Sinhgad Institutes | Date

Group A

Program 1 - Implement depth first search algorithm and Breadth first search algorithm, use a undirected graph & develop a recursive algorithm for searching all the vertices of a graph or tree data structure

1.A - Program on uniformed search methods (BFS).

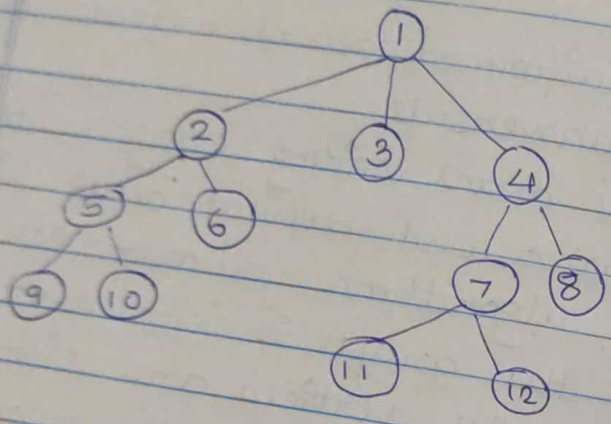
Theory - Graph Transversals

- Graph transversal means visiting every vertex & edge exactly once in a well-defined order. While using certain graph algorithms, you must ensure that each vertex of the graph is visited exactly once. The order in which the vertices are visited are important & may depend upon the algorithm or question that you are solving.
- During a transversal, it is important that you track which vertices have been visited. The most common way of tracking vertices is to mark them.

▷ Breadth first Search (BFS):-

- There are many ways to transverse graphs. BFS is the most commonly used approach. BFS is transversing algorithm where you should start transversing from a selected node (source or starting) & transverse the graph layer wise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next level neighbour nod.

• As g the name BFS suggests, you are requir
to transverse the graph breadthwise as follows:
1) First move horizontally & visit all the nodes of th
current layer.
2) Move to the next layer, in layer 2. Therefore, in
BFS, you must transverse all nodes in layer 1, bef
you move the next layer 2.



- Transversing child nodes:-
• A graph might include cycles, which can load you back to the same node when transverse it use a boolean array to indicate the node after it has been treated to prevent processing it again.
• In the above diagram. Start tranversing from 1 & visit its child node 2, 3, 8, 4. Store them in the order in which they are visited. This will allow you to visit the child nodes of 2 first (i.e. 86) then of 4 (ie. 7 8 8) etc
• To make this process easy, use a queue to store the node & make it as 'visited' until all its neighbours are marked. The queue follows the FIFO queue method, & therefore, the neighbours of the

Teacher's Sign.:

Sinhgad Institutes | Date

node will be ~~used~~ visited in the order in which they were inserted in the node. ie. the node that was inserted first will be visited first & so on on.

Algorithm:-

Step 1 - SET STATOS = 1 (ready state) for each in tree

Step 2 - Enqueue the starting node A & set its status = 2 (waiting state).

Step 3 - Repeat steps 4 & 5 until queue is Empty

Step 4 - Dequeue a node N. Process it & set its STATOS = 3 (Process state).

Step 5 - Enqueue all the neighbour of N that are in the ready state (whose STATUS = 1) & set there STATUS = 2 (waiting state) [END of loop]

Step 6: EXIT

Sample Program -

```
Import Java.util.Iterator;
Import Java.util.LinkedList;

Public class main {
    public static void main (String [] args)
    {  Graph g = new graph (8);

        g.AddEdge (1,2);      g.AddEdge (4,5);
        g.AddEdge (1,5);      g.AddEdge (4,6);
        g.AddEdge (2,3);      g.AddEdge (5,4);
        g.AddEdge (2,5);      g.BFS (1);
        g.AddEdge (3,4);
    }
}
```

```
class Graph {
    Private int NodeNumber;
    Private linkedlist <integer> AdjacentNode[];

    Graph (int v)
    { AdjacentNodes = new linkedlist [v];
      For (int i=0 ; i<AdjacentNodes.length; i++)
      { AdjacentNode [i] = newlinked list ();
        NodeNumber = v;
    }

    Public void AddEdge (int v, int w)
    { Adjacent Nodes [v].add (w);
    }
    Public void BFS (int s)
    { boolean visited [] = new boolean [NodeNumber];
      For (int i=0; i < NodeNumber ; i++)
      { visited [i] = false;
      }
      Linkedlist (integer> q queue = new linkedlist();
      visited [s] = true;
      queue .add (s);

      while (queue.size ()!=0)
      { s = queue.poll ();
        system.out.println ("visiting"+s+" ");
        while( i.hashNext (1))
        {
          int (n) = i.next (1);
          if (! visited [n])
```

```
visited [n] = true;
    queue.add(n);
}
}
}
}
}
```

