Graphs - MSTs **June 4, 2025**

Homework 8

Spring 2025 *COSC 31: Algorithms*

1. **Understanding MSTs**

(a) Let (u, v) be a minimum-weight edge in a connected graph G. Explain how the Greedy Choice Lemma that we proved in class implies that (u, v) belongs to some minimum spanning tree of G.

Answer

Let (u, v) be a minimum-weight edge in the connected graph G. Consider the cut $(S, V \setminus S)$ where $S = \{u\}$ and $v \in V \setminus S$. Since G is connected, such a cut exists and (u, v) crosses it. Because (u, v) has the smallest weight among all edges in G, it is the lightest edge crossing this cut. By the Greedy Choice Lemma, the lightest edge crossing any cut belongs to some minimum spanning tree of G. Thus, (u, v) is included in some MST of G.

(b) Give a simple example of a connected graph such that the set of edges

$$\{(u, v) \in E| \text{ there exists a cut } (S, V - S) \text{ such that } (u, v) \text{ is a light edge crossing } (S, V - S)\}$$

does not form a minimum spanning tree

Answer

Consider the graph G with vertices $V = \{a, b, c\}$ and edges $E = \{(a, b), (b, c), (a, c)\}$ with weights:

$$w(a, b) = 2, \quad w(b, c) = 2, \quad w(a, c) = 2.$$

In this graph, every edge is a light edge crossing the cut $(S, V - S)$ for any non-empty subset S of V. However, the set of edges $\{(a, b), (b, c), (a, c)\}$ does not form a minimum spanning tree because it contains all edges hence forming a cycle. The minimum spanning tree of this graph can be formed by any two edges, for example, $\{(a, b), (b, c)\}$ or $\{(a, c), (b, c)\}$.

(c) Prove that a graph has a unique minimum spanning tree if, for every cut of the graph, there is a unique light edge crossing the cut.

Answer

Suppose, for contradiction, that $G = (V, E)$ has two distinct MSTs, T_1 and T_2. Since $T_1 \notin T_2$, $\exists(u, v) \in E$ such that $(u, v) \in T_1$ but $(u, v) \notin T_2$. Consider the cut $(S, V \setminus S)$ of T_1 where $u \in S$ and $v \in V \setminus S$. Since (u, v) is in T_1, it must cross this cut. By hypothesis, this cut has a unique light edge, and since (u, v) is in T_1, it must be that unique light edge. But T_2, not containing (u, v), must include a different edge (x, y) crossing the same cut. This contradicts the uniqueness of the light edge. Therefore, no such T_2 can exist, and the MST must be unique.

(d) Show that the converse is not true by giving an example of a graph that has a unique MST even though some cut does not have a unique light edge.

Answer

Consider the graph G with vertices $V = \{a, b, c\}$ and edges:

$$w(a,b) = 1, \quad w(a,c) = 1, \quad w(b,c) = 2.$$

Apply Kruskal's Algorithm:

- Pick (a, b) — weight 1
- Pick (a, c) — weight 1
- Edge (b, c) is skipped (would create a cycle)

So the unique MST is $\{(a, b), (a, c)\}$.
Now consider the cut $(\{b, c\}, \{a\})$. Both edges (a, b) and (a, c) cross this cut and have the same minimum weight of 1.
Thus, this cut has two lightest edges, but the MST is unique. Therefore, the converse is false.

2. **Borůvka's Algorithm**

(a) An English explanation of Borůvka's Algorithm

Answer

Given an undirected, connected graph $G = (V, E)$, Borůvka's Algorithm finds a minimum spanning tree (MST) by progressively merging components using the lightest outgoing edges. The algorithm proceeds in phases, each consisting of the following steps:

(a) For each component (initially, each vertex is its own component), find the lightest edge connecting it to a different component.

(b) Add all such lightest edges to the MST. This merges some components together.

(c) Repeat the process on the new set of components until only one component remains — the MST.

(b) Pseudocode for Borůvka's Algorithm

Answer

Algorithm 1 Borůvka's Algorithm

```
Function Boruvka($G = (V, E)$):
    $T \leftarrow \emptyset$
    $components \leftarrow$ MakeSet($V$)

    while $numComponents(components) > 1$ do
        $lightestEdges \leftarrow \emptyset$
        foreach $C \in components$ do
            $minEdge \leftarrow$ null
            foreach $e = (u, v) \in E$ where $u \in C,\ v \notin C$ do
                if $minEdge = $ null or $w(e) < w(minEdge)$ then
                    $minEdge \leftarrow e$
            if $minEdge \neq$ null then
                $lightestEdges \leftarrow lightestEdges \cup \{minEdge\}$
        foreach $e \in lightestEdges$ do
            if $e$ connects two different components then
                $T \leftarrow T \cup \{e\}$
                Union(components, endpoints of $e$)
    return $T$
```

(c) A proof of correctness for Borůvka's Algorithm

Answer

We prove Borůvka's Algorithm is correct by showing that it always produces a minimum spanning tree (MST) of the input graph $G = (V, E)$. The key invariant maintained throughout the algorithm is:

Each edge added to the tree is a light edge crossing some cut of the current components.

Cut Property: In any connected, undirected, weighted graph, the lightest edge crossing any cut is guaranteed to be part of some MST.

- In each iteration, Borůvka's algorithm identifies the lightest edge connecting each component to another.
- Each such edge crosses a cut between that component and the rest of the graph.
- By the cut property, each edge is safe to add to the MST.

Hence, all edges added are part of some MST. Since edges are only added between distinct components, cycles are never introduced. If we merge two trees of sizes n_1 and n_2, each with $n_1 - 1$ and $n_2 - 1$ edges respectively, and add one edge between them, the total becomes:

$$(n_1 - 1) + (n_2 - 1) + 1 = n_1 + n_2 - 1,$$

which is exactly the number of edges in a tree on $n_1 + n_2$ vertices.

Because the algorithm keeps merging trees without creating cycles and continues until all vertices are connected, the final result is a spanning tree. Since all added edges are safe (by the cut property), this tree is a valid MST.

(d) The time complexity of Borůvka's Algorithm, along with an explanation of how that time complexity is derived.

Answer

The time complexity of Borůvka's Algorithm is $O(m \log n)$, where $m = |E|$ is the number of edges and $n = |V|$ is the number of vertices.

- **Finding the lightest edge for each component:** In each phase, we scan all edges to find the lightest outgoing edge from each component. This takes $O(m)$ time.
- **Merging components:** Each added edge connects two distinct components, reducing the number of components. Since the number of components at least halves in each phase, there are at most $O(\log n)$ phases.
- **Total work:** Each phase costs $O(m)$ and there are $O(\log n)$ phases, leading to a total time complexity of $O(m \log n)$.

Thus, the overall time complexity of Borůvka's Algorithm is $O(m \log n)$.

3. **A Reflection on Algorithms**

1. Importance of the foundations of algorithms and theory of computation
   I believe the foundations of algorithms are essential for understanding efficient problem-solving techniques, and they are especially necessary when building complex systems that process large data sets or serve many users concurrently. While I don't have much experience with the theory of computation—largely due to the pivots I've made in my educational path at Dartmouth—I recognize its value. It provides the theoretical underpinnings for what can be computed and how efficiently. This is crucial for designing algorithms that are not only correct but also optimal in terms of time and space complexity. Overall, I believe algorithms help automate redundant tasks and enable us to focus on more complex and creative aspects of problem-solving.
2. The importance of algorithms in your everyday life.
   Algorithms play a significant role in my everyday life, even if I don't always recognize them. From the way my smartphone organizes photos to how social media platforms recommend content, algorithms operate behind the scenes. They help me find information quickly, automate routine tasks, and even optimize daily activities.

   In my academic work, algorithms help me analyze complex biological data—such as tracking pixel intensity changes in fluorescence microscopy images—to understand how cells take up imaging agents. This process involves a combination of physics-based models, biological computations, and image processing algorithms. Together, they enable deeper insights that would be difficult to achieve manually.
3. The (expected) importance of algorithms in your career
   In the near future, I will be working as a software engineer, and a strong understanding of algorithms—especially legacy and foundational ones—will be essential. Algorithms will be fundamental to writing efficient code, debugging existing systems, and designing scalable solutions. Whether optimizing some infrastructure or handling large-scale data, algorithmic thinking will be central to my career.
4. The (expected) importance of algorithms in your career
   Personally, I find mathematical reasoning challenging at first, but I've come to appreciate how algorithms bring structure to problem-solving. As I prepare for a career in software engineering, I expect algorithms to be central to my work—whether it's improving performance, ensuring scalability, or understanding how systems behave under the hood. Even when the math is tough, it's rewarding to see how those principles power real-world applications.