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

Problem Set 1

Aidan Reese

Due Date ..... September 5, 2022
Name ..... **Aidan Reese**
Student ID **108418975**
Collaborators ..... **Miles Sanders, Tyler Carr**

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.

Agreed (Aidan Reese). □

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. - The Proposition states that $2^n < n!$ for $n \geq 4$, however in the **Inductive Hypothesis**, the student uses $k \geq 6$ and therefore not including when k or n is equal to 5. Instead they should have used $k \geq 4$. □

(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 problem is that the answer the student provided was not justified or elaborated in anyway. In addition to being built on an incorrect **Inductive Hypothesis**, the student needed to show steps of how $2^{k+1} < (k+1)!$ is true instead of just stating so. □

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.
Base Case :

$$\text{For } n = 2,\ T_{n-2} = 2,\ T_{n-1} = 22$$

$$\begin{aligned} (-1) \cdot (-7)^n + 3 \cdot (5)^n &= -2T_{n-1} + 35T_{n-2} \\ (-1) \cdot (-7)^2 + 3 \cdot (5)^2 &= -2(22) + 35(2) \\ &\ldots \\ 26 = 26 &\rightarrow \textbf{True} \end{aligned}$$

Inductive Hypothesis :
Assume that for every k, $k \geq 2, T_k = (-1) \cdot (-7)^k + 3 \cdot (5)^k = -2T_{k-1} + 35T_k$ is **True**

Inductive Step :

$$\begin{aligned} T_{k+1} = (-1) \cdot (-7)^{k+1} + 3 \cdot (5)^{k+1} &= -2T_k + 35T_{k-1} \\ (-1) \cdot (-7)^{k+1} + 3 \cdot (5)^{k+1} &= -2T_k + 35T_{k-1} \\ (-1) \cdot (-7)^k \cdot (-7) + 3 \cdot (5)^k \cdot (5) &= (-2[(-1) \cdot (-7)^k + 3 \cdot (5)^k] + 35[(-1) \cdot \frac{(-7)^k}{-7} + 3 \cdot \frac{(5)^k}{5}]) \\ (7)(-7^k) + (15)(5^k) &= (-2)(-7^k) + (-6)(5^k) + (5)(-7^k) + (21)(5^k) \\ (7)(-7^k) + (15)(5^k) &= (15)(5^k) + (7)(-7^k) \\ 0 = 0 &\rightarrow \textbf{True} \end{aligned}$$

□

1.3 Problem 3

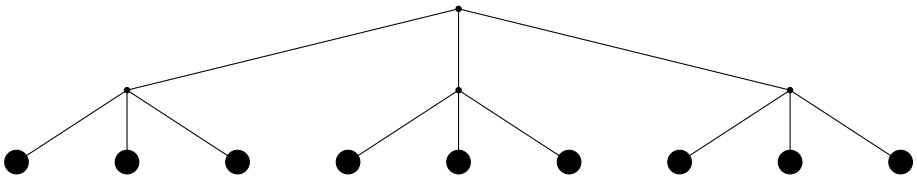
Problem 3. The complete, balanced 3-ary tree of depth d, denoted $\mathcal{T}(d)$, is defined as follows.

- $\mathcal{T}(0)$ 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.
Base Case :
$\mathcal{T}(2)$: Has $3^2 = 9$ leafs. Shown here there are 9 leafs for T(2).

□



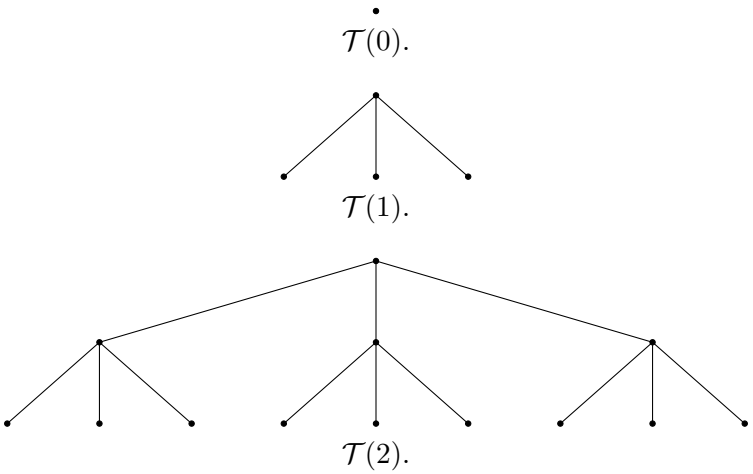
Inductive Hypothesis :
Assume that for all k ≥ 0 that $\mathcal{T}(k)$ has 3^k leafs to be True. $\mathcal{T}(0)$ has 1,$\mathcal{T}(1)$ has 3, $\mathcal{T}(2)$ has 9.

Inductive Step :
Assuming above holds, we should be able to find an equation for $\mathcal{T}(k+1)$ that fits the series stated in the Inductive Hypothesis.

$$NumLeafs(\mathcal{T}(k+1)) = 3^{k+1}$$
$$NumLeafs(\mathcal{T}(k+1)) = 3^k \cdot 3$$

Clearly shown the number of leafs of any given itteration is 3 times the previous itteration, proving 3^k to be True for all k ≥ 0.

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, we can start at vertex u and run a BFS algorithm that visits vertexs and their neighbors until it finds v. Otherwise if vertex v is never visited then v and u are not connected in the Graph.

□

```
Initialize VisitedNodes to False
Initialize Queue
while (len(queue) > 0):
    curr = queue.pro
    for i in neighbors(curr):
        if i = v
             return true
         if not visited[i]:
                visited[i] = True
                queue.append(i)

return false  #Checked every neighbor possible and doesnt find v, therfore not connected
```

(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).
From starting at vertici u we can use BFS or DFS (in this I will be using BFS) to parse the local graph. Keeping a count of how many verticies I have been too, once the BFS has explored every possible option it will either 1.) have visited all nodes and graph is connected and complete or 2.) it will compare the length of the vertici list with the number of visited nodes and compare the two to see if it has searched the whole graph.

□

```
Initialize VisitedNodes to False
Initialize Queue
while (len(queue) > 0):
    curr = queue.pro
    for i in neighbors(curr):
    NumberOfVisited++
         if not visited[i]:
                visited[i] = True
                queue.append(i)

if NumberOfVisited = V.len
    return True #number of visited nodes is same as total number of nodes.
else
    return False #not evert node was visited and therfore there is an unconnected subgraph.
```

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

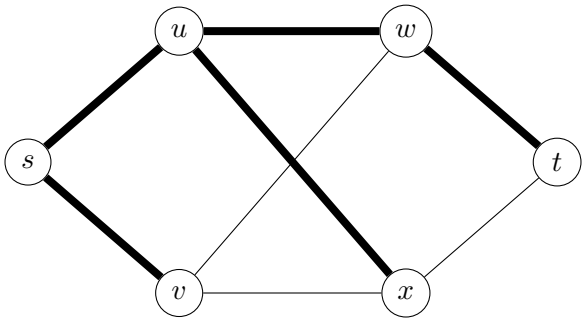
(a) Provide a drawing of the graph G. [**Note:** We have provided TikZ code below if you wish to use LaTeX to draw the graph. Alternatively, you may hand-draw G and embed it as an image below, provided that (i) your drawing is legible and (ii) we do not have to rotate our screens to grade your work.]

(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.
The following is a graph with E_T in the Bold path.

Because of the existence of cycles within G, the shortest path will not have every node within T. For every starting location on G there will exist at least one or more edges that exist within BFS that do not exist within E_t. □



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.]

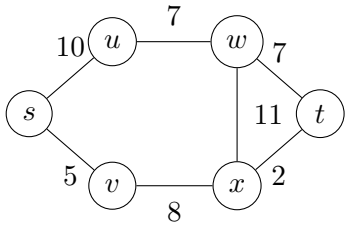
(d) Include a clear explanation of why the search-tree output by breadth-first search is not a valid single-source shortest path tree of G.

Answer.
V : [s,u,v,w,x,t]
E : [(s,u,10),(s,v,5),(v,x,8),(u,w,7),(x,w,10),(w,t,7),(x,t,2)]

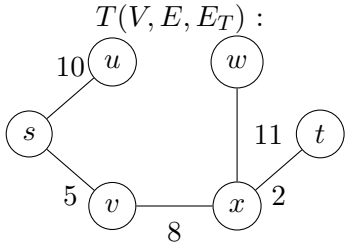
Answer: The issue with BFS is that it does not take into accunt edge weights and instead takes the shortest number of nodes to the location and not the cheapest path. Hense why BFS takes x,w which has a weight of 11 instead of x,t,w that has a totalled weight of 9.

□

$G(V, E, w)$:



$T(V, E, E_T)$:



$T'(V, E_T)$: