GROUP B
Assignment no.10

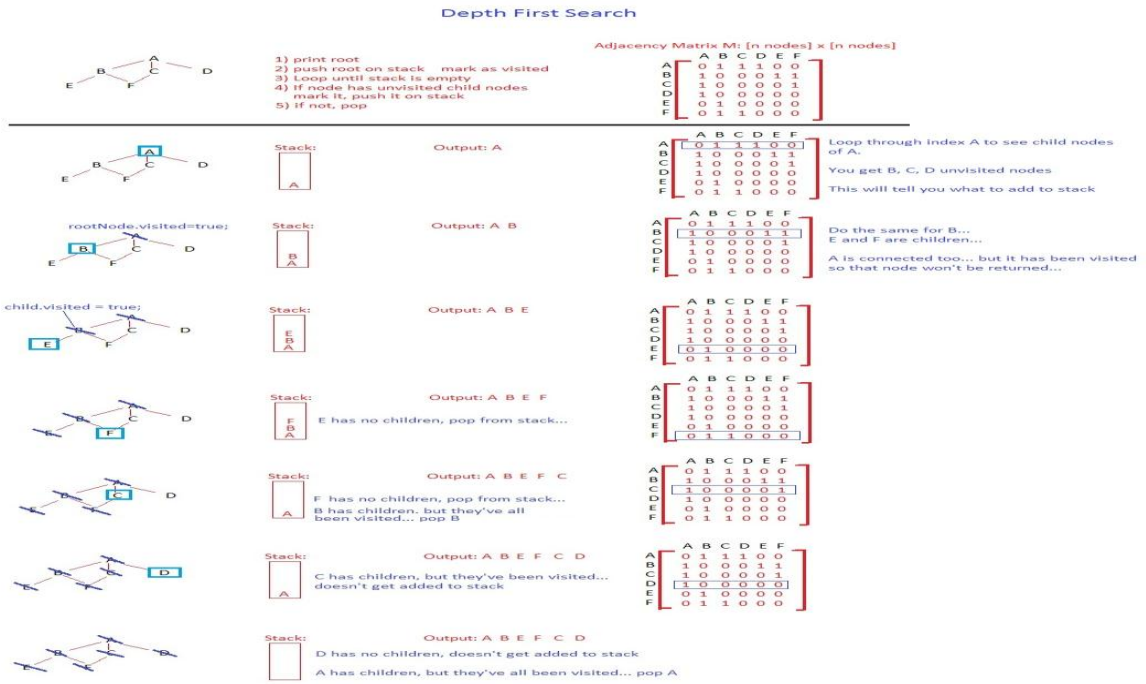
1. Problem Statement: Implementation of any 2 uninformed search methods for a LPG company that wants to install a gas pipeline between five cities. The cost of pipeline installation is given in a table maintained using XML/JSON use C++/ Python/ Java/ Scala with Eclipse for the application. Calculate time and space complexities.

2. Objective:
1. To apply algorithmic strategies while solving problems
2. To develop time and space efficient algorithms

3. Theory:
Strategies that know whether one non-goal state is "more promising" than another are called **informed search** or **heuristic search** strategies; they will be covered in Chapter 4. All search strategies are distinguished by the order in which nodes are expanded.

Depth-First Search:
Depth-first search always expands the *deepest* node in the current fringe of the search tree. The progress of the search is illustrated in Figure 3.12. The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors. As those nodes are expanded, they are dropped from the fringe, so then the search "backs up" to the next shallowest node that still has unexplored successors.
This strategy can be implemented by TREE-SEARCH with a last-in-first-out (LIFO)
queue, also known as a stack.

Breadth-First Search:
Breadth-first search is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on. In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.
Breadth-first search can be implemented by calling TREE-SEARCH with an empty fringe that is a first-in-first-out (FIFO) queue, assuring that the nodes that are visited first will be expanded first. In other words, calling TREE-SEARCH results in a breadth-first search.

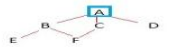


Depth First Search

1) print root
2) push root on stack mark as visited
3) Loop until stack is empty
4) If node has unvisited child nodes
   mark it, push it on stack
5) If not, pop

Adjacency Matrix M: [n nodes] x [n nodes]

	A	B	C	D	E	F
A	0	1	1	1	0	0
B	1	0	0	0	1	1
C	1	0	0	0	0	1
D	1	0	0	0	0	0
E	0	1	0	0	0	0
F	0	1	1	0	0	0



Stack: A

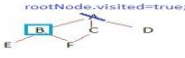
Output: A

Loop through index A to see child nodes of A.

You get B, C, D unvisited nodes

This will tell you what to add to stack

rootNode.visited=true;



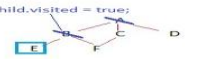
Stack: B A

Output: A B

Do the same for B...
E and F are children...

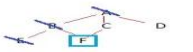
A is connected too... but it has been visited so that node won't be returned...

child.visited = true;



Stack: E B A

Output: A B E



Stack: F B A

Output: A B E F

E has no children, pop from stack...

Stack: A

Output: A B E F C

F has no children, pop from stack...
B has children. but they've all been visited... pop B

Stack: A

Output: A B E F C D

C has children, but they've been visited... doesn't get added to stack

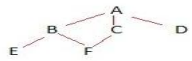
Stack:

Output: A B E F C D

D has no children, doesn't get added to stack

A has children, but they've all been visited... pop A

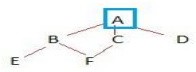
Breadth First Search



1) Print root node
2) Push root into Queue - mark as visited
3) Loop until Queue is empty
4) Remove node from queue, examine it
5) If examined node has unvisited children... mark them as visited...
   Print node
6) Insert unvisited nodes into Queue

Adjacency Matrix M: [n nodes] x [n nodes]

	A	B	C	D	E	F
A	0	1	1	1	0	0
B	1	0	0	0	1	1
C	1	0	0	0	0	1
D	1	0	0	0	0	0
E	0	1	0	0	0	0
F	0	1	1	0	0	0



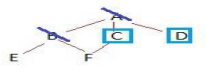
Queue: A

Output:

A has 3 unvisited children nodes

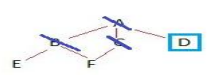
Queue: B C D

Output: A



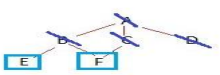
Queue: C D

Output: A B



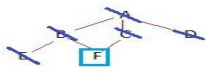
Queue: D

Output: A B C



Queue: E F

Output: A B C D

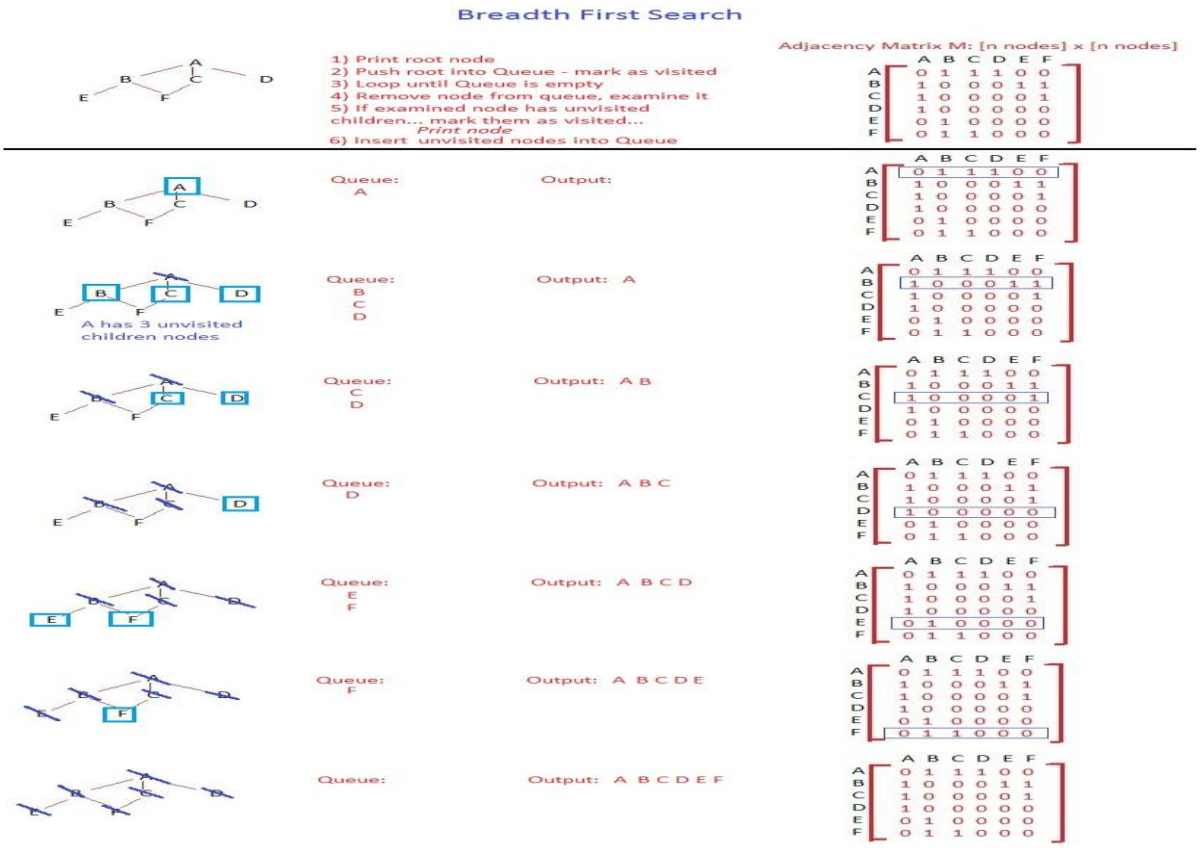


Queue: F

Output: A B C D E

Queue:

Output: A B C D E F



4. Algorithm:

1. Declare openlist
2. Add root node to our openlist
   while openlist not empty do following loops:
   a. retrieve then remove first node of our openlist
   b. check status of retrieved node
3. if it is the goal node then break loop and print solution
   if it is not goal node then:
   – expand retrieved node
   – ADD expanded node at THE BEGINNING of our openlist
   – continue loop
4. Openlist
   Using linked list data structure for openlist
5. Additional structure
   a. Hashmap to store currentState and its parent (without it we will not able to traceback)
   b. Hashmap to store currentState and its current level in tree (otherwise we can not limiting level and know how much step we already had)

5. Input For Assignment:
Path Adjancency matrix- File.xml

6. Expected Output: All the cities from start and finish connecting through common pipeline

7. Math Model:

Let S be the solution of the problem

S={s, e, i, o}

Where,

s=initial state
e=end state
i=input
o=output

s = {q, x, l| L!=NULL}
q=Queue/stack
x=starting vertex
l=List of vertices

i={v}

v=vertex of the tree

$v \in l$
$x \in l$

Enqueue(x)|Push(x)
$x \in v$

x=visited

o=Vertex reachable from v labelled as discovered and goal established
e= The system will be end in following conditions:
(i) If incorrect input is provided
(ii) Destination not mentioned correctly

8. Test Case(Testing):

Step	Test Step	Expected Result	Actual Result	Status
1	Enter valid data	Should accept the data	Accepted data	Pass
2	Enter valid goal state	Should accept the goal state	Successfully reached to goal state	Pass
3	Enter random goal state	Should accept goal state	Unsuccessful reaching goal	Fail
4	Random Initial State	Should accept state	Accepting as initial state	Pass

9. Time and Space Complexity:

Number of loops executing::BFS:5, DFS:3
Number recursive functions::3 calling each other and itself
Number of if cases::8loops*2=16 if cases executed 27 times

10. Conclusion:
Hence we have implemented the 2-uninformed search algorithms in order to install a pipeline between 5 cities.

11. Outcomes achieved (mark the outcomes achieved)

COURSE OUTCOME	ACHIEVED(√)
Problem solving abilities for smart devices.	√
Problem solving abilities for gamifications.	√
Problem solving abilities of pervasiveness, embedded security and NLP.	
To solve problems for multicore or distributed, concurrent/Parallel environments	

```
//lpg pipeline
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Map;
import java.util.Queue;

class Node {
  String city;
    LinkedList<Node> list;
}
public class SearchRomania {

    static Node[] node = new Node[20];
    static LinkedList<Node> path = new LinkedList<Node>();
    static LinkedList<Node> visited = new LinkedList<Node>();
    static Queue<Node> queue = new LinkedList<Node>();
    static boolean goalReached = false;
    static Map map = new HashMap();

    public static void initialize() {
        for (int i = 0; i < 5; i++) {
            node[i] = new Node();
            node[i].list = new LinkedList();
        }
        node[0].city = "oradea";
        node[1].city = "zerind";
        node[2].city = "arad";
        node[3].city = "timisoara";
        node[4].city = "lugoj";

//read from xml as integer
ReadXml rf=new ReadXml();
rf.readXML();
int city1=ReadXml.city1.charAt(1)-'0';

        node[0].list.add(node[ReadXml.city1.charAt(0)-'0']);
        node[0].list.add(node[ReadXml.city1.charAt(1)-'0']);
        node[1].list.add(node[ReadXml.city2.charAt(0)-'0']);
        node[1].list.add(node[ReadXml.city2.charAt(1)-'0']);
        node[2].list.add(node[ReadXml.city3.charAt(0)-'0']);
        node[2].list.add(node[ReadXml.city3.charAt(1)-'0']);
        node[2].list.add(node[ReadXml.city3.charAt(2)-'0']);
        node[3].list.add(node[ReadXml.city4.charAt(0)-'0']);
        node[3].list.add(node[ReadXml.city4.charAt(1)-'0']);
        node[4].list.add(node[ReadXml.city5.charAt(0)-'0']);
        node[4].list.add(node[ReadXml.city5.charAt(1)-'0']);

        System.out.println("Map for LPG pipeline installation::\n"+node[0].city+"-
>"+node[3].city+","+node[1].city+"\n"+node[1].city+"-
>"+node[2].city+","+node[0].city+"\n"+node[2].city+"-
>"+node[4].city+","+node[3].city+","+node[1].city+"\n"+node[3].city+"-
```

```
>"+node[2].city+","+node[4].city+"\n"+node[4].city+"->"+node[4].city+","+node[1].city);
    }
static void dfs(Node current, Node goal) {
  if (goalReached == false) {
          path.add(current);
          visited.add(current);

          if (current.city.equals(goal.city)) {
              goalReached = true;
             for (Node p : path) {
                System.out.println(p.city);
             }
          } else {
             for (Node next : current.list) {
                if (!visited.contains(next)) {
                   dfs(next, goal);
                   System.out.println(next.city);
             }
             }
             path.removeLast();
          }
       }
    }

    static void bfs(Node current, Node goal) {
       if (!visited.contains(current)) {
          visited.add(current);
          }
       if (goalReached == false) {
          for (Node child : current.list) {
             if (!visited.contains(child)) {
                visited.add(child);
                queue.add(child);
                map.put(child, current);

                if (child.city.equals(goal.city)) {
                   goalReached = true;
                   path.add(child);
                   path.add(current);

                   while (path.getLast() != visited.get(0)) {
                      current = (Node) map.get(current);
                      path.add(current);
                   }
                   while(!queue.isEmpty()){
                      System.out.println(queue.remove().city);
                    }
                   while (!path.isEmpty()) {
                      System.out.println(path.removeLast().city);
                   }
                   return;
                }
```

```
            }
          }
          bfs(queue.remove(), goal);
       }
    }

    public static void main(String[] args) {
       initialize();
       Node start=node[0];
       Node goal=node[1];
       String argr="oradea",argr1="arad";
       System.out.println("__________LPG Pipeline Installation between 5 cities_________");
System.out.println("Path starts from "+argr+" to "+argr1);

      for (int i = 0; i < 5; i++) {
          if (node[i].city.equals(argr)) {
             start = node[i];
          }
          if (node[i].city.equals(argr1)) {
             goal =node[i];
          }
       }
      System.out.println("Using DFS-----------");
         dfs(start, goal);

     System.out.println("Using BFS-----------");

         bfs(start, goal);
            System.out.println("Total number of nodes expanded: " + visited.size());
    }
}
```

