ELEC 278: Fundamentals of Information Structures
Lab 5: Binary Search Trees

Fall 2023–Instructors: Ni & Mertin

November 2, 2023

Please read the entire document to understand the requirements and process for completing the lab.

1 Objectives

The objectives for this lab are for you to demonstrate your understanding of binary search trees as a concept and the implementation of trees in C. Using a provided implementation of a binary tree data structure, you will implement four modification algorithms on trees.

2 Instructions

Download the file `lab5.zip` from OnQ and unzip it. Open the `lab5` folder that you extracted in either CLion or VS Code (note: you need to make sure you open the correct folder, which is, the one that directly contains the code files). Then, complete the following tasks.

This lab is due 11:30 AM (end of the lab session) on Thursday, November 23 for students in all sections. The scheduling of lab sessions is as follows:

- Tuesdays 12:30–2:30
- Wednesdays 11:30–1:30
- Thursdays 9:30–11:30

2.1 Task 1: Insertion

Implement the function `bst_insert` to insert a new element into a binary search tree. The function returns a `bool` value, which should be `true` if the element was actually inserted and `false` if it was not (i.e., it was already present). The worst-case time complexity of the algorithm must be at most $\mathrm{O}(n)$, where n is the number of elements currently in the tree.

2.2 Task 2: Removal

Implement the function `bst_remove` to remove an element from a binary search tree. Similarly to Task 1, the function returns a `bool` value, which should be `true` if the element was actually removed and `false` if it was not (i.e., it was not present). The worst-case time complexity of the algorithm must be at most $\mathrm{O}(n)$, where n is the number of elements currently in the tree.

2.3 Task 3: Union

Implement the function `bst_union` to take the union of two binary search trees, viewing the trees as a representation of finite mathematical sets of values. The result should be a new tree which is independent of the input trees (i.e., one should be able to freely modify each tree without affecting the others). The worst-case time complexity of the algorithm must be at most $\mathrm{O}(n_1 n_2)$, where n_1 and n_2 are the number of elements currently in the tree.

2.4 Task 3: Intersection

Implement the function `bst_intersection` to take the intersection of two binary search trees, viewing the trees as a representation of finite mathematical sets of values. The result should be a new tree which is independent of the input trees (i.e., one should be able to freely modify each tree without affecting the others). The worst-case time complexity of the algorithm must be at most $O(n_1 n_2)$, where n_1 and n_2 are the number of elements currently in the tree.

3 Marking Criteria

After completing all tasks, call over a graduate TA to mark the lab. Lab 5 has 10 marks in total:

- Is your implementation of Task 1 correct? Does it meet the time complexity requirement? (2 marks)
- Is your implementation of Task 2 correct? Does it meet the time complexity requirement? (2 marks)
- Is your implementation of Task 3 correct? Does it meet the time complexity requirement? (2 marks)
- Is your implementation of Task 4 correct? Does it meet the time complexity requirement? (2 marks)
- Is your code sufficiently well-formatted and commented so that the purpose of each variable/field/parameter and the reason for each function call or data structure manipulation is clear? Refer to the guidance on the course assignment for expected levels of commenting. (2 marks)