CS-3510: Design and Analysis of Algorithms

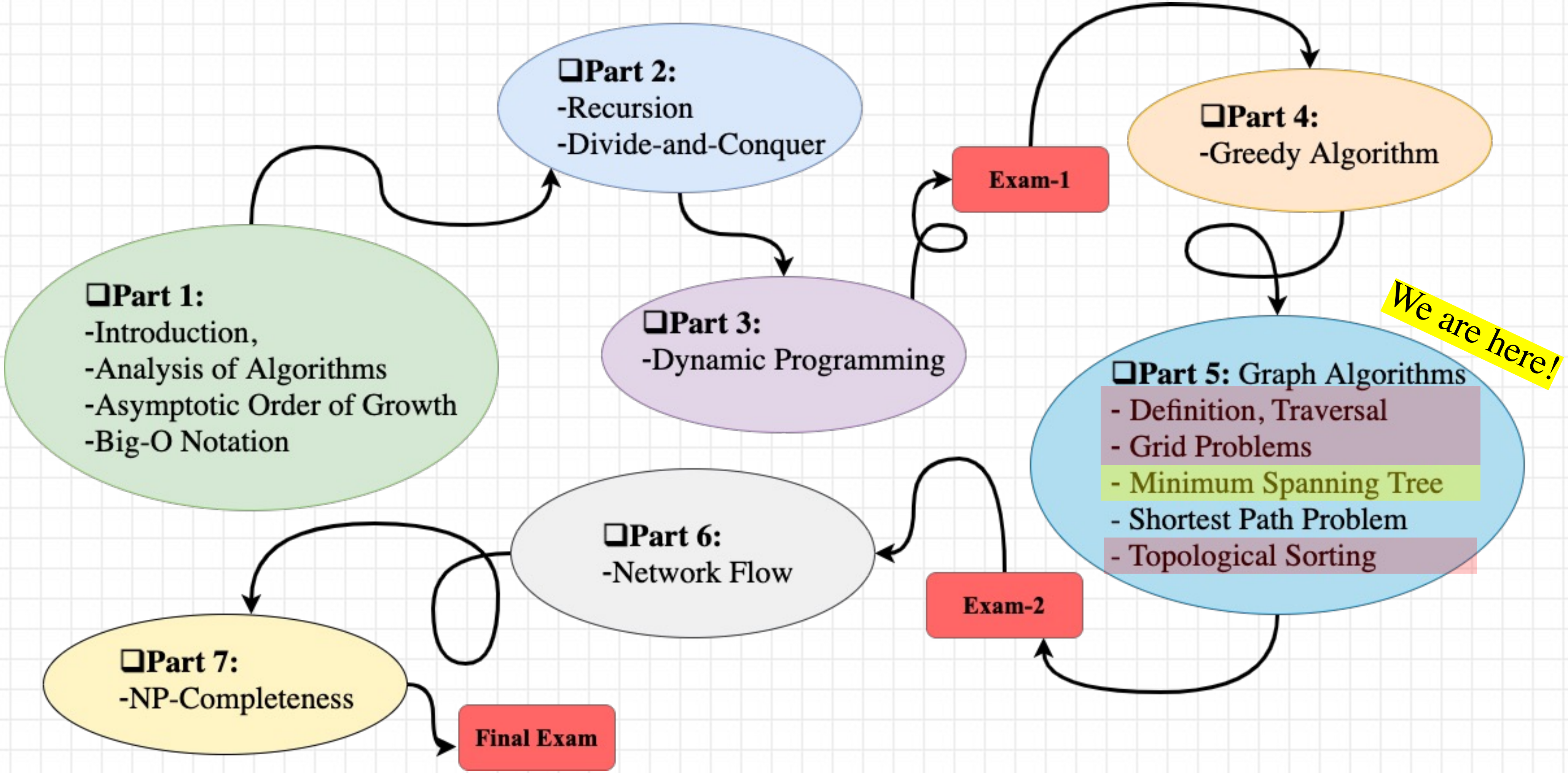
Graph Algorithms: Minimum Spanning Tree

Instructor: Shahrokh Shahi

College of Computing
Georgia Institute of Technology

Summer 2022

Roadmap


❑Part 1:
-Introduction,
-Analysis of Algorithms
-Asymptotic Order of Growth
-Big-O Notation
❑Part 2:
-Recursion
-Divide-and-Conquer
❑Part 3:
-Dynamic Programming
Exam-1
❑Part 4:
-Greedy Algorithm
We are here!
❑Part 5: Graph Algorithms
- Definition, Traversal
- Grid Problems
- Minimum Spanning Tree
- Shortest Path Problem
- Topological Sorting
Exam-2
❑Part 6:
-Network Flow
❑Part 7:
-NP-Completeness
Final Exam

Graph

- Graph definition and representation
  - Adjacency matrix
  - Adjacency list
- Graph traversal
  - Breadth first search (BFS)
    - Shortest path (<u>unweighted</u> graphs)
    - Testing bipartiteness
    - Tree traversal (level-order)
    - Connected components
  - Depth first search (DFS)
    - Topological sorting
    - Tree traversal (in-order, pre-order, post-order)
    - Connected components
- Graph problems/algorithms
  - Minimum spanning tree (MST)
    - Kruskal (greedy)
    - Prim (greedy)
  - Shortest path (directed <u>weighted</u> graphs)
    - Dijkstra (greedy)
    - Bellman-Ford (dynamic programming)
    - Floyd-Warshall (dynamic programming)
  - Flow network
    - Max-flow min-cut theorem
    - Ford-Fulkerson algorithm

Graph

- Graph definition and representation
  - Adjacency matrix
  - Adjacency list
- Graph traversal
  - Breadth first search (BFS)
    - Shortest path (unweighted graphs)
    - Testing bipartiteness
    - Tree traversal (level-order)
    - Connected components
  - Depth first search (DFS)
    - Topological sorting
    - Tree traversal (in-order, pre-order, post-order)
    - Connected components

- Graph problems/algorithms
  - Minimum spanning tree (MST)
    - Kruskal (greedy)
    - Prim (greedy)
  - Shortest path (directed weighted graphs)
    - Dijkstra (greedy)
    - Bellman-Ford (dynamic programming)
    - Floyd-Warshall (dynamic programming)
  - Flow network
    - Max-flow min-cut theorem
    - Ford-Fulkerson algorithm

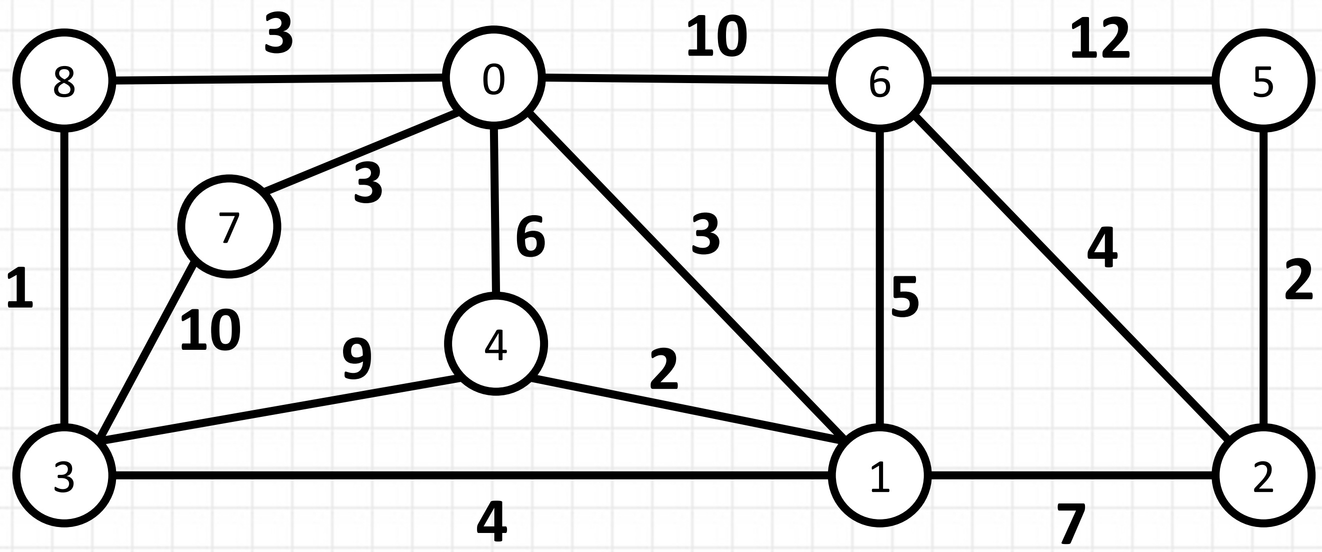
Minimum Spanning Tree

- Weighted graphs
  - Each edge has an associated weight, cost, or distance.
  - Edge (u, v) → w(u, v)

- Spanning tree
  - Given graph G = (V, E), a tree $T = (V, E_T)$ such that $E_T \subseteq E$ is a spanning tree of G.
  - Tree T spans the graph G

