Data structure and Algorithms

Graph

Thanh-Hai Tran

Electronics and Computer Engineering
School of Electronics and Telecommunications

Hanoi University of Science and Technology
1 Dai Co Viet - Hanoi - Vietnam

Outline

- **Shortest paths**
  - Introduction
  - Applications
  - Dijkstra’s algorithm
- **Minimum spanning trees**
  - Introduction
  - Applications
  - Kruskal algorithm

Weighted graphs

- Let G be a weighted path, the length of a path is the sum of the weighted of the edges of P.
- If a path $P = ((v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k))$

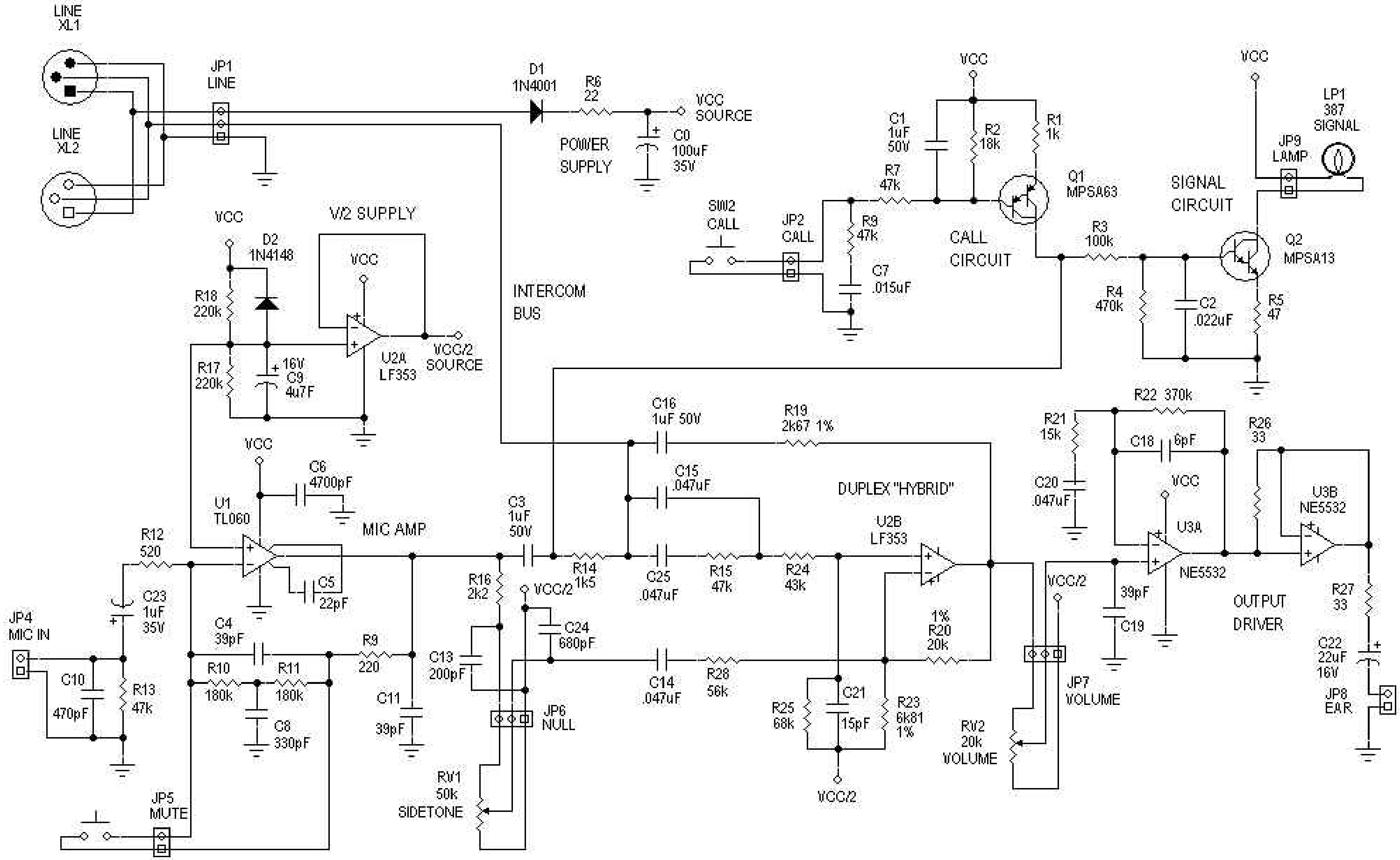
$$w(P) = \sum_{i=0}^{k-1} w((v_i, v_{i+1}))$$

- A distance from a vertex v to a vertex u in G d(v,u) is the length of the minimum length path (shortest path) from v to u if the path exists

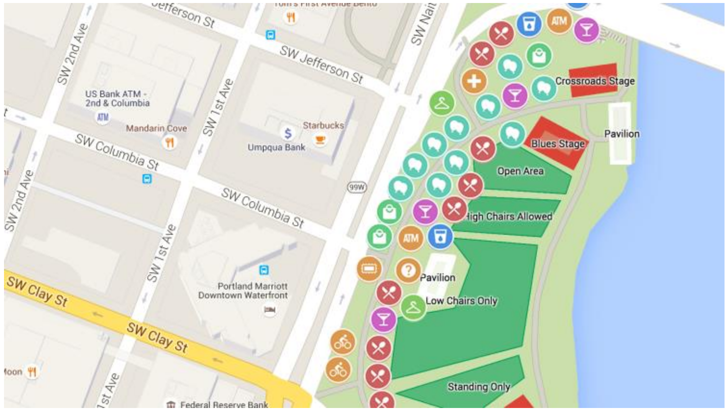
Shortest Path

- Given a weighted directed graph, one common problem is **finding the shortest path between two given vertices**
- Recall that in a weighted graph, the *length* of a path is the sum of the weights of each of the edges in that path
- Applications:
  - One application is circuit design: the time it takes for a change in input to affect an output depends on the shortest path

Applications



Applications



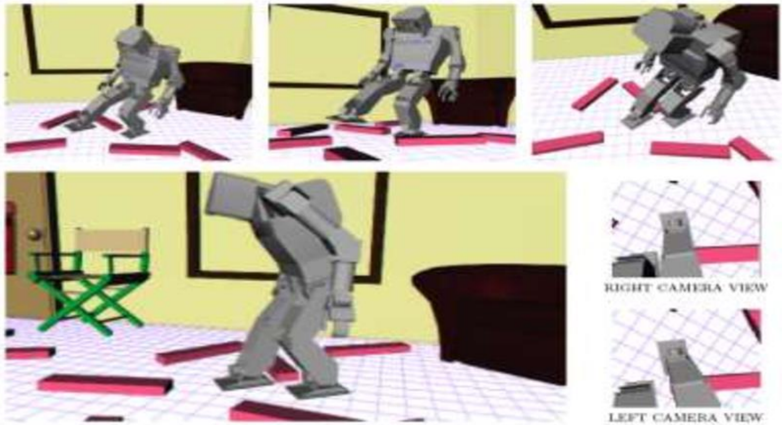
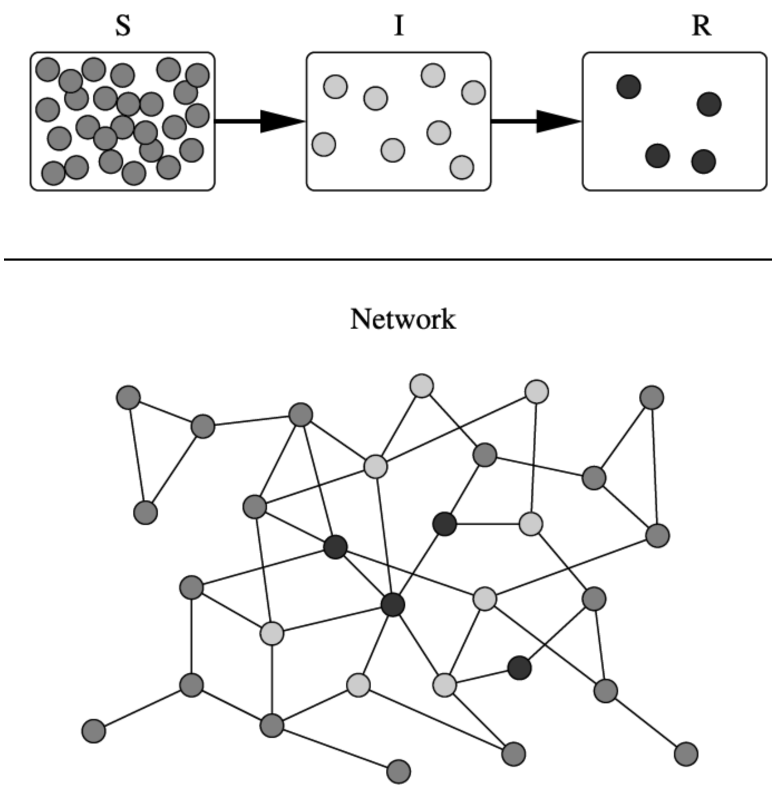


Fig. 1: Path Finding Examples

Applications of Dijkstra's Algorithm

- **Epidemiology:**
  - networks to model the spread of infectious diseases and design prevention and response strategies.
  - Vertices represent individuals, and edges their possible contacts. It is useful to calculate how a particular individual is connected to others.
  - Knowing the shortest path lengths to other individuals can be a relevant indicator of the potential of a particular individual to infect others.

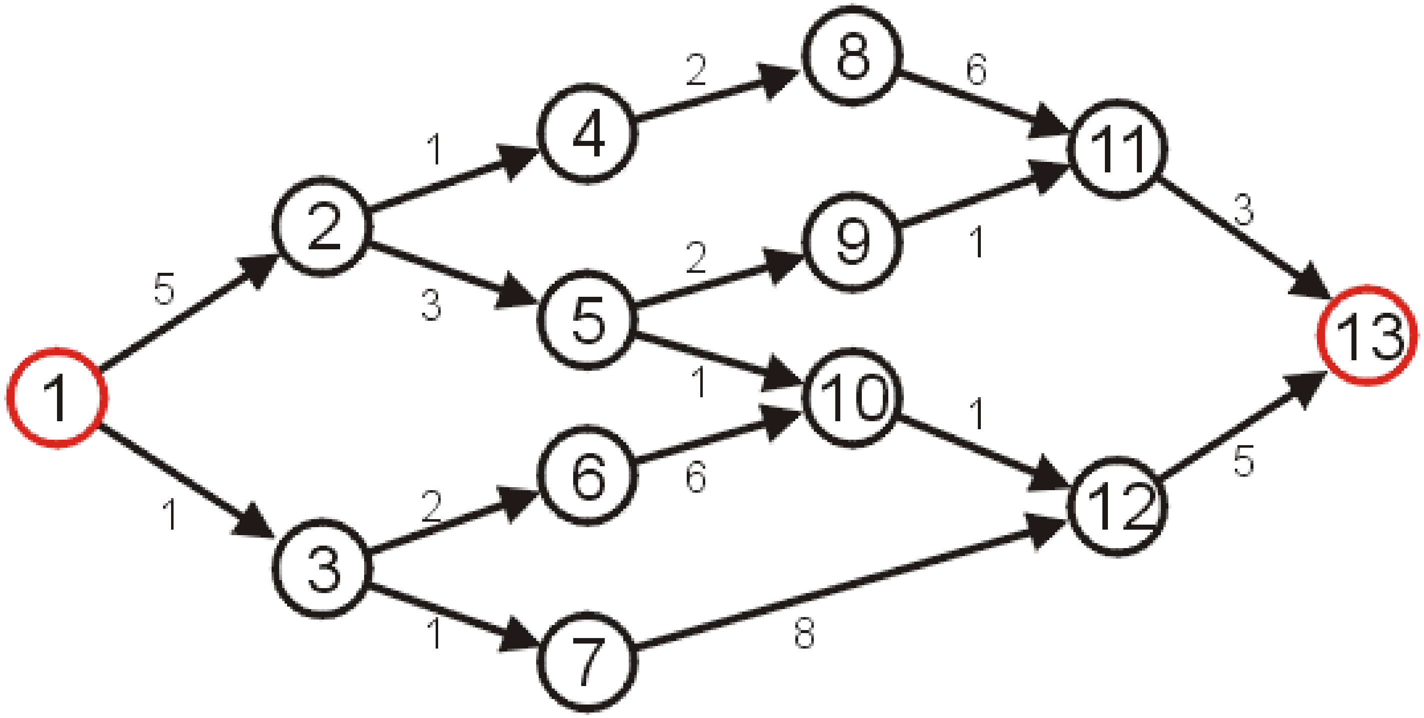


Shortest paths

- BFS (breadth-first search) can be used to find a shortest path from some starting vertex to every other vertex in a connected graph
- This approach makes sense in cases where each edge is as good as any other,
- But there are many situations where this approach is not appropriate
- **Examples:**
  - Computer network: edges can be slow phone-line connections, high-speed, fiber-optic connections
  - Intercity road network: inter-city distances are different
  - They are connected, weighted graphs

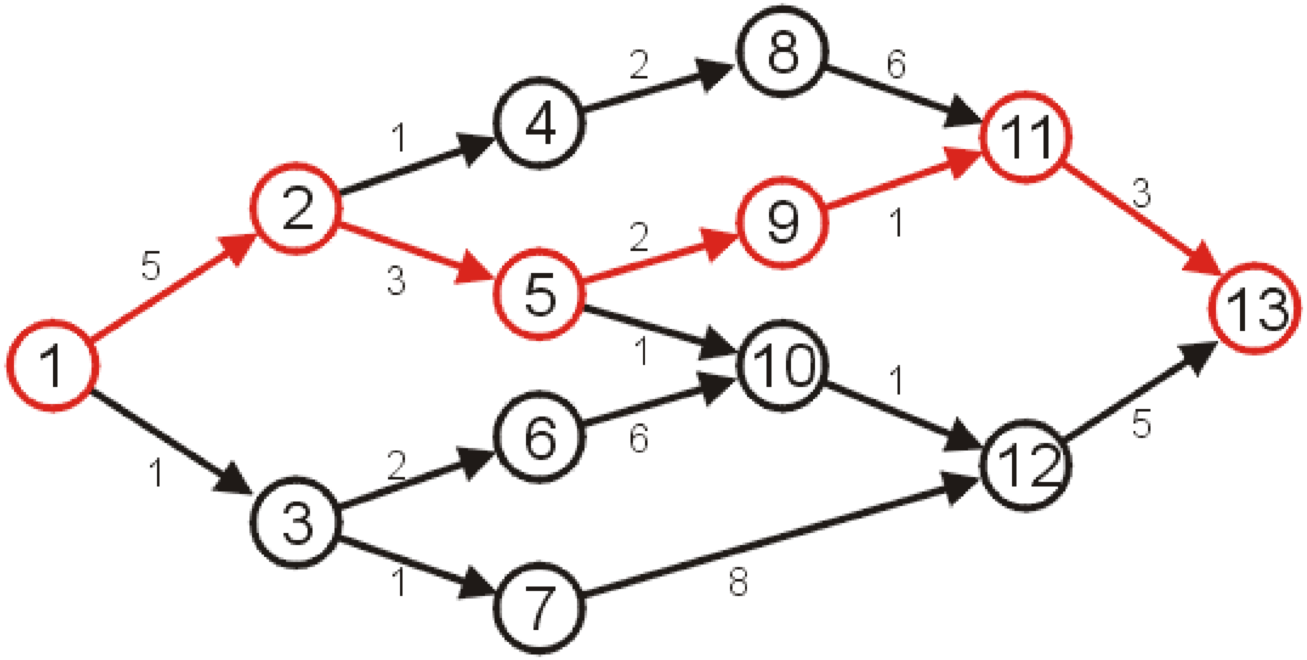
Shortest Path

- **Given the graph below, suppose we wish to find the shortest path from vertex 1 to vertex 13**



Shortest Path

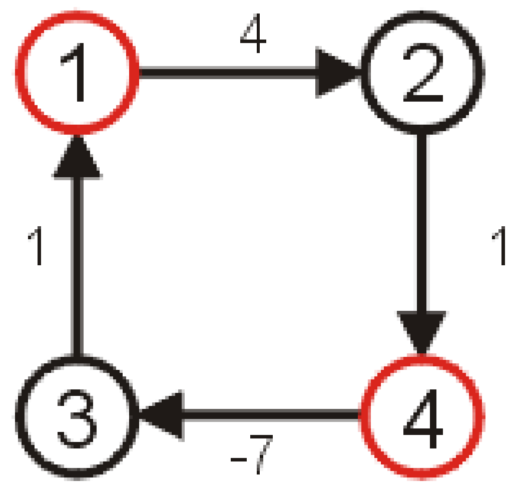
- After some consideration, we may determine that the shortest path is as follows, with length 14



- Other paths exists, but they are longer

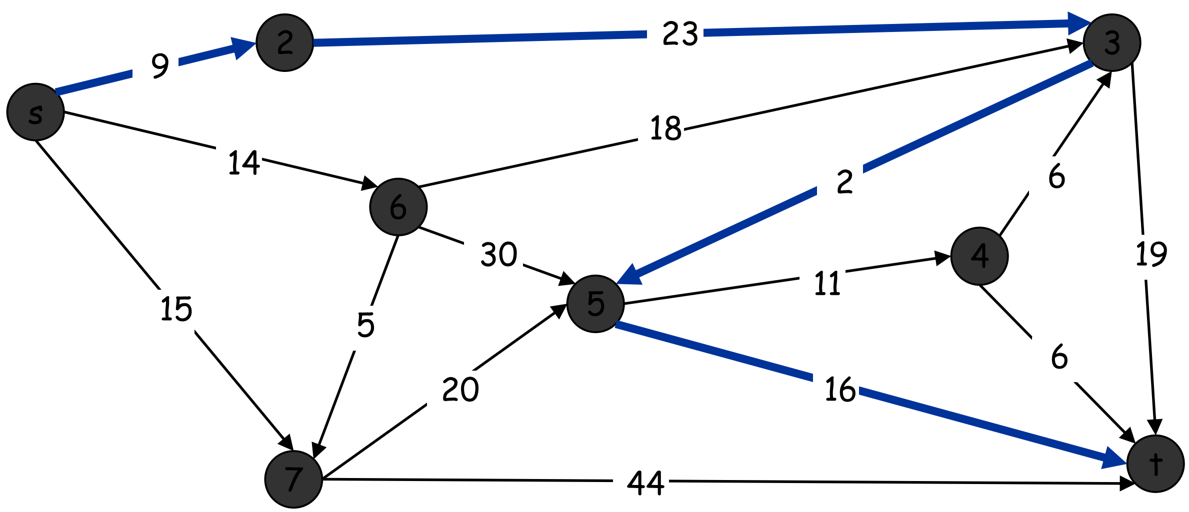
Negative Cycles

- Clearly, if we have **negative** edges, it may be possible to end up in a cycle whereby each pass through the cycle decreases the total *length*
- Thus, a shortest length would be undefined for such a graph
- Consider the shortest path from vertex 1 to 4...
- We will only consider non-negative weights.



Shortest Path Example

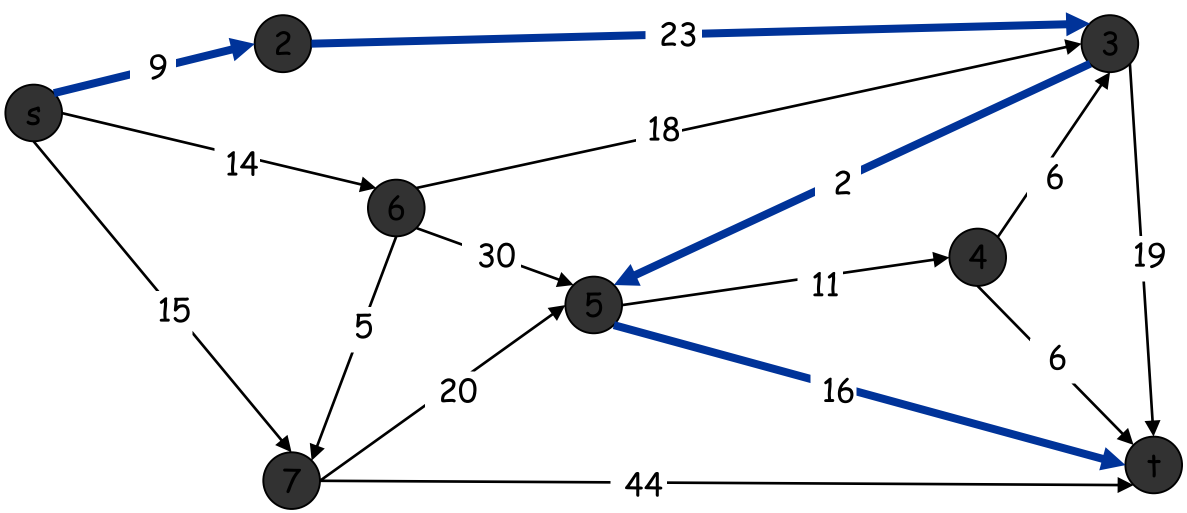
- **Given:**
  - Weighted Directed graph G = (V, E).
  - Source *s*, destination *t*.
- **Find shortest directed path from s to *t*.**



Cost of path s-2-3-5-t
= 9 + 23 + 2 + 16
= 48.

Discussion Items

- How many possible paths are there from *s* to *t*?
- Can we safely ignore cycles? If so, how?
- Any suggestions on how to reduce the set of possibilities?
- Can we determine a lower bound on the complexity like we did for comparison sorting?



Key Observation

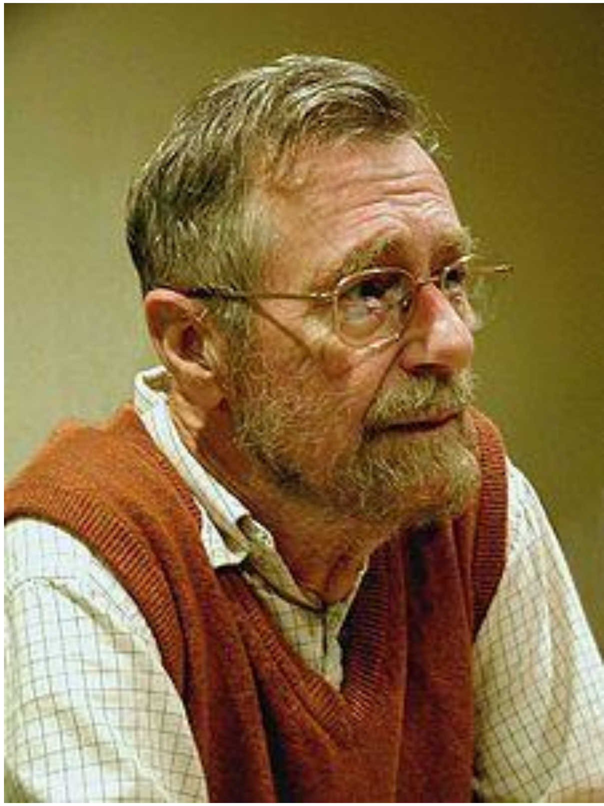
- **A key observation is that if the shortest path contains the node *v*, then:**
  - It will only contain *v* once, as any cycles will only add to the length => **NO CYCLE**
  - The path from *s* to *v* must be the shortest path to *v* from *s*.
  - The path from *v* to *t* must be the shortest path to *t* from *v*.
- **Thus, if we can determine the shortest path to all other vertices that are incident to the target vertex we can easily compute the shortest path.**
  - Implies a set of sub-problems on the graph with the target vertex removed.

Outline

- **Shortest paths**
  - Introduction
  - Applications
  - Dijkstra's algorithm
- **Minimum spanning trees**
  - Introduction
  - Applications
  - Kruskal algorithm

Edsger Wybe Dijkstra

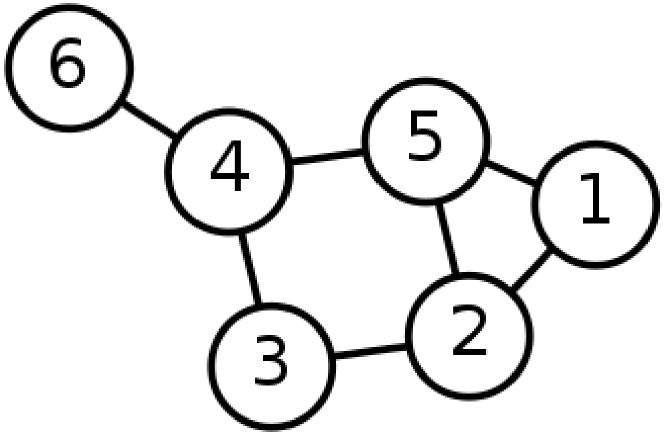
- May 11, 1930 – August 6, 2002
- Received the 1972 A. M. Turing Award, widely considered the most prestigious award in computer science
- The Schlumberger Centennial Chair of Computer Sciences at The University of Texas at Austin from 1984 until 2000
- Made a strong case against use of the GOTO statement in programming languages and helped lead to its deprecation
- Known for his many essays on programming



Edsger Wybe Dijkstra

Single-Source Shortest Path Problem

- **Single-Source Shortest Path Problem:** The problem of finding shortest paths from a source vertex *v* to all other vertices in the graph.



Dijkstra’s algorithm

Dijkstra's algorithm - is a solution to the single-source shortest path problem in graph theory.

Works on both directed and undirected graphs. However, all edges must have nonnegative weights.

Approach: Greedy

Input: Weighted graph G={E,V} and source vertex *v*∈V, such that all edge weights are nonnegative

Output: Lengths of shortest paths (or the shortest paths themselves) from a given source vertex *v*∈V to all other vertices

Dijkstra's algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
    do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$ $\triangleright$ Q is a priority queue maintaining $V - S$
while $Q \neq \varnothing$
    do $u \leftarrow$ EXTRACT-MIN(Q)
        $S \leftarrow S \cup \{u\}$
        for each $v \in Adj[u]$
            do if $d[v] > d[u] + w(u, v)$
                then $d[v] \leftarrow d[u] + w(u, v)$
                $p[v] \leftarrow u$

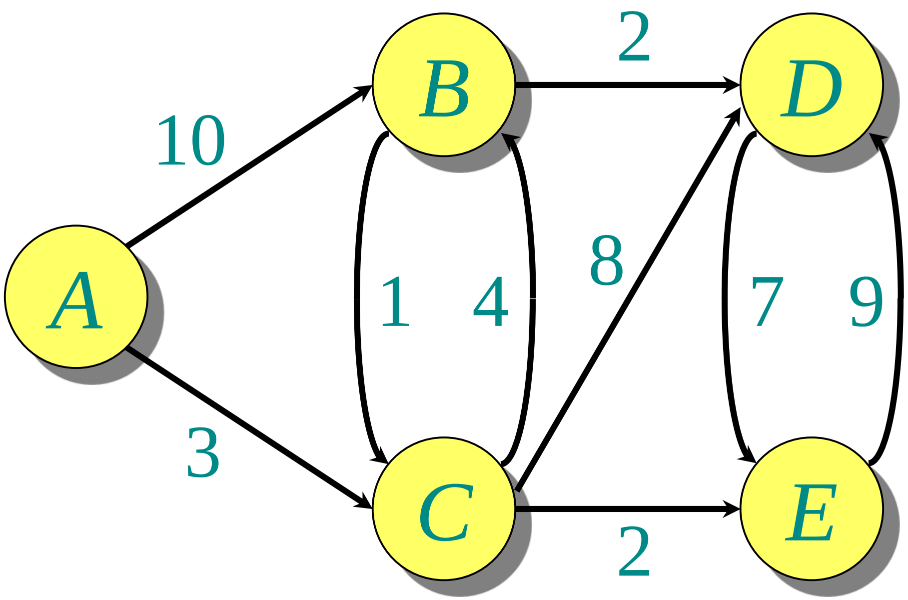
Dijkstra's algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$ $\rhd$ Q is a priority queue maintaining $V - S$
while $Q \neq \varnothing$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    for each $v \in Adj[u]$
      do if $d[v] > d[u] + w(u, v)$ ***relaxation***
        then $d[v] \leftarrow d[u] + w(u, v)$ ***step***
        $p[v] \leftarrow u$

