CSCI 3104 Fall 2022
Instructors: Prof. Grochow and Candra Kanth Nagesh

Problem Set 1

Due Date ..... September 5, 2022
Name ..... **Tyler Huynh**
Student ID **109603994**
Collaborators ..... **No Collaborators**

Contents

Instructions

- The solutions **should be typed**, using proper mathematical notation. We cannot accept hand-written solutions. Here's a short intro to LaTeX.
- You should submit your work through the **class Gradescope page** only (linked from Canvas). Please submit one PDF file, compiled using this LaTeX template.
- You may not need a full page for your solutions; pagebreaks are there to help Gradescope automatically find where each problem is. Even if you do not attempt every problem, please submit this document with no fewer pages than the blank template (or Gradescope has issues with it).
- You are welcome and encouraged to collaborate with your classmates, as well as consult outside resources. You must **cite your sources in this document. Copying from any source is an Honor Code violation. Furthermore, all submissions must be in your own words and reflect your understanding of the material.** If there is any confusion about this policy, it is your responsibility to clarify before the due date.

- Posting to **any** service including, but not limited to Chegg, Reddit, StackExchange, etc., for help on an assignment is a violation of the Honor Code.
- You **must** virtually sign the Honor Code (see Section). Failure to do so will result in your assignment not being graded.

Honor Code (Make Sure to Virtually Sign)

Problem HC.
- My submission is in my own words and reflects my understanding of the material.
- Any collaborations and external sources have been clearly cited in this document.
- I have not posted to external services including, but not limited to Chegg, Reddit, StackExchange, etc.
- I have neither copied nor provided others solutions they can copy.

I agree to the above, Tyler Huynh. □

1 Standard 1: Proof by Induction

1.1 Problem 1

Problem 1. A student is trying to prove by induction that $2^n < n!$ for $n \geq 4$.

Student's Proof. The proof is by induction on $n \geq 4$.

- **Base Case:** When $n = 4$, we have that:

$$\begin{aligned} 2^4 &= 16 \\ &\leq 24 \\ &= 4! \end{aligned}$$

- **Inductive Hypothesis:** Now suppose that for all $k \geq 6$ we have that $2^k < k!$.
- **Inductive Step:** We now consider the $k+1$ case. As $k+1 > 6$, we have from the inductive hypothesis that $2^{k+1} < (k+1)!$. The result follows by induction.

□

There are two errors in this proof.

(a) The Inductive Hypothesis is not correct. Write an explanation to the student explaining why their Inductive Hypothesis is not correct. [**Note:** You are being asked to explain why the Inductive Hypothesis is wrong, and **not** to rewrite a corrected Inductive Hypothesis.]

Answer. Their inductive hypotheses is not correct because they assume that all values of k ≥ 6, such that 2^k $<$ k!. This is not correct because they cannot assume that for the values of k that are greater than or equal to 6 it will prove their hypothesis as being correct, as they are assuming what they are trying to prove. □

(b) The Inductive Step is not correct. Write an explanation to the student explaining why their Inductive Step is not correct. [**Note:** You are being asked to explain why the Inductive Step is wrong, and **not** to rewrite a corrected Inductive Step.]

Answer. The inductive step is not correct because the student is assuming that $k + 1 > 6$, such that $2^k + 1$ $< (k + 1)!$. This is incorrect due to the fact that they don't provide any steps on how to prove that they reached this conclusion.This concludes that their inductive hypothesis is incorrect leading to their inductive step also being incorrect. □

1.2 Problem 2

Problem 2. Consider the recurrence relation, defined as follows:

$$T_n = \begin{cases} 2 & : n = 0, \\ 22 & : n = 1, \\ -2T_{n-1} + 35T_{n-2} & : n \geq 2. \end{cases}$$

Prove by induction that $T_n = (-1) \cdot (-7)^n + 3 \cdot (5)^n$, for all integers $n \in \mathbb{N}$. [**Recall:** $\mathbb{N} = \{0, 1, 2, \ldots\}$ is the set of non-negative integers.]

Proof. **Proof by Strong Induction.**

Base Case:
When
$n = 0$ then $T_0 = 2$, we have that:
$T_0 = (-1) * (-7)^0 + 3 * (5)^0$
= -1 + 3
$2 = 2$
$n = 1$ then $T_1 = 22$, we have that:
$T_1 = (-1) * (-7)^1 + 3 * (5)^1$
= 7+15
$22 = 22$

Inductive Hypothesis:

For n ≥ 2, such that $k < n$ $T_k = (-1) * (-7)^k + 3 * (5)^k$, such that our recurrence relation:
$-2T_{k-1} + 35T_{k-2} = (-1) * (-7)^k + 3 * (5)^k$, will stand true.

Inductive Step:
We know that by the inductive hypothesis, that $k < n$, such that:
$n - 1 < n$ and $n - 2 < n$
Plugging this into T_n:
$T_{n-1} = (-1) * (-7)^{n-1} + 3 * (5)^{n-1}$
$T_{n-2} = (-1) * (-7)^{n-2} + 3 * (5)^{n-2}$
Further by our inductive hypothesis we know that:
$T_n = -2T_{k-1} + 35T_{k-2}$
So plugging in previously:
$T_n = -2(-1 * (-7)^{n-1} + 3 * (5)^{n-1}) + 35(-1 * (-7)^{n-2} + 3 * (5)^{n-2})$
Solving for this:
$T_n = ((2 * (-7)^{n-1} - 6 * (5)^{n-1}) + (-35 * (-7)^{n-2} + 105 * (5)^{n-2}))$
$T_n = ((-\frac{2}{7}(-7)^n + -\frac{6}{5}(5)^n) + (-\frac{35}{49}(-7)^n + \frac{105}{25}(5)^n))$
$T_n = ((-\frac{2}{7}(-7)^n + -\frac{6}{5}(5)^n) + (-\frac{5}{7}(-7)^n + \frac{21}{5}(5)^n))$
$T_n = ((-\frac{2}{7}(-7)^n - \frac{5}{7}(-7)^n) + (-\frac{6}{5}(5)^n + \frac{21}{5}(5)^n))$
$T_n = ((-1(-7)^n + 3(5)^n))$
Thus, as you can see from the above work, we can see that that $T_n = (-1) * (-7)^n + 3 * (5)^n$, for all integers $n \in \mathbb{N}$ has been proved by using strong induction. □

1.3 Problem 3

Problem 3. The complete, balanced 3-ary tree of depth d, denoted $\mathcal{T}(d)$, is defined as follows.

- $\mathcal{T}(0)$ consists of a single vertex.
- For $d > 0$, $\mathcal{T}(d)$ is obtained by starting with a single vertex and setting each of its three children to be copies of $\mathcal{T}(d-1)$.

Prove by induction that $\mathcal{T}(d)$ has 3^d leaf nodes. To help clarify the definition of $\mathcal{T}(d)$, illustrations of $\mathcal{T}(0), \mathcal{T}(1)$, and $\mathcal{T}(2)$ are on the next page. [**Note:** $\mathcal{T}(d)$ is a tree and **not** the number of leaves on the tree. Avoid writing $\mathcal{T}(d) = 3^d$, as these data types are incomparable: a tree is not a number.]

Proof. **Proof by Strong Induction**
Base Case:
T(0) has only one leaf node
$T(0) = 3^0$
$= 1$

Inductive Hypothesis:
For any k < d, $T(k)$ has 3^k leaf nodes.

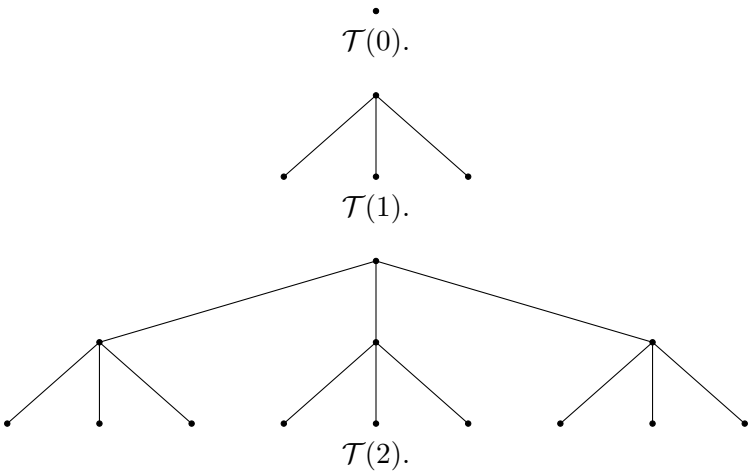
Inductive Step:
Since by the inductive hypothesis $k < d$, $T(k)$ has 3^k leaf nodes:

