Optimization Theory and Methods

2025 Autumn

魏可佶 kejiwei@tongji.edu.cn
https://kejiwei.github.io/

- **Origins of Graph Theory**
  - Bridges of Königsberg
- **Analysis of Run Times**
- **Shortest Path Problem**
  - Dijkstra's algorithm
- **MST basics**
- **Kruskal's algorithm**
- **Prim's algorithm**

↳ Bridges of Königsberg (Euler 1736)

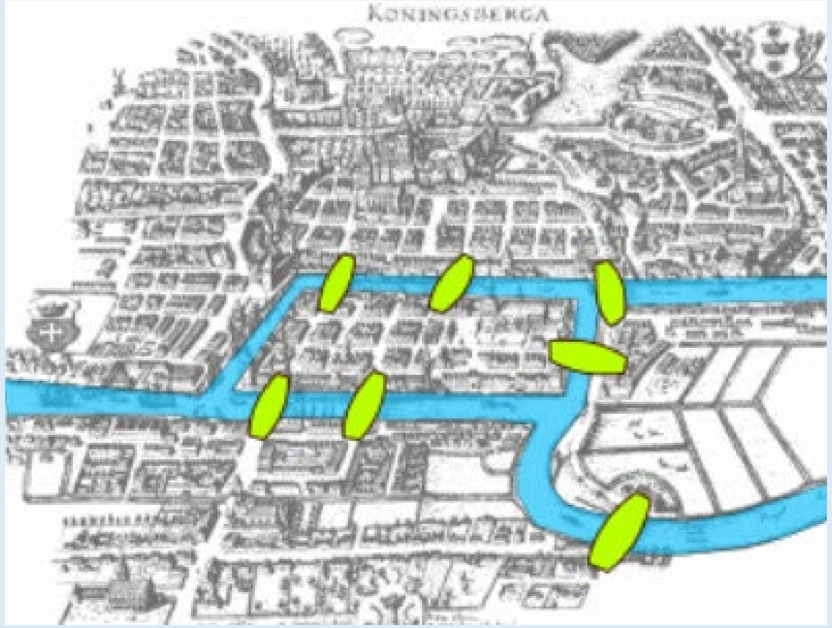
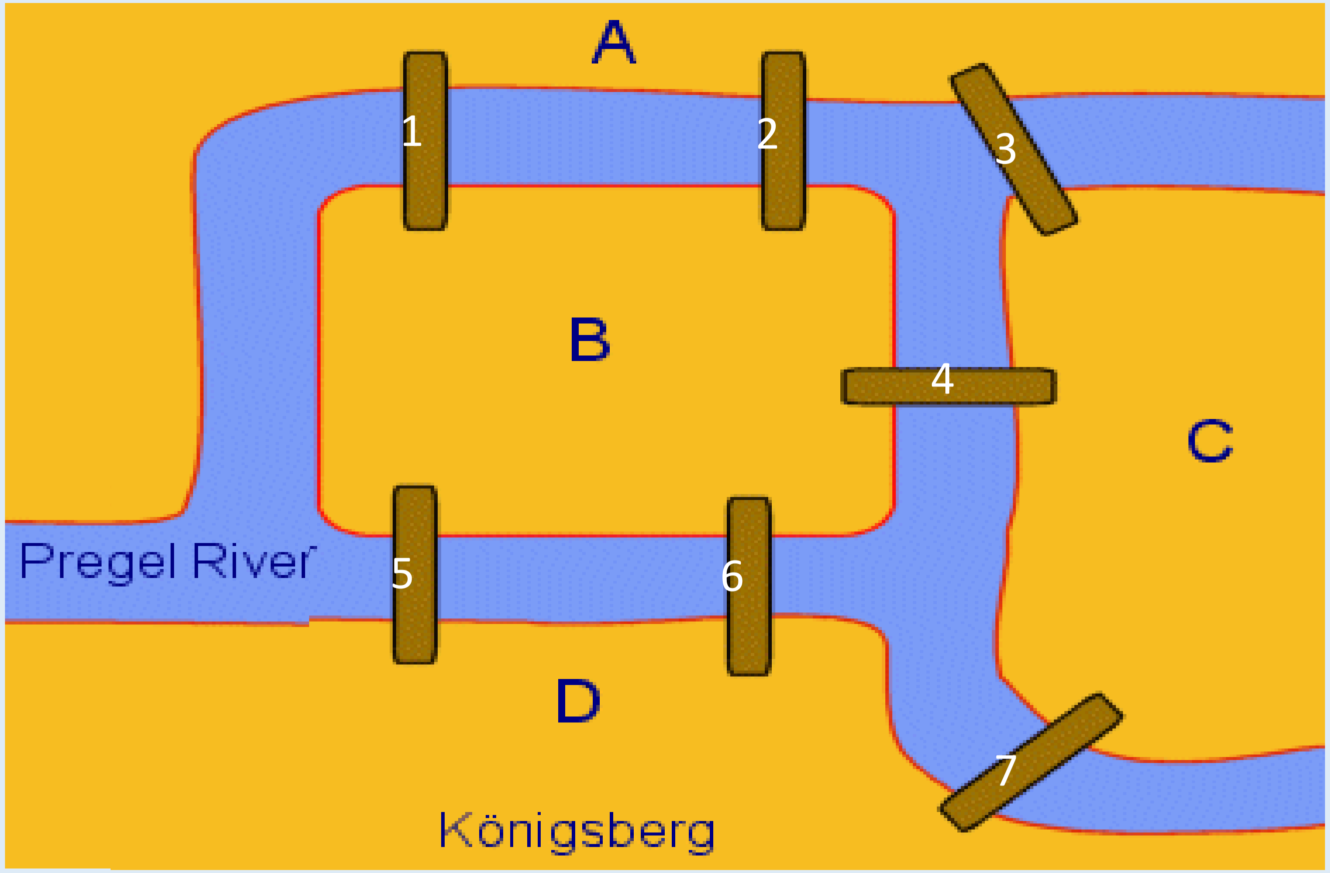


Image Source: http://www.mathsisfun.com/activity/seven-bridges-konigsberg.html

- Can you take a walk through this town such that you take each bridge exactly once and return to the point you started from?
- People often wondered, but usually thought it to be impossible.
- Leonhard Euler came along in 1736 and proved it impossible.
- Often considered the beginning of the discipline of graph theory!

同济经管 TONGJI SEM

- Is it possible to ***start at A***, cross each bridge exactly once and then end back at A?



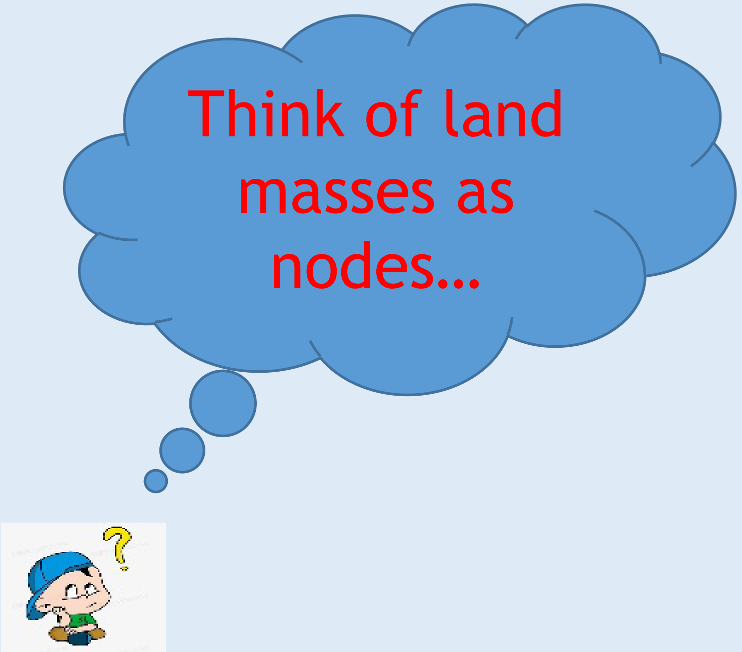


Image Source: http://www.infovis.net/imagenes/T1_N137_A4_Konigsberg_en.gif

同济经管 TONGJI SEM

- Think of land masses as *nodes* and bridges as *arcs* of the network.

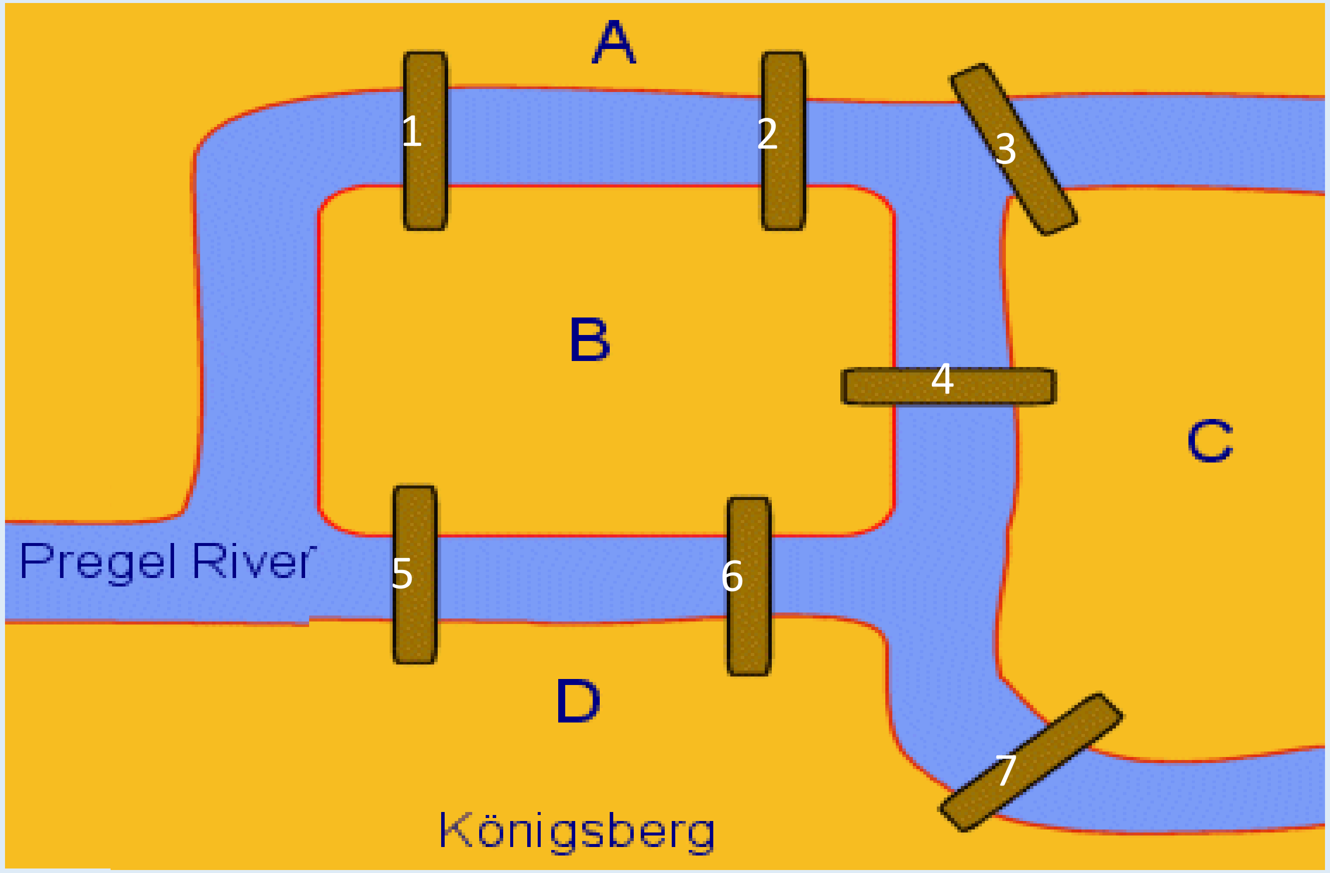
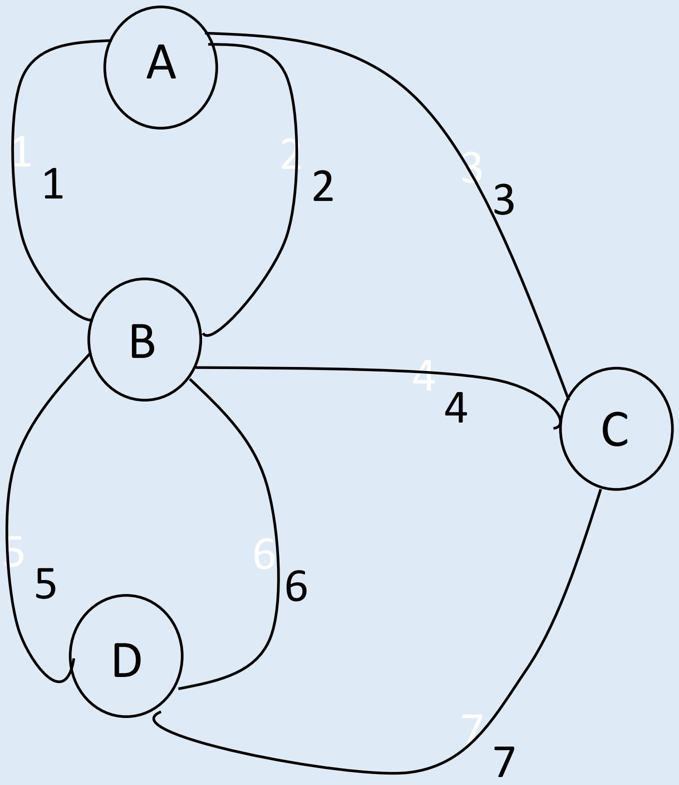