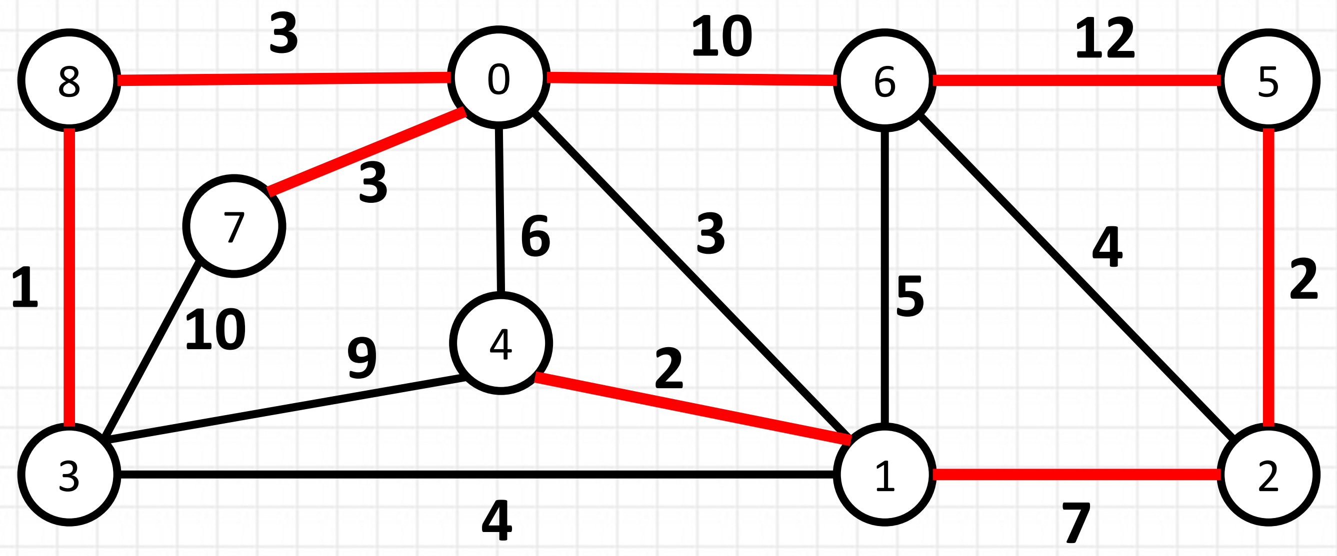
Minimum Spanning Tree

- Spanning tree
  - Given graph G = (V, E), a tree $T = (V, E_T)$ such that $E_T \subseteq E$ is a spanning tree of G.
  - Tree T spans the graph G



Minimum Spanning Tree

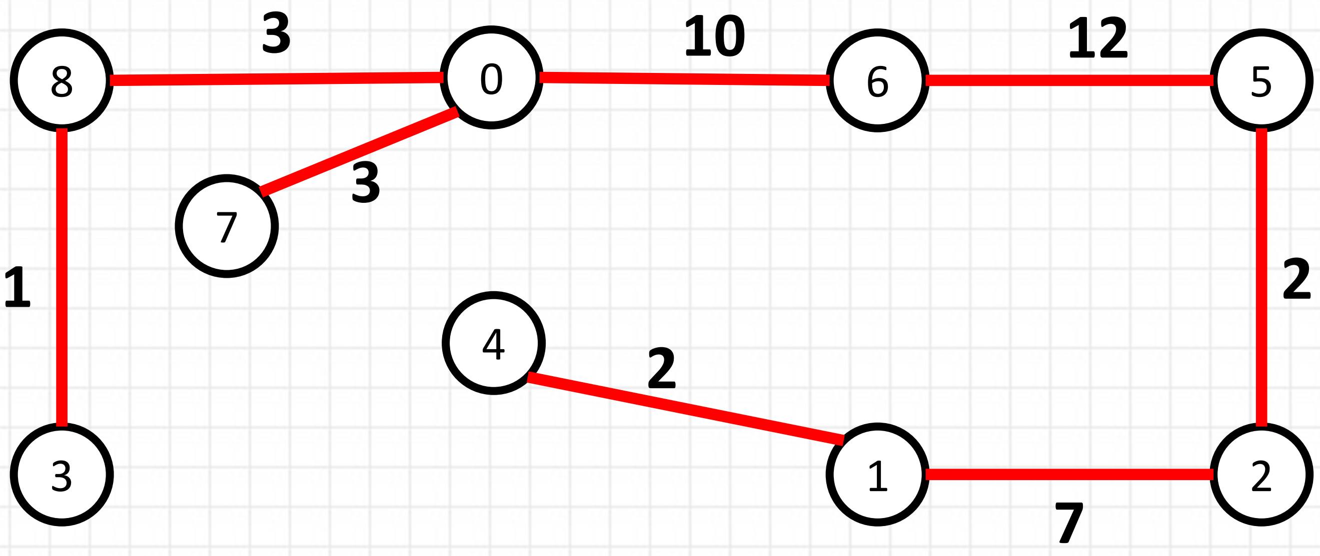
- Spanning tree
  - Given graph G = (V, E), a tree $T = (V, E_T)$ such that $E_T \subseteq E$ is a spanning tree of G.
  - Tree T spans the graph G



Minimum Spanning Tree

- Spanning tree
  - Given graph G = (V, E), a tree $T = (V, E_T)$ such that $E_T \subseteq E$ is a spanning tree of G.
  - Tree T spans the graph G

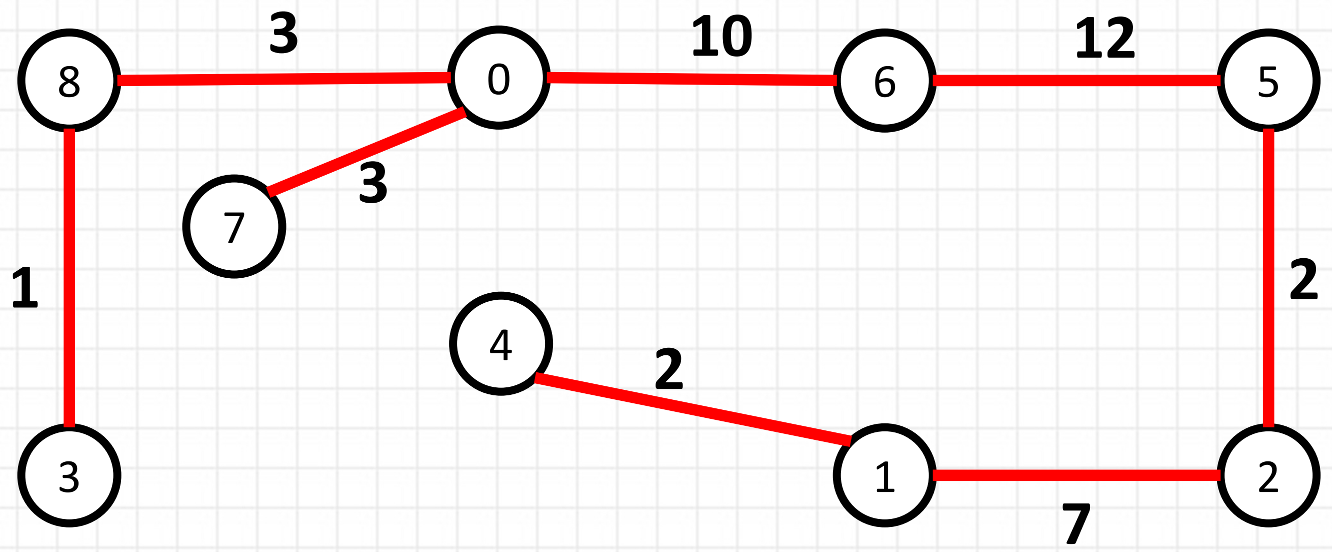
Spanning tree:



Minimum Spanning Tree

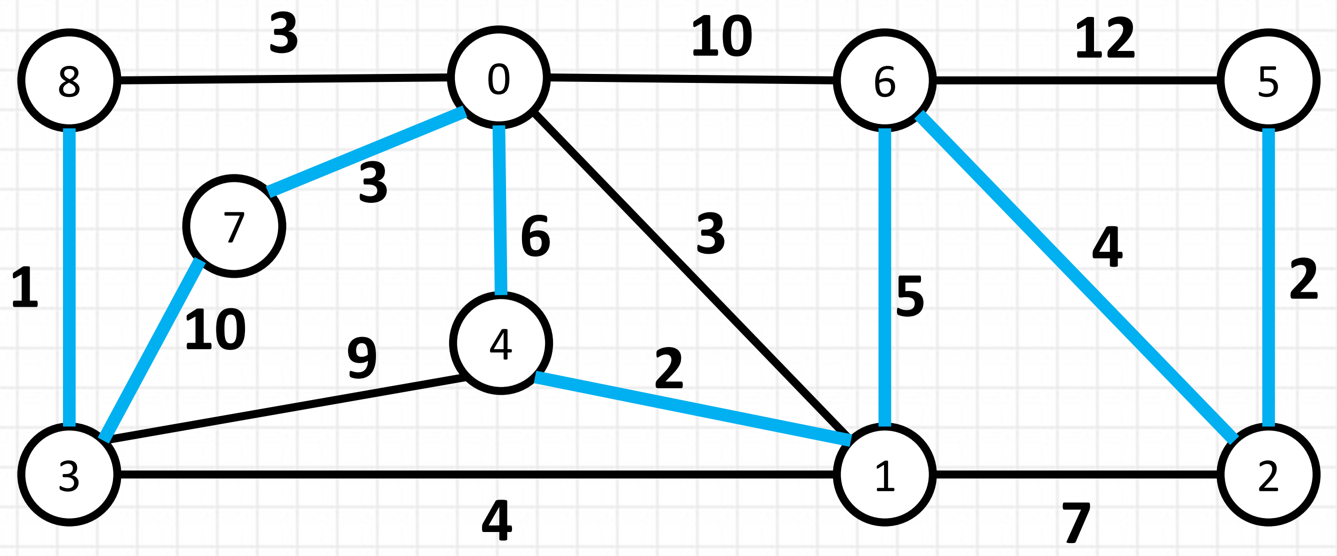
- Spanning tree
  - Given graph G = (V, E), a tree $T = (V, E_T)$ such that $E_T \subseteq E$ is a spanning tree of G.
  - Tree T spans the graph G

Total weight = 1 + 3 + 3 + 10 + 12 + 2 + 7 + 2 = 40



Minimum Spanning Tree

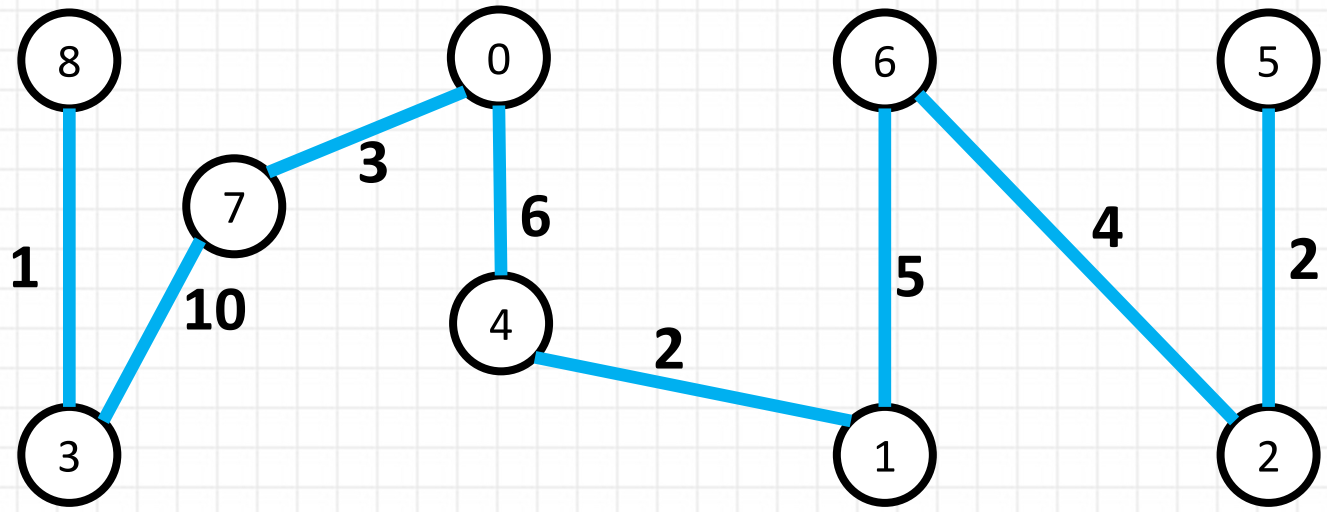
- Spanning tree
  - Given graph G = (V, E), a tree $T = (V, E_T)$ such that $E_T \subseteq E$ is a spanning tree of G.
  - Tree T spans the graph G



Minimum Spanning Tree

- Spanning tree
  - Given graph G = (V, E), a tree $T = (V, E_T)$ such that $E_T \subseteq E$ is a spanning tree of G.
  - Tree T spans the graph G

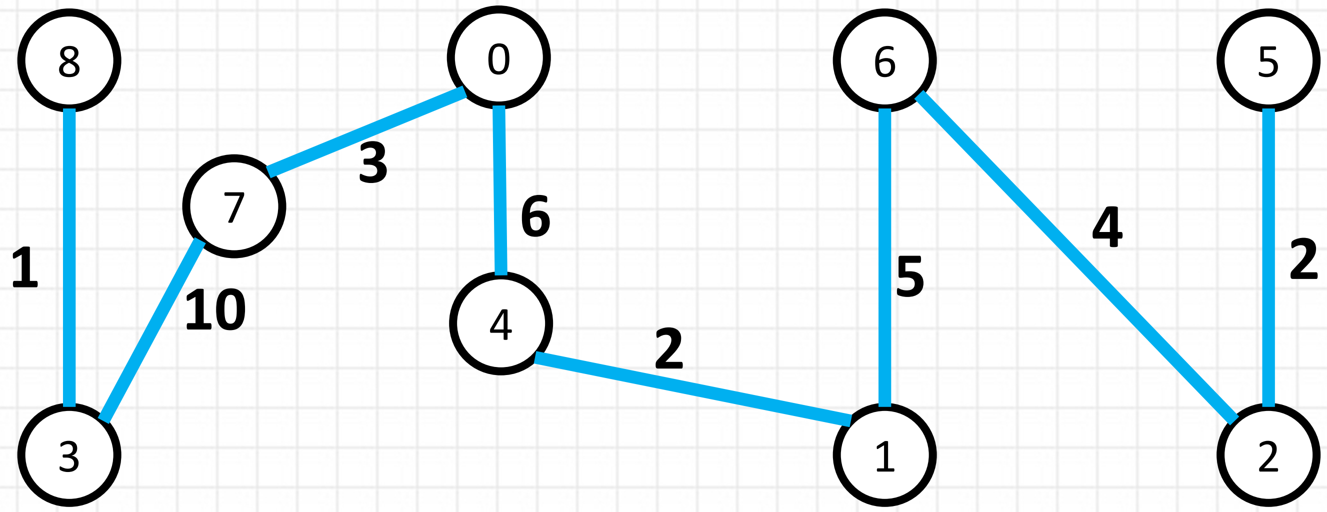
Another spanning tree:



Minimum Spanning Tree

- Spanning tree
  - Given graph G = (V, E), a tree $T = (V, E_T)$ such that $E_T \subseteq E$ is a spanning tree of G.
  - Tree T spans the graph G

Total weight = 1 + 10 + 3 + 6 + 2 + 5 + 4 + 2 = 33



Minimum Spanning Tree (MST)

- Weighted graphs
  - Each edge has an associated weight, cost, or distance.
  - Edge (u, v) → w(u, v)

- Spanning tree
  - Given graph G = (V, E), a tree $T = (V, E_T)$ such that $E_T \subseteq E$ is a spanning tree of G.
  - Tree T spans the graph G
- Minimum spanning tree = Minimum-weight spanning tree
- Spanning tree T for G such that the sum $w(T) = \sum_{(u,v) \in T} w(u, v)$ is minimized

Minimum Spanning Tree (MST)

- Spanning tree
  - Given graph G = (V, E), a tree $T = (V, E_T)$ such that $E_T \subseteq E$ is a spanning tree of G.
  - Tree T spans the graph G
- Minimum spanning tree = Minimum-weight spanning tree
- Spanning tree T for G such that the sum $w(T) = \sum_{(u,v) \in T} w(u, v)$ is minimized
- Approach: "Greedy choice"
- Algorithms:
  - Kruskal
  - Prim

Growing a Minimum Spanning Tree

- This greedy strategy is captured by the following generic method, which grows the minimum spanning tree one edge at a time.
- The generic method manages a set of edges A, maintaining the following loop invariant:
  - **Prior to each iteration, A is a subset of some minimum spanning tree**.
- At each step, we determine an edge (u, v) that we can add to A without violating this invariant $A \cup \{(u, v)\}$ is also a subset of an MST
- An edge is safe edge if adding it to A will not violate the invariant.

Growing a Minimum Spanning Tree

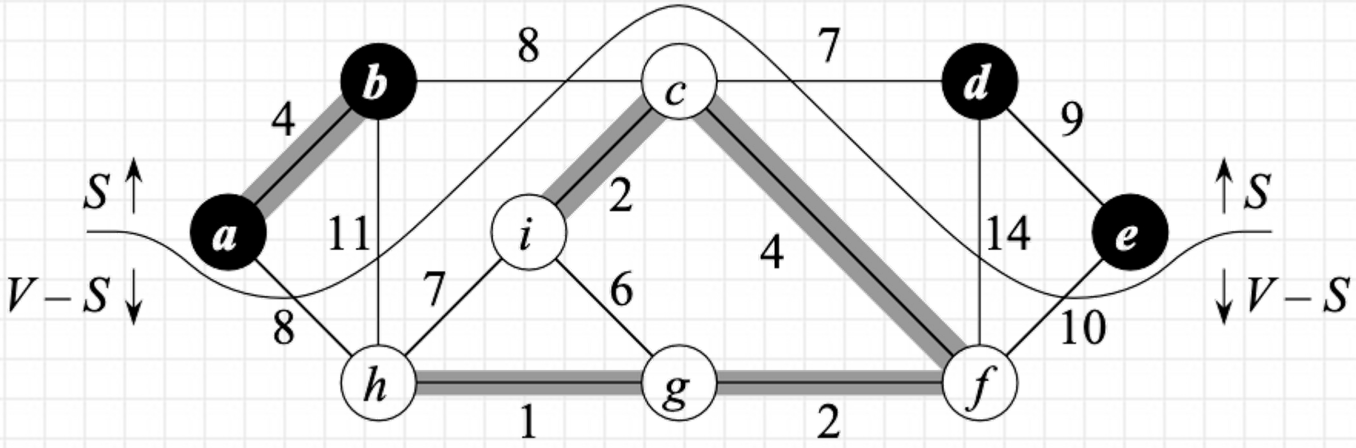
- This greedy strategy is captured by the following generic method, which grows the minimum spanning tree one edge at a time.

```
GENERIC-MST(G, w)
A = ∅
while A does not form a spanning tree
    find an edge (u, v) that is safe for A
    A = A ∪ {(u, v)}
return A
```

- Tricky part? Finding a safe edge at each iteration!

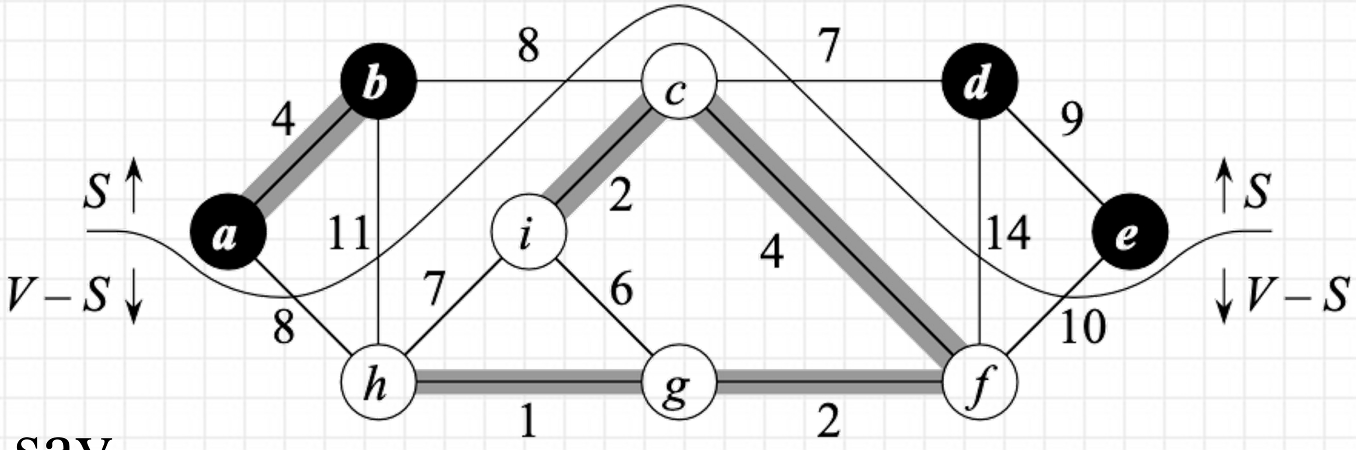
Some Definitions

- Cut
  - A cut $(S, V - S)$ of an undirected graph $G = (V, E)$ is a partition of V .