We will now prove for all trees on $k+1$ leaf nodes:
By the definition of the tree $T(k)$, we can see that for any non-negative integer value of k that the leaf nodes of the next depth $T(k+1)$, will be 3 times the leaf nodes of $T(k)$
$T(k+1)$ has $(3^k)(3)$ leaf nodes,
thus, 3^{k+1} leaf nodes

Take for example:
Let $k = 1$, such that:
$T((1)+1)$ has $(3^1)(3)$
$T(2)$ has 9 leaf nodes

By proving that $T(k+1) = 3^{k+1}$ for $k < d$ will have 3 leaf nodes, which then satisfies the original proof of T(d) will have 3^d leaf nodes. □

Example 1. We have the following:



$\mathcal{T}(0)$.

$\mathcal{T}(1)$.

$\mathcal{T}(2)$.

2 Standard 2: BFS and DFS

2.4 Problem 4

Problem 4. Consider the Connectivity problem:

- Instance: Let $G(V, E)$ be a simple, undirected graph. Let $u, v \in V(G)$.
- Decision: Is there a path from u to v in G?

Do the following. [**Note:** There are parts (a) and (b). Part (b) is on the next page.]

(a) Design an algorithm to solve the Connectivity problem. Your solution should provide enough detail that a CSCI 2270 student could reasonably be expected to implement your solution.

Answer for Part (a). Using BFS
An algorithm to solve the connectivity problem would use BFS to solve it, as it would give the shortest paths, thus confirming if a path were to exist between u and v. Where BFS searches a graph by level it would visit every vertex on that level first. From that level it will then traverse to the next level.

Pseudocode and comments:

```
isThereConnection(u, v, G)
init visited = [false,... false]; -- Initializing an array to make every node in the array marked
    as unvisited
init Q; -- Initializing a queue since BFS uses a queue data structure
Q.push(u); -- Pushing or enqueuing vertex u
visited[u] = true; -- Marking u as visited after we push it to our queue
init isPath = false; -- Creating a boolean variable to mark vertex's as visited, such that if we
    find v then a path exists

while(Q.is not empty){, -- Creating a while loop to traverse through the graph G
   x = Q.pop; -- Popping off the vertex so we mark it as visited
   if(x == vertex v){, -- Checking vertex X to see if it is vertex V
      isPath = true; -- If it is vertex v than mark it as true, such that a path exists between u
          and v
   }
   for (each neighbor vertex){, -- For loop to traverse through the adjacent vertexes of x
      if (neighbor vertex has not been visited){, -- If the neighbor vertex has not been visited
          Q.push(adjacent vertex); -- Pushing onto the queue the neighbor vertex to be visited
          visited[x] = true; -- Marking the vertex visited as true
      }
   }
}
return isPath; -- Returning the boolean variable such that a path between u and v exists
```

□

(b) We say that the graph G is *connected* if for every pair of vertices $u, v \in V(G)$, there exists a path from u to v. Design an algorithm to determine whether G is connected. Your algorithm should only traverse the graph once- this means that you should **not** apply BFS or DFS more than once. Your solution should provide enough detail that a CSCI 2270 student could reasonably be expected to implement your solution.

Answer for Part (b). Pseudocode and comments:

```
isThereConnection(u, v, G){
init visited = [false,... false]; -- Initializing an array to make every node in the array marked
    as unvisited
init Q; -- Initializing a queue since BFS uses a queue data structure
Q.push(u); -- Pushing or enqueuing vertex u
visited[u] = true; -- Marking u as visited after we push it to our queue
init isPath = false; -- Creating a boolean variable to mark vertex's as visited, such that if we
    find v then a path exists

while(Q.is not empty){, -- Creating a while loop to traverse through the graph G
   x = Q.pop; -- Popping off the vertex so we mark it as visited
     for (each neighbor vertex){, -- For loop to traverse through the adjacent vertexes of x
      if (neighbor vertex has not been visited){, -- If the neighbor vertex has not been visited
          Q.push(adjacent vertex); -- Pushing onto the queue the neighbor vertex to be visited
          visited[x] = true; -- Marking the vertex visited as true
          }
     }
}
for(y in visited){, -- Creating another for loop to traverse through the visited array to see if
    we have visited every node
    if(y == false){, -- If we have not visited every node
      return false; -- Then the graph of G is not connected
    }
  }
}
return true; -- Returning true if the graph of G is connected
```

□

2.5 Problem 5

Problem 5. Give an example of a simple, undirected, and unweighted graph $G(V, E)$ that has a single source shortest path tree which a **breadth-first traversal** will not return for any ordering of its vertices. Your answer must

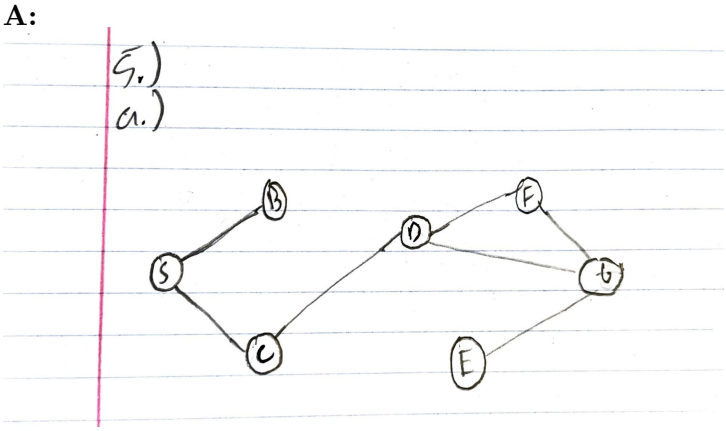
(a) Provide a drawing of the graph G. [**Note:** We have provided TikZ code below if you wish to use LaTeX to draw the graph. Alternatively, you may hand-draw G and embed it as an image below, provided that (i) your drawing is legible and (ii) we do not have to rotate our screens to grade your work.]

(b) Specify the single source shortest path tree $T = (V, E_T)$ by specifying E_T and also specifying the root $s \in V$. [**Note:** You may again hand-draw this tree. If you wish, you may clearly mark the edges of T on your drawing of G. Please make it easy on the graders to identify the edges of T.]

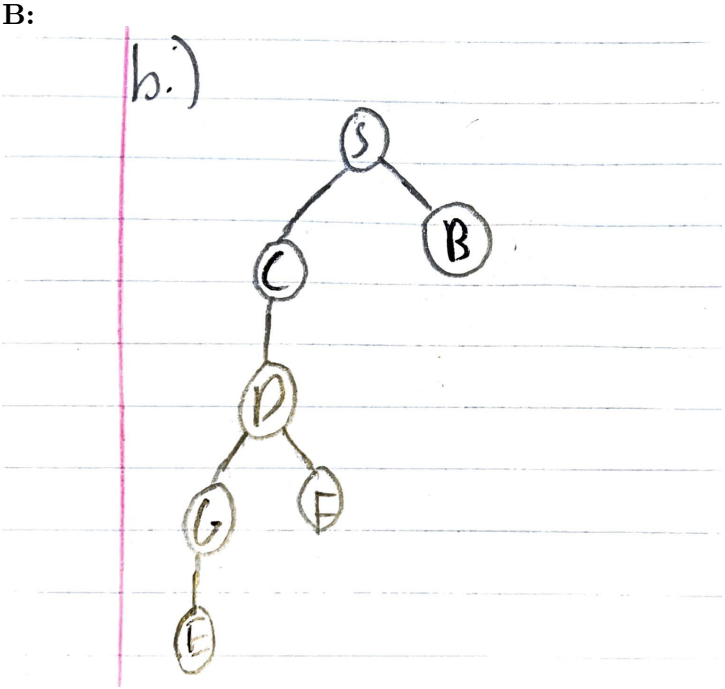
(c) Include a clear explanation of why the breadth-first search algorithm we discussed in class will never produce T for any orderings of the vertices.

Answer. BFS Graph

A:



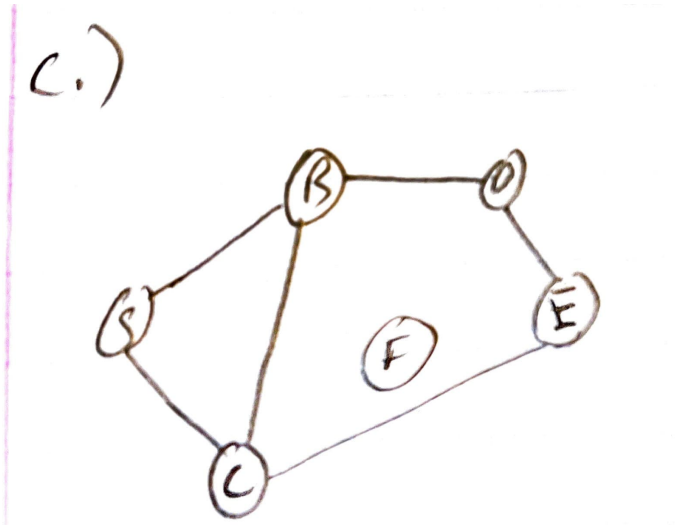
B:



C:

The breadth-first-search algorithm that we had from class will never produce T for any ordering of the vertices because the BFS algorithm in order for it to work, the graph must be connected for all vertices regardless of the ordering. Take for example:

c.)



Within this graph we can see that the vertex F will never be reached from the BFS algorithm because F is not connected to the rest of the graph. If we were to create a SSSPT from the BFS algorithm starting from vertex s, F would never be included within that traversal. This violates the definition of the SSSPT, as by definition, the SSSPT must include all vertices V, where V is all the vertices in graph G, regardless of ordering. Thus, the BFS algorithm from class in this case will never be able to reproduce T for any orderings of vertices because it will never visit a vertex that is not connected to the graph.

□

2.6 Problem 6

Problem 6. Give an example of a simple, undirected, weighted graph such that a breadth-first traversal outputs a search-tree that is not a single source shortest path tree. (That is, BFS is not sufficiently powerful to solve the shortest-path problem on weighted graphs. This motivates Dijkstra's algorithm, which will be discussed in the near future.) Your answer must

(a) Draw the graph $G = (V, E, w)$ by specifying V and E, clearly labeling the edge weights. [**Note:** We have provided TikZ code below if you wish to use LaTeX to draw the graph. Alternatively, you may hand-draw G and embed it as an image below, provided that (i) your drawing is legible and (ii) we do not have to rotate our screens to grade your work.]

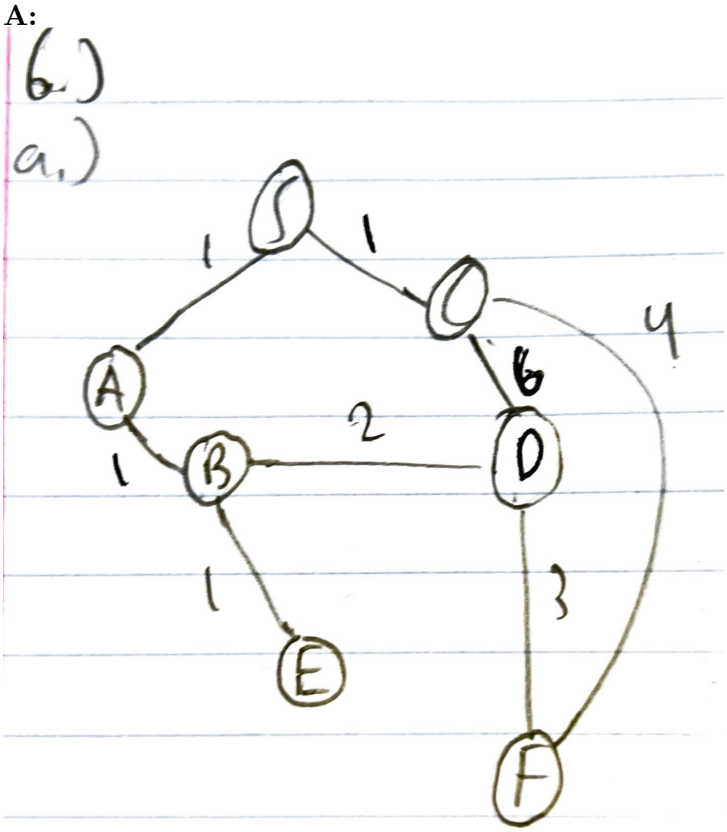
(b) Specify a spanning tree $T(V, E_T)$ that is returned by BFS, but is not a single-source shortest path tree. [**Note:** You may again hand-draw this tree. If you wish, you may clearly mark the edges of T on your drawing of G. Please make it easy on the graders to identify the edges of T.]

