

Mathematical Programming II

Chapter 3: Network Optimization

Raquel Bernardino

rbernardino@iseg.ulisboa.pt
Office 511 Quelhas 6

Outline

Motivation

Introduction

Graph Theory

Minimum Spanning Tree Problem

Shortest Path Problem

Outline

Motivation

Introduction

Graph Theory

Minimum Spanning Tree Problem

Shortest Path Problem

Motivation

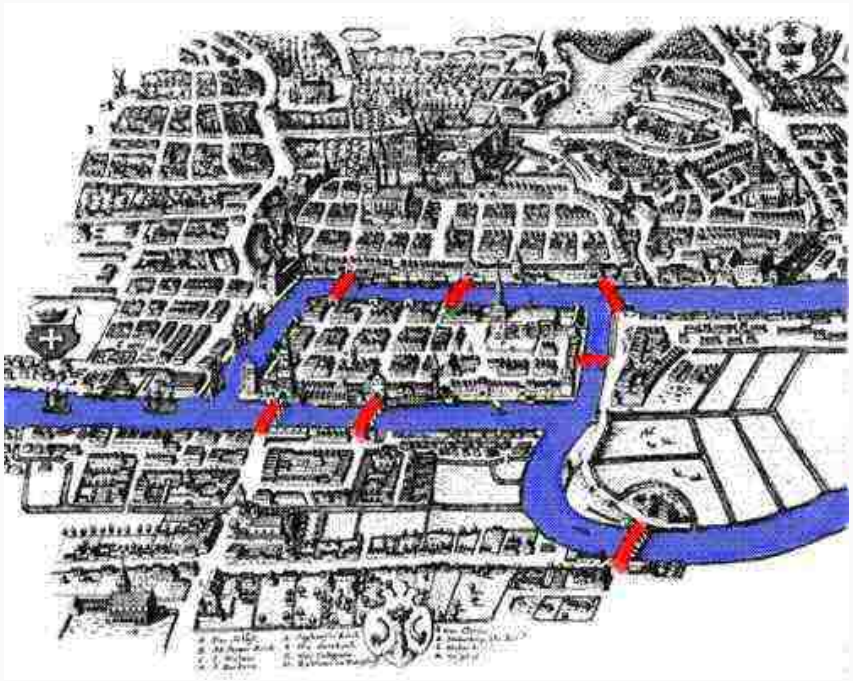


Figure 1: Bridges of Königsberg

- Is it possible to find a route that crosses every bridge exactly once and starts and ends at the same location?
  - No! This was proved by Leonhard Euler in 1736, in the first paper ever published in Graph Theory.

Outline

Motivation

Introduction

Graph Theory

Minimum Spanning Tree Problem

Shortest Path Problem

Introduction

Network optimization is used to solve problems that can be modelled as networks, comprising any system with interconnected components.

Solution algorithms can exploit this structure, thereby reducing computational time.

Historically, transportation problems have been a major application of network optimization and remain so today.

- Examples: Physical goods, communications, water, and electricity.

Introduction

- Application examples:
  - Design an offshore natural gas pipeline network connecting wells in the Gulf of Mexico to an onshore delivery point, minimizing pipeline construction costs.
  - Determine the shortest route between two cities in a road network.
  - Determine the maximum capacity of a coal pipeline network connecting coal mines in Wyoming to power plants in Houston.
  - Determine the schedule (start and completion dates) of activities in a construction project.
  - Determine the minimum-cost flow from oil fields to refineries through a pipeline network.

Outline

Motivation

Introduction

Graph Theory

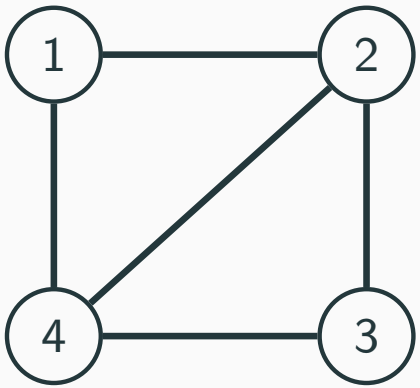
Minimum Spanning Tree Problem

Shortest Path Problem

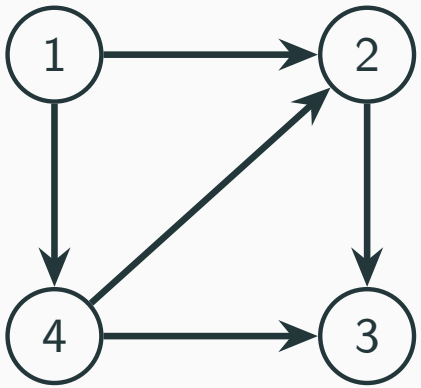
Definition

A graph is an ordered pair $G = (V, A)$, where V is a finite, non-empty set and $A \subseteq V \times V$.

The elements of V are called nodes (or vertices), and the elements of A are called edges or arcs.



(a) Undirected graph - edges



(b) Directed graph - arcs

Figure 2: Undirected graph *versus* directed graph

Graph Theory

Definition – Undirected Graphs

Consider an undirected graph $G = (V, A)$ and an edge $\{i, j\} \in A$. Then:

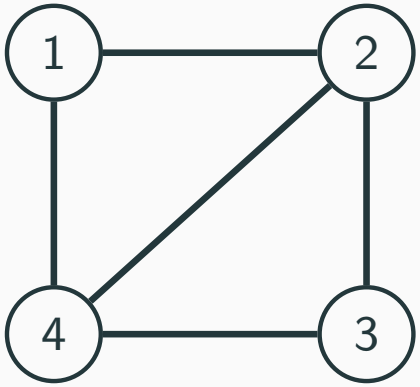
- i and j are the endpoints of edge $\{i, j\}$.
- i and j are adjacent nodes, and $\Gamma(i) = \{j \in V : j \text{ is adjacent to } i\}$ is the set of nodes adjacent to i.
- $|\Gamma(i)|$ is the degree of node i.

Definition – Directed Graphs

Consider a directed graph $G = (V, A)$ and an arc $(i, j) \in A$. Then:

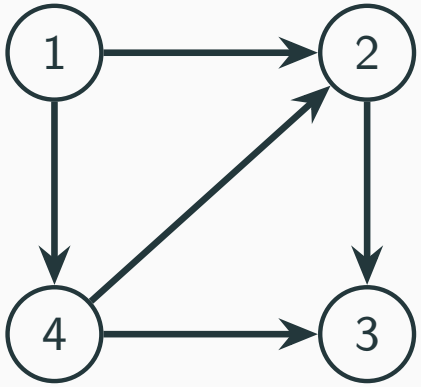
- i is the initial node and j is the terminal node of arc (i, j).
- j is a successor of i, and $\Gamma^+(i) = \{j \in V : j \text{ is a successor of } i\}$ is the set of successors of i.
- $|\Gamma^+(i)|$ is the out-degree of node i.
- i is a predecessor of j, and $\Gamma^-(j) = \{i \in V : i \text{ is a predecessor of } j\}$ is the set of predecessors of j.
- $|\Gamma^-(j)|$ is the in-degree of node j.

Example



Undirected graph

- $V = \{1, 2, 3, 4\}$
- $A = \{\{1, 2\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}$
- $\Gamma(1) = \{2, 4\}$ and $|\Gamma(1)| = 2$
- $\Gamma(2) = \{3, 4\}$ and $|\Gamma(2)| = 2$
- $\Gamma(3) = \{2, 4\}$ and $|\Gamma(2)| = 2$
- $\Gamma(4) = \{1, 2, 3\}$ and $|\Gamma(4)| = 3$



Directed graph

- $V = \{1, 2, 3, 4\}$
- $A = \{(1, 2), (1, 4), (2, 3), (4, 2), (4, 3)\}$
- $\Gamma^+(1) = \{2, 4\}$ and $\Gamma^-(1) = \emptyset$
- $\Gamma^+(2) = \{3\}$ and $\Gamma^-(2) = \{1, 4\}$
- $\Gamma^+(3) = \emptyset$ and $\Gamma^-(3) = \{2, 4\}$
- $\Gamma^+(4) = \{2, 3\}$ and $\Gamma^-(4) = \{1\}$

Definition – Undirected Graphs

Consider an undirected graph $G = (V, A)$. A chain is a sequence of edges such that one endpoint of an edge coincides with an endpoint of the next edge, i.e.,

$$\{i_1, i_2\}, \{i_2, i_3\}, \ldots, \{i_{k-1}, i_k\}.$$

Node i_1 is the initial node of the chain, and node i_k is its terminal node.

A cycle is a chain whose initial and terminal nodes coincide, i.e., $i_1 = i_k$.

Definition – Directed Graphs

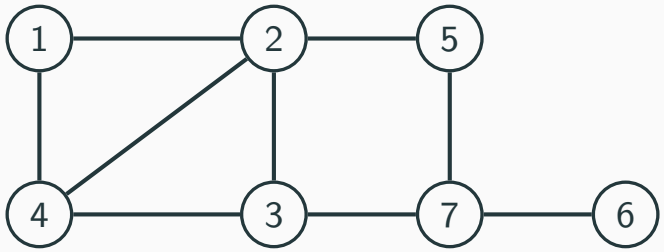
Consider a directed graph $G = (V, A)$. A path is a sequence of arcs such that the terminal node of each arc coincides with the initial node of the next arc, i.e.,

$$(i_1, i_2), (i_2, i_3), \ldots, (i_{k-1}, i_k).$$

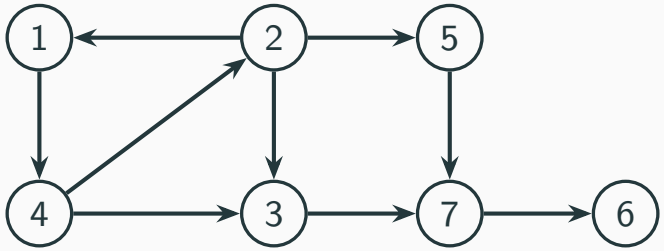
A circuit is a path whose initial and terminal nodes coincide, i.e., $i_1 = i_k$.

Example

Undirected graph



Directed graph



Examples:

- Chains: $\{\{1,2\},\{2,3\},\{3,7\}\}$ or $\{\{1,4\},\{4,2\},\{2,5\}\}$
- Cycles: $\{\{1,2\},\{2,4\},\{4,1\}\}$ or $\{\{2,5\},\{5,7\},\{7,3\},\{3,2\}\}$

Examples:

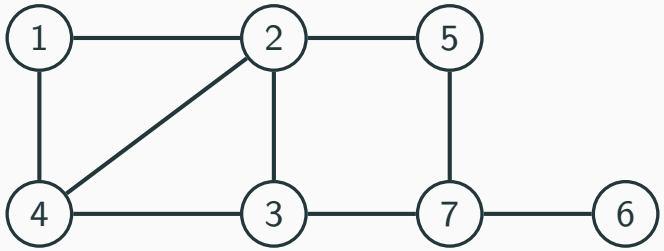
- Paths: $\{(1,4),(4,3),(3,7),(7,6)\}$ or $\{(2,1),(1,4),(4,3)\}$
- Circuit: $\{(1,4),(4,2),(2,1)\}$

Definition

A chain/cycle (path/circuit) is said to be elementary if it does not repeat nodes.

A chain/cycle (path/circuit) is said to be simple if it does not repeat edges (arcs).

- Example:



- $\{\{1,2\},\{2,3\},\{3,7\}\} \rightarrow$ Simple and elementary chain
- $\{\{2,4\},\{4,3\},\{3,2\},\{2,5\},\{5,7\}\} \rightarrow$ Simple but non- elementary
- $\{\{2,4\},\{4,3\},\{3,2\},\{2,4\},\{4,1\}\} \rightarrow$ Neither simple nor elementary chain

Definition

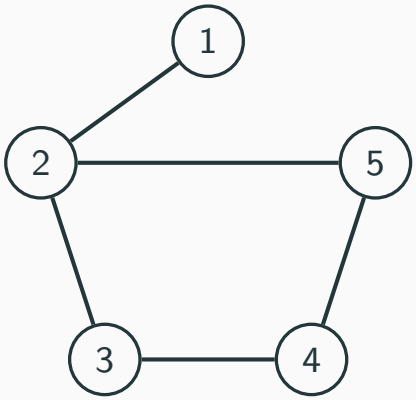
An undirected graph is said to be connected if there is a chain between every pair of nodes. Otherwise, it is said to be disconnected.

Definition

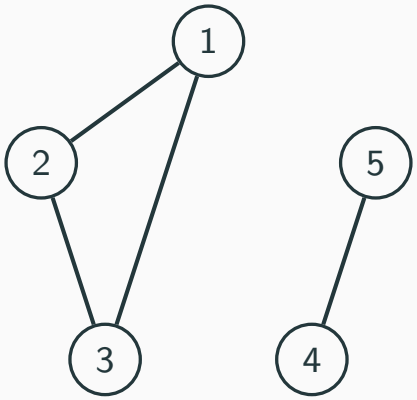
A directed graph is said to be connected if the undirected graph obtained by ignoring the directions of its arcs is connected.

A directed graph is said to be strongly connected if there is a path between every pair of nodes.

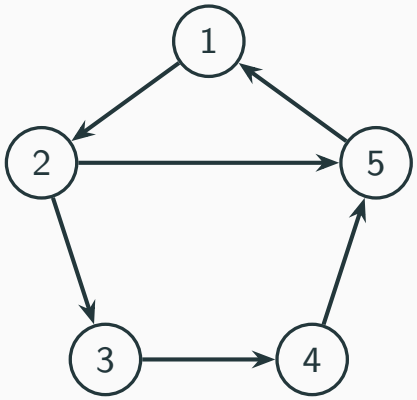
Example



Connected graph



Disconnected graph



Strongly connected graph

Definition

Consider a connected undirected graph $G = (X, A)$.

(i) A cycle (or circuit) C is said to be Hamiltonian if it is elementary and contains all the nodes of G.

(ii) A graph G is said to be Hamiltonian if it contains a Hamiltonian cycle (or circuit).

■ The Travelling Salesman Problem consists of finding a minimum-cost Hamiltonian cycle (or circuit).

Definition

Consider a connected undirected graph $G = (X, A)$.

(i) A cycle C is said to be *Eulerian* if it traverses every edge of G exactly once.

(ii) A graph G is said to be *Eulerian* if it contains an Eulerian cycle.

Eulerian Graphs

Result

Let $G = (V, A)$ be an undirected graph in which every node has degree at least two. Then G contains a cycle with at least three nodes.

Proof.

Let $i_1 \in V$. Construct a chain by choosing, at each step, a node adjacent to the current node and different from the preceding one. This is always possible because every node has degree at least two. Since V is finite, the sequence must encounter a previously visited node, thus yielding a cycle in G. □

■ This result will be used to prove the necessary and sufficient condition for a graph to be Eulerian.

Eulerian Graphs

Result

Let $G = (V, A)$ be a connected undirected graph. The graph G is Eulerian if and only if every node has even degree.

Proof.

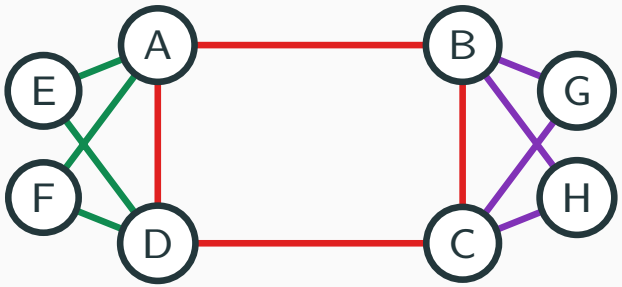
Eulerian $\implies$ Even degree. The graph contains an Eulerian cycle. Every time the cycle passes through a node, it contributes 2 to the degree of that node. Since every edge in A belongs to the Eulerian cycle, the degree of every node is even.

Even degree $\implies$ Eulerian. Assume that the result holds for every graph with fewer than $|A|$ edges. Since every node has degree at least 2, G contains a cycle C. Suppose that C does not contain every edge of A. Consider the graph $H = (V, A \setminus C)$, in which every node still has even degree. Let $H_1, \ldots, H_p$ be the connected components of H containing at least one edge. By the induction hypothesis, each such component contains an Eulerian cycle C_k. Each component H_k has at least one node i_k in common with the original cycle C. Without loss of generality, suppose that, when traversing C, these nodes are encountered in the order $i_1, \ldots, i_p$. Start at i_1 and traverse C_1, returning to i_1. Continue along C until reaching i_2, then traverse C_2 and return to i_2. Repeat this process until returning to the starting node i_1. The resulting cycle is Eulerian. $\square$

Example: Eulerian $\Longrightarrow$ Even degree

Consider the Eulerian cycle

$$A \to E \to D \to F \to A \to B \to G \to C \to H \to B \to C \to D \to A.$$



At each node, the incident edges are paired: one entering and one leaving.

At the starting node A, the last edge DA pairs with the first edge AE.

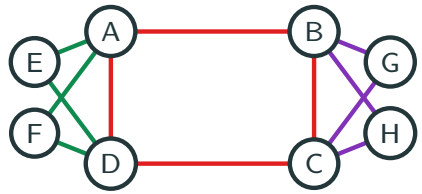
Node	Pairs of incident edges	Degree
A	(DA, AE), (FA, AB)	4
B	(AB, BG), (HB, BC)	4
C	(GC, CH), (BC, CD)	4
D	(ED, DF), (CD, DA)	4
E	(AE, ED)	2
F	(DF, FA)	2
G	(BG, GC)	2
H	(CH, HB)	2

Every edge is used exactly once, so all incident edges belong to pairs. Hence every node has even degree.

Example: Even degree $\Longrightarrow$ Eulerian

Let G be connected, with every node of even degree. Choose $C = (A, B, C, D, A)$. Removing its edges leaves two components.

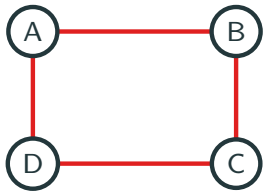
1. The graph G



$\deg(A) = \deg(B) = \deg(C) = \deg(D) = 4$

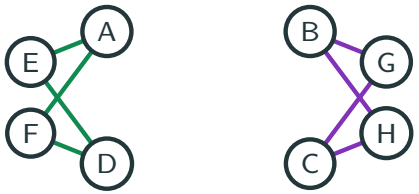
$\deg(E) = \deg(F) = \deg(G) = \deg(H) = 2$

2. A cycle C in G



$C = (A, B, C, D, A)$

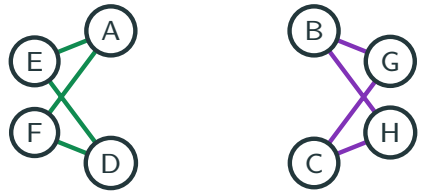
3. $H = (V, A \setminus C)$



H_1: A, E, D, F

H_2: B, G, C, H

4. Cycles in the components



$C_1 = (A, E, D, F, A)$

$C_2 = (B, G, C, H, B)$

5. Combining the cycles

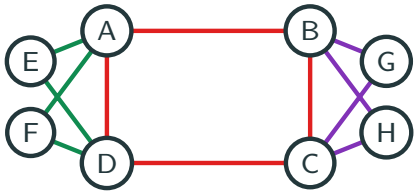
1. Start at A and traverse C_1, returning to A.

2. Follow AB to B; traverse C_2, returning to B.

3. Follow BC, CD and DA to return to A.

No edge is repeated.

6. An Eulerian cycle in G



$A \to E \to D \to F \to A \to B \to G \to C \to H \to B \to C \to D \to A$

All 12 edges are traversed exactly once.

Example

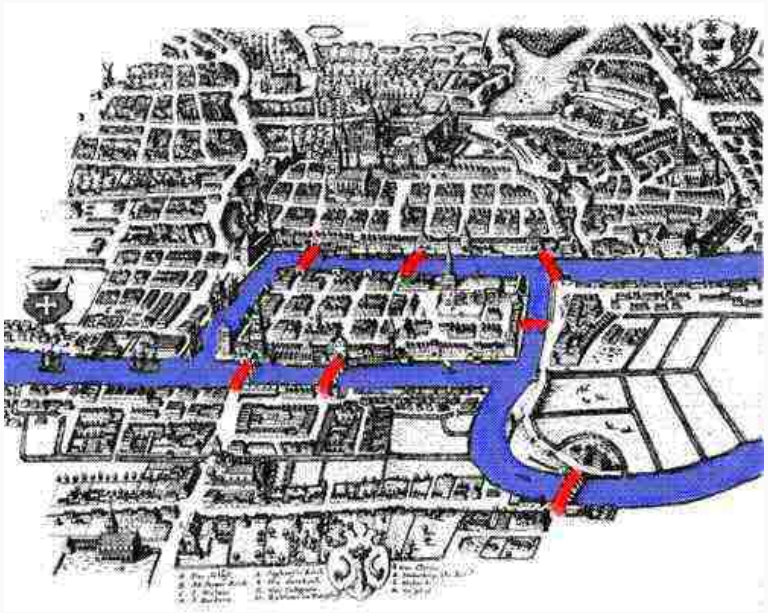


Figure 4: Bridges of Königsberg

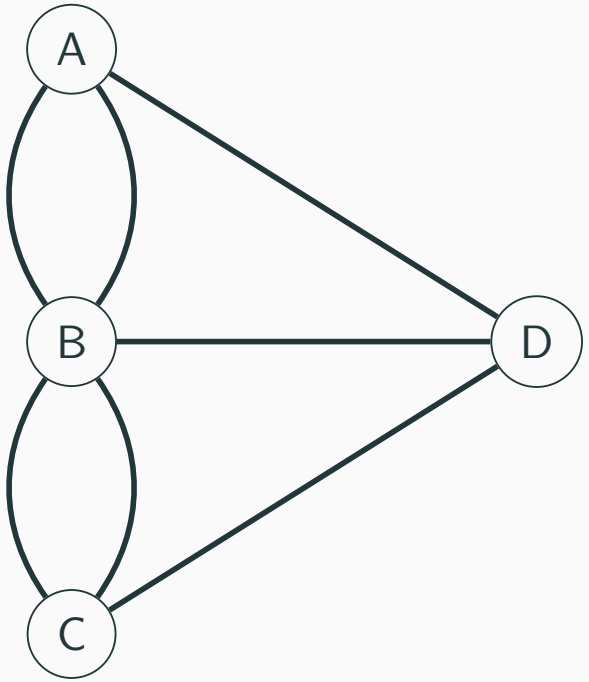


Figure 5: Graph representation of the Königsberg bridges

- The graph is not Eulerian because some nodes have odd degree. Therefore, there is no cycle that crosses every bridge exactly once.

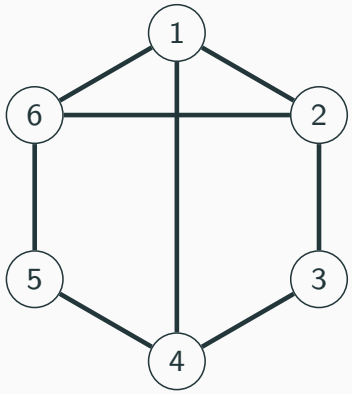
Definition

Let $G = (V, A)$ be a graph. A subgraph of G is a graph of the form $G_S = (V_S, A_S)$, where $V_S \subseteq V$ and $A_S = \{\{i, j\} \in A : i, j \in V_S\}$.

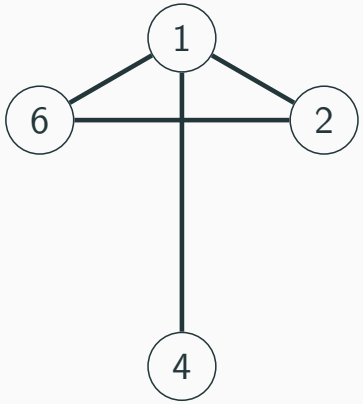
Definition

Let $G = (V, A)$ be a graph. A partial graph of G is a graph of the form $G_P = (V, A_P)$, where $A_P \subseteq A$.

Example

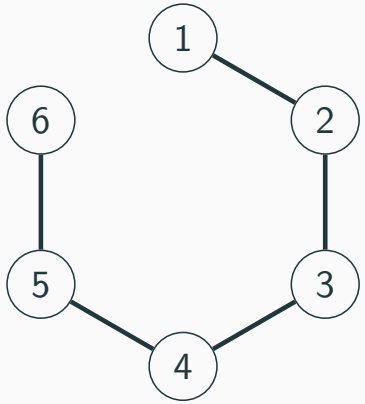


Original Graph $G = (V, A)$



Subgraph $G_S = (V_S, A_S)$

- $V_S = \{1, 2, 4, 6\}$
- $A_S = \{\{1, 2\}, \{1, 6\}, \{2, 6\}, \{4, 1\}\}$



Partial graph $G_P = (V, A_P)$

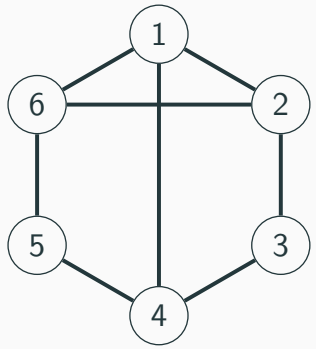
- $V_S = V$
- $A_P = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{5, 6\}\}$

Definition

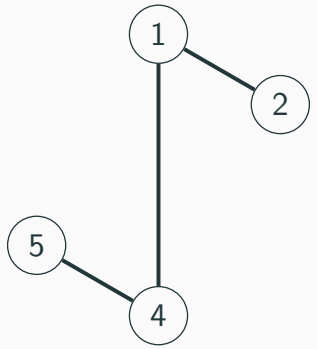
A tree is a connected graph with no cycles.

Definition

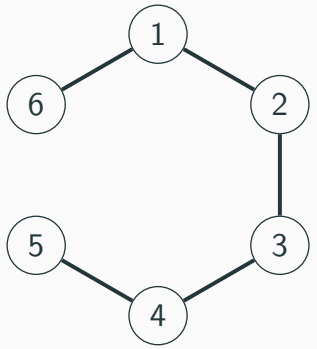
Let $G = (V, A)$ be an undirected graph. A spanning tree of G is a connected, acyclic partial graph of G.



Original graph $G = (V, A)$



Tree



Spanning tree of G

Result

A spanning tree of a graph $G = (V, A)$ has $|V| - 1$ edges.

Proof.

The proof is by induction on the number of nodes of the graph.

Case $|V| = 1$: Immediate, since a graph with one node has no edges.

General case: Assume that the result holds for every graph with fewer than $|V|$ nodes.

Consider a spanning tree T of G. Removing one edge of T produces two trees T_1 and T_2. Let V_1 and V_2 denote their respective node sets. Notice that V_1 and V_2 form a partition of V.

By the induction hypothesis, T_1 has $|V_1| - 1$ edges and T_2 has $|V_2| - 1$ edges. Together they have $|V_1| + |V_2| - 2 = |V| - 2$ edges. Adding back the edge removed from T, we obtain $|V| - 2 + 1 = |V| - 1$ edges in T. □

Outline

Motivation

Introduction

Graph Theory

Minimum Spanning Tree Problem

Shortest Path Problem

Minimum Spanning Tree Problem

Definition

Consider an undirected graph $G = (V, A)$. Each edge $\{i, j\} \in A$ has an associated value c_{ij} and let T be a spanning tree of G. The cost of T is the sum of the values c_{ij} associated with its edges, that is,

$$Cost(T) = \sum_{\{i,j\} \in T} c_{ij}.$$

Among all spanning trees of G, the one with the lowest cost is called a minimum spanning tree.

Minimum Spanning Tree Problem

Two algorithms are available to determine the minimum spanning tree:

- Kruskal's Algorithm
  - Based on the cycle optimality condition
- Prim's Algorithm
  - Based on the cut optimality condition

Cycle Optimality Condition

General idea:

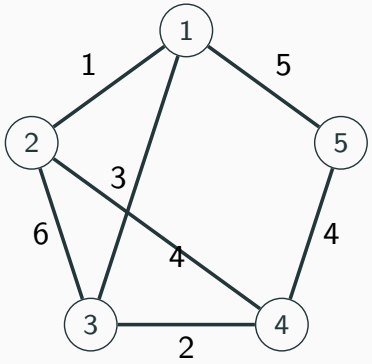
Adding any edge to a spanning tree creates a cycle. The edge that does not belong to the minimum spanning tree is the most expensive edge in the cycle.

Result

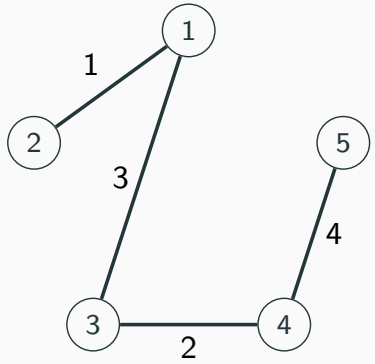
A spanning tree $T = (V_T, A_T)$ is optimal if and only if

$$\forall \{i,j\} \notin A_T, \forall \{u,v\} \in cycle(V_T \cup \{i,j\}), c_{ij} \geq c_{uv}.$$

Example



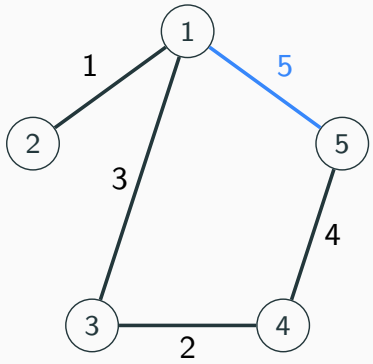
Original graph G



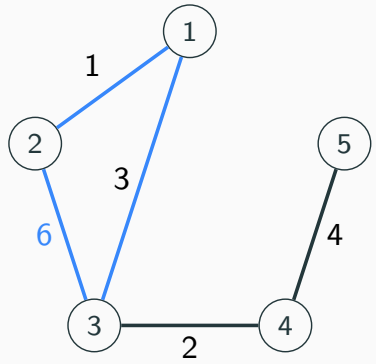
Minimum spanning tree T of G

Edge	Cost
$\{1, 5\}$	5
$\{2, 3\}$	6
$\{2, 4\}$	4

Edges $\notin T$



Edge $\{1, 5\}$ is the most expensive edge in the cycle $(1, 5, 4, 3, 1)$.



Edge $\{2, 3\}$ is the most expensive edge in the cycle $(1, 3, 2, 1)$.

Kruskal's Algorithm

Kruskal's Algorithm

1: Initialize $V_T \leftarrow \emptyset$, $A_T \leftarrow \emptyset$, $\overline{A_T} \leftarrow A$ and $Cost \leftarrow 0$.
2: **while** $|A_T| \neq |V| - 1$ **do**
3: Select the minimum-cost edge $\{i^*, j^*\}$ in $\overline{A_T}$.
4: **if** $A_T \cup \{i^*, j^*\}$ does not contain a cycle **then**
5: Set $V_T \leftarrow V_T \cup \{i^*, j^*\}$, $A_T \leftarrow A_T \cup \{i^*, j^*\}$ and $Cost \leftarrow Cost + c_{ij}$.
6: **end if**
7: Set $\overline{A_T} \leftarrow \overline{A_T} \setminus \{i^*, j^*\}$.
8: **end while**

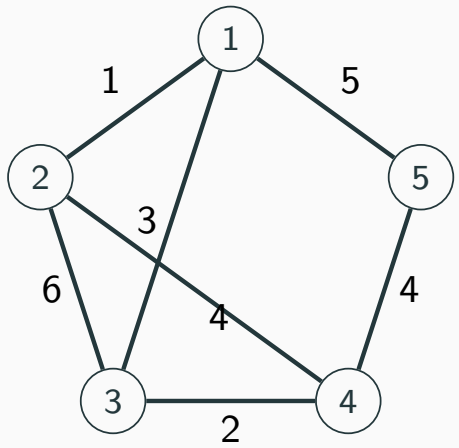
A minimum spanning tree of G is $T = (V_T, A_T)$.

Example

Execution of Kruskal's algorithm

Initialization: $V_T \leftarrow \emptyset$, $A_T \leftarrow \emptyset$, $\overline{A_T} \leftarrow A$ and $Cost \leftarrow 0$.

- $V \setminus V_T \neq \emptyset$? Yes.
  - $\{2,1\}$ is the minimum-cost edge $\overline{A_T}$.
  - $A_T \cup \{2,1\}$ contains a cycle? No.
    - $V_T = \{2,1\}$, $A_T = \{\{2,1\}\}$ and $Cost = 1$.
  - $\overline{A_T} \leftarrow \overline{A_T} \setminus \{2,1\}$.
- $V \setminus V_T = \{3,4,5\} \neq \emptyset$? Yes.
  - $\{3,4\}$ is the minimum-cost edge $\overline{A_T}$.
  - $A_T \cup \{3,4\}$ contains a cycle? No.
    - $V_T = \{2,1,3,4\}$, $A_T = \{\{2,1\},\{3,4\}\}$ and $Cost = 3$.
  - $\overline{A_T} \leftarrow \overline{A_T} \setminus \{3,4\}$.



- $V \setminus V_T = \{5\} \neq \emptyset$? Yes.
  - $\{1,3\}$ the minimum-cost edge $\overline{A_T}$.
  - $A_T \cup \{1,3\}$ contains a cycle? No.
    - $V_T = \{2,1,3,4\}$, $A_T = \{\{2,1\},\{3,4\},\{2,4\}\}$ and $Cost = 6$.
  - $\overline{A_T} \leftarrow \overline{A_T} \setminus \{1,3\}$.

Example

- $V \setminus V_T = \{5\} \neq \emptyset$? Yes.
  - $\{2,4\}$ is the minimum-cost edge $\overline{A_T}$.
  - $A_T \cup \{2,4\}$ contains a cycle? Yes.
  - $\overline{A_T} \leftarrow \overline{A_T} \setminus \{2,4\}$.
- $V \setminus V_T = \{5\} \neq \emptyset$? Yes.
  - $\{4,5\}$ the minimum-cost edge $\overline{A_T}$.
  - $A_T \cup \{4,5\}$ contains a cycle? No.
    - $V_T = \{2,1,3,4,5\}$, $A_T = \{\{2,1\},\{3,4\},\{1,3\},\{4,5\}\}$ and $Cost = 10$.
  - $\overline{A_T} \leftarrow \overline{A_T} \setminus \{4,5\}$.
- $V \setminus V_T \neq \emptyset$? No. END.

A minimum spanning tree contains the edges $\{\{2,1\},\{3,4\},\{1,3\},\{4,5\}\}$ and has cost 10.

Suggestion: Draw the tree as the algorithm proceeds.