Implicit DECREASE-KEY

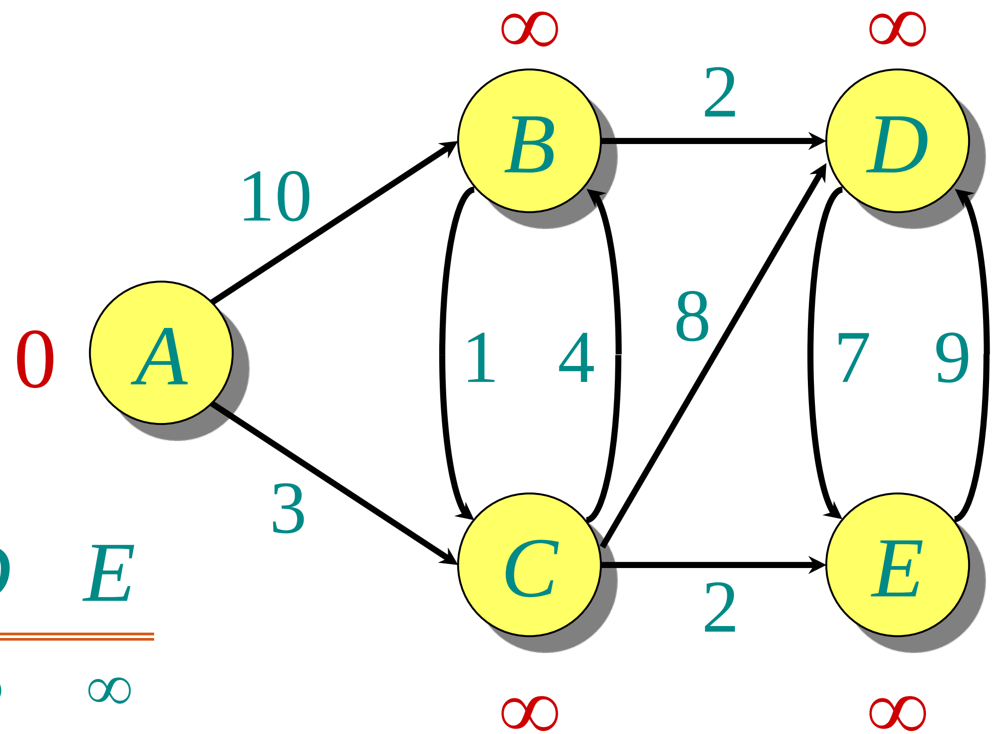
Example of Dijkstra's algorithm

Graph with nonnegative edge weights:



Example of Dijkstra's algorithm

Initialize:

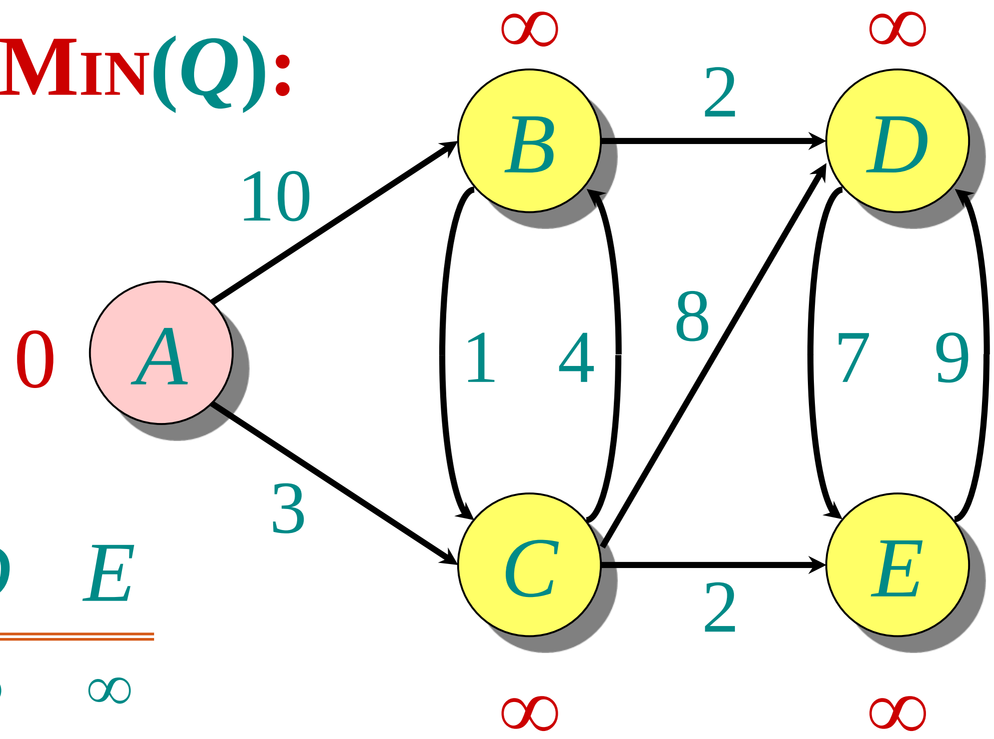


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: { }

Example of Dijkstra's algorithm

"A" ← EXTRACT-MIN(Q):

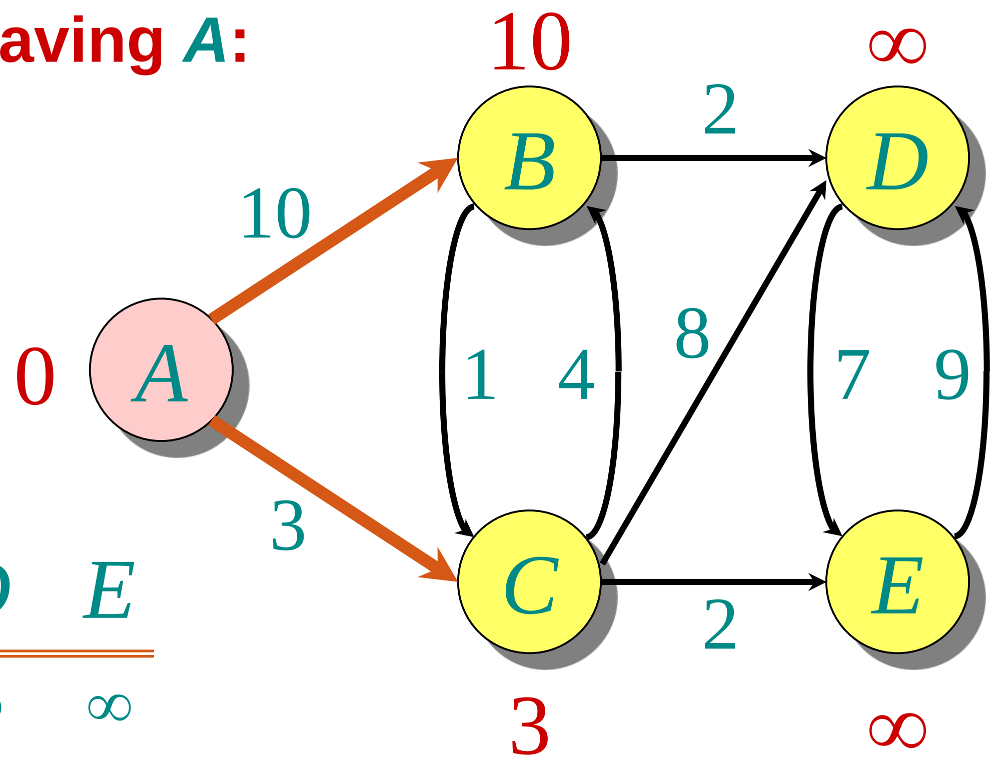


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: { A }

Example of Dijkstra's algorithm

Relax all edges leaving *A*:

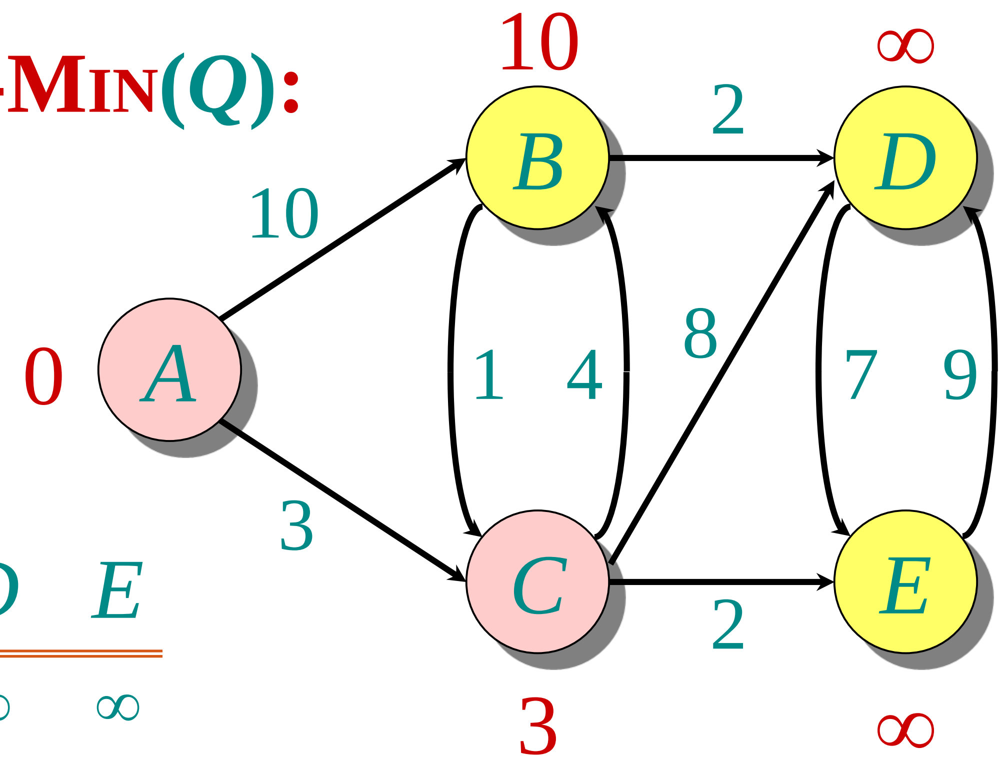


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

Example of Dijkstra's algorithm

"C" $\leftarrow$ EXTRACT-MIN(Q):

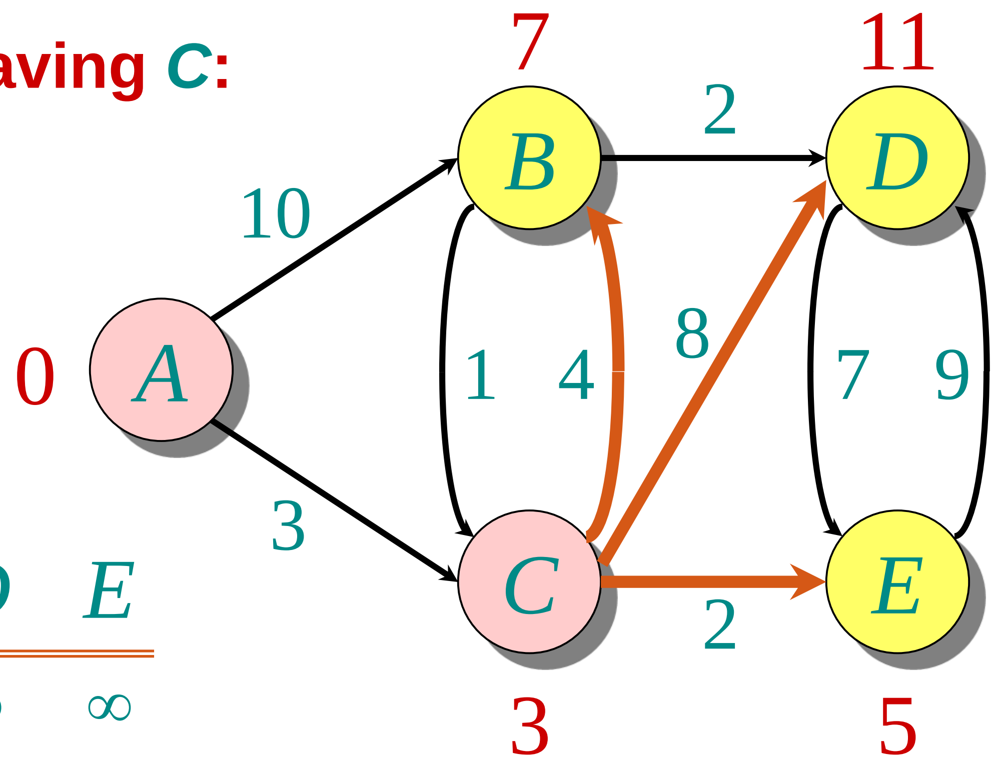


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }

Example of Dijkstra's algorithm

Relax all edges leaving *C*:

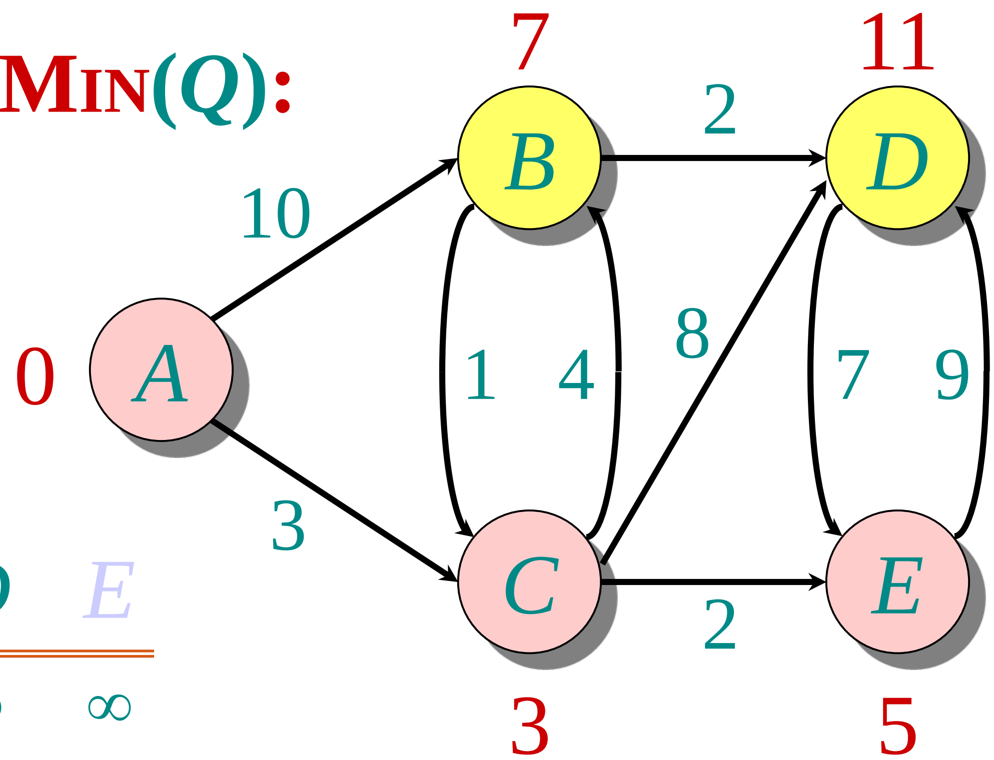


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C }

Example of Dijkstra's algorithm

"*E*" ← EXTRACT-MIN(*Q*):

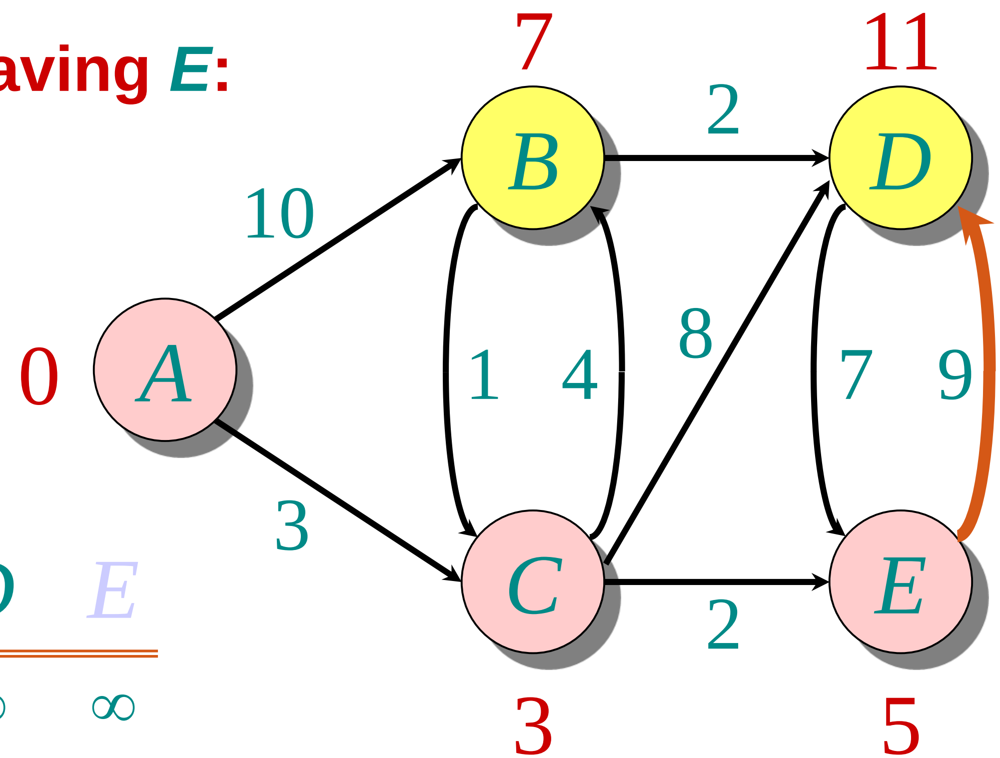


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { *A*, *C*, *E* }

Example of Dijkstra's algorithm

Relax all edges leaving *E*:

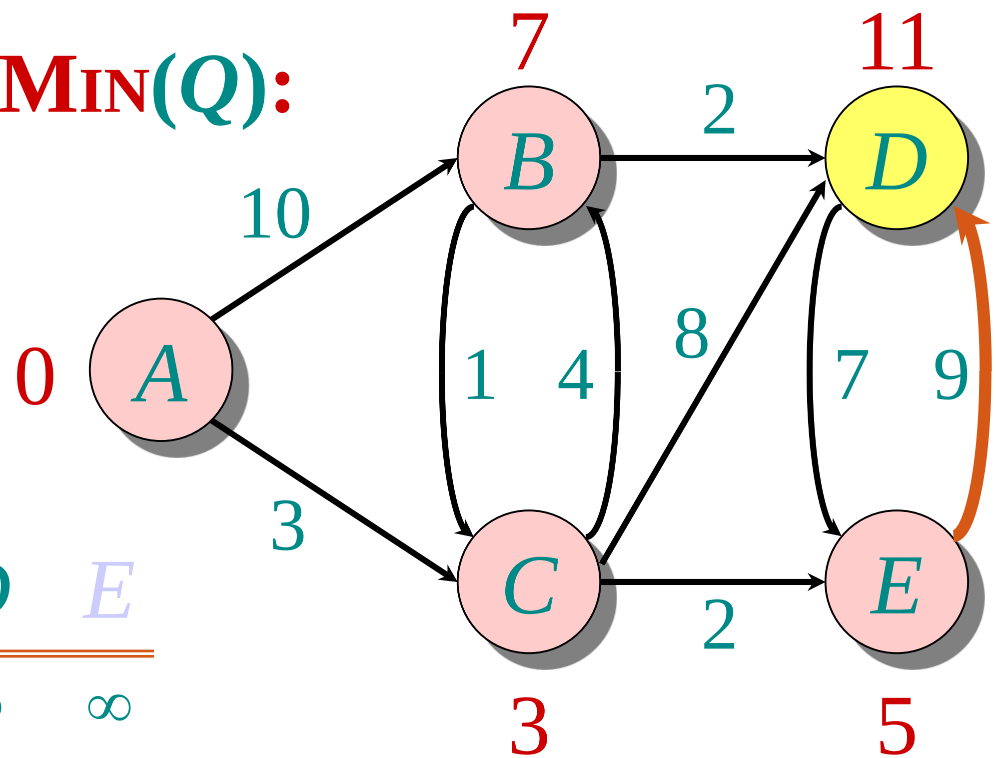


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { *A*, *C*, *E* }

Example of Dijkstra's algorithm

“B” ← **EXTRACT-MIN(*Q*):**

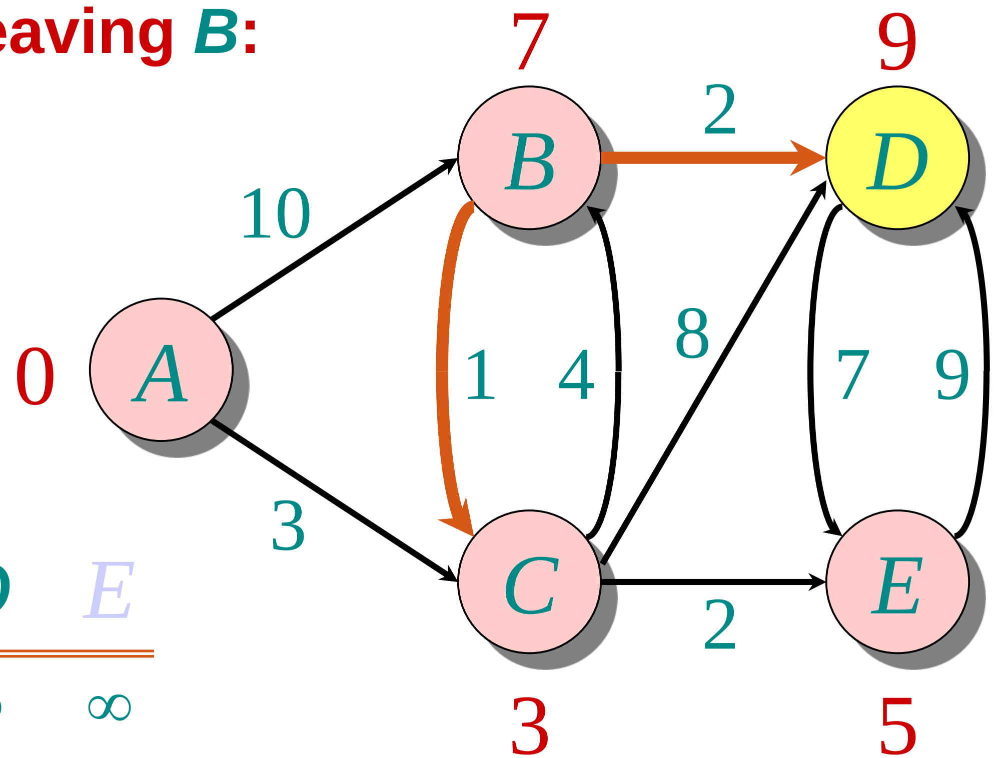


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { *A*, *C*, *E*, *B* }

Example of Dijkstra's algorithm

Relax all edges leaving *B*:

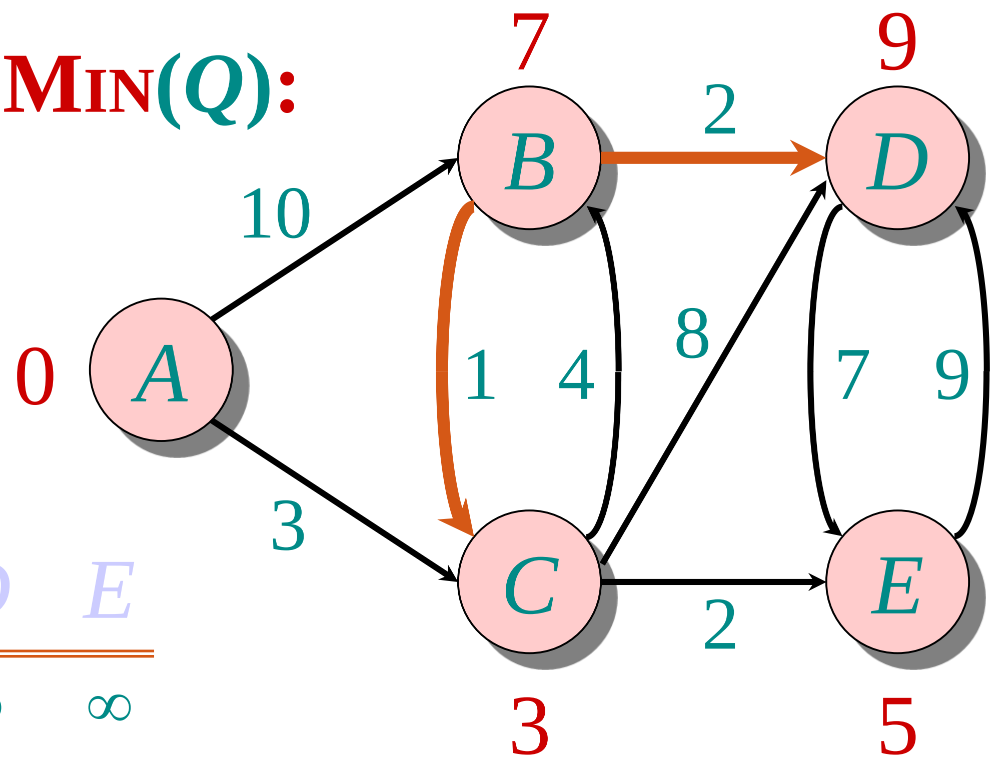


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { A, C, E, B }

Example of Dijkstra's algorithm

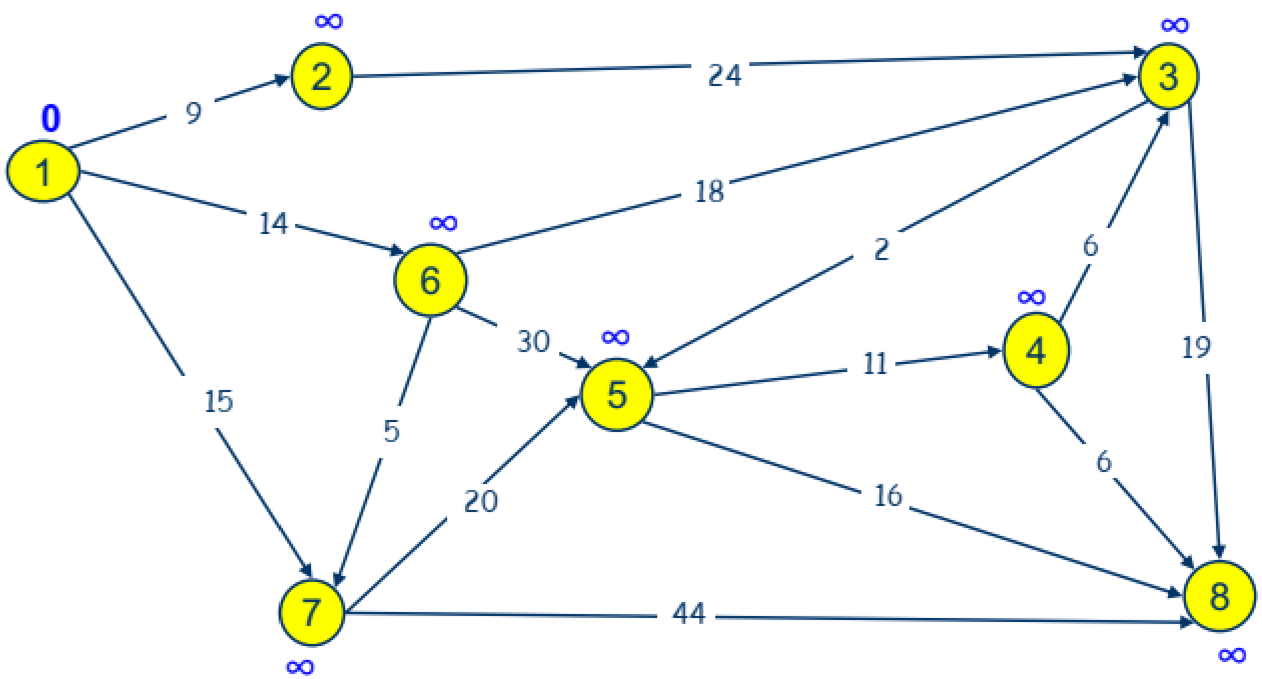
"D" ← EXTRACT-MIN(Q):



