CSCE 411: Design and Analysis of Algorithms

Lecture 13: Finishing DFS; Minimum Spanning Trees

Date: March 4 *Nate Veldt, updated by Samson Zhou*

Course Logistics

- Minimum Spanning Tree: Chapter 23
- Homework 5 due Friday

1 The transpose graph and connected component graph

If $G = (V, E)$ is a graph, a *strongly connected component* is maximal subgraph $S \subseteq V$ in which every node is reachable from every other node by following paths in S.

Let $G = (V, E)$ be a graph and assume that $\{C_1, C_2, \dots, C_k\}$ represent its strongly connected components.
The *connected component graph* $G^{\text{scc}} = (V^{\text{scc}}, E^{\text{scc}})$ is defined as follows:

- There is a node $v_i \in V^{\text{scc}}$ for each component C_i
- There is an edge $(v_i, v_j) \in E^{\text{scc}}$ if and only if there is a directed edge between C_i and C_j

Lemma 1.1. *The connected component graph is* ______________________________

The *transpose graph* of G is $G^T = (V, E^T)$ where

$$E^T = \{(u, v) \colon (v, u) \in E\}$$

Lemma 1.2. *G and G^T have* ______________________________

2 Strongly Connected Components

The following algorithm will compute the strongly connected components of a graph $G = (V, E)$:

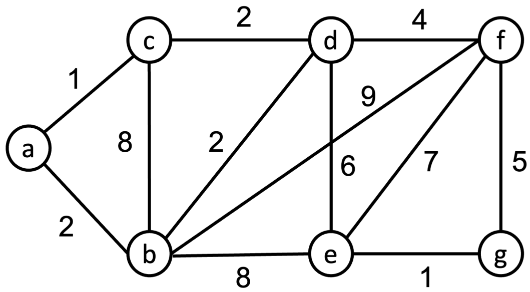
STRONGLY-CONNECTED-COMPONENTS(G)

1. Find a DFS for G to get finish times $u.F$ for each $u \in V$.
2. Compute the *transpose graph* $G^T = (V, E^T)$
3. Find a DFS for G^T, but in the main loop of DFS, always visit nodes based on the reverse order of finish times from the DFS of G.
4. Output the vertices of each tree in the DFS of G^T.

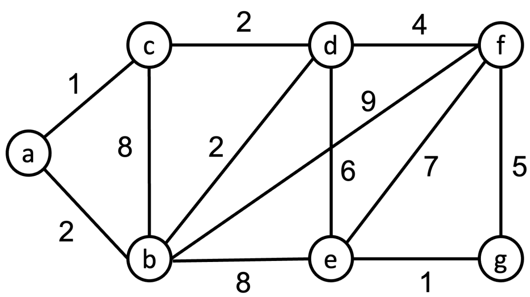
What is the key to making this work? In the second DFS, we essentially visit all of the nodes in the connected components graph in topologically sorted order.

3 Minimum Spanning Trees

Let $G = (V, E, w)$ be an undirected, connected, weighted graph where $w\colon E \to \mathbb{R}^+$ maps each edge to a nonnegative weight.

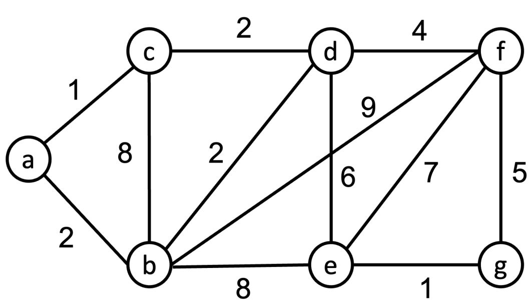


A *spanning tree* of G is a subset of edges $E_T \subseteq E$ such that $T = (V, E_T)$ ____________________



A *minimum spanning tree* of G is a spanning tree T^* that minimizes

over all spanning trees of G.



3.1 Minimum Spanning Tree Terminology

Safe Edges. Let A be a subset of edges that is guaranteed to be in some minimum spanning tree of $G = (V, E)$. An edge $(u, v) \in E$ is *safe* for A if $A \cup \{(u, v)\}$ is contained in some minimum spanning tree of G.

GenericMST(G, w)

1. Set $A = \emptyset$
2. While A is not a spanning tree
3.   find a safe edge (u, v) for A
4. $A \leftarrow A \cup \{(u, v)\}$

The key to implementing this method is *finding a safe edge* (u, v) *at each step.*

Cut terminology Let $G = (V, E)$ and A be a set of its edges.

- If $S \subseteq V$, we call the partition $(S, V - S)$ a ______

- An edge $(u, v) \in E$ ____________ the cut $(S, V - S)$ if ________________________________

- A cut ______________ A if no edge in A is cut

- An edge (u, v) is a _________________ if is has minimum weight among all cut edges.

3.2 Generic greedy strategy

We define the following *greedy* strategy for choosing a *safe* edge to add to A
GenericFindSafe(G, A, w)

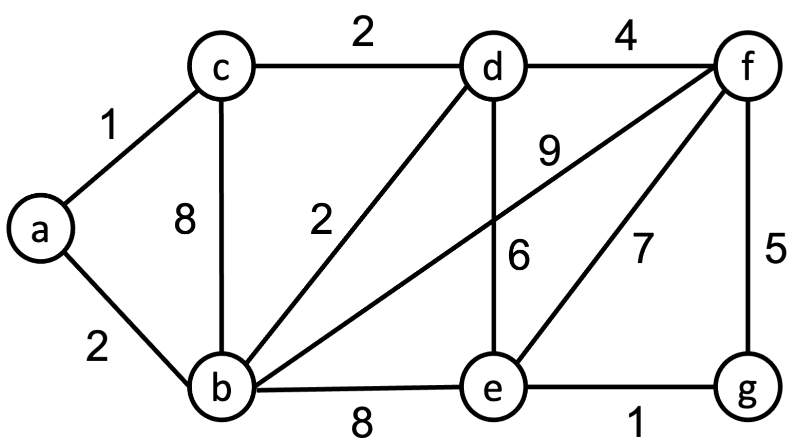
1. Let $(S, V - S)$ be a cut that *respects* A
2. Let (u, v) be a light edge in $(S, V - S)$
3. Return (u, v)

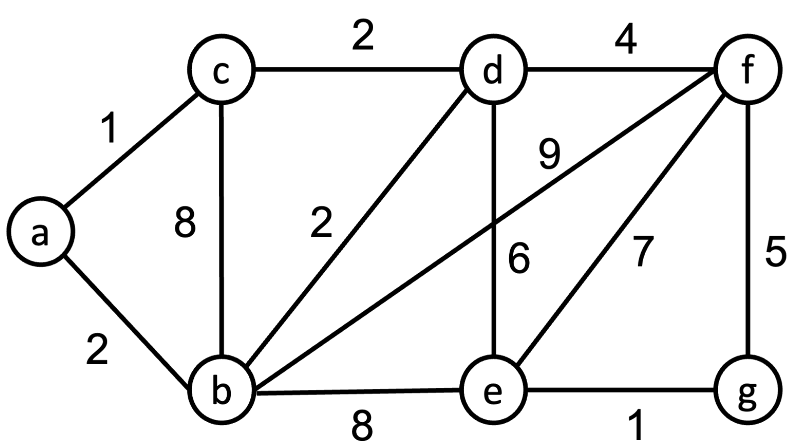
Lemma 3.1. *If A is a subset of the edges in a minimum spanning tree T of G, then* GenericFindSafe *will return a safe edge for A*

3.3 Algorithms of Kruskal and Prim

Kruskal algorithm and Prim's algorithm are two strategies for creating an MST of $G = (V, E)$.

Strategy	**What is A?**	At each step we add:
Kruskal		
Prim		





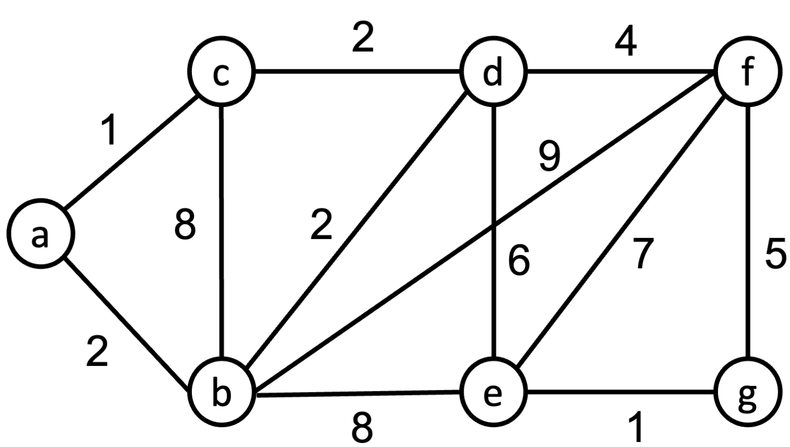
3.4 Prim's Algorithm in More Depth

During Prim's algorithm, we grow out a tree from an arbitrary starting node r. Each node has the following attributes:

- v.parent = parent node of v in the tree we are constructing
- v.key = minimum weight of an edge connecting v to the tree

We will use a min-priority queue Q to store the nodes *not in the tree* with their keys, which allows us to find the node with the smallest key in $O(\log V)$ time.

```
MST-PRIM(G, r)
  for u ∈ V do
      u.key = ∞
      u.parent = NIL
  end for
  r.key = 0
  Q = G.V
  while |Q| > 0 do
      u = EXTRACTMIN Q
      if u ≠ r then
          Add edge (u, u.parent) to A
      end if
      for v ∈ Adj[u] do
          if v ∈ Q and w(u, v) < v.key then
              v.parent = u
              v.key = w(u, v)
          end if
      end for
  end while
  Return A
```



Question 1. *What is the runtime of Prim's algorithm, knowing that it takes $O(\log V)$ time to extract the minimum element from Q or update the key of an element in Q?*

- **A** $\Theta(\log V)$
- **B** $\Theta(\log E)$
- **C** $\Theta(V \log V)$
- **D** $\Theta(E \log V)$
- **E** $\Theta(V^2)$