Cut Optimality Condition

Definition

Let $G = (V, A)$ be an undirected graph and let V_1, V_2 be a partition of V. A cut $[V_1, V_2]$ is the set of edges with one endpoint in V_1 and the other in V_2, that is,

$$[V_1, V_2] = \{\{i, j\} \in A : i \in V_1 \text{ and } j \in V_2\}.$$

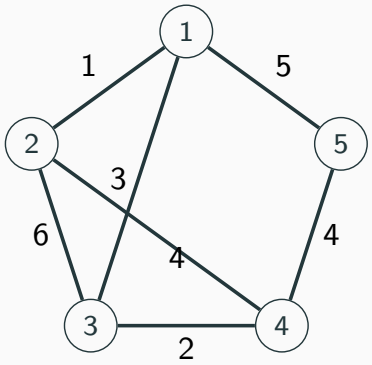
Result

A spanning tree $T = (V_T, A_T)$ is optimal if and only if

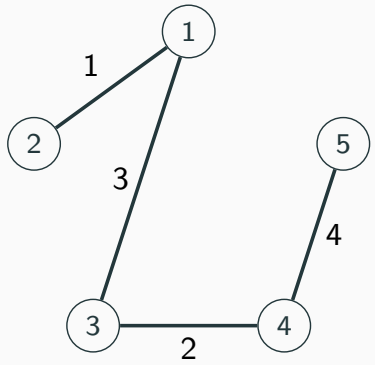
$$\forall \{i, j\} \in A_T, \forall \{u, v\} \in [S_i, S_j], c_{ij} \leq c_{uv},$$

where S_i and S_j are the sets of nodes in the two components obtained by removing edge $\{i, j\}$ from tree T, containing nodes i and j, respectively.

Example

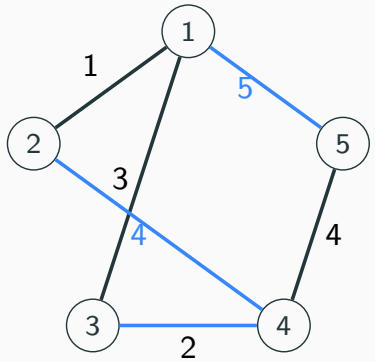


Original graph G



Minimum spanning tree T of G

Edge $\{i,j\} \in T$	S_i	S_j
{1, 2}	{1,3,4,5}	{2}
{1, 3}	{1,2}	{3,4,5}
{3, 4}	{1,2,3}	{4,5}
{4, 5}	{1,2,3,4}	{5}



$[S_3, S_4] = \{\{3,4\},\{2,4\},\{1,5\}\}.$

Prim's Algorithm

Prim's Algorithm

1: Initialize $V_T \leftarrow \{1\}$, $A_T \leftarrow \emptyset$ and $Cost \leftarrow 0$.
2: **while** $V \setminus V_T \neq \emptyset$ **do**
3: Compute the cut $[V_T, V \setminus V_T]$.
4: Select the minimum-cost edge $\{i^*, j^*\}$ in $[V_T, V \setminus V_T]$ such that $i^* \in V_T$ and $j^* \in V \setminus V_T$.
5: Set $V_T \leftarrow V_T \cup \{j^*\}$, $A_T \leftarrow A_T \cup \{i^*, j^*\}$ and $Cost \leftarrow Cost + c_{ij}$.
6: **end while**

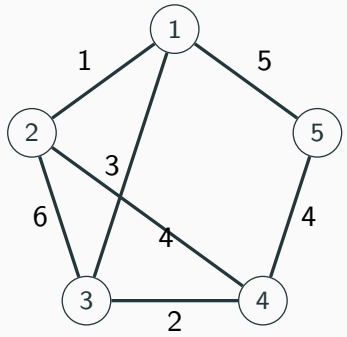
A minimum spanning tree of G is $T = (V_T, A_T)$.

Example

Execution of Prim's algorithm

Initialization: $V_T \leftarrow \{1\}$, $A_T \leftarrow \emptyset$ and $Cost \leftarrow 0$.

- $V \setminus V_T \neq \emptyset$? Yes.
  - $[V_T, V \setminus V_T] = \{\{1,2\},\{1,3\},\{1,5\}\}$.
  - $\{1,2\}$ is the minimum-cost edge.
  - $V_T \leftarrow V_T \cup \{2\}$, $A_T \leftarrow A_T \cup \{1,2\}$ and $Cost \leftarrow 1$.
- $V \setminus V_T = \{3,4,5\} \neq \emptyset$? Yes.
  - $[V_T, V \setminus V_T] = \{\{1,3\},\{1,5\},\{2,3\},\{2,4\}\}$.
  - $\{1,3\}$ is the minimum-cost edge.
  - $V_T \leftarrow V_T \cup \{3\}$, $A_T \leftarrow A_T \cup \{1,3\}$ and $Cost \leftarrow 4$.
- $V \setminus V_T = \{4,5\} \neq \emptyset$? Yes.

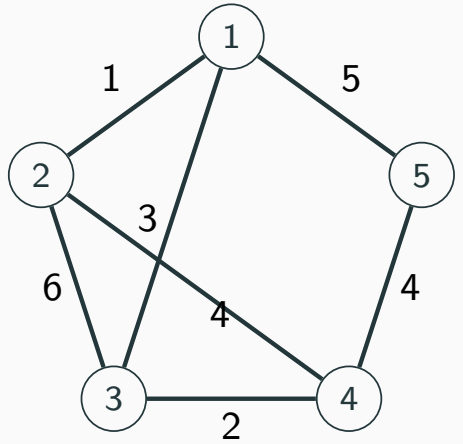


  - $[V_T, V \setminus V_T] = \{\{1,5\},\{2,4\},\{3,4\}\}$.
  - $\{3,4\}$ is the minimum-cost edge.
  - $V_T \leftarrow V_T \cup \{4\}$, $A_T \leftarrow A_T \cup \{3,4\}$ and $Cost \leftarrow 4 + 2$.
- $V \setminus V_T = \{5\} \neq \emptyset$? Yes.
  - $[V_T, V \setminus V_T] = \{\{1,5\},\{4,5\}\}$.
  - $\{4,5\}$ is the minimum-cost edge.
  - $V_T \leftarrow V_T \cup \{5\}$, $A_T \leftarrow A_T \cup \{4,5\}$ and $Cost \leftarrow 6 + 4$.
- $V \setminus V_T \neq \emptyset$? No. END!

Example

- Alternative representation of Prim's algorithm

It.	$\in V_T$	$\in V \setminus V_T$ nearest	c_{ij}	A_T
1	1	2	1	{1,2}
2	1	3	3	{1,3}
	2	4	4	
3	1	5	5	
	2	4	4	
	3	4	2	{4,3}
4	1	5	5	
	4	5	4	{4,5}



A minimum spanning tree consists of the edges {1,2}, {1,3}, {3,4} and {4,5} and has cost 10.

Minimum Spanning Tree Problem

Result

A spanning tree T satisfies the cycle optimality condition if and only if T satisfies the cut optimality condition.

Notes on the Minimum Spanning Tree Problem

- We have studied two algorithms for determining a minimum spanning tree
  - Prim's Algorithm
    - Based on the cut optimality condition.
  - Kruskal's Algorithm
    - Based on the cycle optimality condition.
- The minimum spanning tree problem is particularly important because an optimal solution can be found in polynomial time.
  - It belongs to class $\mathcal{P}$.

Outline

Motivation

Introduction

Graph Theory

Minimum Spanning Tree Problem

Shortest Path Problem

Shortest Path Problem

Definition

Consider a directed graph $G = (V, A)$ with a single source $s \in V$ and a single destination $t \in V$, where $s \neq t$. Each arc $(i, j) \in A$ has an associated value c_{ij}. Let $C = \{(s, i_1), (i_1, i_2), \ldots, (i_{n-1}, i_n), (i_n, t)\}$ a path from s to t. The cost of C is the sum of the values c_{ij} associated with its arcs, that is,

$$Cost(C) = \sum_{(i,j) \in C} c_{ij}.$$

Among all paths from s to t, the one with minimum cost is called a shortest path.

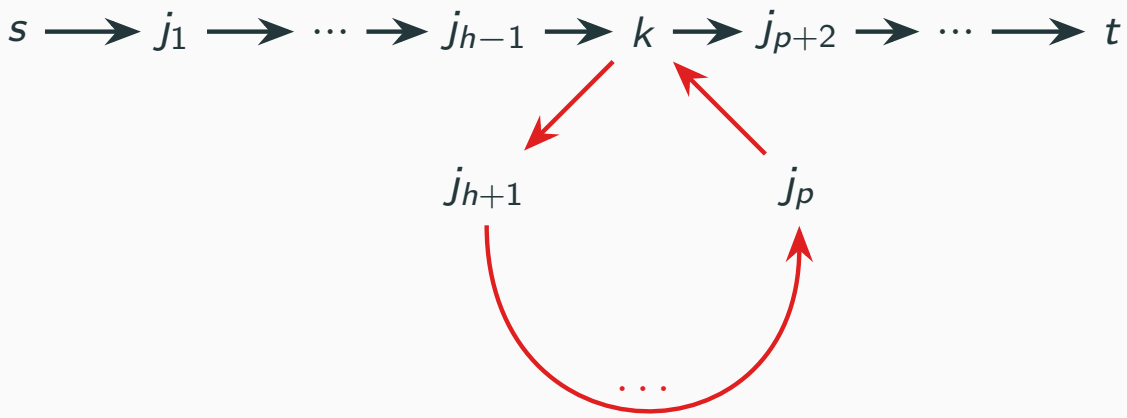
Shortest Path Problem

- Assumptions:
  - Arc costs are integers.
  - The graph contains a path from s to every other node.
    - This assumption can always be satisfied by adding a fictitious arc (s, i) with a sufficiently large cost for each node i not reachable from s.
  - The graph is directed.
    - Otherwise, each edge can be replaced by two equivalent oppositely directed arcs.
  - The graph contains no negative-cost circuit.

Shortest Path Problem

If a path C from s to t contains a circuit, that is, it has the form:

$$C = \{(s, j_1), (j_1, j_2), \ldots, (j_{h-1}, k), (k, j_{h+1}), \ldots, (j_p, k), (k, j_{p+2}), \ldots, (j_q, t)\}$$



We have

$$Cost(C) = Cost(C_1) + Cost(C_2)$$

with $C_1 = \{(s, j_1), \ldots, (j_{h-1}, k), (k, j_{p+2}), \ldots, (j_q, t)\}$ and $C_2 = \{(k, j_{h+1}), \ldots, (j_p, k)\}$.

- If $Cost(C_2) < 0$: There is no shortest path from s to t. $\Longrightarrow$ Unbounded problem.
- If $Cost(C_2) \geq 0$: We have $Cost(C) \geq Cost(C_1)$, so C_1 is a shortest path from s to t.
  - A shortest path between s and t is elementary.

Shortest Path Problem

There are several generalisations of the shortest path between node s and node t:

1. Determining shortest paths between node s and all other nodes in a graph with non-negative costs.
2. Determining shortest paths between node s and all other nodes in a graph with arbitrary costs.
3. Determining shortest paths between all pairs of nodes.

■ Problems 1 and 2 are single-source shortest-path problems, whereas problem 3 is the all-pairs shortest-path problem.

Shortest Path Problem

Shortest-path algorithms are classified as:

- Label-setting algorithms.
- Label-correcting algorithms.

The algorithms are iterative and at each step assign temporary labels to nodes, representing upper bounds on shortest-path lengths.

Shortest Path Problem

These approaches differ in how they update the labels.

- Label-setting algorithms: At each iteration, one label is made permanent (optimal).
- Label-correcting algorithms: All labels remain temporary until the final step, when they become permanent.

Shortest Path Problem

Which class of shortest-path problems can each type of algorithm solve?

- Label-setting algorithms
  - Problems on acyclic graphs with arbitrary arc costs.
  - Problems defined on graphs with non-negative arc costs.
- Label-correcting algorithms
  - All problem classes.

■ Label-setting algorithms are more efficient, whereas label-correcting algorithms apply to a broader class of problems.

Dijkstra's Algorithm

Characteristics

It is a label-setting algorithm that determines shortest paths from a node s to all other nodes in a graph with non-negative arc costs.

General idea:

Starting from source s, Dijkstra's algorithm makes node labels permanent in increasing order of distance from s.

Dijkstra's Algorithm

- Notation:
  - The label of node $i \in N$, representing a path from s to i, is: $[pred(i), d(i)]$
    - $pred(i)$ is the predecessor of i on the path from s to i.
    - $d(i)$ is the cost of the path from s to i.

The label of a node $i \in V$ is:

- permanent when it represents a minimum-cost path from s to i. In this case, it is marked with a $*$.
- temporary when it is not yet known whether it represents a minimum-cost path.

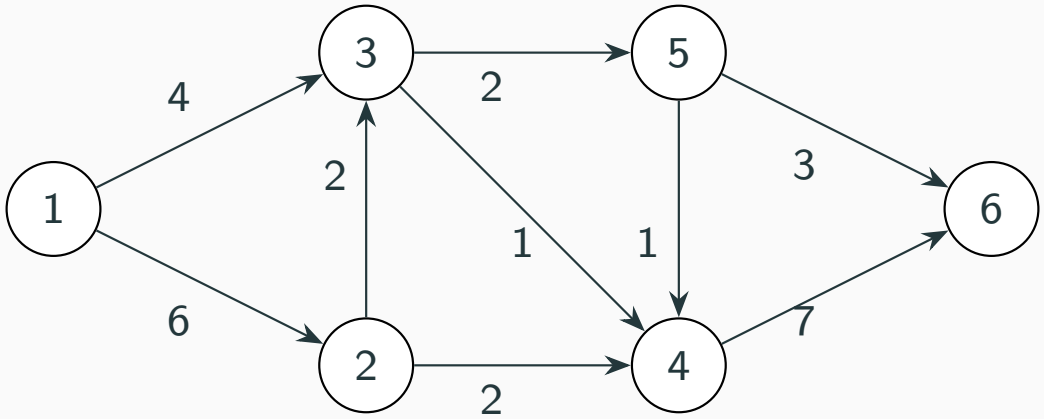
Dijkstra's Algorithm

Dijkstra's Algorithm

1: Label s with $[-, 0]^*$.
2: Label the remaining nodes in $V \setminus \{s\}$ with $[-, +\infty]$.
3: Set $r \leftarrow s$.
4: **while** There are nodes with temporary labels **do**
5: **for all** $v \in \Gamma^+(r)$ with a temporary label **do**
6: **if** $d(r) + c_{rv} < d(v)$ **then**
7: Update the label of v to $[r, d(r) + c_{rv}]$.
8: **end if**
9: **end for**
10: Choose the node r with the minimum-cost temporary label and mark that label permanent.
11: **end while**

Example

Determine the shortest paths from node 1 to all nodes of the graph



Example

	1	2	3	4	5	6	r
Inic.	$[-, 0]^*$	$[-, +\infty]$	$[-, +\infty]$	$[-, +\infty]$	$[-, +\infty]$	$[-, +\infty]$	1
1		$[1, 6]$	$[1, 4]^*$	$[-, +\infty]$	$[-, +\infty]$	$[-, +\infty]$	3
2		$[1, 6]$		$[3, 5]^*$	$[3, 6]$	$[-, +\infty]$	4
3		$[1, 6]^*$			$[3, 6]$	$[4, 12]$	2
4					$[3, 6]^*$	$[4, 12]$	5
5						$[5, 9]^*$	FIM

All labels are permanent. $\Longrightarrow$ End of the algorithm.

Example

The shortest paths obtained are:

- From 1 to 2: $\{\text{pred}(2), 2\} = \{1, 2\}$
  - $d^*(2) = 6$
- From 1 to 3: $\{\text{pred}(3), 2\} = \{1, 3\}$
  - $d^*(3) = 4$
- From 1 to 4: $\{(1,3), (3,4)\}$
  - $d^*(4) = 5$
- From 1 to 5: $\{(1,3), (3,5)\}$
  - $d^*(5) = 6$
- From 1 to 6: $\{(1,3), (3,5), (5,6)\}$
  - $d^*(6) = 9$

Correctness of Dijkstra's Algorithm

Why does making a label permanent identify a shortest path?

Let S be the set of nodes with permanent labels.
The first node added to S is source s, and $d^*(s) = 0$ (non-negative costs).
Suppose node $j \notin S$ is selected to make its label $d(j)$ permanent, although it does not represent a shortest path from s to j. Since j was selected, its temporary label $d(j)$ has the smallest value among nodes outside S. Let P be a path from s to j. Since $s \in S$ and $j \notin S$, path P starts in S and eventually leaves it. Let (x, y) be the first arc of P leaving S, that is, $x \in S$ and $y \notin S$. Since $x \in S$, we know that $d^*(x)$ is the shortest-path distance from s to x. However, since $y \notin S$, we have $d^*(y) \leq d(y) = d^*(x) + c_{xy}$.
Since all arcs of P have non-negative cost, we have $d(y) \leq d(j)$. Therefore node y should have been selected to make its label permanent, a contradiction.

Dijkstra's Algorithm

- Dijkstra's algorithm can be used to find a shortest path from node s to node t by changing the stopping condition to:

  while $r \neq t$ **do**

- Dijkstra's algorithm can also be adapted to find a longest path between two nodes:
  - Replace min with max and initialize $d(i) = -\infty$.
  - A label becomes permanent only after all predecessor labels are permanent.

Bellman–Ford Algorithm

Characteristics

It determines shortest paths from a node s to all other nodes of the graph.
It can detect negative-cost circuits.

General idea:

At the end of iteration k, labels represent shortest-path lengths using at most $k+1$ nodes. In a graph with n nodes, an elementary path has at most $n-1$ arcs.

Algorithm Bellman-Ford

■ Notation: $d^k(i)$ is the shortest-path cost from source s to i using at most $k+1$ nodes.

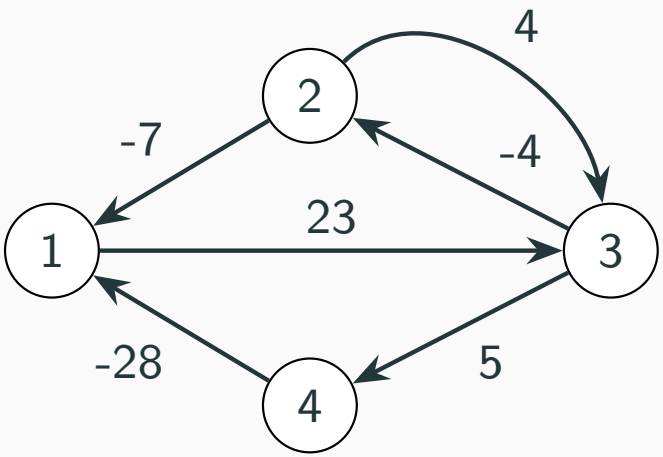
Algorithm Bellman-Ford

1: Set $d^1(s) = 0, k = 1, K = \Gamma^+(s)$, $d^1(i) = c_{si}$, $\forall i \in \Gamma^+(s)$ and $d^1(i) = +\infty$, $\forall i \in V \setminus \Gamma^+(s)$.
2: **while** $k < n-1$ and $\exists i \in V : d^k(i) \neq d^{k-1}(i)$ **do**
3: **for all** $i \in V$ **do**
4: **if** $i \in \Gamma^+(K)$ **then**
5: Update the label by setting

$$d^{k+1}(i) = \min\{d^k(i), \min_{n \in \Gamma^-(j) \cap K} \{d^k(n) + c_{ni}\}.$$

6: **else**
7: Set $d^{k+1}(i) = d^k(i)$.
8: **end if**
9: **end for**
10: Set $K = \{i \in V : d^k(i) \neq d^{k-1}(i)\}$ and $k = k+1$.
11: **if** $k = n-1$ and $\exists i \in V : d^k(i) \neq d^{k-1}(i)$ **then**
12: There is a negative-cost circuit.
13: **end if**
14: **end while**

Example



Auxiliary calculations:

- $d^2(2) = \min\{d^1(2), d^1(3) - 4\} = 19$
- $d^2(4) = \min\{d^1(4), d^1(3) + 5\} = 28$
- $d^3(1) = \min\{d^2(1), d^2(2) - 7, l^2(4) - 28\} = 0$
- $d^3(3) = \min\{l^2(3), l^2(2) + 4\} = 23$

■ Initialization:

i	1	2	3	4	K	$\Gamma^+(K)$	k
$d^1(i)$	0	∞	23	∞	$\{3\}$	$\{2,4\}$	1

$k < n-1$ and have any labels changed? Yes.

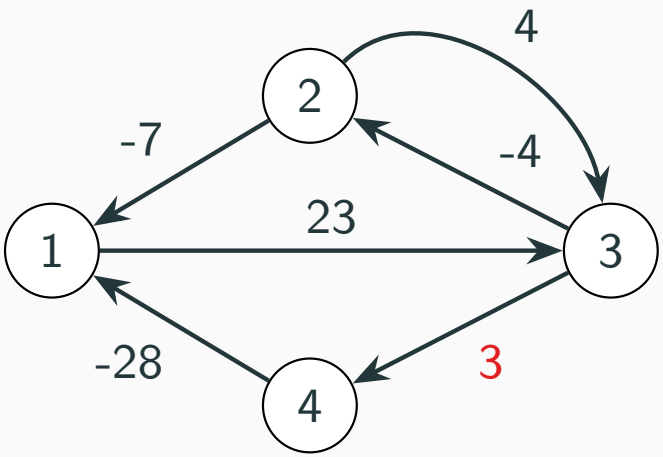
i	1	2	3	4	K	$\Gamma^+(K)$	k
$d^2(i)$	0	19	23	28	$\{2,4\}$	$\{1,3\}$	2

▶ $k = n-1$ and have any labels changed? No.

$k < n-1$ and have any labels changed? Yes.

i	1	2	3	4	K	$\Gamma^+(K)$	k
$d^3(i)$	0	19	23	28	$\emptyset$	$\emptyset$	3

Example



- Initialization:

i	1	2	3	4	K	$\Gamma^+(K)$	k
$d^1(i)$	0	∞	23	∞	$\{3\}$	$\{2,4\}$	1
$d^2(i)$	0	19	23	26	$\{2,4\}$	$\{1,3\}$	2
$d^3(i)$	-2	19	23	26	$\{1\}$	$\{3\}$	3

- Since $k = n - 1$ and some labels have changed, there is a negative-cost circuit containing a node whose label changed.
  - Negative-cost circuit: (3,4,1,3).

Algorithm Bellman-Ford

We know the shortest-path distances from node 1 to all other nodes of the graph. How can we determine the shortest path itself?

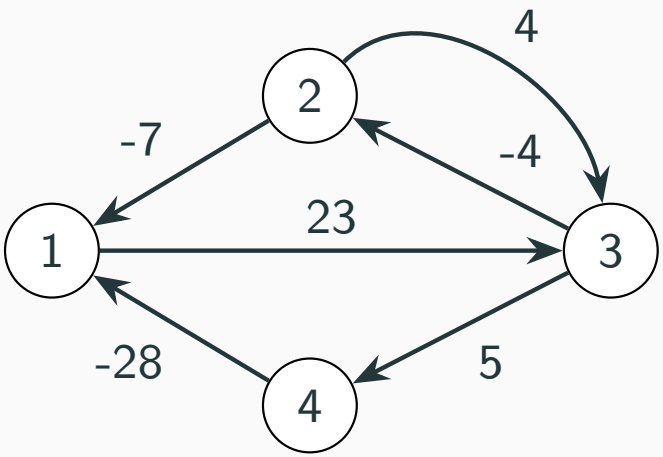
The path is determined by recursively applying the following relation

$$d^k(j) + c_{ji} = d^k(i)$$

where j is the predecessor of i on the shortest path from s to i.

- We can also store the information $pred(i)$ as in the Dijkstra's Algorithm.

Example



i	1	2	3	4	K	$\Gamma^+(K)$	k
$d^1(i)$	0	∞	23	∞	$\{3\}$	$\{2,4\}$	1
$d^2(i)$	0	19	23	28	$\{2,4\}$	$\{1,3\}$	2
$d^3(i)$	0	19	23	28	$\emptyset$	$\emptyset$	3

- Path from 1 to 2: $d^3(2) = d^3(3) - 4 \implies ((1,3),(3,2))$
- Path from 1 to 3: $d^3(3) = d^3(1) + 23 \implies ((1,3)$
- Path from 1 to 4: $d^3(4) = d^3(2) + 5 \implies ((1,3),(3,4))$

■ Alternatively:

i	1	2	3	4	K	$\Gamma^+(K)$	k

Algorithm Bellman-Ford

Result (shortest-path optimality conditions)

For every node $j \in V$, let $d(j)$ be the cost of a path from source s to j. $d(j)$ represents shortest-path costs if and only if $d(j) \leq d(i) + c_{ij}$, $\forall (i,j) \in A$.

Proof.

$\Longrightarrow$ Let $d(j)$ be the shortest-path cost from s to j and suppose there is an arc $(i,j) : d(j) > d(i) + c_{ij}$. We can obtain a shorter path from s to j through node i, so $d(j)$ cannot be the shortest-path length.

$\Longleftarrow$ Let $P = (s = i_1, i_2), (i_2, 1_3), \ldots, (i_{k-1}, i_k = j)$ a path from s to j. The optimality conditions, which hold, imply that:

$$\begin{aligned} d(j) = & d(i_k) \leq d(i_{k-1}) + c_{i_{k-1} i_k} \\ & \ldots \\ & d(i_3) \leq d(i_2) + c_{i_2 i_3} \\ & d(i_2) \leq d(i_1) + c_{i_1 i_2} = c_{i_1 i_2} \end{aligned}$$

Summing these inequalities yields $d(j) = d(i_k) \leq c_{i_{k-1} i_k} + \ldots + c_{i_3 i_2} + c_{i_1 i_2} = \sum_{(i,j) \in P} c_{ij}$. Thus, $d(j)$ is a lower bound on the length of every path from s to j. Since $d(j)$ is also the length of a path from s to j, it is an upper bound on the shortest-path length. Therefore, $d(j)$ is the shortest-path length. $\square$

Floyd's Algorithm

Characteristics

It determines shortest paths between all pairs of nodes in a graph.
It can detect negative-cost circuits.

General idea:

At the end of iteration k, the labels give the lengths of the shortest paths between each pair of nodes whose intermediate nodes can only be $1, 2, \ldots, k$.

Floyd's Algorithm

- Notation:

$d^0(i,j)$ is the shortest-path length between node i and node j without intermediate nodes (0 nodes intermédios).

$$d^0(i,j) = \begin{cases} c_{ij} & \text{if } (i,j) \in A \\ +\infty & \text{otherwise} \end{cases}$$

$d^k(i,j)$ is the shortest-path length between node i and node j which may use nodes $1, 2, \ldots, k-1$ as intermediate nodes.

Propriedade

$$d^{k+1}(i,j) = \min\{d^k(i,j), d^k(i,k) + d^k(k,j)\}$$

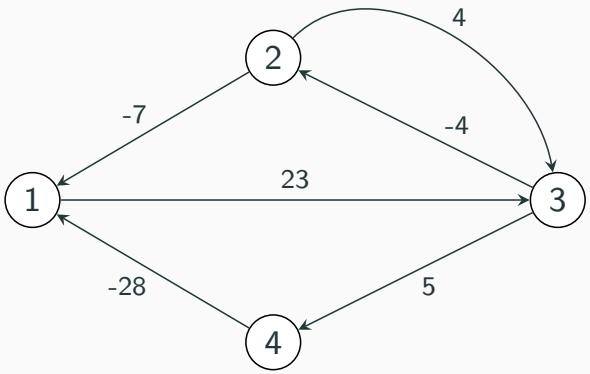
Algorithm Floyd

Floyd's Algorithm

1: Set $d(i,i) \leftarrow 0$, $d(i,j) \leftarrow c_{ij}$ and $p(i,j) \leftarrow i$, $\forall i,j \in V$.
2: **for all** $k \in V$ **do**
3: **for all** $i \in V$ **do**
4: **for all** $j \in V$ **do**
5: **if** $d(i,k) + d(k,j) < d(i,j)$ **then**
6: Set $d(i,j) \leftarrow d(i,k) + d(k,j)$ and $p(i,j) \leftarrow p(k,j)$.
7: **end if**
8: **end for**
9: **if** $d(i,i) < 0$ **then**
10: STOP. There is a negative-cost circuit.
11: **end if**
12: **end for**
13: **end for**

Example

Determine the shortest paths between all pairs of nodes of the graph



- Initialization:

$$d^0 = \begin{bmatrix} 0 & \infty & 23 & \infty \\ -7 & 0 & 4 & \infty \\ \infty & -4 & 0 & 5 \\ -28 & \infty & \infty & 0 \end{bmatrix} \qquad p^0 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{bmatrix}$$

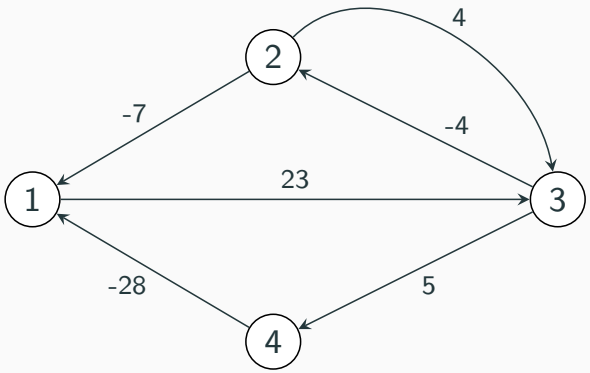
- $k = 1$:

$$d^1 = \begin{bmatrix} 0 & \infty & 23 & \infty \\ -7 & 0 & 4 & \infty \\ \infty & -4 & 0 & 5 \\ -28 & \infty & -5 & 0 \end{bmatrix} \qquad p^1 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 1 & 4 \end{bmatrix}$$

- $k = 2$:

$$d^2 = \begin{bmatrix} 0 & \infty & 23 & \infty \\ -7 & 0 & 4 & \infty \\ -11 & -4 & 0 & 5 \\ -28 & \infty & -5 & 0 \end{bmatrix} \qquad p^2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 2 & 3 & 2 & 3 \\ 4 & 4 & 1 & 4 \end{bmatrix}$$

Example (continued)



- $k = 3$:

$$d^3 = \begin{bmatrix} 0 & 19 & 23 & 28 \\ -7 & 0 & 4 & 9 \\ -11 & -4 & 0 & 5 \\ -28 & -9 & -5 & 0 \end{bmatrix} \qquad p^3 = \begin{bmatrix} 1 & 3 & 1 & 3 \\ 2 & 2 & 2 & 3 \\ 2 & 3 & 2 & 3 \\ 4 & 3 & 1 & 4 \end{bmatrix}$$

- $k = 4$:

$$d^4 = \begin{bmatrix} 0 & 19 & 23 & 28 \\ -19 & 0 & 4 & 9 \\ -23 & -4 & 0 & 5 \\ -28 & -9 & -5 & 0 \end{bmatrix} \qquad p^4 = \begin{bmatrix} 1 & 3 & 1 & 3 \\ 4 & 2 & 2 & 3 \\ 4 & 3 & 2 & 3 \\ 4 & 3 & 1 & 3 \end{bmatrix}$$

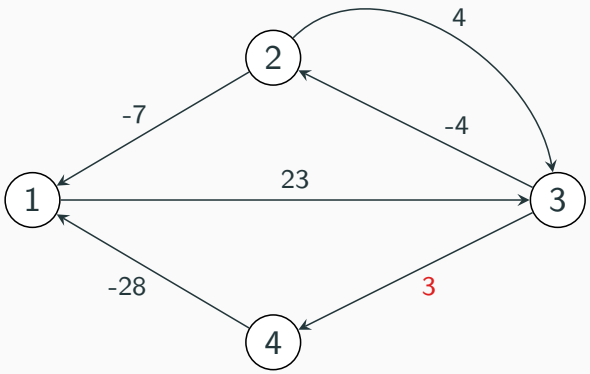
Shortest path from 1 to 4? Since $p^4_{14} = 3$, node 3 is the predecessor of node 4 on the shortest path from 1 to 4.

Shortest path from 1 to 3? Since $p^4_{13} = 1$, the shortest path from 1 to 4 is (1,3,4).

What is the cost of the shortest path from 1 to 4? Its cost is 28, as $d^4_{14} = 28$

Example

Determine the shortest paths between all pairs of nodes of the graph



- Initialization:

$$d^0 = \begin{bmatrix} 0 & \infty & 23 & \infty \\ -7 & 0 & 4 & \infty \\ \infty & -4 & 0 & 3 \\ -28 & \infty & \infty & 0 \end{bmatrix} \qquad p^0 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{bmatrix}$$

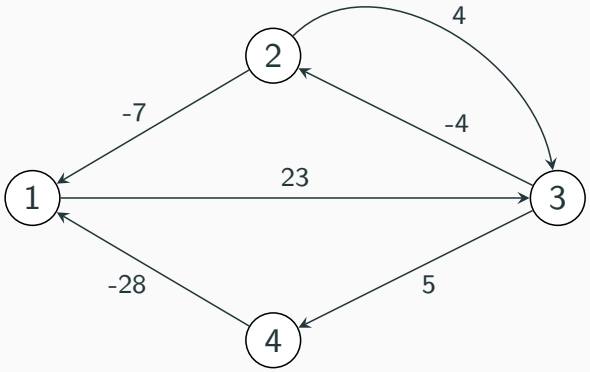
- $k = 1$:

$$d^1 = \begin{bmatrix} 0 & \infty & 23 & \infty \\ -7 & 0 & 4 & \infty \\ \infty & -4 & 0 & 3 \\ -28 & \infty & -5 & 0 \end{bmatrix} \qquad p^1 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 1 & 4 \end{bmatrix}$$

- $k = 2$:

$$d^2 = \begin{bmatrix} 0 & \infty & 23 & \infty \\ -7 & 0 & 4 & \infty \\ -11 & -4 & 0 & 3 \\ -28 & \infty & -5 & 0 \end{bmatrix} \qquad p^2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 2 & 3 & 2 & 3 \\ 4 & 4 & 1 & 4 \end{bmatrix}$$

Example (continued)



- $k = 3$:

$$d^3 = \begin{bmatrix} 0 & 19 & 23 & 26 \\ -7 & 0 & 4 & 7 \\ -11 & -4 & 0 & 3 \\ -28 & -9 & -5 & -2 \end{bmatrix} \qquad p^3 = \begin{bmatrix} 1 & 3 & 1 & 3 \\ 2 & 2 & 2 & 3 \\ 2 & 3 & 2 & 3 \\ 4 & 3 & 1 & 3 \end{bmatrix}$$

Since $d^3_{44} < 0$, there is a negative-cost circuit containing node 4.

Notes on the Shortest Path Problem

- We have studied three shortest-path algorithms
  - Dijkstra's Algorithm
    - Computes shortest paths from the source to all nodes in graphs with non-negative costs.
  - Bellman–Ford Algorithm
    - Computes shortest paths from the source to all nodes and detects negative-cost circuits.
  - Floyd–Warshall Algorithm
    - Computes shortest paths between all pairs of nodes and detects negative-cost circuits.

- The shortest-path problem is particularly important because algorithms can find an optimal solution in polynomial time.
  - It belongs to class $\mathcal{P}$.

References


Ahuja, R.K., T.L. Magnanti, J.B. Orlin (1993). *Network Flows: Theory, Algorithms and Applications*. Prentice Hall.


Hillier, F. S. & Lieberman, G. J. (2021). *Introduction to Operations Research*. Mc Graw-Hill, New York, USA, 11th edition.


Taha, H. A. (2017). *Operations Research – An Introduction*. Pearson Prentice-Hall, New Jersey, 10th edition.