LAB # 05

ImplementING uninformed search techniquies

# OBJECTIVE

Finding shortest path using BFS and DFS search technique in python.

# THEORY

**Breadth First Search (BFS):**

Breadth First Search (BFS) searches breadth-wise in the problem space. Breadth-First search is like traversing a tree where each node is a state which may a be a potential candidate for solution. It expands nodes from the root of the tree and then generates one level of the tree at a time until a solution is found. It is very easily implemented by maintaining a queue of nodes. Initially the queue contains just the root. In each iteration, node at the head of the queue is removed and then expanded. The generated child nodes are then added to the tail of the queue.

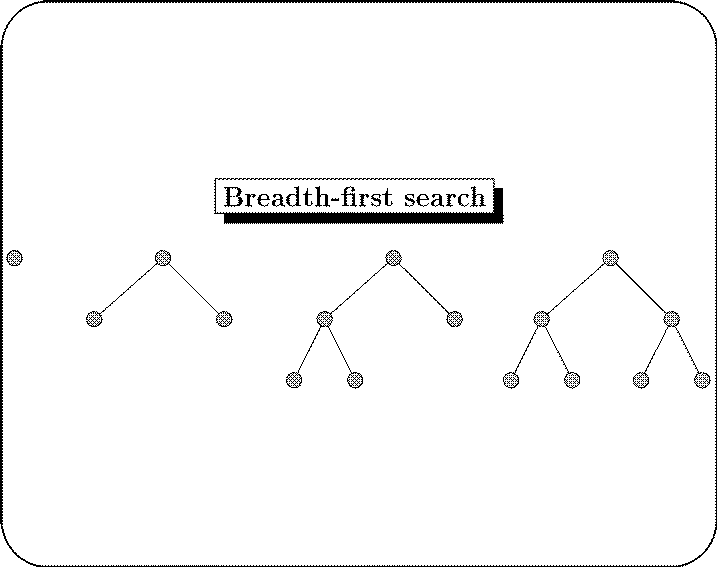
**ALGORITHM: Breadth First Search:**

1. Create a variable called NODE-LIST and set it to the initial state.
2. Loop until the goal state is found or NODE-LIST is empty.

* Remove the first element, say E, from the NODE-LIST. If NODE-LIST was empty then quit.
* For each way that each rule can match the state described in E do:

i) Apply the rule to generate a new state.  
 ii) If the new state is the goal state, quit and return this state.  
 iii) Otherwise add this state to the end of NODE-LIST

Since it never generates a node in the tree until all the nodes at shallower levels have been generated, breadth-first search always finds a shortest path to a goal.



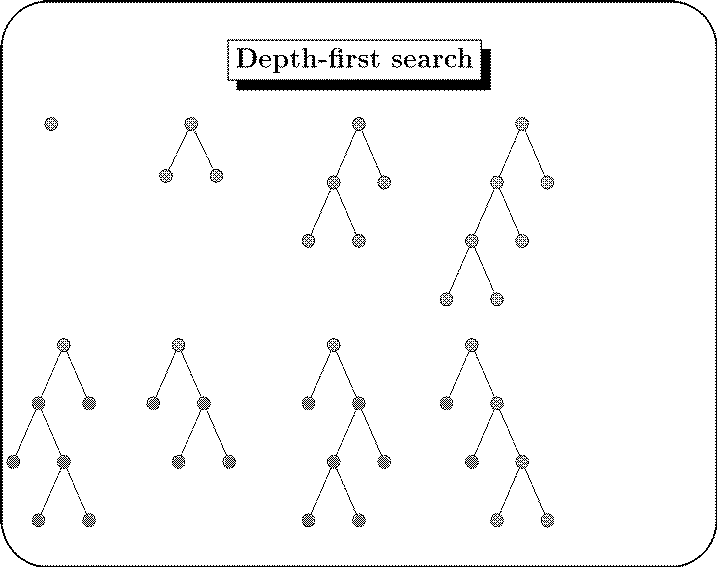
Look at the above tree with nodes starting from root node, R at the first level, A and B at the second level and C, D, E and F at the third level. If we want to [search](http://intelligence.worldofcomputing.net/ai-search/ai-search-techniques.html) for node E then BFS will search level by level. First it will check if E exists at the root. Then it will check nodes at the second level. Finally it will find E a the third level.

**Depth first search (DFS):**

Depth First Search (DFS) searches deeper into the problem space. Breadth-first search always generates successor of the deepest unexpanded node. It uses last-in first-out stack for keeping the unexpanded nodes. More commonly, depth-first search is implemented recursively, with the recursion stack taking the place of an explicit node stack.

**ALGORITHM: Depth first search:**

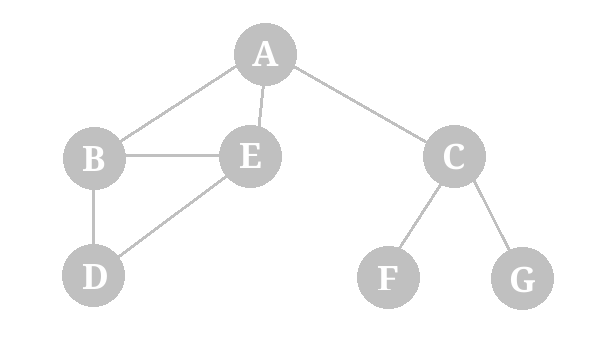
1. In Depth first search edges are explored out of the most recently discovered vertex. Only edges to unexplored vertices are explored.
2. When all of vertices edges have been explored, the search "back-tracks" to explore edges leaving the vertex from which vertex \Vas discovered.
3. The process continues until we have discovered all the vertices that are reachable from the original source vertex.
4. If any undiscovered vertices remain, then one of them is selected as a new source vertex.
5. This process if repeated until all vertices are discovered.



