WIA2005: Algorithm Design and Analysis

Semester 2, Session 2016/17

Lecture 11: Graph Algorithm – Minimum Spanning Tree

Learning objectives

- Know what is Minimum Spanning Tree
  - Prim’s Algorithm
  - Kruskal’s Algorithm

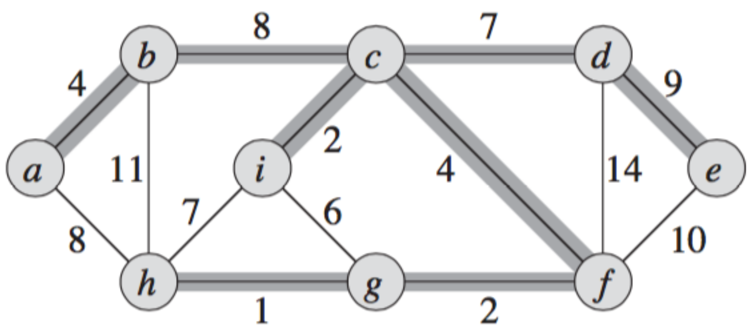
Introduction

- Electronic circuit designs often need to make the pins of several components electrically equivalent by wiring them together.
- To interconnect a set of n pins, we can use an arrangement of n 1 wires, each connecting two pins.
  - the lesser the better.
- We can model this wiring problem with a connected, undirected graph G = (V,E), where V is the set of pins, E is the set of possible interconnections between pairs of pins, and for each edge (u,v) contains E, we have a weight w(u,v) specifying the cost (amount of wire needed) to connect u and v.
- We then wish to find an acyclic T subset E that connects all of the vertices and whose total weight of the following is minimum.

$$w(T) = \sum_{(u,v)\in T} w(u,v)$$

Minimum Spanning Tree

- Since T is acyclic and connects all of the vertices, it must form a tree, which we call a ***spanning tree*** since it “spans” the graph G.
- We call the problem of determining the tree T the ***minimum-spanning-tree problem***.



Growing a minimum spanning tree

- Assume that we have a connected, undirected graph G=(V,E) with a weight function w:E -> R, and we wish to find a minimum spanning tree for G.
- The two algorithms we consider uses a greedy approach to the problem, although they differ in how they apply this approach.

Greedy Algorithm for MST

- 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, in the sense that A U {(u,v)} is also a subset of a minimum spanning tree.
- We call such an edge a ***safe edge*** for A, since we can add it safely to A while maintaining the invariant.

The algorithms of Kruskal and Prim

- The two minimum-spanning-tree algorithms described here elaborate on the generic method.
- They each use a specific rule to determine a safe edge.
- In 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.
- In 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

