ShanghaiTech University

CS101 Algorithms and Data Structures

Fall 2020

Homework 5

Due date: 23:59, October 19, 2020

1. Please write your solutions in English.
2. Submit your solutions to gradescope.com.
3. Set your FULL NAME to your Chinese name and your STUDENT ID correctly in Account Settings.
4. If you want to submit a handwritten version, scan it clearly. Camscanner is recommended.
5. When submitting, match your solutions to the according problem numbers correctly.
6. No late submission will be accepted.
7. Violations to any of the above may result in zero grade.
8. Problem 0 gives you a template on how to organize your answer, so please read it carefully.

Problem 0: Notes and Example

Notes

1. Some problems in this homework requires you to design Divide and Conquer algorithm. When grading these problems, we will put more emphasis on how you reduce a problem to a smaller size problem with Divide and Conquer strategy.
2. Your answer for these problems should include:
   (a) Algorithm Design
   (b) Time Complexity Analysis
3. In Algorithm Design, you should describe your algorithm for each step clearly.
4. Unless required, writing pseudocode is optional. If you write pseudocode, please give some additional descriptions if the pseudocode is not obvious.
5. You are recommended to do this homework with LaTeX.

Example: Binary Search
Given a sorted array a of n elements, write a function to search a given element x in a.

Algorithm Design: We basically ignore half of the elements just after one comparison.

1. Compare x with the middle element.
2. If x matches with middle element, we return the mid index.
3. Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.
4. Else (x is smaller) recur for the left half.

Pseudocode(Optional):
$left$ and $right$ are indexes of the leftmost and rightmost elements in given array a respectively.

```
1:  function BINARYSEARCH(a, value, left, right)
2:      if right < left then
3:          return not found
4:      end if
5:      mid ← ⌊(right − left)/2⌋ + left
6:      if a[mid] = value then
7:          return mid
8:      end if
9:      if value < a[mid] then
10:         return binarySearch(a, value, left, mid-1)
11:     else
12:         return binarySearch(a, value, mid+1, right)
13:     end if
14: end function
```

Time Complexity Analysis: During each recursion, the calculation of mid and comparison can be done in constant time, which is $O(1)$. We ignore half of the elements after each comparison, thus we need $O(\log n)$ recursions.

$$T(n) = T(n/2) + O(1)$$

Therefore, $T(n) = \log n$

1: (2'+1'+1') Single Choice

The following questions are single choice questions, each question has **only one** correct answer. Select the correct answer.
Note: You should write those answers in the box below.

Question 1	Question 2	Question 3
B;A	B	A

Question 1. *What is the common data structure used in implementation of* ***Breadth First Search****?* _____
What is the common data structure used in implementation of ***Depth First Search****?* _____

(A) Stack

(B) Queue

(C) Hash Table

(D) Tree

Question 2. *Given the following pseudo-code, which kind of traversal is it doing?*

```
function order(node) {
  if node has left child then
    order(node.left)
  end if
  if node has right child then
    order(node.right)
  end if
  visit(node)
}
```

(A) Preorder traversal

(B) Postorder traversal

(C) Inorder traversal

(D) None of above

Question 3. *Which traversal strategy should we use if we want to print the hierachical structure ?*

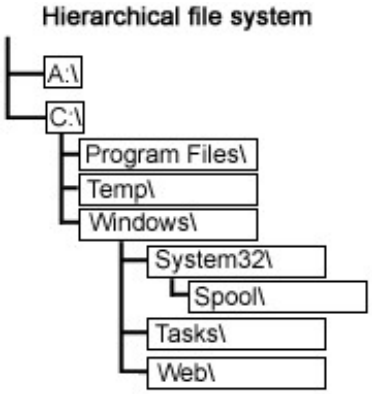


Figure 1: an example of hierachical structure

(A) Preorder traversal

(B) Postorder traversal

(C) Inorder traversal

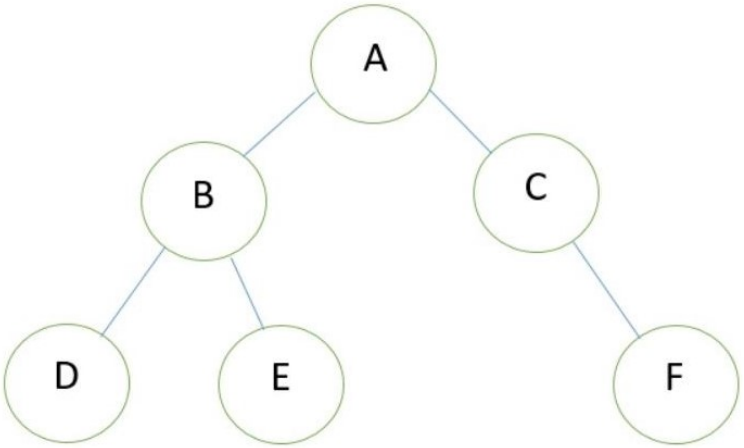
(D) None of above

2: (5') Running BFS & DFS

Please run BFS and DFS(preorder) on the following tree.

Note:

1. *You should select proper data structure for BFS and DFS first (1')*
2. *Please show what is currently in your data structure at each step (2')*
3. *When you want to put more than one child of some node onto your data structure, please push/enqueue them alphabetically.*
4. *Write down the order of sequence after you finish running BFS and DFS. (2')*



(a) BFS

(b) DFS

BFS	queue					
front	A	B	C	D	E	F
		C	D	E	F	
			E			
back						
sequence	ABCDEF					

推荐写法	(可以保持先左后右的遍历顺序)					
DFS preorder	stack					
top						
bottom	A	B	D	E	C	F
sequence	ABDECF					

不推荐写法	(本次hw给分)					
DFS preorder	stack					
top						
		C	F		E	
bottom	A	B	B	B	D	D
sequence	ACFBED					

3: (5')Skyline problem

We can abstract a street into a straight line and assume that the bottoms of all the buildings on one side of the street lie on the x-axis, and the two-dimensional skyline of the buildings can be drawn as follows:

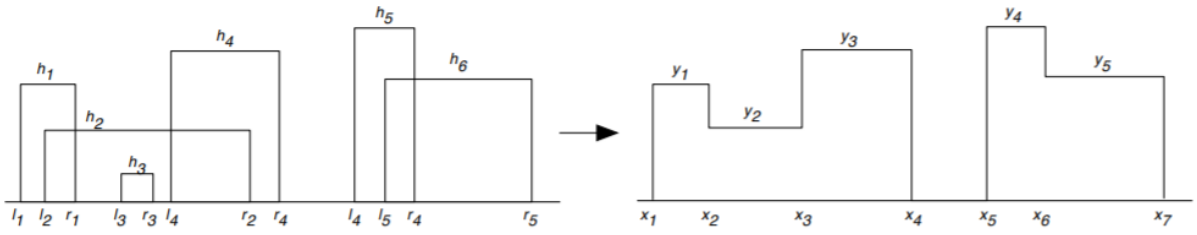


Figure 2: Skyline problem

Building B_i is represented by a triple (l_i, h_i, r_i), where l_i and r_i denote the left and right x coordinates of the building, respectively, and h_i denotes the building's height. A skyline is composed of x coordinates and the heights in the following format:

$$(x_1, y_1, x_2, y_2, x_3 ...)$$

meaning that at x_1 we draw a building at height y_1 until x_2 at which point we draw a line up or down to high y_2 and then continue horizontally until x_3 and so on.

(a) Firstly, let's see how to merge. There are two skylines, one is composed of n buildings, and the other is composed of m buildings. For the $m+n$ buildings, you need to show how to merge the two skylines into one new skyline in $O(m+n)$ steps.

(b) Now, let's design a complete algorithm. There are n buildings on one side of the street, you need to design an algorithm to output their skyline. The time complexity limit is $O(n \log n)$.

Solution:

(a) We are given two skylines A and B, and we want to merge them in linear time. Let x be the next x-coordinate, in order. It can be from A or from B (or both).
We'll keep track of the current skyline height, crtH. Initially crtH=0.
The basic idea is that the skyline will be the larger of A and B.
When we look at x, two things can happen:

- if $height(x) > crtH$ then the skyline jumps up. We output $(x, height(x))$ to the skyline, and update $crtH = height(x)$.
- if $height(x) > crtH$ then the skyline drops. It can drop to $height(x)$ or to the last height in the other skyline, whichever is larger.

