Introduction to Algorithms

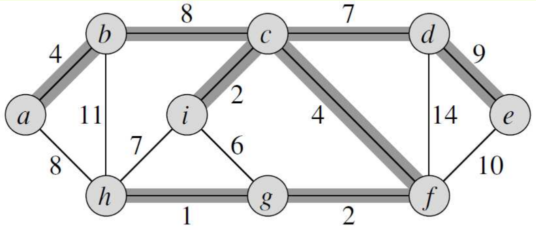
George Voutsadakis[1]

[1]Mathematics and Computer Science
Lake Superior State University

LSSU Math 400

Minimum Spanning Tree Problem

- Consider a connected, undirected graph $G = (V, E)$, where, for each edge $(u, v) \in E$, we have a weight/cost $w(u, v)$ for connecting u, v.
- We wish to find an acyclic subset $T \subseteq E$ that connects all of the vertices and whose total weight $w(T) = \sum_{(u,v)\in T} w(u, v)$ is minimized.
- 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**.



Two Greedy Algorithms

- We examine two algorithms for solving the minimum spanning tree problem.
  - Kruskal's algorithm;
  - Prim's algorithm.
- The two algorithms are greedy algorithms. Each step of a greedy algorithm must make one of several possible choices. The greedy strategy makes the choice that is the best at the moment.
  - Such a strategy does not generally guarantee that it will always find globally optimal solutions to problems.
  - For the minimum spanning tree problem, however, we can prove that certain greedy strategies do yield a spanning tree with minimum weight.

Subsection 1

Growing a Minimum Spanning Tree

Invariant for the Generic Algorithm

- 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 G.
- The greedy strategy of both algorithms is captured by the following "generic" algorithm, which grows the minimum spanning tree one edge at a time.
- The algorithm 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 can be added to A without violating this invariant, in the sense that $A \cup \{(u, v)\}$ is also a subset of a minimum spanning tree.
- We call such an edge a **safe edge** for A, since it can be safely added to A while maintaining the invariant.

The Generic Algorithm

GenericMST(G, w)

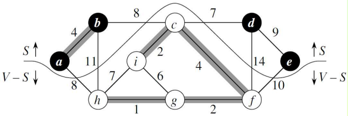
1. $A = \emptyset$
2. while A does not form a spanning tree
3.   find an edge (u, v) that is safe for A
4. $A = A \cup \{(u, v)\}$
5. return A

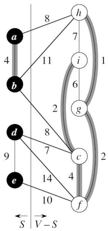
- We use the loop invariant as follows:
  - **Initialization**: After Line 1, the set A trivially satisfies the loop invariant.
  - **Maintenance**: The loop in Lines 2-4 maintains the invariant by adding only safe edges.
  - **Termination**: All edges added to A are in a minimum spanning tree. So the set A returned in Line 5 must be a minimum spanning tree.
- The tricky part is, of course, finding a safe edge in Line 3.

  One must exist, since, when Line 3 is executed, the invariant dictates that there is a spanning tree T such that $A \subseteq T$.

Preparing a Rule to Recognize Safe Edges

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





- 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.
- Note that there can be more than one light edge crossing a cut.
- More generally, we say that an edge is a **light edge** satisfying a given property if its weight is the minimum of any edge satisfying the property.

Recognizing Safe Edges

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 $(S, V - S)$. Then, edge (u, v) is safe for A.

- Let T be a minimum spanning tree that includes A. Assume that T does not contain the light edge (u, v), since if it does, we are done. We shall construct another minimum spanning tree T' that includes $A \cup \{(u, v)\}$ by using a cut-and-paste technique, thereby showing that (u, v) is a safe edge for A. The edge (u, v) forms a cycle with the edges on the path p from u to v in T. Since u and v are on opposite sides of the cut $(S, V - S)$, there is at least one edge in T on the path p that also crosses the cut. Let (x, y) be any such edge.

Recognizing Safe Edges (Cont'd)

- The edge (x, y) is not in A, because the cut respects A. Since (x, y) is on the unique 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 $T' = (T - \{(x, y)\}) \cup \{(u, v)\}$.
  - We show that T' is a minimum spanning tree. Since (u, v) is a light edge crossing $(S, V - S)$ and (x, y) crosses the 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 $w(T) \leq w(T')$. Thus, T' must be a minimum spanning tree also.
  - We show that (u, v) is actually a safe edge for A. We have $A \subseteq T'$, since $A \subseteq T$ and $(x, y) \notin A$. Thus, $A \cup \{(u, v)\} \subseteq T'$. Consequently, since T' is a minimum spanning tree, (u, v) is safe for A.

How GENERICMST Works

- As GENERICMST proceeds on a connected graph $G = (V, E)$, the set A is always acyclic.
- 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.
- Moreover, any safe edge (u, v) for A connects distinct components of G_A, since $A \cup \{(u, v)\}$ must be acyclic.
- The loop in Lines 2-4 is executed $|V| - 1$ times as each of the $|V| - 1$ edges of a minimum spanning tree is successively determined.

  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 algorithm terminates.

Light Edges Between Components are Safe

Corollary

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 $C = (V_C, E_C)$ 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.

- 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.

Subsection 2

Kruskal's Algorithm

Kruskal's Algorithm

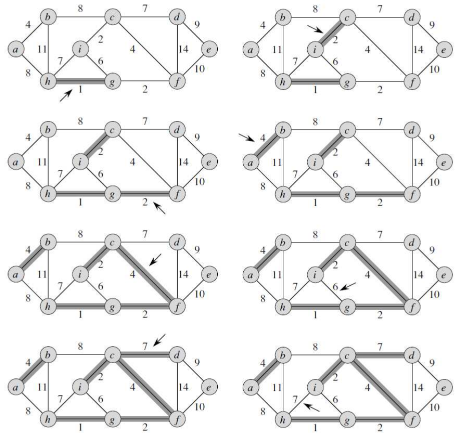
- Kruskal's algorithm finds a safe edge to add to the growing forest by finding, of all the edges that connect any two trees in the forest, an edge (u, v) of least weight.
- Let C_1 and C_2 denote the two trees that are connected by (u, v). Since (u, v) must be a light edge connecting C_1 to some other tree, the corollary implies that (u, v) is a safe edge for C_1.
- Our implementation of Kruskal's 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 $\textsc{FindSet}(u)$ returns a representative element from the set that contains u.
- Thus, we can determine whether two vertices u and v belong to the same tree by testing whether $\textsc{FindSet}(u)$ equals $\textsc{FindSet}(v)$.
- To combine trees, Kruskal's algorithm calls the $\textsc{Union}$ procedure.

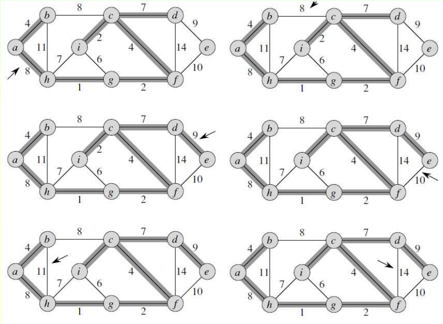
The Kruskal Procedure

MSTKruskal(G, w)

```
A = ∅
for each vertex v ∈ G.V
    MakeSet(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 FinfSet(u) ≠ FindSet(v)
        A = A ∪ {(u, v)}
        Union(u, v)
return A
```

Illustration of Kruskal's Algorithm





How Kruskal's Algorithm Works

- Lines 1-3 initialize the set A to the empty set and create $|V|$ trees, one containing each vertex.
- The for loop in Lines 5-8 examines edges in order of weight, from lowest to highest.

  The loop checks, for each edge (u, v), whether the endpoints u and v belong to the same tree.
  - If they do, then the edge (u, v) cannot be added to the forest without creating a cycle, and the edge is discarded.
  - Otherwise, the two vertices belong to different trees.
    In this case, the edge (u, v) is added to A in Line 7.
    Then, the vertices in the two trees are merged in Line 8.

Running Time of Kruskal's Algorithm

- The running time of Kruskal's algorithm for a graph $G = (V, E)$ depends on the implementation of the disjoint-set data structure.
- An efficient implementation guarantees running time $O(|E| \log |V|)$.

Subsection 3

Prim's Algorithm

Prim's Algorithm

- 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 rule adds only edges that are safe for A, whence, when the algorithm terminates, the edges in A form a minimum spanning tree.

