Problem Set 2 mst3k

data structures and algorithms 2 - fall 2024 due may 18, 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 here

**Sources**: list your sources here

## problem 1 *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, a few good sources are:

- A summary from an offering of our CS2120 course:
  `https://www.cs.virginia.edu/~emo7bf/cs2120/s2023/techniques.html#proof-by-induction`
- A free online textbook (see Chapter 5):
  `https://people.csail.mit.edu/meyer/mcs.pdf`
- Wikipedia has a good description and a few example proofs:
  `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 a 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 2 *Sorry-Oh*

You are playing a new video game about a character named "Sorry-Oh". Sorry-Oh is currently at a point in the game where he needs to find a way to get from one corner of a rectangular room to the opposite corner. The rectangular room can be of any size $x$ by $y$, where $x$ and $y$ are positive integers greater than 1. Unfortunately, some cells contain lava. When Sorry-Oh steps into a cell containing lava, he says "Sorry" and takes a certain amount of damage. The room can be visualized as a rectangular grid of cells whose size is $x$ by $y$. For each move, Sorry-Oh can only step to a cell that is either horizontally or vertically adjacent to his current cell. To get from the starting corner to the opposite corner, Sorry-Oh is only allowed to make a total of ($x + y$ - 2) moves. For example, in the grid shown below (which is 3x3), Sorry-Oh would only be allowed to make 4 (3 + 3 - 2) moves.

Sorry-Oh starts in the top-left corner cell with a damage value of 0. Each cell contains a number representing a certain amount of damage specific to that cell. Cells that do not contain lava have a value of 0. Cells that do contain lava have a positive integer value. When moving from one cell to the next, Sorry-Oh's damage will increase when he arrives in the new cell (assuming the new cell has a value greater than 0).

Your goal is to get Sorry-Oh from the starting cell on the top-left (the starting cell will always be the top-left cell and be designated by $S$) to the ending cell on the bottom-right (the ending cell will always be on the bottom-right), and to have the lowest total damage value after arriving at that ending cell.

Given the following example grid -

| | | |
|---|---|---|
| S | 2 | 4 |
| 2 | 3 | 0 |
| 2 | 2 | 1 |

An example path could be computed as follows:

1. Start cell, damage = 0
2. move Right, d = 2 (0 + 2)
3. move Right, d = 6 (2 + 4))
4. move Down, d = 6 (6 + 0))
5. move Down, d = 7 (6 + 1))
6. Total damage from the moves R-R-D-D is 7.

Another example path:

1. Start cell, damage = 0
2. move Down, d = 2 (0 + 2)
3. move Right, d = 5 (2 + 3)
4. move Down, d = 7 (5 + 2))

5. move Right, d = 8 (7 + 1))
6. Total damage from the moves D-R-D-R is 8.

You are given a grid, but will solve the problem by converting the grid to a graph, then using an algorithm that works on graphs.

1. Give a brief description of how you would translate the grid to a weighted, directed graph suitable to solve the problem.

   **Solution:**

2. Describe an algorithm that can find a path through the graph resulting in the lowest possible damage.

   **Solution:**

3. The runtime of your algorithm must be no worse than $O(|E|log|V|)$. Explain the time complexity of your algorithm.

   **Solution:**

## PROBLEM 3 *Bipartite*

A graph is bipartite if each vertex can be assigned to one of two sets, such that for every edge in the graph $(v_i, v_j)$, the vertices $v_i$ and $v_j$ do not belong to the same set.

Write an algorithm that takes a connected undirected graph $G = (V, E)$ and returns a list of edges $E'$ than can be removed from $G$ to make it bipartite, i.e., the graph $G = (V, E - E')$ is bipartite. If $G$ is bipartite to begin with, the list you return will be empty.

**Solution:**