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

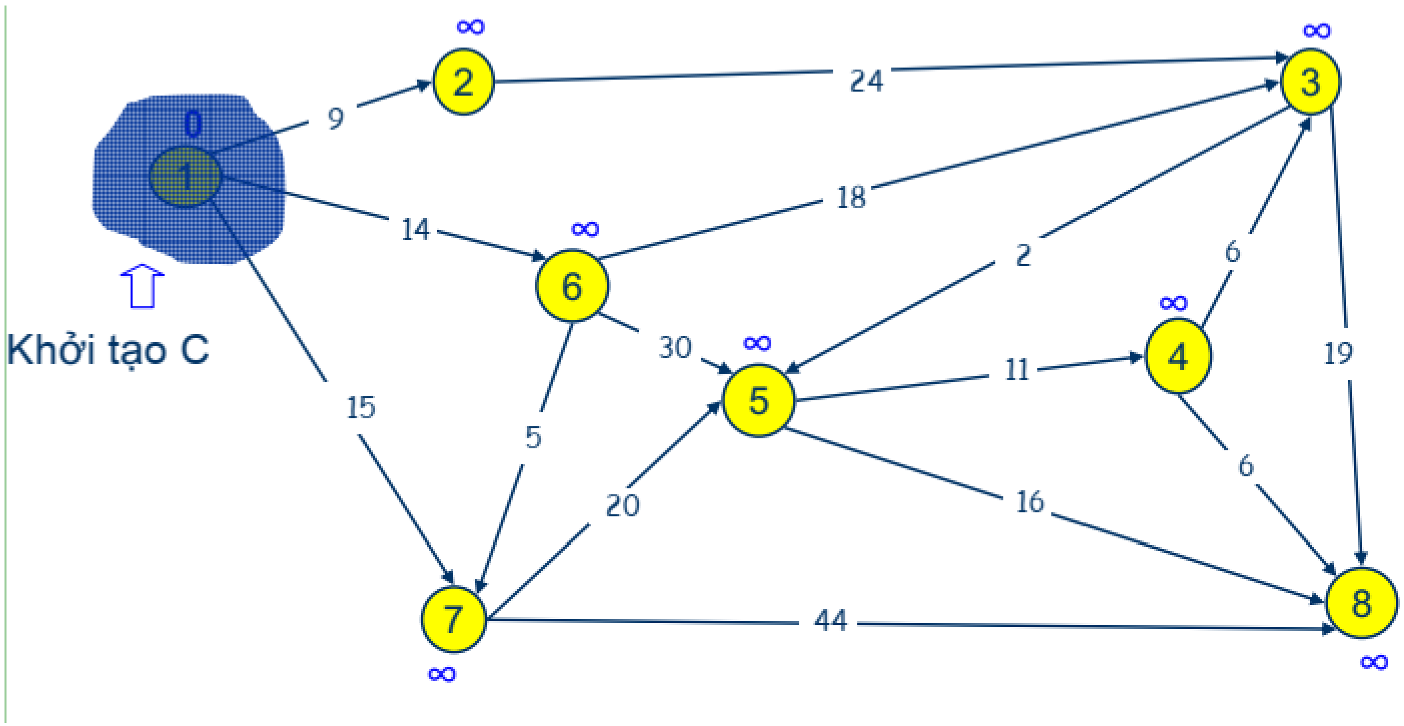
S: { A, C, E, B, D }

Example

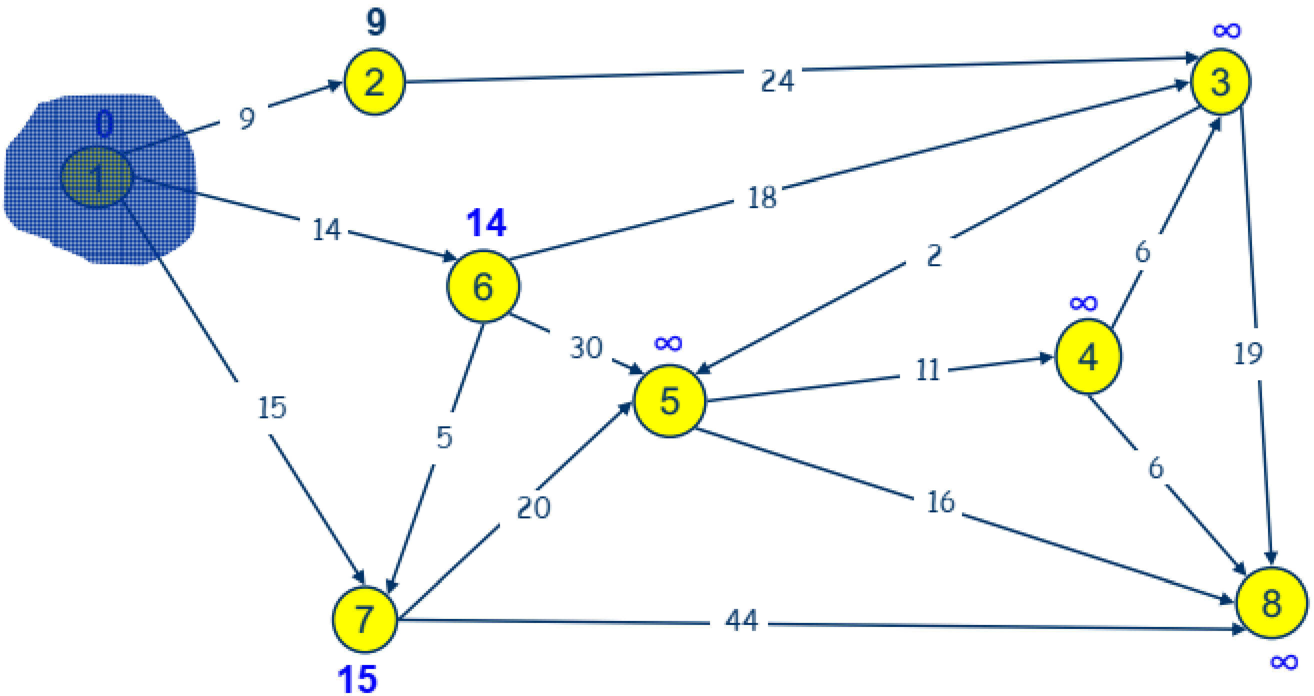


Khởi tạo các giá trị d cho tất cả các đỉnh

Example

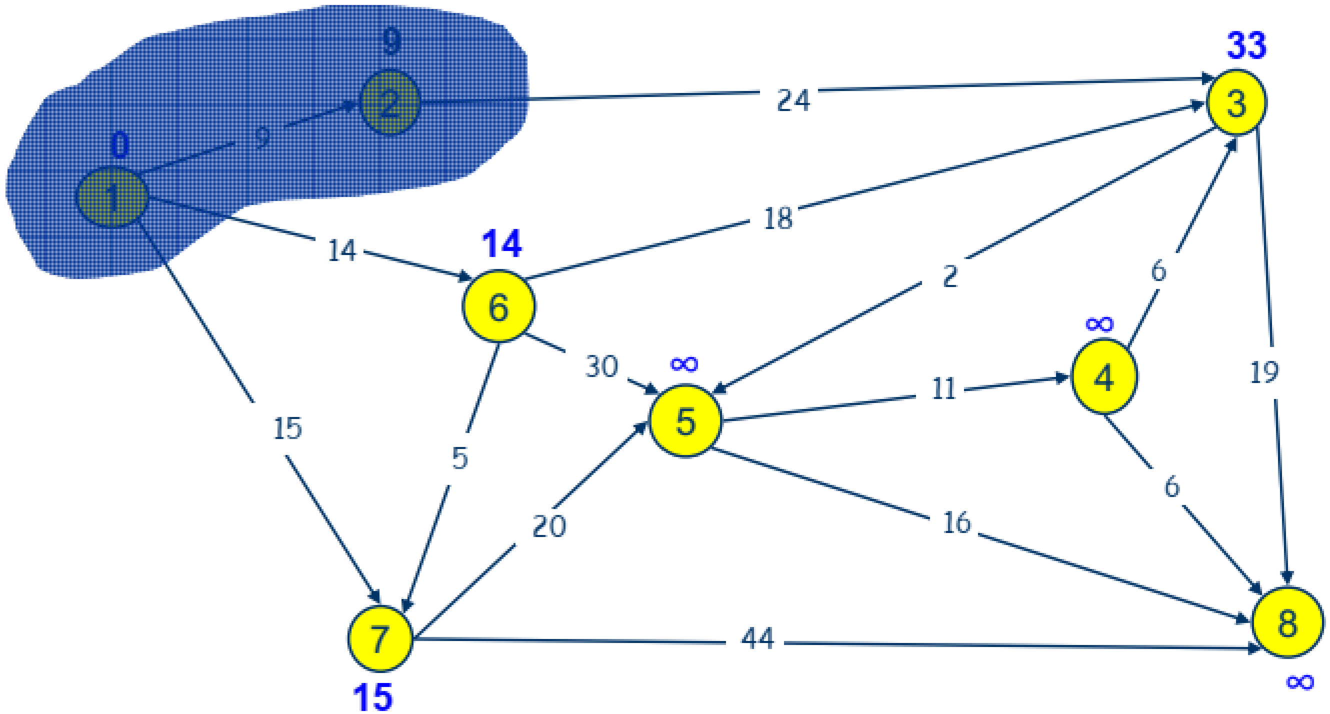


Example



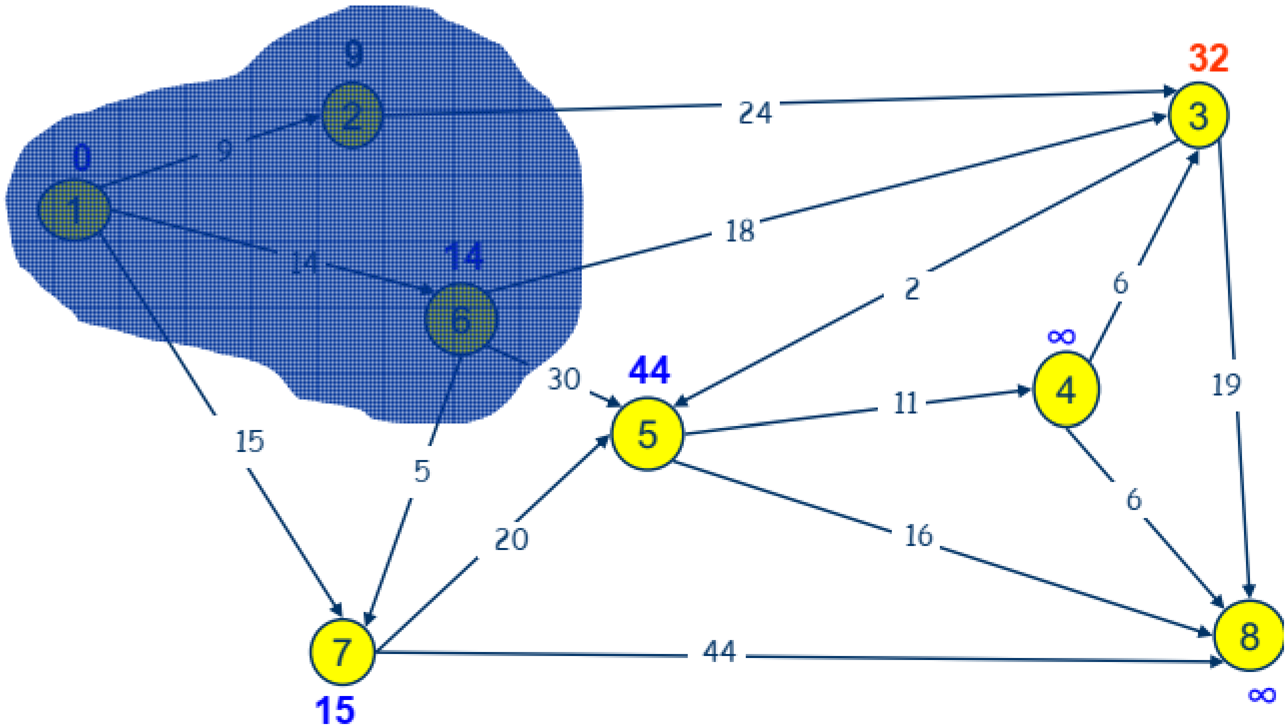
Cập nhật các giá trị d[2] = 9, d[6] = 14, d[7] = 15

Example



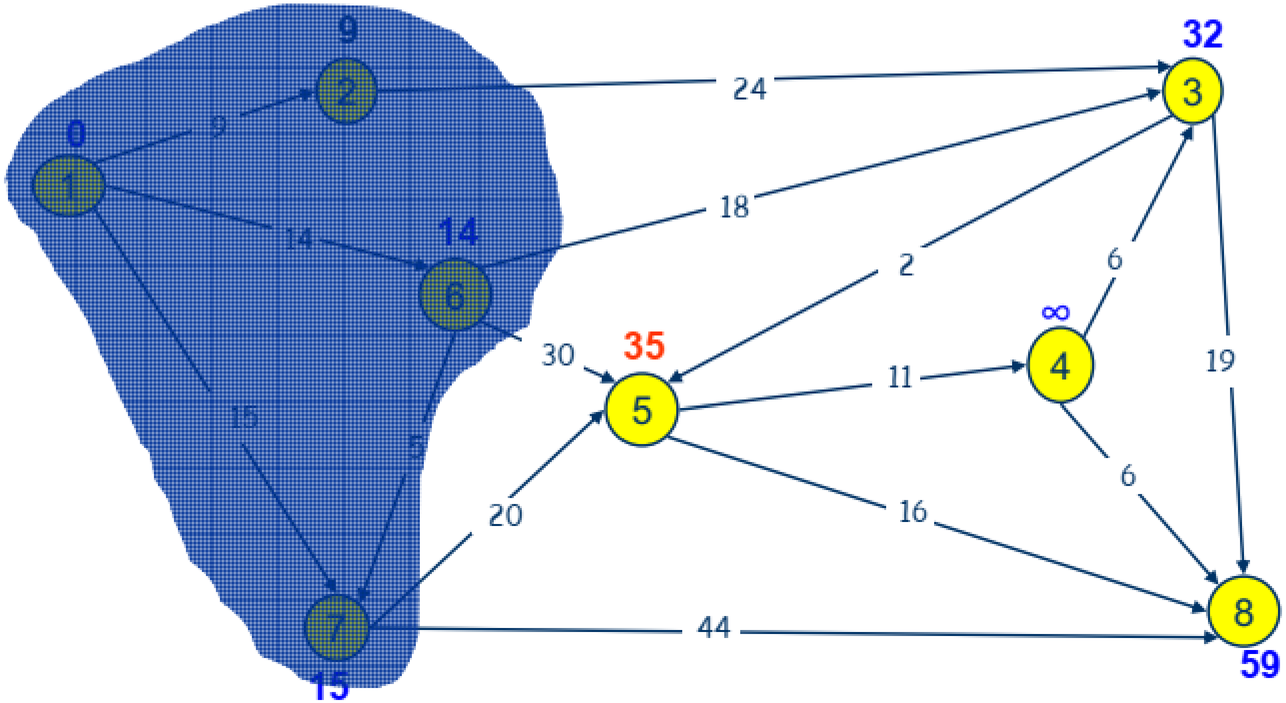
Mở rộng cụm C, đường đi ngắn nhất từ 1 đến 2 có độ dài 9
Cập nhật giá trị d của các đỉnh lân cận của 2