Setting Up the Prim Procedure

- The connected graph G and the root r of the minimum spanning tree to be grown are inputs to the algorithm.
- During execution, 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 $v.\text{key}$ is the minimum weight of any edge connecting v to a vertex in the tree and $v.\text{key} = \infty$ if there is no such edge.
- The attribute $v.\pi$ names the parent of v in the tree.
- The algorithm implicitly maintains the set A from GENERICMST 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

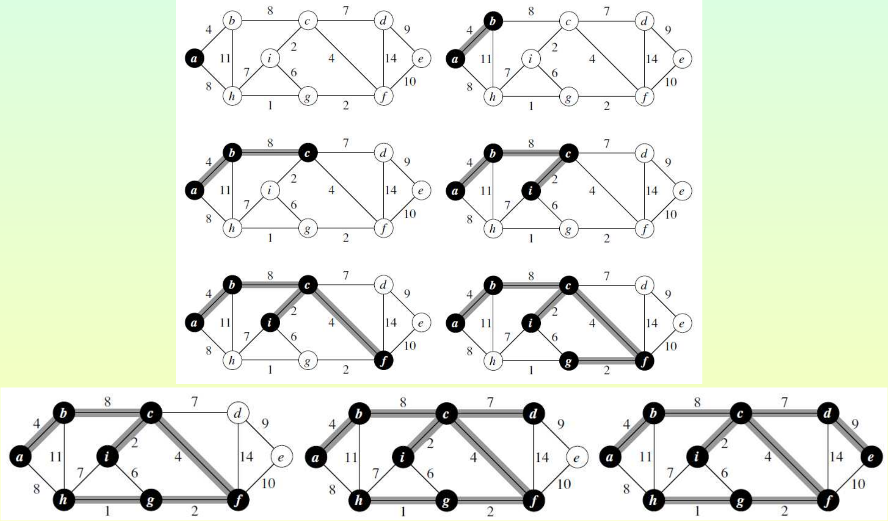
$$A = \{(v, v.\pi) : v \in V - \{r\}\}.$$

The Prim Procedure

$\text{MSTPRIM}(G, w, r)$

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

Illustration of Prim's Algorithm



How Prim's Algorithm Works

- Lines 1-5 initialize.
  - The key of each vertex is set to ∞ (except r, whose key is set to 0);
  - The parent of each vertex is set to NIL;
  - The min priority queue Q is set to contain all the vertices.
- The algorithm maintains the following three-part loop invariant:
  Prior to each iteration of the while loop of Lines 6-11:
  1. $A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$.
  2. The vertices already into the minimum spanning tree are in $V - Q$.
  3. For all vertices $v \in Q$, if $v.\pi \neq$ NIL, then v.key $< \infty$ and v.key is the weight of a light edge $(v, v.\pi)$ connecting v to some vertex already placed into the minimum spanning tree.
- Line 7 identifies a vertex $u \in Q$ incident on a light edge that crosses the cut $(V - Q, Q)$ (except in the first iteration, in which $u = r$).
- Removing u from the set Q adds it to the set $V - Q$ of vertices in the tree, thus adding $(u, u.\pi)$ to A.
- The for loop of Lines 8-11 updates the key and π attributes of every vertex v adjacent to u but not in the tree, maintaining 3.

The Running Time of Prim's Algorithm

- The running time of Prim's algorithm depends on how we implement the min priority queue Q. Suppose we implement Q as a binary min-heap.
  - We use the BUILDMINHEAP to perform Lines 1-5 in $O(|V|)$ time.
  - The body of the while loop executes $|V|$ times. Each EXTRACTMIN operation takes $O(\log|V|)$ time. Thus, the total time for all calls to EXTRACTMIN is $O(|V|\log|V|)$.
  - The for loop in Lines 8-11 executes $O(|E|)$ times altogether, since the sum of the lengths of all adjacency lists is $2|E|$.
    - Within the for loop, we can implement the test for membership in Q in Line 9 in constant time by keeping a bit for each vertex that tells whether or not it is in Q, and updating the bit when the vertex is removed from Q.
    - The assignment in Line 11 involves an implicit DECREASEKEY on the min-heap, supported in a binary min-heap in $O(\log|V|)$.

Thus, the total time is $O(|V|\log|V| + |E|\log|V|) = O(|E|\log|V|)$.