CS 2022 : DATA STRUCTURES & ALGORITHMS

Graphs (Minimum Spanning Trees)

Malaka Walpola

Outline

- Depth First Search (DFS) & Edges
- Minimum Spanning Trees (MST)
- Steiner Minimum Trees
- MST Generation Algorithms
  - General Algorithm
  - Kruskal’s Algorithm
  - Prim’s Algorithm

Learning Outcomes

- After successfully studying contents covered in this lecture, students should be able to,
  - explain the idea of minimum spanning trees and their applications
  - explain the operation of the Kruskal’s and Prim’s algorithms used for the MST generation

Classification of Edges

- Tree Edge
  - In the depth-first forest
  - Found by exploring (u, v)
- Back Edge
  - (u, v), where u is a descendant of v (in the depth-first tree)

Classification of Edges

- Forward Edge
  - (u, v), where v is a descendant of u, but not a tree edge
- Cross Edge
  - Any other edge
  - Can go between vertices in same depth-first tree or in different depth-first trees

DFS & Edges

- DFS of an Undirected Graph
  - Only tree and back edges
  - No forward or cross edges
- Directed Acyclic Graph
  - A directed graph
  - No cyclic paths
  - A directed graph is acyclic iff DFS yields no back edges

Spanning Trees

- Graph G = (V, E)
  - Undirected & connected
- Spanning Tree
  - A **connected, acyclic** subgraph with all vertices
  - An **acyclic** subset of edges $T \subseteq E$ that connects all vertices
  - Tree: acyclic
  - Spanning: spans the graph

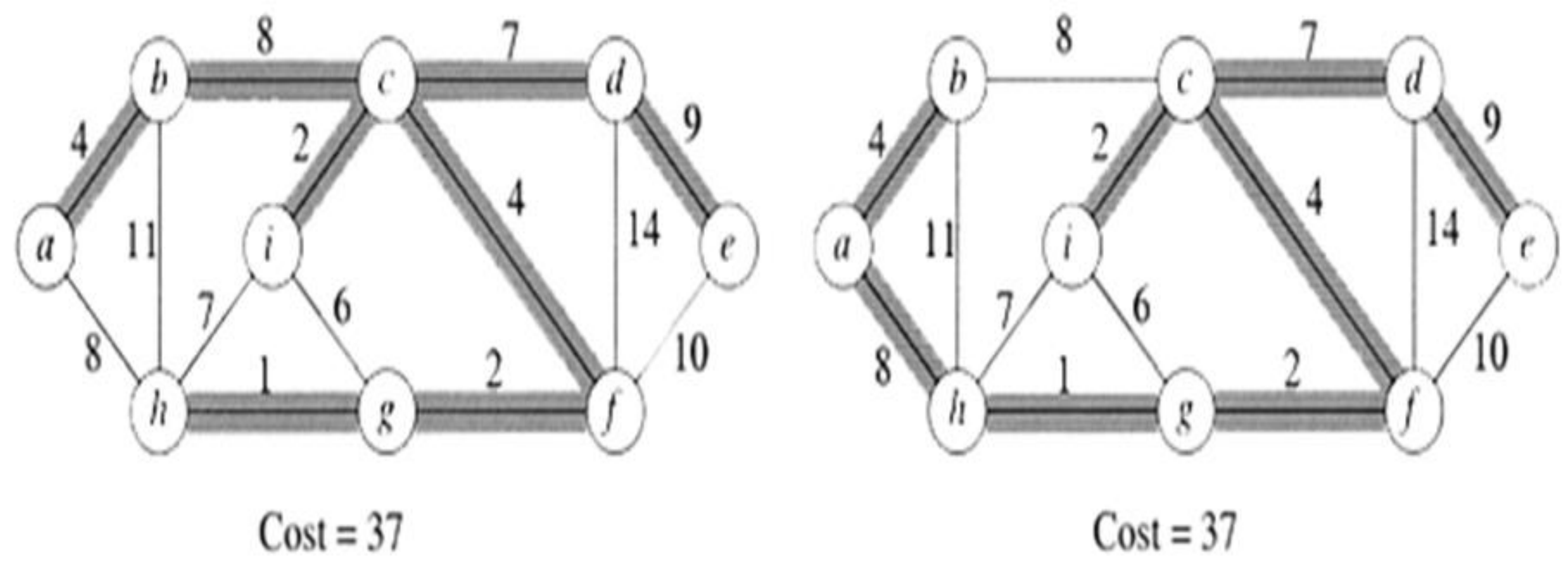
Minimum Spanning Trees

- Consider a Weighted Graph
- Cost of a Spanning Tree
  - Sum of edge weights in the spanning tree
- A Minimum Spanning Tree (MST)
  - A spanning tree with minimum weight

Minimum Spanning Trees

- Applications
  - Communication networks
  - Circuit design
  - Layout of highway systems
- Motivation
  - Minimize the connection cost

MST EXAMPLES



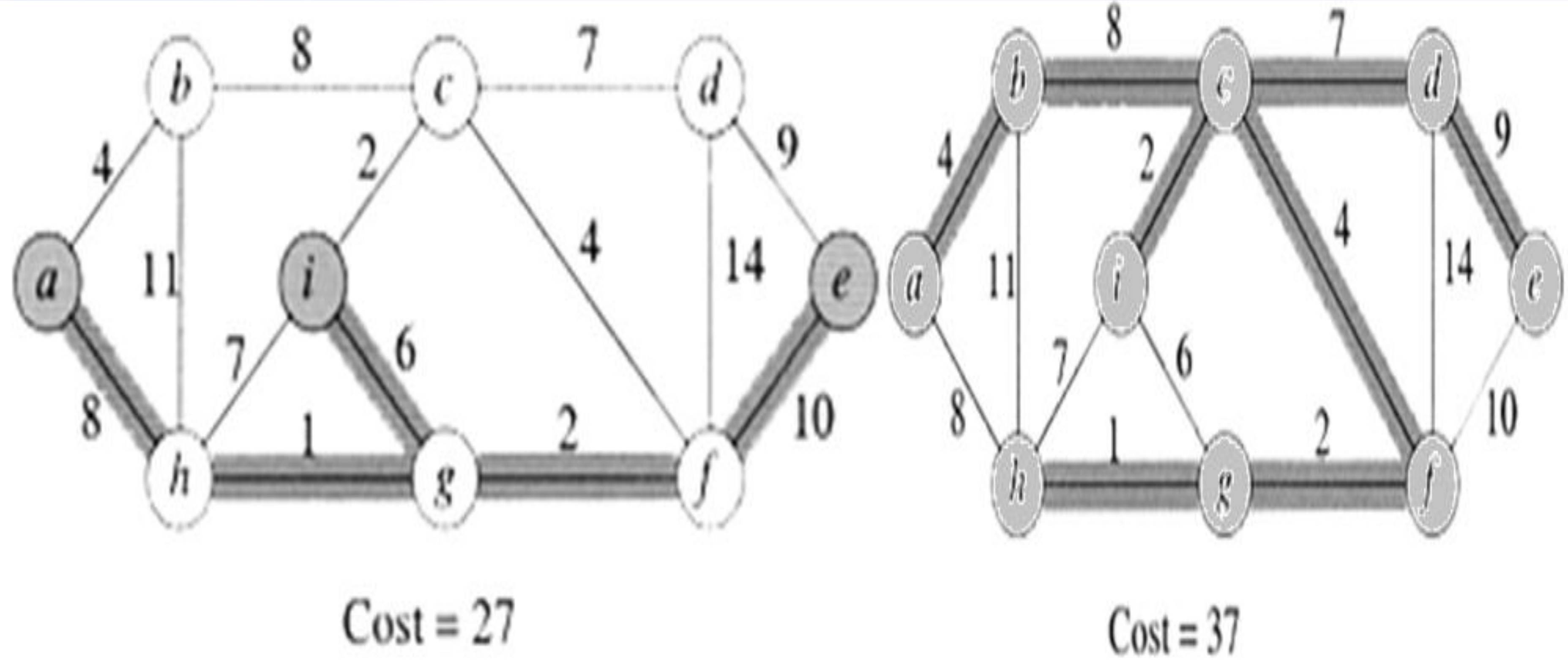
Note: A MST may not be unique

**Figure taken from [2]*

Steiner Minimum Trees

- Graph $G = (V, E)$
  - Weighted, undirected & connected
- Subset of Vertices $V' \subseteq V$, Called **Terminals**
- Steiner Minimum Trees (SMT)
  - A connected acyclic subgraph of G that includes all ***terminals***

SMT Example



- MST is just a SMT with $V' = V$

Generating MST - Idea

- Graph $G = (V, E)$
- Maintain a Subset of Edges A
  - Initially empty
  - Add one edge at a time
  - The edge should be **safe**
- Keep Adding Until $G' = (V, A)$ is MST

Generating MST

```
Generic-MST(G, w)
1. A ← ∅
2. while A does not form a spanning tree
3.    find an edge (u,v) that is safe for A
4. A ← A ∪ {(u,v)}
5. return A
```

Safe Edge

- A Subset of Edges $A \subseteq E$
  - A is a subset of edges in some MST
  - It is possible to extend (V, A) into a MST
- An Edge $(u,v) \in E\text{-}A$ is **Safe** if $A \cup \{(u,v)\}$ is a Subset of Edges in Some MST
  - It is possible to extend $(V, A \cup \{(u,v)\})$ into a MST

Safe Edge

- Definitions
  - A **cut** (S, V-S) is just a partition of the vertices into 2 ***disjoint*** subsets
  - An edge (u, v) **crosses** the cut if one endpoint is in S and the other is in V-S
  - Given a subset of edges A, we say that a cut **respects** A if no edge in A ***crosses*** the cut
  - An edge of E is a **light edge** crossing a cut, if among all edges crossing the cut, it has the minimum weight

Finding a Safe Edge

- Graph $G = (V, E)$
  - Connected, undirected & weighted
- A Subset of Edges $A \subseteq E$
  - A is a subset of edges in some MST
- Theorem
  - A Cut $(S, V\text{-}S)$ Which Respects A
  - (u,v) a Light Edge Crossing This Cut
  - The Edge (u,v) is Safe

MST Generation Algorithms

- Two greedy algorithms for computing MSTs
  - Kruskal's Algorithm
    - Start with a forest with single vertex trees
    - Adds edges in increasing order of weight
    - Trees merge into a single tree
  - Prim's Algorithm
    - Start with a single vertex as the root node of the tree
    - Adds one node at a time to the current tree
    - The tree grows until it spans all the vertices

Kruskal's Algorithm - Idea

- Start With a Forest With Single Vertex Trees
- Adds Edges in Increasing Order of Weights
  - If the next edge does not induce a cycle among the current set of edges, then it is added to A
  - If it does, then this edge is passed over, and the next edge is considered
  - How to detect cycles?

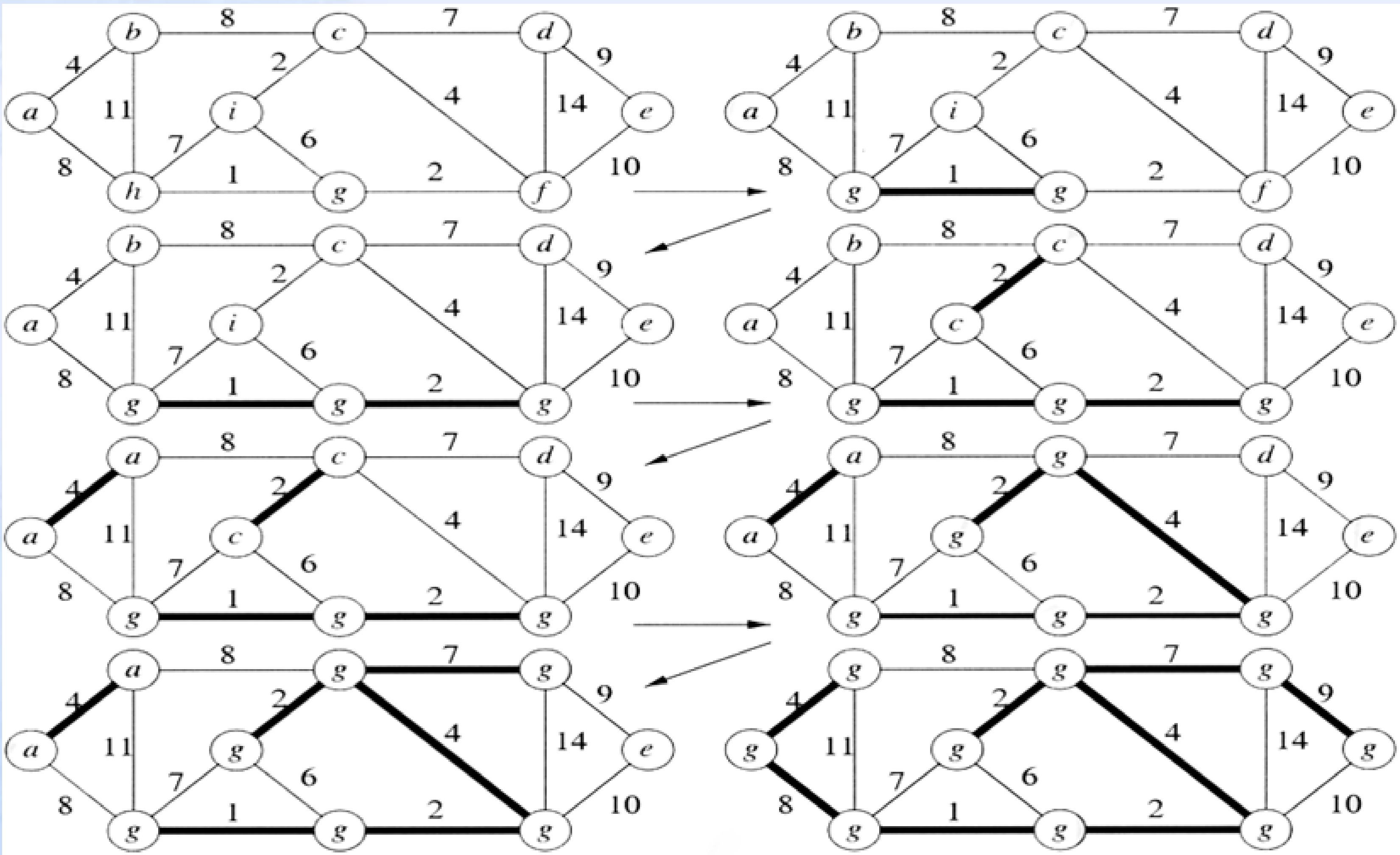
Kruskal's Algorithm - Idea

- Trees Merge Into a Single Tree
  - Each tree is connected
  - If a new edge is added to a tree it will induce a cycle
  - However if we add an edge which connects two trees, there will be no cycles
  - Thus, add such edges
  - After adding the edge, the two trees merge into a single tree

