CSCI 3104 Fall 2021
Instructors: Profs. Grochow and Waggoner

Problem Set 1

Due Date .... September 7, 2021
Name .... **Michael Ghattas**
Student ID **109200649**
Collaborators .... **Francesca Tenney**

Contents

1 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 Canvas page** only. 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 2). Failure to do so will result in your assignment not being graded.

2 Honor Code (Make Sure to Virtually Sign)

Problem 1.

- 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.

Answer:

I agree to the above, Michael Ghattas. □

3 Standard 1- Proof by Induction

3.1 Problem 2

Problem 2. 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 $k \geq 4$ we have that $2^k < k!$.
- **Inductive Step:** We now consider the $k+1$ case. We have that $2^{k+1} < (k+1)!$. It follows that $2^k < k!$. 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 in the Inductive Hypothesis is claiming $[2^k < (k)!]$ holds $\forall k\ (k \geq 4)$ such that $k \in \mathbb{N}$, instead of stating that $\forall k\ (k > 4)$, $[2^{k-1} < (k-1)!]$ where $k \in \mathbb{N}$ holds true, in order to prove the original assumption in the Inductive Step. □

(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 first error in the Inductive Step is they should have stated IF $[2k < k!]$ is true, THEN it should follow that $[2^{k-1} < (k-1)!]$ is also true $\forall k\ (k > 4)$, and then provide the proof of the claim through a step-by step mathematical justification. Additionally, there should have been a conclusion clearly stating the result and its logical link to the Inductive Hypothesis and claim in the Inductive Step. □

3.2 Problem 3

Problem 3. Consider the recurrence relation, defined as follows:

$$T_n = \begin{cases} 1 & : n = 0, \\ 11 & : n = 1, \\ T_{n-1} + 12T_{n-2} & : n \geq 2. \end{cases}$$

Prove by induction that $T_n = (-1) \cdot (-3)^n + 2 \cdot (4)^n$, for all integers $n \in \mathbb{N}$. [**Recall:** $\mathbb{N} = \{0, 1, 2, \ldots\}$ is the set of non-negative integers.]

Proof. **By induction on** $n \in \mathbb{N}$

- **Base Case:**
  When $(n = 0)$**,**

$$\begin{aligned} T_0 &= (-1) * (-3)^0 + 2 * (4)^0 \\ &= (-1) * (1) + 2 * (1) \\ &= -1 + 2 \\ &= 1 \end{aligned}$$

When $(n = 1)$**,**

$$\begin{aligned} T_1 &= (-1) * (-3)^1 + 2 * (4)^1 \\ &= (-1) * (-3) + 2 * (4) \\ &= 3 + 8 \\ &= 11 \end{aligned}$$

When $(n = 2)$**,**

$$\begin{aligned} T_2 &= (-1) * (-3)^2 + 2 * (4)^2 \\ &= (-1) * (9) + 2 * (16) \\ &= -9 + 32 \\ &= 23 \\ \\ &= T_1 + 12 * T_0 \\ &= 11 + 12 * 1 \\ &= 11 + 12 \\ &= 23 \end{aligned}$$

When $(n = 3)$**,**

$$\begin{aligned} T_3 &= (-1) * (-3)^3 + 2 * (4)^3 \\ &= (-1) * (-27) + 2 * (64) \\ &= 27 + 128 \\ &= 155 \\ \\ &= T_2 + 12 * T_1 \\ &= 23 + 12 * 11 \\ &= 23 + 132 \\ &= 155 \end{aligned}$$

Thus the base cases of $[T_n = (-1) \cdot (-3)^n + 2 \cdot (4)^n]$ **for** $(n = 0, 1, 2, 3)$ **hold true.**

- **Inductive Hypothesis: Fix $k \geq 0$, and suppose $(0 \leq n \leq k)$ for $[T_{k+1} = T_k + T_{k-1}]$ holds true.**
- **Inductive Step: We now consider the $(k+1)$ case. If $[T_{k+1} = T_k + T_{k-1}]$ for $k \in \mathbb{N}$ holds true, then it follows that $T_{k+1} = (-1) \cdot (-3)^{k+1} + 2 \cdot (4)^{k+1}$ also is true. The result follows by induction:**

$$\begin{aligned}
T_{k+1} &= T_k + T_{k-1} \\
&= [(-1) * (-3)^k + 2 * (4)^k] + 12 * [(-1) * (-3)^{k-1} + 2 * (4)^{k-1}] \\
&= [(-1) * (-3)^k + 2 * (4)^k] + 12 * [(-1) * (-3)^{-1} * (-3)^k + 2 * (4)^{-1} * (4)^k] \\
&= [(-1) * (-3)^k + 2 * (4)^k] + 12 * [(-1) * (-1/3) * (-3)^k + 2 * (1/4) * (4)^k] \\
&= [(-1) * (-3)^k + 2 * (4)^k] + 12 * [((1/3) * (-3)^k + (2/4) * (4)^k] \\
&= [(-1) * (-3)^k + 2 * (4)^k] + [(12 * ((1/3) * (-3)^k) + (12 * (1/2) * (4)^k)] \\
&= [(-1) * (-3)^k + 2 * (4)^k] + [((12/3) * (-3)^k) + ((12/2) * (4)^k)] \\
&= [(-1) * (-3)^k + 2 * (4)^k] + [((4) * (-3)^k) + ((6) * (4)^k)] \\
&= (-1) * (-3)^k + 2 * (4)^k + (4) * (-3)^k + (6) * (4)^k \\
&= (-1) * (-3)^k + (4) * (-3)^k + 2 * (4)^k + (6) * (4)^k \\
&= 3 * (3)^k + 8 * (4)^k \\
&= (-1 * (-3)) * (-3)^k + (2 * (4)) * (4)^k \\
&= -1 * (-3)^{k+1} + 2 * (4)^{k+1}
\end{aligned}$$

Conclusion: T_{n+1} holds true as per our induction, thus the argument is true $\forall n \; n \in \mathbb{N}$.

□

3.3 Problem 4

Problem 4. The complete, balanced 4-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 four children to be copies of $\mathcal{T}(d-1)$.

Prove by induction that $\mathcal{T}(d)$ has 4^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 $\mathcal{T}(d) = 4^d$, as these data types are incomparable: a tree is not a number.]

Proof. **By induction on $d \in \mathbb{N}$**

- **Base Case:**

  When $(d = 0)$, $T(0)$ consists of a single vertex, which is a single leaf node by definition.
  i.e. $T(0)$ has $[(4^0) = 1]$ leaf node.

  When $(d = 1)$, $T(1)$ consists of four leaf nodes.
  i.e. $T(1)$ has $[(4^1) = 4]$ leaf nodes.

  When $(d = 2)$, $T(1)$ consists of 16 leaf nodes.
  i.e. $T(1)$ has $[(4^2) = 16]$ leaf nodes.

  Thus the claim for $(d = 0, 1, 2)$ holds true.

- **Inductive Hypothesis: Fix $k \geq 0$, and suppose $(0 \leq d \leq k)$ for tree $T(k)$ consists of (4^k) leaf nodes.**

- **Inductive Step: We now consider Consider $T(k+1)$, which by definition, consists of a root node (v), where each of $(v)'s$ four children are copies of $T(k)$, a complete and balanced tree with each leaf of the k^{th} level has four children. Accordingly, by the inductive hypothesis, $T(k)$ has (4^k) leaf nodes, where (v) has four children, and $T(k+1)$ has in total:**

$$\begin{aligned} 4 * [(4^k)] &= (4) * (4^k) \\ &= 4^{k+1} \end{aligned}$$

  Conclusion: As per our induction process above, the claim tree $T(d)$ consists of (4^d) leaf nodes $\forall d\ d \in \mathbb{N}$ holds true.

□

4 Standard 2- BFS and DFS

4.1 Problem 5

Problem 5. 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).

Algorithm utilizing BFS as per the following description and below pseudo code:

We first create a newqueue(Q), and a boolean array visited[] of size ($verticesCount$) which we initialize to False. Then, we enqueue(u) into Q, and while Q is not *empty*, we set the current vertex = dequeue(Q). If the current vertex is v, then u and v are connected, we are done! Otherwise, if it is not v, we mark the visited[current vertex] = True, then enqueue every unvisited vertex directly connected to current into Q. Ultimately, if we reach this step and v is still not found, we can conclude that v is not connected to u.

```
START
$Q$ = newqueue($u$)
 BFS($G, u$){
        for ($1 \leq i \leq verticesCount$){visited[$i$] = False}
        while ($Q! = empty$){
                $temp$ = dequeue($Q$);
                if (visited[$temp$] == False){
                        if ($temp == v$){return("$u$ connected to $v$")}
                        else{visited[$temp$] = True}
                }
        for (each $vertexNeighbor$ of $temp$){
                if ($vertexNeighbor\ != empty$){enqueue($Q, vertexNeighbor$)}
                else{return($u$ not connected to $v$)}
        }
 }
END
```

□

(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).

Algorithm utilizing BFS as per the following description and below pseudo code:

Similar to part (a), we first create a newqueue(Q), and a boolean array visited[] of size ($verticesCount$) which we initialize to False. Then, we enqueue(u) into Q, and while Q is not *empty*, we set the current vertex = dequeue(Q). Then we mark each visited[current vertex] = True, then enqueue every unvisited vertex directly connected to current into Q. Finally, we traverse the entire list of vertices and check if any of them has a value = False. If any vertex has the value set to False, then the graph (G) is **not connected**, otherwise if all the vertices have values = True, then (G) is **connected**.

```
START
Q = newqueue(u)
 BFS(G, u){
        for (0 ≤ i ≤ verticesCount − 1){visited[i] = False}
        while (Q! = empty){
                temp = dequeue(Q);
                if (visited[temp] == False){visited[temp] = True}
                }
        for (each vertexNeighbor of temp){
                if (vertexNeighbor ! = empty){enqueue(Q, vertexNeighbor)}
                else{
                        for (0 ≤ i ≤ verticesCount − 1){
                                if (visited[i] == False){return(G not connected)}
                                else{return(G is connected)}
                        }
                }
        }
 }
END
```

□

4.2 Problem 6

Problem 6. Give an example of a simple, undirected, and unweighted graph $G(V, E)$ that has a single source shortest path tree which a **depth-first traversal** will not return for any ordering of its vertices. Your answer must

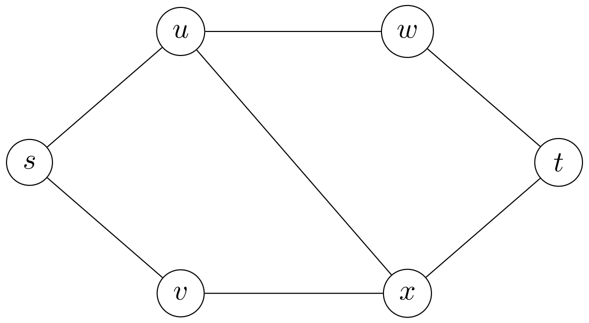
(a) Provide a drawing of the graph G. [**Note:** We have provided TikZ code below if you wish to use LaTeX to draw the graph. Alternatively, you may hand-draw G and embed it as an image below, provided that (i) your drawing is legible and (ii) we do not have to rotate our screens to grade your work.]

(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.]

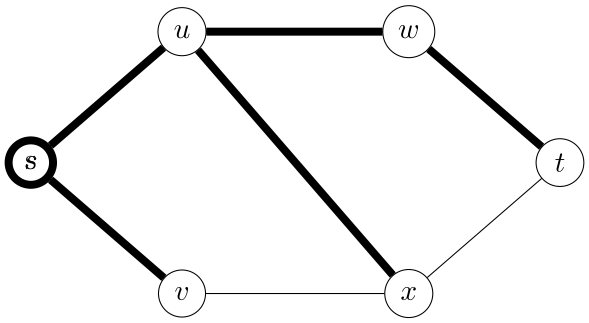
(c) Include a clear explanation of why the breadth-first search algorithm we discussed in class will never produce T for any orderings of the vertices.

Answer:

(a) Graph[V,E]



(b) $[E_T = \{(s,u),(s,v),(u,x),(u,w),(w,t)\}]$



(c) DFS moves in a penetrating process from one connected node to its next connected node until it reaches the end of its path, thus never resulting in the same tree as per part (b) for any orderings of the vertices, resulting in either $[E_T = \{(s,u),(u,w),(w,t),(t,x),(x,v)\}]$**,** $[E_T = \{(s,v),(v,x),(x,u),(u,w),(w,t)\}]$**, or** $[E_T = \{(s,v),(v,x),(x,t),(t,w),(w,u)\}]$**. While BFS traverses in a sweeping method before moving deeper to the next level of nodes as highlighted in part (b).**

□

4.3 Problem 7

Problem 7. 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.]

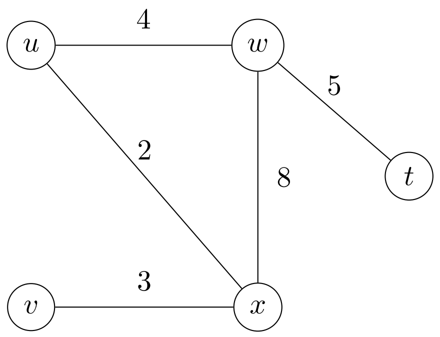
(b) Specify a spanning tree $T(V, E_T)$ that is returned by BFS, but is not a single-source shortest path tree. [**Note:** You may again hand-draw this tree. If you wish, you may clearly mark the edges of T on your drawing of G. Please make it easy on the graders to identify the edges of T.]

(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.]

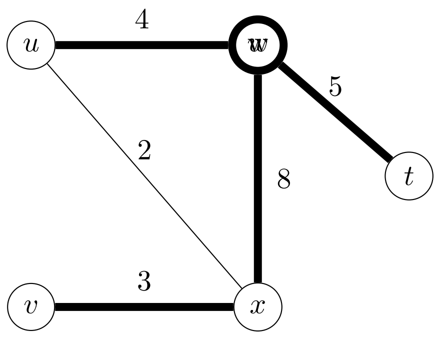
(d) Include a clear explanation of why the search-tree output by breadth-first search is not a valid single-source shortest path tree of G.

Answer:

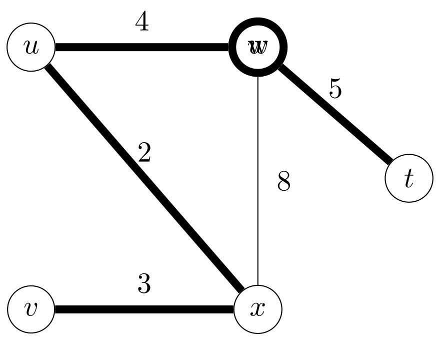
(a) $V = [w, u, x, v, t]$ **and** $E = [8, 4, 2, 3, 5]$



(b) BFS spanning tree



(c) Single source shortest path



(d) The algorithm of BFS does not take the weight of the edges into consideration, it only calculates the quantity of edges it traverses. Accordingly, BFS return the spanning tree with the shortest path (i.e. least quantity of edges) from source to goal regardless of the weight of the edges involved. This is notable in the graph of part (b) $[w - x - v]$ which yields a $weight = 11$ and $Edges = 2$, while the real solution in part (c) $[w - u - x - v]$ yields a $weight = 9$ and $Edges = 3$. Thus in comparison we can clearly see that BFS does not function as well with the weighted graph in part (a) due to its disregard to weights of the edges.

$\square$