SCC.121: Fundamentals of Computer Science
Sorting, Trees and Graphs

Algorithmic Paradigms: Greedy Algorithms

Today’s Lecture

Aim:

- Introduce the concept of greedy algorithms
  - For various problems, including shortest paths
  - Existence of a solution does not imply greedy algorithm will find it.

- Describe greedy algorithms for the minimum-weight spanning tree (MST) problem.

Algorithmic paradigms

Generic framework that underlies a class of algorithms:

- Recursion
- Divide and conquer
- (Sweep-line algorithms)
- **Greedy algorithm**

Greedy algorithms

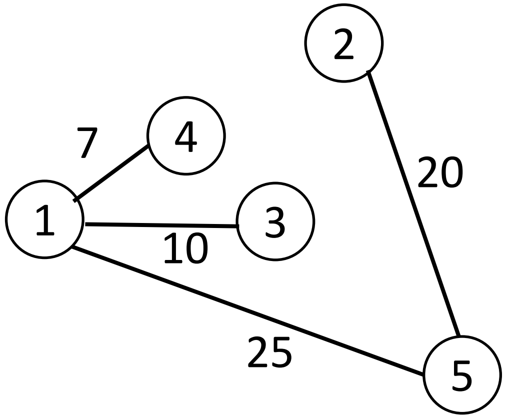
- Greedy algorithms:
  - Solve your problem in stages,
  - In each stage, choose the locally optimal choice.

- Greedy algorithms are **fast**! But for many problems, greedy can be **incorrect**, or give **non-optimal solutions**…

- Maybe surprisingly, it turns out that greedy algorithms:
  - Can approximate (for some problems) the optimal solution (and fast),
  - Solve some very well-known problem.

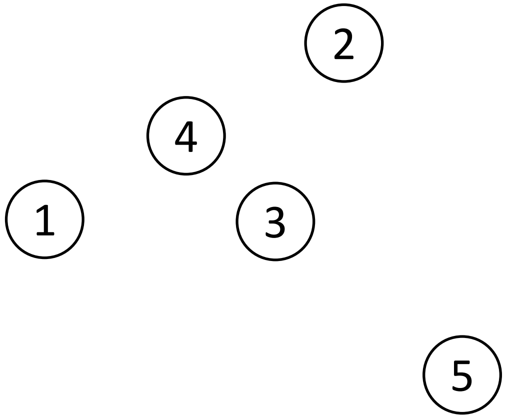
Greedy algorithm for the Euclidean Travelling Salesman Problem

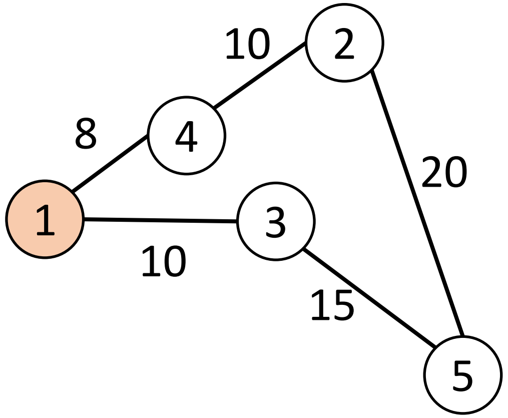
- **(Euclidean) Travelling Salesman Problem:**
  - Given a set of nodes (with associated 2D points) and a starting city, compute the shortest route that leaves the origin city, visits all other nodes exactly once and comes back to the origin city.
  - Where distances between nodes = distances between the corresponding points



Greedy algorithm for the Euclidean Travelling Salesman Problem

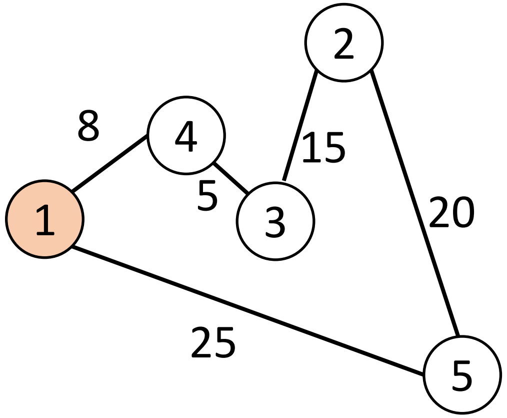
- **(Euclidean) Travelling Salesman Problem:**
  - Given a set of nodes (with associated 2D points) and a starting city, compute the shortest route that leaves the origin city, visits all other nodes exactly once and comes back to the origin city.
  - Where distances between nodes = distances between the corresponding points

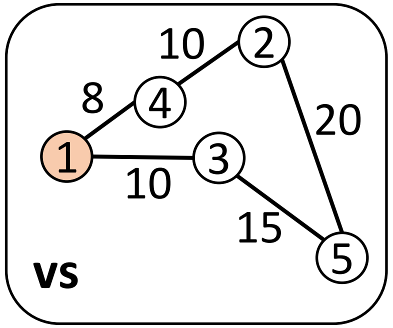




Greedy algorithm for the Euclidean Travelling Salesman Problem

- **Greedy algorithm for TSP, starting at node 1:**
  - Repeatedly visit nearest node (to current)
  - When you have visited all nodes, go back to origin city





(73 vs 63)

- **Greedy does not output the shortest route!**

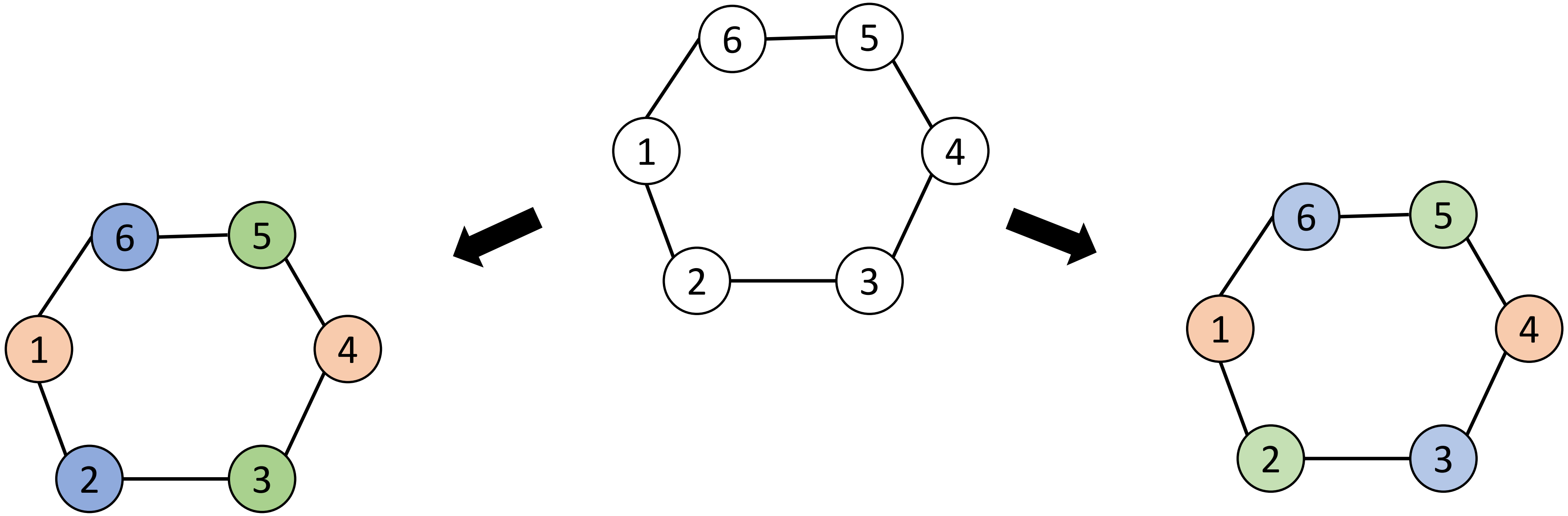
Greedy algorithm for shortest paths

- **Shortest path computation:**
  Given a directed weighted graph $G = (V, E)$, compute shortest path from the source node $s \in V$ to all other nodes in V.

- **Greedy Algorithm:**
  - Start at the source node, and visit in each stage, the (yet) unvisited node which is closest to the source.
  - This is **Dijkstra's algorithm**!
  - But importantly, it gives an **optimal solution to the shortest path computation problem.**

Greedy algorithm for vertex 3-coloring

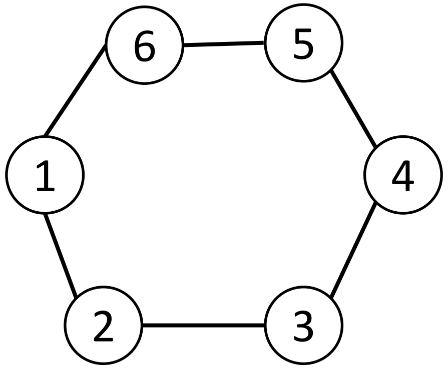
- **3-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2,3\}$ to all nodes such that no two neighbors have the same color

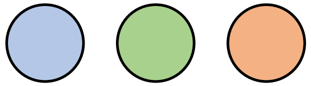


Greedy algorithm for vertex 3-coloring

- **3-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2,3\}$ to all nodes such that no two neighbors have the same color

- **Greedy 3-coloring algorithm:**
  - For $i = 1, \dots, |V|$:
    - Choose for node v_i the smallest color in $\{1,2,3\}$ that no neighbor has chosen already

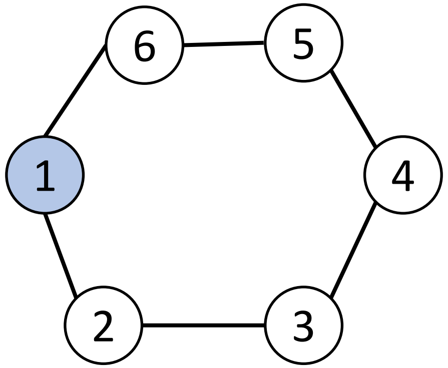


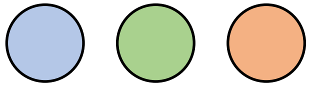
$\{1,2,3\}$ = 

Greedy algorithm for vertex 3-coloring

- **3-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2,3\}$ to all nodes such that no two neighbors have the same color

- **Greedy 3-coloring algorithm:**
  - For $i = 1, \dots, |V|$:
    - Choose for node v_i the smallest color in $\{1,2,3\}$ that no neighbor has chosen already



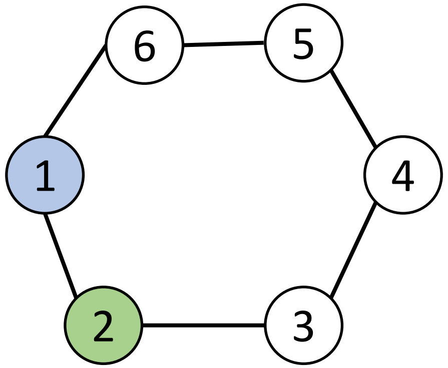
$\{1,2,3\}$ = 

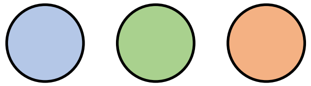
Greedy algorithm for vertex 3-coloring

Lancaster University

- **3-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2,3\}$ to all nodes such that no two neighbors have the same color

- **Greedy 3-coloring algorithm:**
  - For $i = 1, \dots, |V|$:
    - Choose for node v_i the smallest color in $\{1,2,3\}$ that no neighbor has chosen already

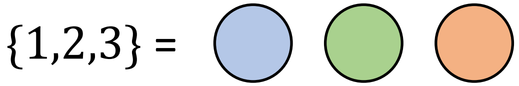


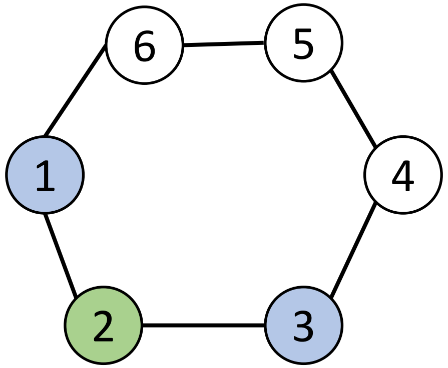
$\{1,2,3\}$ =


Greedy algorithm for vertex 3-coloring

- **3-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2,3\}$ to all nodes such that no two neighbors have the same color

- **Greedy 3-coloring algorithm:**
  - For $i = 1, \dots, |V|$:
    - Choose for node v_i the smallest color in $\{1,2,3\}$ that no neighbor has chosen already

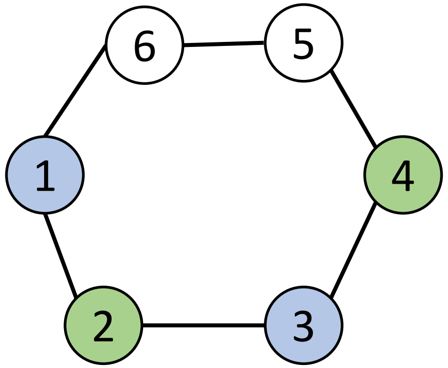


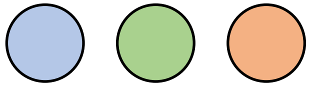


Greedy algorithm for vertex 3-coloring

- **3-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2,3\}$ to all nodes such that no two neighbors have the same color

- **Greedy 3-coloring algorithm:**
  - For $i = 1, \ldots, |V|$:
    - Choose for node v_i the smallest color in $\{1,2,3\}$ that no neighbor has chosen already

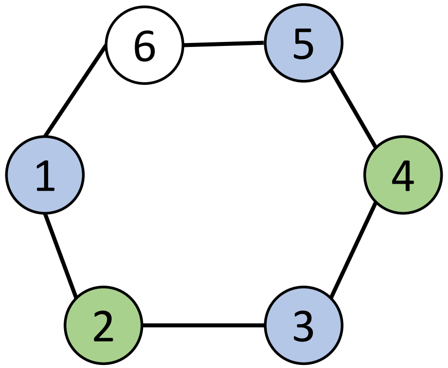


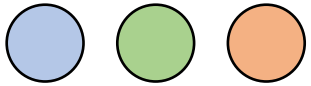
$\{1,2,3\}$ = 

Greedy algorithm for vertex 3-coloring

- **3-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2,3\}$ to all nodes such that no two neighbors have the same color

- **Greedy 3-coloring algorithm:**
  - For $i = 1, \dots, |V|$:
    - Choose for node v_i the smallest color in $\{1,2,3\}$ that no neighbor has chosen already

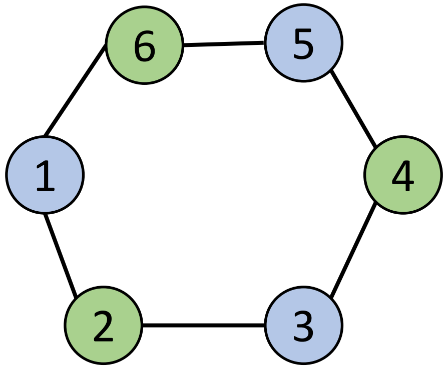


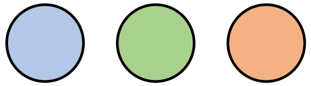
$\{1,2,3\}$ = 

Greedy algorithm for vertex 3-coloring

- **3-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2,3\}$ to all nodes such that no two neighbors have the same color

- **Greedy 3-coloring algorithm:**
  - For $i = 1, \dots, |V|$:
    - Choose for node v_i the smallest color in $\{1,2,3\}$ that no neighbor has chosen already

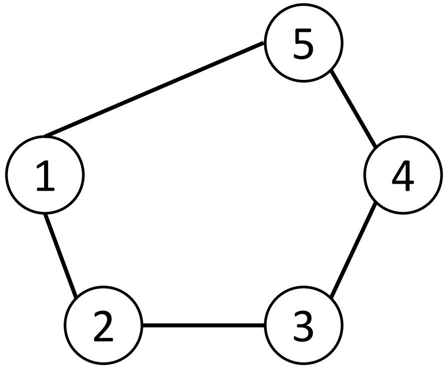


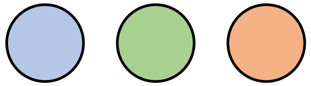
$\{1,2,3\}$ = 

Greedy algorithm for vertex 3-coloring

- **3-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2,3\}$ to all nodes such that no two neighbors have the same color

- **Greedy 3-coloring algorithm:**
  - For $i = 1, \dots, |V|$:
    - Choose for node v_i the smallest color in $\{1,2,3\}$ that no neighbor has chosen already

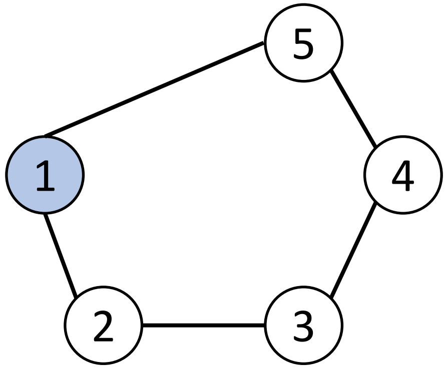


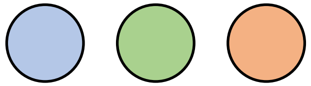
$\{1,2,3\}$ = 

Greedy algorithm for vertex 3-coloring

- **3-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2,3\}$ to all nodes such that no two neighbors have the same color

- **Greedy 3-coloring algorithm:**
  - For $i = 1, \dots, |V|$:
    - Choose for node v_i the smallest color in $\{1,2,3\}$ that no neighbor has chosen already



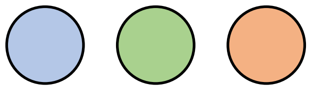
$\{1,2,3\}$ = 

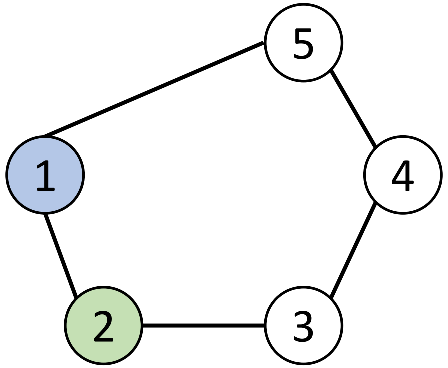
Greedy algorithm for vertex 3-coloring

- **3-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2,3\}$ to all nodes such that no two neighbors have the same color

- **Greedy 3-coloring algorithm:**
  - For $i = 1, \dots, |V|$:
    - Choose for node v_i the smallest color in $\{1,2,3\}$ that no neighbor has chosen already

$\{1,2,3\}$ =

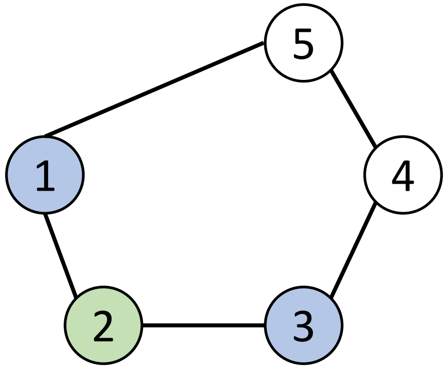


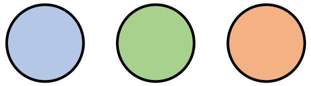


Greedy algorithm for vertex 3-coloring

- **3-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2,3\}$ to all nodes such that no two neighbors have the same color

- **Greedy 3-coloring algorithm:**
  - For $i = 1, \dots, |V|$:
    - Choose for node v_i the smallest color in $\{1,2,3\}$ that no neighbor has chosen already

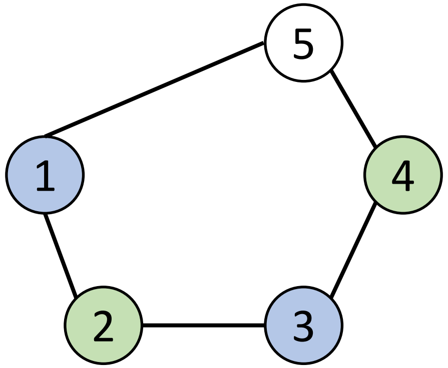


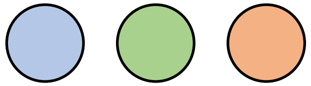
$\{1,2,3\}$ = 

Greedy algorithm for vertex 3-coloring

- **3-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2,3\}$ to all nodes such that no two neighbors have the same color

