Lecture 15: Minimum Spanning Tree Problem

MCS/CS 401 Computer Algorithms I — July 22, 2022
Instructor: Danko Adrovic

Lecture Outline

- Minimum Spanning Tree Problem
  - Growing a minimum spanning tree
  - Kruskals algorithm
  - Time complexity of Kruskals algorithm
  - Prim's algorithm
  - Time complexity of Prims algorithm

1 Minimum Spanning Tree Problem

A minimum spanning tree (MST) of a edge-weighted, connected, undirected graph $G = (V, E)$, is a subset of edges consisting of a *minimum* possible total edge weight, which leaves all vertices connected with each other.

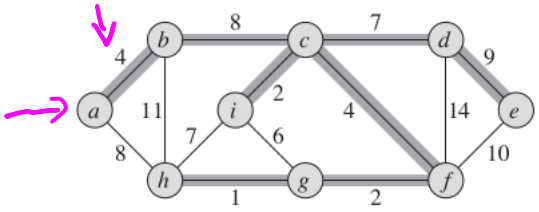


Figure 23.1 A minimum spanning tree for a connected graph.

Such minimum spanning trees (or minimum weight spanning tree) are often used to model problems in which we want to keep all vertices connect with each other, such that total weight of the edges is minimized. This usually leads to a reduction of some kind of a cost associated with the problem. In the image above, the edges may be representing wiring in an object - and the question is what is the smallest amount of wire needed to keep all those ports (vertices) connected.

It can be easily seen that there are several ways to lay down the wire but many of those configurations would require more wire than is really needed. The goal of this lecture is related to the question: given a graph G, can we systematically discover its edge-weighted minimum spanning tree?

Formally, we can rephrase the question in the following way. Let $G = (V, E)$ denote a graph, where V is the set of vertices and E is the set of edges in G. For each edge $(u, v) \in E$, with weight $w(u, v)$, we wish to find an acyclic subset $T \subseteq E$, which connects all of the vertices and whose total weight

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

is minimized. We call the problem of determining the tree T the *minimum-spanning-tree problem*.

1.1 Growing a minimum spanning tree

Assume that we have a connected, undirected graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}$, and we wish to find a minimum spanning tree for graph G. Consider now the following **greedy** strategy, captured by the following *generic* method, which grows the tree one edge at a time.

```
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
```

This generic method manages a set of edges, denoted A, while maintaining the following, *tricky*, loop invariant: *Prior to each iteration, A is a subset of some minimum spanning tree.* This invariant is *tricky* because we do not know how to determine whether an edge is safe or not - a **safe edge** is an edge that we can add to the set A. However, we know that one such edge must exist, since when line 3 is executed, the invariant dictates that there is a spanning tree T such that $A \subseteq T$.

This generic method forms a basis for two **greedy** algorithms, Kurskal's and Prim's, which solve the minimum-spanning-tree problem, differing only in the way they apply the greedy approach.

Before we can prove the correctness of the generic approach method *GENERIC-MST* and its use of the **invariant**, we need some definitions.

A **cut** $(S, V - S)$ of an undirected graph $G = (V, E)$ is a partition of V (see Figure 23.2). We say that an edge $(u, v) \in E$ **crosses** the cut $(S, V - S)$ if one of its endpoints is in S and the other is in $V - S$. We say that 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.