//**lpg.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<lpg>
 <city id="1">
  <oradea>41</oradea>
  <zerind>20</zerind>
<arad>431</arad>
  <timisoara>24</timisoara>
<lugoj>13</lugoj>
</city>
</lpg>
```

//**ReadXml.java**

```
import java.io.File;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
```

```
public class ReadXml {
          public static String city1,city2,city3,city4,city5;
 public void readXML() {
  try {

   File xmlFile = new File("E:/BE/Cl2 Pract/xmlFiles/lpg.xml");
   DocumentBuilderFactory documentFactory = DocumentBuilderFactory
     .newInstance();
   DocumentBuilder documentBuilder = documentFactory
     .newDocumentBuilder();
   Document doc = documentBuilder.parse(xmlFile);

   doc.getDocumentElement().normalize();
   NodeList nodeList = doc.getElementsByTagName("city");

   for (int temp = 0; temp < nodeList.getLength(); temp++) {
    Node node = nodeList.item(temp);
    if (node.getNodeType() == Node.ELEMENT_NODE) {

     Element student = (Element) node;

     /*System.out.println("city id : "
       + student.getAttribute("id"));

     System.out.println("oradea : "
       + student.getElementsByTagName("oradea").item(0)
         .getTextContent());*/
     city1=student.getElementsByTagName("oradea").item(0).getTextContent();
    /* System.out.println("zerind : "
       + student.getElementsByTagName("zerind").item(0)
         .getTextContent());*/
    city2=student.getElementsByTagName("zerind").item(0).getTextContent();
    /* System.out.println("arad : "
       + student.getElementsByTagName("arad").item(0).getTextContent());*/
    city3=student.getElementsByTagName("arad").item(0).getTextContent();
    /* System.out.println("timisoara : "
       + student.getElementsByTagName("timisoara").item(0)
         .getTextContent());*/
   city4=student.getElementsByTagName("timisoara").item(0).getTextContent();
     /*System.out.println("lugoj : "+ student.getElementsByTagName("lugoj").item(0)
                 .getTextContent());*/
   city5=student.getElementsByTagName("lugoj").item(0).getTextContent();

    }

   }
  } catch (Exception e) {
   e.printStackTrace();
  }
 }
}
```

Output:

Map for LPG pipeline installation::
oradea->timisoara,zerind
zerind->arad,oradea
arad->lugoj,timisoara,zerind
timisoara->arad,lugoj
lugoj->lugoj,zerind
_____LPG Pipeline Installation between 5 cities_____
Path starts from oradea to arad
Using DFS----------
oradea
lugoj
zerind
arad
arad
zerind
timisoara
lugoj
Using BFS----------
Total number of nodes expanded: 4