Final Examination

Algorithms & Data Structures
NTNU IDATA 2302

Dec. 2024

1 Basic Knowledge

Question 1.1 (1 pt.). *In the* binary search tree *(BST) shown below, which node is the successor of* Node L*? Explain your reasoning. The nodes are ordered alphabetically.*

Question 1.2 (1 pt.). *Consider the following algorithm, shown as a Java program. Does this algorithm always terminate? Explain your reasoning?*

```
int factorial(int n) {
 if (n == 1) return 1
 else return n * factorial(n+1);
}
```

Question 1.3 (1 pt.). *Which data structure ensures that every node is greater than its children? What is this data structure used for?*

Question 1.4 (1 pt.). *Consider an algorithm that has two steps: The first step runs in $O(n)$ (linear time), but the second step runs in $O(n^2)$ (quadratic time). What is the runtime complexity of the complete algorithm? Explain your reasoning.*

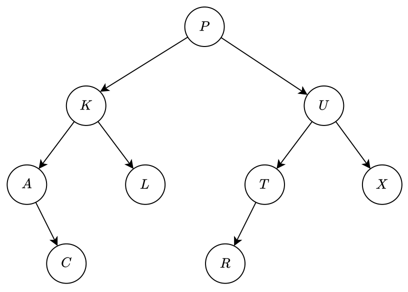

Figure 1: A sample binary search tree, where nodes are ordered by alphabetical order

Question 1.5 (1 pt.). *A directed graph G, with four vertices A, B, C and D, is described by the following adjacency matrix. Does G contain a cycle? Explain your reasoning.*

$$G = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \tag{1}$$

2 Longest Common Prefix (5 pts.)

In this section, we look at the *longest common prefix* (LCP) problem. Given two sequences of characters (i.e., strings in Java), the task is to find the longest subsequence of characters that is shared from the start of the two given strings. Consider the following examples:

- The LCP of ”apple” and ”apply” is ”appl”;
- The LCP of ”apple” and ”apricot” is ”ap”;
- The LCP of ”apple” and ”apple” is ”apple”;
- The LCP of ”apple” and ”banana” is ”” (the empty string);
- The LCP of ”apple” and ”” is ”” (the empty string).

Below is the recursive algorithm, which computes the LCP of two strings, namely ”left” and ”right”, here shown as a Java method:

```
1  String longestCommonPrefix(String left, String right) {
2    if (left.isEmpty() || right.isEmpty()) {
3      return "";
4    } else if (left.charAt(0) != right.charAt(0)) {
5      return "";
6    } else {
7      return left.charAt(0) + longestCommonPrefix(left.substring(1),
8                                                  right.substring(1));
9    }
10 }
```

Question 2.1 (1 pt.). *In this recursive LCP algorithm, what is/are the base case(s)? What is/are the recursive case(s)?*

Question 2.2 (1 pt.). *Consider the number of string concatenations performed by our LCP algorithm. What is the worst-case scenario? Given an example of the input strings that would trigger that worst-case scenario?*

Note that the LCP algorithm contains only one string concatenation, on line 7. It is denoted by a `+` *sign in Java.*

Question 2.3 (2 pt.). *In the worst-case scenario, how many* string concatenations *does this algorithm performs. Express this number as a mathematical function f. Explain your reasoning and calculations.*

Question 2.4 (1 pt.). *Propose an alternative algorithm solving the LCP, but more efficient than the recursive one used in the previous questions. Explain why your solution is more efficient.*

3 Number Guessing Game

We now look at the "Number Guessing Game", where the first player chooses a number that the other player tries to find. First the two players must agree upon a minimum and maximum value. Then, after each attempt, the first player tells the other whether that was correct, too large or too small. A typical game between a player and a "CPU" could look like:

CPU: I have chosen a number between 0 and 100 (included). Can you find it?
PLAYER: Let's try 97.
CPU: Too large!
PLAYER: Let's try 56.
CPU: Too small!
PLAYER: Let's try 61.
CPU: Too large
PLAYER: Let's try 59.
CPU: Well done! You've found it in 4 tries.