$A = \{(a,b), (h,g), (c,i), (g,f), (c,f)\}$

the cut $(S, V-S)$ respects A

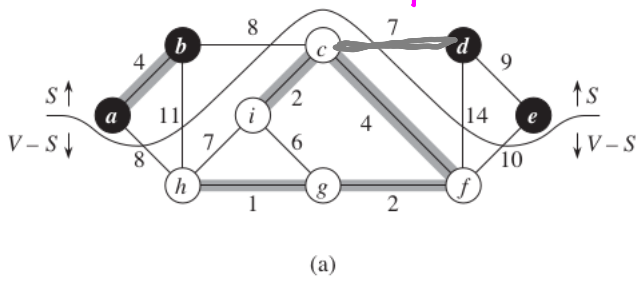


Figure 23.2 Two ways of viewing a cut $(S, V - S)$ of the graph from Figure 23.1. **(a)** Black vertices are in the set S, and white vertices are in $V - S$. The edges crossing the cut are those connecting white vertices with black vertices. The edge (d, c) is the unique light edge crossing the cut. A subset A of the edges is shaded; note that the cut $(S, V - S)$ respects A, since no edge of A crosses the cut. **(b)** The same graph with the vertices in the set S on the left and the vertices in the set $V - S$ on the right. An edge crosses the cut if it connects a vertex on the left with a vertex on the right.

The rule for recognizing **safe edges** is given by the following theorem.

Theorem 1. *Let $G = (V, E)$ be a connected, undirected graph with 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 $(S, V - S)$. Then edge (u, v) is safe for A.*

Proof. Let T be a minimum spanning tree that includes A, and assume that T does not contain the light edge (u, v), since if it does, we are done. We will construct another minimum spanning tree T' that includes $A \cup \{(u, v)\}$, thereby showing that (u, v) is a safe edge for A.

The edge (u, v) forms a cycle with the edges on the simple path p from u to v in T, as Figure 23.3 illustrates. Since u and v are on opposite sides of the cut $(S, V - S)$, at least one edge in T lies on the simple path p and also crosses the cut. Let (x, y) be any such edge. The edge (x, y) is not in A, because the cut respects A. Since (x, y) is on the unique simple path from u to v in T, removing (x, y) breaks T into two components. Adding (u, v) reconnects them to form a new spanning tree $T' = T \setminus \{(x, y)\} \cup \{(u, v)\}$.

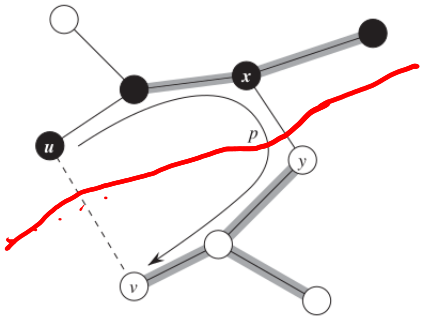


Figure 23.3 The proof of Theorem 23.1. Black vertices are in S, and white vertices are in $V - S$. The edges in the minimum spanning tree T are shown, but the edges in the graph G are not. The edges in A are shaded, and (u, v) is a light edge crossing the cut $(S, V - S)$. The edge (x, y) is an edge on the unique simple path p from u to v in T. To form a minimum spanning tree T' that contains (u, v), remove the edge (x, y) from T and add the edge (u, v).

Next, we show that T' is a minimum spanning tree. Since (u, v) is a light edge crossing $(S, V - S)$ and (x, y) also crosses this cut, $w(u, v) \leq w(x, y)$. Therefore,

$$w(T') = w(T) - w(x, y) + w(u, v) \leq w(T).$$

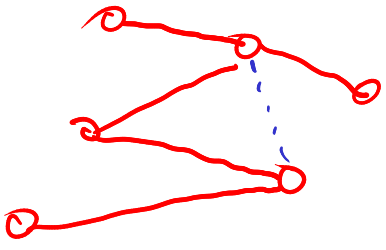
But T is a minimum spanning tree, so that $w(T) \leq w(T')$; therefore, T' must be a minimum spanning tree as well.

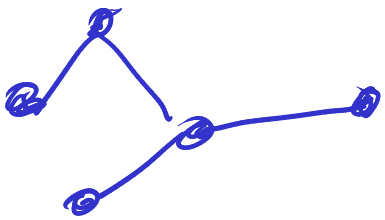
It remains to show that (u, v) is actually a safe edge for A. We have $A \subseteq T'$, since $A \subseteq T$ and $(x, y) \notin A$; Therefore, $A \cup \{(u, v)\} \subseteq T'$. Consequently, since T' is a minimum spanning tree, (u, v) is safe for A. □

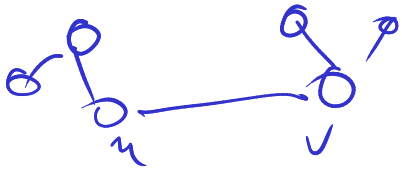
This theorem gives us a better understanding of the workings of the *GENERIC-MST* method on a connected graph $G = (V, E)$. As the method proceeds, the set A is always acyclic; otherwise, a minimum spanning tree including A would contain a cycle, which is a contradiction. At any point in the execution, the graph $G_A = (V, A)$ is a forest, and each of the connected components of G_A is a tree. Some of the trees may contain just one vertex, as is the case, for example, when the method begins: A is empty and the forest contains $|V|$ trees, one for each vertex. Furthermore, any safe edge (u, v) for A connects distinct components of G_A, since $A \cup \{(u, v)\}$ must be acyclic.

The while loop in lines 24 of *GENERIC-MST* executes $|V| - 1$ times because it finds one of the $|V| - 1$ edges of a minimum spanning tree in each iteration. Initially, when $A = \emptyset$, there are $|V|$ trees in G_A, and each iteration reduces that number by 1. When the forest contains only a single tree, the method terminates.

The two algorithms, Kruskal's and Prim's, use the following corollary to Theorem we just proved.







Corollary 1. *Let $G = (V, E)$ be a connected, undirected graph with 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, and 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 (u, v) is safe for A.*

Proof. The cut $(V_C, V - V_C)$ respects A, and (u, v) is a light edge for this cut. Therefore, (u, v) is safe for A. □

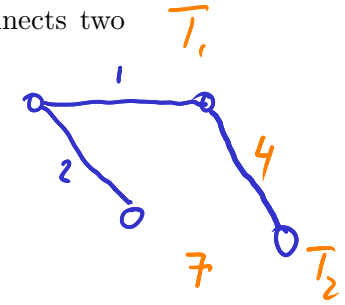
Proposition 1. *If an edge (u, v) is contained in some minimum spanning tree, then it is a light edge crossing some cut of the graph.*

Proof. Let the MST T be a union of two trees T_1 and T_2, separated by an edge (u, v). For the sake of contradiction, assume that there is an edge (x, y) in this cut with a weight less than that of (u, v). Then, by using the edge (x, y), we can construct a tree T' as $T' = T_1 \cup \{(x, y)\} \cup T_2$. This would mean that T' is a MST with total weight less than that of T, our original MST. This contradicts the fact that T was an MST to begin with, and so our assumption is wrong, and the proof is complete. □

1.2 Kruskals algorithm

The Kruskal's algorithm expands on the generic method *GENERIC-MST* from the previous section. It use a specific rule to determine a safe edge in line 3 of *GENERIC-MST*. In Kruskals 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-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
```



$A = \{(h,g), (i,c), (g,f), \ldots$

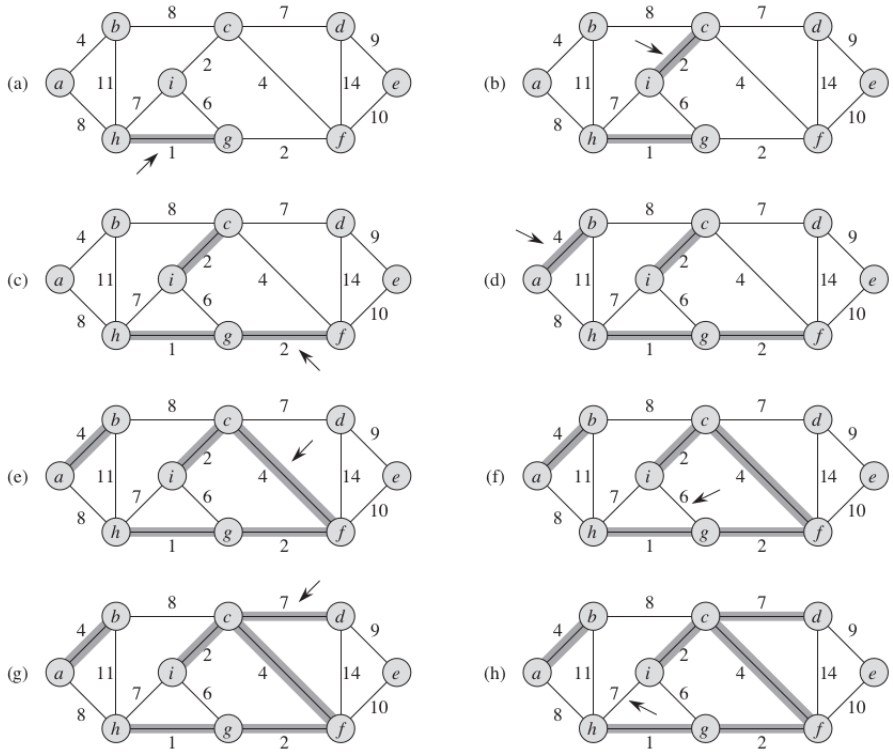


Figure 23.4 The execution of Kruskal's algorithm on the graph from Figure 23.1. Shaded edges belong to the forest A being grown. The algorithm considers each edge in sorted order by weight. An arrow points to the edge under consideration at each step of the algorithm. If the edge joins two distinct trees in the forest, it is added to the forest, thereby merging the two trees.

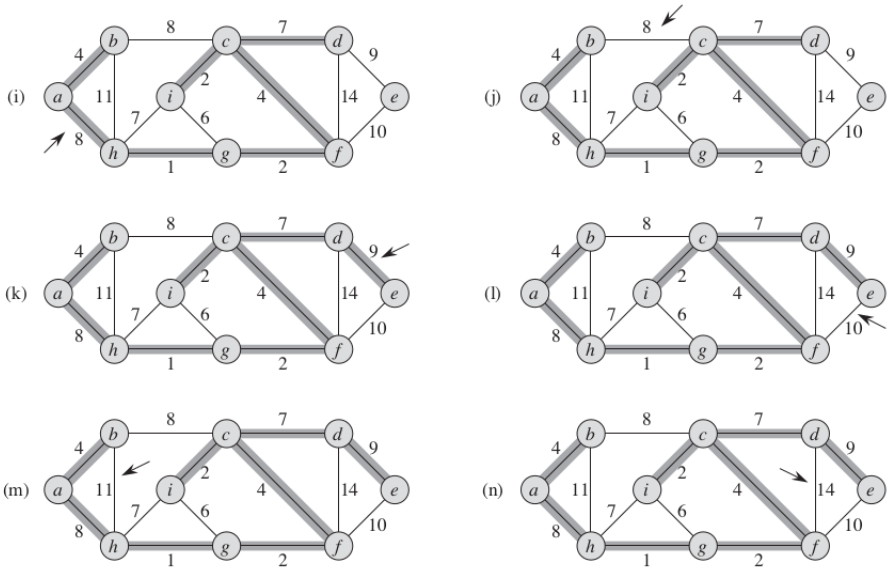


Figure 23.4, continued Further steps in the execution of Kruskal's algorithm.

1.3 Time complexity of Kruskal's algorithm

The running time of Kruskals algorithm for a graph $G = (V, E)$ depends on how we implement the disjoint-set data structure. An implementation using disjoint-set-forest implementation (method we have not covered) leads to a time $O(E \cdot lg(V))$.

1.4 Prim's algorithm

Prims algorithm is also a special case of the generic minimum-spanning-tree method *GENERIC-MST*. In Prims 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.

Prims 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. By Corollary for the previous section, this rule adds only edges that are safe for A; therefore, when the algorithm terminates, the edges in A form a minimum spanning tree. 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.

In order to implement Prims 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 v, the attribute $v.key$ is the minimum weight of any edge connecting v to a vertex in the tree; by convention, $v.key = \infty$ if there is no such edge. The attribute $v.\pi$ names the parent of v in the tree. The algorithm imlicitly maintains a set A from *GENERIC-MST* as $A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$.

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

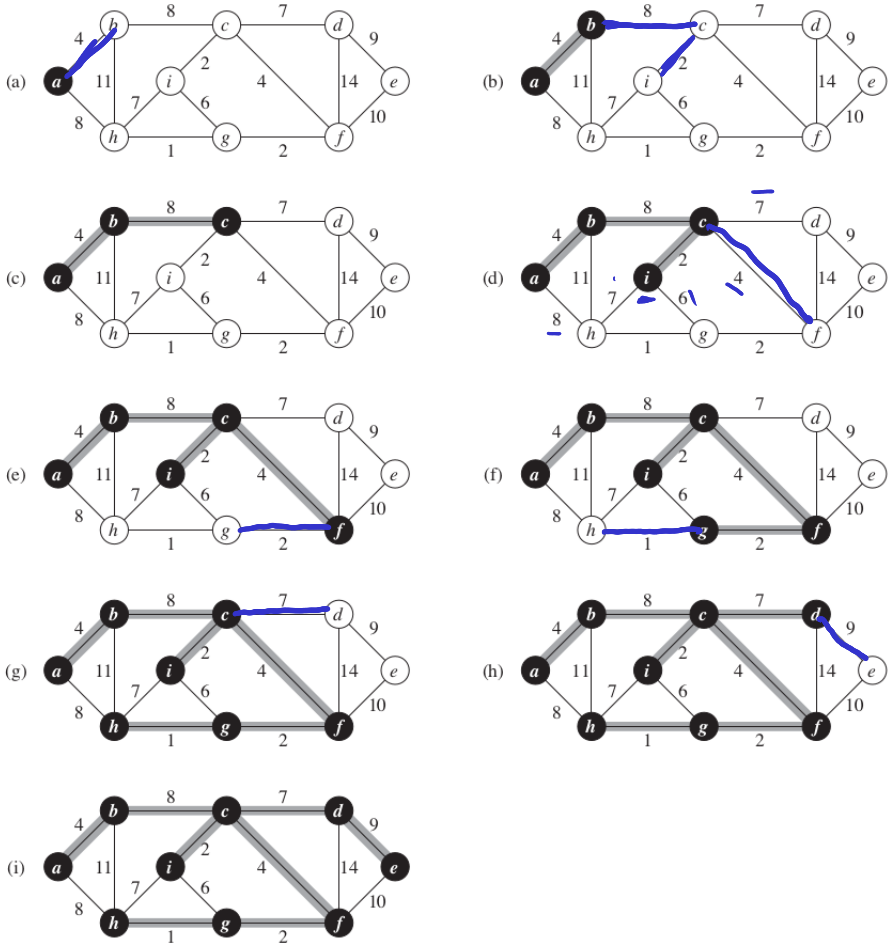


Figure 23.5 The execution of Prim's algorithm on the graph from Figure 23.1. The root vertex is a. Shaded edges are in the tree being grown, and black vertices are in the tree. At each step of the algorithm, the vertices in the tree determine a cut of the graph, and a light edge crossing the cut is added to the tree. In the second step, for example, the algorithm has a choice of adding either edge (b, c) or edge (a, h) to the tree since both are light edges crossing the cut.

1.5 Time complexity of Prim's algorithm

The running time of Prim's algorithm for a graph $G = (V, E)$ is like that of Kruskal's algorithm: $O(E\ lg(V))$. However, if a Fibonacci heap is used to implement the min-priority queue Q, the running time of Prim's algorithm can be improved to $O(E + V\ lg(V))$.