Image Source:
http://www.infovis.net/imagenes/T1_N137_A4_Konigsberg_en.gif

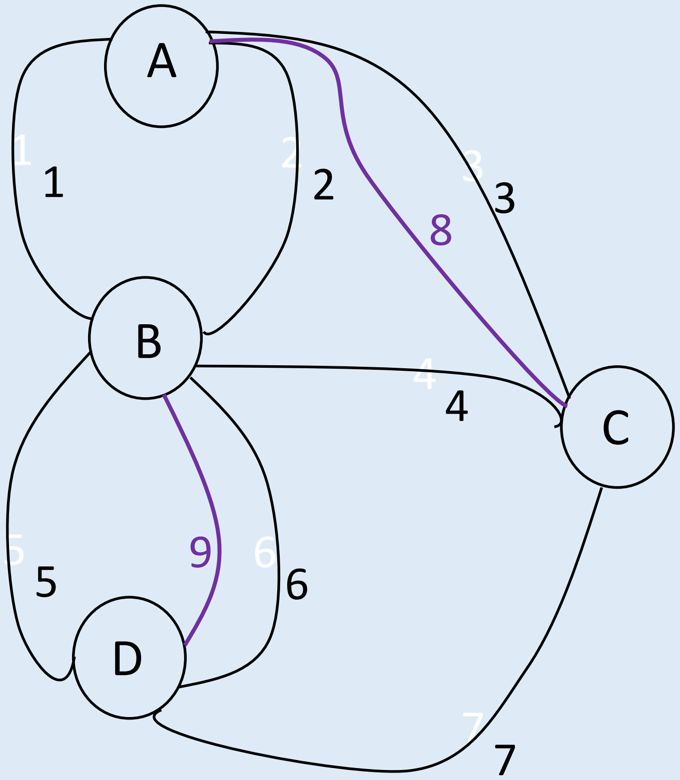
CAMEA 中国高质量MBA教育认证 AACSB ACCREDITED EQUIS ACCREDITED

↳ Bridges of Königsberg (Euler 1736) (cont.)

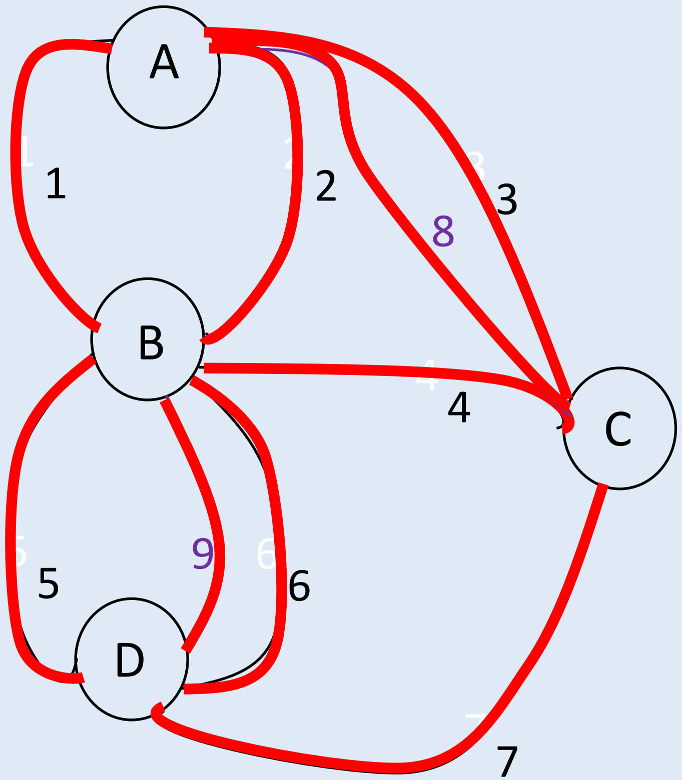


- Is there a walk starting from A, ending at A, and covering each arc exactly once?
- The answer is No!
- Why not?

- Now we can create a walk.



- The walk: A, 1, B, 5, D, 6, B, 4, C, 8, A, 3, C, 7, D, 9, B, 2, A



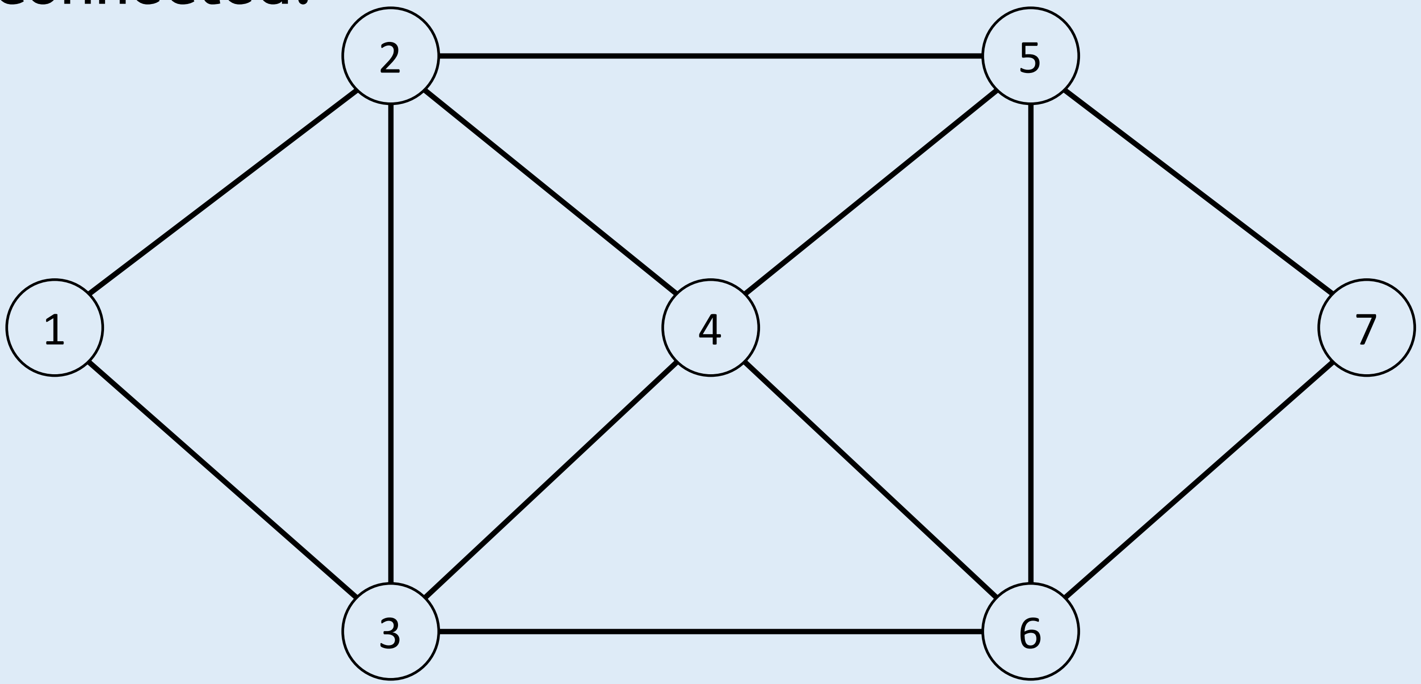
- Why does it work now and didn’t work before?

同济经管 TONGJI SEM

- **Closed Walk:** A walk that ends at the same node where it starts.
- **Eulerian Cycle**: A closed walk that passes along each arc exactly once.
- **Degree of a node**: Number of arcs incident to the node.
- **Theorem:** An undirected network has an Eulerian cycle if and only if the network is connected and every node has an even degree.
  - **Proof:** The degree of a node is twice the number of times it appears on the walk (except for the starting/ending node).

- How to find an Eulerian cycle once you know one exists?

- Start from node 1. Move along arcs while ensuring that the network is not disconnected.

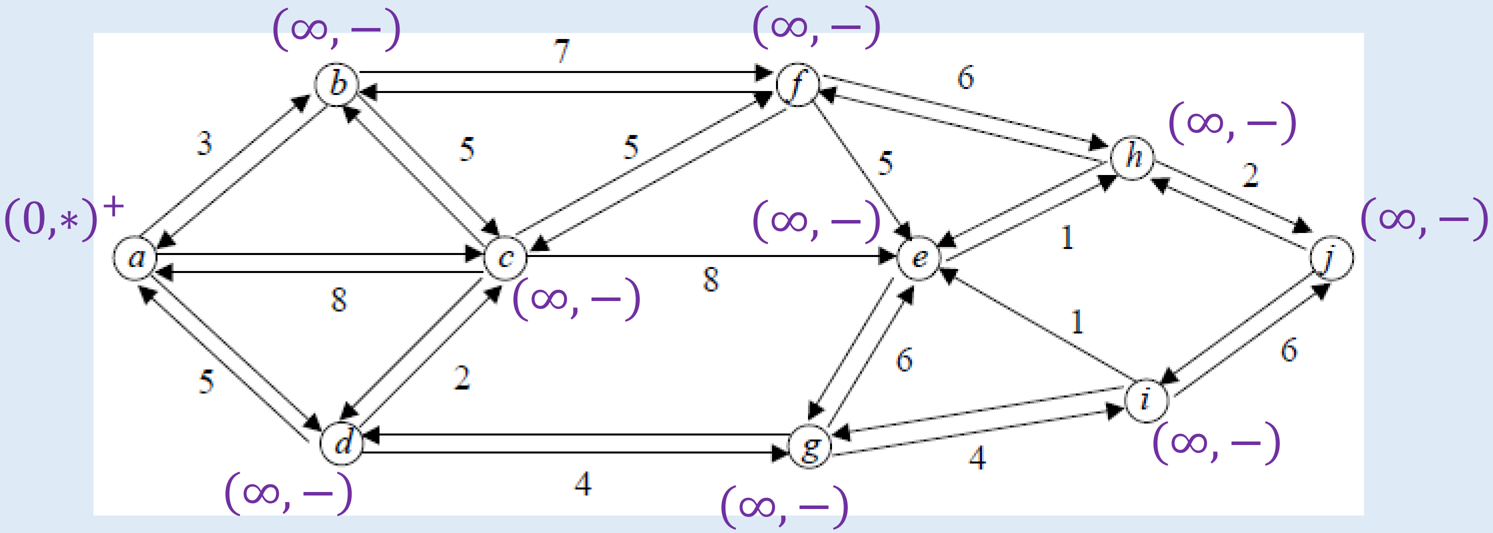


- Ensuring that is very difficult: Takes *too much time* to find out whether the network is connected, each time.

- **Objective:** Find the path of minimum cost (or length) from a specified source node s to another specified sink node t, assuming that each arc $(i,j) \in A$ has an associated cost (or length) $c_{ij} \geq 0$.
- **Applications:**
  - Project scheduling, cash flow management, message routing in communication systems, traffic flow through a congested city.
  - DNA sequence alignment.
  - Dynamic lot sizing in production and inventory management.
  - Approximating piecewise linear functions.
  - Optimizing the spacing between words in a text editor.
  - Telephone operator scheduling.
  - Optimizing the path of a postman, etc.

- Find shortest path from node s to all the other nodes in a directed network.
- Notation: At each iteration,
  - $d(j)$: Length of shortest path from s to j discovered so far.
  - $p(j)$: Immediate predecessor to node j on the shortest path from s to j discovered so far.
  - k: Last node selected by the algorithm so far.
- Step 1: Initialization
  - $d(s) = 0$; $p(s) =*$.
  - $d(j) = \infty$ and $p(j) = -$, for all other nodes $j \neq s$.
  - $k = s$.
  - All nodes except s are *open*. Node s is *closed* (the objective is to eventually close all nodes).

- Find the shortest path from node a to all other nodes.



- Initialization:

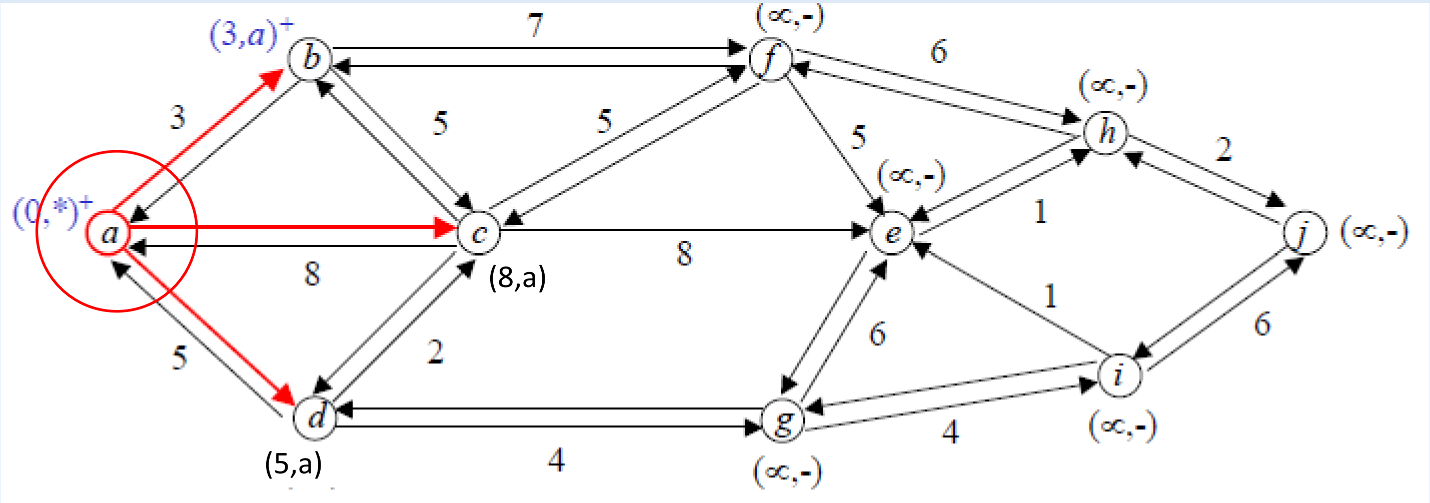
$$d(a) = 0, p(a) = *$$

$$d(k) = \infty, p(k) = -, \forall k \in N, k \neq a$$

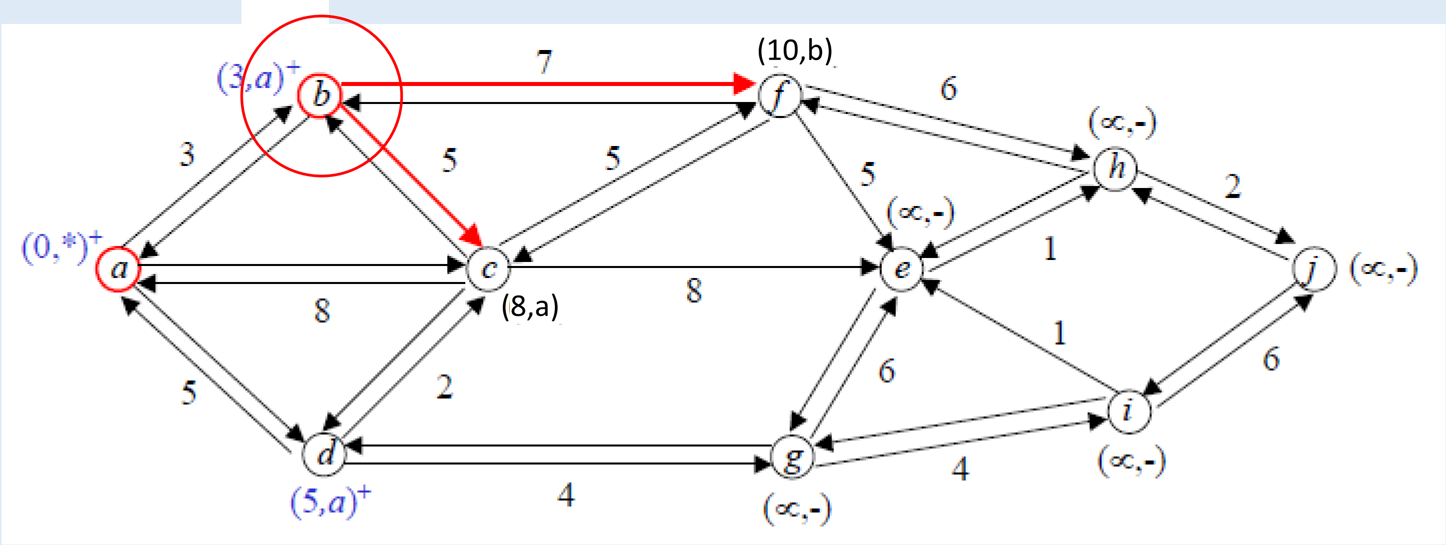
(Indicate closed nodes with '+' superscript.)

↳ Dijkstra's : Example (cont.)

- Iteration 1:

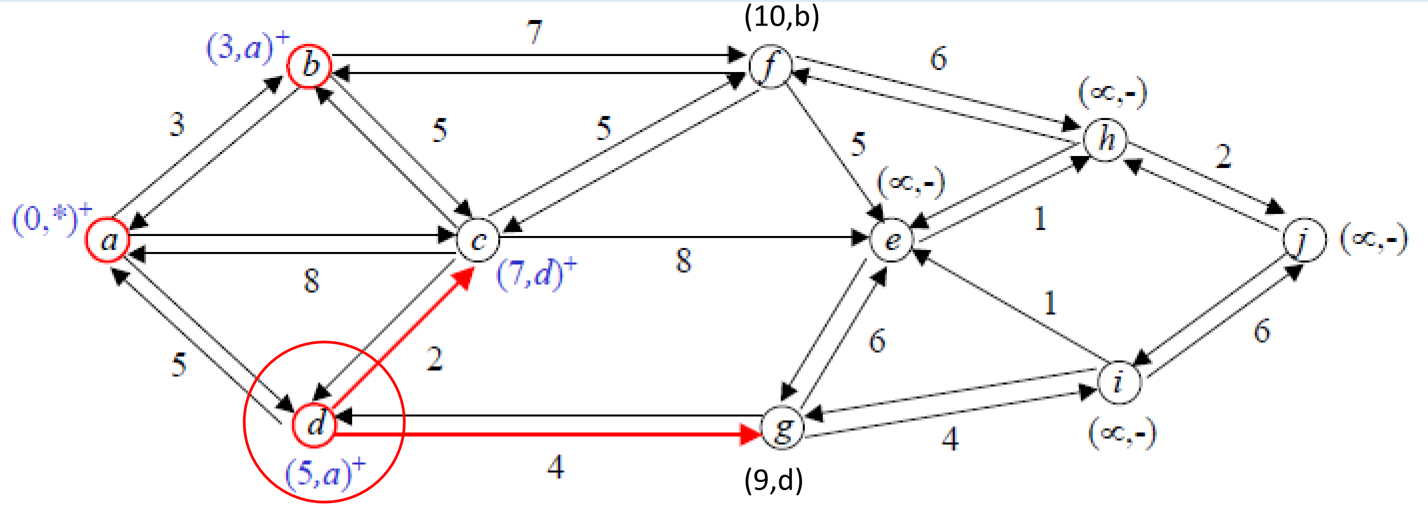


- Iteration 2:

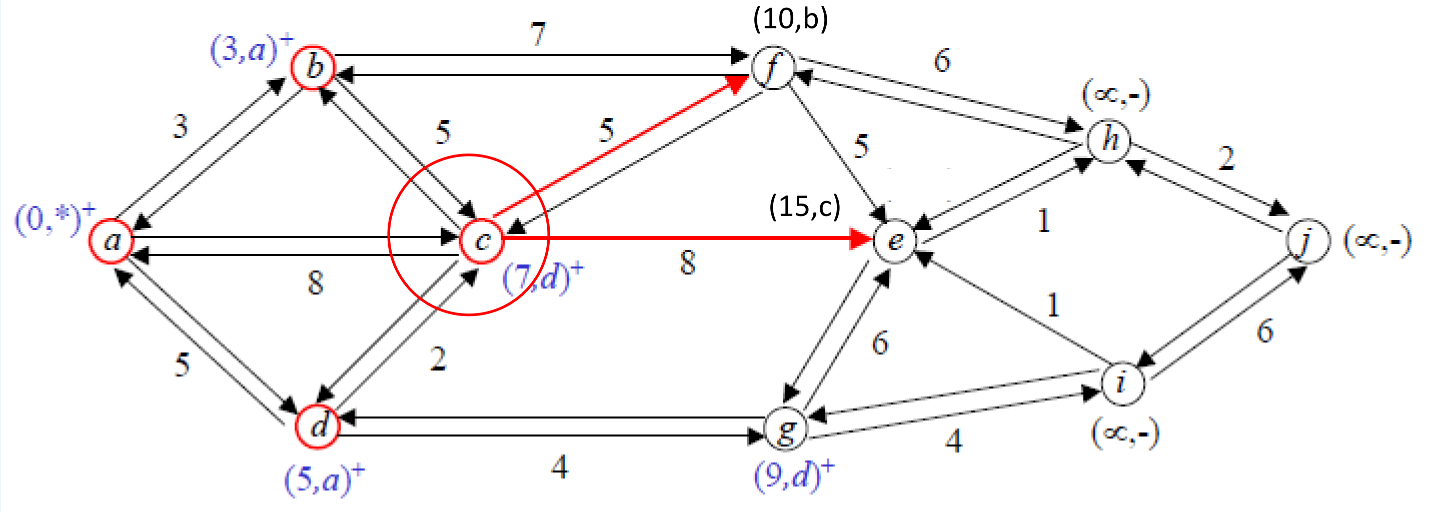


↳ Dijkstra's : Example (cont.)

- Iteration 3:

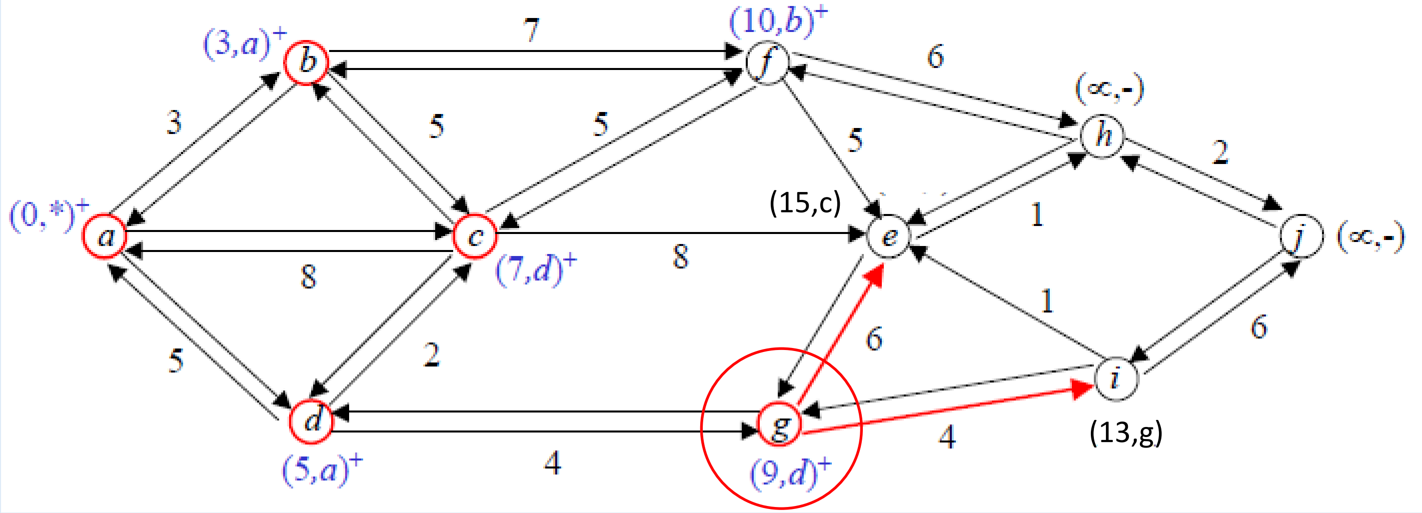


- Iteration 4:

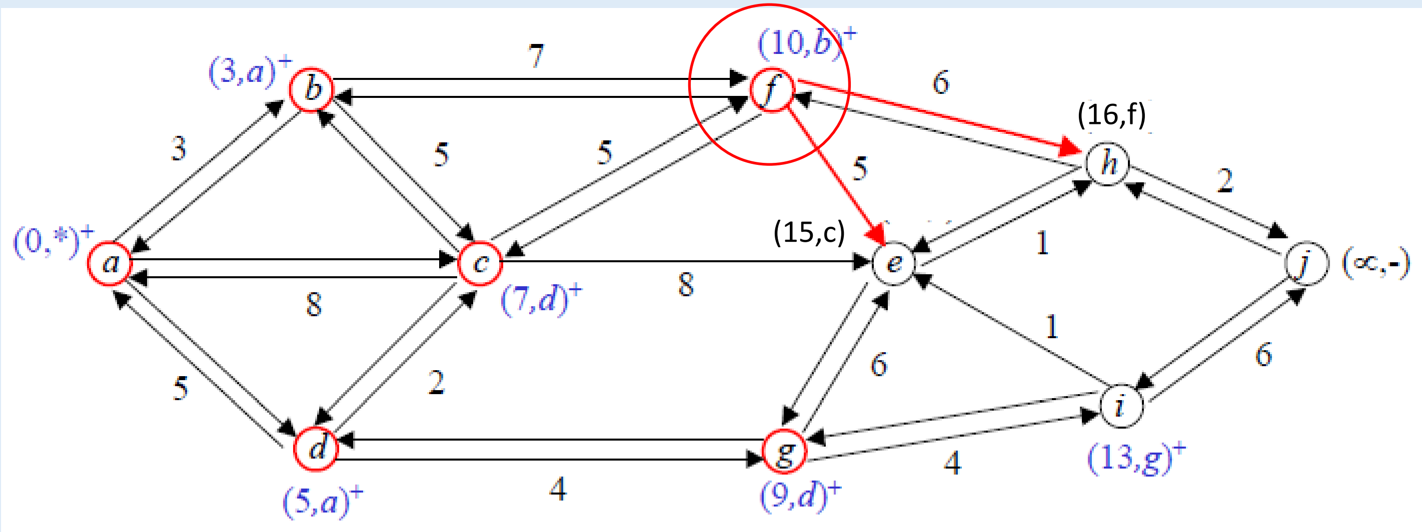


↳ Dijkstra's : Example (cont.)

