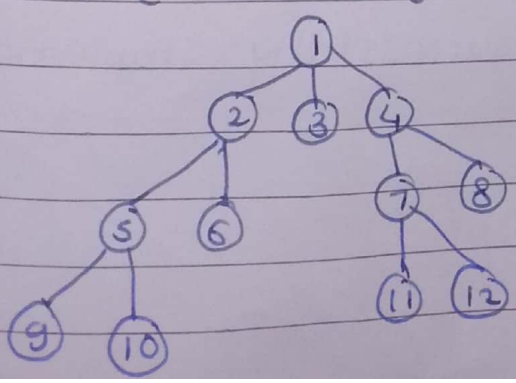
Assignment NO 1

A program on uninformed search methods.

- Aim :- To study uninformed search strategy : BFS.
- Theory :- Graph traversal.
  - Graph traversal means visiting every vertex & edge exactly once in a well-defined order. While using certain graph algorithms, you must ensure that each vertex of graph is visited exactly once. The order in which vertices are visited are important & may depend upon algorithms or question that you are solving.
  - During a traversal, 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 :-
  - There are many ways to traverse graphs. BFS is most commonly used approach BFS is a traversing algorithm where you should start traversing from a selected node & traverse the graph layer wise thus exploring the neighbor nodes.
  - As the names BFS suggests, you are required to traverse graph breadwise :

1) First move horizontally & visit all nodes of current layer.
2) move to the next layer. in layer 2



- Traversing child nodes :-
- ~~There are many ways~~
- A graph might include cycles, which can lead back to same node when traversing it. Use a boolean array to indicate node after it has been treated to prevent processing it again: while processing the nodes in graph's layer.
- In above diagram start traversing from 1 & visit its child nodes 2, 3 & 4. store them in order in which they are visited. This will allow you to visit the child nodes of 2 first; then of 4 & etc.
- To make this process easy, use queue to store the node & mark it as visited until all its neighbors are marked. queue follows FIFO.

◉ Algorithm :-

- step 1: set status = 1 (ready state) for each node in
- step 2: Enqueue the starting node A & set its status = 2 (waiting)
- step 3 :- repeat step 4 & 5 until queue is empty
- step 4 :- Dequeue a node N. process it & set its status = 3 (process state)
- step 5: Enqueue all neighbors of N that are in ready state & set status = 2
- step 6: Exit.

◉ conclusion :-

Thus we have successfully implemented BFS.

program on uninformed search methods

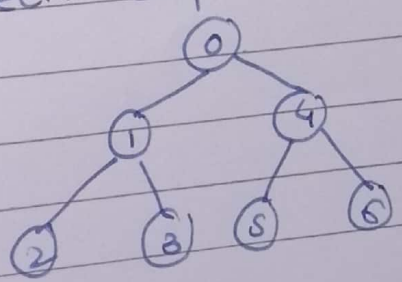
= Aim:- To implement uninformed search strategy
DFS

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

- DFS search is like walking through a corn maze.

= Depth first search:-

- It is implemented in reccursion with LIFO. it creates same set of nodes as BFS, only in a different manner.
- As the node on single path are stored in each iteration from root to leaf; the space requirement to store nodes is linear.
- This algo may not terminate & go on infinitely on one path. The soln to this issue is to choose cut-off depth
- Its complexity depends on no. of paths. It cannot check duplicate nodes.



= Algorithm:-

1) set status = 1 for each node in G
2) push the starting node A on stack & set status = 2
3) Repeat 4 & 5 until stack is empty
4) pop top Node N. process it & set its stat
5) push on stack all the neighbours of N tha are in ready state.
6) Exit.

= conclusion:-

Thus we have successfully implemented DFS.

Programiz

Java Online Compiler

Main.java

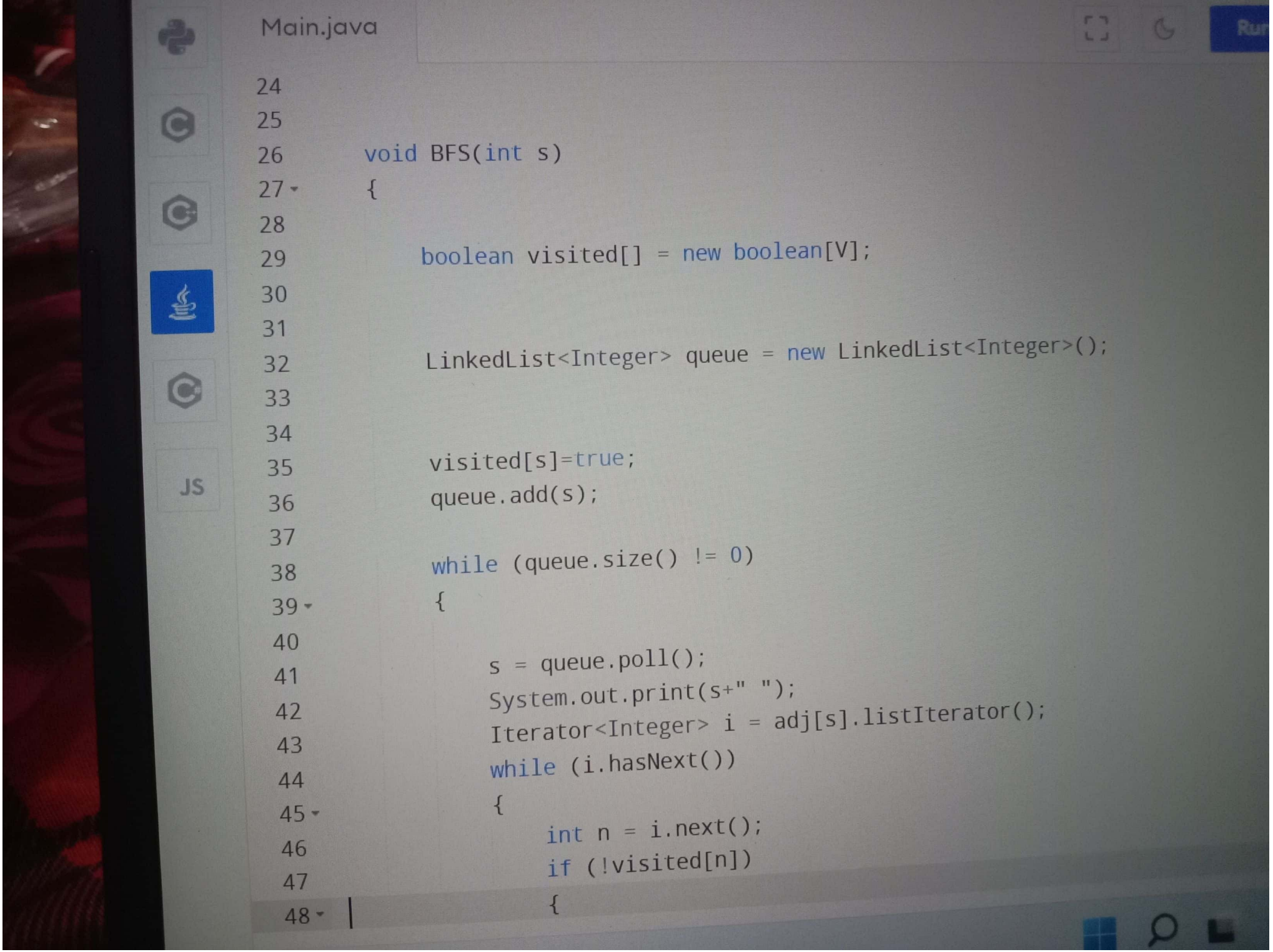
```java
import java.io.*;
import java.util.*;


class Graph
{
    private int V;
    private LinkedList<Integer> adj[];


    Graph(int v)
    {
        V = v;
        adj = new LinkedList[v];
        for (int i=0; i<v; ++i)
            adj[i] = new LinkedList();
    }


    void addEdge(int v,int w)
    {
        adj[v].add(w);
    }


```



Main.java

```
48                  {
49                      visited[n] = true;
50                      queue.add(n);
51                  }
52              }
53          }
54      }
55
56
57      public static void main(String args[])
58      {
59          Graph g = new Graph(8);
60
61  System.out.println(" Breadth First Traversal Is:\n");
62          g.addEdge(1,2);
63          g.addEdge(1,5);
64          g.addEdge(2,3);
65          g.addEdge(2,5);
66          g.addEdge(3,4);
67          g.addEdge(4,5);
68          g.addEdge(4,6);
69          g.addEdge(5,4);
70          g.BFS(1);
71
72
```

```
        }
    }


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

System.out.println(" Breadth First Traversal Is:\n");
        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(5,4);
        g.BFS(1);



    }
}
```

Compiler

ava

Run

Output

```
    }
}


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

System.out.println(" Breadth First Traversal Is:\n");
      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(5,4);
      g.BFS(1);



   }
}
```

```
java -cp /tmp/Kp3do8PCq3 Graph

Breadth First Traversal Is:

1 2 5 3 4 6
```

Main.java

Run

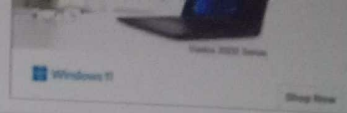
```
1  import java.io.*;
2  import java.util.*;
3
4
5  class Graph {
6      private int V;
7
8
9      private LinkedList<Integer> adj[];
10      Graph(int v)
11     {
12         V = v;
13         adj = new LinkedList[v];
14         for (int i = 0; i < v; ++i)
15             adj[i] = new LinkedList();
16     }
17     void addEdge(int v, int w)
18     {
19         adj[v].add(w);
20     }
21     void DFSUtil(int v, boolean visited[])
22     {
23
24         visited[v] = true;
25         System.out.print(v + " ");
```

Waiting for s.update.rubiconproject.com...

Main.java

```
24          visited[v] = true;
25          System.out.print(v + " ");
26          Iterator<Integer> i = adj[v].listIterator();
27          while (i.hasNext()) {
28              int n = i.next();
29              if (!visited[n])
30                  DFSUtil(n, visited);
31          }
32      }
33
34
35      void DFS(int v)
36      {
37          boolean visited[] = new boolean[V];
38          DFSUtil(v, visited);
39      }
40
41
42      public static void main(String args[])
43      {
44          Graph g = new Graph(8);
45           System.out.println(
46              " Depth First Traversal is: \n ");
47          g.addEdge(1,2);
48          g.addEdge(1,5);
```

Java Online Compiler



Main.java

```
36      {
37          boolean visited[] = new boolean[V];
38          DFSUtil(v, visited);
39      }
40
41
42      public static void main(String args[])
43      {
44          Graph g = new Graph(8);
45           System.out.println(
46              " Depth First Traversal is: \n ");
47          g.addEdge(1,2);
48          g.addEdge(1,5);
49          g.addEdge(2,3);
50          g.addEdge(2,5);
51          g.addEdge(3,4);
52          g.addEdge(4,5);
53          g.addEdge(4,6);
54          g.addEdge(5,6);
55
56
57
58          g.DFS(1);
59      }
60  }
```

33°C
Mostly cloudy

```
an visited[] = new boolean[V];
il(v, visited);


atic void main(String args[])

 g = new Graph(8);
em.out.println(
 Depth First Traversal is: \n ");
Edge(1,2);
Edge(1,5);
Edge(2,3);
Edge(2,5);
Edge(3,4);
Edge(4,5);
Edge(4,6);
Edge(5,6);



(1);
```

Run

Output

```
java -cp /tmp/Kp3do8PCq3 Graph
Depth First Traversal is:

1 2 3 4 5 6
```