

Pre-requirement

Structure, and objects, Pointers, Understanding of Array -Random Access, time-memory trade-off. *Exposure to different kinds of operations and tasks.

1) What:

- **purpose: driven by the kind of applications**
- **'this' method,**
 - 'this' ==> Node(elem, next) constructor
- **(data) type of element that can be stored in a node, what kind of nodes (parent/next) can a node link to making it different from array,**
 - Node intNode = new Node<int>(5,null)
 - // element E will be of type "int"
 - //if in addition, next node is set to be of unspecified type "Node" only, then you could have the following:
 - Node floatNode = new Node<float>(2.5,intNode)
- **can node element be changed,**
 - Yes: by using setElement any number of times
- **empty node possible**
 - YES: 4 possibilities:either of the two components (element and next pointer) could be "null"
- **accessing a node from a collection,**
 - YES: Possible if you preserve references to each node in the collection either directly or through the "next" value
 - Node<Integer> myNode = new Node<Integer>(1,null);
myNode = new Node<Integer>(2,myNode) ;
myNode = new Node<Integer>(3,myNode) ;
myNode = new Node<Integer>(4,myNode) ;
System.out.println(myNode.getElement()+" "+
myNode.getNext().getElement()+"
"+myNode.getNext().getNext().getElement());
- **defining a node,**
 - Already addressed
- **way of referencing a node without referencing a collection,**
 - Already addressed
- **next of a node stores element of next node or its address,**
 - Ans: Contrast the following two
 - System.out.println(myNode.getNext());
 - System.out.println(myNode.getNext().getElement());
 - The first prints some address whereas the

second one prints some value (integer)

- **size limitation,**
 - No: Only size limitation is RAM
- **can two nodes hold address of a common node,**
 - YES
- **address (next) is relative or direct,**
 - Depends
 - `myNodeArray[3].getElement(); //relative address`
 - `myNode.getNext().getNext().getNext().getElement(); //direct address`
- **is node size fixed,**
 - NO: Depends on “type” of E. E could be Integer or could also be int[]
- **can node point to two different elements at the same time,**
 - YES: You can if you expand the Node class
- **what happens if a node gets ‘corrupt’?**
 - Object variable corrupt means the reference (pointer) gets corrupt
 - Either the pointer to the object can get corrupt OR the contents of the object being pointed to get corrupt
 - You lose info about all “following nodes”

2) How:

- Exposure to different kinds of operations and tasks such as fast (constant time) insertion and deletion which arrays cannot provide.

Exploratory Questions/Topics.

1) DS: Different ways of connecting nodes:

- flow charts.
 - Element == the operation at a particular point in flow chart
 - Next ==> Control flow arrows
 - So all arbitrary directed graphs?
- trees.
 - 0 or 1 directed path from one node to another
 - No loops
 - Need to expand the node and have two or more next pointers
 - Number of edges = number of vertices - 1
- binary tree
 - Tree with two "next pointers"
- search tree
 - all values on the left hand subtree are less than the value at that node
 - all values on the right hand subtree are greater than the value at that node
- tree with pointer from each node to its parent
- "circular trees".
 - You meant "complete directed graph"?
- Undirected nodes?
 - By design "Node" class supports directed (next) edges
- Make a node a "Hub" node that points to every other node and every other node points to the "hub" node
 - Like railways junctions
 - That is a simple tree with one "parent" and all others as children

- Groups,
 - Let us simplify it to "sets"
 - Uniqueness of elements, ease of adding/deleting elements ..
- ordered, based on element
 - probably a (binary) search tree?
- non-contiguous or contiguous memory blocks,
 - linked list serves this purpose
- previous/next node,
 - doubly linked list
- "contiguous" linked list,
 - generally called an array
- use collection as array,
 - linked list?
- chain of nodes vs. array?,
 - tradeoff involved: element access, efficient memory usage, dynamic size increase, memory allocation, addition/deletion of elements,
- cyclic list of nodes possible,
 - last element in linked list points to the first element
- meshes,
 - can be interpreted as a two dimensional array/grid
 - two dimensional linked list
- circular "queue" (by connecting last node to the first),
 - same as cyclic list of nodes
- social network graph
 - Social network graph could have different types of edges (friends undirected vs. Follows directed)
 - You might even consider having a "Node" to represent every edge!

2) READ:

- Access time to other nodes (slowed down),
- no predefined length/size,
- 2 units to store a single node should it not make it inefficient?,
- can backtrack if "prev" link is defined,

3) WRITE:

- low time complexity for insertion,
- deletion (especially with an 'expanded' node),
- unwanted elements removal,
- modifications are easy (?),
- Merging,
- splitting,

- other operations,
- direct link to an element,
- sorting is easier than in array -- change links instead of values

4) EXPANDING NODE: Modifying the node DS:

- multiple quantities stored,
- multiple (100s of) pointers stored,
- node type can change at any node in the structure,
- along with element and next also store relation with another node in order to allow creation of binary and B+ trees,
- add more methods,
- graphs,
- inheritance from a node possible to give "multi-nodes",
- "previous" address,
- methods to modify/read extra pointers,

Expected Exploratory Topics.

Link List, D-Linklist, Stack Queue, C-LL, Tree, Graphs