Data Structures & Algorithms MidTerm Class Test Solutions

Class Test 2020/21

Data Structures & Algorithms MidTerm Class Test

Note

Answer ALL questions. Each question will be marked out of 20. The paper will be marked out of 61, which will be rescaled to a mark out of 100.

Question 1

Stacks are often used to implement *"undo"* operations in applications like text editors or Web browsers such that making a change in the text or clicking on a link in the browser pushes a record of that operation on to the stack, while invoking undo pops the record off the stack and executes the actions necessary to undo the operation.

In theory, an unbounded stack can provide unlimited undo operations, but, to save memory, many applications support only limited undo history. Thus invoking *"push"*, when the stack is already at full capacity, accepts the pushed elements at the top of the stack and *leaks* the oldest element from the bottom of the stack rather than throwing an exception, while invoking `pop` when the stack is empty still throws a `StackEmptyException`.

(a) Describe, as clearly and simply as possible, your proposed strategy to implement such a stack using a fixed-capacity array, so that each update operation runs in $O(1)$ time. **[6 marks]**

(b) Write the pseudocode for the `push` operation of such a stack. **[7 marks]**

(c) Write the pseudocode for the `pop` operation of such a stack **[7 marks]**

Model answer:

(a) The idea here is to use a circular array *[This was covered in the module as a possible implemention of a queue but was never mentioned in the context of stacks]*.

The stack is maintained in an array of some capacity MAXSTACK. A size variable is maintained to keep track of the total number of elements in the stack at any time. A top variable is maintained to keep track of the index in the array where the next value to be pushed on to the stack will be saved to.

As values are pushed onto the stack the top index is incremented modulo MAXSTACK but the size variable is incremented only if the stack is not full.

When attempting to pop a value off the stack, an error is reported if the stack is empty. Otherwise the top index is decremented modulo MAXSTACK and the size variable is decremented.

(b)
```
// Initialize an empty leaky stack
stack = new int[MAXSTACK];
top = 0;
size = 0;

void push (int val)
{
  stack[top] = val;
  top = (top + 1) mod MAXSTACK
  if (size < MAXSTACK)
    size ++;
}
```

(c)
```
int pop ()
{
  if (size == 0)
  {
     throw StackEmptyException;
  }
  size--;
  top = (top + MAXSTACK - 1) mod MAXSTACK;
  return stack[top];
}
```

Question 2

An inefficient recursive function `isbst` was presented in the module in the handout document (pages 46–47) and the slides (dsa-slides-04-03-binary-search-trees.pdf).

Write an efficient, recursive `isbst` function in pseudocode to check that a binary tree is a valid Binary Search Tree based on checking that subtrees are within closed intervals starting with [MIN_INT, MAX_INT] and calculate the complexity, in terms of $O(g(n))$, with respect to the size of the tree. Justify your calculation of the complexity with a short explanation. **[20 marks]**

Model answer:

[For full correctness, getting the comparison operators as $\leq$ rather than $<$ and excluding the value at the root of the sub-tree at each level, is important to allow the full range of integers in the tree. Note that the definition of BSTs in the module excludes duplicate values.]

```
1  isbst(tree t)
2  {
3     return isbst(t, MIN_INT, MAX_INT)
4  }
5
6  isbst(tree t, int min, int max)
7  {
8     if ( isEmpty(t) )
9       return true
10    else
11      return ( min <= value(t) and
12               value(t) <= max and
13               isbst( left(t), min, value(t)-1 ) and
14               isbst( right(t), value(t) + 1, max) )
15 }
```

This function checks the value at every node exactly once and recursively walks the entire tree, visiting each node once, so the complexity is $O(n)$

Question 3

Construct an AVL tree of positive integers and of height 3 such that insertion of the value 7 will trigger a double rotation (i.e. a right rotation followed by a left rotation or a left rotation followed by a right rotation) and, following the insertion of 7, an insertion of the value 8 will also trigger a double rotation.

(a) Draw the valid AVL tree before the specified insertions. **[6 marks]**

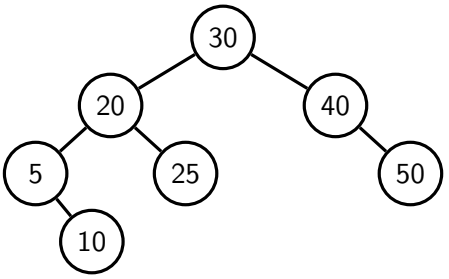
(b) Draw the valid AVL tree after insertion of 7 but before insertion of 8. **[7 marks]**

(c) Draw the valid AVL tree after insertion of 7 and of 8 in that order **[7 marks]**

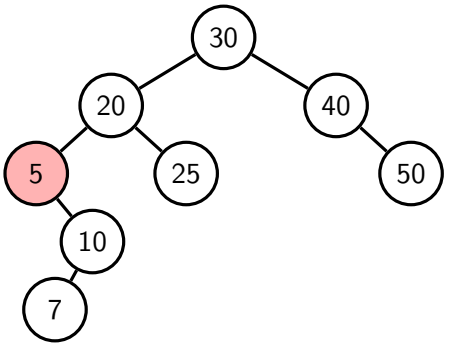
Model answer:

[This is one example of many possible].

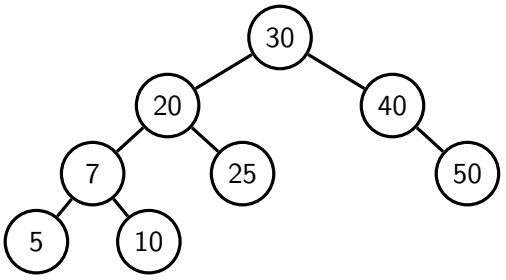
Answer (a)
A. Before insertion of
7 and 8



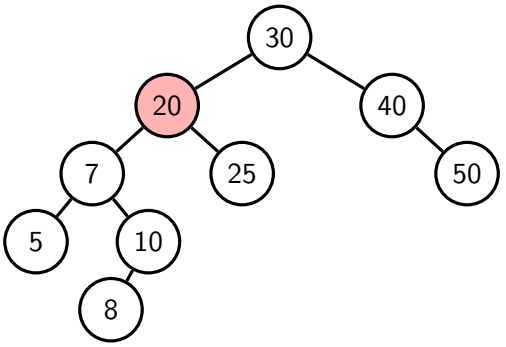
B. After inserting 7,
before double rotation



Answer (b)
C. After double rot at 5
before inserting 8



D. After inserting 8
Before double rotation



Answer (c)
E. After double
rotation at 20