OCR Computer Science A Level

1.4.2 Data Structures

Concise Notes

Specification

1.4.2 a)

- Arrays
- Records
- Lists
- Tuples

1.4.2 b)

- Linked List
- Graphs
- Stack
- Queue
- Tree
- Binary Search Tree
- Hash Table

1.4.2 c)

- Traversing data structures
- Adding data and removing data from data structures

Arrays, Records, Lists, and Tuples

Arrays
- An array is an ordered, finite set of elements of a single type.
- A 1D (one- dimensional) array is a linear array.
- A 2D (two-dimensional) array can be visualised as a table/spreadsheet.
- When searching an array, first go down the rows and then across the columns.
- A 3D (three-dimensional) array can be visualised as a multi-page spreadsheet
    - An element in a 3D array using: `threeDimensionalArray[z,y,x]`
      z = array number, y = row number, x = column number.

Records
- More commonly referred to as a row in a file,
- A record is made up of fields, and is widely used in databases.

Lists
- Consists of a number of items, where items can occur more than once.
- Data can be stored in non-contiguous locations and be of more than one data type.

Tuples
- An ordered set of values of any data type .
- Cannot be changed: elements cannot be added, edited or removed once initialised.
- Initialised with regular brackets rather than square brackets

Linked Lists, Graphs, Stacks, Queues, and Trees

Linked Lists
- Dynamic data structure used to hold an ordered sequence
- Items do not have to be in contiguous data locations
- Each item is called a node, and contains a data field and a link or pointer field.
- Data field: contains the actual data associated with the list
- Pointer field: contains the address of the next item in the list

Graphs
- Set of vertices/nodes connected by edges/arcs. ,
    - Directed Graph: Edges can only be traversed in one direction
    - Undirected Graph:Edges can be traversed in both directions,
    - Weighted Graph: Each arc has a cost attached to it
- Implemented using an adjacency matrix or an adjacency list.

Advantages of using Adjacency Matrix	Advantages of using Adjacency List
Convenient to work with	Space efficient for large sparse networks
Easy to add nodes	

Stacks

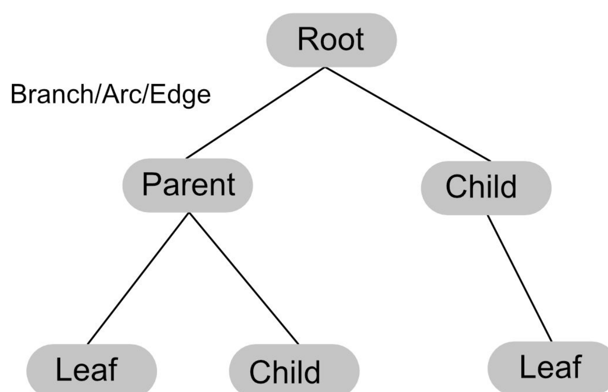
- Last in first out (LIFO) data structure:
    - Items can only be added to/ removed from the top of the stack.
- Used to reverse actions, eg. back buttons and undo buttons use stacks
- Can be implemented as a static or dynamic structure.

Queues

- First in first out (FIFO) data structure:
    - Items are added to the end and are removed from the front of the queue.
- Used in printers, keyboards and simulators.
- Linear queue: items are added into the next available space, starting from the front.
    - Items are removed from the front of the queue
    - Uses two pointers: pointing to the front and back of the queue.
    - Use space inefficiently, as positions from which data has been removed cannot be reused
- Circular queues have a rear pointer that can loop back to the front of the queue and utilise empty space at the front.
    - Are harder to implement.

Trees

- A connected graph, with a root and child nodes.
- Node: Item in the tree,
- Edge: Connects two nodes together and is also called a branch/arc
- Root: Node with no incoming nodes
- Child: Node with incoming edges
- Parent: Node with outgoing edges
- Subtree: Section of a tree consisting of a parent and its children
- Leaf: Node with no children.



- A binary tree is a type of tree where each node has a maximum of two children.
- Store information in a way that is easy to search through
- Commonly represented by storing each node with a left pointer and a right pointer.

Hash Tables

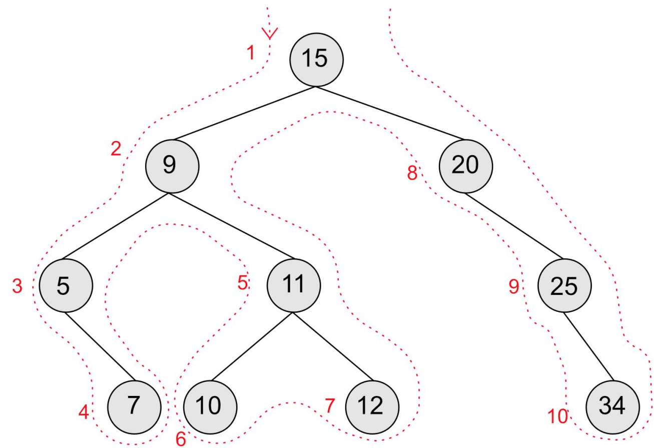
- An array coupled with a hash function.
- Hash function takes in data (a key) to produce a unique output (the hash).
- Typically used to map the key to a unique index in the hash table.
- Two keys producing the same hashed value is called a collision
- If this occurs, the item is typically placed in the next available location.
- A good hashing algorithm should have a low rate of collisions.

Traversing Data Structures

Pre-order Traversal

Pre-order traversal follows the order: root node, left subtree, then right subtree.
Using the outline method, nodes are traversed in the order in which you pass them on the left, beginning at the left-hand side of the root node.

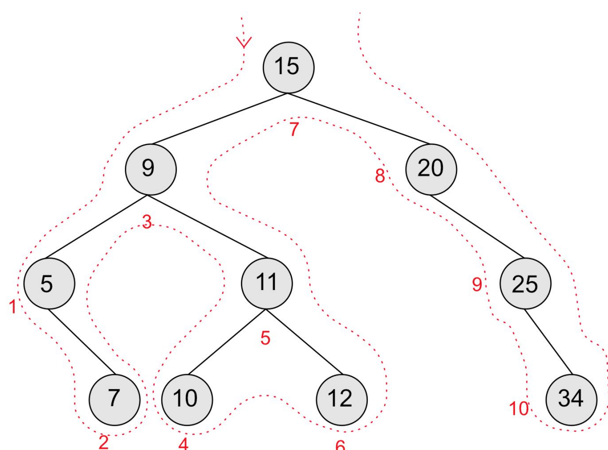
The order of traversal is: 15, 9, 5, 7, 11, 10, 12, 20, 25, 34



In-order Traversal

In-order traversal follows the order: left subtree, root node, right subtree.
Using the outline method, nodes are traversed in the order in which you pass under them, beginning at the first node from the left which does not have two child nodes.
This is useful for traversing the nodes in sequential order by size.

Order: 5, 7, 9, 10, 11, 12, 15, 20, 25, 34

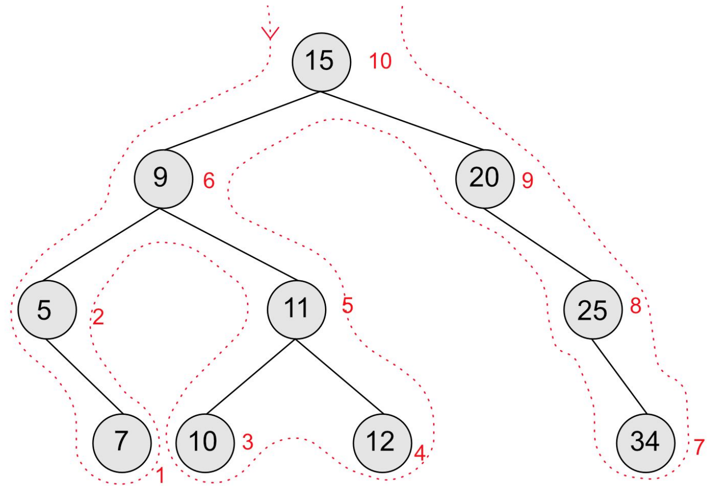


Post-order Traversal

Post order traversal follows the order: left subtree, right subtree, root node.

Using the outline method, nodes are traversed in the order in which they are passed on the right, beginning at the left of the root node.

Order: 7, 5, 10, 12, 11, 9, 34, 25, 20, 15



Manipulating Data Structures

Lists

List Operations	Description
isEmpty()	Checks if the list is empty
append(value)	Adds a new value to the end of the list
remove(value)	Removes the value the first time it occurs in the list
search(value)	Searches for a value in the list.
length()	Returns the length of the list
index(value)	Returns the position of the item
insert(position, value)	Inserts a value at a given position
pop()	Returns and removes the last item in the list
pop(position)	Returns and removes the item at the given position

Queues

Queue Operations	Description
enQueue(value)	Adds a new item to the end of the queue
deQueue()	Removes the item from the front of the queue

isEmpty()	Checks if the queue if empty
isFull()	Checks if the queue is full

Linked List

The following procedure is used to add the word ‘OCR’ after the word ‘Example’:

Index	Data	Pointer
0	‘Linked’	2
1	‘Example’	0
2	‘List’	-
3		

1. Add the new value to the end of the linked list and update the ‘NextFree’ pointer.

3	‘OCR’	

Start = 1 NextFree=4

2. The pointer field of the word ‘Example’ is updated to point to ‘OCR’, at position 3.

1	‘Example’’	3

3. The pointer field of the word ‘OCR’ is updated to point to ‘Linked’, at position 0.

3	‘OCR’	0

4. When traversed, this linked list will now output ‘Example’, ‘OCR’, ‘Linked’, ‘List’.

The following procedure is used to remove the word ‘List’ from the original linked list:

1. Update the pointer field of ‘Example’ to point to ‘List’ at index 2.

0	‘Linked’	2

1	‘Example’	2
2	‘List’	-

2. When traversed, this linked list will now print ‘Example’, ‘List’.

Stacks

Stack Operations	Example	Description
isEmpty()	`Stack.isEmpty()` `>> True`	Checks if the stack is empty. Works by checking the value of the top pointer.

push(value)	`Stack.append(“Nadia”)`
	`>>`
	`Stack.append(“Elijah”)`
	`>>`
		Adds a new value to the end of the list. Needs to check that the stack is not full before pushing to the stack.

peek()	`Stack.peek()` `>> “Elijah”`	Returns the top value from the stack. First checks the stack is not empty by looking at value of top pointer.
pop()	`Stack.pop()` `>> “Elijah”`	Removes and returns the top value of the stack. First checks the stack is not empty by looking at value of top pointer.
size()	`Stack.size()` `>> 2`	Returns the size of the stack
isFull()	`Stack.isFull()` `>> False`	Checks if the stack if full and returns a Boolean value. Works by comparing stack size to the top pointer.