Theory of Algorithms Homework 1

Evan Dreher

August 2022

1 Problem 1

In your own words, formally state input and output conditions for a Longest Increasing Subsequence algorithm.

- **Input:** a sequence a of numbers $< a_1, a_2 ... >$
- **Output:** The longest subsequence of a (which need not be contiguous) denoted $< a'_1, a'_2, ...a'_n >$ such that for every $1 \leq i \leq n$, $a'_i \leq a'_{i+1}$ and a'_i appears before a'_{i+1} in the original sequence.

2 Problem 2

In your own words, formally state input and output conditions for a Graph Coloring Algorithm.

- **Input:** A graph G, with vertex set V and edge set E.
- **Output:** An integer representing the minimum number of colors required to color each vertex of G such that no adjacent vertices share the same color.

3 Problem 3

In your own words, formally state input and output conditions for a Longest Path Problem algorithm.

- **Input:** A weighted graph G, with vertex set V and edge set E.
- **Output:** A list of vertices $[v_1, v_2, ...]$ such that each vertex in the list is adjacent to the next vertex in the list, no vertex is in the list twice, and the sum of the weights of the edges between adjacent vertices in the list is greater than or equal to any other such list of vertices that can be constructed on G.

4 Problem 4

Consider a “Reverse Search” Algorithm like the following:

I claim that this program will only return NIL is x is not in the array. Write a loop invariant that allows you to prove my claim.

Algorithm 1 Reverse-Search(A, x)

```
for i = A.length down to 1 do
    if A[i] = x then
        return $x$
return $Nil$
```

- **Loop Invariant:** At the start of loop, every element in $A[i+1...length]$ is not equal to x.
- **Initialization:** At the start of the first loop $i = A.length$. According to the LI, every element in $A[A.length+1...A.length]$ is not equal to x. Since that slice of A contains 0 elements, it is vacuously true. Thus the LI holds.
- **Maintenance:** Assume the LI holds at the start of the loop where $i = k$. In the loop we check to see if $A[k] == x$. If it does, then the program terminates early by returning x immediately. Otherwise, $A[k]! = x$, and every element in $A[k+1...length]! = x$ (by the assumption), thus every element in $A[k...length]! = x$ so the LI holds for the next loop.
- **Termination:** The loop terminates when $i = 0$ or when $i = k$ where $A[k] = x$. In the former case, the LI asserts that every element in $A[1...length]! = x$, i.e $x \notin A$, the method then returns NIL. In the other case, $x \in A$ and x is returned, thus the method only returns NIL if $x \notin A$.

5 Problem 5

Consider Bubble Sort, as defined by the following:

Algorithm 2 Bubble-Sort(A)

```
for i = 1 to A.length do
    for j = 1 to A.length - 1 do
        if A[j] > A[j+1] then
            swap(A[j], A[j+1])
```

State a loop invariant for Bubble Sort that would allow you to demonstrate that this algorithm does, indeed, sort the array.

- **Loop Invariant:** At the start of the loop, the $(i-1)$-th largest elements of A will be sorted in ascending order in the rightmost $i-1$ indeces of A.
- **Initialization:** At the start of the algorithm, $i = 1$. According to the LI, the 0 largest elements of A will be sorted in ascending order in the 0 rightmost indeces of A. This is vacuously true, thus the LI holds.
- **Maintenance:** Assume the LI holds at the start of the loop where $i = k$. By the LI we know that $A[k+1]$ is greater than or equal to every element in $A[1...k]$, so the j-loop will bring the largest element in $A[1...k]$ to the index $A[k]$, let us call this element x. By the LI, we know that x is the kth largest element of A, which means at the end of the loop, the k

largest elements of A are sorted in ascending order in the rightmost k indeces of A, this the LI holds for the next loop.

- **Termination:** The loop terminates when $i = A.length + 1$. According to the LI, the $A.length + 1 - 1$ largest elements of A will be sorted in ascending order in the rightmost $A.length - 1 + 1$ indeces of A. Since that slice of A is the whole array, it follows that the whole array will be sorted in ascending order.

6 Problem 6

In our in-class work, we described the Shinjuku overlap problem in terms similar to the following:

- **Input:** as sequence of tuples $< a_1, a_2, ...a_n >$, where each tuple consists of three values ¡ID, start, finish¿.
- **Output:** A pair of ID values ¡x, y¿ such that x and y are the IDs that most frequently overlap.

We're going to build a more formal definition of the final output here. Let's look at formal ways to say a couple of things.

- Formally, in terms of the input, what can you state about where x and y come from?

  x and y are unique and distinct IDs of people from the sequence of tuples.

- Formally define what it would mean for two tuple a_i and a_j to overlap.

  Tuples a_i and a_j are said to overlap if $a_i.id \neq a_j.id$ and $a_i.start < a_j.start < a_i.finish$.

- Formally define "The number of times x and x overlap."

  Let A_x and A_y be the set of all tuples a where $a.id = x$ or $a.id = y$ respectively. The number of times that x and y overlap is defined as the number of tuples in A_x such that there is a tuple in A_y where the two tuples overlap.

- Formally state the output.

  Let S be the set of all non-reflexive combinations of ID values in the input.

  Let $O_{x,y}$ be the number of times that two IDs x and y overlap.

  Output: An element of S denoted (x, y) such that for every other element of S denoted (a, b), $O_{x,y} \geq O_{a,b}$.

7 Problem 7

Suppose we decide to process our subway data by sorting our input data in order of how long each entry spends in the subway: shortest visits first, longest visits last. Keeping the same input values as before, how would you describe the output of this pre-sorting step?

Output: A permutation of the input sequence $< a'_1, a'_2, ...a'_n >$ such that for all $1 \leq i \leq n$, $a'_i.finish - a'_i.start \leq a'_{i+1}.finish - a'_{i+1}$.start.