Problem Set 1

mst3k

data structures and algorithms 2 - spring 2024

due january 31, 2024 at 11:59p

Collaboration Policy: You are encouraged to collaborate with up to 4 other students, but all work submitted must be your own *independently* written solution. List the computing ids of all of your collaborators in the `collabs` command at the top of the tex file. Do not share written notes, documents (including Google docs, Overleaf docs, discussion notes, PDFs), or code. Do not seek published or online solutions for any assignments. If you use any published or online resources (which may not include solutions) when completing this assignment, be sure to cite them. Do not submit a solution that you are unable to explain orally to a member of the course staff. Any solutions that share similar text/code will be considered in breach of this policy. Please refer to the syllabus for a complete description of the collaboration policy.

Collaborators: list your collaborators

Sources: list your sources

problem 1 *Asymptotics*

1. Consider the following functions, $f(n) = n(\log n)^2$ and $g(n) = n^{1.5}$. Which grows more quickly? (That is, which would be a "worse" time-complexity.) Express your answer in terms of one of the order-classes we've studied, i.e. something in a form like $f(n) = \Theta(g(n))$ but with something other than Big-Theta. Explain your answer with a short proof; you may use any definitions from the course slides.

Solution:

problem 2 *Proving Quicksort*

For this problem, we'll be reviewing how a **proof by induction** works. More specifically, we will be using strong induction. For a refresher on induction, Wikipedia has a great description and a few example proofs at: https://en.wikipedia.org/wiki/Mathematical_induction. To prove the correctness of an algorithm using induction, we must consider the parts of the proof: the base case, the inductive hypothesis, and the inductive step.

We will use induction to prove that Quicksort is correct. We will let you assume that the `partition` operation works correctly. Recall that `partition(list,first,last)` returns the location p of an item in the sublist `list[first:last]`, where all items in positions before p are $<$ `list[p]`, and all items after position p are $>$ `list[p]`. The item at location p is the pivot-value, and `partition` puts it into its correct position but does not sort what's before it or after it. After `partition` is done, Quicksort is called recursively on the sublists before and after the pivot-value.

1. In induction we start with the **base case**. If $n = 1$ (i.e., there is one item in the list), explain why `partition` and Quicksort produce the correct answer for that list of size 1.

   Solution:

2. We must next make an assumption: our **inductive hypothesis**. Describe briefly this assumption. *For any list of size less than n, ...*

   Solution:

3. **Inductive step**. Now we need to use this **inductive hypothesis** to make an argument that, for a list of size n, Quicksort correctly produces a list that's sorted. Namely, assuming that `partition` works correctly and the **inductive hypothesis** is true, write an argument below

that shows, for a list of size n, the call to `partition` and the recursive calls to Quicksort correctly sorts that list.

Solution:

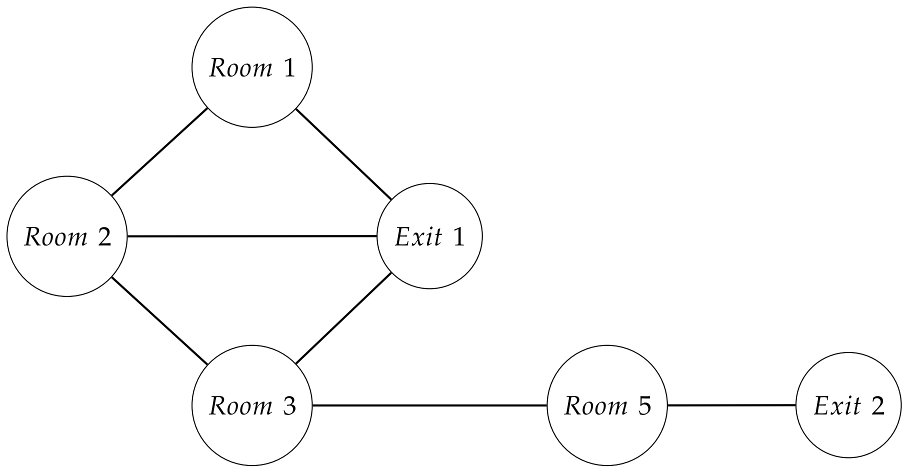
PROBLEM 3 *Optimal Study Spot*

Alderman Library is finally open after years of renovations! But now it's time to find your new optimal study spot. Your main concern is being able to get to each of the building exits as quickly as possible (unlike Clemons, strange and unexpected things can happen in Alderman stacks and you never know which way you may need to exit). You are given a graph $G = (V, E)$ where each node $v \in V$ represents either a room or an exit. Each edge in the graph represents a doorway connecting a room to either another room or an exit. From any room, there is a path to every exit. Your algorithm must find the room (or rooms) such that the distance to the farthest exit is minimized.

Give a clear description of an algorithm to solve this problem. State and explain the time complexity of your algorithm. Base your algorithm design on an algorithm we have studied in this unit of the course.

Note: you may later decide to find an optimal study spot in one of the other buildings currently being built, so your algorithm should not be specific only to Alderman's new layout! Design your algorithm to work on any graph of similar style.

For example, in the graph below, the optimal rooms are *Room 3* and *Room 5*. You can get to either exit from either *Room 3* or *Room 5* by going through at most 2 doors. *Room 1* would require going through 4 doors to get to *Exit 2* and *Room 2* would require going through 3 doors to get to *Exit 2*.



Solution: