Sabancı University
Faculty of Engineering and Natural Sciences

CS301 – Algorithms

Homework 3

Due: March 19, 2024 @ 23.55
(upload to SUCourse)

PLEASE NOTE: AyçaElif Aktaş –29802

- Provide only the requested information and nothing more. Unreadable, unintelligible, and irrelevant answers will not be considered.
- Submit only a PDF file. (-20 pts penalty for any other format)
- Not every question of this homework will be graded. We will announce the question(s) that will be graded after the submission.
- You can collaborate with your `TA/INSTRUCTOR ONLY` and discuss the solutions of the problems. However, you have to write down the solutions on your own.
- Plagiarism will not be tolerated.

Late Submission Policy:

- Your homework grade will be decided by multiplying what you normally get from your answers by a “submission time factor (STF)”.
- If you submit on time (i.e. before the deadline), your STF is 1. So, you don’t lose anything.
- If you submit late, you will lose 0.01 of your STF for every 5 mins of delay.
- We will not accept any homework later than 500 mins after the deadline.
- SUCourse’s timestamp will be used for STF computation.
- If you submit multiple times, the last submission time will be used.

Question 1

Consider a uniquely designed museum where rooms are arranged in a tree structure. Each room can have up to two child rooms connected by a path. Your task is to develop an algorithm to place a minimum number of security guards so that the entire museum is guarded. A guard placed in a room can guard that room, its parent room, and its direct child rooms.

(a) Develop an algorithm to find the minimum number of security guards required for any given museum structured as a standard binary tree. Analyze the worst-case time and space complexity of your algorithm.
Hint: Consider using DFS (for a bottom–up traversal of the rooms).

Answer:

To solve this problem we can use DFS to traverse the tree structure of the museum. The main idea is to start from leaf nodes and propagate upwards at each step we will determine if a room (node) needs a security guard classify each node into three states: { Guarded , Covered , Uncovered }

- Guarded: A security guard is placed in this room
- Covered: No guard is needed in this room because either a child or the parent room has a guard
- Uncovered: This room has no guard and not covered by any adjacent rooms

→ Recursively visit every node and decide guard placement based on states of the node's children.

```
def guard (node):
    if not node:
        return "covered"
    left_state = guard(node.left)
    righ_state  = guard (node.right)
    if left_state == "uncovered" or righ_state == "uncovered"
        guard_counter += 1
        return "Guarded"
    if left_state == "Guarded" or right_state == "covered"
        return "covered"
    return "uncovered"

guard_counter = 0
root_state = guard (root)
if root_state == "uncovered":
    guard counter += 1
```

root node requires special attention after whole traversal if root is uncovered need to place a guard

Time complexity :
The worst case time complexity is O(n) where n is number of rooms (nodes in tree). This is the case that every node is exactly visited once in DFS traversal

Space complexity : The worst case space complexity is O(h) where h is height of the tree. In worst case h can be n, leading to O(n) space complexity. But for a balanced tree the space complexity would be O (logn)

(b) Discuss the alterations needed in the algorithm, as well as the changes in worst-case time and space complexity when the museum structure is known to be a red-black tree.

Answer:
Red - Black trees are special kind of binary search tree with additional properties that ensure the tree remains balanced. For the given problem the approach of placing guards wouldn't change significantly because the solution does not rely on BST properties but rather tree structure. So algorithm would remain largely same however due to balancing properties of Red Black Trees which would effect worst-case time and space complexities.

Time complexity: The worst case time complexity remains O(n) because each node in tree still must be visited once

Space complexity: The space complexity benefits from the red-black tree properties. Because RBT are balanced with max height of 2 log(n+1) so the worst-case space complexity become O(log n)

(c) Given that each room has a number, from the viewpoint of a visitor intending to find/visit room X in the museum starting from the entrance room (i.e., root node), explain the differences experienced when the museum structure is a standard binary search tree versus a red-black tree.

