

Generic Data Structure Viewer

Definition of Array Representation of Data Structures

The following document consists of the rules for representing the supported data structures into an array of Objects. The Objects in the array are the actual data held within the data structure. The data is displayed via a call to the Object's *toString()*.

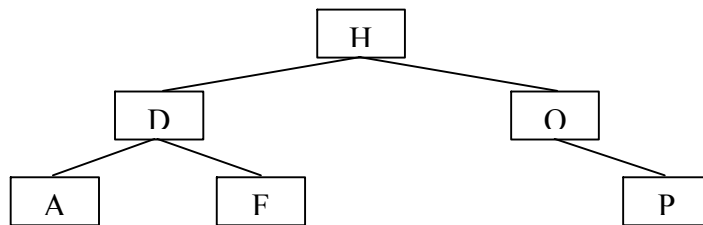
Binary Tree

The GDSV array representation of the binary tree follows the normal convention.

The following is a description of the algorithm:

- The 0th position is ignored (left *null*)
- Given the position of a node, i
 - Left child: $i * 2$
 - Right child: $i * 2 + 1$

Here is an example of the algorithm given the binary tree below:



All we do is go through the algorithm recursively. First we start with an empty array length 2^l (where l is the number of levels in the tree).

First we start with an array of size $2^3 = 8$:

index	0	1	2	3	4	5	6	7

In the case of the GDSV, this will be an array of Objects. Thus, for this example we will be filling the array with Character's. The 0th position in the array is ignored.

Place "H" into the array at position 1 (root starts at position 1)

index	0	1	2	3	4	5	6	7
		H						

It's left child ("D") will be $i * 2$: $1 * 2 = 2$

It's right child ("O") will be $i * 2 + 1$: $1 * 2 + 1 = 3$

index	0	1	2	3	4	5	6	7

		H	D	O				
--	--	---	---	---	--	--	--	--

Continue this process recursively until all nodes have been placed into the array.

index	0	1	2	3	4	5	6	7
		H	D	O	A	F		P

You may notice that the array is a level-order representation of the binary tree.

CODE: The following is sample code that will create the array representation of a binary tree:

```
private Object[] arrayRepresentationOfDataStructure() {
    Object[] arrayRep = new Object[(int)Math.pow(2,height())];
    arrayRepHelper(arrayRep, myRoot, 1);
    return arrayRep;
}

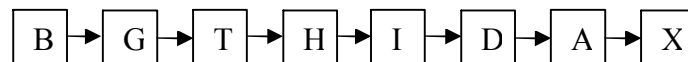
private void arrayRepHelper(Object[] array, TreeNode root, int position) {
    if(root==null) return;
    array[position] = root.getValue();
    arrayRepHelper(array, root.getLeft(), position * 2);
    arrayRepHelper(array, root.getRight(), position * 2 + 1);
}
```

NOTE: The above code assumes that...

1. The binary tree is constructed using the AP `TreeNode` class
2. `myRoot` is a reference to the root `TreeNode`
3. `height()` has been coded to return the height of the binary tree (returns the number of levels)

LinkedList Tree

A linked-list is represented as a normal array; the linked-list is merely copied into an array. Example:



The above linked-list would become the array shown below:

index	0	1	2	3	4	5	6	7
	B	G	T	H	I	D	A	X

CODE: The following is sample code that will create the array representation of a linked-list:

```

import java.util.ArrayList;

private Object[] arrayRepresentationOfDataStructure() {
    ArrayList tempList = new ArrayList();
    for(ListNode currentNode = front;
        currentNode!=null;
        currentNode = currentNode.getNext()) {
        tempList.add(currentNode.getValue());
    }
    return tempList.toArray();
}

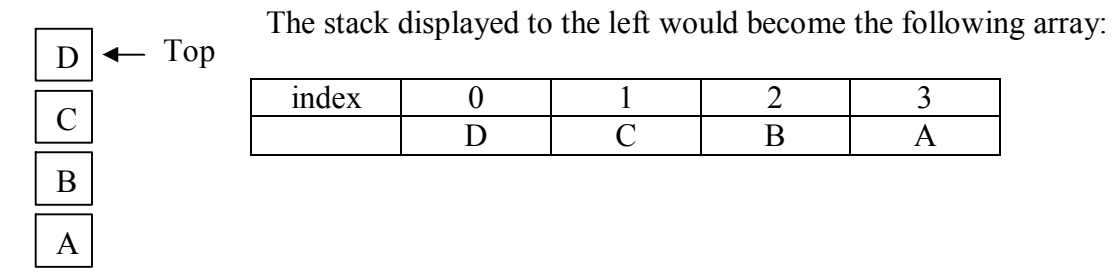
```

NOTE: The above code assumes that...

1. The linked-list is constructed using the AP ListNode class
2. *front* is a reference to the first node in the linked-list

Stack

A stack is represented as a normal array with the 0th element being the **top** of the stack. For example:

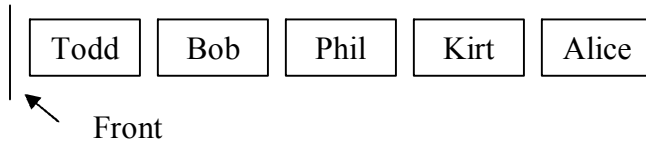


CODE: The code for the array representation of the stack is very dependent on the implementation of the stack. For example, if the stack was implemented using an `ArrayList`, the `arrayRepresentationOfDataStructure()` would simply be `myArrayList.toArray()`. However, be careful that the 0th element is truly the top of the stack.

Queue

A queue is represented quite logically as an array with the 0th element being the *front* of the queue.

For example:



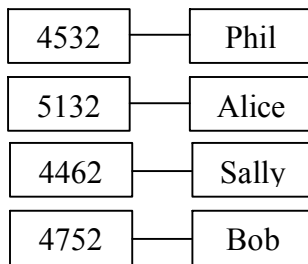
The queue above would become the following array:

index	0	1	2	3	4
	Todd	Bob	Phil	Kirt	Alice

CODE: The code for the array representation is too dependent on the implementation of the queue that no sample code can be given. For example, if a queue was implemented using an ArrayList, the *arrayRepresentationOfDataStructure()* (with the front being the 0th element of the ArrayList) would be `myArrayList.toArray()`.

Map

A map is represented by pair values. The array must be of even length (divisible by 2). The following example illustrates the representation:



The map to the left consists of ID number mapped to student names. This map would be translated into an array in pair values:

index	0	1	2	3	4	5	6	7
	4532	Phil	5132	Alice	4462	Sally	4752	Bob

CODE: The following is high level code to get the array representation of a map:

```
import java.util.ArrayList;  
  
public Object[] arrayRepresentationOfDataStructure() {
```

```

Iterator keySetIter = keySet().iterator();
ArrayList arrayRep = new ArrayList();
while(keySetIter.hasNext()) {
    Object key = keySetIter.next();
    arrayRep.add(key);
    arrayRep.add(get(key));
}
return arrayRep.toArray();
}

```

NOTE: The code above assumes the following:

1. Iterator and Set follow AP guidelines
2. The method is placed in map class that implements the AP Map interface
3. *keySet()* and *get()* work correctly

Bag

The following is a short description of the “bag” data structure, since it is not that well known, or that useful (but just fun to mess around with):

The Bag is a generic data structure similar to stacks and queues in that it only has two basic functions: *add(Object o)* and *remove()*. Also, both these operations are performed at constant time. A bag is only supposed to be a collection of objects in which order is unimportant. The *remove* function thus removes objects randomly from the collection. This is fulfilled as follows using an array of Objects:

- The *add* function simply adds an Object to the end of the array based on a temporary size variable. The size variable is then incremented.
- The *remove* function selects a random occupied location in the array, saves the value to be returned later, sets the value of that location in the array to be the object at the end of the array, decrements the size variable, and returns.

The array representation of the bag is just an array. An array of any amount of object will be taken care of. The bag viewer is a fun and interactive viewer to view either bags, queues, stacks, or linked-lists.