- **Greedy 3-coloring algorithm:**
  - For $i = 1, \dots, |V|$:
    - Choose for node v_i the smallest color in $\{1,2,3\}$ that no neighbor has chosen already

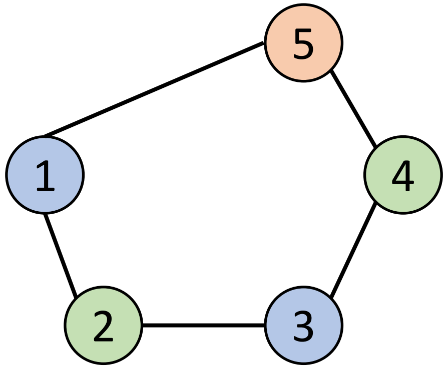


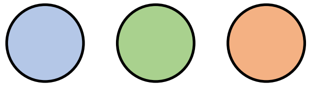
$\{1,2,3\}$ = 

Greedy algorithm for vertex 3-coloring

- **3-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2,3\}$ to all nodes such that no two neighbors have the same color

- **Greedy 3-coloring algorithm:**
  - For $i = 1, \dots, |V|$:
    - Choose for node v_i the smallest color in $\{1,2,3\}$ that no neighbor has chosen already

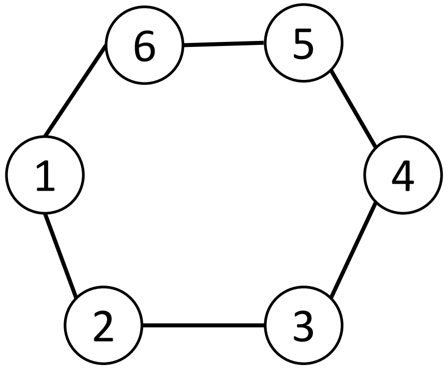


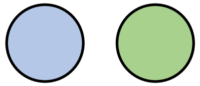
$\{1,2,3\}$ = 

Greedy algorithm for vertex 2-coloring

- **2-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2\}$ to all nodes such that no two neighbors have the same color

- **Greedy 2-coloring algorithm:**
  - For $i = 1, \dots, |V|$:
    - Choose for node v_i the smallest color in $\{1,2\}$ that no neighbor has chosen already

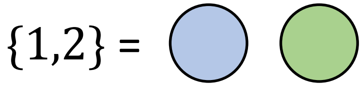


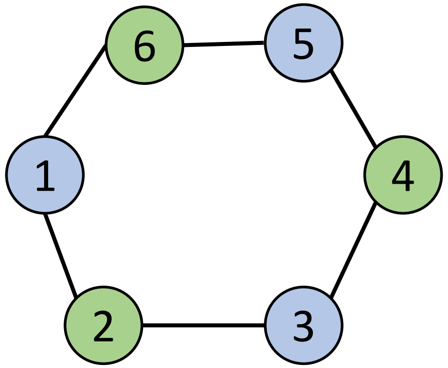
$\{1,2\}$ = 

Greedy algorithm for vertex 2-coloring

- **2-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2\}$ to all nodes such that no two neighbors have the same color

- **Greedy 2-coloring algorithm:**
  - For $i = 1, \dots, |V|$:
    - Choose for node v_i the smallest color in $\{1,2\}$ that no neighbor has chosen already

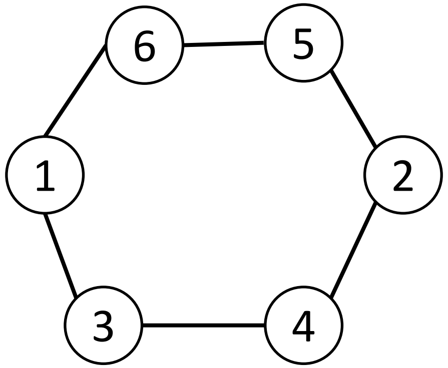


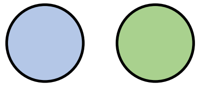


Greedy algorithm for vertex 2-coloring

- **2-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2\}$ to all nodes such that no two neighbors have the same color

- **Greedy 2-coloring algorithm:**
  - For $i = 1, \dots, |V|$:
    - Choose for node v_i the smallest color in $\{1,2\}$ that no neighbor has chosen already

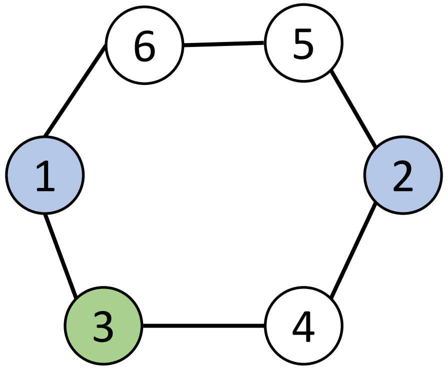


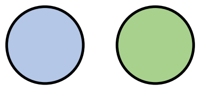
$\{1,2\}$ = 

Greedy algorithm for vertex 2-coloring

- **2-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2\}$ to all nodes such that no two neighbors have the same color

- **Greedy 2-coloring algorithm:**
  - For $i = 1, \ldots, |V|$:
    - Choose for node v_i the smallest color in $\{1,2\}$ that no neighbor has chosen already



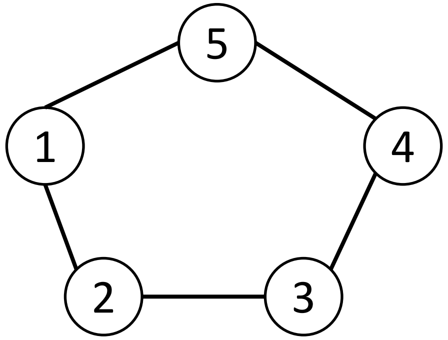
$\{1,2\}$ = 

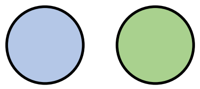
What color to give to node 4?
Algorithm fails…

Greedy algorithm for vertex 2-coloring

- **2-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2\}$ to all nodes such that no two neighbors have the same color

- **Greedy 2-coloring algorithm:**
  - For $i = 1, \dots, |V|$:
    - Choose for node v_i the smallest color in $\{1,2\}$ that no neighbor has chosen already

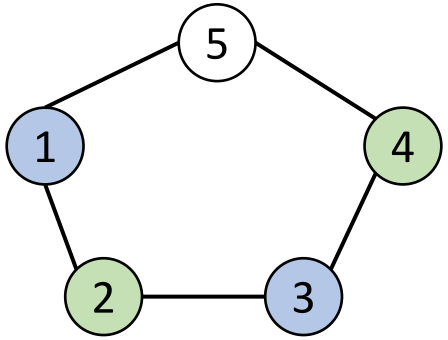


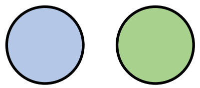
$\{1,2\}$ = 

Greedy algorithm for vertex 2-coloring

- **2-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2\}$ to all nodes such that no two neighbors have the same color

- **Greedy 2-coloring algorithm:**
  - For $i = 1, \ldots, |V|$:
    - Choose for node v_i the smallest color in $\{1,2\}$ that no neighbor has chosen already



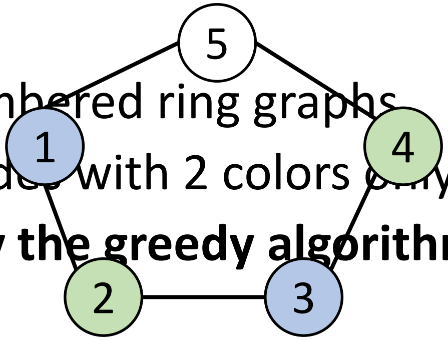
$\{1,2\}$ = 

What color to give to node 5?
Algorithm fails...

Greedy algorithm for vertex 2-coloring

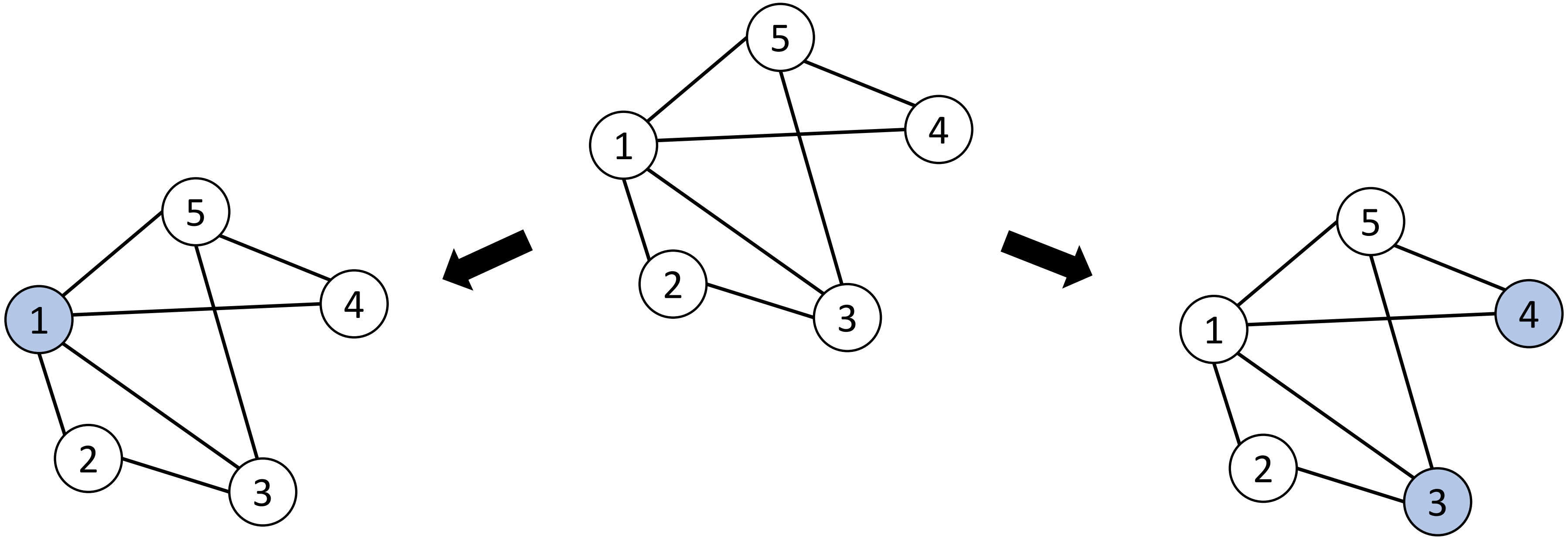
- **2-coloring (ring) graphs:**
  Given an undirected ring graph $G = (V, E)$, assign a color in $\{1,2\}$ to all nodes such that no two neighbors have the same color

- Although it is similar to 3-coloring problem:
  - Greedy algorithm will fail on odd-numbered ring graphs, **because these graphs cannot be colored using only 2 colors.**
  - <u>But more importantly:</u>
    - Greedy algorithm will also fail on even-numbered ring graphs
    - Even though there is a way to color the nodes with 2 colors only!
    - **The existence of a solution does not imply the greedy algorithm will find that solution.**



Greedy algorithm for maximal independent set

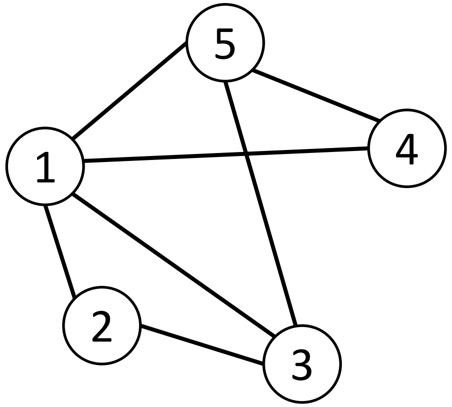
- **Maximal Independent Set (MIS):**
  Given an undirected graph $G = (V, E)$, compute a subset of nodes $I \subseteq V$ such that there are no two neighbors in I, and all nodes not in I have a neighbor in I.



Greedy algorithm for maximal independent set

- **Maximal Independent Set (MIS):**
  Given an undirected graph $G = (V, E)$, compute a subset of nodes $I \subseteq V$ such that there are no two neighbors in I, and all nodes not in I have a neighbor in I.

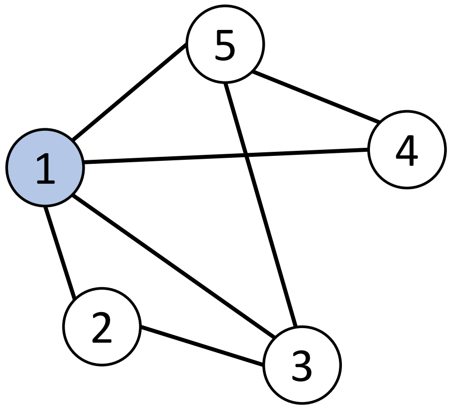
- **Greedy MIS algorithm:**
  - Initialize $I = \emptyset$
  - For $i = 1, \dots, |V|$:
    - Check if node v_i has any neighbors in I.
    - If not, then add v_i to I, or also: $I = I \cup \{v_i\}$



Greedy algorithm for maximal independent set

- **Maximal Independent Set (MIS):**
  Given an undirected graph $G = (V, E)$, compute a subset of nodes $I \subseteq V$ such that there are no two neighbors in I, and all nodes not in I have a neighbor in I.

- **Greedy MIS algorithm:**
  - Initialize $I = \emptyset$
  - For $i = 1, \ldots, |V|$:
    - Check if node v_i has any neighbors in I.
    - If not, then add v_i to I, or also: $I = I \cup \{v_i\}$

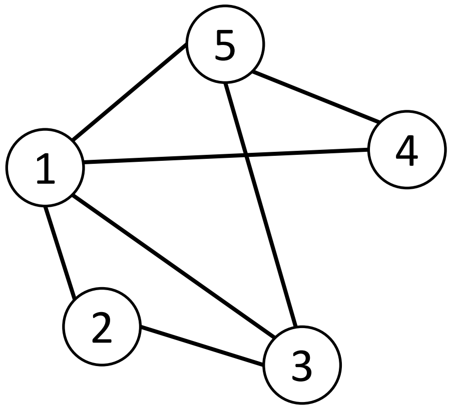


- Outputs a correct solution (i.e., an MIS).

Greedy algorithm for maximal independent set

- **Maximal Independent Set (MIS):**
  Given an undirected graph $G = (V, E)$, compute a subset of nodes $I \subseteq V$ such that there are no two neighbors in I, and all nodes not in I have a neighbor in I.

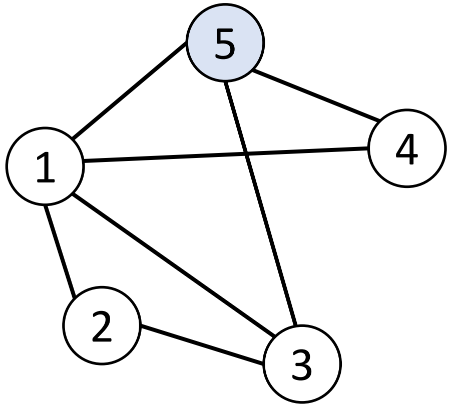
- **Greedy MIS algorithm:**
  - Initialize $I = \emptyset$
  - For $i = 1, \dots, |V|$:
    - Check if node v_i has any neighbors in I.
    - If not, then add v_i to I, or also: $I = I \cup \{v_i\}$



Greedy algorithm for maximal independent set

- **Maximal Independent Set (MIS):**
  Given an undirected graph $G = (V, E)$, compute a subset of nodes $I \subseteq V$ such that there are no two neighbors in I, and all nodes not in I have a neighbor in I.

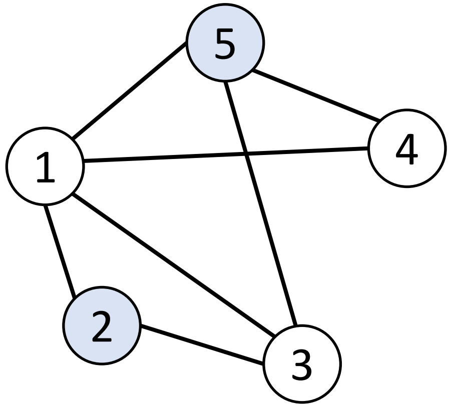
- **Greedy MIS algorithm:**
  - Initialize $I = \emptyset$
  - For $i = 1, \dots, |V|$:
    - Check if node v_i has any neighbors in I.
    - If not, then add v_i to I, or also: $I = I \cup \{v_i\}$



Greedy algorithm for maximal independent set

- **Maximal Independent Set (MIS):**
  Given an undirected graph $G = (V, E)$, compute a subset of nodes $I \subseteq V$ such that there are no two neighbors in I, and all nodes not in I have a neighbor in I.

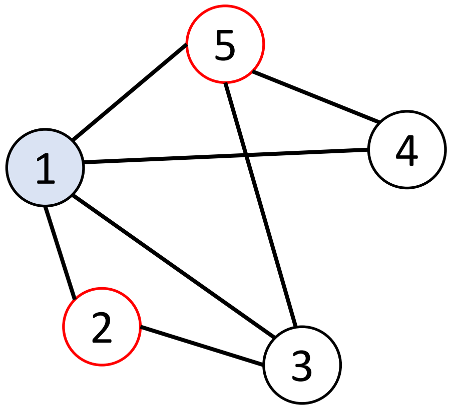
- **Greedy MIS algorithm:**
  - Initialize $I = \emptyset$
  - For $i = 1, \dots, |V|$:
    - Check if node v_i has any neighbors in I.
    - If not, then add v_i to I, or also: $I = I \cup \{v_i\}$



Greedy algorithm for maximum independent set

- **<u>Maximum</u> Independent Set (MaxIS):**
  Given an undirected graph $G = (V, E)$, compute **the largest subset** of nodes $I \subseteq V$ such that there are no two neighbors in I, and all nodes not in I have a neighbor in I.

- **Greedy algorithm:**
  - Initialize $I = \emptyset$
  - For $i = 1, \ldots, |V|$:
    - Check if node v_i has any neighbors in I.
    - If not, then add v_i to I, or also: $I = I \cup \{v_i\}$



- **Will not output a correct solution** (MaxIS) but can output an **approximation if all nodes have few neighbors.**

Summary for greedy algorithms

- **Greedy algorithms:**
  - Solve your problem in stages,
  - In each stage, choose the locally optimal choice.

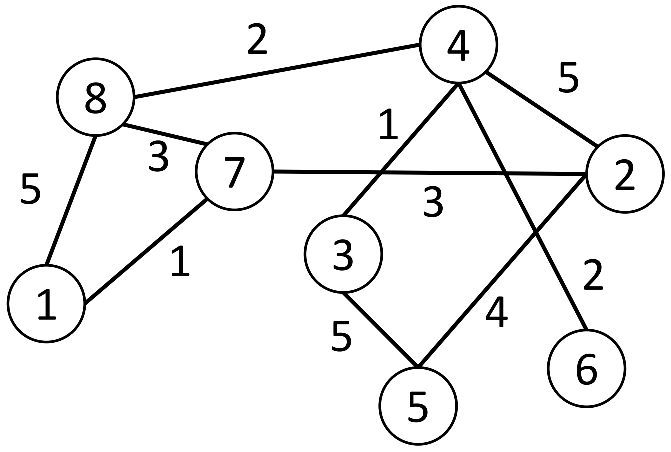
- Greedy algorithms are **fast**! But for many problems, greedy can be **incorrect**, or give **non-optimal solutions**…

- Maybe surprisingly, it turns out that greedy algorithms:
  - Can approximate (for some problems) the optimal solution (and fast),
  - Solve some very well-known problem.

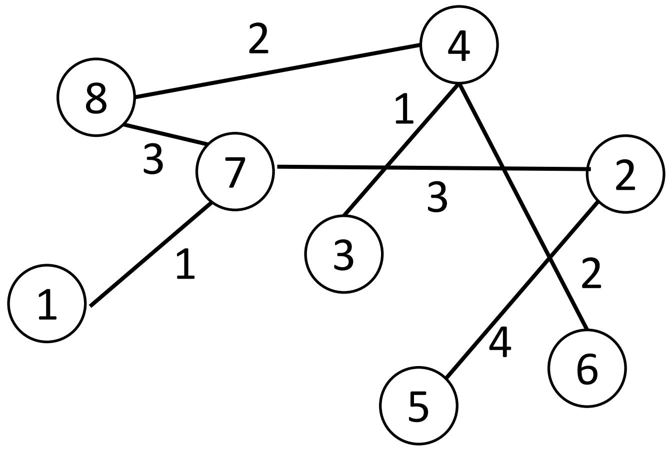
Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

G



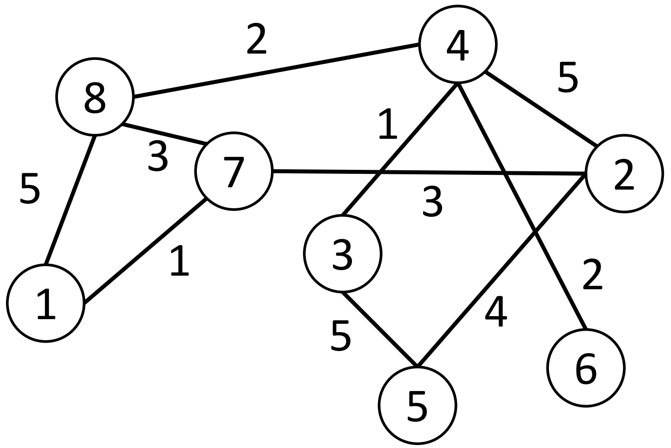
$MST(G)$



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

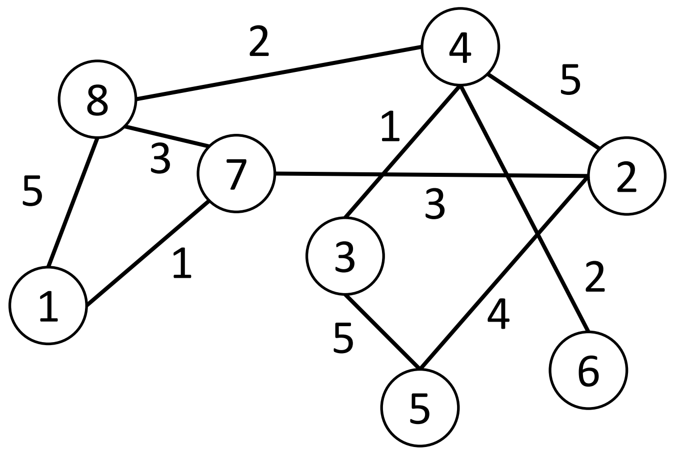
- **Greedy algorithm?**



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

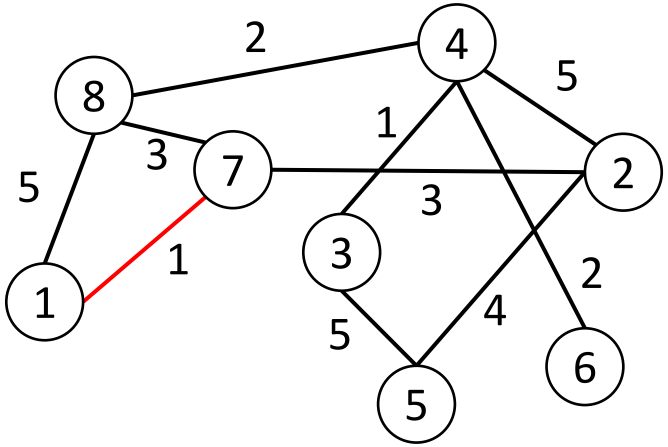
- **Kruskal's algorithm:**
  - Start with tree $T = \emptyset$,
  - Sort edges from lowest to highest weight,
  - For $i = 1, \dots, |E|$:
    - If edge e_i does not form a cycle when added to T, then add e_i to T, or also: $T = T \cup \{e_i\}$



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

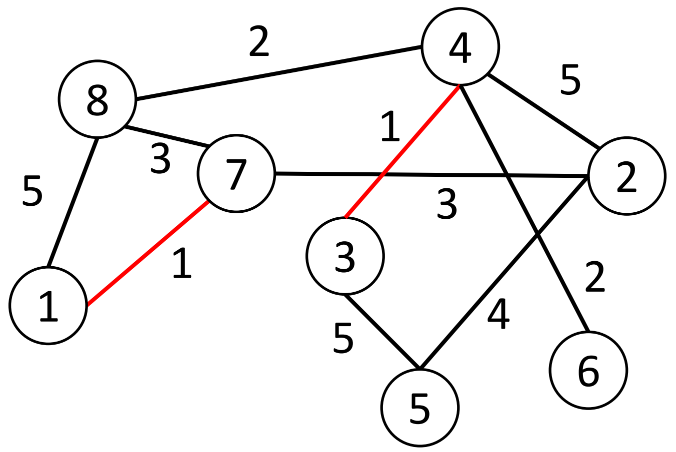
- **Kruskal's algorithm:**
  - Start with tree $T = \emptyset$,
  - Sort edges from lowest to highest weight,
  - For $i = 1, \dots, |E|$:
    - If edge e_i does not form a cycle when added to T, then add e_i to T, or also: $T = T \cup \{e_i\}$



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

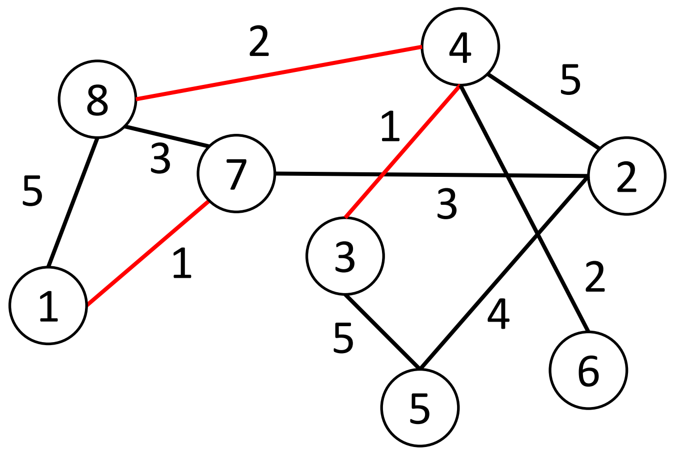
- **Kruskal's algorithm:**
  - Start with tree $T = \emptyset$,
  - Sort edges from lowest to highest weight,
  - For $i = 1, \dots, |E|$:
    - If edge e_i does not form a cycle when added to T, then add e_i to T, or also: $T = T \cup \{e_i\}$



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

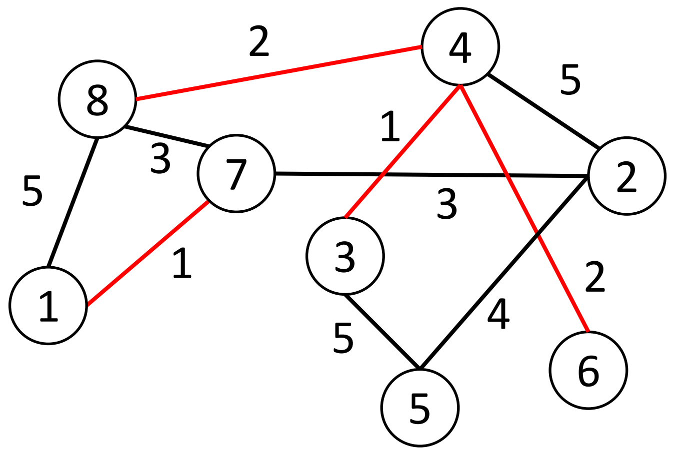
- **Kruskal's algorithm:**
  - Start with tree $T = \emptyset$,
  - Sort edges from lowest to highest weight,
  - For $i = 1, \dots, |E|$:
    - If edge e_i does not form a cycle when added to T, then add e_i to T, or also: $T = T \cup \{e_i\}$



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:** Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

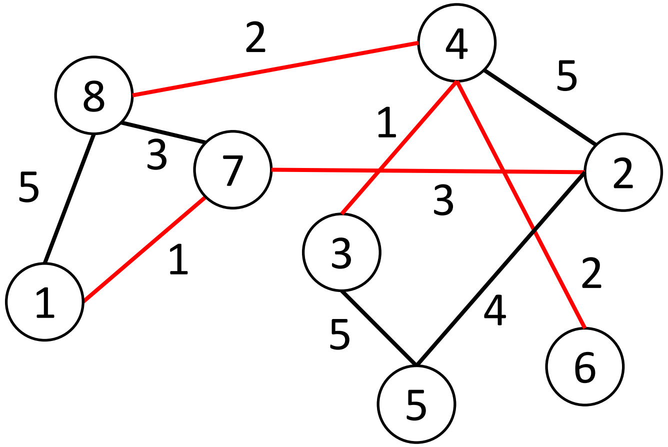
- **Kruskal's algorithm:**
  - Start with tree $T = \emptyset$,
  - Sort edges from lowest to highest weight,
  - For $i = 1, \dots, |E|$:
    - If edge e_i does not form a cycle when added to T, then add e_i to T, or also: $T = T \cup \{e_i\}$



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

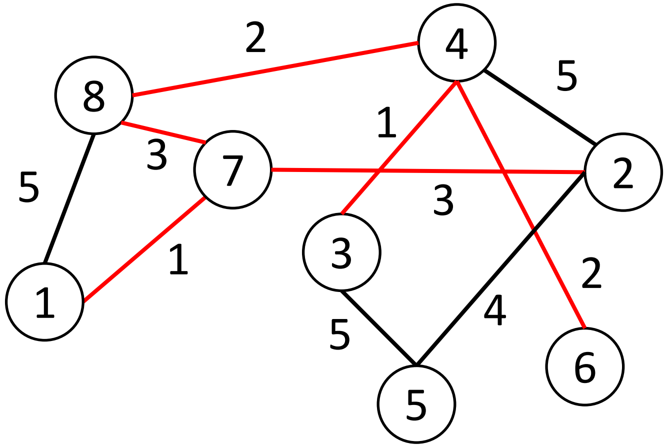
- **Kruskal's algorithm:**
  - Start with tree $T = \emptyset$,
  - Sort edges from lowest to highest weight,
  - For $i = 1, \ldots, |E|$:
    - If edge e_i does not form a cycle when added to T, then add e_i to T, or also: $T = T \cup \{e_i\}$



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

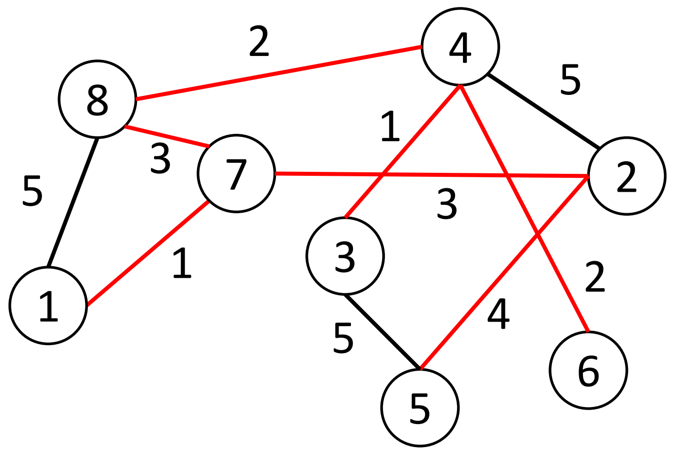
- **Kruskal's algorithm:**
  - Start with tree $T = \emptyset$,
  - Sort edges from lowest to highest weight,
  - For $i = 1, \dots, |E|$:
    - If edge e_i does not form a cycle when added to T, then add e_i to T, or also: $T = T \cup \{e_i\}$



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

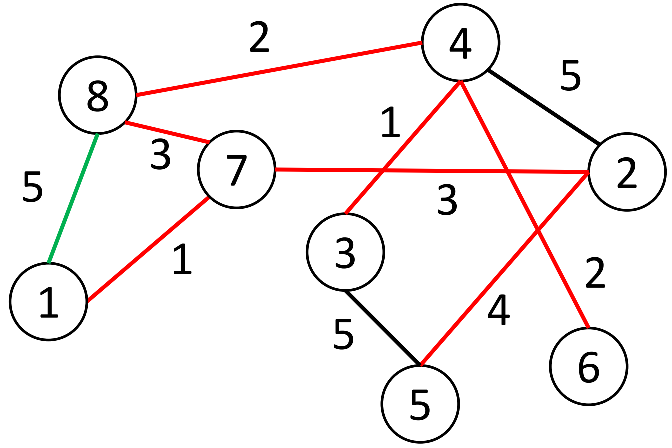
- **Kruskal's algorithm:**
  - Start with tree $T = \emptyset$,
  - Sort edges from lowest to highest weight,
  - For $i = 1, \dots, |E|$:
    - If edge e_i does not form a cycle when added to T, then add e_i to T, or also: $T = T \cup \{e_i\}$



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

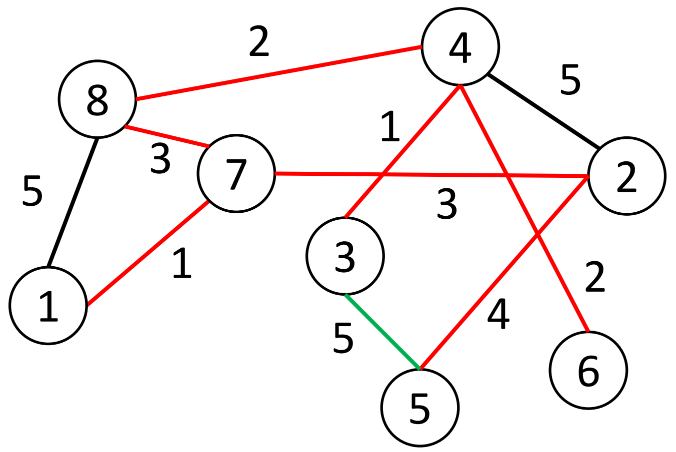
- **Kruskal's algorithm:**
  - Start with tree $T = \emptyset$,
  - Sort edges from lowest to highest weight,
  - For $i = 1, \dots, |E|$:
    - If edge e_i does not form a cycle when added to T, then add e_i to T, or also: $T = T \cup \{e_i\}$



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

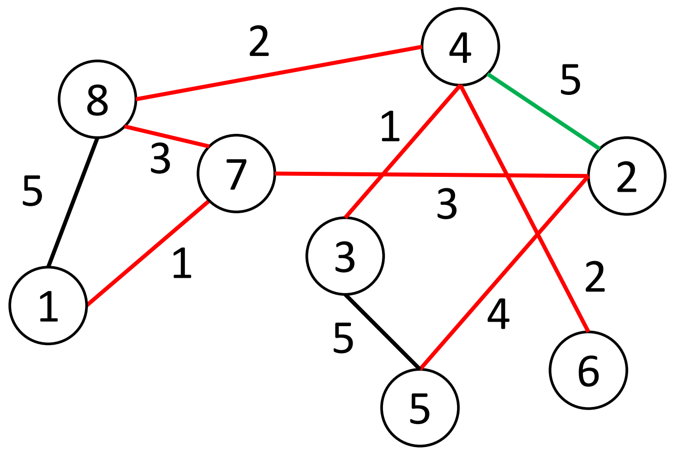
- **Kruskal's algorithm:**
  - Start with tree $T = \emptyset$,
  - Sort edges from lowest to highest weight,
  - For $i = 1, \dots, |E|$:
    - If edge e_i does not form a cycle when added to T, then add e_i to T, or also: $T = T \cup \{e_i\}$



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

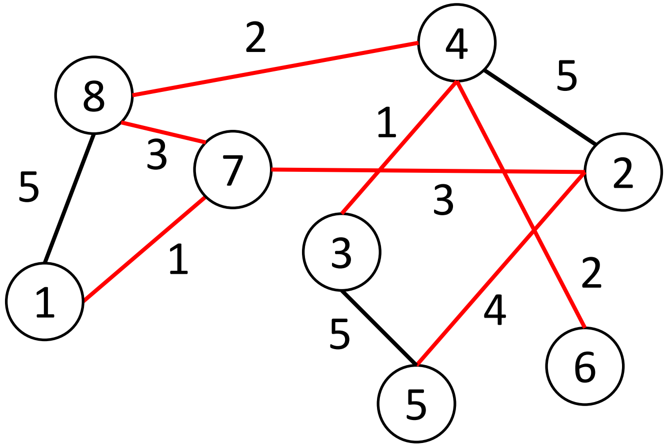
- **Kruskal's algorithm:**
  - Start with tree $T = \emptyset$,
  - Sort edges from lowest to highest weight,
  - For $i = 1, \dots, |E|$:
    - If edge e_i does not form a cycle when added to T, then add e_i to T, or also: $T = T \cup \{e_i\}$



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

- **Kruskal's algorithm:**
  - Start with tree $T = \emptyset$,
  - Sort edges from lowest to highest weight,
  - For $i = 1, \dots, |E|$:
    - If edge e_i does not form a cycle when added to T, then add e_i to T, or also: $T = T \cup \{e_i\}$



Greedy algorithms for minimum-weight spanning trees (MST)

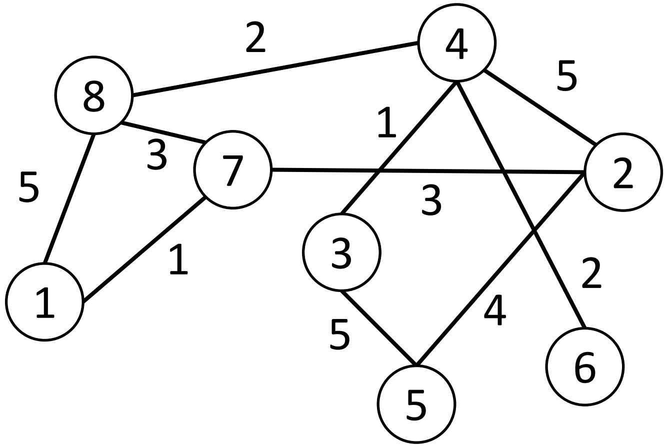
- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

- **Kruskal's algorithm:**
  - Takes $O(|E| \log|V|)$ worst-case time
    - $O(|E| \log|V|)$ time for sorting edges
    - Checking for all $|E|$ edges whether they create a cycle is a bit harder to bound.
  - Output is a spanning tree is trivial to show (spanning + no cycles)
  - Minimum-weight can be shown by proof of induction.
    - Induction step is a bit tricky.

Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

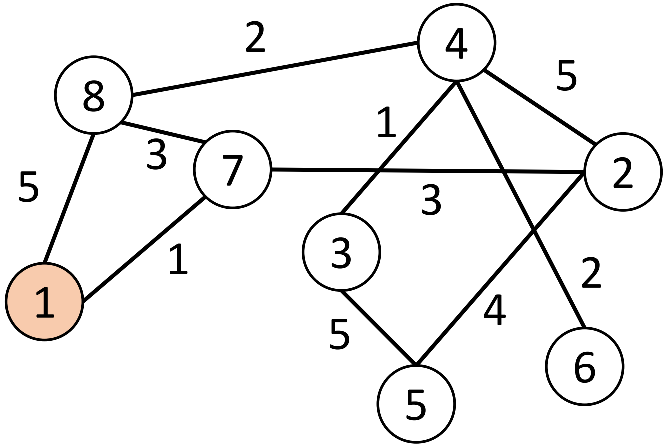
- **Prim's algorithm:**
  - Start with $V_T = v_1$ and $E_T = \emptyset$
  - While $V_T \neq V$:
    - Find the minimum weight edge $\{u, w\}$ going from a node in T to a node outside T,
    - Add $\{u, w\}$ to E_T,
    - Without loss of generality, $u \in T$. Then add w to V_T.



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

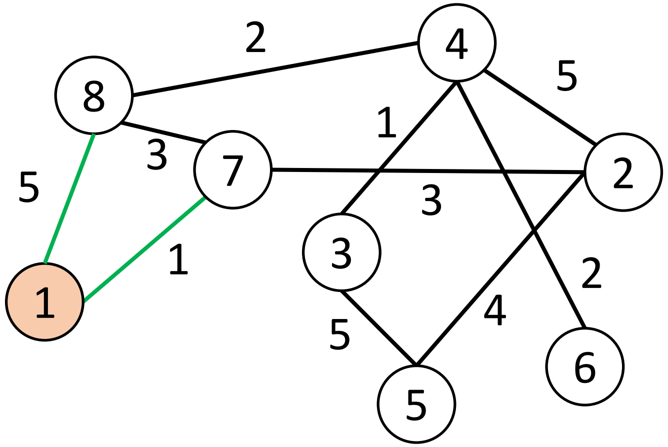
- **Prim's algorithm:**
  - Start with $V_T = v_1$ and $E_T = \emptyset$
  - While $V_T \neq V$:
    - Find the minimum weight edge $\{u, w\}$ going from a node in T to a node outside T,
    - Add $\{u, w\}$ to E_T,
    - Without loss of generality, $u \in T$. Then add w to V_T.



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

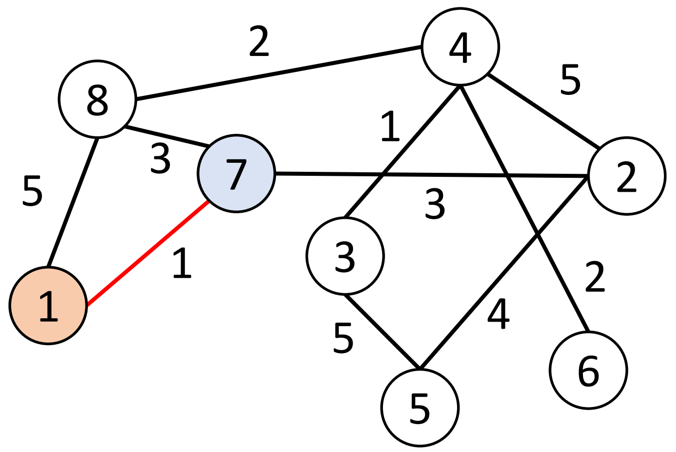
- **Prim's algorithm:**
  - Start with $V_T = v_1$ and $E_T = \emptyset$
  - While $V_T \neq V$:
    - Find the minimum weight edge $\{u, w\}$ going from a node in T to a node outside T,
    - Add $\{u, w\}$ to E_T,
    - Without loss of generality, $u \in T$. Then add w to V_T.



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

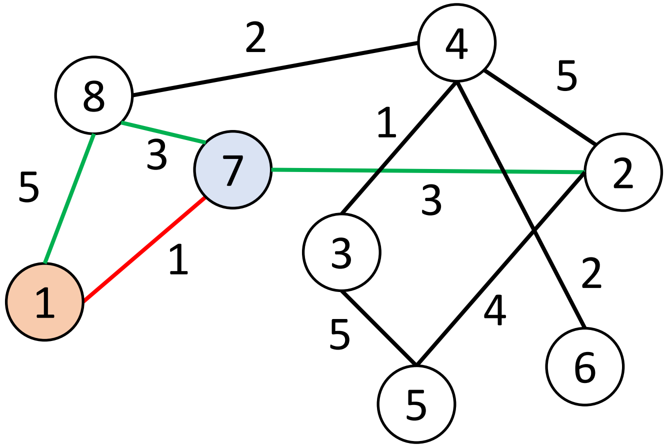
- **Prim's algorithm:**
  - Start with $V_T = v_1$ and $E_T = \emptyset$
  - While $V_T \neq V$:
    - Find the minimum weight edge $\{u, w\}$ going from a node in T to a node outside T,
    - Add $\{u, w\}$ to E_T,
    - Without loss of generality, $u \in T$. Then add w to V_T.



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:** Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

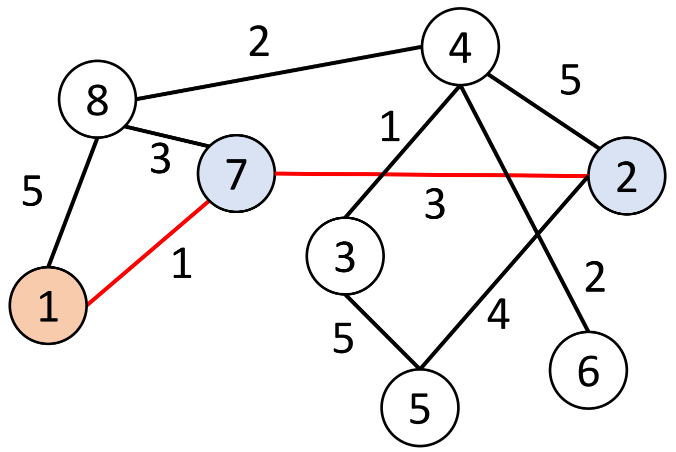
- **Prim's algorithm:**
  - Start with $V_T = v_1$ and $E_T = \emptyset$
  - While $V_T \neq V$:
    - Find the minimum weight edge $\{u, w\}$ going from a node in T to a node outside T,
    - Add $\{u, w\}$ to E_T,
    - Without loss of generality, $u \in T$. Then add w to V_T.



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

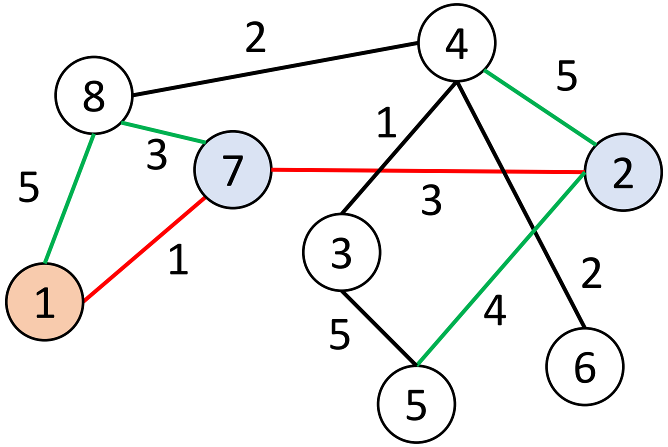
- **Prim's algorithm:**
  - Start with $V_T = v_1$ and $E_T = \emptyset$
  - While $V_T \neq V$:
    - Find the minimum weight edge $\{u, w\}$ going from a node in T to a node outside T,
    - Add $\{u, w\}$ to E_T,
    - Without loss of generality, $u \in T$. Then add w to V_T.



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

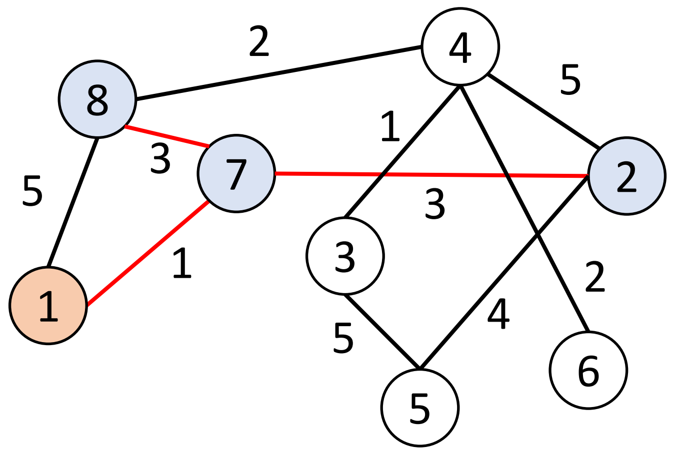
- **Prim's algorithm:**
  - Start with $V_T = v_1$ and $E_T = \emptyset$
  - While $V_T \neq V$:
    - Find the minimum weight edge $\{u, w\}$ going from a node in T to a node outside T,
    - Add $\{u, w\}$ to E_T,
    - Without loss of generality, $u \in T$. Then add w to V_T.



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

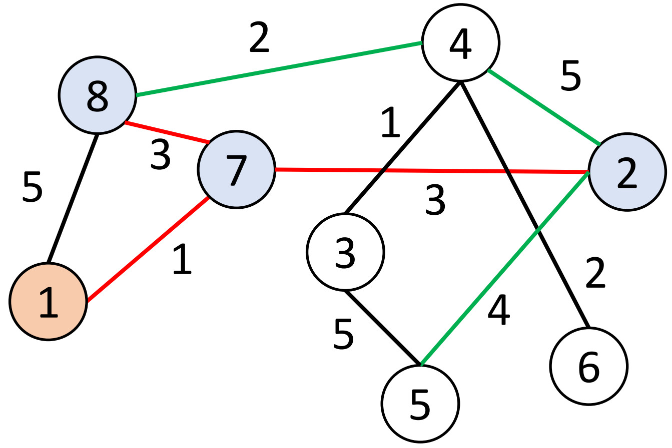
- **Prim's algorithm:**
  - Start with $V_T = v_1$ and $E_T = \emptyset$
  - While $V_T \neq V$:
    - Find the minimum weight edge $\{u, w\}$ going from a node in T to a node outside T,
    - Add $\{u, w\}$ to E_T,
    - Without loss of generality, $u \in T$. Then add w to V_T.



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

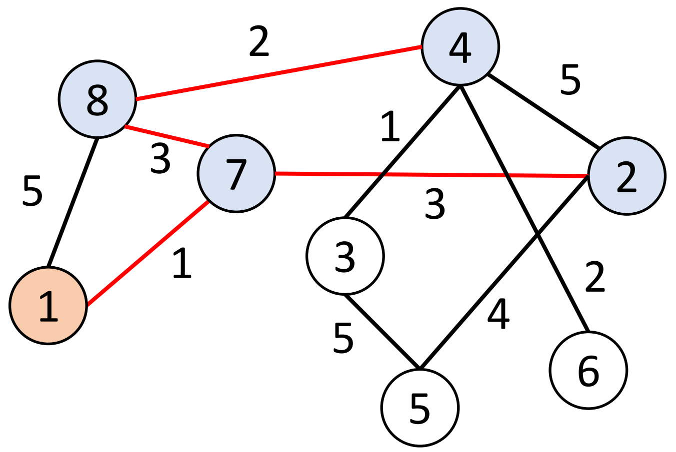
- **Prim's algorithm:**
  - Start with $V_T = v_1$ and $E_T = \emptyset$
  - While $V_T \neq V$:
    - Find the minimum weight edge $\{u, w\}$ going from a node in T to a node outside T,
    - Add $\{u, w\}$ to E_T,
    - Without loss of generality, $u \in T$. Then add w to V_T.



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

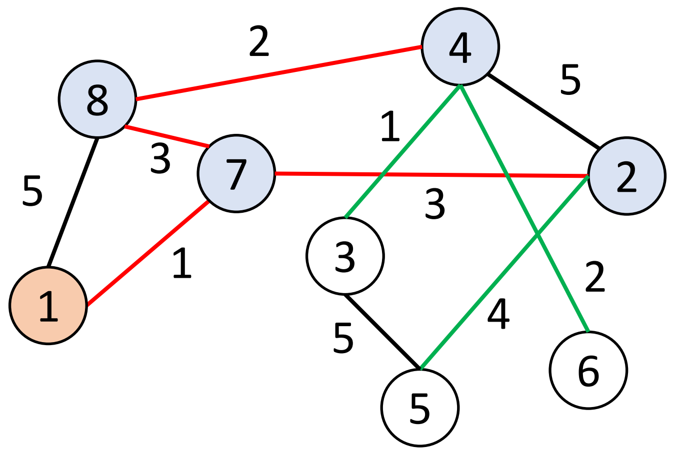
- **Prim's algorithm:**
  - Start with $V_T = v_1$ and $E_T = \emptyset$
  - While $V_T \neq V$:
    - Find the minimum weight edge $\{u, w\}$ going from a node in T to a node outside T,
    - Add $\{u, w\}$ to E_T,
    - Without loss of generality, $u \in T$. Then add w to V_T.



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

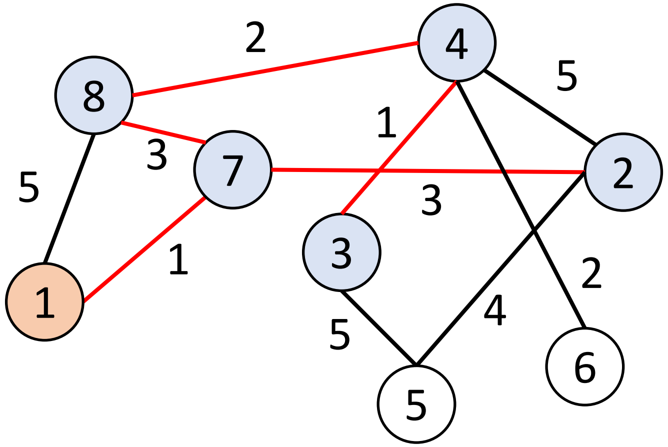
- **Prim's algorithm:**
  - Start with $V_T = v_1$ and $E_T = \emptyset$
  - While $V_T \neq V$:
    - Find the minimum weight edge $\{u, w\}$ going from a node in T to a node outside T,
    - Add $\{u, w\}$ to E_T,
    - Without loss of generality, $u \in T$. Then add w to V_T.



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

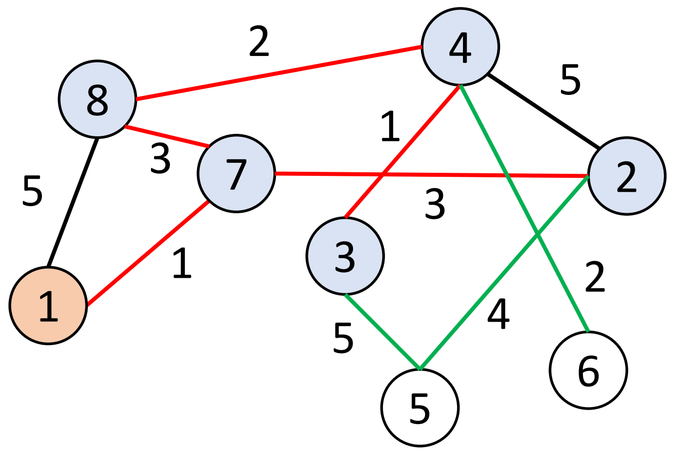
- **Prim's algorithm:**
  - Start with $V_T = v_1$ and $E_T = \emptyset$
  - While $V_T \neq V$:
    - Find the minimum weight edge $\{u, w\}$ going from a node in T to a node outside T,
    - Add $\{u, w\}$ to E_T,
    - Without loss of generality, $u \in T$. Then add w to V_T.



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

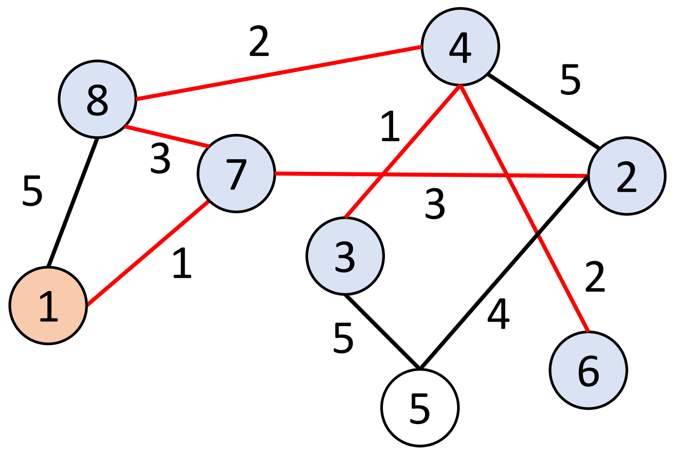
- **Prim's algorithm:**
  - Start with $V_T = v_1$ and $E_T = \emptyset$
  - While $V_T \neq V$:
    - Find the minimum weight edge $\{u, w\}$ going from a node in T to a node outside T,
    - Add $\{u, w\}$ to E_T,
    - Without loss of generality, $u \in T$. Then add w to V_T.



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

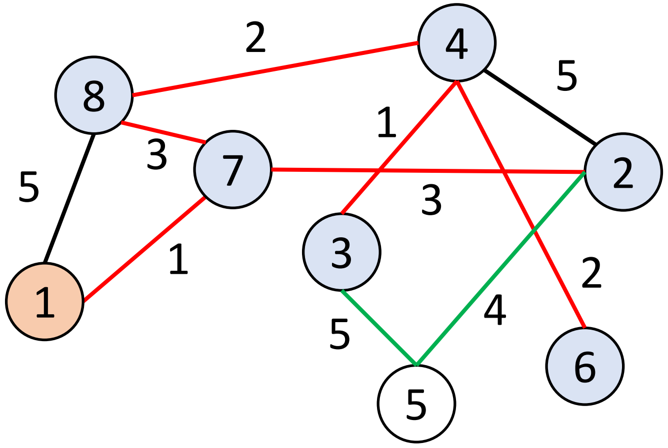
- **Prim's algorithm:**
  - Start with $V_T = v_1$ and $E_T = \emptyset$
  - While $V_T \neq V$:
    - Find the minimum weight edge $\{u, w\}$ going from a node in T to a node outside T,
    - Add $\{u, w\}$ to E_T,
    - Without loss of generality, $u \in T$. Then add w to V_T.



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:** Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

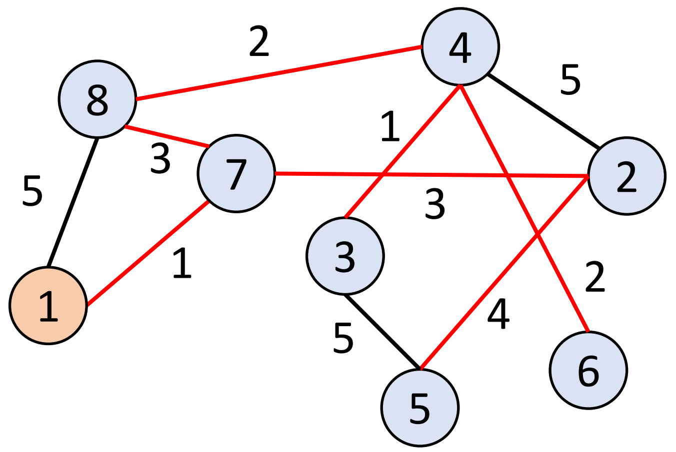
- **Prim's algorithm:**
  - Start with $V_T = v_1$ and $E_T = \emptyset$
  - While $V_T \neq V$:
    - Find the minimum weight edge $\{u, w\}$ going from a node in T to a node outside T,
    - Add $\{u, w\}$ to E_T,
    - Without loss of generality, $u \in T$. Then add w to V_T.



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

- **Prim's algorithm:**
  - Start with $V_T = v_1$ and $E_T = \emptyset$
  - While $V_T \neq V$:
    - Find the minimum weight edge $\{u, w\}$ going from a node in T to a node outside T,
    - Add $\{u, w\}$ to E_T,
    - Without loss of generality, $u \in T$. Then add w to V_T.



Greedy algorithms for minimum-weight spanning trees (MST)

- **Minimum-weight spanning tree (MST) problem:**
  Given an undirected weighted graph $G = (V, E)$, compute the spanning tree with minimum total edge weights.

- **Prim's algorithm:**
  - Best implementation gives $O(|E| + |V| \log|V|)$ worst-case time
  - Easier implementations give $O(|V|^2)$ or $O(|E| \log|V|)$ worst-case time

  - Output is a spanning tree is trivial to show (spanning + no cycles)
  - Minimum-weight also a bit tricky to show (just as for Kruskal's algorithm).

Summary

Today's lecture:

Introduced:

- Greedy algorithms, with multiple examples
- Algorithms for the Minimum-Weight Spanning Tree problem:
  - Kruskal's algorithm
  - Prim's algorithm

- **Next Lecture:** Dynamic programming
- **Any questions?**