❖ Uninformed Search: -

Blind Search

This type of search takes all nodes of tree in specific order until it reaches to goal. The order can be in breath and the strategy will be called breadth–first–search, or in depth and the strategy will be called depth, first search.

1- Breadth – First- Search

First search, when a state is examined, all of its siblings are examined before any of its children. The space is searched level-by-level, proceeding all the way a cross one level before doing down to the next level.

Breadth –first –search Algorithm

```
• Begin
• Open: = [start];
• Closed: = [ ];
• While open $\neq$ [ ] do
• Begin
• Remove left most state from open, call it x;
• If x is a goal the return (success)
• Else
• Begin
• Generate children of x;
• Put x on closed;
• Eliminate children of x on open or closed; (Removing the repeated
  child or node)
• Put remaining children on right end of open
• End
• End
```

- *Return (failure)*
- *End.*

- Breadth – First- Search

$$\begin{tikzcd}
 & & & & & A \arrow[dll] \arrow[d] \arrow[drr] & & & \\
 & & & B \arrow[dll] \arrow[d] & & C \arrow[dl] \arrow[d] & & D \arrow[dl] \arrow[d] & \\
 & E \arrow[dl] \arrow[dr] & & F \arrow[d] & G \arrow[d] & H \arrow[d] \arrow[dr] & I \arrow[d] \arrow[dr] & J \arrow[dr] & \\
K \arrow[d] & & L \arrow[d] & M & N & O & P \arrow[dr] & Q & R \\
S & & T & & & & & U &
\end{tikzcd}$$

1 –Open= [A]; closed = [ ].

2 –Open= [B, C, D]; closed = [A].

3 –Open= [C, D, E, F]; closed = [B, A].

4 –Open= [D, E, F, G, H]; closed = [C, B, A].

5 –Open= [E, F, G, H, I, J]; closed = [D, C, B, A].

6 –Open= [F, G, H, I, J, K, L]; closed = [E, D, C, B, A].

7 –Open= [G, H, I, J, K, L, M]; closed = [F, E, D, C, B, A].

8 –Open= [H, I, J, K, L, M, N,]; closed = [G, F, E, D, C, B, A].

9 –and so on until either U is found or open = [ ].

<u>Depth- First-Search</u>

- In depth -search, when a state is examined, all of its children and their descendants are examined before any of its siblings.

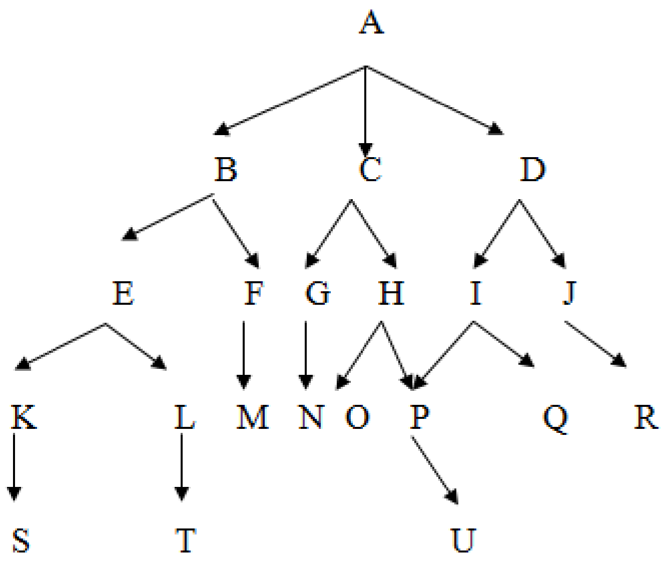
- Depth–first search goes deeper into the search space whenever this is possible only when no further descendants of a state can found

Depth – first – search Algorithm

Begin

- *Open: = [start];*
- *Closed: = [];*
- *While open ≠ [] do*

- *Remove leftmost state from open, call it x;*
- *If x is a goal then return (success)*
- *Else begin*
- *Generate children of x;*
- *Put x on closed;*
- *Eliminate children of x on open or closed; put remaining children on left end of open end*
- *End;*
- *Return (failure)*
- *End.*



1 –Open= [A]; closed = [ ].

2 –Open= [B, C, D]; closed = [A].

3 –Open= [E, F, C, D]; closed = [B, A].

4 –Open= [K, L, F, C, D]; closed = [E, B, A].

5 –Open= [S, L, F, C, D]; closed = [K, E, B, A].

6 –Open= [L, F, C, D]; closed = [S, K, E, B, A].

7 –Open= [T, F, C, D]; closed = [L, S, K, E, B, A].

8 –Open= [F, C, D,]; closed = [T, L, S, K, E, B, A].

9–Open= [M, C, D]; closed = [F, T, L, S, K, E, B, A]

10-Open= [C, D]; closed= [M, F, T, L, S, K, E, B, A]

11-Open= [G, H, D]; closed= [C, M, F, T, L, S, K, E, B, A]

12-Open= [N, H, D]; closed= [G, C, M, F, T, L, S, K, E, B, A]

13-Open= [H, D]; closed= [N, G, C, M, F, T, L, S, K, E, B, A]

14- Open= [O, P, D]; closed= [H, N, G, M, F, T, L, S, K, E, B, A]

15- Open= [P, D]; closed= [O, H, N, G, M, F, T, L, S, K, E, B, A]

16- Open= [U, D]; closed= [P, O, H, N, G, M, F, T, L, S, K, E, B, A]

and so on until either U is found or open = [ ].

❖ ***Deference Between Depth and Breath Search algorithm:***

Depth- First Search (DFS)

1- Use stack, nodes are added to the top of list.
2- Examines all the node's siblings and their descendent before the node's siblings.
3- It is not complete, it may be stack in an infinite branch.
4- It is not optimal, it will not find the shortest path.
5- Requires less memory, only memory for states of one path needed.
6- Is efficient if solution path is known to be long.
7- There is no guarantee of getting a solution.

Breath- First-Search (BFS)

1- Uses a queue, nodes are added to the end of list.
2- Examine all node's siblings and their children.
3- Is complete. It always finds a solution if it exists.
4- Is optimal, it always finds shortest path.
5- Requires more memory.
6- Is efficient if the branching is very high.
7- If there is solution, then BFS is guaranteed to find it.