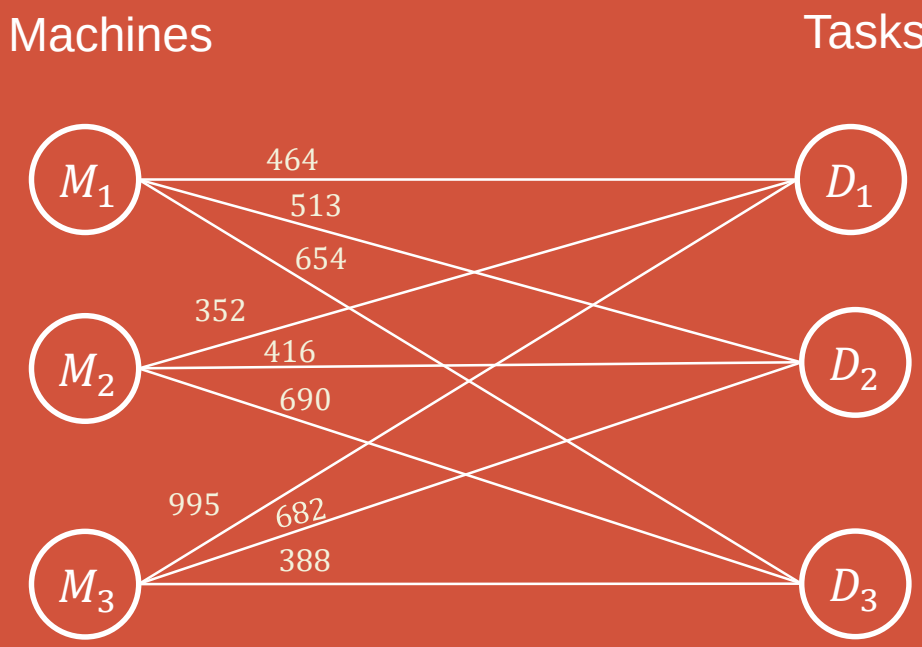
Plan for today

- Assignment Problem
  - Solution method: Hungarian algorithm
- Network Optimization Models
  - Terminologies
  - Minimum Spanning Tree Problem



ASSIGNMENT PROBLEM

… where we see the assignment problem and an algorithm to solve it

Example

- 4 machines and 4 tasks
- Cost for machine i to process task j is c_{ij} as given below
- Each task to be assigned to exactly one machine
- Each machine can process exactly one task

Locations
Jobs
Tasks

Franchisees
Employees
Machines

	T_1	T_2	T_3	T_4
M_1	5	9	3	6
M_2	8	7	8	2
M_3	6	10	12	7
M_4	3	10	8	6

Question: Which task should be assigned to which machine so that the total processing cost is minimized?

Formulation

$$\begin{aligned}\min Z = {} & 5\,x_{11} + 9\,x_{12} + 3\,x_{13} + 6\,x_{14} \\ & +8\,x_{21} + 7\,x_{22} + 8\,x_{23} + 2\,x_{24} \\ & +6\,x_{31} + 10\,x_{32} + 12\,x_{33} + 7\,x_{34} \\ & +3\,x_{41} + 10\,x_{42} + 8\,x_{43} + 6\,x_{44}\end{aligned}$$

$$\begin{aligned}x_{11} + x_{12} + x_{13} + x_{14} &= 1 \\ x_{21} + x_{22} + x_{23} + x_{24} &= 1 \\ x_{31} + x_{32} + x_{33} + x_{34} &= 1 \\ x_{41} + x_{42} + x_{43} + x_{44} &= 1 \\ x_{11} + x_{21} + x_{31} + x_{41} &= 1 \\ x_{12} + x_{22} + x_{32} + x_{42} &= 1 \\ x_{13} + x_{23} + x_{33} + x_{43} &= 1 \\ x_{14} + x_{24} + x_{34} + x_{44} &= 1 \\ x_{ij} \in \{0,1\}, i = 1,2,3,4, j &= 1,2,3,4\end{aligned}$$

Note: Every feasible solution has exactly 4 variables that take value one, i.e., non-zero

	T_1	T_2	T_3	T_4
M_1	5	(9)	3	6
M_2	8	7	(8)	2
M_3	(6)	10	12	7
M_4	3	10	8	(6)

A non-zero variable is also known as an **assignment**

Assignment Problem: General Case

- Number of machines = Number of tasks, say they are $= n$
- Given: Cost $c_{ij} (\geq 0)$ associated with machine i performing task j
- Each machine is to be assigned to exactly one task
- Each task is to be performed by exactly one machine

	T_1	...	T_j	...	T_n
M_1	c_{11}	...	c_{1j}	...	c_{1n}
⋮	⋮	⋱	⋮	⋱	⋮
M_i	c_{i1}	...	c_{ij}	...	c_{in}
⋮	⋮	⋱	⋮	⋱	⋮
M_n	c_{n1}	...	c_{nj}	...	c_{nn}

Question: Which task should be assigned to which machine so that the total processing cost is minimized?

Formulation of the general case assignment problem

$$\min Z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \text{ for every } i \in \{1, \dots, n\}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \text{ for every } j \in \{1, \dots, n\}$$

$$x_{ij} \in \{0,1\} \text{ for every } i \in \{1, \dots, n\}, j \in \{1, \dots, n\}$$

Note: Every feasible solution has exactly n non-zero variables

A non-zero variable is also known as an **assignment**

HUNGARIAN ALGORITHM

… special purpose algorithm to solve the assignment problem

Steps 1 & 2

- Step 1: For each row, subtract row min from that row
- Step 2: For each col, subtract column minimum from that col
- Reduced matrix
  - each row will have at least one zero and
  - each column will have at least one zero

Original Matrix

	T_1	T_2	T_3	T_4
M_1	13	16	12	11
M_2	15	M	13	20
M_3	5	7	10	6
M_4	0	0	0	0

Reduced Matrix

	T_1	T_2	T_3	T_4
M_1	2	5	1	0
M_2	2	M	0	7
M_3	0	2	5	1
M_4	0	0	0	0

Step 3

- Find a maximum assignment using zero entries in the reduced cost matrix

Choose a maximum possible number of zeroes such that each row and each col has at most one chosen zero

If we can find n assignments in the reduced cost matrix, then we have an optimum

Original Matrix

	T_1	T_2	T_3	T_4
M_1	13	16	12	11
M_2	15	M	13	20
M_3	5	7	10	6
M_4	0	0	0	0

Optimum solution is
$x_{14}^* = 1, x_{23}^* = 1, x_{31}^* = 1, x_{42}^* = 1$

Optimal cost is $11 + 13 + 5 + 0 = 29$

Reduced Matrix

	T_1	T_2	T_3	T_4
M_1	2	5	1	(0)
M_2	2	M	(0)	7
M_3	(0)	2	5	1
M_4	0	(0)	0	0

Optimum solution is
$x_{14}^* = 1, x_{23}^* = 1, x_{31}^* = 1, x_{42}^* = 1$

Optimal cost is 0

Step 3: How to look for a maximum zero-cost assignment?

Another rule:

Repeat until no more assignments can be made:

a. If any remaining col or row has exactly one zero, then make an assignment

b. If all rows and cols have at least 2 zeroes, then pick an arbitrary assignment

- Once an assignment is made, the other zeros in the corresponding row and column cannot be used for the assignment
  - so cross out the row and column

	T_1	T_2	T_3	T_4
M_1	2	5	1	0
M_2	2	M	0	7
M_3	0	2	5	1
M_4	0	0	0	0

Watch out: Rule may still not give a maximum assignment

→ Always inspect for a zero-cost assignment

Steps 1, 2 & 3 may not lead to optimal solution

- An example

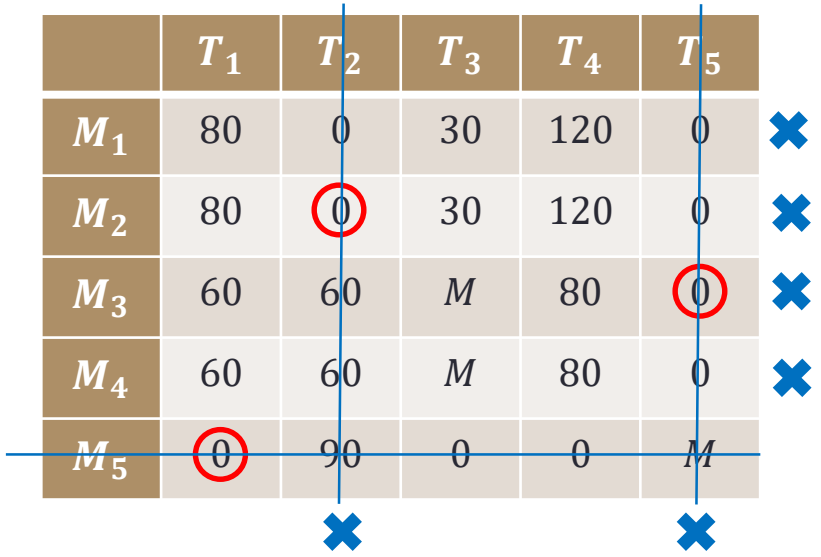
	T_1	T_2	T_3	T_4	T_5
M_1	80	0	30	120	0
M_2	80	0	30	120	0
M_3	60	60	M	80	0
M_4	60	60	M	80	0
M_5	0	90	0	0	M

- Cannot reduce further
  … since every row and every column has at least one zero
- Rule does not give a zero-cost assignment

Step 4

Step 4: Find the minimum number of rows and columns to cover all zeroes

1. Mark all unassigned rows
2. For all marked rows:
   - If a marked row has a zero in it, mark the corresponding column
3. For all marked cols:
   - If the marked col has an assignment, then mark the corresponding row
4. Repeat 2. and 3. until no more marking is possible
5. Draw lines through unmarked rows and marked cols



	T_1	T_2	T_3	T_4	T_5	
M_1	80	0	30	120	0	✖
M_2	80	0	30	120	0	✖
M_3	60	60	M	80	0	✖
M_4	60	60	M	80	0	✖
M_5	0	90	0	0	M	
		✖			✖	

No of lines gives the min number of rows and cols to cover all zeroes

If number of lines is n, then the rule in Step 3 did not give a maximum assignment

→ Return to Step 3 & manually inspect the reduced matrix for a zero-cost assignment

Step 5

- Find θ = smallest uncovered number
- Subtract θ from every uncovered number
- Add θ to numbers that are at intersections of covering lines
- Repeat Steps 3, 4 and 5 until Step 3 finds n assignments using zero-cost entries

	T_1	T_2	T_3	T_4	T_5
M_1	80	0	30	120	0
M_2	80	0	30	120	0
M_3	60	60	M	80	0
M_4	60	60	M	80	0
M_5	0	90	0	0	M

$\theta = 30$

	T_1	T_2	T_3	T_4	T_5
M_1	50	0	0	90	0
M_2	50	0	0	90	0
M_3	30	60	M	50	0
M_4	30	60	M	50	0
M_5	0	120	0	0	M

Step 3: Look for a maximum assignment

Rule:

Repeat until no more assignments can be made:

a. If any remaining col or row has exactly one zero, then make an assignment

b. If all rows and cols have at least 2 zeroes, then pick an arbitrary assignment

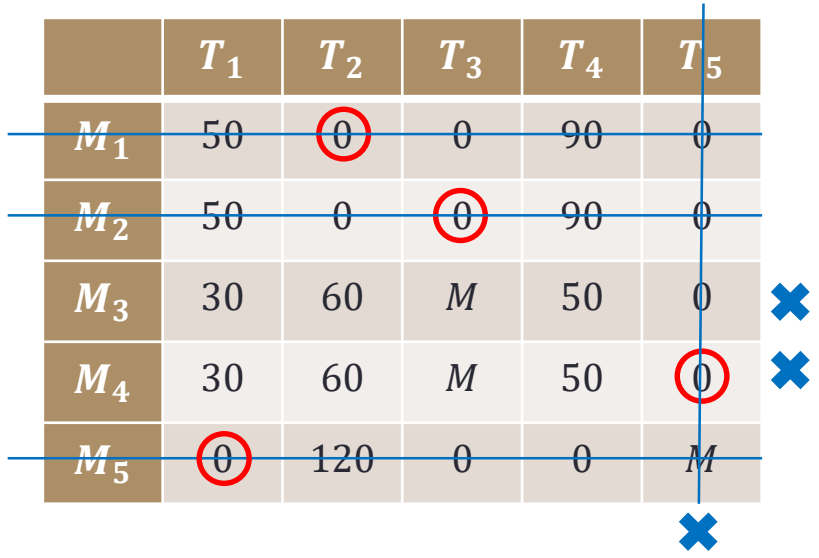
- Once an assignment is made, the other zeros in the corresponding row and column cannot be used for the assignment
  - so cross out the row and column

	T_1	T_2	T_3	T_4	T_5
M_1	50	0	0	90	0
M_2	50	0	0	90	0
M_3	30	60	M	50	0
M_4	30	60	M	50	0
M_5	0	120	0	0	M

Step 4

Step 4: Find the minimum number of rows and columns to cover all zeroes

1. Mark all unassigned rows
2. For all marked rows:
   - If a marked row has a zero in it, mark the corresponding column
3. For all marked cols:
   - If the marked col has an assignment, then mark the corresponding row
4. Repeat steps 2 and 3 until no more marking is possible
5. Draw lines through unmarked rows and marked cols



	T_1	T_2	T_3	T_4	T_5
M_1	50	0	0	90	0
M_2	50	0	0	90	0
M_3	30	60	M	50	0
M_4	30	60	M	50	0
M_5	0	120	0	0	M

No of lines gives the min number of rows and cols to cover all zeroes

Step 5

- Let θ = smallest uncovered number
- Subtract θ from every uncovered number
- Add θ to numbers that are at intersections of covering lines
- Repeat Steps 3, 4 and 5 until Step 3 finds n assignments using zero-cost entries

	T_1	T_2	T_3	T_4	T_5
M_1	50	0	0	90	0
M_2	50	0	0	90	0
M_3	30	60	M	50	0
M_4	30	60	M	50	0
M_5	0	120	0	0	M

$\theta = 30$

	T_1	T_2	T_3	T_4	T_5
M_1	50	0	0	90	30
M_2	50	0	0	90	30
M_3	0	30	M	20	0
M_4	0	30	M	20	0
M_5	0	120	0	0	M

Step 3: Look for a maximum assignment

Rule:

Repeat until no more assignments can be made:

a. If any remaining col or row has exactly one zero, then make an assignment

b. If all rows and cols have at least 2 zeroes, then pick an arbitrary assignment

- Once an assignment is made, the other zeros in the corresponding row and column cannot be used for the assignment
  - so cross out the row and column

	T_1	T_2	T_3	T_4	T_5
M_1	50	0	0	90	30
M_2	50	0	0	90	30
M_3	0	30	M	20	0
M_4	0	30	M	20	0
M_5	0	120	0	0	M

Terminate!

Cost of the optimum solution is

$$Z = c_{12} + c_{23} + c_{35} + c_{41} + c_{54} = 0 + 30 + 0 + 60 + 0 = 90$$

Kuhn Konig Egervary Jacobi




Hungarian Algorithm

- Step 1: For each row, subtract row min from that row
- Step 2: For each col, subtract col min from that col
- Step 3: Find a maximum assignment using zero cost entries in the reduced matrix  We have a rule; sometimes needs manual inspection
  - If we have n assignments, then terminate and output the assignment and its cost
- Step 4: Find the minimum number of rows and columns to cover all zeroes
- Step 5:
  - Find θ = smallest uncovered number
  - Subtract θ from every uncovered number
  - Add θ to numbers that are at intersections of covering lines
- Repeat Steps 3, 4 and 5 until until Step 3 finds n assignments

Telephone Network

Power Transmission Network

Internet Network

Airline Network

Railway Network

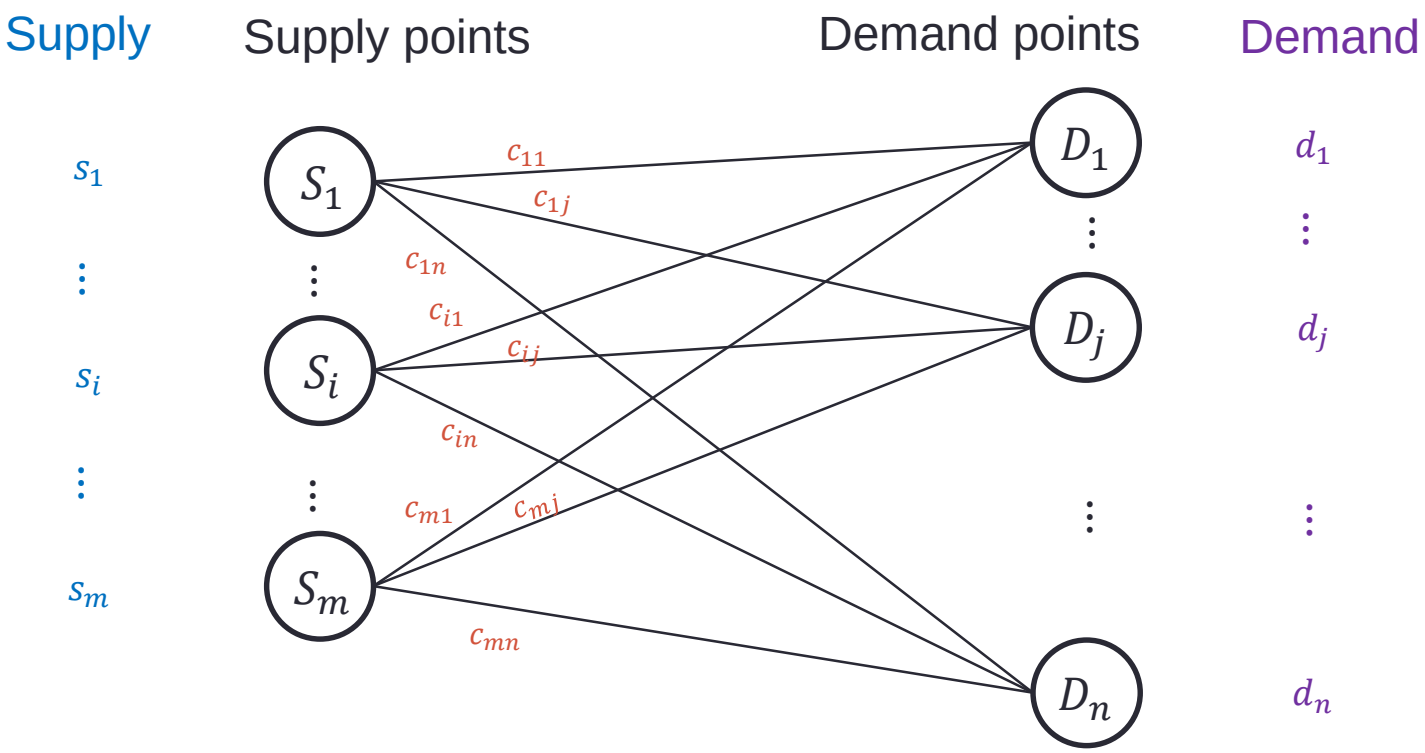
Transportation Network

Social Network

NETWORK OPTIMIZATION MODELS

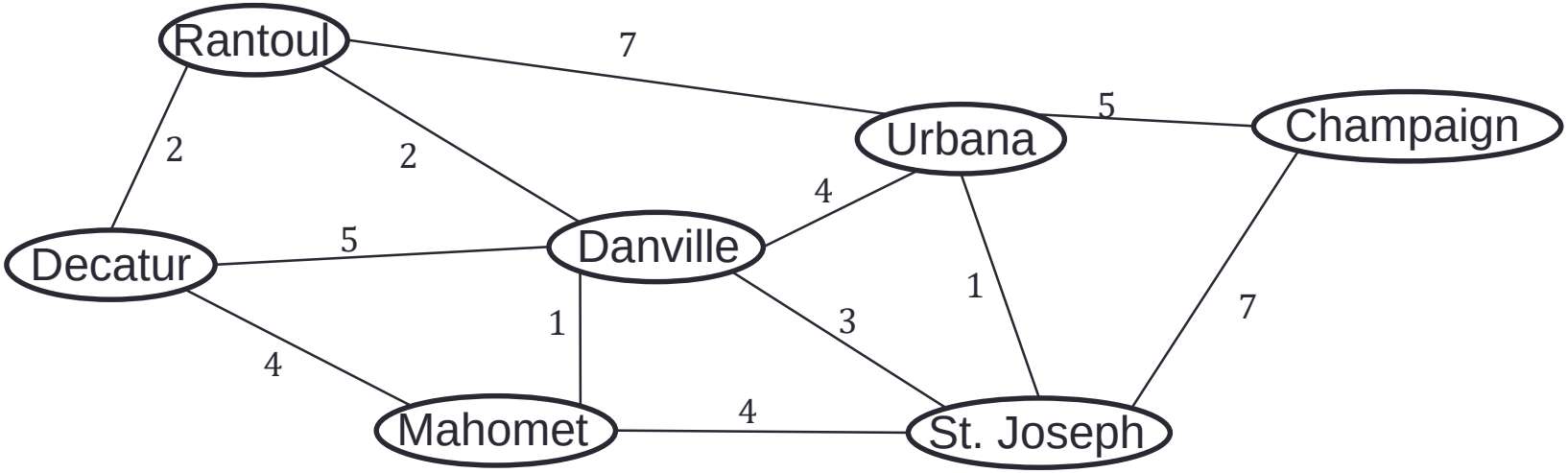
… where we introduce network models

Recall: Transportation Problem



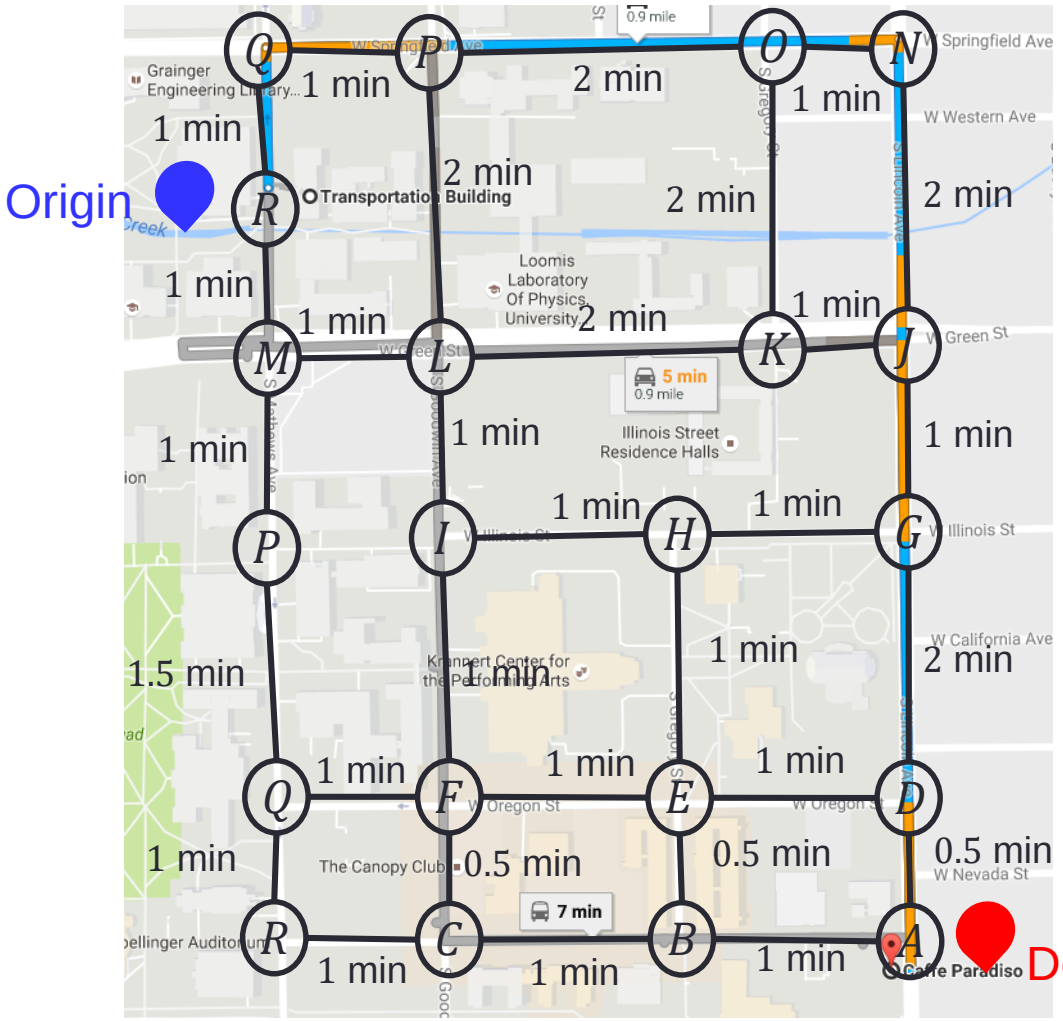
Application 1

Design of power transmission networks



- The edge values represent the cost of laying a link
- How to achieve connectivity while minimizing the link-laying cost?

Application 2



Network Optimization Problems

… that we will consider

1. Minimum Spanning Tree problem
   - in undirected networks

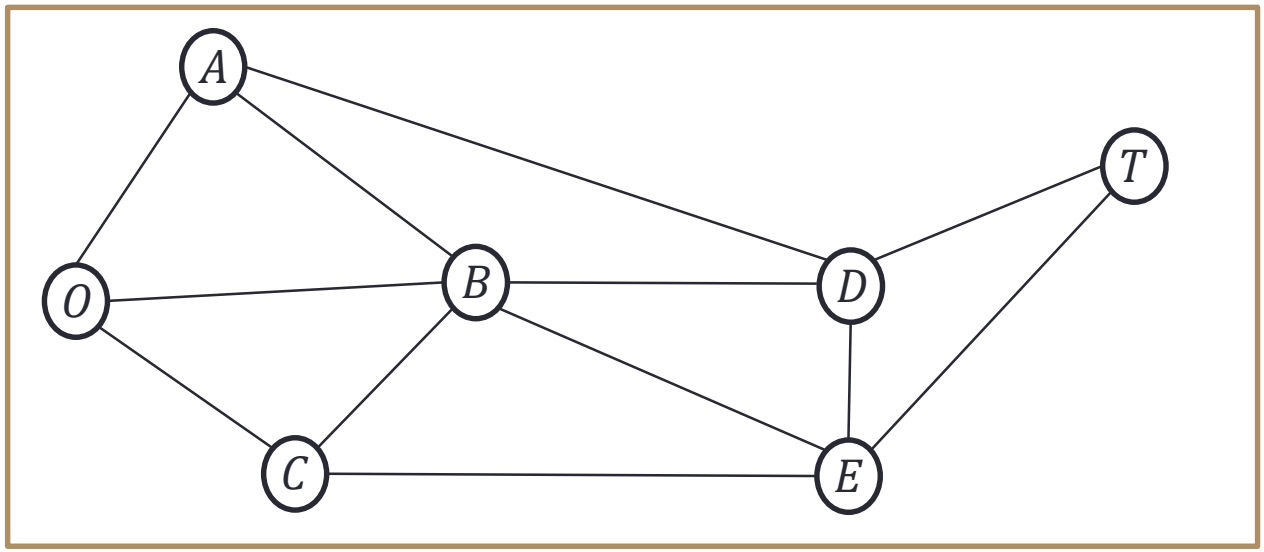
2. Shortest Path problem
   - in undirected networks
   - in directed networks

3. Maximum Flow problem
   - in directed networks

NETWORK TERMINOLOGIES

Network (Graph)

- A **network** (graph) $\mathcal{G}$ is a collection of **nodes** (vertices) and **arcs** (edges)
  - Denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{A})$
- **Arcs** represent the links/connectivities between nodes

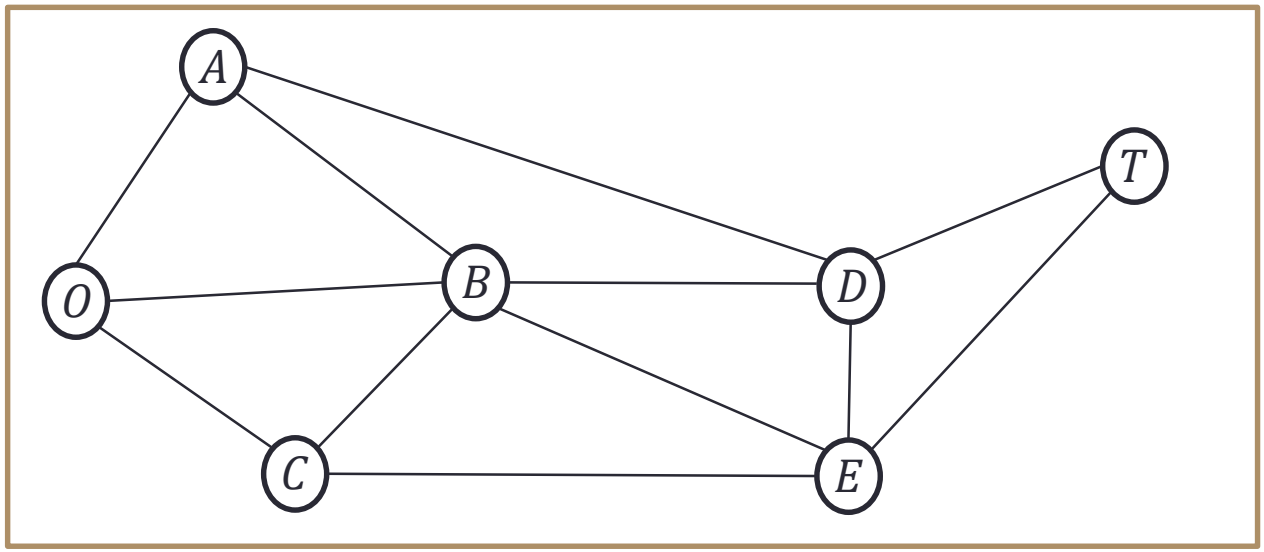


Nodes $\mathcal{V} = \{A, B, C, D, E, T, O\}$

Arcs $\mathcal{A} = \{AB, AO, AD, BC, BO, BD, BE, CE, CO, DE, DT, ET\}$

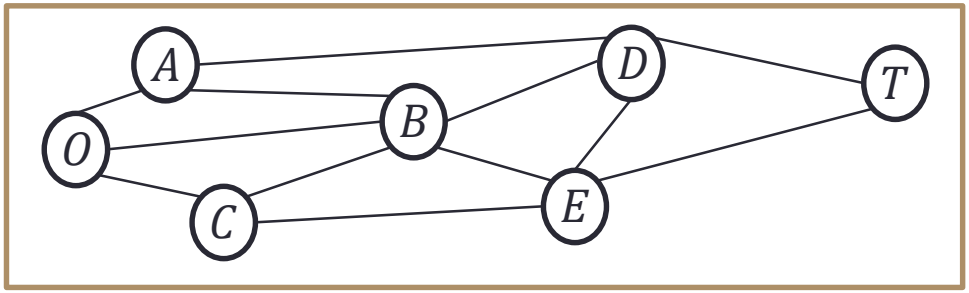
Subgraph

- A **subgraph** $\mathcal{H}$ of a graph $\mathcal{G}$ is
  - a subcollection S of nodes of $\mathcal{G}$ and
  - a subcollection of arcs of $\mathcal{G}$ between the nodes in S

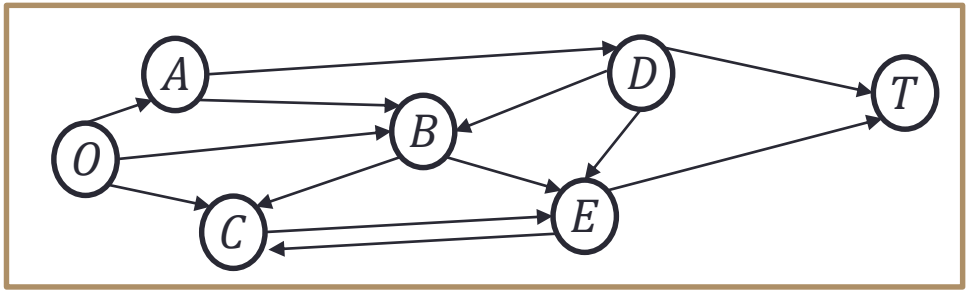


Network (Graph)

- **Undirected arc:** Arc has no direction and can be used to traverse both directions (such arcs are known as links/edges)
  - A network with undirected arcs is an **undirected network**



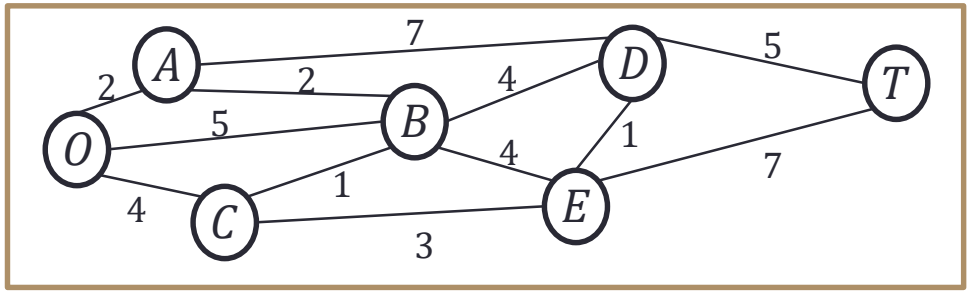
- **Directed arc:** Arc has a direction and can be used only in that direction
  - A network with directed arcs is a **directed network**

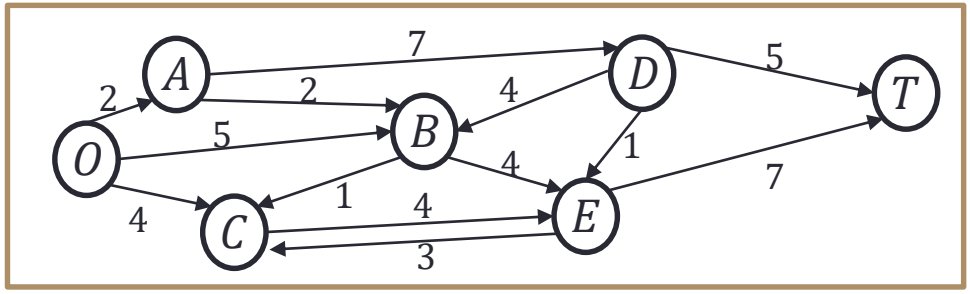


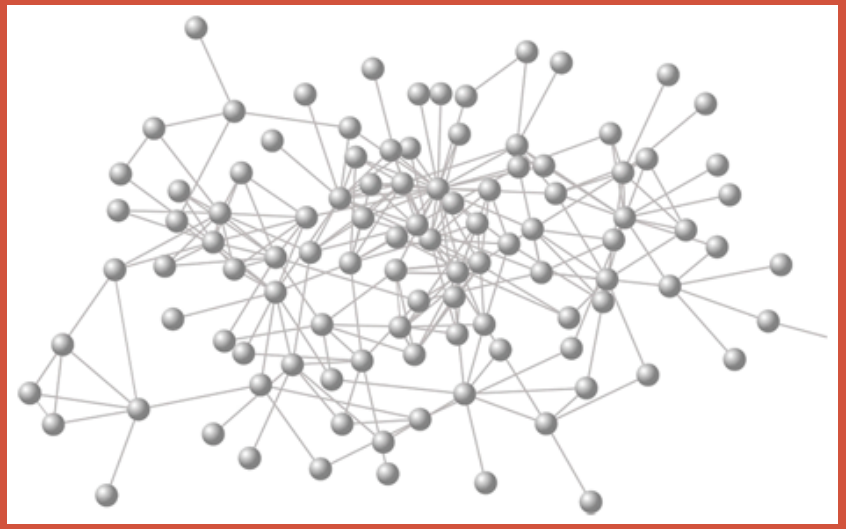
- A network is either a directed network or an undirected network i.e., all arcs are either directed or undirected simultaneously

Network (Graph)

- **Weighted network:** usually weights on arcs
  - E.g., Weights represent cost (time/distance) of using the arc or capacity of the arc
  - If weights are unspecified, then they are assumed to be one







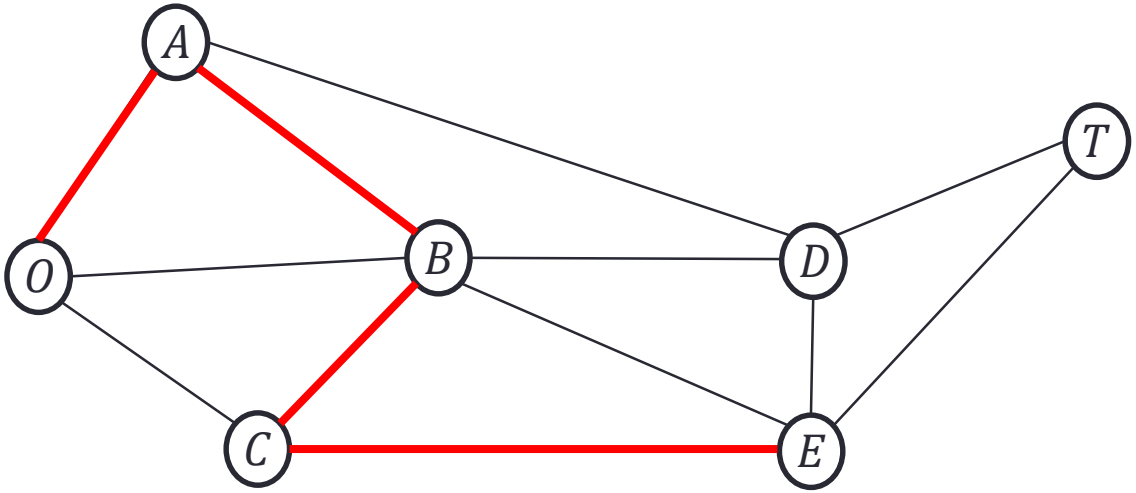
OPTIMIZATION PROBLEMS IN *UNDIRECTED* NETWORKS

MINIMUM SPANNING TREE PROBLEM

… where we see an algorithm to solve the minimum spanning tree problem

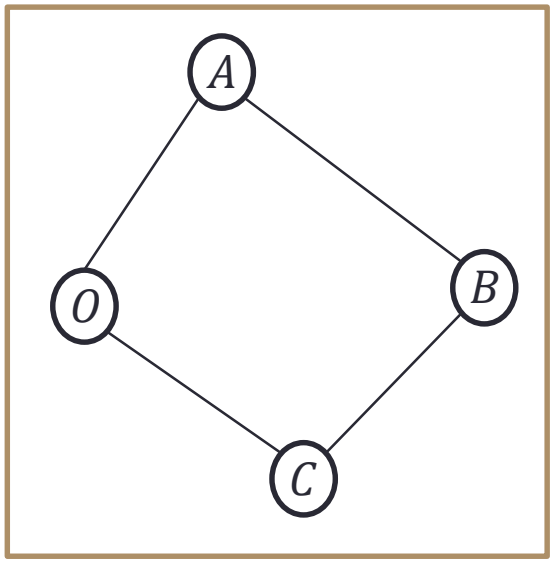
Paths

- **Undirected Path** from node i to node j: a sequence of *distinct* arcs connecting i to j with no node visited more than once

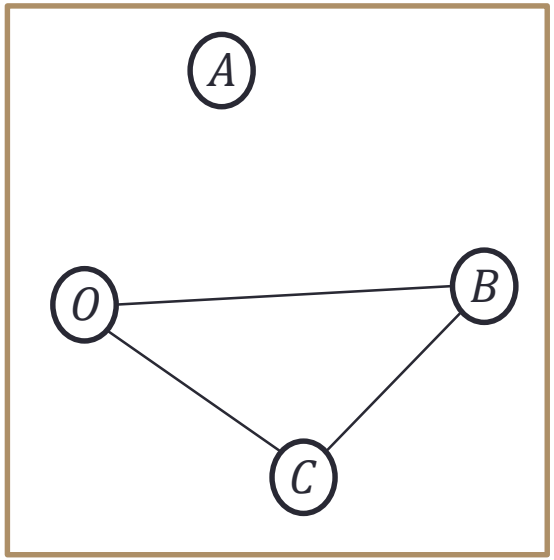


Connected networks

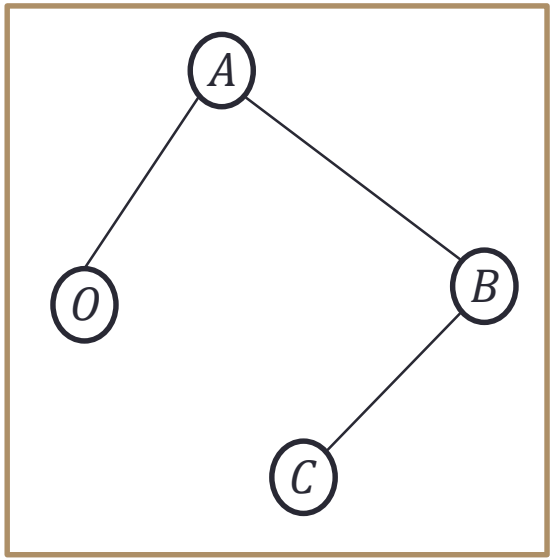
- **Connected network:** A network with a path between every pair of nodes



Connected network



Disconnected network

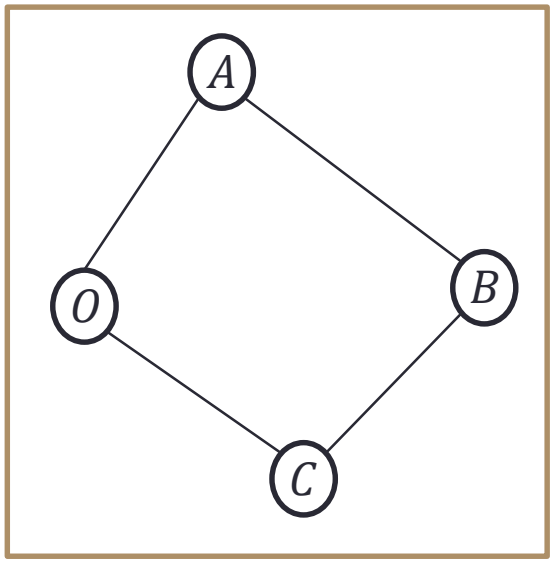


Connected network

We would like telephone networks, internet networks, powerline networks to be connected

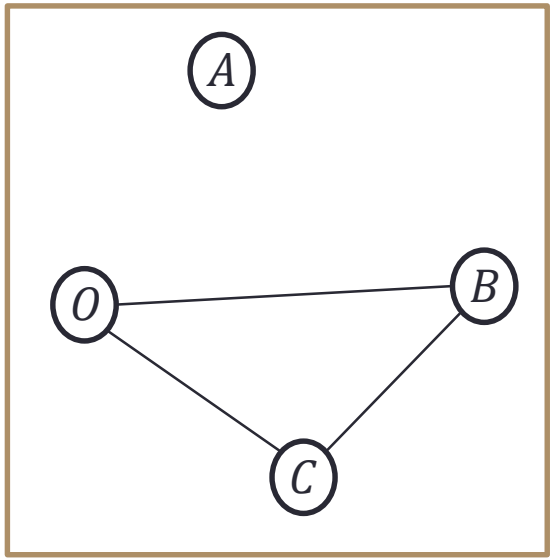
Trees

- **Tree**: A connected network $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ with $|\mathcal{A}| = |\mathcal{V}| - 1$



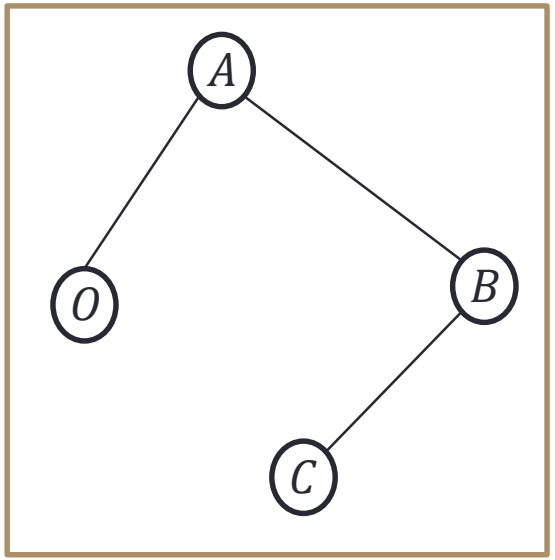
Connected network

Not a tree



Disconnected network

Not a tree

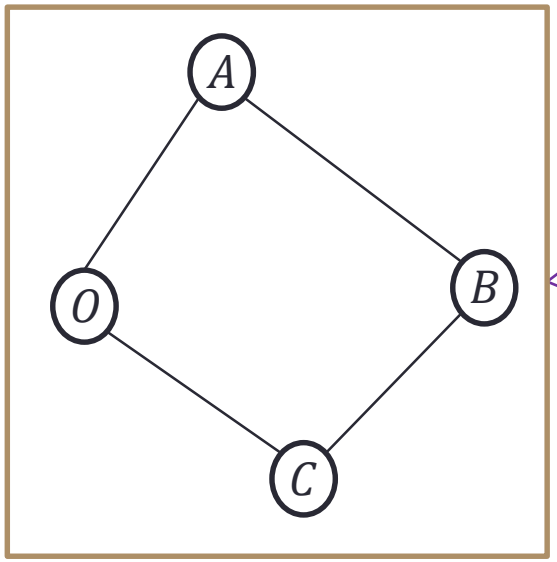


Connected network

Tree

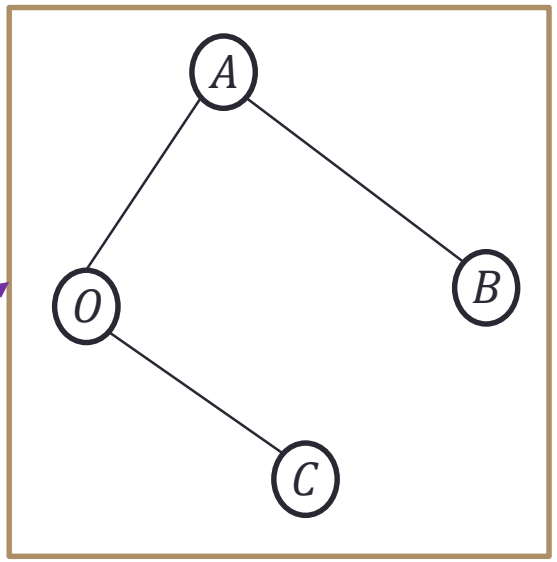
Spanning Trees

Spanning tree of a graph $\mathcal{H}$ is a subgraph that
1. is a tree and
2. contains all vertices of $\mathcal{H}$



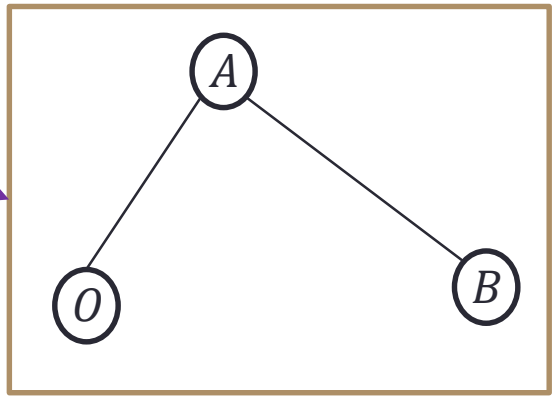
Connected network

Not a tree



Subgraph that is a tree

Spanning tree



Another subgraph that is a tree

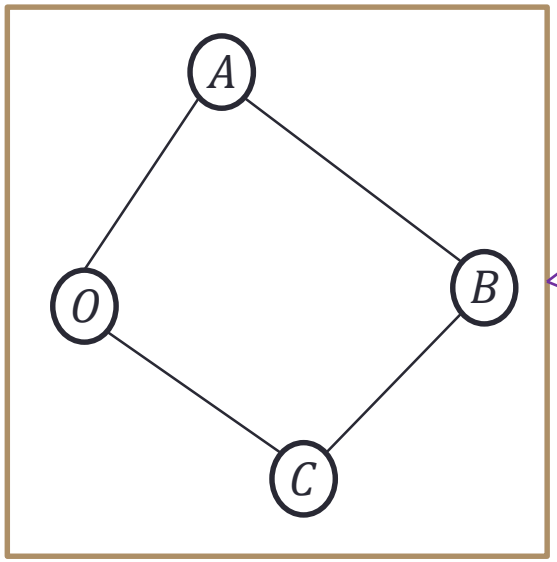
Not a spanning tree

Spanning Trees

- **Spanning tree** of a graph $\mathcal{H}$ is a subgraph that
  1. is a tree and
  2. contains all vertices of $\mathcal{H}$

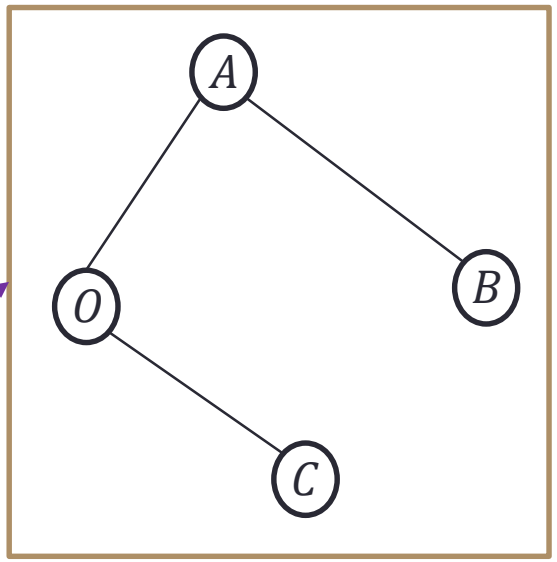
- Every connected graph has a spanning tree
  - It could have more than one spanning tree

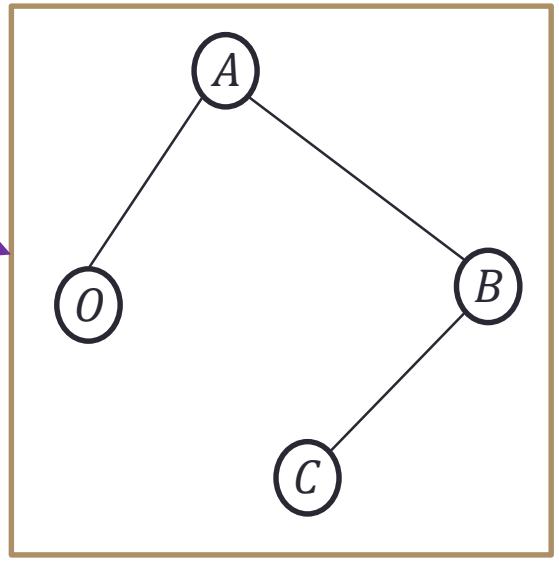
Spanning Trees



Connected network

Not a tree





- Every connected graph has a spanning tree
  - It could have more than one spanning tree

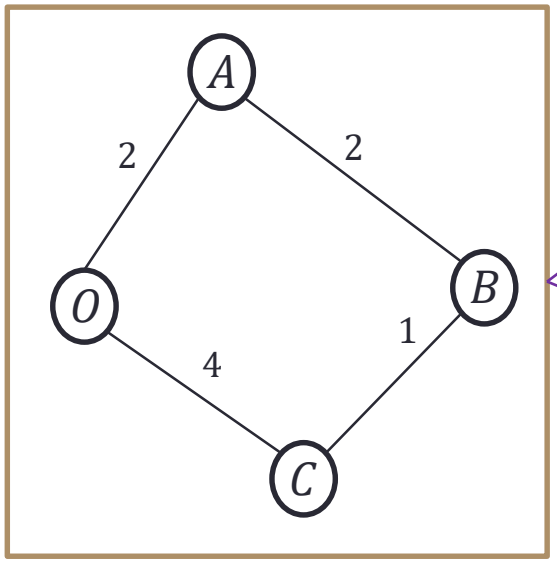
Spanning Trees

- **Spanning tree** of a graph $\mathcal{H}$ is a subgraph that
  1. is a tree and
  2. contains all vertices of $\mathcal{H}$

- Every connected graph has a spanning tree
  - It could have more than one spanning tree

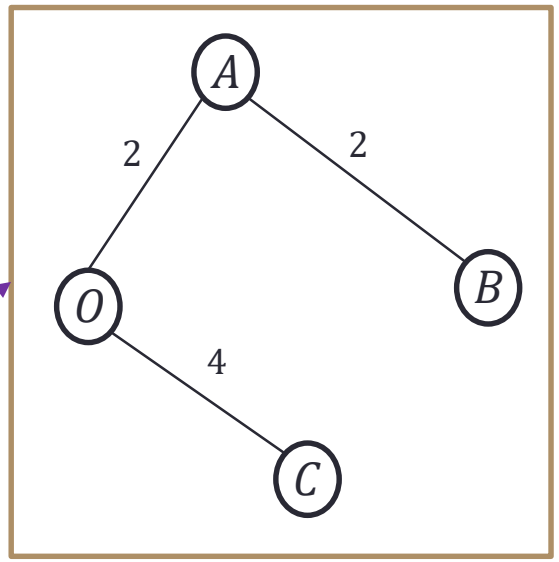
- Given a weighted graph $\mathcal{H}$, and a spanning tree $\mathcal{T}$,
  the **weight of $\mathcal{T}$** is the sum of the weights on the edges of $\mathcal{T}$

Spanning Trees

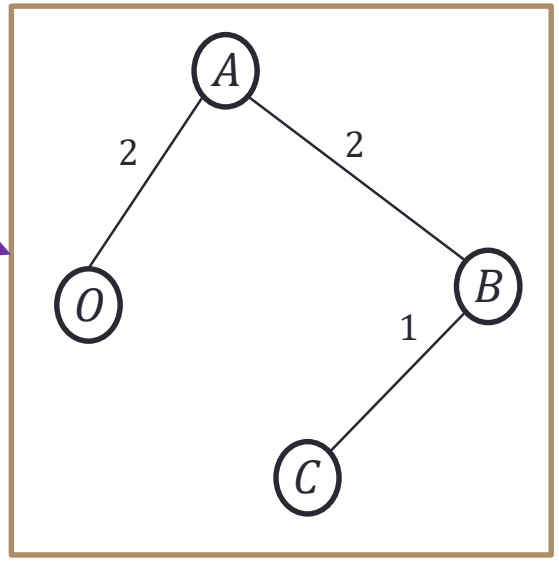


Connected network

Not a tree



Cost of the tree = 8

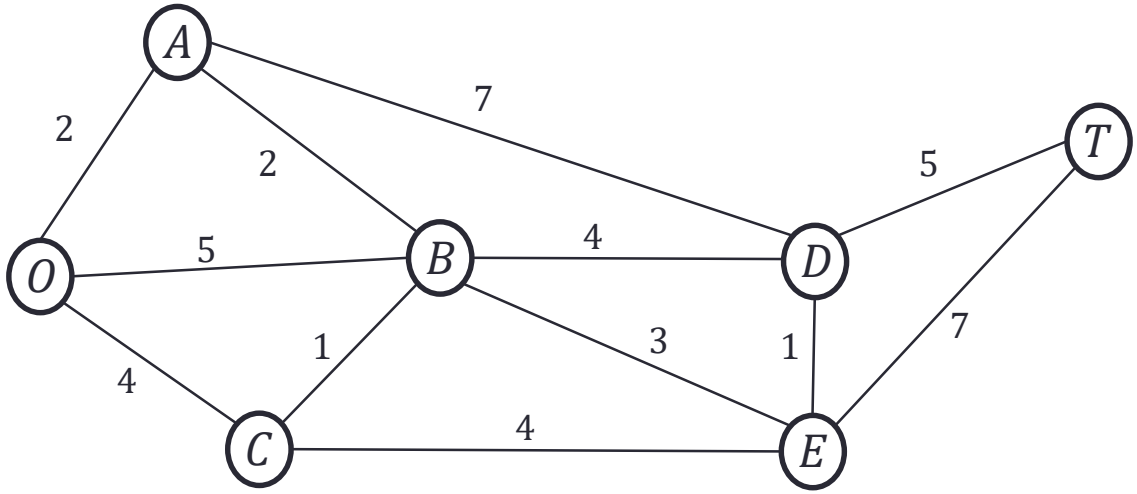


Cost of the tree = 5

Min Spanning Tree (MST) Problem

- Given: a weighted undirected graph $\mathcal{H}$
- **Goal:** Find a minimum weight spanning tree

Example:



Applications of the MST Problem

Design of

- Telecommunication networks
- Transportation networks
- Power transmission networks

- The edge weights represent the cost of laying the link
- We would like to achieve connectivity while minimizing the cost in all the above situations

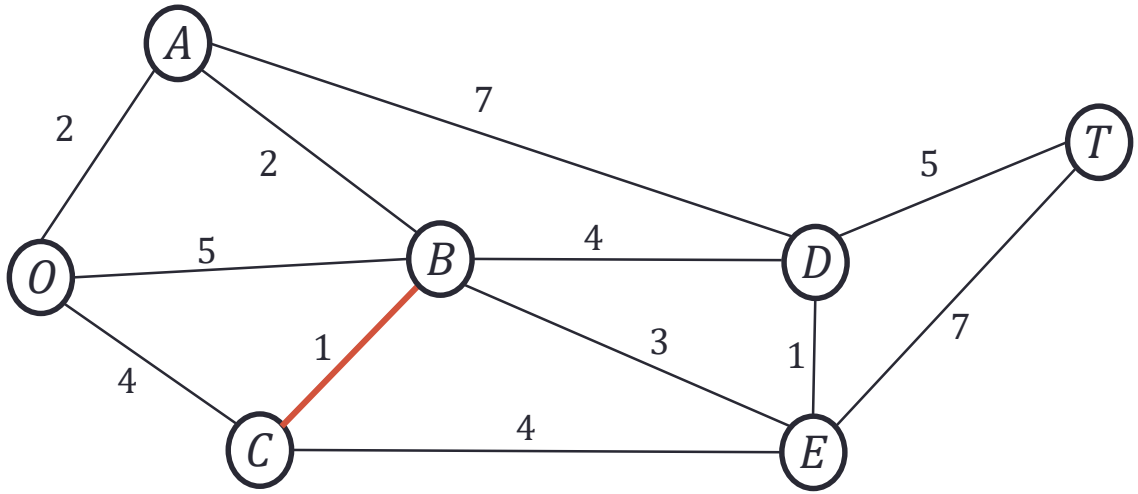
PRIM'S ALGORITHM

… a greedy approach that gives an optimal solution

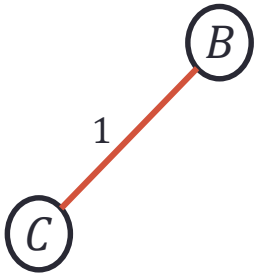
Prim's algorithm

Step 1: Select the cheapest edge and connect the nodes adjacent to that edge using that cheapest edge

Given weighted graph $\mathcal{H}$:



Tree after step 1:

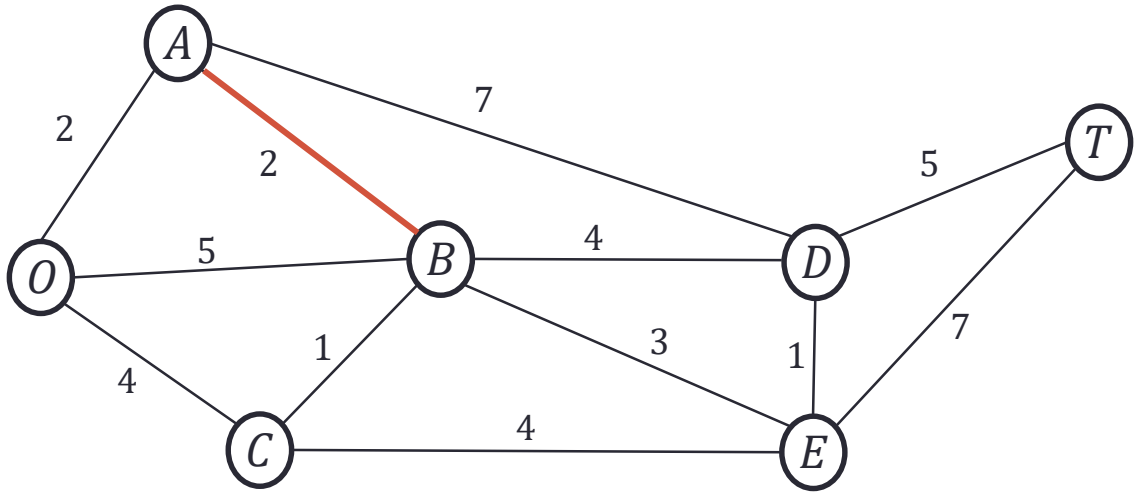


Connected nodes: B, C

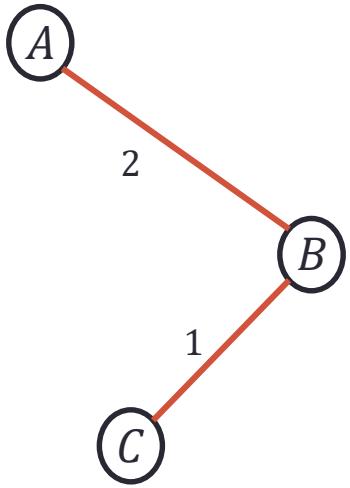
Prim's algorithm

Step 2: Find an unconnected node that has the cheapest edge to a connected node and add that edge

Given weighted graph $\mathcal{H}$:



Tree after step 2, iteration 1:

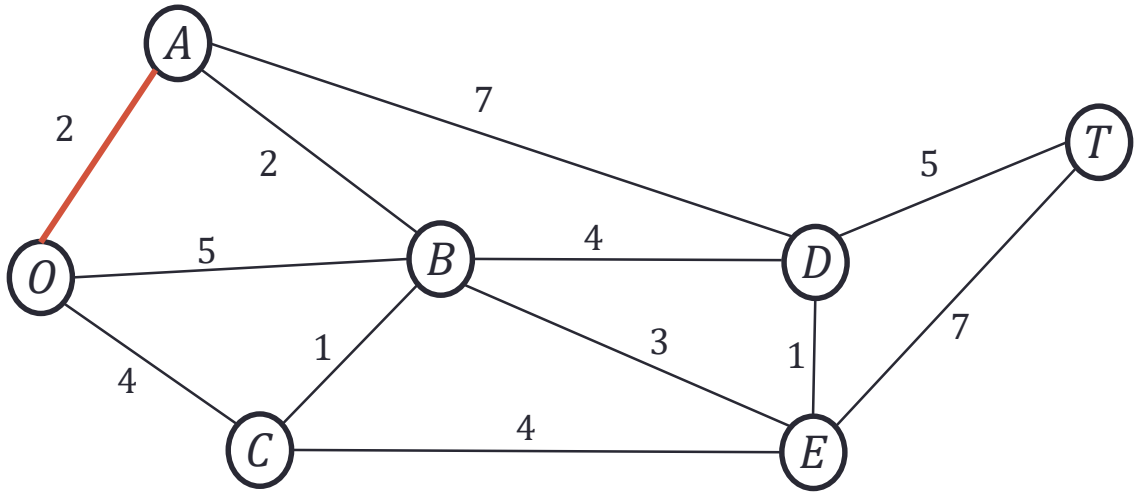


Connected nodes: B, C, A

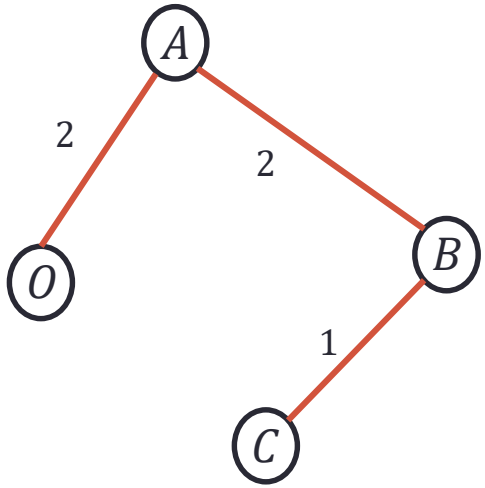
Prim's algorithm

Step 2: Find an unconnected node that has the cheapest edge to a connected node and add that edge

Given weighted graph $\mathcal{H}$:



Tree after step 2, iteration 2:

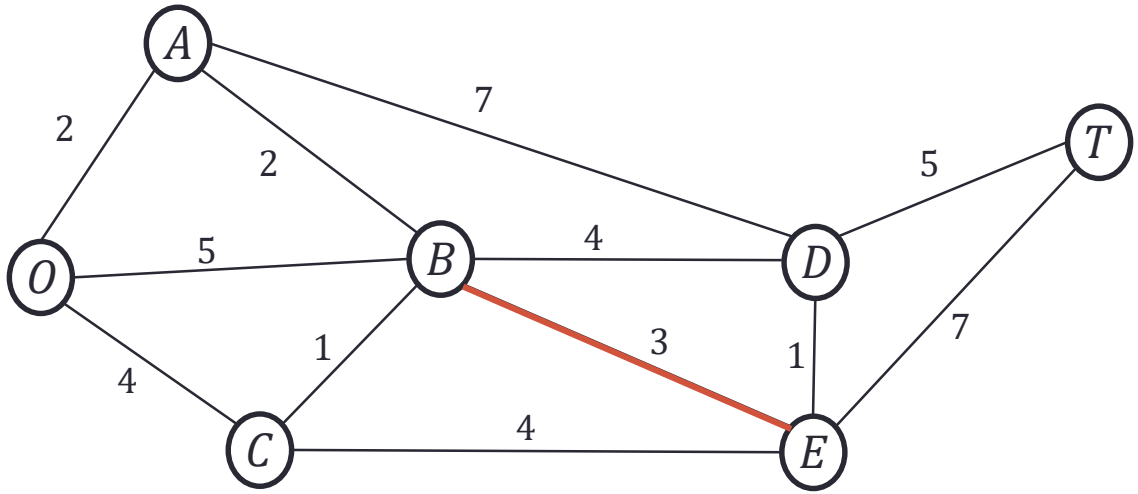


Connected nodes: B, C, A, O

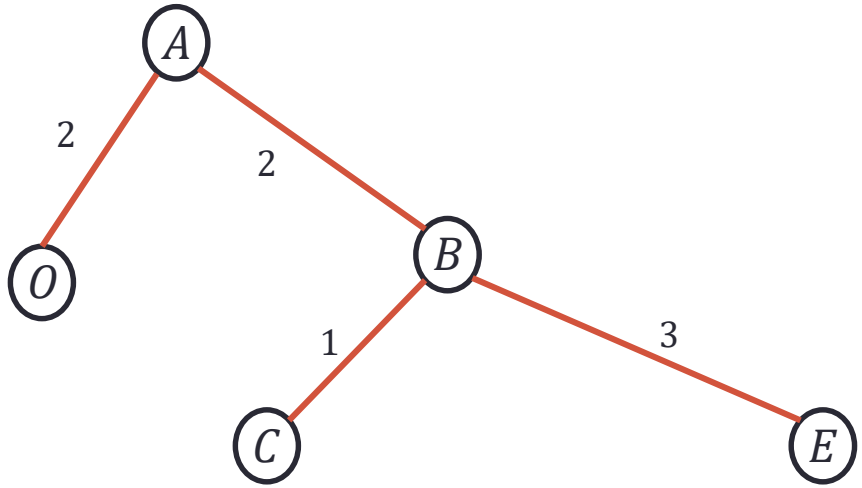
Prim's algorithm

Step 2: Find an unconnected node that has the cheapest edge to a connected node and add that edge

Given weighted graph $\mathcal{H}$:



Tree after step 2, iteration 3:

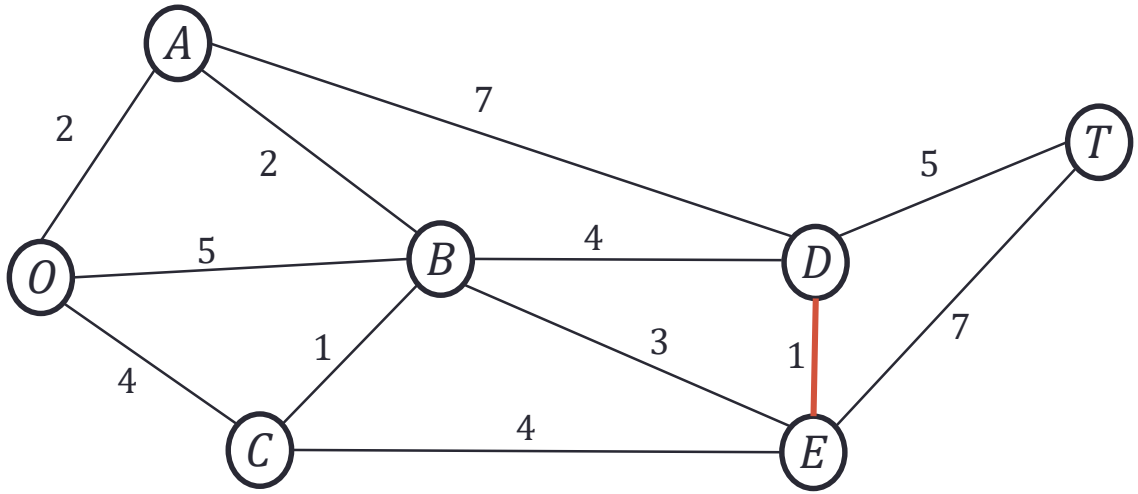


Connected nodes: B, C, A, O, E

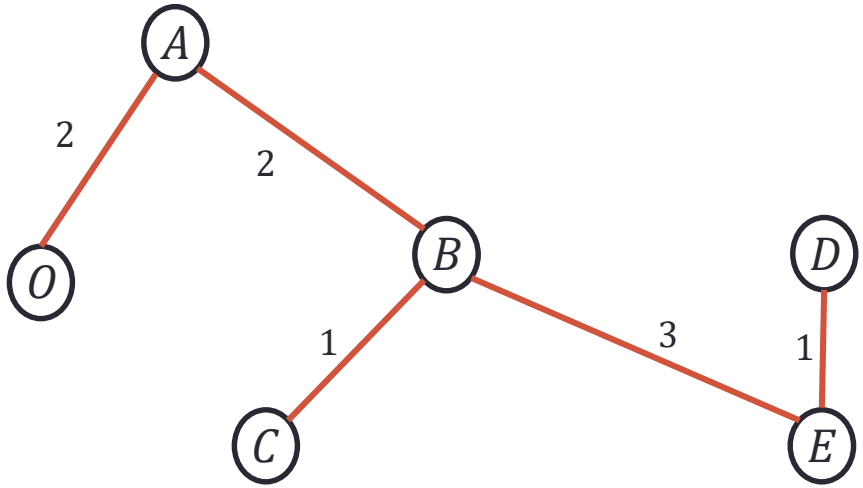
Prim's algorithm

Step 2: Find an unconnected node that has the cheapest edge to a connected node and add that edge

Given weighted graph $\mathcal{H}$:



Tree after step 2, iteration 4:

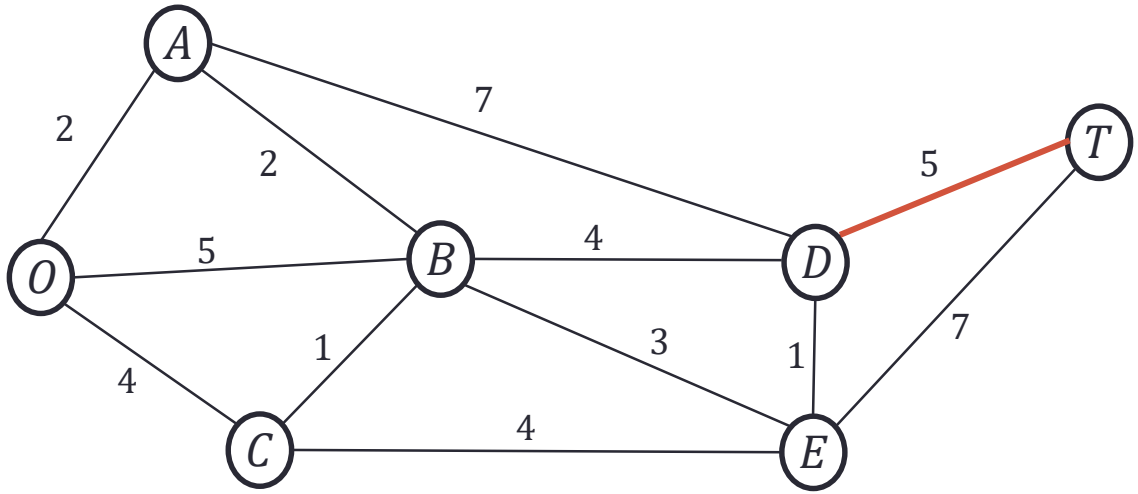


Connected nodes: B, C, A, O, E, D

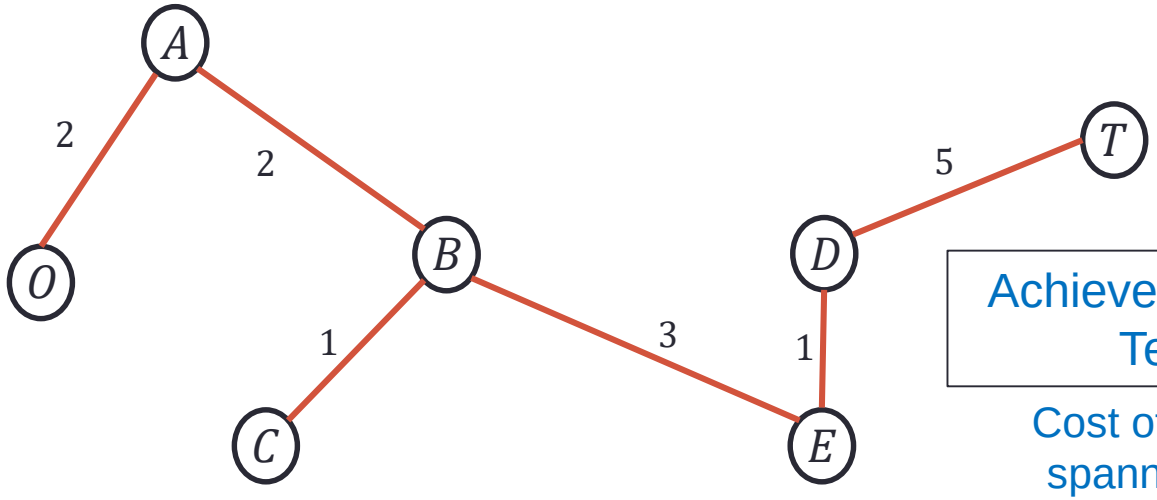
Prim's algorithm

Step 2: Find an unconnected node that has the cheapest edge to a connected node and add that edge

Given weighted graph $\mathcal{H}$:



Tree after step 2, iteration 5:



Connected nodes: B, C, A, O, E, D, T

Achieved connectivity!
Terminate

Cost of the cheapest spanning tree = 14

Prim's algorithm

Greedy Strategy, Builds a MST iteratively

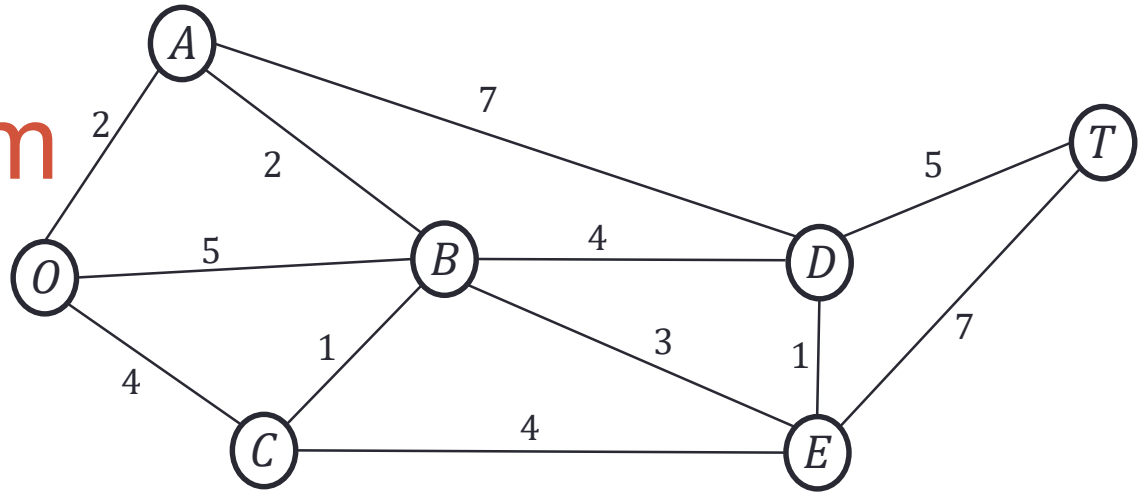
Step 1: Select the cheapest edge and connect the nodes adjacent to that edge using that cheapest edge

Step 2: Find an unconnected node that is closest to a connected node (has the cheapest edge to a connected node) and add that edge (break ties arbitrarily)

- Repeat Step 2 until all nodes have been connected

Prim's algorithm

Given weighted graph $\mathcal{H}$:



n	Connected nodes	Unconnected nodes	Cheapest edge to connected node	Cost of cheapest edge to connected node	Cost of cheapest edge	Cheapest edge	Newly connected node
1	B, C	O A D E T	CO BA BD BE –	4 2 4 3 –	2	BA	A
2	A, B, C	O D E T	AO BD BE –	2 4 3 –	2	AO	O
3	O, A, B, C	D E T	BD BE –	4 3 –	3	BE	E
4	O, A, B, C, E	D T	ED ET	1 7	1	ED	D
5	O, A, B, C, E, D	T	DT	5	5	DT	T