Homework 2

CS 4104 (Spring 2024)

Assigned on February 7, 2024.
Submit a PDF file containing your solutions on Canvas by 11:59pm on February 14, 2024.

Instructions:

- The Honor Code applies to this homework with the following exception:
  - You can pair up with another student to solve the homework. Please form teams yourselves. Of course, you can ask the instructor for help if you cannot find a team-mate. You may choose to work alone.
  - You are allowed to discuss possible algorithms and bounce ideas with your team-mate. **Do not discuss proofs of correctness or running time in detail with your team-mate. You must write down your solution individually and independently. Do not send a written solution to your team-mate for any reason whatsoever.**
  - *In your solution, write down the name of the other member in your team. If you do not have a team-mate, please say so.*
  - Apart from your team-mate, you are not allowed to consult sources other than your textbook, the slides on the course web page, your own class notes, the TAs, and the instructor. In particular, do not use a search engine or any other online sources including ChatGPT or generative AI software of that ilk.
- **Do not forget to typeset your solutions.** *Every mathematical expression must be typeset as a mathematical expression, e.g., the square of n must appear as n^2 and not as "n^2".* You can use the LaTeX version of the homework problems to start entering your solutions.
- Do not make any assumptions not stated in the problem. If you do make any assumptions, state them clearly, and explain why the assumption does not decrease the generality of your solution.
- You must also provide a clear proof that your solution is correct (or a counter-example, where applicable). Type out all the statements you need to complete your proof. *You must convince us that you can write out the complete proof. You will lose points if you work out some details of the proof in your head but do not type them out in your solution.*
- If you are proposing an algorithm as the solution to a problem, keep the following in mind (the strategies are based on mistakes made by students over the years):
  - Describe your algorithms as clearly as possible. The style used in the book is fine, as long as your description is not ambiguous. Explain your algorithm in words. A step-wise description is fine. *However, if you submit detailed code or pseudo-code without an explanation, we will not grade your solutions.*
  - Do not describe your algorithms only for a specific example you may have worked out.
  - Make sure to state and prove the running time of your algorithm. You will only get partial credit if your analysis is not tight, i.e., if the bound you prove for your algorithm is not the best upper bound possible.
  - You will get partial credit if your algorithm is not the most efficient one that is possible to develop for the problem.
- In general for a graph problem, you may assume that the graph is stored in an adjacency list and that the input size is $m + n$, where n is the number of nodes and m is the number of edges in the graph. Therefore, a linear time graph algorithm will run in $O(m + n)$ time.

Problem 1 (5 points) Consider the version of BFS where we store both the node index and its distance from the starting node s in the queue L. In other words, when we want to push a node v into L, we actually push the pair $(v, d(s, v))$. Here, $d(s, v)$ is the length of the shortest path between s and v. Now suppose that u and w are two consecutive nodes that we pop from L. State and prove the relationship between $d(s, u)$ and $d(s, w)$.

Problem 2 (25 points) In class, we presented an iterative algorithm for computing the connected component of an undirected graph $G = (V, E)$ that contains a specific node s. This algorithm appears on page 54 of the slides on priority queues, BFS, and DFS. Recall that we used R to denote the set of nodes computed by this algorithm. Suppose that C is the actual connected component of G that contains s; C is also a set of nodes. For the algorithm to be correct, we must convince ourselves that for an arbitrary graph G and node s, $R = C$. In class, we completed half the proof that this algorithm was correct. If R is the set of nodes computed by the algorithm, we proved that every node in C must be in R. In other words, we proved that C is a subset of R. We used a proof by contradiction.

Your task in this problem is to complete the proof of correctness by showing the opposite, i.e., R is a subset of C. In other words,

> for every node v in R (the set of nodes computed by the algorithm), there must be a path from s to v in G, i.e., v is indeed a member of C.

We want to prove something about the output of the algorithm, i.e., the elements of R. The intuition underlying the proof is that since the algorithm is iterative, we might be able to prove an invariant that holds after each iteration of the algorithm. This idea is similar to a proof by induction.

We need some "index" over which to do the induction, i.e., some number that is increasing over the course of the algorithm. While there is no explicit counter in the algorithm, since it is iterative, we can number the vertices consecutively from 1 onwards based on when they are added to the set R. To make the proof simple, let us rename the vertices as $v_0 = s, v_1, v_2, \ldots v_{r-1}, v_r$, where r is the size of R when the algorithm terminates, and for $1 \leq i \leq r$, v_i is the vertex added to R in iteration i of the algorithm.

There are three components to this proof. Each part of this problem corresponds to one of these components.

(i) (3 points) *Base case:* $i = 0$. There is a path from s to v_0. Write down a statement explaining why this claim is true.

(ii) (7 points) *Inductive hypothesis:* Select one of the following choices as the inductive hypothesis to use in the proof, keeping in mind that the successful completion of part (iii) will depend on the correct answer here. It will be helpful to think about parts (ii) and (iii) together. In your answer, you simply have to state your choice.

- (i) There is a path from s to v_k.
- (ii) There is a path from v_{k-1} to v_k ($k \geq 1$).
- (iii) For every i between 1 and k, there is a path from s to v_i.
- (iv) For every i between 1 and k, there is a path from v_{i-1} to v_i.

(iii) (15 points) *Inductive step:* In this step, complete the proof (based on your answer for the inductive hypothesis) that there is a path from s to v_{k+1}.

Problem 3 (20 points) Given an undirected graph G and an edge e in this graph, develop a linear time algorithm to determine if there is a cycle in G that contains e.

Problem 4 (20 points) Solve exercise 7 in Chapter 3 (page 108–109) of your textbook. I provide the essential part of the problem here.

> *Claim: Let G be an undirected graph on n nodes, where n is an even number. If every node of G has degree at least $n/2$, then G is connected.*

Decide whether you think the claim is true or false, and give a proof of either the claim or its negation.

Problem 5 (30 points) Solve exercise 9 in Chapter 3 (page 110) of your textbook. Suppose that an n-node undirected graph G contains two nodes s and t such that the distance between s and t is strictly greater than $n/2$. Show that G must contain some node v, not equal to s or t, such that deleting v from the graph destroys all s-t paths. (In other words, the graph obtained from G by deleting v contains no path from s to t.) Give an algorithm with running time $O(m + n)$ to find such a node v. *Hint:* At least one of the layers in the BFS tree rooted at s has a special property.