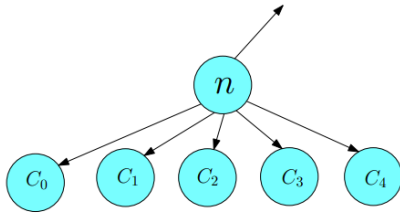


## Past paper questions

### (c) Trees and Recursion (8 marks)

(i) We've been dealing with binary trees in this module, but in this exam, let's deal with 5-ary trees. In a 5-ary tree, each node can have up to five children and a parent (the only node in the tree that has no parent is the root of the tree).



The node  $n$  contains a reference to an object and references to five children. Each child has an index  $C_0$  through  $C_4$  from left to right. The node  $n$  does not contain a reference to the parent. The node has only one constructor that takes in a reference to the object stored in this node and an array of five children (you do not need to check if there are five). If a child does not exist, the reference is null. It has a method that returns the object stored at this node `getElement()`. It also has a method `getChild(int i)` that takes in an index  $i$  (0 through 4) and returns a reference to `Node5<T>` which is the appropriate child (which can be null).

Write a java generic `Node5<T>` that implements the above-described node. [5 marks]

(ii) In the module, we studied a preorder traversal of a tree. Pretend you have the following class that fully implements a 5-ary tree that uses your `Node5<T>` class.

```
public class Tree5<T> {
    private Node5<T> root;
    ...
    //you should use methods defined in part (i)
    //assume they work as described above.
```

```
...
    public void preOrder() {
        ... //to implement in (ii) ...
    }
}
```

Pretend that there is a 5-ary tree constructed and stored in the root beforehand. Given this tree, implement the method `public void preOrder()` that conducts a preorder traversal of the 5-ary tree inside the `Tree5<T>` class. To visit the node in the tree, simply print out its contents to the screen. Hint: you may need a private helper method. [3 marks]

### (b) Linked Lists

Recall that you have a `Link` class which forms the basis of a linked list.

```
public class Link {
    private Object element;
    private Link next;

    public Link (Object e, QueueElement n) {
        this.element = e;
        this.next = n;
    }
    ...
}
```

Assume that `Link` has all get and set methods already implemented.

i) Write a `LinkedList` class. This class should have all necessary attributes for a linked list and a constructor that takes no parameters and constructs an empty list. Do not implement any other methods. You will lose marks. [3 marks]

ii) Inside your linked list class, assume that a linked list has already been defined and contained in your attributes. Write a method `public void removeMiddle()` that removes the middle element of your linked list. If there is an odd number of elements, the middle is clearly defined. If there is an even number of elements, take the lower of the two. [5 marks]

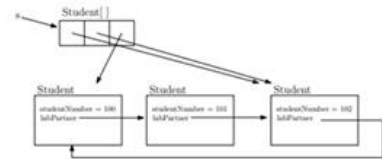
### Reference Diagrams

Consider the following class:

```
1 public class Student {
2     private int studentNumber;
3     private Student labPartner;
4     private static int nextStudentNumber = 100;
5
6     public Student () {
7         this.studentNumber = Student.nextStudentNumber;
8         Student.nextStudentNumber++;
9         this.labPartner = null;
10    }
11
12    public void setLabPartner (Student labPartner) {
13        this.labPartner = labPartner;
14    }
15 }
```

(a) Write code that is able to construct an array of 10 student objects and loads them in increasing student number order into that array. The lab partners of these students should all be set to null. [3 marks]

(b) Consider the reference diagram below:



Write code that constructs the above reference diagram. [5 marks]

### (c) Generics, Linked List, and Stacks/Recursion

(i) A stack could be implemented with a linked list. We also learnt generics in this module. Provide the *class definition and the data only* for a `Link<T>` class using a generic. No other methods are required. [2 marks]

(ii) Recall that a stack could have the following behaviours:

```
public class Stack<T> {
    public boolean isEmpty () {...};
    public void push (T newItem) {...};
    public void pop () {...};
    public T peek () {...};
}
```

You don't need to implement the stack. You can assume that it exists and is functional. Also, pretend that you have a functional implementation of a linked list and a link with the following methods:

```
public class Link<T> {
    //your (i) data here...

    //returns the next link in the list
    public Link<T> next () {...}
    //returns the data of this link
    public T getData () {...}
}

public class LinkedList<T> {
    private Link<T> head; //list front
    private Link<T> tail; //list back

    ...
}
```

Write either a **recursive method** or an **iterative method** using a **stack** that prints out the linked list in reverse order. The name of your method should be `public void rprinter()` and should be defined in the `LinkedList<T>` class. Assume `toString()` has been implemented for the data and returns a string representation of it. [6 marks]

(d) Bag of Words (9 marks)

In a bag of words model of a document, each word is considered independently and all grammatical structure is ignored. To model a document in this way, we create a list of all possible words in a document collection. Then, for each document you can count the number of instances of a particular word. If the word does not exist in the document, the count is zero. For this question, we will be making a `BagOfWordsDocument` class.

my document: *Aardvarks play with zebra. Zebra?*

	<i>aardvarks</i>	1
	...	...
	<i>play</i>	1
	...	...
mydocument =	<i>Tokyo</i>	0
	...	...
	<i>with</i>	1
	...	...
	<i>zebra</i>	2

For the purposes of this exam, we assume that verbs and nouns with different endings are different words (work and working are different, car and cars are different etc). New line characters only occur when a new paragraph in the document has been started.

We want to ensure that *zebra* and *Zebra* are the same word. The Java API's `String` class has a method `public String toLowerCase ()` that converts all the characters of a string to lower case. For example:

```
String myString = "ZeBrA";
String lowerZebra = myString.toLowerCase();
System.out.println (lowerZebra); //prints zebra
```

Suppose we have a specialised `HashMap` data structure for this purpose. The class has the following methods which are implemented and you can use.

- `HashMap ()` - constructor to construct an empty `HashMap`
- `boolean isEmpty()` - true if the `HashMap` is empty, false otherwise
- `public void put(String key, int value)` - sets the integer value with the `String` key
- `public int get(String key)` - returns the count `int` stored with this string. If the key is not in this `HashMap` it returns 0.

in this `HashMap` it returns 0.

- `String[] items ()` - returns the list of all strings stored in this `HashMap`

(i) Write a class `BagOfWordsDocument` that models a bag of words. The class should only have one attribute: a private data structure that models the bag of words. You should make a default constructor that creates an empty bag of words. [2 marks]

(ii) Write a java method `public void initialise (String filename)` in `BagOfWordsDocument` that opens the file, reads it, and initialises the the data structure in (i). You are responsible for converting all characters to lower case using the information specified above. [4 marks]

(iii) Write a method `public ArrayList<String> commonWords (BagOfWordsDocument b)` that returns an array list of all words common to this document and the passed document. [3 marks]

(d) Graph Flood Fill (9 marks)

Consider the following linked representation of a graph:

```
public class Node {
    private int id;
    private int numEdges; //number of direct neighbours
    private Node[] edges;

    public int getId () {
        return this.id;
    }
}

public class Graph {
    private int numNodes; //total number of nodes in Graph
    private Node[] nodes;
    ...
}
```

For all parts of this question, assume the graph is constructed and stored in this linked data structure. In this question, we simulate the spread of information through a graph from a seed set. Picture it as a flood where a node gets filled with liquid in the next step if one of its neighbours is filled with liquid in the current step.

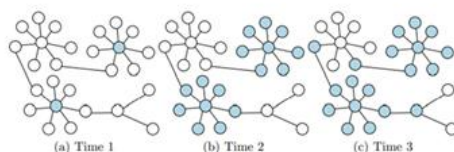


Figure 8: How flood fill works on a graph.

There is an array `isFilled` that contains the current filled state of each node. If the filled state is `true`, the node is filled with liquid (shaded in the above image). If the filled state is `false`, the node is not filled with liquid.

(i) Write a public method `boolean filled (boolean[] isFilled)` in the `Node` class that returns true if the node will be filled in the next step and false otherwise. The node id is the same as the index used in the `isFilled` array. [3 marks]

(ii) Write a public method `boolean[] nextState (boolean[] isFilled)` in the `Graph` class that creates and returns the filled state of every node in the graph in the next step given the current state array `isFilled`. The node id is the same as the index used in the `isFilled` array. [2 marks]

(iii) Write a public method `ArrayList<boolean[]> simulate (int t, boolean[] seeds)` in the `Graph` class that returns the simulation history after `t` timesteps of flooding starting from `seeds`. The node id is the same as the index used in the `seed` array. `ArrayList<boolean[]>` has a method `add (boolean[] e)` that adds `e` to the end of the array list. [2 marks]

(iv) If you completed to (ii) or (iii), your solution likely needs to traverse every node in the graph at every timestep which is inefficient. Using a data structure taught in this module, you can only visit the nodes flooded in the previous timestep. Using two sentences only name the data structure and state how it can be used to accelerate the algorithm. Be concise. Any incorrect information will lose marks (even if the correct information is present) [2 marks].

(c) **Euler Diagrams** Suppose we have a `HashSet<E>` data structure. This data structure functions like a hash map, but stores the existence or absences of keys without a value. The class has the following methods.

- `HashSet ()` - constructor to construct an empty set
- `boolean isEmpty()` - true if the set is empty, false otherwise
- `public boolean add(E e)` - adds an element `e` to the set
- `boolean contains(E o)` - true, if the element is there and false otherwise
- `E[] items ()` - returns an array of items in the set

We have the notion of an Euler diagram (seen below).

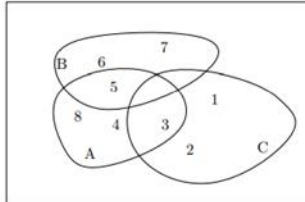


Figure 1

Here are some properties of this diagram:

- A contains: 3, 4, 5, 8
- B contains: 5, 6, 7
- The intersection of A and B is: 5
- The intersection of B and C is: empty

In our version of an Euler diagram, sets have string labels and elements are instances of the `Integer` class. To compare two instances of `Integer` use `boolean equals(Integer obj)`. To create an instance, you can simply call `new Integer (int)`.

i) Write a `Set` class. This class should have attributes to store the name of the set as a string and all unique integers in it. It should have a constructor

to create an empty set with a label, a public method to add elements, and a public method to check if an element is in the set. [4 marks]

ii) Write a method `public Set intersect (Set s)` in the `Set` class. This non-static method should intersect the passed set `s` with the set it is called upon. It should return a new set called "inter" with all elements in common to both sets. [3 marks]

iii) Write a method `public Set intersections (Set[] s)` in the `Set` class. This non-static method should intersect all sets in the passed array with this set and each other. It should return a new set called "inter" with all elements in common to all sets. [2 marks]

### Question 3: Trees and Queues (8 marks)

For the following question, assume you have a `TreeNode` class that implements a node of an arbitrary binary tree that stores an `int`. The tree node class has the following methods:

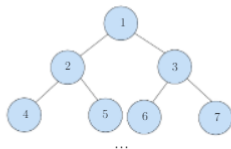
```
public class TreeNode {
    private int value;
    private TreeNode left;
    private TreeNode right;

    //Constructs a TreeNode with a specified left and right
    public TreeNode (int value, TreeNode left, TreeNode right);

    public int getValue();
    public TreeNode leftChild();
    public TreeNode rightChild();
}
```

(a) Write the following parts of the class `Tree` that includes all necessary attributes to store an arbitrary binary tree. It has a default constructor `public Tree ()` that constructs an empty tree. It also has a constructor `public Tree (int rt, int l, int r)` that constructs a tree with `rt` as the root, `l` as the left child, and `r` as the right child. In this case, `l` and `r` have no children. No other methods are required at this point. [3 marks]

(b) The diagram below depicts the *level order* traversal of a tree.



You can assume that an arbitrary binary tree is constructed and stored in `Tree`.

Write an implementation of a method `public void levelOrder ()` in `Tree` that prints to the screen the level order traversal of a binary tree stored in `Tree`. You can use the following data structure that is already implemented for you:

```
public Queue<T> {
    public Queue ();
    public T peek ();
    public boolean isEmpty();
    public void enqueue (T element);
    public void dequeue ();
}
```

[5 marks]

#### Question 4: Infection (9 marks)

Consider the following linked representation of a graph:

```
public class Node {
    private int id;
    private int numEdges; //number of direct neighbours to this node
    private Node[] edges;

    public int getId ();
}

public class Graph {
    private int numNodes; //total number of nodes in Graph
    private Node[] nodes;

    ...
}
```

For all parts of this question, assume the graph is constructed and stored in this linked data structure. In this question, we simulate the spread of information on a graph. Think of it as a cold through a social network as people become exposed to it. Also, to some extent, internet memes spread in a similar way.

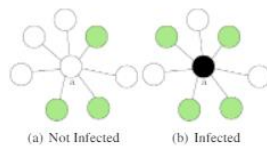


Figure 1: How infection works.

The state of a node is either `true` or `false`. If the value is `true` the node is infected. Otherwise, it is `false`. A value `thresh` is a number between 0 and 1 that is the percentage (0.51 is 51%) of direct neighbours of a node that need to be infected for this node to become infected at the next timestep. For example, if we have a `thresh` of 0.51, the value of `a` in figure 1(a) is `false` in the next timestep because  $3.0/7.0 < 0.51$ . However, in figure 1(b) `a` is `true` in the next timestep because  $4.0/7.0 > 0.51$ . You can assume the value of the `id` stored in each node corresponds to its position in the node array of the `Graph` class.

(a) Write a public method `boolean infected (float thresh, boolean[] state)` in the `Node` class that returns the infection state of this node. The node `id` is the same as the index used in the `state` array. **[4 marks]**

(b) Write a public method `boolean[] nextState (float thresh, boolean[] curState)` in the `Graph` class that creates and returns the infection state of every node in the graph given the passed `curState`. The node `id` is the same as the index used in the `curState` array. **[2 marks]**

(c) Write a public method `ArrayList<boolean[]> simulate (float thresh, int t, boolean[] start)` in the `Graph` class that returns the simulation history after `t` timesteps starting from `start`. The node `id` is the same as the index used in the `start` array. `ArrayList<boolean[]>` has a method `add (boolean[] e)` that adds `e` to the end of the array list. **[3 marks]**