Some Definitions

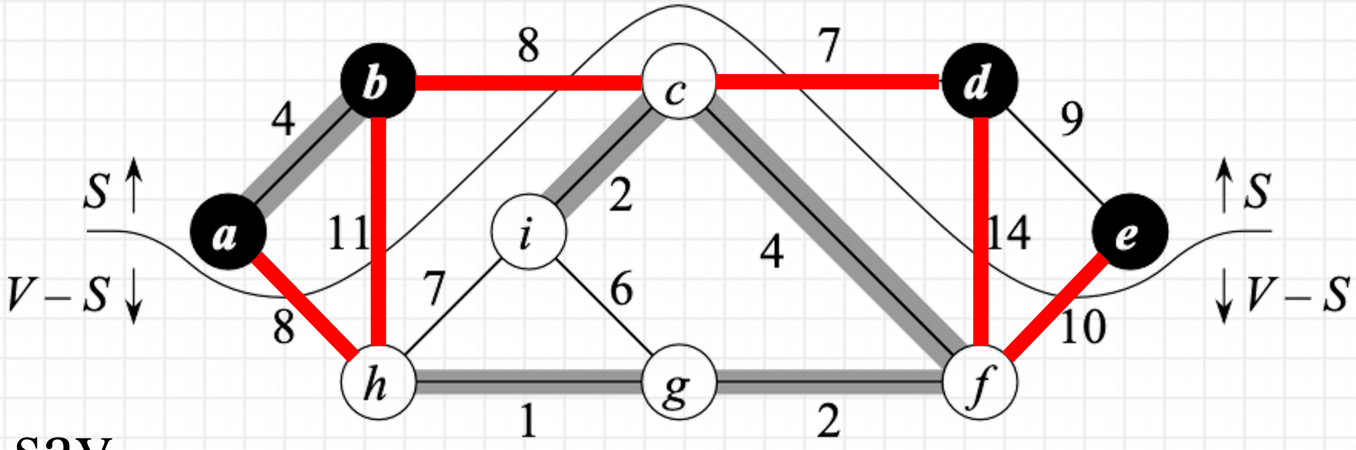
- Cut
  - A cut $(S, V - S)$ of an undirected graph $G = (V, E)$ is a partition of V .



- With this definition, we say
  - An edge $(u, v) \in E$ crosses the cut $(S, V - S)$ if one of this endpoints is in S, and the other in $V - S$

Some Definitions

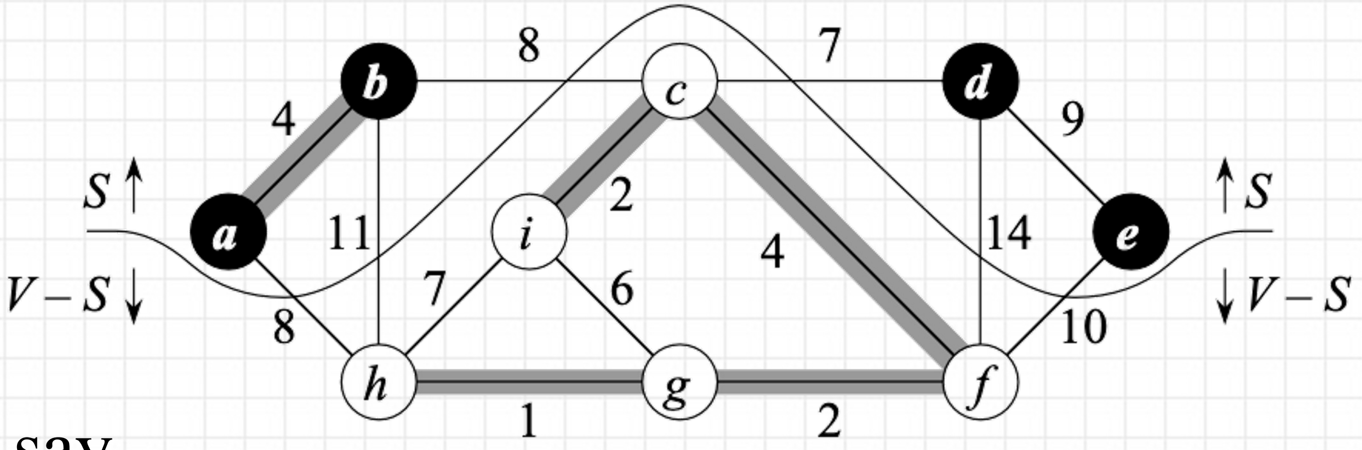
- Cut
  - A cut $(S, V - S)$ of an undirected graph $G = (V, E)$ is a partition of V .



- With this definition, we say
  - An edge $(u, v) \in E$ crosses the cut $(S, V - S)$ if one of this endpoints is in S, and the other in $V - S$

Some Definitions

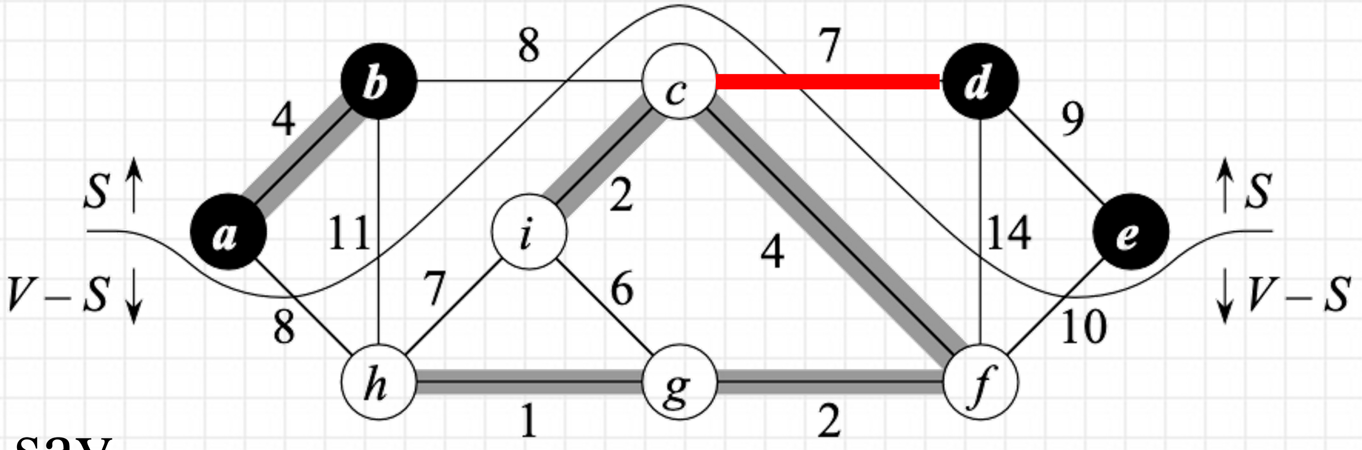
- Cut
  - A cut $(S, V - S)$ of an undirected graph $G = (V, E)$ is a partition of V .



- With this definition, we say
  - A cut respects a set A of edges if no edge in A crosses the cut.

Some Definitions

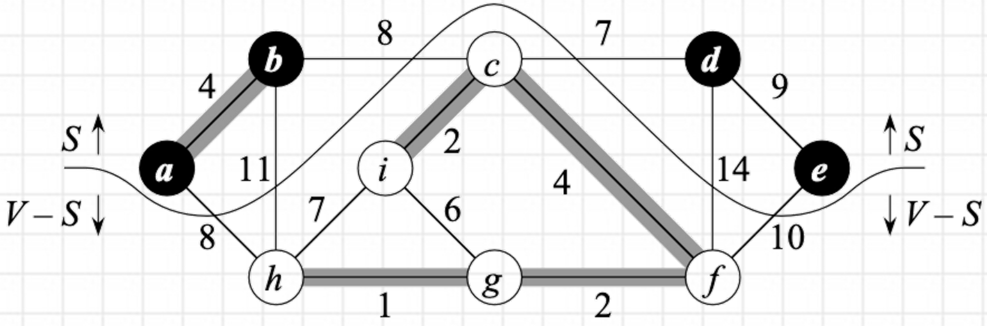
- Cut
  - A cut $(S, V - S)$ of an undirected graph $G = (V, E)$ is a partition of V .



- With this definition, we say
  - An edge is a light edge crossing a cut if its weight is the minimum of any edge crossing the cut.
  - Note that there can be more than one light edge crossing a cut in the case of ties.

Some Definitions

- Cut
  - A cut $(S, V - S)$ of an undirected graph $G = (V, E)$ is a partition of V .



- With this definition, we say
  - An edge $(u, v) \in E$ crosses the cut $(S, V - S)$ if one of this endpoints is in S, and the other in $V - S$
  - A cut respects a set A of edges if no edge in A crosses the cut.
  - An edge is a light edge crossing a cut if its weight is the minimum of any edge crossing the cut.

Generic-MST

- Theorem:

Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, let $(S, V - S)$ be any cut of G that respects A, and let (u, v) be a light edge crossing this cut. Then edge (u, v) is safe for A.

Generic-MST

- Theorem:

Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, let $(S, V - S)$ be any cut of G that respects A, and let (u, v) be a light edge crossing this cut. Then edge (u, v) is safe for A.

```
GENERIC-MST(G, w)
A = ∅
while A does not form a spanning tree
    find an edge (u, v) that is safe for A
    A = A ∪ {(u, v)}
return A
```

Generic-MST

- Notes
  - The set A is always acyclic.
  - At any point $G_A = (V, A)$ is a forest
  - At first when $A = \phi$, we have |V| trees in the forest G_A, each a tree of one vertices
  - At each iteration, the number of trees is reduced by one.
  - While loop (line 2-4) runs for |V|-1 times to find the edges required to form the minimum spanning tree.
  - The method terminates when we have one tree (clearly, with |V|-1 edges).

GENERIC-MST(G, w)

1 $A = \emptyset$
2 **while** A does not form a spanning tree
3   find an edge (u, v) that is safe for A
4     $A = A \cup \{(u, v)\}$
5 **return** A

Generic-MST

- Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G,
- [**Theorem**:] let $(S, V - S)$ be any cut of G that respects A, and let (u, v) be a light edge crossing this cut. Then edge (u, v) is safe for A.
- [**Corollary**:] let $C = (V_C, E_C)$ be a connected component (tree) in the forest $G_A = (V, A)$. If (u, v) is a light edge connecting C to some other component in G_A, Then edge (u, v) is safe for A.
  - Pf. Cut $(V_C, V - V_C)$ respects A, and (u, v) is a light edge for this cut → safe

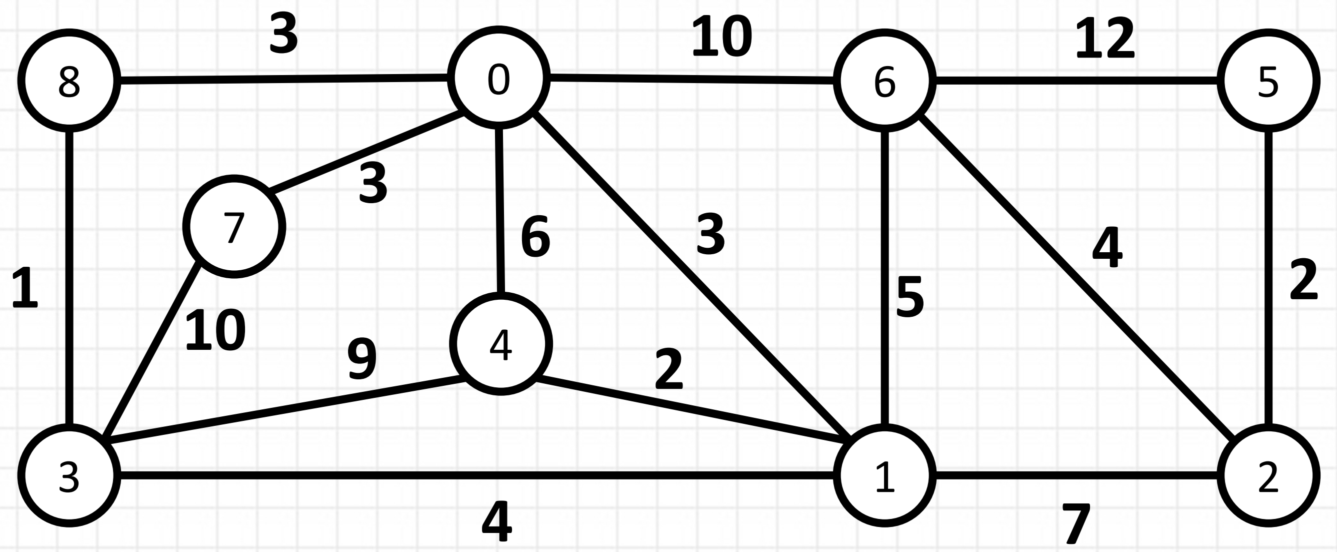
MST Algorithms

- Kruskal's algorithm
  - The set A is a forest whose vertices are all those of the given graph.
  - The safe edge added to A is always a least-weight edge in the graph that connects two distinct components. (<u>so it is not creating a loop</u>)

- Prim's algorithm
  - The set A forms a single tree.
  - The safe edge added to A is always a least-weight edge connecting the tree to a vertex not in the tree.

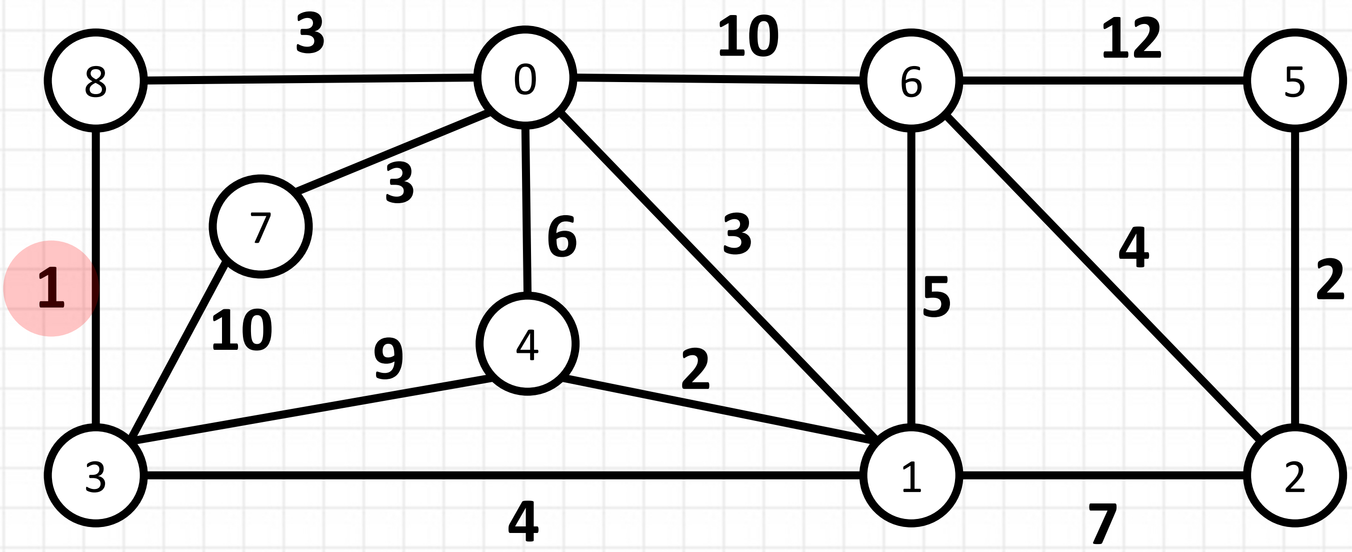
Kruskal’s Algorithm

- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.



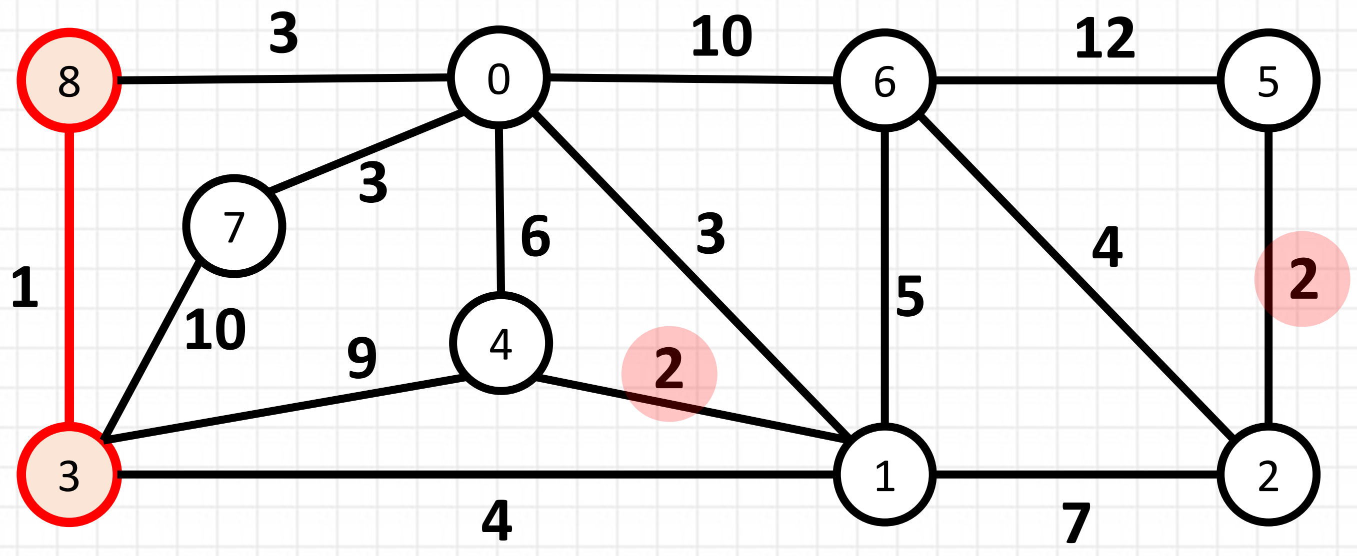
Kruskal's Algorithm

- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.



