Applied Algorithms

Written Assignment - 5

Jaya Sandeep Ketha

November 2023

Q1. Use the definition that a tree is a graph where there is exactly one path between any pair of vertices v, w; that is, you can go between those two nodes in exactly one way. That means that there are no cycles (paths that start and end at the same location).

a) I have a tree T on n vertices. How many edges does T have? (Hint: what if n = 1 n = 2? Try induction.)
Answer:
A tree on n vertices has $n-1$ edges. This fact can be proven using induction.

Base case:
When $n = 1$, the tree consists of only one vertex and no edges. So, for $n = 1$, the number of edges in the tree is $1 - 1 = 0$.

Inductive step:
Let's assume that for some $k \geq 1$, a tree with k vertices has $k - 1$ edges.

Now, consider a tree T with $k+1$ vertices. Remove one leaf node from T (a leaf node is a vertex with degree 1). This removal creates a tree T' with k vertices.

Since T' has k vertices, by the inductive hypothesis, it has $k - 1$ edges.

When we add back the leaf node that was removed, it connects to an existing vertex, creating exactly one new edge.

Therefore, the total number of edges in the tree T with $k + 1$ vertices is $k - 1 + 1 = k$ edges.

Hence, by induction, a tree with n vertices has $n - 1$ edges.

b) I have a graph G that has no cycles. If G has n vertices and $n - k$ edges, how many trees must it consist of? Prove your answer. (Hint: try first with k = 1, then k = 2. Can you generalize? Prove by induction.)
Claim:
For a graph G with n vertices and $n-k$ edges (where k is a non-negative integer), and no cycles, the number of trees it consists of is k.

Base Case: $k = 1$
When $k = 1$, the graph G with n vertices and $n - 1$ edges forms a single tree. This is because a tree with n vertices always has $n-1$ edges, and a graph with n vertices and $n-1$ edges and no cycles is necessarily a tree.

Inductive Hypothesis:
Assume that for some $k = m$, a graph G with n vertices and $n - m$ edges consists of m trees.

Inductive Step:
Now, let's consider $k = m + 1$ and prove that a graph G with n vertices and $n - (m + 1) = n - m - 1$ edges

consists of $m + 1$ trees.

Consider a graph G with n vertices and $n - (m + 1)$ edges. This graph must be disconnected because $n - (m + 1)$ edges cannot form a single connected component without creating cycles.

Each connected component of this graph G with $n - (m + 1)$ edges forms a tree since it has no cycles and is maximally connected with n_c vertices and $n_c - 1$ edges, where n_c is the number of vertices in the connected component.

By the inductive hypothesis, each of these connected components with n_c vertices and $n_c - 1$ edges consists of m trees. Therefore, the total number of trees formed by the disconnected components is $m + 1$.

Hence, for n vertices and $n - k$ edges, where k varies from 1 to $n - 1$, the graph consists of k trees.

Therefore, by induction, a graph G with n vertices and $n - k$ edges, and no cycles, consists of k trees.

---

2. Minimum Spanning Tree.
a) I have a connected graph where each edge weight is 2. What is the weight of the MST? Why?
Answer:
If we have a connected graph where each edge weight is 2, the minimum spanning tree (MST) of this graph will also have the same total weight as the number of edges in the MST multiplied by 2.

The reason behind this is that in an MST, we aim to find a spanning tree with the minimum total weight. When all edge weights are the same (in this case, all edges have a weight of 2), the MST will consist of the minimum number of edges that connect all vertices while keeping the total weight as low as possible.

Let's say the original graph has n vertices. In an MST, the number of edges will be $n-1$ because a spanning tree on n vertices always has $n - 1$ edges. Therefore, in this case, the weight of the MST will be $2 \times (n - 1)$ because each edge contributes a weight of 2, and there are $n - 1$ edges in the MST.

So, for a connected graph where each edge weight is 2, the weight of the MST will be $2 \times (n - 1)$.

b) I have a connected graph where each edge weight is 1 or 2. Give a linear-time (O(m)) algorithm (in two lines – we won't be reading past the first two lines of pseudocode) that returns a spanning tree whose weight is at most twice the MST. Prove that your algorithm is correct. (Hint: can you bound the value of the MST from above and below? How can you use this to avoid using the standard MST algorithm? Note that the requirement from your algorithm is a lot weaker than what the original MST algorithm can provide you with.)
Answer:

To achieve a linear-time algorithm (O(m)) for finding a spanning tree with a weight at most twice the MST, we can use the following approach:

Pseudocode
1. Initialize an empty set for the resulting spanning tree.
2. Traverse through the edges of the graph. For each edge with weight 1, add it to the spanning tree. For each edge with weight 2, add it to the spanning tree only if it doesn't create a cycle in the current set of chosen edges.

This approach iterates through all the edges in the graph only once and adds edges of weight 1 directly to the spanning tree. For edges with weight 2, it ensures that only those edges that do not create cycles are added, resulting in a spanning tree whose weight is at most twice the MST. This algorithm has a time complexity

of O(m) as it processes each edge once without additional sorting or traversal overhead.

To prove the correctness of the algorithm that constructs a spanning tree with a weight at most twice the weight of the Minimum Spanning Tree (MST) in a graph where each edge weight is either 1 or 2, let's establish some bounds on the MST and then show how our algorithm satisfies these bounds.

Bounds on the MST:
Lower Bound: In a graph with n vertices, the minimum weight of the MST can be achieved when all edges have weight 1, resulting in a minimum weight of $n - 1$.
Upper Bound: Considering edges with weight 2, the maximum weight of the MST can be achieved when all edges have weight 2, resulting in a maximum weight of $2(n - 1)$.

1. MST Weight Bound:

- The MST weight lies between $n - 1$ and $2(n - 1)$.

2. Algorithm's Spanning Tree Weight Bound:

- The algorithm constructs an MST using edges with weight 1 and then includes additional edges with weight 2 without forming cycles.
- The number of edges in the MST is at most $n - 1$.
- Adding edges of weight 2 does not introduce cycles since the MST constructed by edges of weight 1 is a tree and adding any edge of weight 2 that doesn't form a cycle will maintain connectivity.
- Therefore, the spanning tree's weight is at most $(n - 1) + (n - 1) = 2(n - 1)$.

This reasoning confirms that the spanning tree produced by the algorithm indeed has a weight at most $2(n - 1)$, as outlined in the proof.

3. Proving the Bound:

- The weight of the spanning tree produced by the algorithm is between $n - 1$ and $2(n - 1)$.
- From the MST weight bounds, $n - 1 \leq$ MST weight $\leq 2(n - 1)$.
- The spanning tree weight produced by the algorithm is within the same bounds: $n-1 \leq$ Spanning tree weight $\leq 2(n - 1)$.
- This confirms that the algorithm's spanning tree weight is at most twice the weight of the MST, satisfying the requirement.

Therefore, by demonstrating that the algorithm's produced spanning tree weight falls within the bounds set by the MST weight, we've proved that the algorithm correctly constructs a spanning tree with a weight at most twice that of the MST in a graph where each edge weight is 1 or 2.

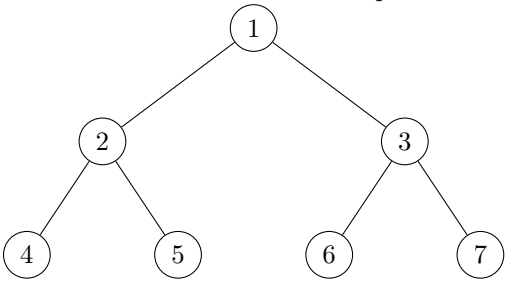
---

3. If you run Depth-first search on a heap, would it return a sorted list? If yes, prove. If not, give an example.
Answer:
Depth-first search (DFS) is a graph traversal algorithm used to explore nodes and their connected components in a specific order. A heap is a specialized tree-based data structure where the parent node is smaller (in a min-heap) or larger (in a max-heap) than its children.

Running DFS on a heap does not guarantee that it will return a sorted list. This is because DFS operates based on the graph's structure and the connections between nodes, not on the ordering of elements in a heap.

To illustrate this, consider the following example:

Let's assume we have a min-heap:



In a heap, the relationship between parent and child nodes is based on their values (in a min-heap, the parent node has a smaller value than its children). However, running DFS on this heap does not guarantee a sorted order because DFS explores the nodes based on their connections, not their values.