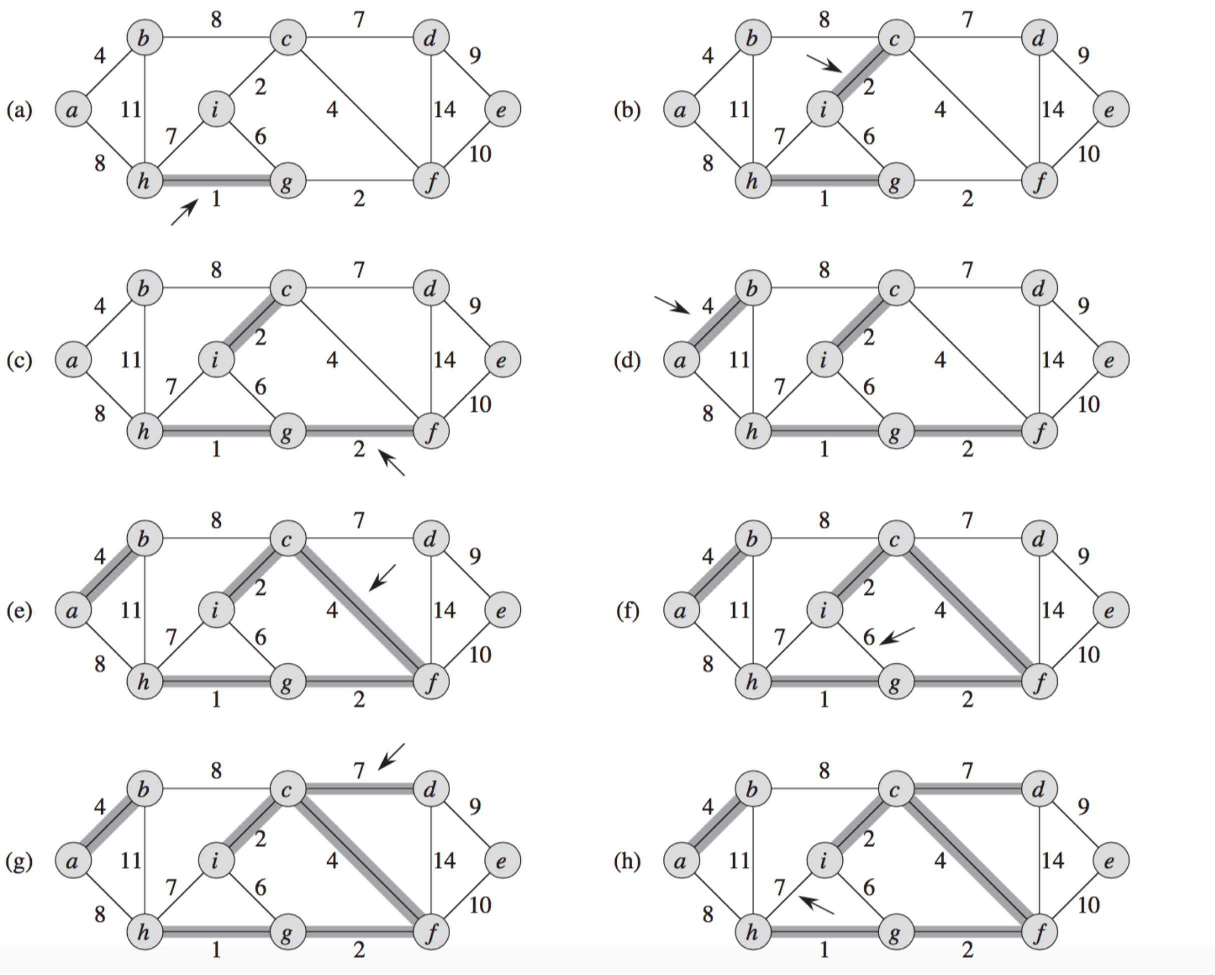
- Kruskal’s algorithm finds a safe edge to add to the growing forest by finding, of all the edges that connect any two trees in the forest, an edge (u,v) of least weight.
- It uses a disjoint-set data structure to maintain several disjoint sets of elements.
- Each set contains the vertices in one tree of the current forest. The operation FIND-SET(u) returns a representative element from the set that contains u.
- Thus, we can determine whether two vertices u and belong to the same tree by testing whether FIND-SET(u) equals FIND-SET(v).
- To combine trees, Kruskal’s algorithm calls the UNION procedure.

Kruskal's algorithm

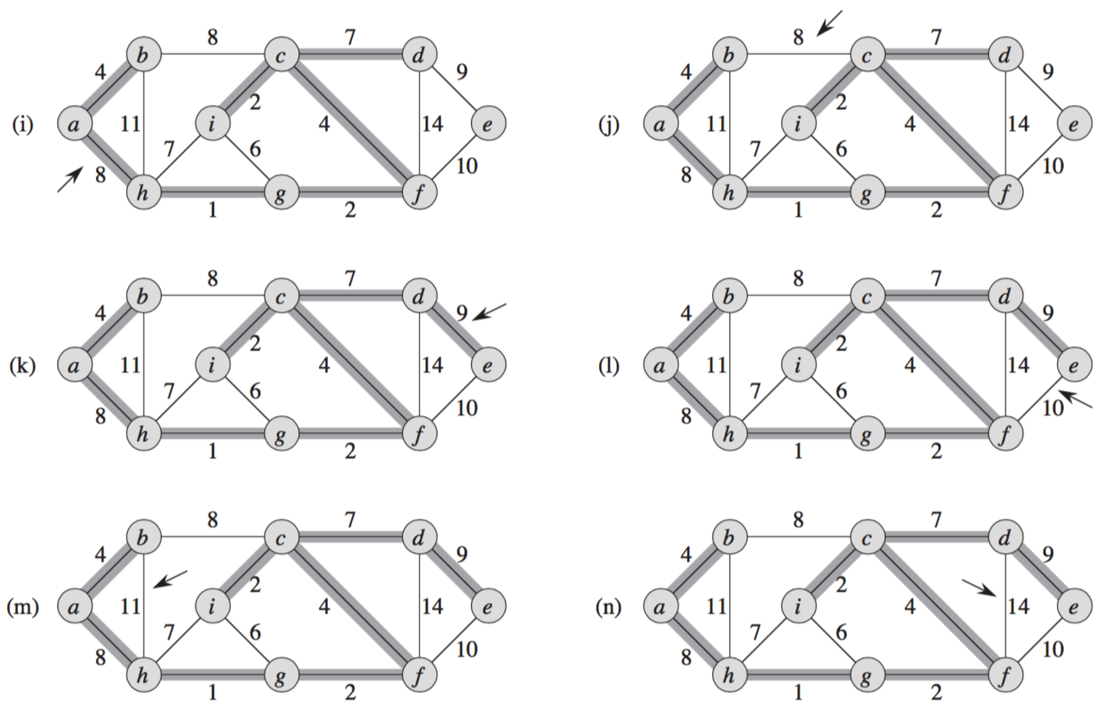
- Pseudocode

```
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
```

Kruskal's algorithm



Kruskal's algorithm



Running time of Kruskal's algorithm

- The running time of Kruskal's algorithm for a graph G=(V,E) depends on how we implement the disjoint-set data structure.
- The running time of Kruskal's algorithm as O(E lg V).

Prim's algorithm

- Prim's algorithm operates much like Dijkstra's algorithm for finding shortest paths in a graph.
- Prim's algorithm has the property that the edges in the set A always form a single tree.
- The tree starts from an arbitrary root vertex r and grows until the tree spans all the vertices in V .
- Each step adds to the tree A a light edge that connects A to an isolated vertex—one on which no edge of A is incident.
- This strategy qualifies as greedy since at each step it adds to the tree an edge that contributes the minimum amount possible to the tree's weight.

Prim’s algorithm

- In order to implement Prim’s algorithm efficiently, we need a fast way to select a new edge to add to the tree formed by the edges in A.
- In the pseudocode below, the connected graph G and the root r of the minimum spanning tree to be grown are inputs to the algorithm.
- During execution of the algorithm, all vertices that are *not* in the tree reside in a min-priority queue Q based on a *key* attribute.
- For each vertex , the attribute v.*key* is the minimum weight of any edge connecting to a vertex in the tree; by convention, v.*key* = ∞ if there is no such edge.
- The attribute : names the parent of in the tree.

Prim's algorithm

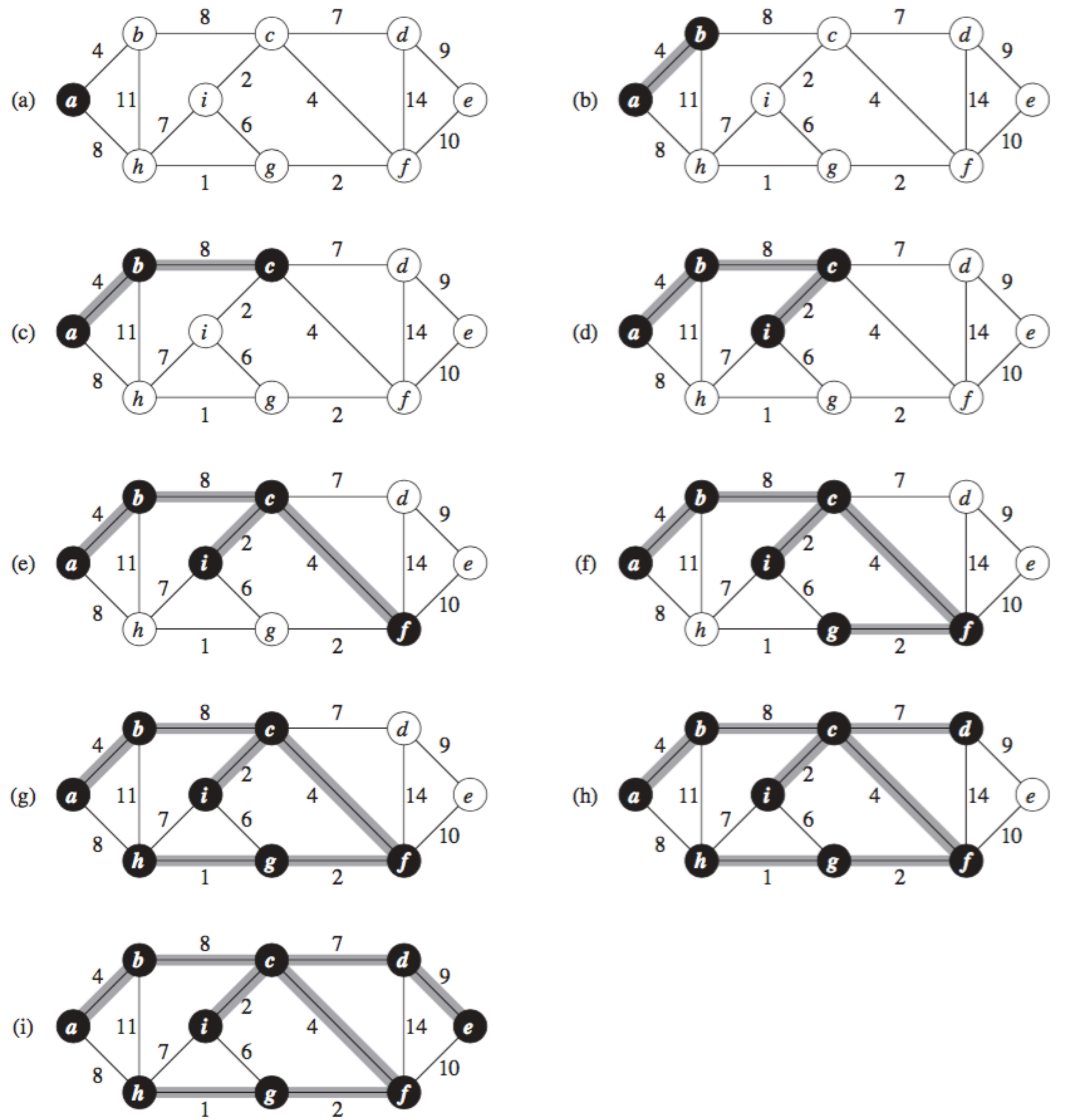
Pseudocode

- The algorithm implicitly maintains the set A from GENERIC-MST as $A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$
- When the algorithm terminates, the min-priority queue Q is empty; the minimum spanning tree A for G is thus

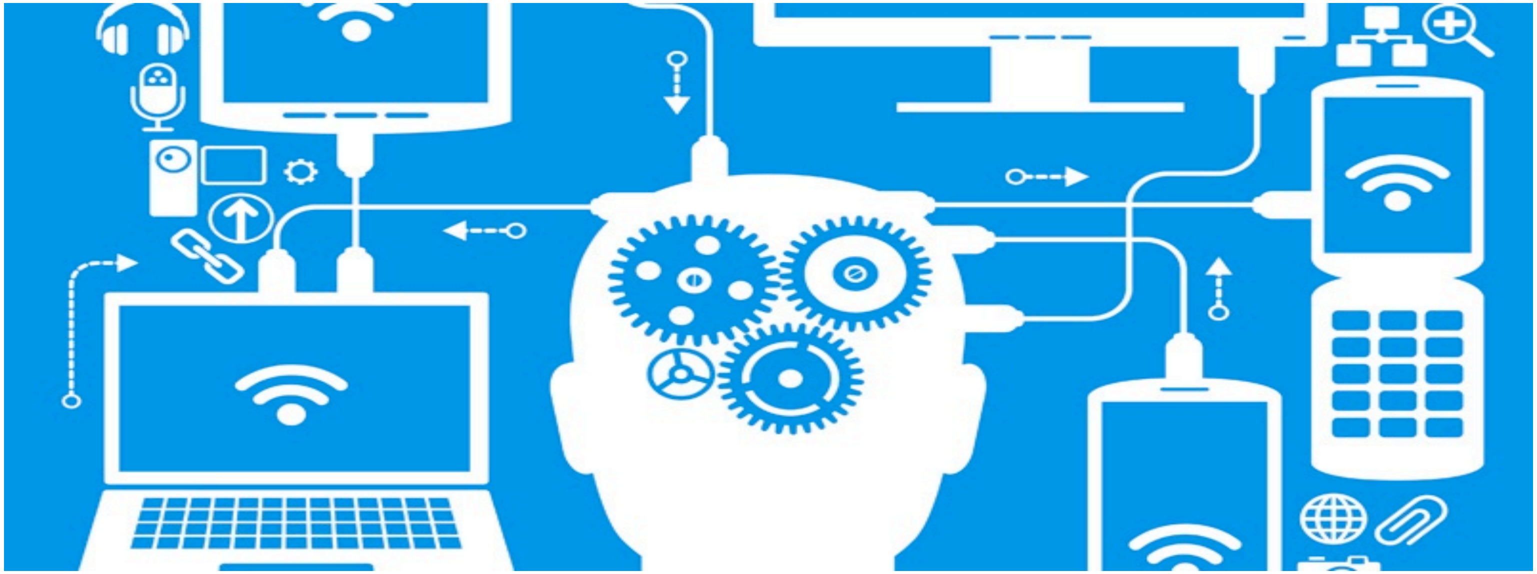
$$A = \{(v, v.\pi) : v \in V - \{r\}\} \ .$$

```
MST-PRIM(G, w, r)
for each 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



In the next lecture..



Lecture 12: String Matching