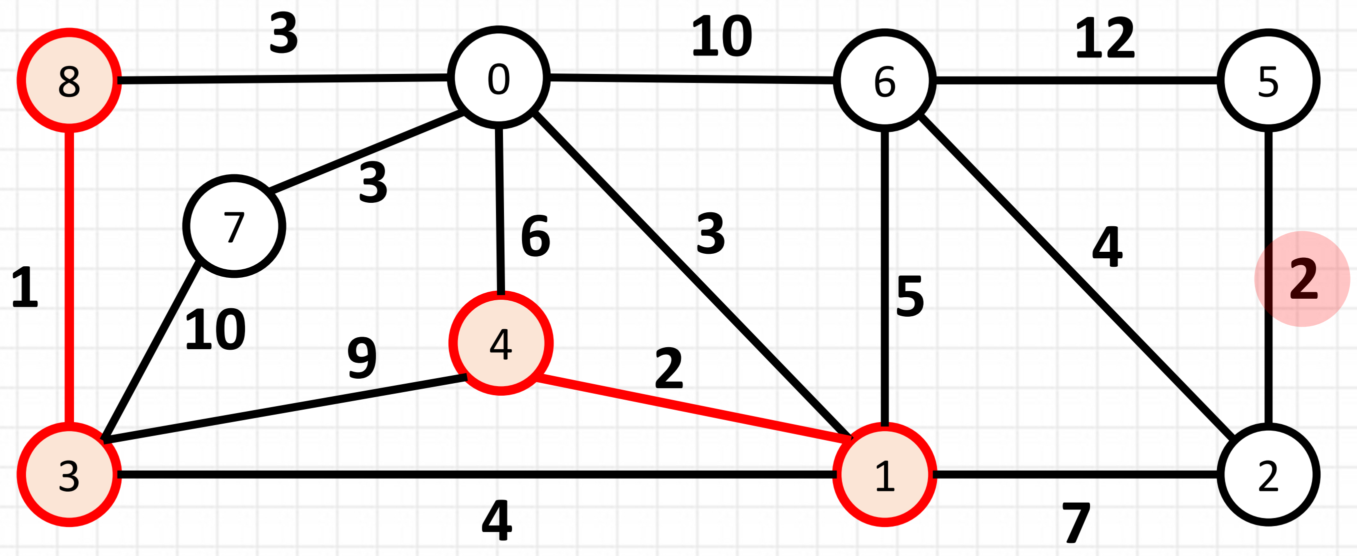
Kruskal's Algorithm

- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.



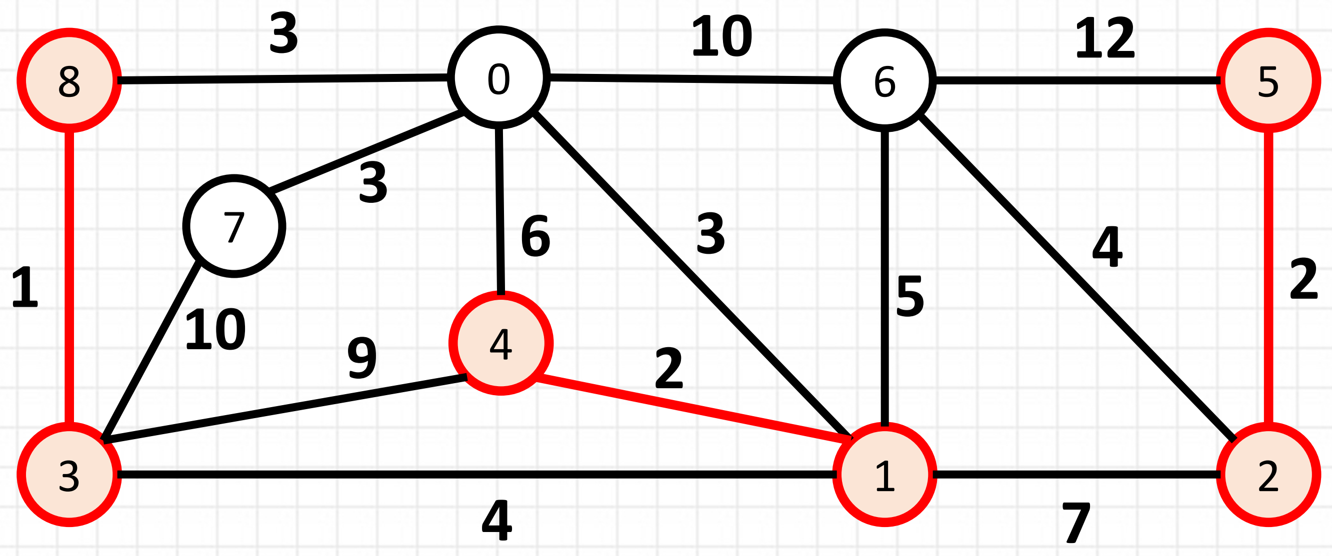
Kruskal's Algorithm

- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.



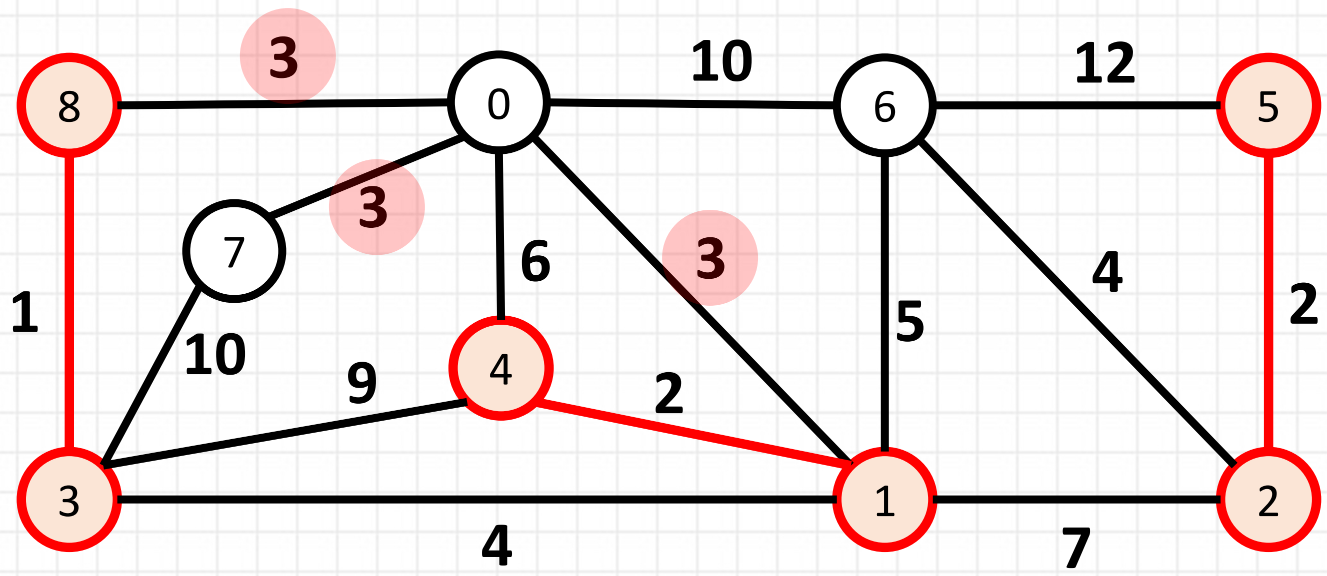
Kruskal's Algorithm

- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.



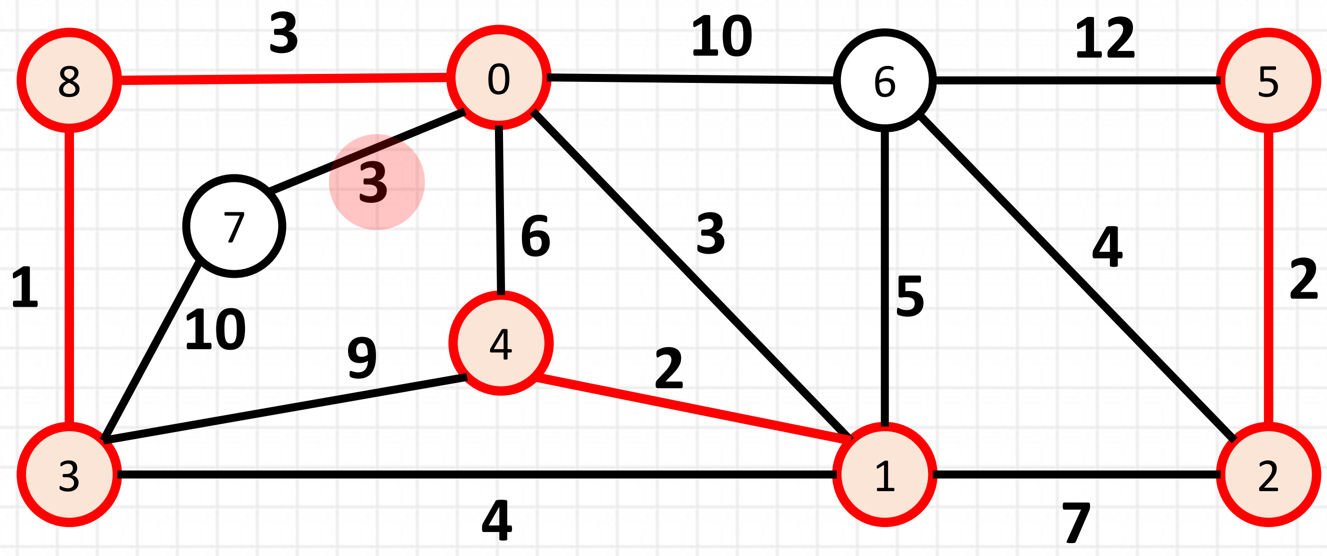
Kruskal's Algorithm

- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.