Output -

Following it Breath First Transversal

visiting 1
visiting 2
visiting 3
visiting 4
visiting 5
visiting 6.

Conclusion -

Thus we have successfully implemented BFS with example

Sinhgad Institutes | Date

1.B - Program on Uniformed Search methods (DFS):-

Aim:- To implement uniformed search strategy DFS.

Theory :- DFS is an algorithm for transversing or searching tree or graph data structure. The algorithm starts at the root node & explores as far as possible along each branch before backtracking

- Depth - First Search (DFS) it is a method for exploring a tree or graph In a DFS, you go as deep as possible down one path before backing up & trying different one.
- DFS is like walking through a corn maze. You explore one path, hit a dead end & go back & try a diff. one.

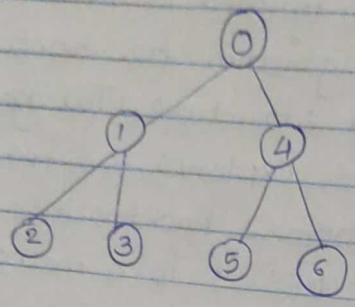
Depth First Search :-

- It is implemented in recursion with LIFO stack data structure. It creates the same set of nodes as the Breadth - First method, only in a different order.
- As the nodes on the single path are stored in each iteration from root to leaf node, the space requirement to store nodes is linear. With branching factor b and depth as m, the storage space is bm

Disadvantage :-

- This algorithm may not terminate & go on infinitely on one path. The solution of this issue is to choose a cut-off depth. If the ideal cut-off is d & if choosen cut-off is lesser than d, then this algorithm may fail. If the choosen cut-off is more than d, then execution time increases.

• Its complexity depend on the number of path it co
check duplicate nodes.



Algorithm :-

Steps

1) SET STATUS = 1 (ready state) for each node.
2) Push the starting node A on the stack & set its STATUS = 2 (waiting state).
3) Repeat Steps 4 & 5 until STACK is empty.
4) POP the top node N. process it & set its STATUS = 3 (Processed state).
5) Push on the stack all the ~~as~~ stack all the neighbours of N that are in the ready state (whose STATUS = 1) & set their STATUS = 2 (waiting state).
6) Exit [End of loop]

Sample Program :-

```
Import Java.util.Iterator;
Import Java.util.LinkedList;
Import Java.util.List;
Public class Main
{
Public static void main (String [] args)
```

Date

```
{ Graph g = new Graph (8);
    g.AddEdge (1,2);
    g.AddEdge (1,5);
    g.AddEdge (2,3);
    g.AddEdge (2,5);
    g.AddEdge (3,4);
    g.AddEdge (4,5)
    g.AddEdge (4,6);
    g.AddEdge (5,6);

  g.DFS();
}
}
Class Graph {
    private int NodeNumber;
    Private LinkedList<Integer> AdjacentNodes[];
    Graph (int v) {
        AdjacentNodes = new LinkedList[v];

    For (int i=0; i< AdjacentNode.length; i++)
    { AdjacentNode[i] = new LinkedList();
    }
      NodeNumber = v;
    }

    while (i.ha-Next())
    { int n = i.next();
      if (!visited[n])
      {
          DFSUtil (n visited);
      }
    }
}
```

Date

Output :-

visiting 0
visiting 1
visiting 2
visiting 3
visiting 4
visiting 5
visiting 6
visiting 7

Conclusion - Thus we have successfully implemented DFS example.