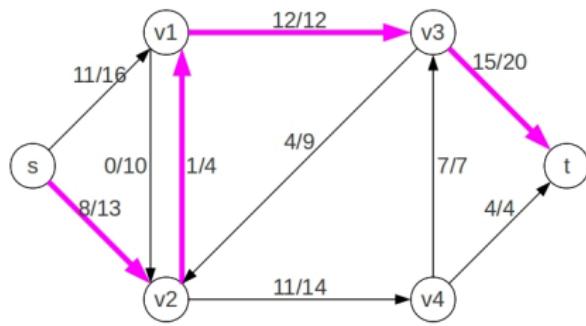
**Algorithm:** `fordFulkerson`

- 1 Initialize  $f$  to 0 for each edge
  - 2 **while** there exists an augmenting path  $p$  **do**
  - 3   | augment flow,  $f$ , along  $p$
- 

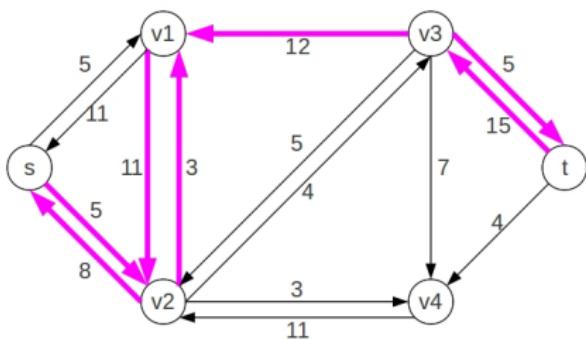
*Update the network flow graph*

# Ford-Fulkerson Algorithm

The Network Flow



The Residual Graph



---

**Algorithm: fordFulkerson**

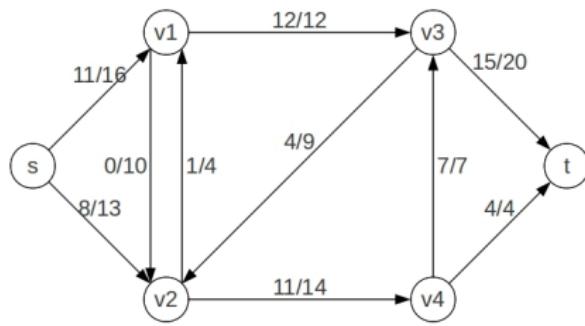
---

- 1 Initialize  $f$  to 0 for each edge
  - 2 **while** there exists an augmenting path  $p$  **do**
  - 3   | augment flow,  $f$ , along  $p$
- 

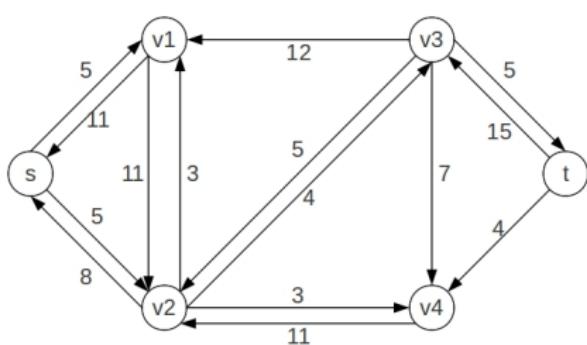
*Update the residual graph*

# Ford-Fulkerson Algorithm

The Network Flow



The Residual Graph



---

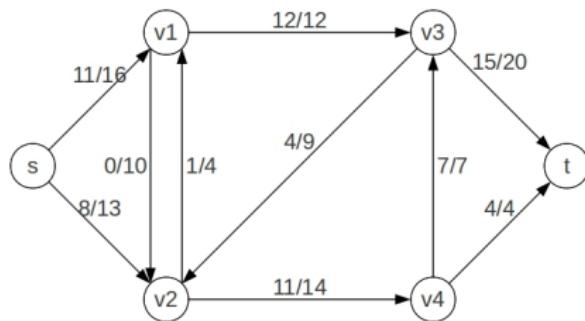
**Algorithm: fordFulkerson**

---

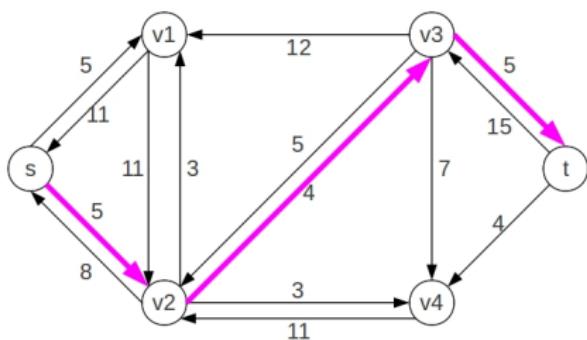
- 1 Initialize  $f$  to 0 for each edge
  - 2 **while there exists an augmenting path  $p$  do**
  - 3   | augment flow,  $f$ , along  $p$
-

# Ford-Fulkerson Algorithm

## The Network Flow



## The Residual Graph



---

### Algorithm: fordFulkerson

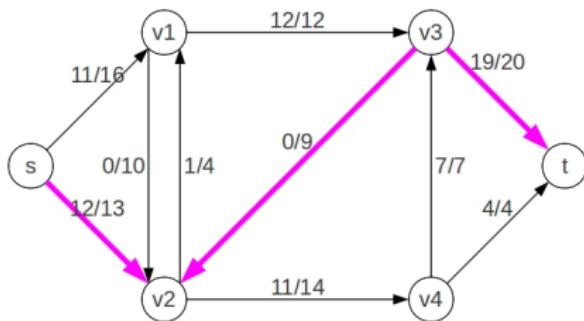
---

- 1 Initialize  $f$  to 0 for each edge
  - 2 **while** there exists an augmenting path  $p$  **do**
  - 3   | augment flow,  $f$ , along  $p$
- 

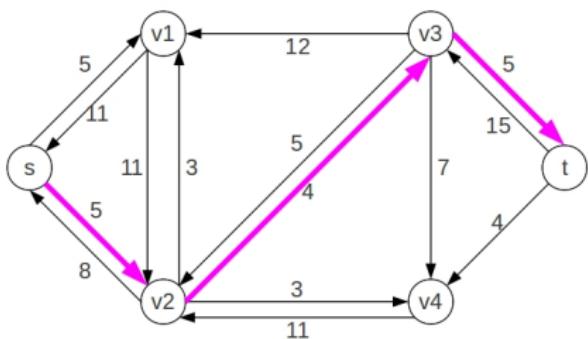
*Use a traversal algorithm to look for an augmenting path on the residual graph*

# Ford-Fulkerson Algorithm

The Network Flow



The Residual Graph



---

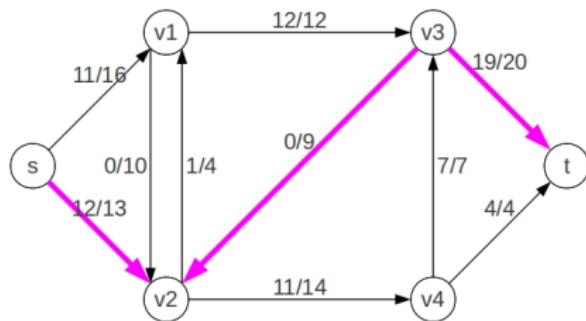
**Algorithm:** `fordFulkerson`

- 1 Initialize  $f$  to 0 for each edge
  - 2 **while** there exists an augmenting path  $p$  **do**
  - 3   | augment flow,  $f$ , along  $p$
- 

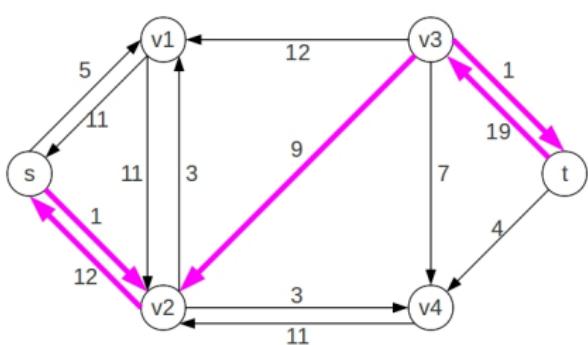
*Update the network flow graph*

# Ford-Fulkerson Algorithm

The Network Flow



The Residual Graph



---

**Algorithm: fordFulkerson**

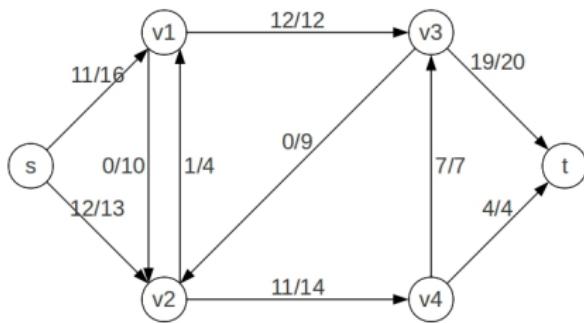
---

- 1 Initialize  $f$  to 0 for each edge
  - 2 **while** there exists an augmenting path  $p$  **do**
  - 3   | augment flow,  $f$ , along  $p$
- 

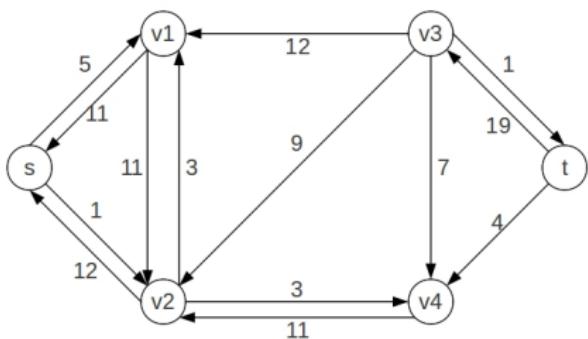
*Update the residual graph*

# Ford-Fulkerson Algorithm

The Network Flow



The Residual Graph



---

**Algorithm: fordFulkerson**

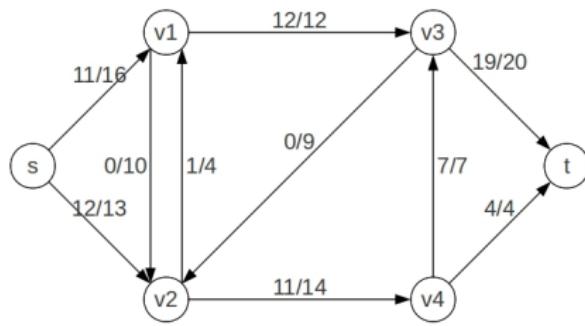
---

- 1 Initialize  $f$  to 0 for each edge
  - 2 **while** there exists an augmenting path  $p$  **do**
  - 3   | augment flow,  $f$ , along  $p$
- 

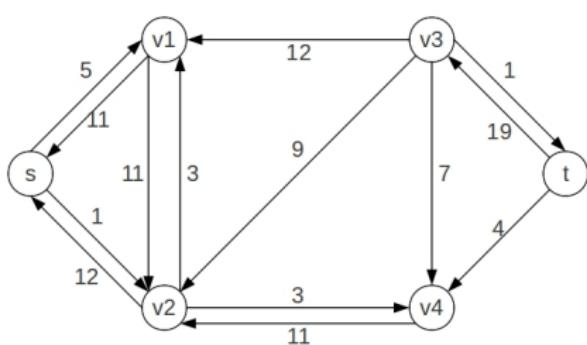
No augmenting path exists so the algorithm is finished

# Ford-Fulkerson Algorithm

The Network Flow



The Residual Graph



---

**Algorithm:** `fordFulkerson`

- 1 Initialize  $f$  to 0 for each edge
  - 2 **while** there exists an augmenting path  $p$  **do**
  - 3   | augment flow,  $f$ , along  $p$
-

# Cuts

- $\text{Cut}(S, T)$  of a flow network is a partition of  $V$  into  $S$  and  $T = V - S$  (where  $s$  is in  $S$ ,  $t$  is in  $T$ ).

# Cuts

- $\text{Cut}(S, T)$  of a flow network is a partition of  $V$  into  $S$  and  $T = V - S$  (where  $s$  is in  $S$ ,  $t$  is in  $T$ ).
- Define **net flow**  $f(S, T)$  across a cut

# Cuts

- $\text{Cut}(S, T)$  of a flow network is a partition of  $V$  into  $S$  and  $T = V - S$  (where  $s$  is in  $S$ ,  $t$  is in  $T$ ).
- Define **net flow**  $f(S, T)$  across a cut
- Define **capacity of cut**  $c(S, T)$

# Cuts

- $\text{Cut}(S, T)$  of a flow network is a partition of  $V$  into  $S$  and  $T = V - S$  (where  $s$  is in  $S$ ,  $t$  is in  $T$ ).
- Define **net flow**  $f(S, T)$  across a cut
- Define **capacity of cut**  $c(S, T)$
- Max flow cannot exceed capacity of the cut

# Max-Flow Min-Cut Theorem

The following statements are equivalent

- ①  $f$  is a maximum flow in  $G$
- ② The residual network (corresponding to  $f$ ) contains no augmenting paths
- ③  $|f| = c(S, T)$  for some cut  $(S, T)$  of  $G$

# Complexity

Product of

- # of flow augmentations performed by FF.
  - Each augmentation increases flow by at least 1.
  - So, at most  $|f|$  augmentations.
- Run time of one flow augmentation is  $O(n + m) = O(m)$ .

# Edmonds-Karp Algorithm

- Identical to Ford-Fulkerson except

# Edmonds-Karp Algorithm

- Identical to Ford-Fulkerson except
- Require augmenting path to be computed with a BF traversal.

# Edmonds-Karp Algorithm

- Identical to Ford-Fulkerson except
- Require augmenting path to be computed with a BF traversal.
- So, augmenting path is a shortest path from  $s$  to  $t$ .

# Edmonds-Karp Algorithm

- Identical to Ford-Fulkerson except
- Require augmenting path to be computed with a BF traversal.
- So, augmenting path is a shortest path from  $s$  to  $t$ .
- Theorem: total number of flow augmentations performed by the E-K algorithm is  $O(nm)$ .

# Edmonds-Karp Algorithm

- Identical to Ford-Fulkerson except
- Require augmenting path to be computed with a BF traversal.
- So, augmenting path is a shortest path from  $s$  to  $t$ .
- Theorem: total number of flow augmentations performed by the E-K algorithm is  $O(nm)$ .
- Overall run time is  $O(nm^2)$

# Maximum Matching in Bipartite Graphs

**Bipartite:** Connected undirected graph  $G$  where

- ① Vertices of  $G$  can be split into two sets  $X$  and  $Y$  so that ...
- ② Every edge of  $G$  has one end point in  $X$  and the other in  $Y$ .

# Maximum Matching in Bipartite Graphs

**Bipartite:** Connected undirected graph  $G$  where

- ① Vertices of  $G$  can be split into two sets  $X$  and  $Y$  so that ...
- ② Every edge of  $G$  has one end point in  $X$  and the other in  $Y$ .

**Matching**  $M$  in  $G$  is a set of edges that have no end points in common.

# Maximum Matching in Bipartite Graphs

**Bipartite:** Connected undirected graph  $G$  where

- ① Vertices of  $G$  can be split into two sets  $X$  and  $Y$  so that ...
- ② Every edge of  $G$  has one end point in  $X$  and the other in  $Y$ .

**Matching  $M$**  in  $G$  is a set of edges that have no end points in common.

**Maximum Bipartite Matching Problem:** Find a matching with the greatest number of edges.

# Application

- $X$  is a set of college courses.
- $Y$  is a set of classrooms.
- An edge joins  $x$  in  $X$  and  $y$  in  $Y$  if course  $x$  can be taught in classroom  $y$  (based on audio-visual needs, enrollment, etc.).

# Reduction to the Maximum Flow Problem

- ① Set up flow network  $H$  corresponding to bipartite graph  $G$ .
- ② Run max-flow on  $H$ .
- ③ Use results of max-flow on  $H$  to construct matching in  $G$ .

## Set up Flow Network $H$ from $G$

- $H$  contains all the vertices of  $G$  plus a new source vertex  $s$  and a new sink vertex  $t$ .

## Set up Flow Network $H$ from $G$

- $H$  contains all the vertices of  $G$  plus a new source vertex  $s$  and a new sink vertex  $t$ .
- Add every edge of  $G$  to  $H$  but direct it so that it goes from the vertex in  $X$  to the vertex in  $Y$ .

## Set up Flow Network $H$ from $G$

- $H$  contains all the vertices of  $G$  plus a new source vertex  $s$  and a new sink vertex  $t$ .
- Add every edge of  $G$  to  $H$  but direct it so that it goes from the vertex in  $X$  to the vertex in  $Y$ .
- Add a directed edge from  $s$  to each vertex in  $X$ .

## Set up Flow Network $H$ from $G$

- $H$  contains all the vertices of  $G$  plus a new source vertex  $s$  and a new sink vertex  $t$ .
- Add every edge of  $G$  to  $H$  but direct it so that it goes from the vertex in  $X$  to the vertex in  $Y$ .
- Add a directed edge from  $s$  to each vertex in  $X$ .
- Add a directed edge from each vertex in  $Y$  to  $t$ .

## Set up Flow Network $H$ from $G$

- $H$  contains all the vertices of  $G$  plus a new source vertex  $s$  and a new sink vertex  $t$ .
- Add every edge of  $G$  to  $H$  but direct it so that it goes from the vertex in  $X$  to the vertex in  $Y$ .
- Add a directed edge from  $s$  to each vertex in  $X$ .
- Add a directed edge from each vertex in  $Y$  to  $t$ .
- Assign a capacity of 1 to each edge.

# Solve the maximum matching problem.

- Run the max-flow algorithm on  $H$ .

## Solve the maximum matching problem.

- Run the max-flow algorithm on  $H$ .
- The flow in each edge is either 0 or 1 (why?).

## Solve the maximum matching problem.

- Run the max-flow algorithm on  $H$ .
- The flow in each edge is either 0 or 1 (why?).
- For each vertex in  $X$ , there is at most one outgoing edge with a flow of 1 (why?).

## Solve the maximum matching problem.

- Run the max-flow algorithm on  $H$ .
- The flow in each edge is either 0 or 1 (why?).
- For each vertex in  $X$ , there is at most one outgoing edge with a flow of 1 (why?).
- Similarly, for each vertex in  $Y$ , there is at most one incoming edge with a flow of 1.

## Solve the maximum matching problem.

- Run the max-flow algorithm on  $H$ .
- The flow in each edge is either 0 or 1 (why?).
- For each vertex in  $X$ , there is at most one outgoing edge with a flow of 1 (why?).
- Similarly, for each vertex in  $Y$ , there is at most one incoming edge with a flow of 1.
- Matching = set of all  $(x, y)$ s with a flow of 1 from  $x$  to  $y$ .

## Solve the maximum matching problem.

- Run the max-flow algorithm on  $H$ .
- The flow in each edge is either 0 or 1 (why?).
- For each vertex in  $X$ , there is at most one outgoing edge with a flow of 1 (why?).
- Similarly, for each vertex in  $Y$ , there is at most one incoming edge with a flow of 1.
- Matching = set of all  $(x, y)$ s with a flow of 1 from  $x$  to  $y$ .
- Max flow implies max matching!

# Complexity Analysis

- Construct  $H$  from  $G$ :  $O(n + m)$ .

# Complexity Analysis

- Construct  $H$  from  $G$ :  $O(n + m)$ .
- Run FF algorithm:  $(|f| \times m)$ .

# Complexity Analysis

- Construct  $H$  from  $G$ :  $O(n + m)$ .
- Run FF algorithm:  $(|f| \times m)$ .
- But  $|f| = |M| \leq n/2$ .

# Complexity Analysis

- Construct  $H$  from  $G$ :  $O(n + m)$ .
- Run FF algorithm:  $(|f| \times m)$ .
- But  $|f| = |M| \leq n/2$ .
- So  $O(nm)$ .