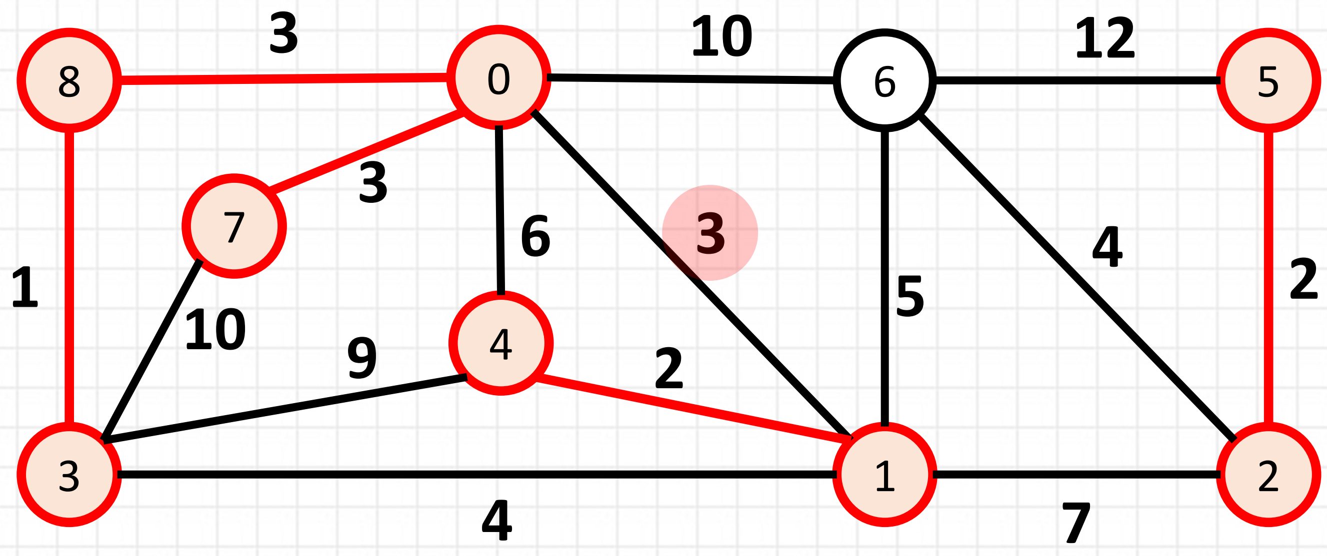
Kruskal’s Algorithm

- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.



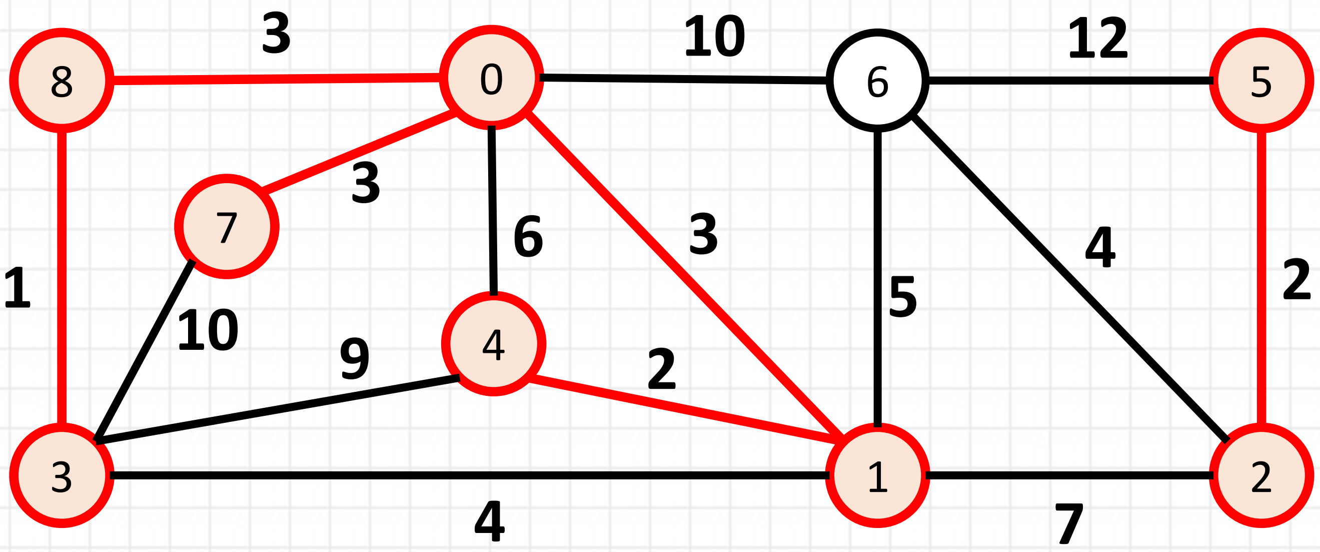
Kruskal's Algorithm

- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.



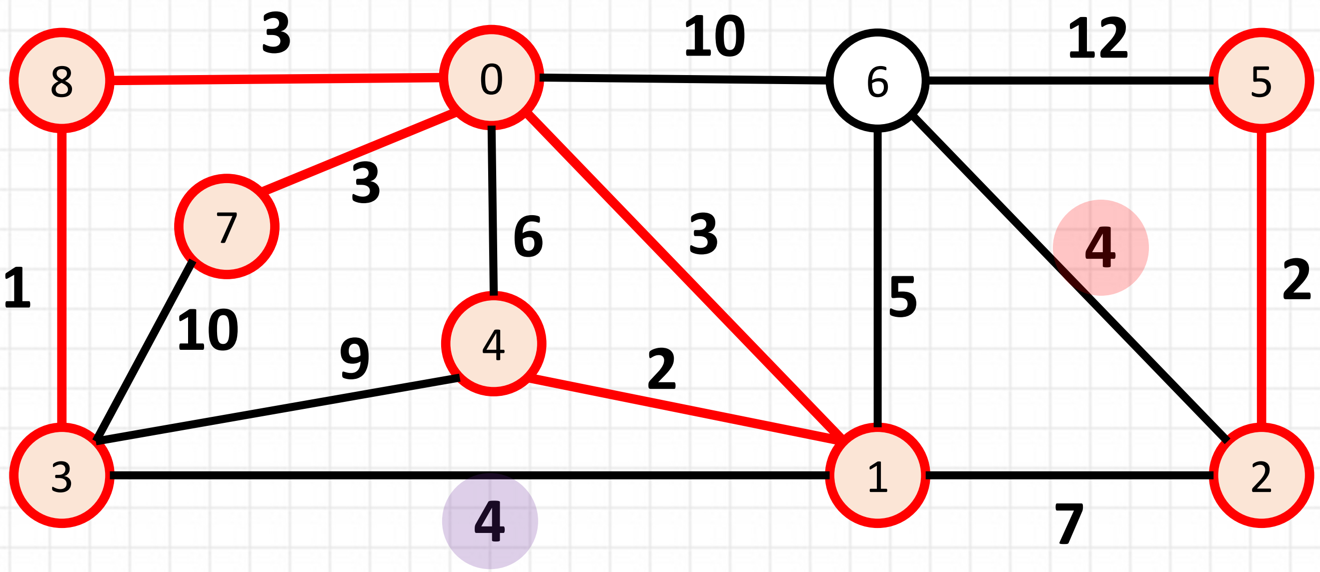
Kruskal's Algorithm

- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.



Kruskal's Algorithm

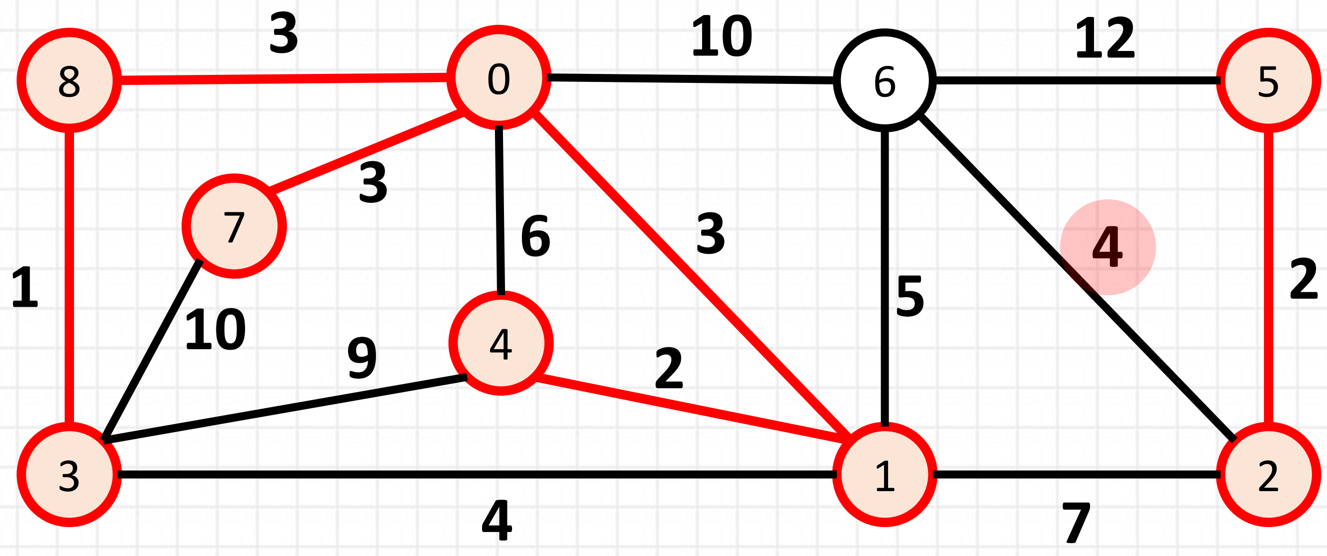
- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.



Cannot add this one (not two distinct components!)

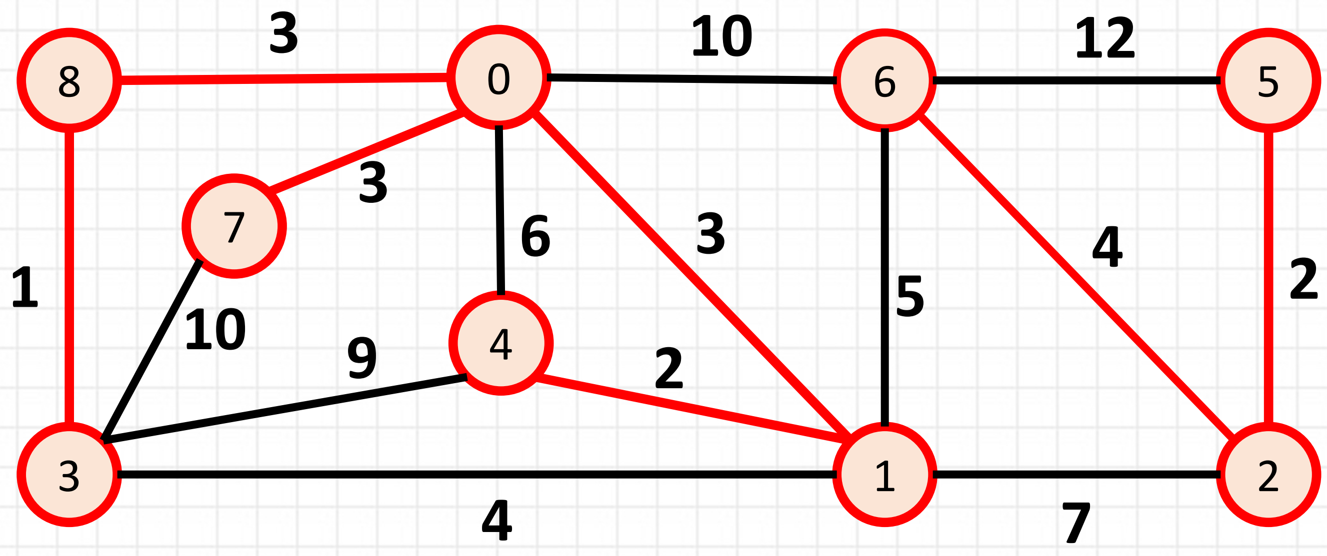
Kruskal's Algorithm

- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.



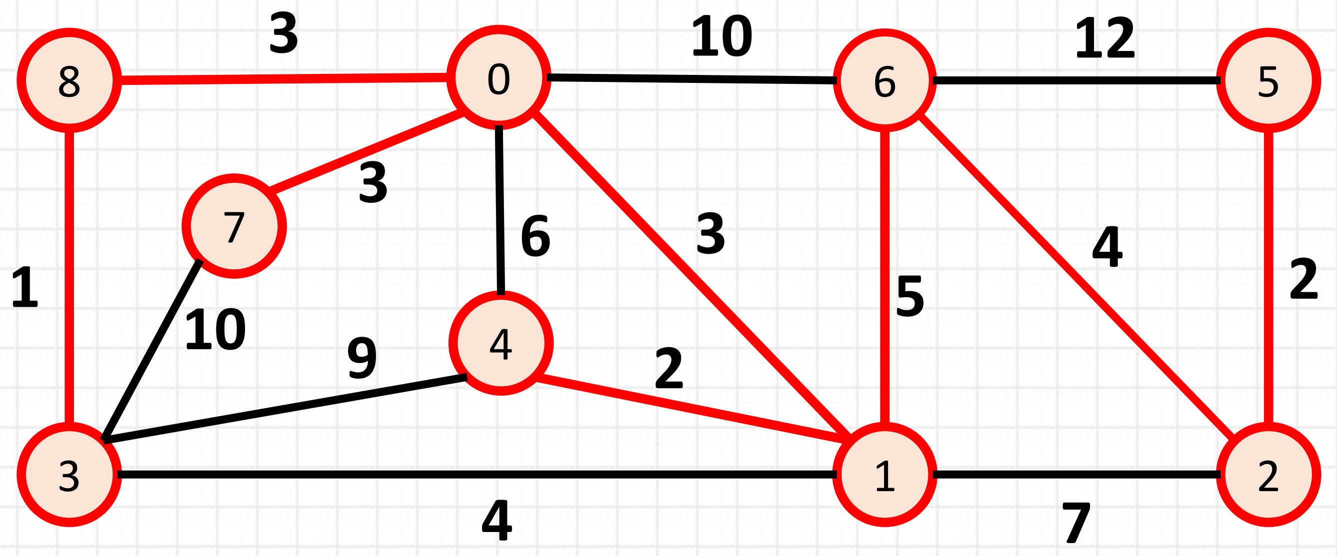
Kruskal's Algorithm

- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.



Kruskal's Algorithm

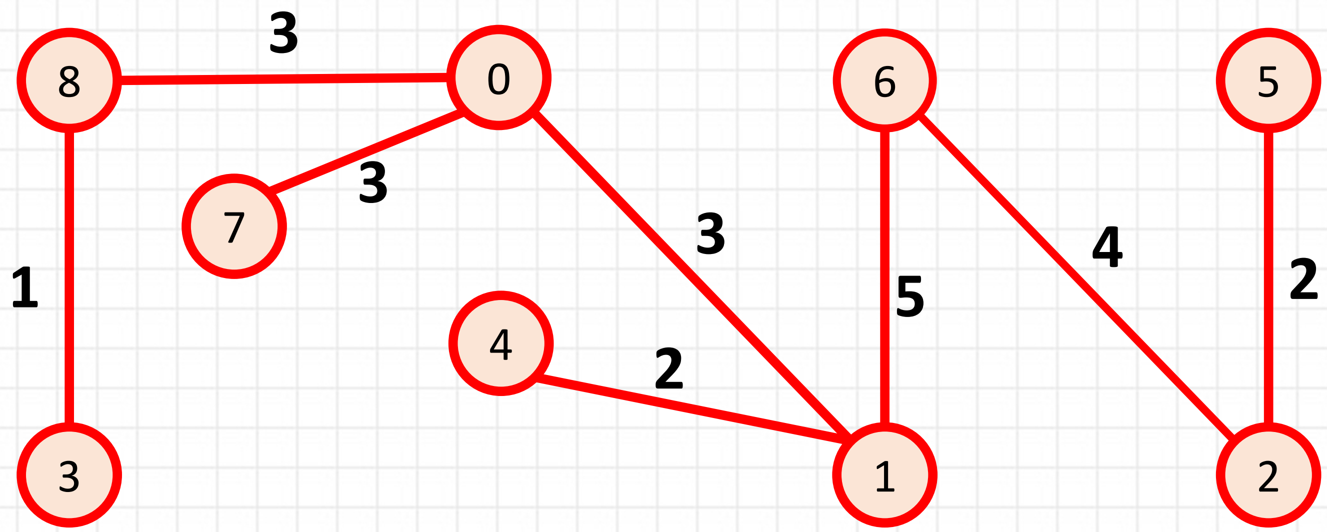
- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.



Kruskal's Algorithm

- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.

MST Weight = 1 + 2 + 2 + 3 + 3 + 3 + 4 + 5 = 23



Kruskal's Algorithm

- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.

Greedy choice

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

Uses "disjoint-set" (also known as "union-find") data structure

Kruskal's Algorithm

- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.

Greedy choice

```
MST-KRUSKAL(G, w)
A = Ø
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

Creating one disjoint-set per each graph vertex

Uses "disjoint-set" (also known as "union-find") data structure

Kruskal's Algorithm

- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.

Greedy choice

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

To find the light weight at each step

Uses “disjoint-set” (also known as “union-find”) data structure

Kruskal's Algorithm

- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.

Greedy choice

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

For the current min-weight (light weight) edge

Uses "disjoint-set" (also known as "union-find") data structure

Kruskal's Algorithm

- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.

Greedy choice

MST-KRUSKAL(G, w)

```
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

If u and v belongs to different trees (disjoint sets), then add (u,v) to the growing MST and merge the two tress

Uses "disjoint-set" (also known as "union-find") data structure

Kruskal's Algorithm

- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.

Greedy choice

MST-KRUSKAL(G, w)

```
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

If u and v belongs to different trees (disjoint sets), then add (u,v) to the growing MST and merge the two tress

Uses "disjoint-set" (also known as "union-find") data structure

Kruskal's Algorithm

- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.

Greedy choice

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

If u and v belongs to different trees (disjoint sets), then add (u,v) to the growing MST and merge the two tress

Uses "disjoint-set" (also known as "union-find") data structure

Kruskal's Algorithm

- The set A is a forest whose vertices are all those of the given graph.
- The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.

Greedy choice

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

Running time?

Kruskal's Algorithm

- Running time
  - Depends on disjoint-set implementation
    - Most efficient:
      union-by-rank with path compression
    - CLRS 21
  - Make-Set O(|V|)
  - Sorting edges O(|E| log$|E|$)
  - For loop (lines 5-8)
    - Find-Set and Union O(|E|)
    - $O((|V| + |E|)\alpha(|V|))$
    - Assume G is connected: |E| $\geq$ |V|-1
    - $O\big((|V| + |E|)\alpha(|V|)\big) \rightarrow O\big(|E|\alpha(|V|)\big)$

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

Running time?

Kruskal's Algorithm

- Running time
  - Depends on disjoint-set implementation
    - Most efficient:
      union-by-rank with path compression
    - CLRS 21
  - Make-Set O(|V|)
  - Sorting edges $O(|E| \log|E|)$
  - For loop (lines 5-8)
    - $O((|V| + |E|)\alpha(|V|)) \rightarrow O(|E|\alpha(|V|))$
    - $\alpha(|V|) = O(\log|V|) = O(\log|E|)$
  - Also, observing $|E| < |V|^2$
  - $O(|E| \log|V|)$

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

Running time?

Prim's Algorithm

- The set A forms a single tree.
- The safe edge added to A is always a least-weight edge connecting the tree to a vertex not in the tree.

- Very similar to Kruskal's algorithm
  - Greedy → At each step, it adds to the tree an edge that contributes the minimum amount possible to the tree's weight.
  - Growing MST
- Main difference
  - The edges in the growing set A always form a single tree, i.e., instead of starting from a forest of single-node trees, we start with an arbitrary node and grow the MST from that node by making greedy decisions, one at a time.
  - Greedy choice: At each step, we choose a "light edge" (min-weight) that connects current set A (the growing MST) to an uncovered vertex.

References

- The lecture slides are mainly based on the suggested textbooks and the corresponding published lecture notes:

  - CLRS: Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. Introduction to Algorithms, Third Edition, MIT Press, 2009.
  - KT: Kleinberg, J., & Tardos, E. Algorithm design. Pearson/Addison-Wesley, 2006.
  - DPV: Dasgupta, S., Papadimitriou, C. H., & Vazirani, U. V. Algorithms, McGraw-Hill Higher Education., 2008.
  - Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley.