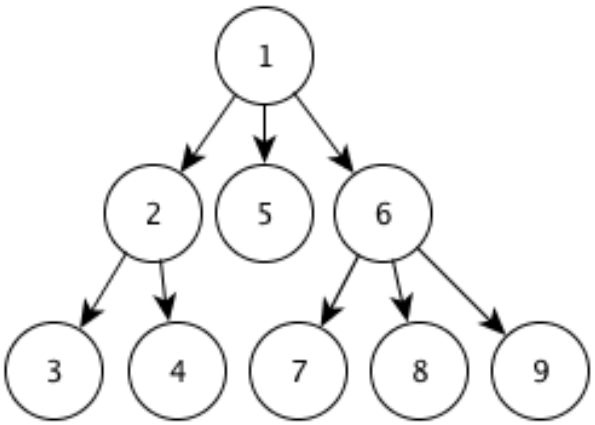
CECS 277 – Lecture 7 – Graph and Tree Traversal Algorithms

Depth First Search – Searches a tree or graph by successively exploring child nodes until a leaf node is found before visiting any siblings. This essentially explores each branch of the tree from top to bottom, moving toward the right. Depth First Search is often performed recursively, but it is not always necessary to do so.

Example: Order of explored nodes in a DFS.



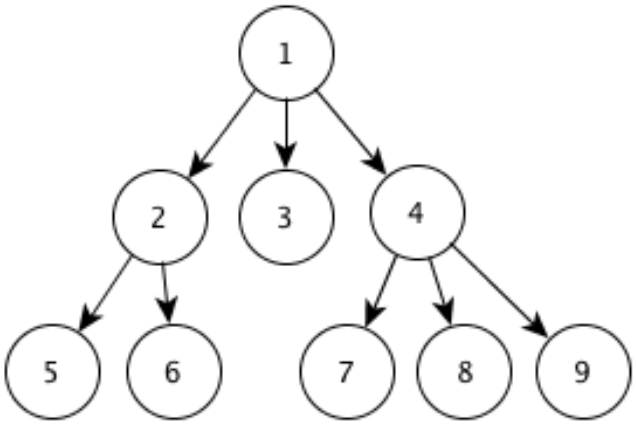
DFS Algorithm:

```
dfs(Node start, Node end){
      Stack s = new Stack();
      boolean found = false;
      s.push(start);
      do{
            Node n = s.pop();
            n.setVisited(true);
            if(n == end){
                  found = true;
            }else{
                  for (each child c of n){
                        if(!c.getVisited()){
                              s.push(c);
                        }
                  }
            }
      }while(!found && !s.isEmpty());
}
```

Once you have found the goal, the path can be found by propagating back through the nodes (as long as each node kept track of its parent). This can be accomplished by adding in a node object for each node, and then when children are added to the stack, a note is made of n as being their parent.

Breadth First Search – Searches a tree or graph by visiting all of the siblings of a child node before visiting its own child nodes. This essentially explores each level of the tree from left to right, top to bottom.

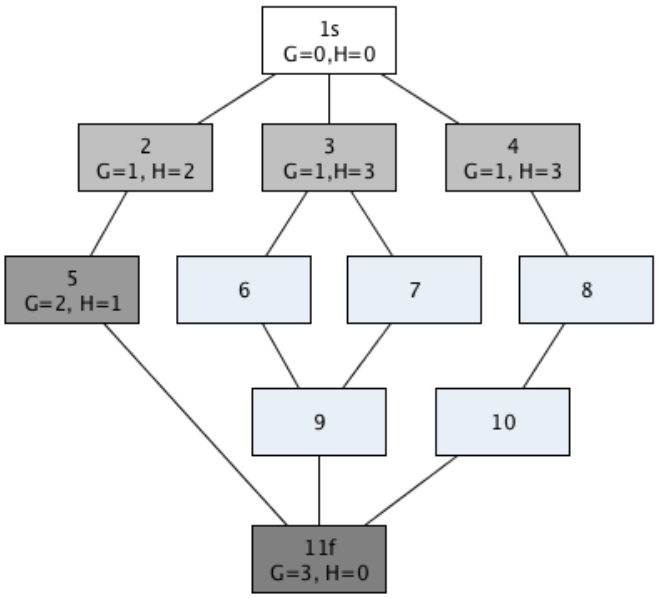
Example: Order of explored nodes in a BFS.



BFS Algorithm:

```
bfs(Node start, Node end){
      Queue q = new Queue();
      boolean found = false;
      q.push(start);
      do{
            Node n = q.pop();
            n.setVisited(true);
            if(n == end){
                  found = true;
            }else{
                  for (each child c of n){
                        if(!c.getVisited()){
                              q.push(c);
                        }
                  }
            }
      }while(!found && !q.isEmpty());
}
```

A* – Searches a tree or graph by visiting nodes that are determined to be the best based on a heuristic (calculated guess). The heuristic is based on the estimated distance (or cost) to move from that node to the goal node.



A* Algorithm:

```
aStar(Node start, Node end){
      List list = new List();
      List closed = new List();
      boolean found = false;
      list.add(start);
      start.setCost(0);
      do{
            Node n = list.lowestCost();
            closed.add(list.remove(n));
            if(n == end){
                  found = true;
            }else{
                  for (each neighbor g of n){
                        if(!closed.contains(g)){
                              if(!list.contains(g)){
                                    g.setParent(n);
                                    g.setCost(g.calcCost());
                                    list.add(g);
                              }else{
                                    if(g.getCost()> g.calcCost())
                                          g.setParent(n);
                                          g.setCost(g.calcCost());
                                    }
                              }
                        }
                  }
            }
      }while(!found && !list.isEmpty());
}
```