LinkedList

- LinkedLists are compose of nodes and references / pointers pointing from one node to the other
- The last reference is pointing to a NULL

A single node:

- Contains data -> interger, double or custom object
- Contains a reference poiting to the next node in the linkedlist

```
class Node{
    data
    Node nextNode
    ...
}
```

- Each node is composed of a data ans a reference / link to the next node in the sequence
- Simple and very common data structure
- The can be used to implement several other commin data types: stacks, queues
- Simple linked lists by themselves do not allow random access to he data // so we can not use indexes ... `getItem(int index)`
- Many basic operations such as obtaining the last node of the list or finding a node that contains a given data or locating the place where a new that contains a given data or lacting the place where a

new node should be inserted - require sequential scanning of most or all of the list elements

Advantages

- LinkedLists are dynamic data structures (arrays are not)
- It can allocate the needed memory in tun-time
- Very efficient if we want to manipulate the first elements
- EASY IMPLEMENTATION
- Can store items with different sizes: an array assumes every element to be exactly the same
- It´s easier for a linkedlist to grow organically. An array’s size needs to be known ahead of time, or re-created when it needs to grow

Disadvantages

- Waste memory becaouse of th references
- Nodes in a linkedlist must be read in order from the beginning as linkedlist sequencial access (array items can be reached via indexes in O(1) time)
- Difficulties arise in linkedlist when it comes to reverse traversing. Singly linkedlists are extremely difficult to navigate backwars,
- Solution: doubly linkedlist -> easier to read, but memory is wasted in allocating space for a back pointer

Resume

	Linked List	**Arrays**

Search	O(N)	O(1)
Insert at the start	O(1)	O(N)
Insert at the end	O(N)	O(1)
Waste space	O(N)	0

Bruna Santos - January 29, 2018 11:58 am