Answer:
When navigating a museum structured as binary search tree or red black tree to find a specific room starting from entrance (root node) a visitor experience will be influenced based on characteristics of these tree structures.

The key difference for a visitor looking for a room lies on the "search efficency" and predictability. In standard BST the search efficiency can vary widely from logarithmic to linear, depending on tree's balance. In contrast RBT is self balancing properties guarantee more uniform and predictable and efficient search experience for the visitor with logarithmic search time. makes it faster and more efficient for visitor to find room X.

Question 2

We are given an array A with $2n+1$ distinct elements. Suppose that we are using the randomized selection algorithm to find the median. In a worst–case scenario of this algorithm, the median is found at the very last step, where each step before that gets rid of only one element in the array. How many different worst–case scenarios are there for finding the median in A?

Example: Suppose that we have $A = [4, 5, 1, 3, 2]$.

One of the worst–case scenarios is to pick the following elements as the pivots in this order $1, 2, 5, 4$. Because (randomly but unluckily) if we pick these numbers in this order as the pivots, we will get rid of only the pivot in each step, and only at the very end (when we have only element 3 remaining, which is the median of the original input array) we will find the median.

The other worst–case scenarios are:
1,5,2,4
1,5,4,2
5,1,4,2
5,1,2,4
5,4,1,2

Answer: In worst case scenario each pivot selection eliminates exactly one element until median is the only element left. The median by definition is the element in the middle position when array is sorted. It will not be chosen until very last step. The order in which elements other than median are chosen matters until we are left with the median.

Given $2n+1$ element, median positioned at $n+1$ when array sorted.

→ To reach the worst case scenario where median found at very last we need to select n elements from either side of the median (in the sorted view) as pivots in any order until pivot remains

→ Since selection of pivots defines a scenario each side of pivot can contribute independently we will calculate number of scenarios based on selecting pivots from left and right sides of the median.

left : $n!$ ways to pick n elements
right : $n!$ ways to pick n elements

However pivot selection involves choosing from either side without fixed sequence so we need to consider interleavings of these selections. A interleaving of $2n$ elements, there are $\binom{2n}{n}$ ways to interleave them ie choosing n positions out of $2n$

→ Total number of different worst case. scenarios are

$$\boxed{n! \cdot n! \cdot \binom{2n}{n}}$$

→ In our example A : $2!\ 2!\ \binom{4}{2} = 24$

Question 3

In the WCL Select algorithm, suppose that we modify the approach to partition the array into groups of size 2k+1 instead of the usual groups of 5. Write down the recurrence for the running time of the algorithm for this general case.

Answer:

→ partitioning: Divide the array into $n/_{2k+1}$ groups of 2k+1 elements each

→ finding medians: finding median of each of this groups takes O(1) time, performing this step takes $O(n/_{2k+1})$ time

→ median of medians: recursively apply algorithm to find median of these $n/_{2k+1}$ medians. If there are $m = n/_{2k+1}$ medians step recurrence is $T(m)$, $m = n/2k+1$

→ partitioning with pivot: use median of medians as pivot to partition the original array. This takes $O(n)$ time.

→ Recursively apply the algorithm to the appropriate partition containing the kth smallest element. The worst case size of the partition is reduced to a fraction of n, which we will determine based on the grouping strategy

→ For groups of 2k+1 we ensure that at least half of the groups contribute an element smaller or equal to (and another half greater or equal to) the median of medians. This guarantees that at least $1/2 \cdot \frac{n}{2k+1}(k+1)$ elements are less than or equal to the pivot (and similarly greater or equal) ensuring we discard significant fraction of the elements each time we partition

→ The recurrence relation considering partitioning and recursive selection can be:

$$\boxed{T(n) = T\left(\frac{n}{2k+1}\right) + T\left(n - \frac{n}{2(2k+1)}(k+1)\right) + O(n)}$$