**Breadth-First Search**

**Breadth First Search (BFS)** searches for breadth-wise in the problem space. Breadth-First search is like traversing a tree, where each node is a state which may be a potential candidate for the 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 can be very easily implemented by maintaining a queue of nodes. Initially the queue contains just a root. In each iteration, a 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***

**Step 1:** Create a new variable called NODE-LIST and set it to the initial state.

**Step 2:** Loop continuously until the goal state is found or NODE-LIST is empty.

1. Remove the first element, say E, from the NODE-LIST. If NODE\_LIST was empty then quit.
2. For every way that each rule can match the state described in E do:
3. Apply the rule to generate a new state.
4. If the new state is the goal state, then quit and return this state.
5. 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 the goal. Since each node can be generated in constant time and the amount of time used by Breadth first search is proportional to the number of nodes generated.

* **Advantages of Breadth-First Search**

1. BFS will never get trapped exploring the useless path forever.
2. If there is a solution, BFS will definitely find it out.
3. If there is more than one solution then BFS can find the minimal one that requires less number of steps.

* **Dis-advantages of Breadth-First Search**

1. The main drawback of Breadth first search is, its [memory](http://www.worldofcomputing.net/memory/computer-memory.html) requirement. Since each level of the tree must be saved in order to generate the next level, and the amount of [memory](http://www.worldofcomputing.net/memory/computer-memory.html) is proportional to the number of nodes stored, the space complexity of BFS is O(b^d). As a result, BFS is severely space-bound in practice so will exhaust the [memory](http://www.worldofcomputing.net/memory/computer-memory.html) available on typical computers in a matter of minutes.
2. If the solution is farther away from the root, breath first search will consume lot of time.

**Example:BREADTH-FIRST SEARCH**

**a**

**G**

**c**

**b**

**f**

**e**

**d**

**S**

**r**

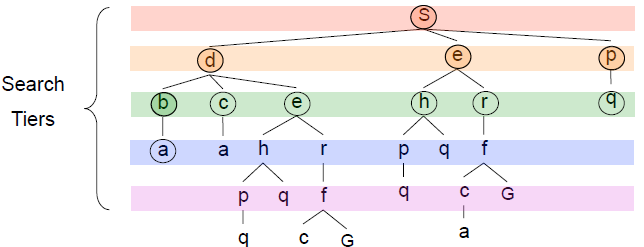
**h**

**q**

**p**

**Strategy:** Expandshallowest nodefirst.

**Implementation:**Fringe is a FIFOqueue.

****

**Expansion order:**(S,d,e,p,b,c,e,h,r,q,a,a,h,r,p,q,f,p,q,f,q,c,G)

[**DEPTH FIRST SEARCH**](http://intelligence.worldofcomputing.net/ai-search/depth-first-search.html)

Depth First Search (DFS) searches deeper into the problem space. [Breadth-first search](http://intelligence.worldofcomputing.net/ai-search/breadth-first-search.html) 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. If the initial state is a goal state, quit and return success.
2. Otherwise loop until success or failure occurred.
3. Generate a state, say E, and let it be the successor of the initial state. If there is no successor, signal failure
4. Call Depth-First Search with E as the initial state.
5. If success is returned, signal success. Otherwise continue in this loop.

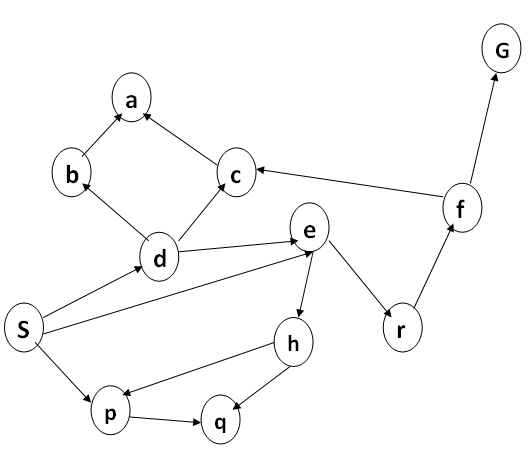
* **Advantages of Depth-First Search:**

1. The advantage of depth-first Search is that [memory](http://www.worldofcomputing.net/memory/computer-memory.html)requirement is only linear with respect to the search graph. This is in contrast with [breadth-first search](http://intelligence.worldofcomputing.net/ai-search/breadth-first-search.html) which requires more space. The reason is that the algorithm only needs to store a stack of nodes on the path from the root to the current node.
2. The time complexity of a depth-first Search to depth d is O(b^d) since it generates the same set of nodes as [breadth-first search](http://intelligence.worldofcomputing.net/ai-search/breadth-first-search.html), but simply in a different order. Thus practically depth-first search is time-limited rather than space-limited.
3. If depth-first search finds solution without exploring much in a path then the time and space it takes will be very less.

* **Disadvantages of Depth-First Search:**

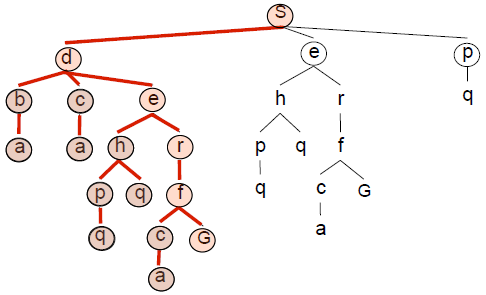
1. The disadvantage of Depth-First Search is that there is a possibility that it may go down the left-most path forever. Even a finite graph can generate an infinite tree. One solution to this problem is to impose a cutoff depth on the search. Although the ideal cutoff is the solution depth d and this value is rarely known in advance of actually solving the problem. If the chosen cutoff depth is less than d, the algorithm will fail to find a solution, whereas if the cutoff depth is greater than d, a large price is paid in execution time, and the first solution found may not be an optimal one.
2. Depth-First Search is not guaranteed to find the solution.
3. And there is no guarantee to find a minimal solution, if more than one solution exists.

**Example: Depth First Search**

****

**Strategy:** expanddeepest node first.

**Implementation:** Fringe is a LIFO queue (a stack).



**Expansion ordering:**(d,b,a,c,a,e,h,p,q,q,r,f,c,a,G)