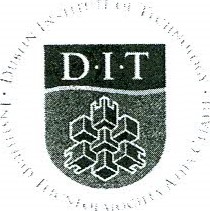
S228/212C, S282/212C



DUBLIN INSTITUTE OF TECHNOLOGY



DT228 BSc. (Honours) Degree in Computer Science

Year 2

DT282 BSc. (Honours) Degree in Computer Science

(International)

Year 2



SUMMER EXAMINATIONS 2016/2017



ALGORITHM DESIGN AND DATA STRUCTURES [CMPU20011

MR. RICHARD LAWLOR

DR. DEIRDRE LILLIS

MR. PAUL COLLINS

FRIDAY 12TH MAY 4.00 P.M. - 6.00 P.M.

Two HOURS

ATTEMPT THREE OUT OF FOUR QUESTIONS.

ALL QUESTIONS CARRY EQUAL MARKS.

ONE COMPLEMENTARY MARK FOR THE PAPER.

PAGE OF

1. (a) Write a simple Java interface to express the services provided by the Abstract Data Type (ADT) Queue. You can assume the queue stores int values.

(5 marks)

***Interface Queue***

***{***

***Public void enQueue(int x);***

***Public int deQueue();***

***Public Boolean isEmpty();***

***}***

* 1. Provide a Java class which implements the Queue interface based on a linked list implementation but showing only data structure, constructor and no other methods.
     1. marks)

***class QueueLinkedList implements Queue***

***{***

***LinkedList head, tail;***

***Int size;***

***Public QueueLinkedList()***

***{***

***Head = tail = null;***

***Size = 0;***

***}***

***}***

* 1. Give the implementation of the method deQueue() for the class in part (b). Explain why you might use or not use a tail reference in your implementation. What is the complexity of deQueue() with and without the use of a tail pointer?

<http://gmetrics.sourceforge.net/gmetrics-CyclomaticComplexityMetric.html>

(9 marks)

Int deQueue()

{

If (!(isEmpty()))

{

Int toReturn = head.data;

Head = head.next;

Size--;

Return toReturn;

}

Return -1;

}

I would use a tail to keep track of the position of the contents of the queue as it would be visually easier to find any part of the queue by starting and the head and working till you get to the tail

The complexity of the either usage is O(2) as you are checking if the queue is empty, then doing another check on if the first is false. That is what is being done for both of them.

* 1. Provide a partial circular buffer implementation of ADT Queue. Show the data structure used and the code for deQueue() only.

(9 marks)

Class circularBuffer Implements Queue

{

Int queue[];

Int head, tail;

Public int deQueue()

{

If (!(isEmpty()))

{

Int toReturn = Queue[head++];

Return toReturn;

}

Return -1;

}

}

* 1. Write down a simple equation for a Stack in terms of pop() and push() that expresses the last-in-first-out (LIFO) behaviour of the Stack.
     1. marks)

The simple equation for expressing an stack in terms of using pop() and push() is the following: pop() ==> element = stack[size--] && push() ==> stack[++size] = element

1. (a) Using diagrams, show the detailed workings for the first two iterations of the outer loop of a bubble sort on the following array.



(4 marks)

* 1. Explain what is meant by Tortoises and Hares with reference to bubble sort. Provide a simple example to illustrate.

(4 marks)

* 1. Write an adaptation of bubble sort in pseudocode which helps with the problem of Tortoises and Hares.

(10 marks)

* 1. Show how your algorithm from part (c) sorts the above array.
     1. marks)
  2. Using diagrams, show the array in part (a) can be converted to a heap.
     1. marks)
  3. Using diagrams, show how heap sort works on the heapified array from part (d).
     1. marks)

1. (a) Provide a brief explanation on of how Kruskal's MST algorithm works and then write the algorithm in pseudocode.

(8 marks)

Kruskal Algorithm is a greedy algorithm that builds a minimum spanning tree. It does this by reading each edge and connects the smaller edges to their vertices. This could lead to a few different trees built. Eventually all vertices are connected in a single tree using the smallest edges possible.

KRUSKAL(G):

T = ∅

foreach v ∈ G.V:

MAKE-SET(v)

foreach (u, v) ordered by weight(u, v), increasing:

if FIND-SET(u) ≠ FIND-SET(v):

T = T ∪ {(u, v)}

UNION(u, v)

return T

* 1. What representation of a graph is appropriate for implementing Kruskal's algorithm? What is the complexity of creating this data structure and removing the edges as

required from it?

* + 1. marks)

Adjacency List or Adjacency Matrix are both appropriate representation for implementing Kruskal algorithm. The time complexity is O(E log V) where E is the number of edges and V is the number of vertices.

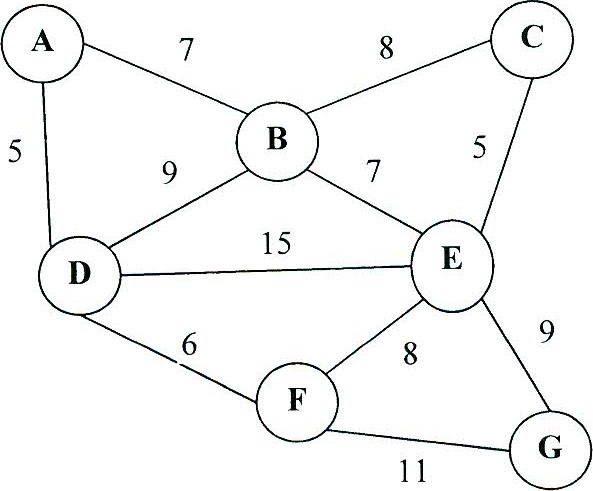
* 1. Explain the Union-Find data structure and what it is used for in Kruskal's algorithm. Also, with the aid of diagrams, outline a possible implementation of this data structure and give an example showing how its two significant operations work.

(10 marks)

The Union-Find is a data structure that connects two different sets into one, or if more than two sets it will eventually connect them all into one. When in Kruskal, it will treat each vertex as a member of individual disjoint set, joining them all until there is only one set, containing all the vertices and their respective edges.

Lets say there is 4 vertices, called A B C and D. There is an edge with a weight of 7 connecting A and B, another weighted 4 connected A and C, another weighted 2 connected B and D and another weighted 7 connected B and C. With Union-|Find, each vertex are a disjoint set. Then going through each edge, it checks if they are member of the same set by using the find() on them, finding the parent of each set. If the parent of the two sets are different, it will join them into one set and set the parent to represent the set for both members.

* 1. **Illustrate** in detail how Kruskal's algorithm computes a MST for the graph below **showing the contents of the union-find sets at each stage. *(Draw the Kruskal Tree algorithm on the graph below as well as draw the union-find)***



(10 marks)

1. (a) **Show** how binary search works when searching for 17 in the following array: ***(Draw using algorithm)***

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | 10 | 12 | 14 | 17 | 21 |

(5 marks)

* 1. What is a binary search tree (BST)? Mention any specific advantage or possible disadvantage. What is the complexity of searching a BST?

(6 marks)

A binary search tree is a binary tree storing keys (or key-element pairs) at its internal nodes and satisfying the following property: Let u, v, and w be three nodes such that u is in the left subtree of v and w is in the right subtree of v. We have key(u) < key(v) < key(w).

The advantage is the speed of finding an specific ordered item is greater then most of other search methods. Based off its algorithm, the number gets chopped in half everytime we search for an item.

findElement takes O(log n) time, using binary search

* 1. Write in pseudocode the algorithm for searching a BST.

(6 marks)

Algorithm findElement(k, v)

if T.isExternal (v)

return NO\_SUCH\_KEY

if k < key(v)

return findElement(k, T.leftChild(v))

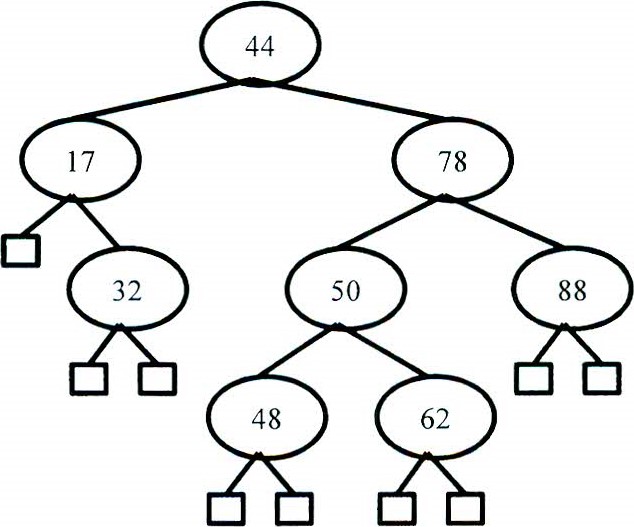
else if k = key(v)

return element(v)

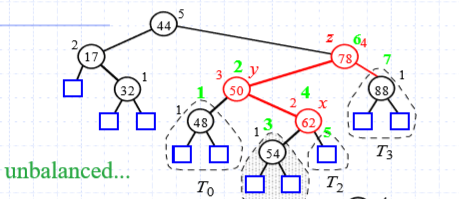
else { k > key(v) }

return findElement(k, T.rightChild(v))

(d) Given the following binary search tree, show how it would be modified by inserting 54.



* + 1. marks)



(e) What is an AVL-tree? Include in your answer the idea of a rotation.

Show how the tree that results from inserting 54 in part (d) would be rebalanced if it were an AVL-tree.

(11 marks)

An AVL is a binary search tree with a balancing property that the height can only be 1, 0 or -1 depending on the children descent from the lash hand side and the right hand side. If the tree balance value is not these values, it is deemed unbalanced and needs to be rotated depending on what way it is unbalanced. There are two ways of rotation. There is a single rotation and a double rotation. 