(c) Specify a valid single-source shortest path tree $T' = (V, E_{T'})$. [**Note:** You may again hand-draw this tree. If you wish, you may clearly mark the edges of T on your drawing of G. Please make it easy on the graders to identify the edges of T.]

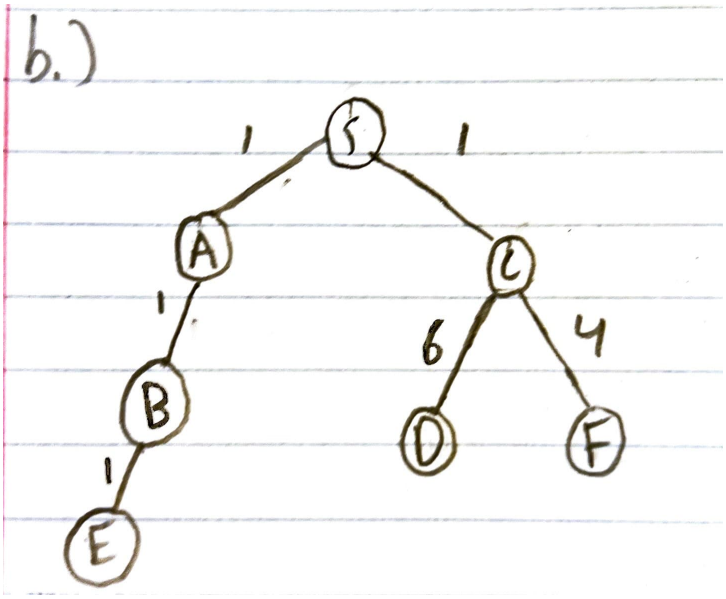
(d) Include a clear explanation of why the search-tree output by breadth-first search is not a valid single-source shortest path tree of G.

Answer. Referenced Michael Levet notes, link: *https://michaellevet.github.io/Algorithms_Notes.pdf*

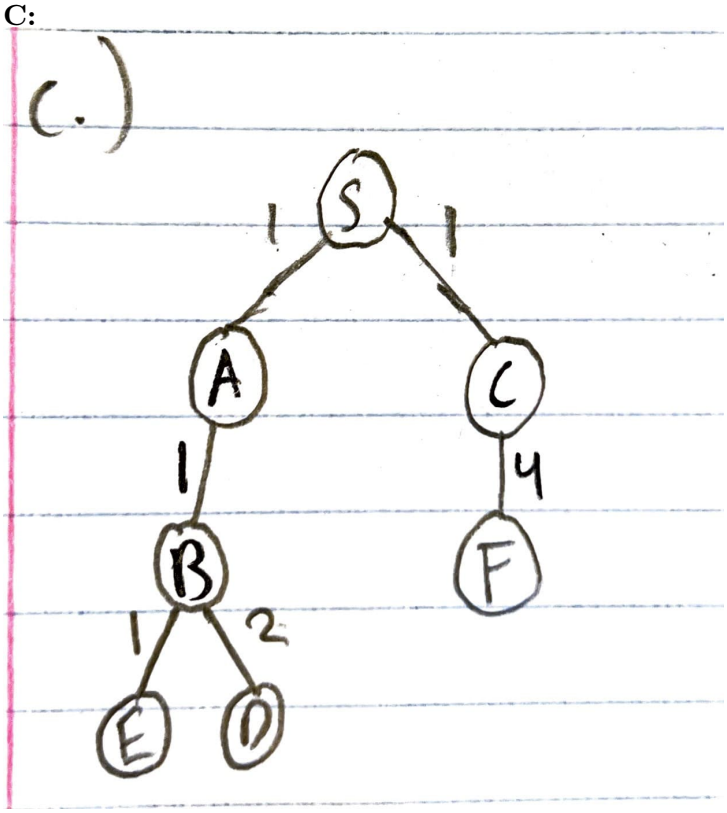
A:



B:



C:



D:

The breadth-first search output is not a valid single-source shortest path tree of G because it does not consider weights on each edge when the algorithm is ran. BFS when ran on a graph will treat each edge weights as equal, thus only working for unweighted graphs. Dijkstra's algorithm when ran on a graph will account for edge weight and will further prioritize that weight, thus finding the shortest path when given a weighted graph. □