Example



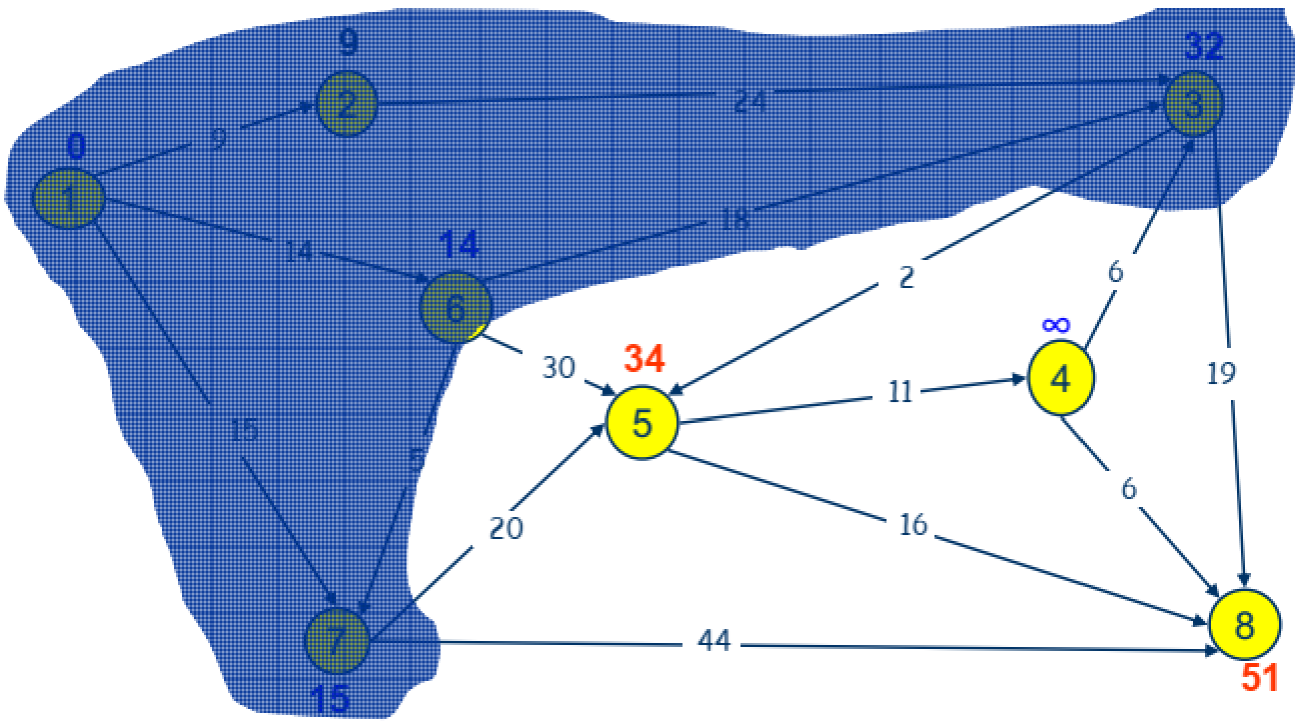
Mở rộng cụm C, đường đi ngắn nhất từ 1 đến 6 có độ dài 14
Cập nhật giá trị d của các đỉnh lân cận với 6

Example



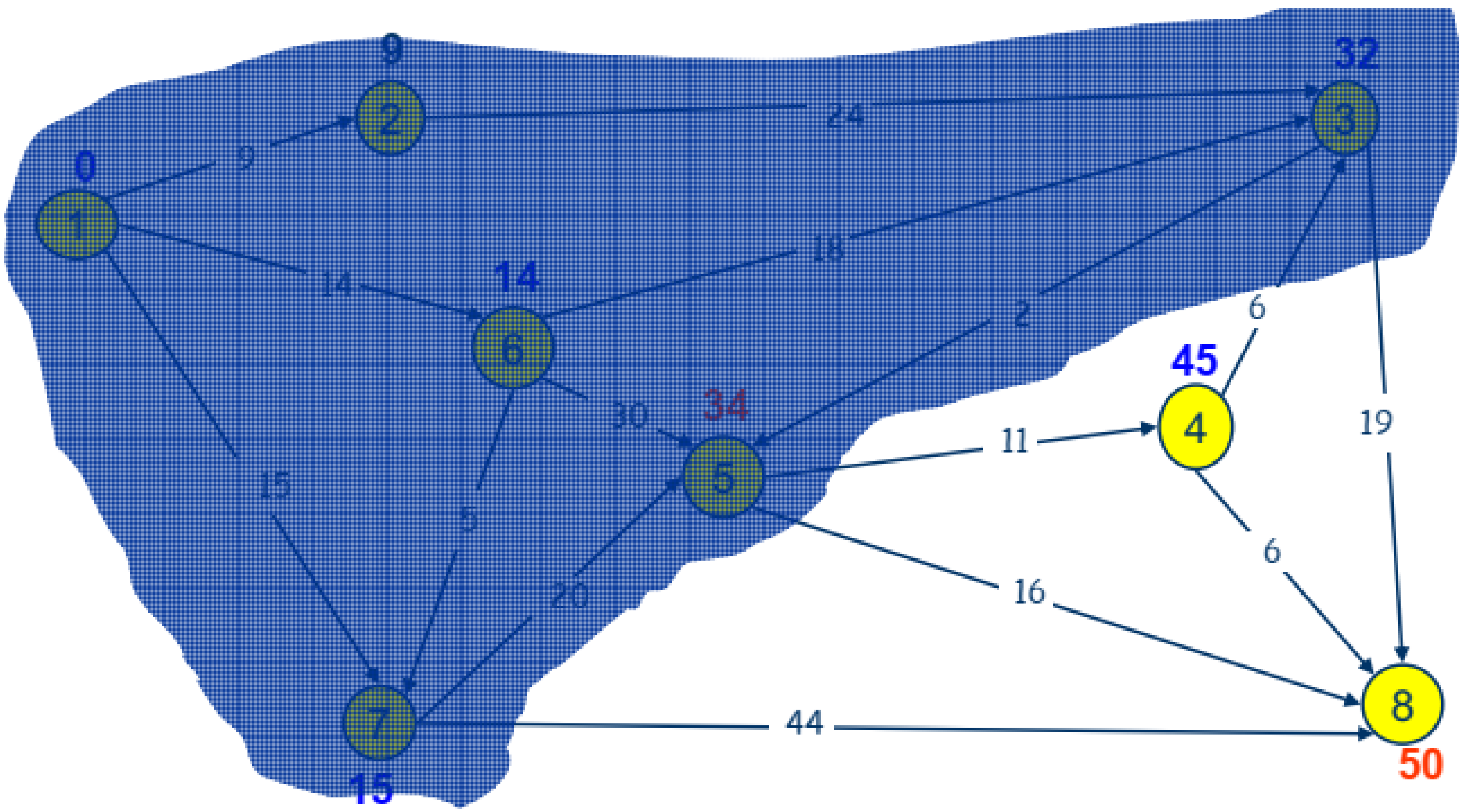
Mở rộng cụm C, đường đi ngắn nhất từ 1 đến 7 có độ dài 15
Cập nhật giá trị d của các đỉnh lân cận với 7

Example



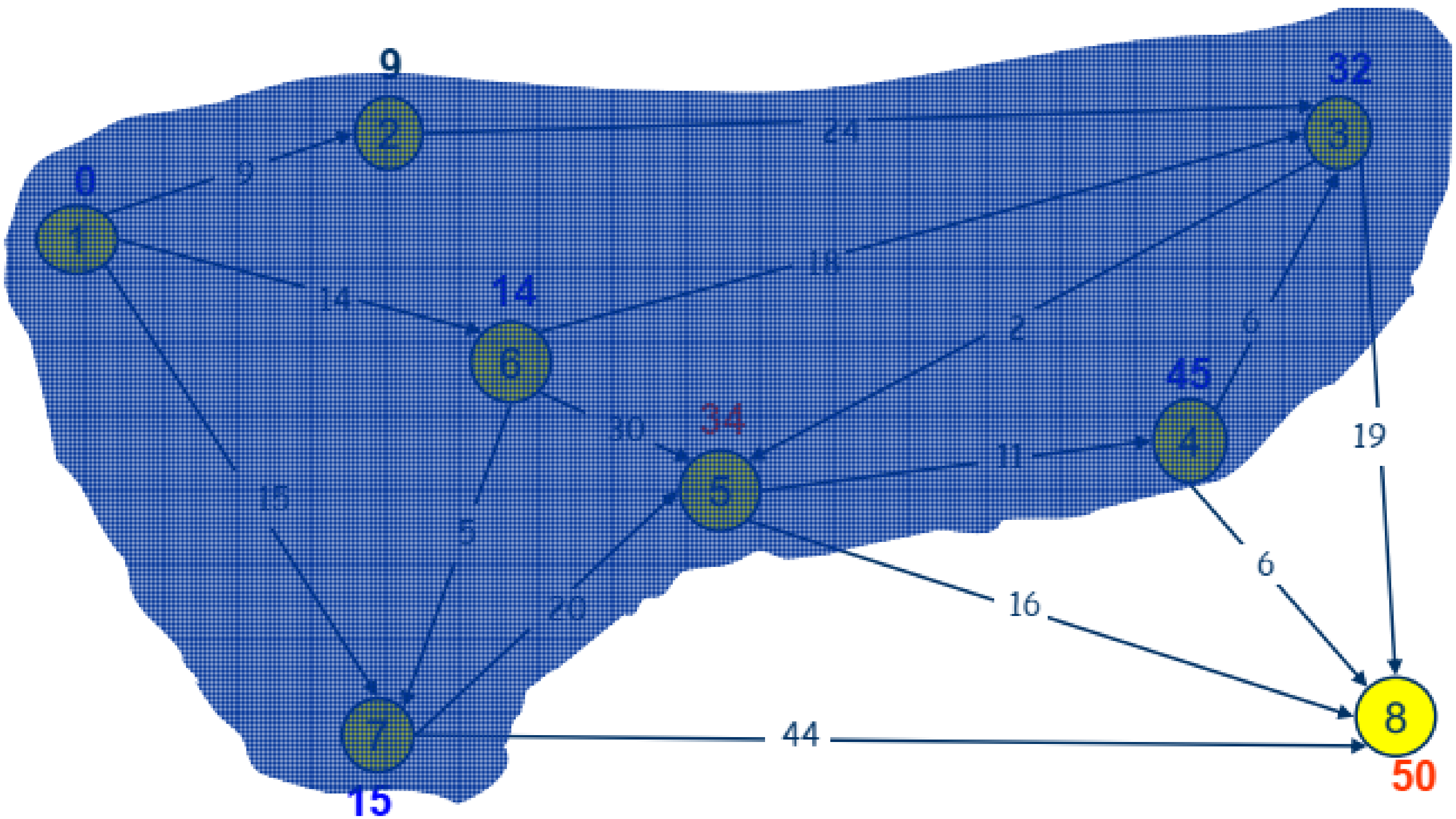
Mở rộng cụm C, đường đi ngắn nhất từ 1 đến 3 có độ dài 32, đi qua 6
Cập nhật giá trị d của các đỉnh lân cận với 3

Example



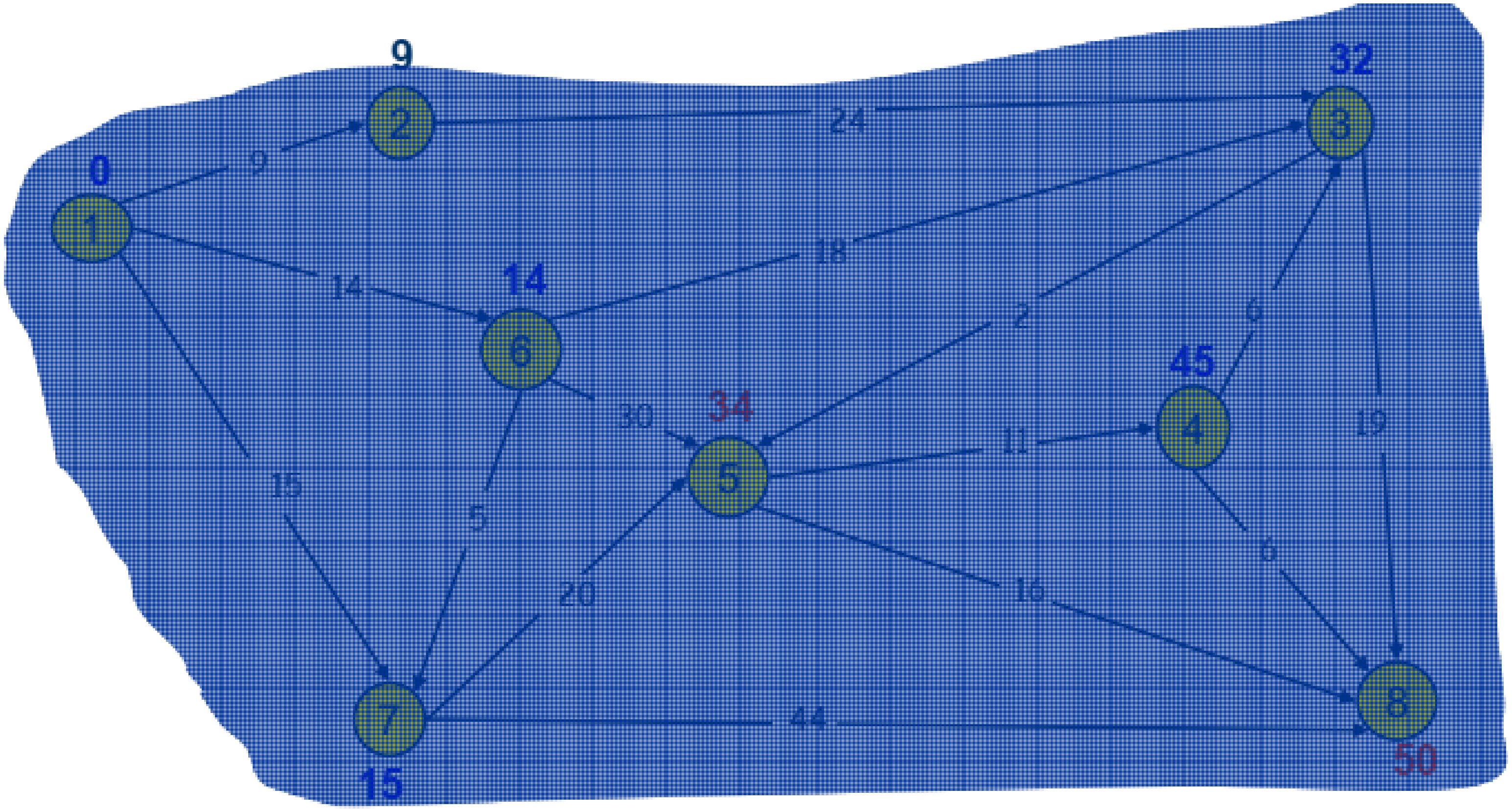
Mở rộng cụm C, đường đi ngắn nhất từ 1 đến 5 có độ dài 34, đi qua 6,3
Cập nhật giá trị d của các đỉnh lân cận với 5

Example



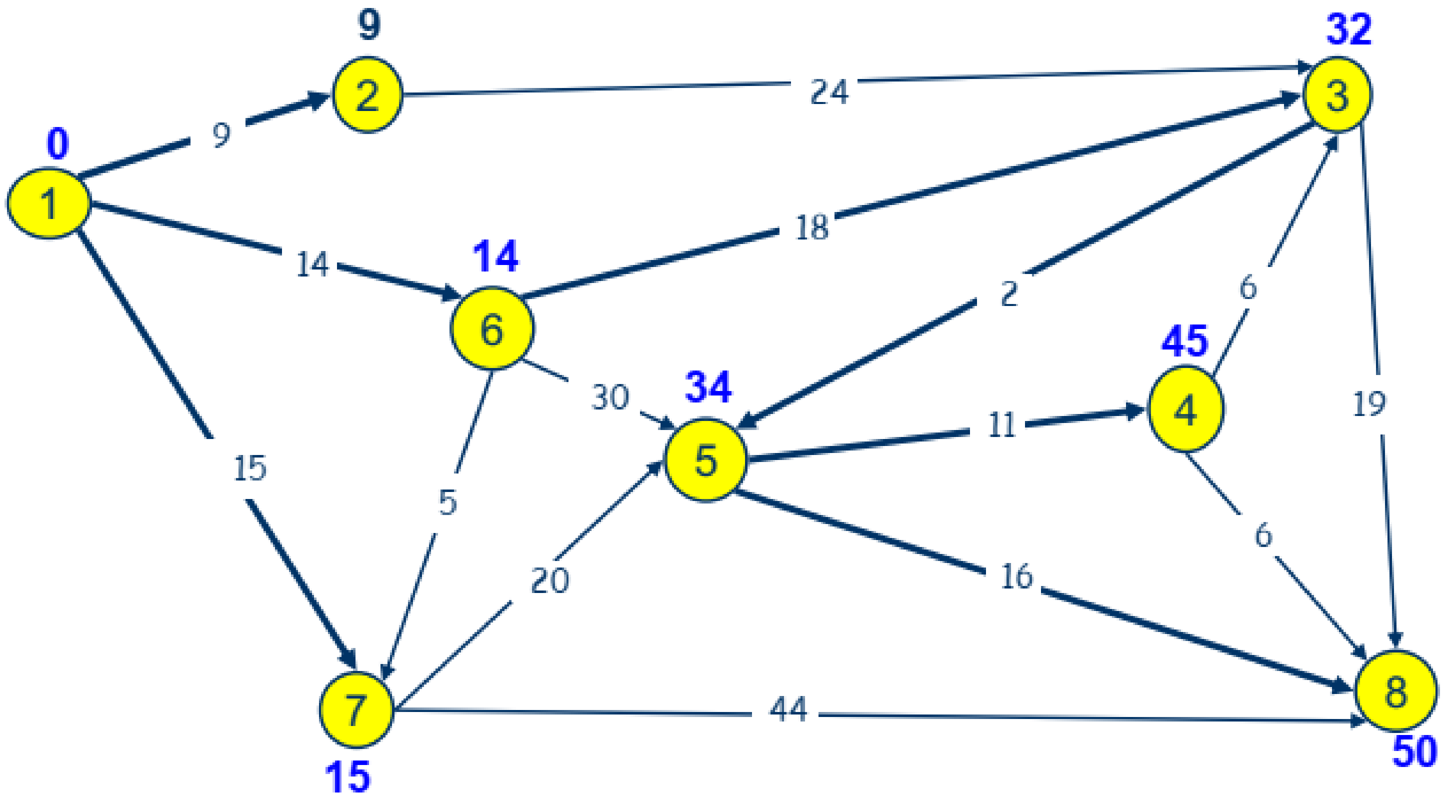
Mở rộng cụm C, đường đi ngắn nhất từ 1 đến 4 có độ dài 45, đi qua 6,3,5
Cập nhật giá trị d của các đỉnh lân cận với 4

Example



Mở rộng cụm C, đường đi ngắn nhất từ 1 đến 8 có độ dài 50, đi qua 1,6,3,5

Example

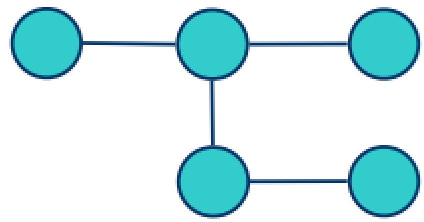


Outline

- **Shortest paths**
  - Introduction
  - Applications
  - Dijkstra’s algorithm
- **Minimum spanning trees**
  - Introduction
  - Applications
  - Kruskal’s algorithm

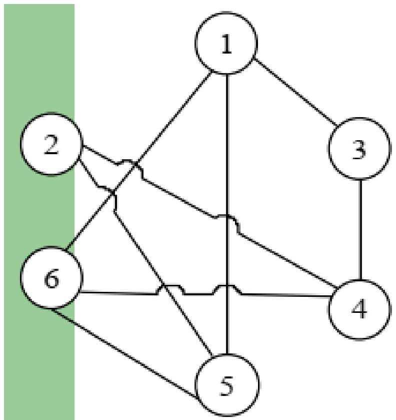
Spanning tree

- Tree: undirected graph, no cycle

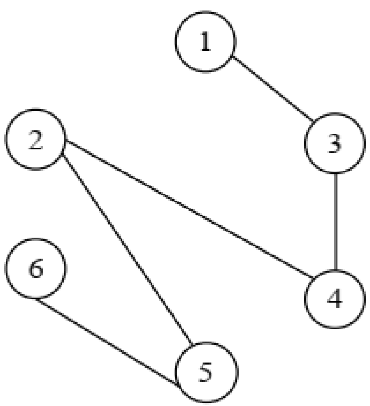


Cây

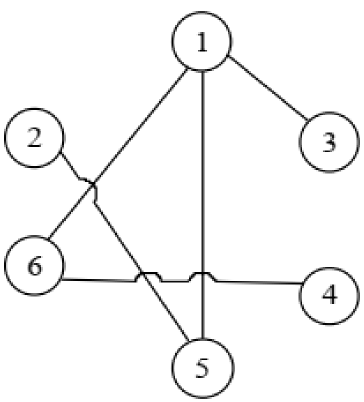
- In an undirected and connected graph: a spanning tree contains all vertices of the graph



Đồ thị



Cây khung



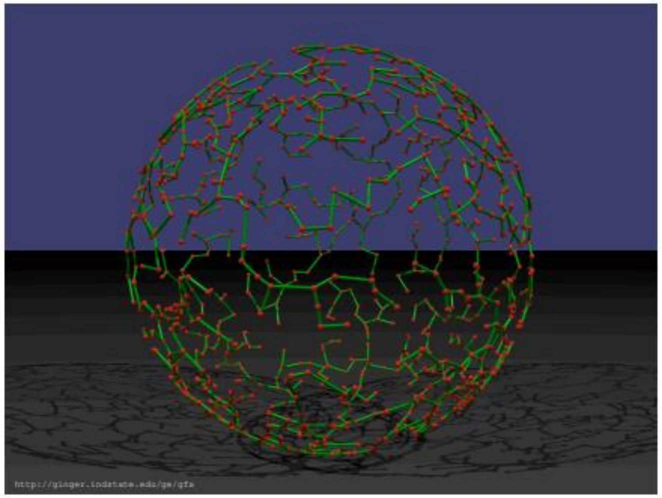
Cây khung

Minimum Spanning Tree

- A minimum spanning tree (MST) is defined as a spanning tree with weight less than or equal to the weight of every other spanning tree.
- A minimum spanning tree is a spanning tree that has weights associated with its edges, and the total weight of the tree (the sum of the weights of its edges) is at a minimum

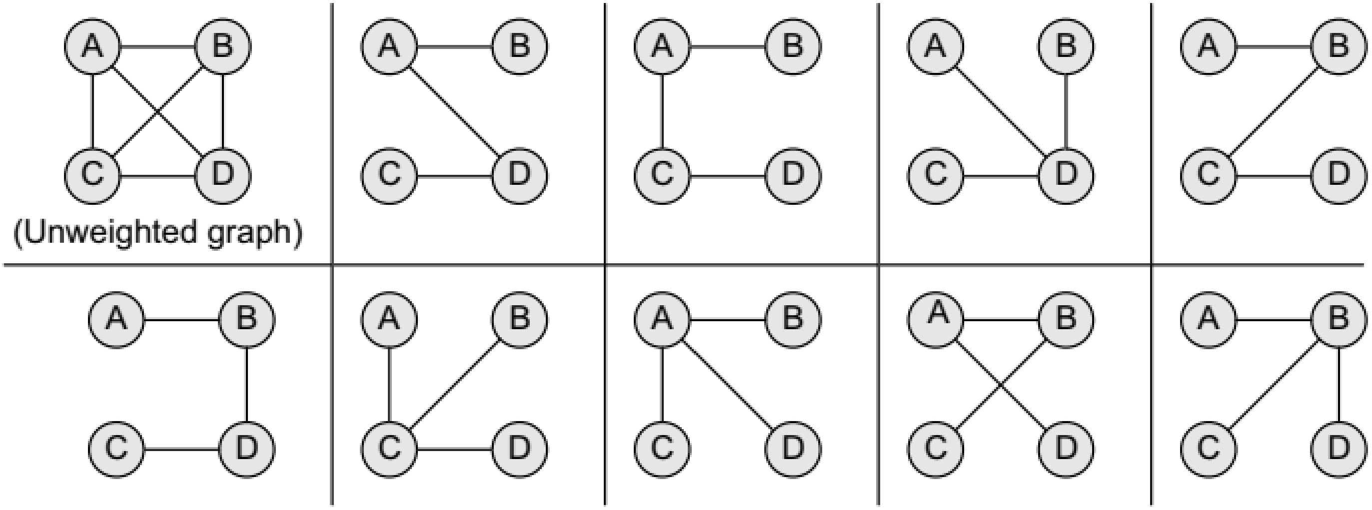
MST applications

- MSTs are widely used for designing networks
- MSTs are used to find airline routes
- MSTs are used to find airline routes
- MSTs are applied in routing algorithms for finding the most efficient path

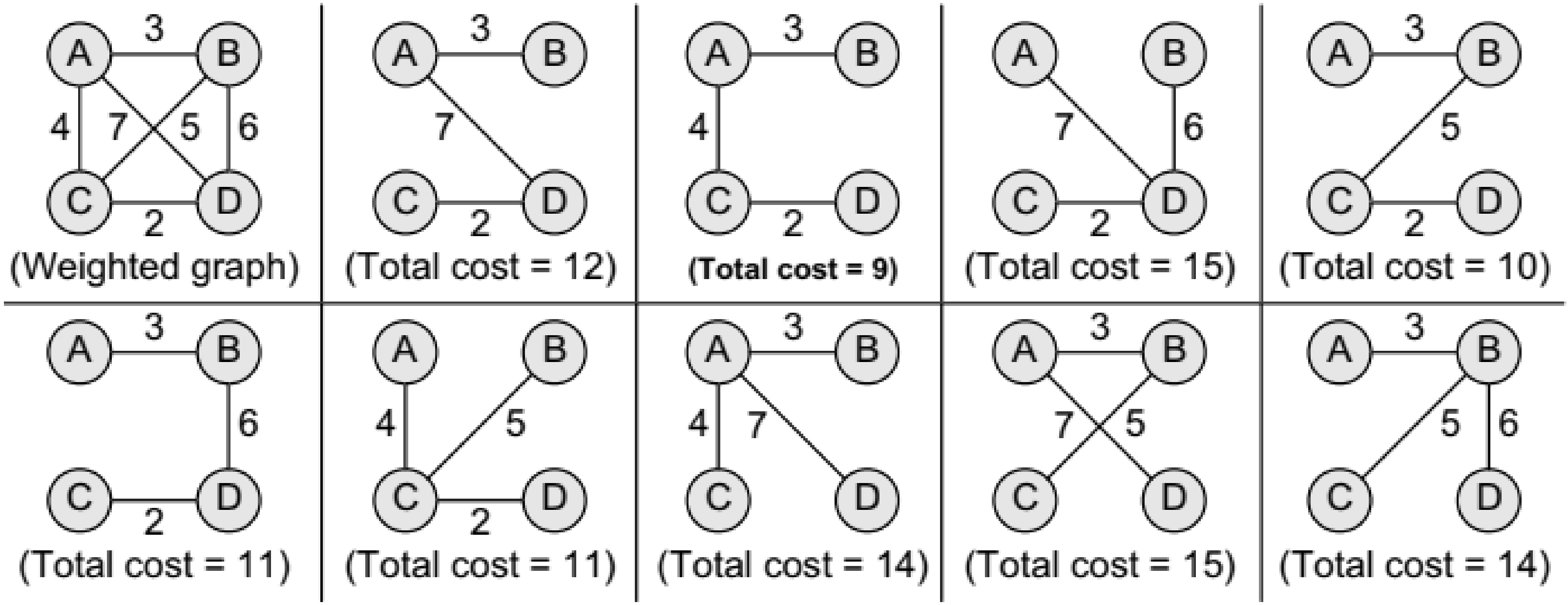


Properties

- Possible multiplicity
- Uniqueness
- Minimum-cost subgraph
- Cycle property
- Usefulness
- Simplicity



Properties

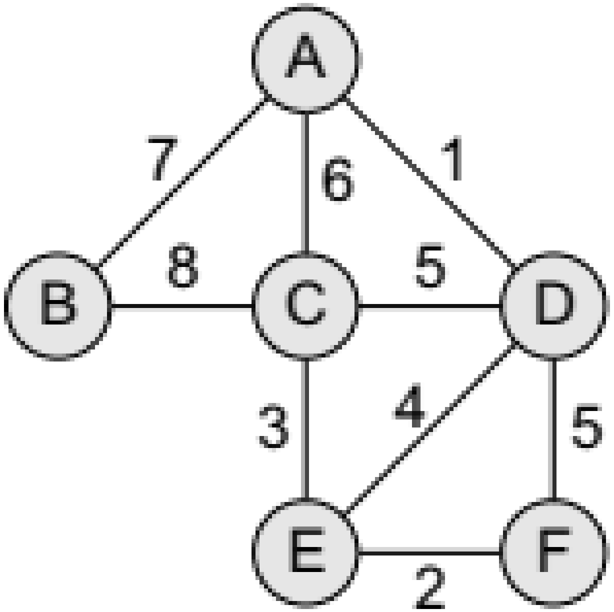

(Weighted graph)
(Total cost = 12)
(Total cost = 9)
(Total cost = 15)
(Total cost = 10)
(Total cost = 11)
(Total cost = 11)
(Total cost = 14)
(Total cost = 15)
(Total cost = 14)

Kruskal's algorithm

```
Step 1: Create a forest in such a way that each graph is a separate
        tree.
Step 2: Create a priority queue Q that contains all the edges of the
        graph.
Step 3: Repeat Steps 4 and 5 while Q is NOT EMPTY
Step 4:      Remove an edge from Q
Step 5: IF the edge obtained in Step 4 connects two different trees,
        then Add it to the forest (for combining two trees into one
        tree).
        ELSE
             Discard the edge
Step 6: END
```

Kruskal’s algorithm

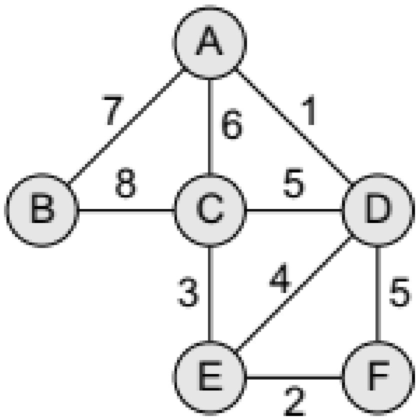
- Init:
  - F = {{A}, {B}, {C}, {D}, {E}, {F}}
  - MST = {}
  - Q = {(A, D), (E, F), (C, E), (E, D), (C, D), (D, F), (A, C), (A, B), (B, C)}



SET
2020

Kruskal's algorithm

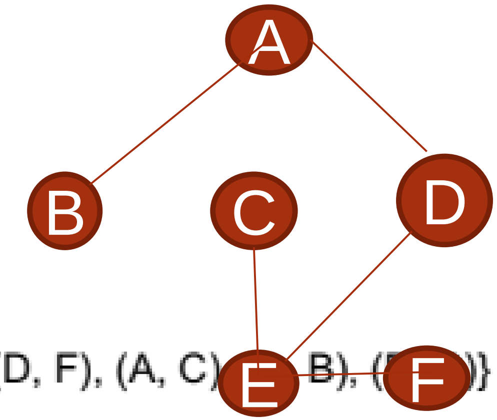
- **Step 1: remove edge (A,D) from Q and make following changes:**



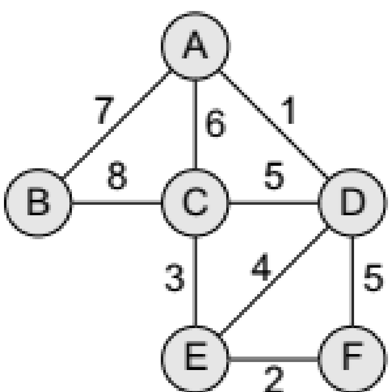
F = {{A, D}, {B}, {C}, {E}, {F}}

MST = {A, D}

Q = {~~(E, F)~~, ~~(C, E)~~, (E, D), (C, D), (D, F), (A, C), (A, B), (B, C)}



- **Step 2: Remove (E, F) from Q and make following changes**



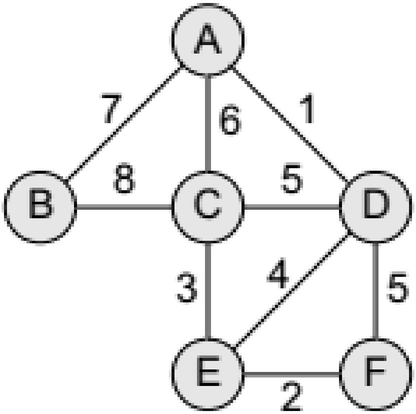
F = {{A, D}, {B}, {C}, {E, F}}

MST = {(A, D), (E, F)}

Q = {(C, E), (E, D), (C, D), (D, F), (A, C), (A, B), (B, C)}

Giải thuật Kruskal

- **Step 4: Remove (E,D) from Q and make changes:**



F = {{A, C, D, E, F}, {B}}
MST = {(A, D), (C, E), (E, F), (E, D)}
Q = {(C, D), (D, F), (A, C), (A, B), (B, C)}

- **Step 5: Remove (C,D) from Q and do:**

$$
\begin{aligned}
F &= \{\{A, C, D, E, F\}, \{B\}\} \\
MST &= \{(A, D), (C, E), (E, F), (E, D)\} \\
Q &= \{(D, F), (A, C), (A, B), (B, C)\}
\end{aligned}
$$

Giải thuật Kruskal

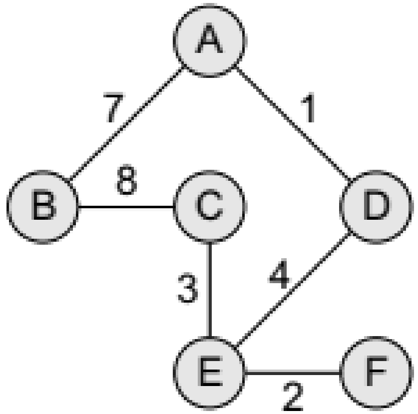
- **Step 6: Remove (D,F)**

$$\begin{aligned} F &= \{\{A, C, D, E, F\}, \{B\}\} \\ MST &= \{(A, D), (C, E), (E, F), (E, D)\} \\ Q &= \{(A, C), (A, B), (B, C)\} \end{aligned}$$

- **Step 7: Remove (A,C)**

$$\begin{aligned} F &= \{\{A, C, D, E, F\}, \{B\}\} \\ MST &= \{(A, D), (C, E), (E, F), (E, D)\} \\ Q &= \{(A, B), (B, C)\} \end{aligned}$$

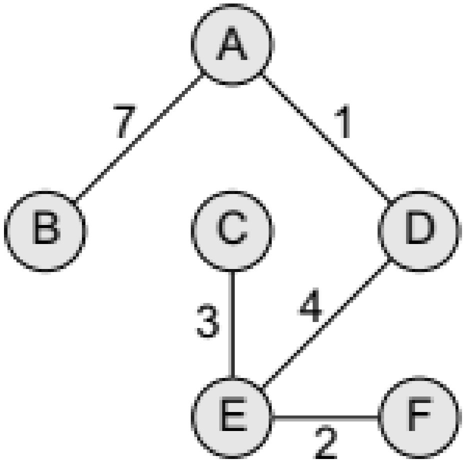
- **Step 8: Remove (A,B)**



$$\begin{aligned} F &= \{A, B, C, D, E, F\} \\ MST &= \{(A, D), (C, E), (E, F), (E, D), (A, B)\} \\ Q &= \{(B, C)\} \end{aligned}$$

Giải thuật Kruskal

- **Step 9: Continue until Q = empty**



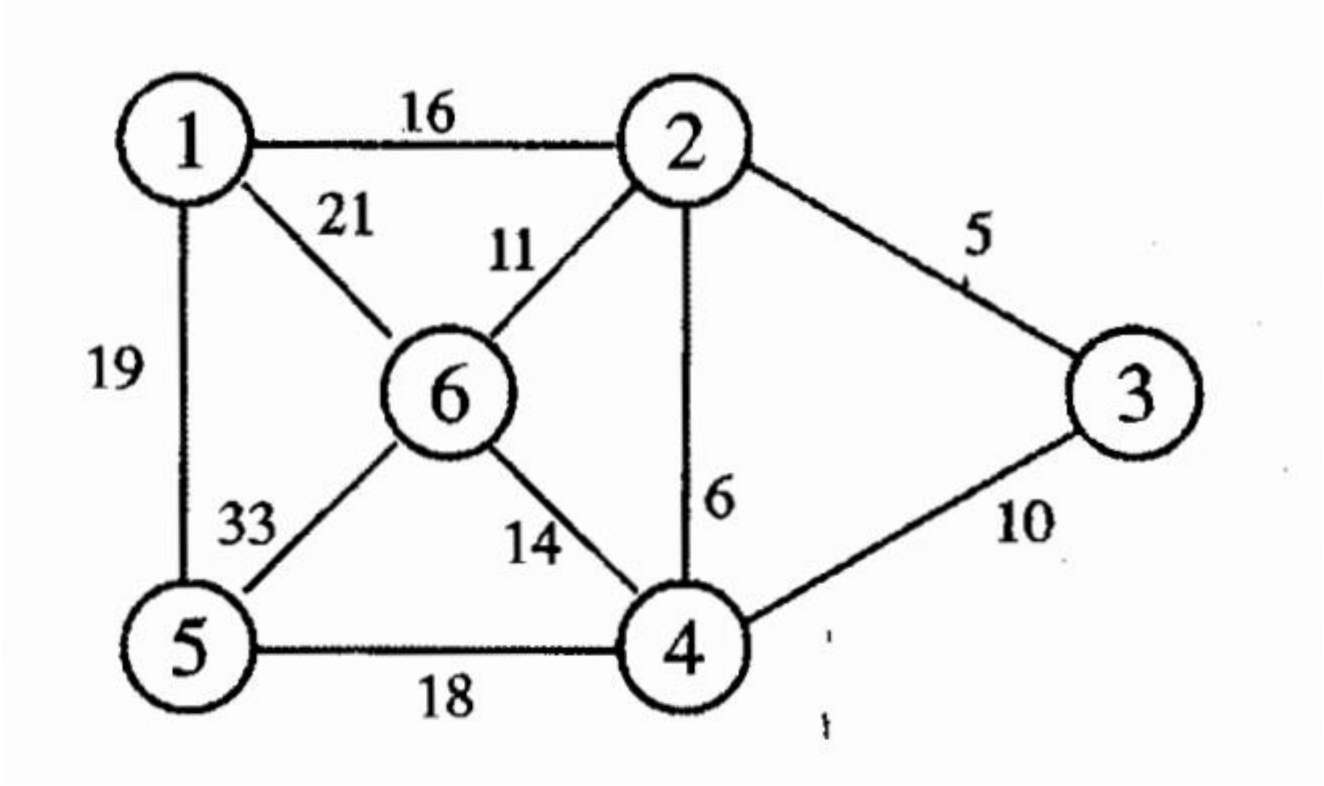
F = {A, B, C, D, E, F}

MST = {(A, D), (C, E), (E, F), (E, D), (A, B)}

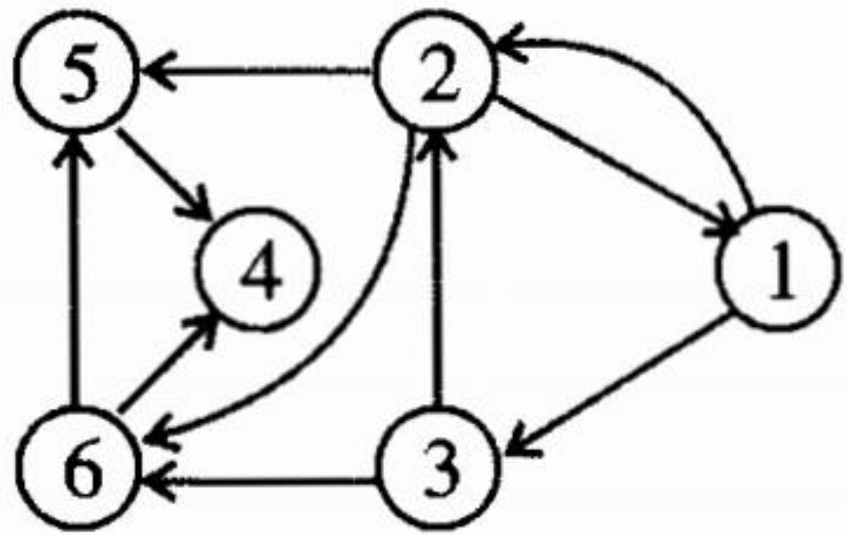
Q = {}

Exercise

- **Find MST**



Exercise



- Find Shortest Paths using Dijkstra's algorithm

Assignment

- WAP to
  - Initialize a undirected weighted graph G(V,E)
  - BSF algo
  - DSF algo
  - Find the MST using Kruskal’s algo
  - Find shortest paths from a source vertex to a destination vertex using Dijkstra’s algo

References

- Slide– Nguyễn Thanh Bình - ĐTVT
- Slide – Đỗ Bích Diệp
- Chapter 13 - Graph – Data structure in C – Rema Thareja – 2014
- Slide: CSE 680 Prof. Roger Crawfis