Question 3.1 (1 pt.). *As a player that plays against the CPU, what would be your strategy (i.e., algorithm) to win the game? Explain how you would choose your next try.*

Question 3.2 (1 pt.). *Considering your total number of attempts, what is the best-case scenario of your strategy? When does it occur? In that best-case scenario, how many attempts will you need to find a value chosen in an interval $[a, b]$ where a and b are two integers such that $a < b$. Explain your reasoning.*

Question 3.3 (2 pt.). *Considering again your total number of attempts, what is the* worst-case scenario *for your strategy? When does it occur? In that worst-case scenario, how many attempts will you need?*

Question 3.4 (1 pt.). *Without proving it, do you think there exists (or not) another algorithm, faster than your solution. Explain your reasoning.*

4 Course Schedule

We now look at the *Course schedule problem.* We are given a list of university courses, along with their set of prerequisite courses. Students who would like to enroll in a course must have already completed all of its prerequisite courses.

Consider, for example, the following courses:

- *Discrete math* (DM) has no prerequisite: Any student can join.
- *Computer architecture* (CA) has no prerequisite.
- *Algorithms and data structures* (ADS) requires DM and CA
- *Database* (DB) requires ADS
- *Operating systems* (OS) requires ADS

In this scenario, a student must complete these courses in an order that complies with their prerequisites. There may be many valid orderings, such as (DM, CA, ADS, DB, OS) or (CA, DM, ADS, OS, DB), but the order within each prerequisite group must be respected. In contrast, the ordering (DM, CA, OS, ADS, DB) is invalid because one cannot enroll in the operating system (OS) course without having completed ADS first. Note that there may not be any valid ordering whenever the dependencies between courses form a cycle, for instance, if DM had OS as prerequisite.

In this problem, we aim at:

1. Find *one* valid ordering, represented as a sequence of all the courses;
2. Detect cases where there is no valid ordering.

Question 4.1 (1 pt.)**.** *What data structure (or mathematical structure) would you use to reason about courses and their prerequisites? Explain how this data structure would represent the courses and their prerequisites.*

Question 4.2 (1 pt.)**.** *Consider the following API that enables manipulating courses.*

```
class Course {

   /**
    * Returns "true" if and only if "this" and the "other" object
    * represent the very same course.
    * For example:
    *  - dm.equals(dm) is true
    *  - ads.equals(db) is false
    */
   boolean equals(Course other) { ... }

   /**
```

```
     * Returns the sequence of direct prerequisites of this course.
     * For example:
     *  - ads.prerequisites() yields [] (i.e., the empty list)
     *  - ads.prerequisites() yields [dm, ca]
     */
    List<Course> prerequisites() { ... }
}
```

Assuming prerequisites do NOT *form a cycle, propose an algorithm to list the prerequisites (direct or indirect) of a single given course, in a valid order. Below is the signature of such an algorithm:*

```
List<Course> orderPrerequisites(Course target) {
    // Your algorithm here
}
```

Note that you do not *have to implement the Course class. It is there for you to use in your algorithm.*

Question 4.3 (1 pt.)**.** *We now address the problem of cycles in the prerequisites. How can we modify the* `orderPrerequisites(...)` *algorithm to detect and handle cycles in the prerequisite graph? A cycle occurs whenever a course is directly or indirectly a prerequisite of itself. Propose an updated algorithm.*

Question 4.4 (2 pts. BONUS)**.** *Finally, we look at finding a valid ordering for a set of courses (as opposed to a single course in the previous questions). Given a set of courses and their prerequisite relationships, design an algorithm to find a valid ordering such that all prerequisites of every course are completed before the course itself. Assume you have access to the* `orderPrerequisites(Course)` *procedure from the previous question, which returns a valid ordering for a single course and its prerequisites.*