**Lab#3 Exercise:**

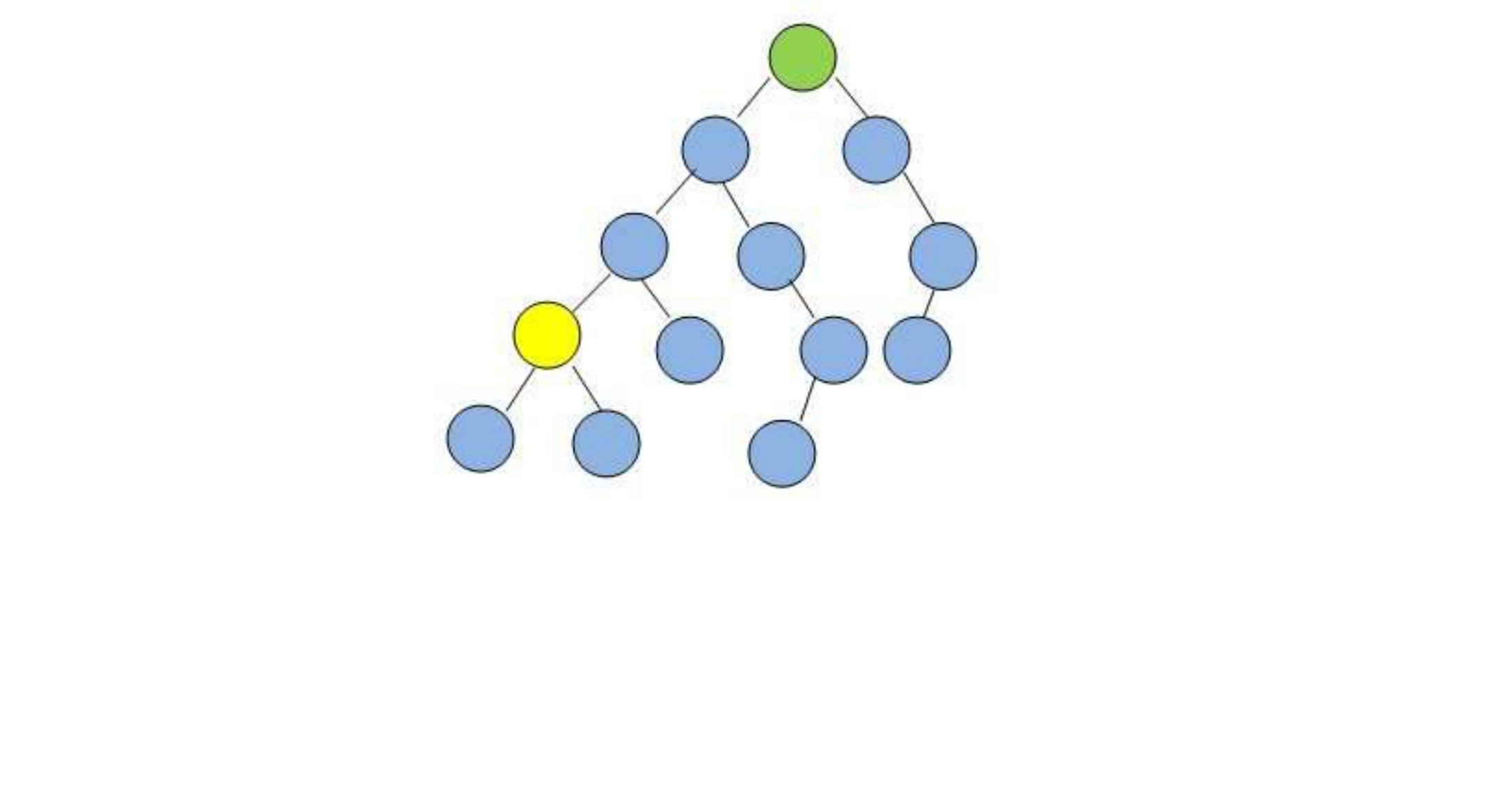
1. Consider the following graph. If there is ever a decision between multiple neighbor nodes in the BFS or DFS algorithms, assume we always choose the letter closest to the beginning of the alphabet first. A [connected graph](http://mathworld.wolfram.com/ConnectedGraph.html) with 7 nodes and 7 edges. The edges are [undirected](http://mathinsight.org/definition/undirected_graph) and [unweight](http://www.radford.edu/~nokie/classes/360/graphs-terms.html).  Distance between two nodes will be measured based on the number of edges separating two vertices.

Represent a graph with [adjacency list](https://en.wikipedia.org/wiki/Adjacency_list) using dictionaries. The keys of the dictionary represent nodes; the values have a list of neighbours.



Define function name ‘connected\_component’, this function keep track of all the visited nodes with BFS, is as simple as implementing the steps of the algorithm and assign ‘queue’ variable  already has a node to be checked, i.e., the starting vertex that is used as an entry point to explore the graph. The next step is to implement a loop that keeps cycling until queue is empty. At each iteration of the loop, a node is checked.  If this wasn’t visited already, its neighbours are added to queue. Once the loop is exited, the function (connected\_component) returns all of the visited nodes.

2. In the following graphs, assume that if there is ever a choice amongst multiple nodes, both the BFS and DFS algorithms will choose the left-most node first.



Starting from the green node at the top, which algorithm will visit the least number of nodes before visiting the yellow goal node?

1. Using question#1, now BFS implements traverses a graph. Finding the shortest path between two nodes.  The function we implement should be able to accept as argument a graph, a starting node (e.g., ‘G’) and a node goal (e.g., ‘D’). If the algorithm is able to connect the start and the goal nodes, it has to return the path.