- Iteration 5:

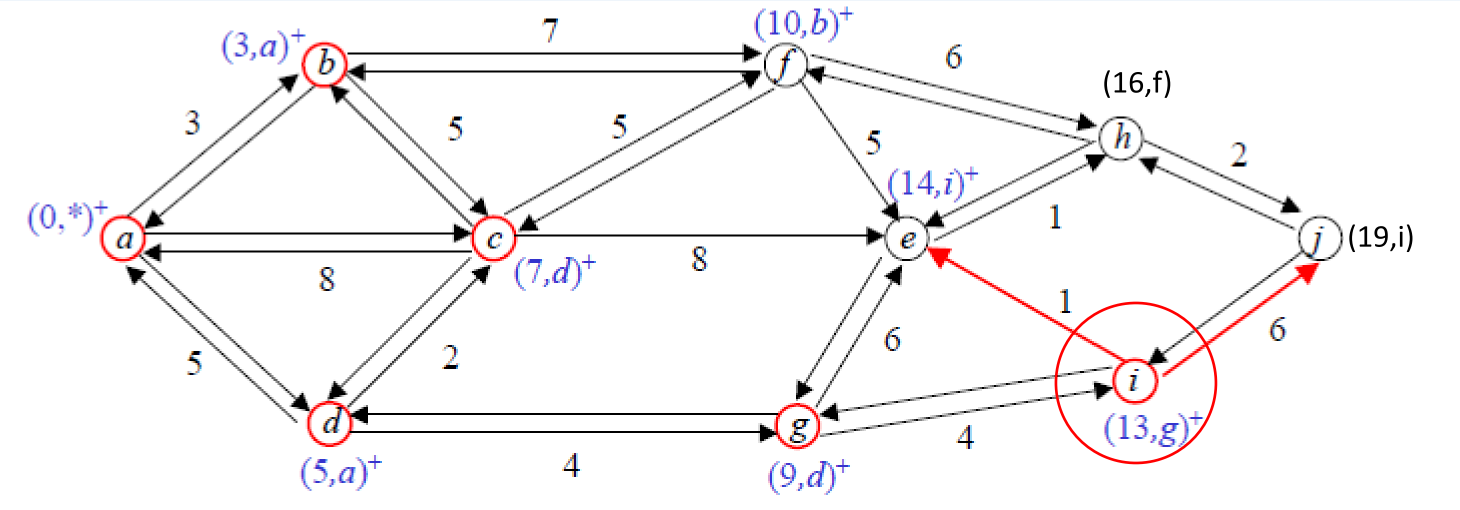


- Iteration 6:

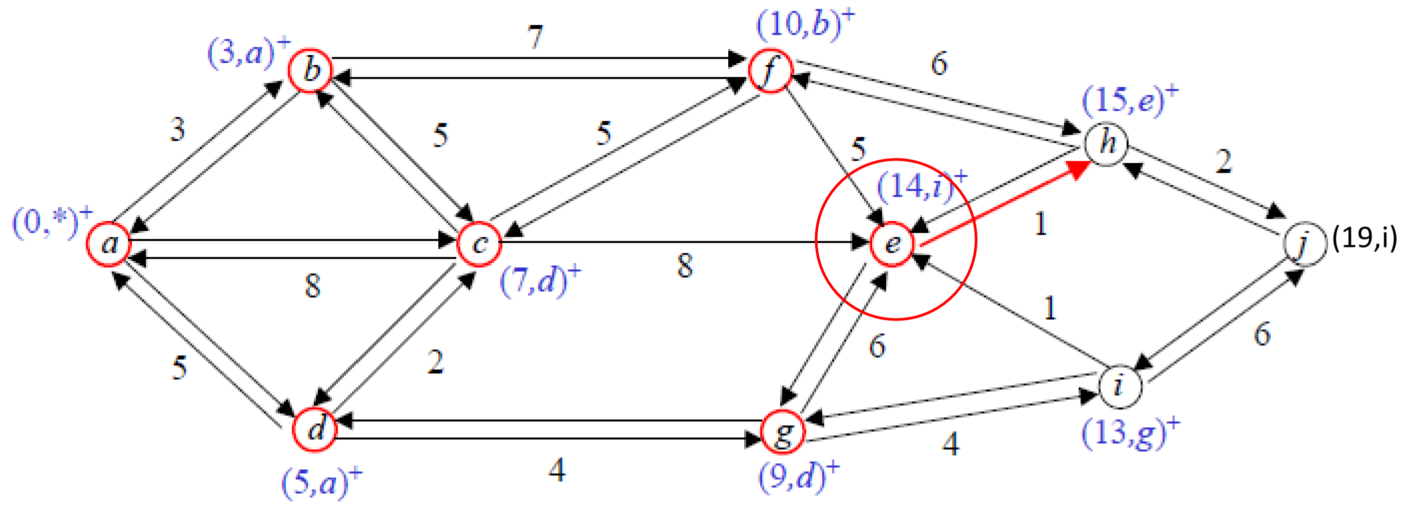


↳ Dijkstra's : Example (cont.)

- Iteration 7:

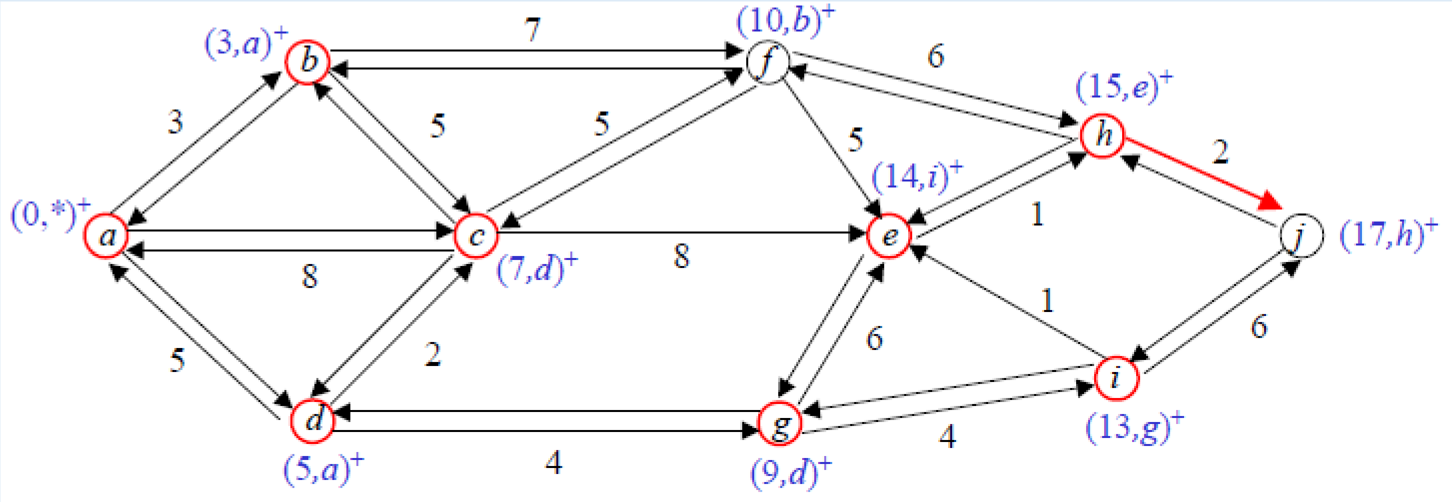


- Iteration 8:

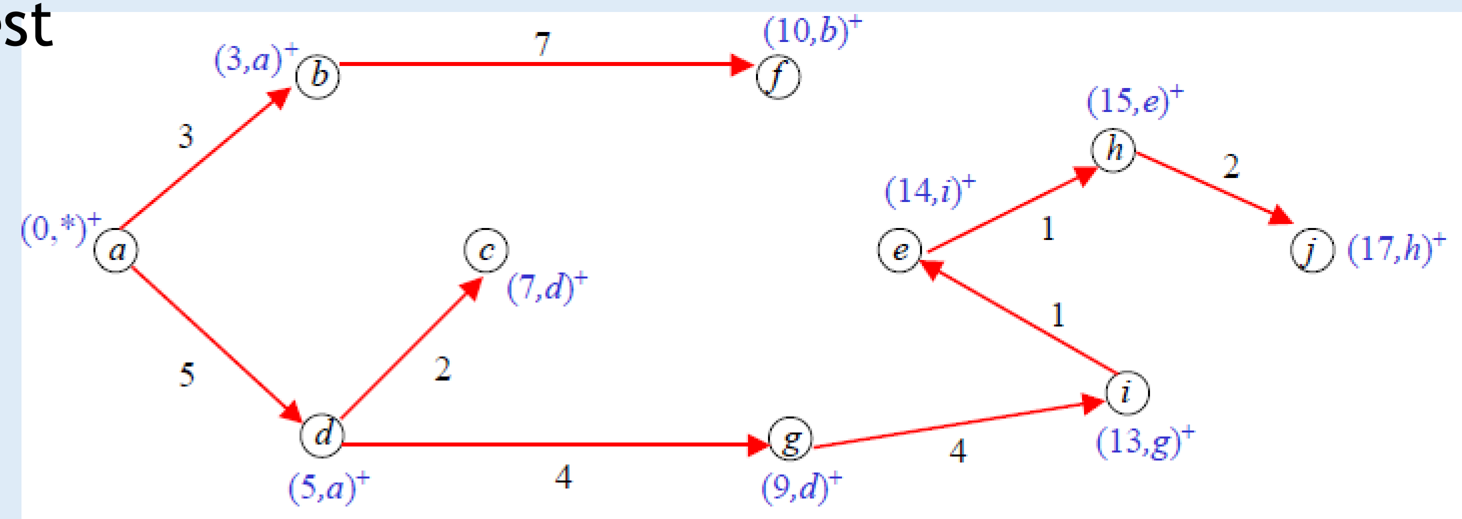


↳ Dijkstra's Algorithm: Example (cont.)

- Iteration 9:



- Final shortest path tree:

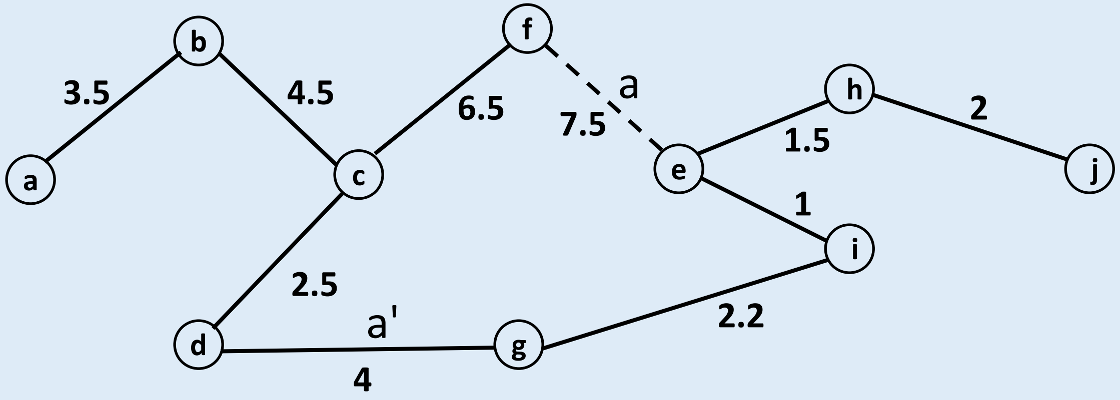


↳ Dijkstra's Algorithm

- Always results in a tree, called shortest path tree. (Why tree?)
- Very fast: Naïve implementation is $O(n^2)$: Because n iterations, and each iteration requires $O(n)$ operations.
- Improves to $O(m * \log(n))$ using efficient data structures.
- The algorithm (as stated) is not valid for $c_{ij} < 0$. (Why not?)
- Can be used to solve,
  - one-to-one
  - one-to-many
  - many-to-one

  … shortest path problems.
- **Interesting Fact**: Shortest path distance ($d(j)$) of each node j equals to the negative of the dual variable corresponding to the flow-balance constraint at that node.

- **An important fact:** If T is a spanning tree, and arc $a \notin T$, then adding a to T creates a unique cycle C. For any arc $a' \in C$, $T + a - a'$ is spanning tree.



↳ Why Kruskal's Algorithm gives optimal solution

- Suppose tree T created by Kruskal's algorithm is not optimal.
- Let T^* be the optimal tree.
- Let a be the first arc selected in T (by Kruskal's algorithm) that is not in T^*.
- So $T^* \cup \{a\}$ creates a unique cycle C.
- a was selected by Kruskal's, because it did not create a cycle containing exclusively the arcs with a lower cost than a.
- So at least one other arc in C has greater cost than that of a.
- So deleting the highest cost arc a' in C creates a tree $T' = T^* \cup \{a\} - \{a'\}$ with lower cost than T^*, leading to a contradiction.

- So this greedy algorithm *works*.

↳ Prim's Algorithm

- Start from any node and grow the tree greedily.
  - Consider all outgoing arcs from the tree nodes to the non-tree nodes.
  - Select the one with the least cost among them; add to the tree.
- Repeat until no non-tree nodes are left.

↳ Why Prim's Algorithm gives an optimal solution

- Suppose tree T is created by Prim's algorithm is not optimal.
- Let T^* be the optimal tree.
- Let a be the first arc selected in T (by the Prim's algorithm) that is not in T^*.
- So $T^* \cup \{a\}$ creates a unique cycle C and has at least one other arc that connects the nodes in Prim's tree at that time, to other nodes. Let that arc be a'. Cost of a' is greater than that of a.
- Deleting a' gives a new tree $T' = T^* \cup \{a\} - \{a'\}$ with lower cost than T^*, leading to a contradiction.

- So this greedy algorithm also *works*.

↳ Complexity of Prim's Algorithm

- Number of iterations =_____.
- Number of operations per iteration = _____: Spent to find the minimum cost arc connecting tree nodes to non-tree nodes.
- So, naïve implementation can take $O(\_\_\_)$ steps.
- With better data structures, the complexity improves to $O(m * \log(m))$.

Objective :

Key Concepts :