Kruskal's Algorithm

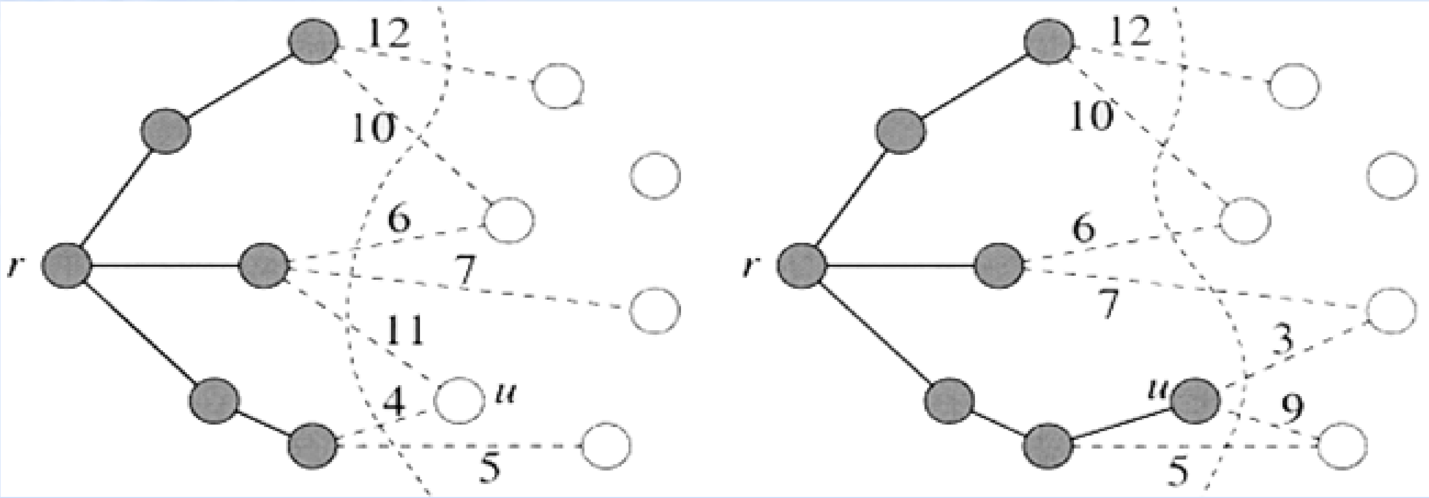
```
MST-Kruskal(G, w)
1. A ← ∅
2. For each vertex v ∈ G.V
3.    MAKE-SET(v)
4. sort the edges of G.E in nondecreasing
   order of weight
5. for each edge (u, v)∈G.E, in order of
   nondecreasing weight
6.    if FIND-SET(u) ≠ FIND-SET (v)
7.        A ← A ∪ {(u,v)}
8.        UNION(u,v)
9. return A
```

Kruskal's Algorithm - Example



Prim's Algorithm - Idea

- A Tree With a Single Vertex as the Root Node
- Adds One Leave (and a Vertex) at a Time to the Current Tree
  - At any time, the subset of edges A forms a single tree; S = vertices of A



Prim's Algorithm - Idea

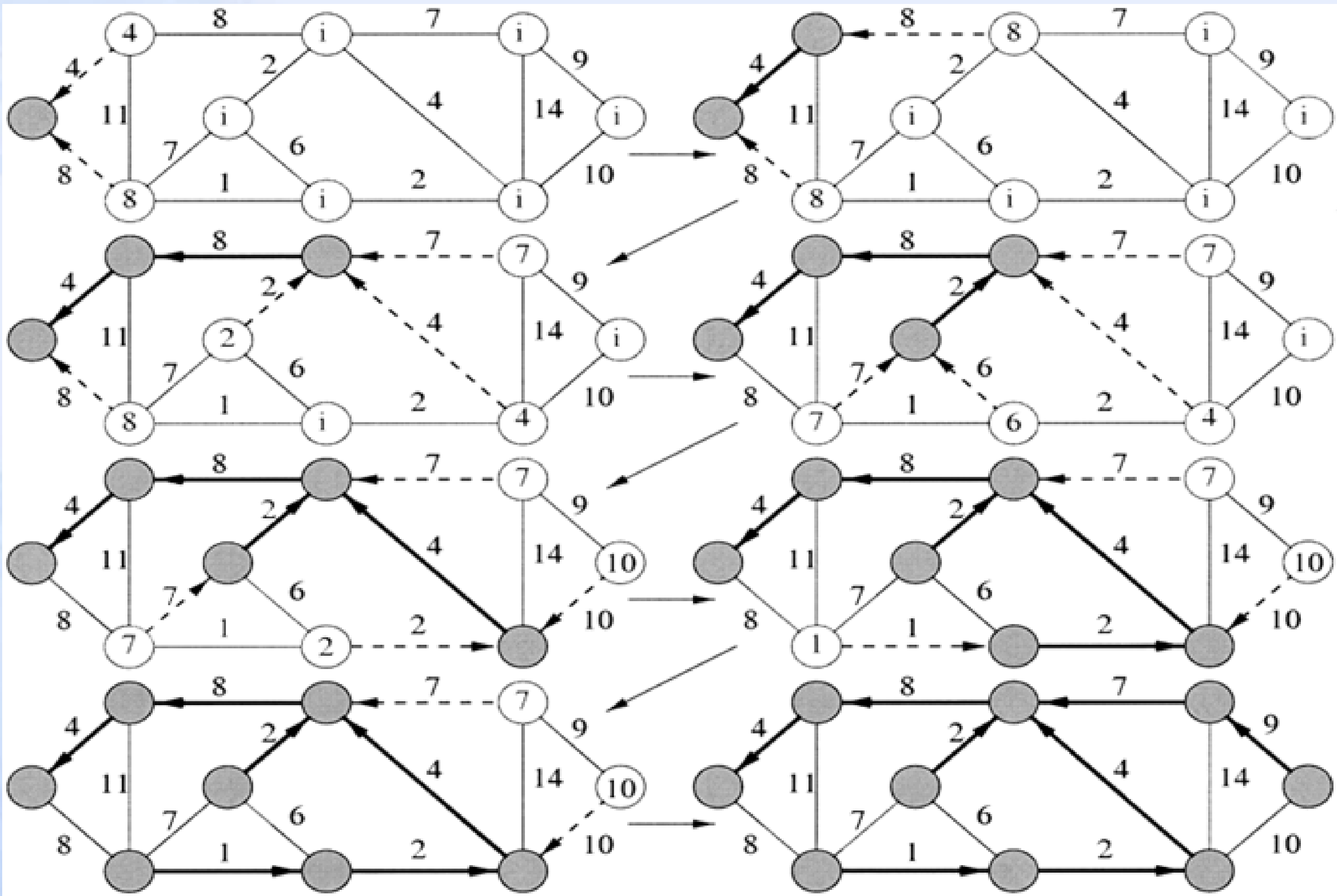
- Consider the set of vertices S currently part of the tree, and its complement (V-S)
- We have a cut of the graph
- the current set of tree edges A is respected by this cut
- Which edge should we add next? **Light edge**!

- The tree grows until it spans all the vertices in V

Prim's Algorithm

```
MST-Prim(G, w, r)
for each vertex u ∈ G.V
    u.key = ∞
    u. π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u,v)
```

Prim's Algorithm - Example



Self Studying

Self Studying

- Reading Assignment
  - Chapter 23
    - 23.1: Growing a Minimum Spanning Tree
    - 23.2: The Algorithms of Kruskal and Prim

REFERENCES

[1] T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, *Introduction to Algorithms*, 3rd Ed. Cambridge, MA, MIT Press, 2009.

[2] Lecture slides available at http://www.cs.unc.edu/~plaisted/comp550/24.ppt

[3] Lecture slides available at http://www.cs.unc.edu/~plaisted/comp550/25.ppt