Overall, each x can be handled in O(1) time.

(b) To get an $O(n \log n)$ algorithm for finding the skyline of n buildings we use divide-and-conquer. This is divide-and-conquer at its best — simple and elegant!!
Skyline(n buildings)

- if $n == 1$ return the building

- A =Skyline(the first n/2 buildings)
- B =Skyline(the last n/2 buildings)
- merge A and B as in (a) above and return merged skyline

The runtime is bounded by the reccurrence:

$$T(n) \leq 2T(n/2) + O(n),$$

which implies that $T(n) = O(nlogn)$.
Note: The beauty of it is that you can do this part without having done part (a) — it's sufficient to know that it is possible to merge two skylines in linear time

4: (5') Finding Majority

Suppose you're a TA for the CS101 course, and you need to help the TA group solve the following problem. They have a collection of n students' code solutions for the programming task, suspecting them of academic plagiarism. It's difficult to judge whether two code solutions are equivalent, but the TA group has a high-tech "plagiarism detection machine" that takes two code solutions and determines whether they are equivalent after performing some operations.

Their question is the following: among the collection of n code solutions, is there a set of more than $n/2$ of them that are all equivalent to one another? Assume that the only feasible operations you can do with these code solutions are to pick two of them and plug them in to the plagiarism detection machine. Show that, with the help of the plagiarism detection machine, how to decide the answer to their question with only $O(n \log n)$ invocations of the machine.

Solution:

Via divide and conquer:

Let $e_1, \cdots, e_n$ denote the equivalence classes of the solutions: solution i and j are equivalent if $e_i = e_j$. What we are looking for is a value x so that more than $n/2$ of the indices have $e_i = x$.

Divide the set of code solutions into two roughly equal piles: a set of $\lfloor n/2 \rfloor$ students and a second set for the remaining $\lceil n/2 \rceil$ studetns. We will recursively run the algorithm on the two sides, and will assume that if the algorithm finds an equivalence class containing more than half of the code solutions, then it returns a sample value in the equivalence class.

Note that if there are more than $n/2$ students'solutions that are equivalent in the whole set, say have equivalence class x, then at least one of the two sides will have more than half the solutions also equivalent to x. So at least one of the two recursive calls will return a solution that has equivalence class x.

The correctness of the algorithm follows from the observation above: if there is a majority equivalence class, then this must be a majority equivalence class for at least one of the two sides. To analyze the running time, let $T(n)$ denote the maximum number of tests the algorithm does for any set of n solutions. The algorithm has two recursive calls, and does at most $2n$ tests outside of the recursive calls. So we get the following recurrence (assuming n is divisible by 2):

$$T(n) \leq 2T(n/2) + 2n$$

The recurrence implies that $T(n) = O(nlogn)$.

In linear time:

Pair up all solutions, and test all pairs for equivalence. If n was odd, one student is unmatched. For each pair that is not equivalent, discard both. For pairs that are equivalent, keep one of the two. Keep also the unmatched students, if n is odd. We can call this subroutine ELIMINATE.

The observation that leads to the linear time algorithm is as follows. If there is an equivalence class with more than $n/2$ solutions, then the same equivalence class must also have more than half of the solutions after calling ELIMINATE. This is true, as when we discard both solutions in a pair, then at most one of them can be from the majority equivalence class. One call to ELIMINATE on a set of n students takes $n/2$ tests, and as a result, we have only $\leq \lceil n/2 \rceil$ students left. When we are down to a single solution, then its equivalence is the only candidate for having a majority. We test this solution against all others to check if its equivalence class has more than $n/2$ elements.

This method takes $n/2 + n/4 + \cdots$ tests for all the eliminates, plus $n - 1$ tests for the final counting, for a total of less than $2n$ tests. Therefore the total time complexity is $O(n)$.

5: (5') Polynomial Multiplication

Design an efficient algorithm for the following problem:
Input: n numbers $\{a_1, \cdots, a_n\}$
Goal: Compute the polynomial with $\{a_1, \cdots, a_n\}$ as its roots. In other words, compute coefficients $\{b_0, \cdots, b_n\}$ so that $(x - a_1) \cdot (x - a_2) \cdots (x - a_n) = b_0 + b_1 x + \cdots + b_n x^n$.
(Hint: Try divide and conquer & use $O(nlogn)$ time **polynomial multiplication algorithm** as a blackbox)

Describe your algorithm(3'), write the recurrence for the running time(1') and solve the recurrence to compute the time complexity(Do not only write the answer)(2').

Solution:

Pseudocode:

procedure PolynomialWithRoots($\{a_1, \ldots, a_n\}$):

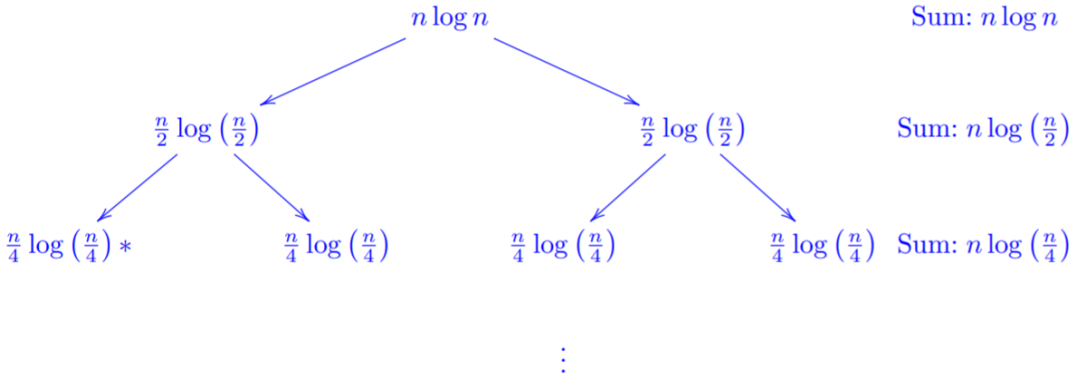
if n = 1 **then**
  $b_0 \leftarrow -a_1$
  $b_1 \leftarrow 1$
  return b_0, b_1
end if
$q(x) \leftarrow$ PolynomialWithRoots($\{a_1, \ldots, a_{\lfloor n/2 \rfloor}\}$)
$r(x) \leftarrow$ PolynomialWithRoots($\{a_{\lfloor n/2 \rfloor}, \ldots, a_n\}$)
$p(x) \leftarrow$ MultiplyPolynomials($q(x)$, $r(x)$)
return coefficients of $p(x)$

The recurrence formula:

$$T(n) = 2T(\frac{n}{2}) + O(n \log n)$$

The time complexity is T(n)=$O(n \log^2 n)$.

To show this, we use the recurrence tree:



The total, then, is

$$
\begin{aligned}
& n \log n + n \log \left(\frac{n}{2}\right) + \log \left(\frac{n}{4}\right) + \cdots \\
&= n \left(\log n + \log \left(\frac{n}{2}\right) + \log \left(\frac{n}{4}\right) + \cdots\right) \\
&= n \left(\log n + (\log n - \log 2) + (\log n - \log 4) + \cdots\right) \\
&= n \left((\log n + \log n + \cdots + \log n) - (\log 2 + \log 4 + \cdots + \log n)\right) \\
&= n \left((\log n)^2 - (1 + 2 + \cdots + \log n)\right) \\
&\approx n \left(\log^2 n - \frac{\log^2 n}{2}\right) \\
&= \frac{n \log^2 n}{2} \\
&= \Theta \left(n \log^2 n\right)
\end{aligned}
$$

We could also use a more general version of the Master Theorem, which states that if $T(n) = aT(n/b) + f(n)$, where $a \geq 1, b > 1$, and $f(n) = \theta(n^c \log^k n)$ where $c = log_b a$, then $T(n) = \theta(n^c \log^{k+1} n)$. In this case, $a = b = 2$, $c = 1 = \log_b a = \log_2 2$, and $k = 1$.