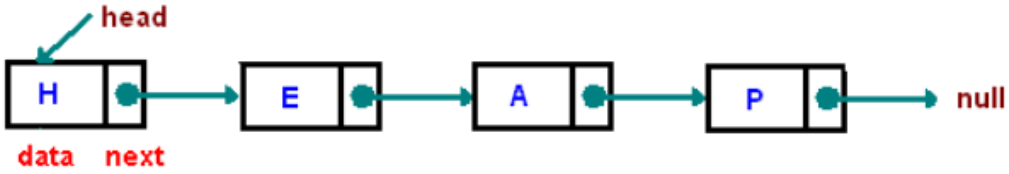
“Linked Lists”

- Problem with using arrays are:
  - they are static structures.
  - Cannot be easily extended or reduced to fit the data set.
  - Arrays are also expensive to maintain new insertions and deletions.

- A linked list is a linear data structure where each element is a separate object.



- Each element (we will call it a **node**) of a list is comprising of two items
  - the data & a reference to the next node. The last node has a reference to null.
- The entry point into a linked list is called the **head** of the list. It should be noted that **head** is not a separate node, but the reference to the first node.
- If the list is empty, then the **head** is a null reference.

- It’s a dynamic data structure.
- The number of nodes in a list is not fixed and can grow and shrink on demand.
- Application which must deal with an unknown number of objects will need to use a linked list.

- One disadvantage of a linked list against an array is that it does not allow direct access to the individual elements.
- If you want to access an item, then you have to start at the head and follow the references until you get to that item.
- Another disadvantage is that a linked list uses more memory compare with an array - we extra 4 bytes (on 32-bit CPU) to store a reference to the next node.

LinkedLists.pdf

Detail PDF document attached.

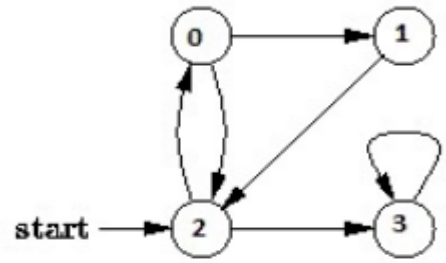
“Queue & Stack”

Stacks and Queues.pdf

Detail PDF document attached.

Reference: https://www.cs.cmu.edu/

Breadth First Traversal or BFS for a Graph (revisited)




```
// Java program to print BFS traversal from a given source vertex.
        // BFS(int s) traverses vertices reachable from s.
        import java.io.*;
        import java.util.*;

        // This class represents a directed graph using adjacency list representation
        class Graph
        {
                private int V; // No. of vertices
                private LinkedList<Integer> adj[]; //Adjacency Lists

                // Constructor
                Graph(int v)
                {
                        V = v;
                        adj = new LinkedList[v];
                        for (int i=0; i<v; ++i)
                                adj[i] = new LinkedList();
                }

                // Function to add an edge into the graph
                void addEdge(int v,int w)
                {
                        adj[v].add(w);
                }

                // prints BFS traversal from a given source s
                void BFS(int s)
                {
                        // Mark all the vertices as not visited(By default set as false)
                        boolean visited[] = new boolean[V];
```

```
        // Create a queue for BFS
        LinkedList<Integer> queue = new LinkedList<Integer>();

        // Mark the current node as visited and enqueue it
        visited[s]=true;
        queue.add(s);

        while (queue.size() != 0)
        {
            // Dequeue a vertex from queue and print it
            s = queue.poll();
            System.out.print(s+" ");

            // Get all adjacent vertices of the dequeued vertex s. If a adjacent has not
            // been visited, then mark it visited and enqueue it
            Iterator<Integer> i = adj[s].listIterator();
            while (i.hasNext())
            {
                int n = i.next();
                if (!visited[n])
                {
                    visited[n] = true;
                    queue.add(n);
                }
            }
        }
    }

    // Driver method to
    public static void main(String args[])
    {
        Graph g = new Graph(4);

        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);

        System.out.println("Following is Breadth First Traversal "+
                           "(starting from vertex 2)");

        g.BFS(2);
    }
}
```

DFS

Reference: http://www.geeksforgeeks.org/breadth-first-traversal-for-a-graph/

For DFS Example: http://www.geeksforgeeks.org/depth-first-traversal-for-a-graph/