ECE 512 – Topics in Data Science
Homework 4

Dror Baron and Hangjin Liu; Fall 2023

Due: September 27, 2023

Administrative instructions:

1. For any clarification or doubts, the TA Hangjin Liu (hliu25 AT ncsu DOT edu) is in charge of homeworks. She should be your first point of contact on homework-related issues.
2. The homework can be submitted individually, in pairs, or triples.
3. You should submit electronically through Moodle by midnight the day that the homework is due.
4. Please justify your answers carefully.

1. **Computational complexity:** Prove the following (provide constants c_1, c_2, N_0, etc., as needed). Make sure to justify your answers.
   a. $n^2+100n=\Theta(n^2)$.
   b. $(\log(n))^3=O(n)$.
   c. $n^{0.5}=\Omega((\log(n))^3)$.

2. (Cormen et al. Chapter 1.) **Searching problem.** Consider the following searching problem. The inputs are comprised of a sequence of N numbers, $X=<x_1,\ldots,x_N>$, and a scalar value v. The output is an index i such that $v=x_i$, else a special NIL value if v does not appear in X.
   a. Write pseudocode for a linear time search that scans through X searching for v.
   b. How many elements of the array need to be searched on average? (Assume that v appears once in X, and is equally likely to appear in any among the N location.) What's the worst case? Please express the average and worst cases in Θ notation.
   c. Suppose that you plan to perform lots of searches for various possible values. Please propose and discuss a data structure for storing X in a way that allows fast search. Discuss the complexity of setting up this data structure and complexity of each individual search. Please also describe the memory complexity and additional logic that you may used for improving search speed.

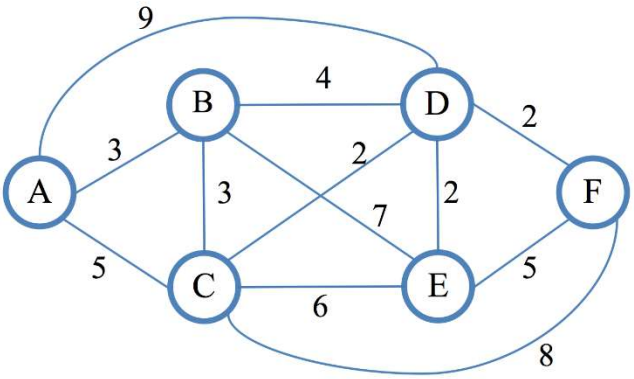
3. **Paths and simple paths.** Prove that if a directed or undirected graph contains a path from node u to node v, then it contains a simple path from u to v.

4. **Mathematical induction:** Recall that many divide and conquer style algorithms can be characterized computationally in terms of a recursive function for their running time. In this question, you will prove that the solution to the following formula,

$$T(n)=\begin{cases}2, if\ n=2, k\leq 1\\ 2T\left(\frac{n}{2}\right)+n, if\ n=2^k, k>1'\end{cases}$$

   is $T(n)=n\times\log_2(n)$. To prove this recursive result, your inductive approach will proceed as follows.
   a. *Basis case* – provide some small value of n for which you can argue "somehow" that the relation holds. (Hint: using k=1 or 2 makes sense.)
   b. *Inductive hypothesis* – state your hypothesis in terms of "$T(n)=n\times\log_2(n)$ for n taking on so and so value." (Note that the recursive formula only applies to integer valued k.)
   c. *Inductive step* – prove that if the hypothesis holds for k=1, 2, …, K, then it also holds for K+1.

5. **Dijkstra's algorithm:** Below is a plot of an undirected graph, G(V,E), where the vertices are labeled {A, B, …, F}, some edges between vertices are shown, and each edge has a corresponding length. The graph can be interpreted as follows: (i) vertices correspond to locations; (ii) edges correspond to roads; and (iii) edge lengths correspond to the distance or travel time for these roads. Dijkstra's algorithm uses a mathematical optimization technique known as dynamic programming, which we will encounter later this semester, to find the shortest path between nodes on a graph.



   a. Please read about Dijkstra's algorithm (the Wikipedia entry seems reasonable). After you've learned about it, run it "by hand" on the graph below. Compute the shortest distance between nodes A and F.
   b. A related algorithm by Bellman and Ford (you can read about it on the Wikipedia algorithm for the Bellman-Ford algorithm) computes the shortest paths from one node to all other nodes. The resulting algorithm is faster than running Dijkstra's algorithm |V| times. As before, run the algorithm "by hand" for the node B. That is, compute the shortest paths between B and the other nodes.

Discussion of Dijkstra vs. Bellman-Ford: The Bellman Ford algorithm computes the shortest paths from a single source vertex to all the vertices and its worst case complexity is O(|V|*|E|). Dijkstra's algorithm only computes the path from one vertex to a single other vertex. There are two main implementations of Dijkstra's algorithm. The advanced one is O(|E|+|V|log(|V|)), and uses a data structure called a heap, which we didn't cover in ECE592. For very sparse graphs (the adjacency matrix is dominated by zeros), |V|log(|V|) might be larger than |E|, but for most graphs this is O(|E|). Without the heap data structure, Dijkstra requires O(|V|^2) runtime.

Suppose that we want to obtain the Bellman-Ford functionality of shortest paths from one vertex to all others by running Dijkstra |V| times.
1. The sophisticated heap version is O(|V|*|E|+|V|^2*log(|V|), which is no less, and possibly more (slower), than Bellman-Ford.
2. The regular Dijkstra version is O(|V|^3). Recall that |E|=O(|V|^2), implying that Bellman-Ford=O(|V|*|E|) while running Dijkstra |V| times is O(|V|^3). That is, Bellman-Ford is more efficient for the problem it's designed for.