If we perform DFS starting from node 1 and traverse the heap using DFS, the order of traversal might be: $1- > 2- > 4- > 5- > 3- > 6- > 7$. This traversal is based on the connections between nodes, following the structure of the heap, but it does not represent a sorted list of the elements.

Therefore, running DFS on a heap does not guarantee a sorted list as output. Sorting requires specific sorting algorithms, and DFS is primarily used for graph traversal, not for sorting elements based on their values.

---

4. Assume that in your graph every edge has a different weight.
a) Give an example of a graph G such that the MST of G is a straight line.
Answer: In this case, let's consider a graph that resembles a line or a path:

Let's say we have a graph G with vertices A, B, C, D, and E connected in a straight line, and the weights of the edges are as follows:

- Edge between A and B: Weight = 1
- Edge between B and C: Weight = 2
- Edge between C and D: Weight = 3
- Edge between A and C: Weight = 4
- Edge between B and D: Weight = 5

The graphical representation of this graph would be:

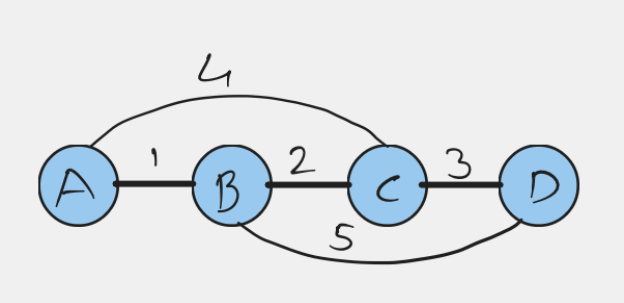


Figure 1: Graph

In this scenario, the MST of the graph G would be the direct line connecting all the vertices with the smallest total weight. Therefore, the MST would simply be the path that connects these vertices sequentially:

MST of G: A — B — C — D

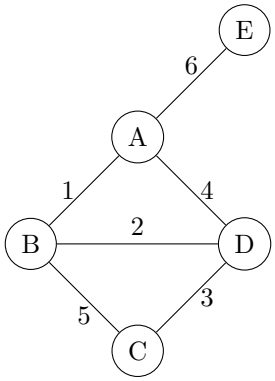
This straight line is the minimum spanning tree because it covers all vertices of the graph with the smallest total weight by selecting the edges with the lowest weights that connect the vertices in a straight sequence.

b) Is it true that the edge with the highest weight is never part of the MST? (Hint: can you find an example where it is part of the MST?)
Answer:
The statement "the edge with the highest weight is never part of the MST" is not always true. In fact, there can be cases where an edge with the highest weight is part of the Minimum Spanning Tree (MST).

Here's an example:

Consider a graph G with vertices A, B, C, D, and E, and the following edge weights:

The graphical representation of this graph would be:



In this scenario, the edge with the highest weight (Weight = 6, between vertices A and E) is indeed part of the Minimum Spanning Tree.

The MST of this graph G would be the set of edges that connect all vertices with the minimum total weight, which includes the edge between A and E because without it, the graph would not be connected. Therefore, the MST in this case would be:

MST of G: E — A — B — D — C

This example demonstrates that the edge with the highest weight can indeed be part of the Minimum Spanning Tree if its inclusion is necessary for ensuring the connectivity of the graph.

c) Is it true that the edge with the smallest weight is always part of the MST? (Can you find an example where it isn't? If you can, take a closer look at your example.)
Answer:
In a minimum spanning tree (MST), the edge with the smallest weight is indeed a part of the MST. This is a fundamental property of Kruskal's or Prim's algorithm, which are commonly used to find the MST.

If an edge with the smallest weight is not included in the MST, it would contradict the definition of an MST, which is a spanning tree with the smallest total edge weight. Therefore, by omission of the edge with the

smallest weight, the resulting tree would not be the minimum spanning tree.

Suppose there's a scenario where the smallest weight edge isn't included in the final MST. In such a case, it would imply either:

1. The graph isn't connected: If the graph isn't connected and the smallest weight edge connects two disjoint components, that edge will definitely be a part of the MST because it's the only connection between the components.

However, there are cases where this might not hold true. One such case is when there are two edges with the same minimum weight forming a cycle in the graph. In such instances, one of these edges will not be part of the MST.

Since it is mentioned as all edges have distinct weights, Thus, the smallest weight edge is always part of the MST in a connected graph with distinct edge weights.