IB Computer Science Revision Notes

Topic 5 - Abstract data structures

5.1 Abstract data structures

Thinking Recursively

5.1.1 Identify a situation that requires the use of recursive thinking

A recursive solution is one where the problem is broken up in to a smaller sub-problem repeatedly until the problem can be easily solved. Each successively larger problem is then solved using the solution to the smaller sub problem. This type of thinking can be used in many problems and sometimes (but not always) it is the easiest method of solving the problem. Some examples of recursion that are commonly used in Computer Science are binary tree traversals and the Towers of Hanoi problem. In general any problem that can be broken up into a sub-problem and a base case (the smallest possible sub-problem) can or may require the use of recursive thinking.

5.1.2 Identify recursive thinking in a specified problem solution

5.1.3 Trace a recursive algorithm to express a solution to a problem

A common basic example of a recursive algorithm is the algorithm to return the factorial of a number. The factorial is the product of all integers from 1 up to the number. For example 4! = 1 x 2 x 3 x 4 = 24.

```
private int factorial(int number){
  if(number < 1){return 0;}
  else if(number == 1){return 1;}
  else {return factorial(number-1)*number;}
}
```

Here is the trace table for the above algorithm for the number 4:

Number	Stack contains	Note
4	4	After calling the function, the return statement waits for a number to be returned from the function called
3	4, 3	
2	4, 3, 2	
1	4, 3, 2, 1	This is returned to the function which called it which multiplies every number in the stack

Abstract Data Structures

5.1.4 Describe the characteristics of a two-dimensional array

A two-dimensional array is an array of arrays. This means that each entry in the first array is an array that contains data. The set-up is similar to that of a table. The whole table is an array and every row within the table is one of the secondary arrays.

Array A			
Array 1	Entry A.1.1	Entry A.1.2	Entry A.1.3
Array 2	Entry A.2.1	Entry A.1.2	Entry A.2.3
Array 3	Entry A.3.1	Entry A.1.2	Entry A.3.3
Array 4	Entry A.4.1	Entry A.1.2	Entry A.4.3

Like a table, each entry can be accessed in a method similar to a coordinate system. First the number for the position in the outer array, then the number for the position of the entry in the inner array.

5.1.5 Construct algorithms using two-dimensional arrays

5.1.6 Describe the characteristics and applications of a stack

A stack is a data list structure where items are added and removed from the same end. This is called LIFO (last in, first out). A stack of plates in a cafeteria is a great example of this. Plates are added and taken off from only the top. In computer programming a stack is often used in recursive programs where each result is put in the stack until the base case is reached when the items are taken off the stack in the same order they were out on.

5.1.7 Construct algorithms using the access methods of a stack

5.1.8 Describe the characteristics and applications of a queue

A queue is a data list structure where items are added from 1 end and removed from the other end. This is called FIFO (first in, first out)

5.1.9 Construct algorithms using the access methods of a queue

5.1.10 Explain the use of arrays as static stacks and queues

Linked List

5.1.11 Describe the features and characteristics of a dynamic data structure

5.1.12 Describe how linked lists operate logically

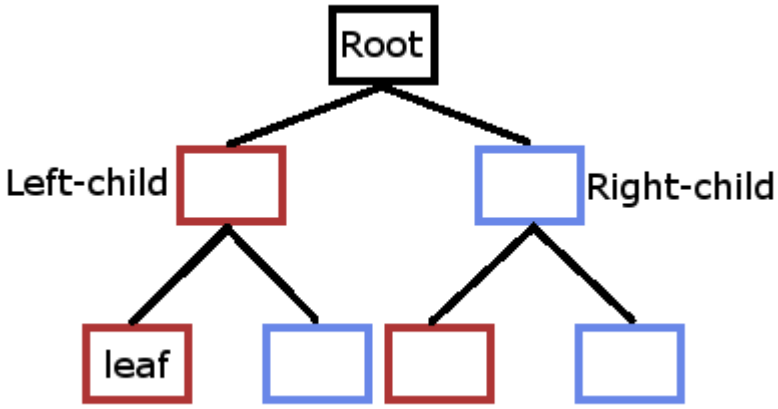
5.1.13 Sketch linked lists (single, double and circular)

Trees

> You are only expected to be able to work with trees in diagrams and not in pseudocode.

5.1.14 Describe how trees operate logically (both binary and non-binary)

5.1.15 Define the terms: parent, left-child, right-child, subtree, root and leaf

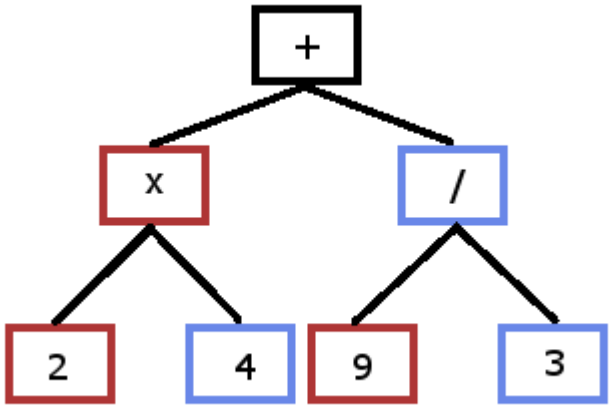


- **Parent:** A node that has one or more children is a parent node.
- **Left-child:** The child node that is to the left of the parent. In binary trees the node is less than the parent node.
- **Right-child:** The child node that is to the right of the parent. In binary trees the node is greater than the parent node.
- **Subtree:** A portion of another tree that is a tree itself.
- **Root:** A node that has no parent, it is at the top of the tree.
- **Leaf:** A node that has no children, they are at the bottom of the tree.

Left-child and right-child apply only to binary trees. All the other definitions apply to all trees.

5.1.16 State the result of in-order, postorder, and preorder tree traversal

All of the traversals work with a recursive algorithm that is applied from each node, starting with the root.



In-order: The in-order traversal of the above tree will result in `2 x 4 + 9 / 3`. The in-order traversal means that the parent node is in between the two children nodes.

Postorder: The postorder traversal of the above tree will result in `2 4 x 9 3 / +`. Post order traversal means that the parent node comes after the children nodes.

Preorder: The preorder traversal of the above tree will result in `+ x 2 4 / 9 3`. Pre order traversal means that the parent node comes before the children nodes.

5.1.17 Sketch binary trees

Sketching binary trees is very easy if you know how the tree is supposed to look. Just make sure that each node has no more than 2 children. If a node is the only child of its parent then it cannot have any children itself. If you don't know how the tree is supposed to look then it will take a bit more work to sketch.

Application

5.1.18 Define the term dynamic data structure

5.1.19 Compare the use of static and dynamic data structures

5.1.20 Suggest a suitable structure for a given situation

---

topicfive.txt · Last modified: 2018/03/04 00:00 (external edit)