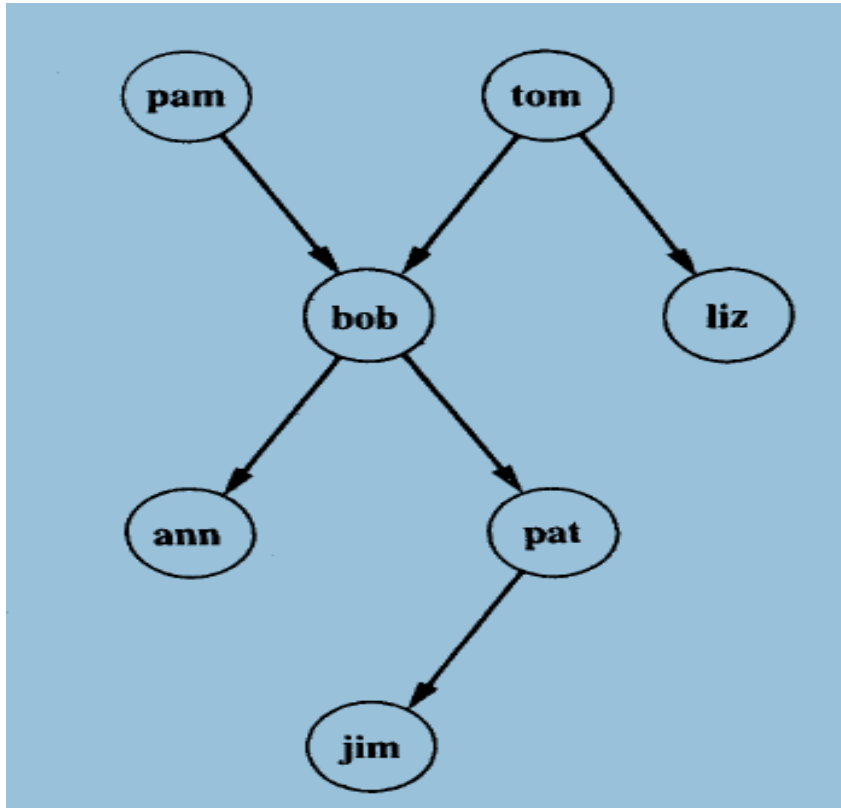


Program and Output



a family relation

```
parent( pam, bob).  
parent( tom, bob).  
parent( tom, liz).  
parent( bob, ann).  
parent( bob, pat).  
parent( pat, jim).
```

Output :

```
GNU Prolog console
File Edit Terminal Prolog Help
GNU Prolog 1.5.0 (64 bits)
Compiled Jul  8 2021, 12:22:53 with gcc
Copyright (C) 1999-2021 Daniel Diaz

compiling C:/GNU-Prolog/Expi/Experiment1.pl for byte code...
C:/GNU-Prolog/Expi/Experiment1.pl compiled, 6 lines read - 687 bytes written, 10 ms
| ?- change_directory('C:/GNU-Prolog/Expi').

yes
| ?- parent(jim, X).

no
| ?- parent(X, jim).

X = pat

yes
| ?- parent(paffi, X), parent(X, pat).

no
| ?- parent(paffi, X), parent(X, Y), parent(Y, jim).

no
| ?- parent(X, pat).|

X = bob ?

yes
| ?- parent(liz, X).

no
| ?- parent(X, pat), parent(Y, X).

X = bob
Y = pam ?

yes
| ?- 'C:/GNU-Prolog/Expi/Experiment1.pl'
```

Water Jug Problem

Aim : To implement a water jug problem.

Theory :

You are on the side of the river. You are given a m litre jug and a n litre jug where $0 < m < n$. Both the jugs are initially empty. The jugs don't have markings to allow measuring smaller quantities. You have to use the jugs to measure d litres of water where $d < n$. Determine the minimum number of operations to be performed to obtain d litres of water in one of jug.

The operations you can perform are:

Empty a Jug

Fill a Jug

Pour water from one jug to the other until one of the jugs is either empty or full.

There are several ways of solving this problem including BFS and DP

The operators to be used to solve the problem can be describes as shown below. They are represented as rules whose left side are matched against the current state and whose right side describes the new state that results from applying the rules.

We have two jugs a 4 gallon and a 3 gallon.

Consider the following Rule set:

1. $(x,y) \rightarrow (4,y)$ fill the 4 gallon jug

If $x < 4$.

1. $(x,y) \rightarrow (x,3)$ fill the 3 gallon jug

If $x < 3$

1. $(x,y) \rightarrow (x-d,y)$ pour some water out of the 4-gallon jug.

If $x > 0$

1. $(x,y) \rightarrow (x-d,y)$ pour some water out of the 3-gallon jug.

If $y > 0$

1. $(x,y) \rightarrow (0,y)$ empty the 4-gallon jug on the ground

If $x > 0$

1. $(x,y) \rightarrow (x,0)$ empty the 3-gallon jug on the ground

If $y > 0$

1. $(x,y) \rightarrow (4,y-(4-x))$ pour water from the 3-gallon jug into the 4-gallon

If $x+y \geq 4$ and $y > 0$ jug until the 4-gallon jug is full

1. $(x,y) \rightarrow (x-(3-y),3)$ pour water from the 4-gallon jug into the 3-gallon

If $x+y \geq 3$ and $x > 0$ jug until the 3-gallon jug is full.

1. $(x,y) \rightarrow (x+y,0)$ pour all the water from the 3-gallon jug into

If $x+y \leq 4$ and $y > 0$ the 3-gallon jug.

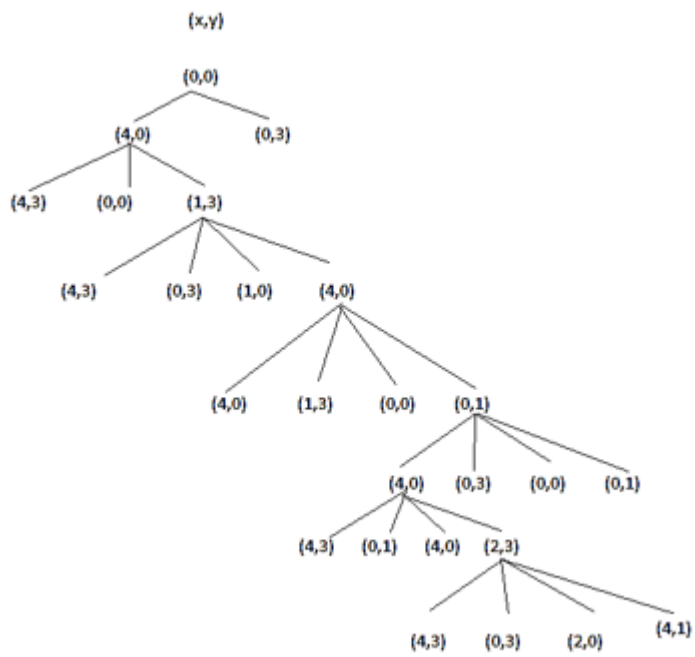
1. $(x,y) \rightarrow (0,x+y)$ pour all the water from the 4-gallon jug into

If $x+y \leq 3$ and $x > 0$ the 3-gallon jug.

1. $(0,2) \rightarrow (2,0)$ pour the 2-gallon from the 3-gallon jug into the 4-gallon jug.
2. $(2,y) \rightarrow (0,x)$ empty the 2 gallon in the 4 gallon on the ground

Gallons in the 4-gallon jug	Gallons in the 3-gallon	Rule Applied
0	0	
0	3	2
3	0	9
3	3	2
4	2	7
0	2	5 or 12
2	0	9 or 11

One solution to the water jug problem.



Program and Output

1. Breadth First Search (BFS)

Program :

```
// Java program to print BFS traversal from a given source vertex.  
// BFS(int s) traverses vertices reachable from s.
```

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

```
// This class represents a directed graph using adjacency list  
// representation
```

```
public class Graph  
{  
    private int V; // No. of vertices  
    private LinkedList<Integer> adj[]; //Adjacency Lists  
  
    // Constructor  
    Graph(int v)  
    {  
        V = v;  
        adj = new LinkedList[v];  
        for (int i=0; i<v; ++i)  
            adj[i] = new LinkedList();  
    }  
  
    // Function to add an edge into the graph  
  
    void addEdge(int v,int w)  
    {  
        adj[v].add(w);  
    }  
  
    // prints BFS traversal from a given source s
```

```

void BFS(int s)
{
    // Mark all the vertices as not visited(By default
    // set as false)

    boolean visited[] = new boolean[V];

    // Create a queue for BFS
    LinkedList<Integer> queue = new LinkedList<Integer>();

    // Mark the current node as visited and enqueue it

    visited[s]=true;
    queue.add(s);
    while (queue.size() != 0)
    {
        // Dequeue a vertex from queue and print it
        s = queue.poll();
        System.out.print(s+" ");
        // Get all adjacent vertices of the dequeued vertex s
        // If a adjacent has not been visited, then mark it
        // visited and enqueue it

        Iterator<Integer> i = adj[s].listIterator();
        while (i.hasNext())
        {
            int n = i.next();
            if (!visited[n])
            {
                visited[n] = true;
                queue.add(n);
            }
        }
    }

}

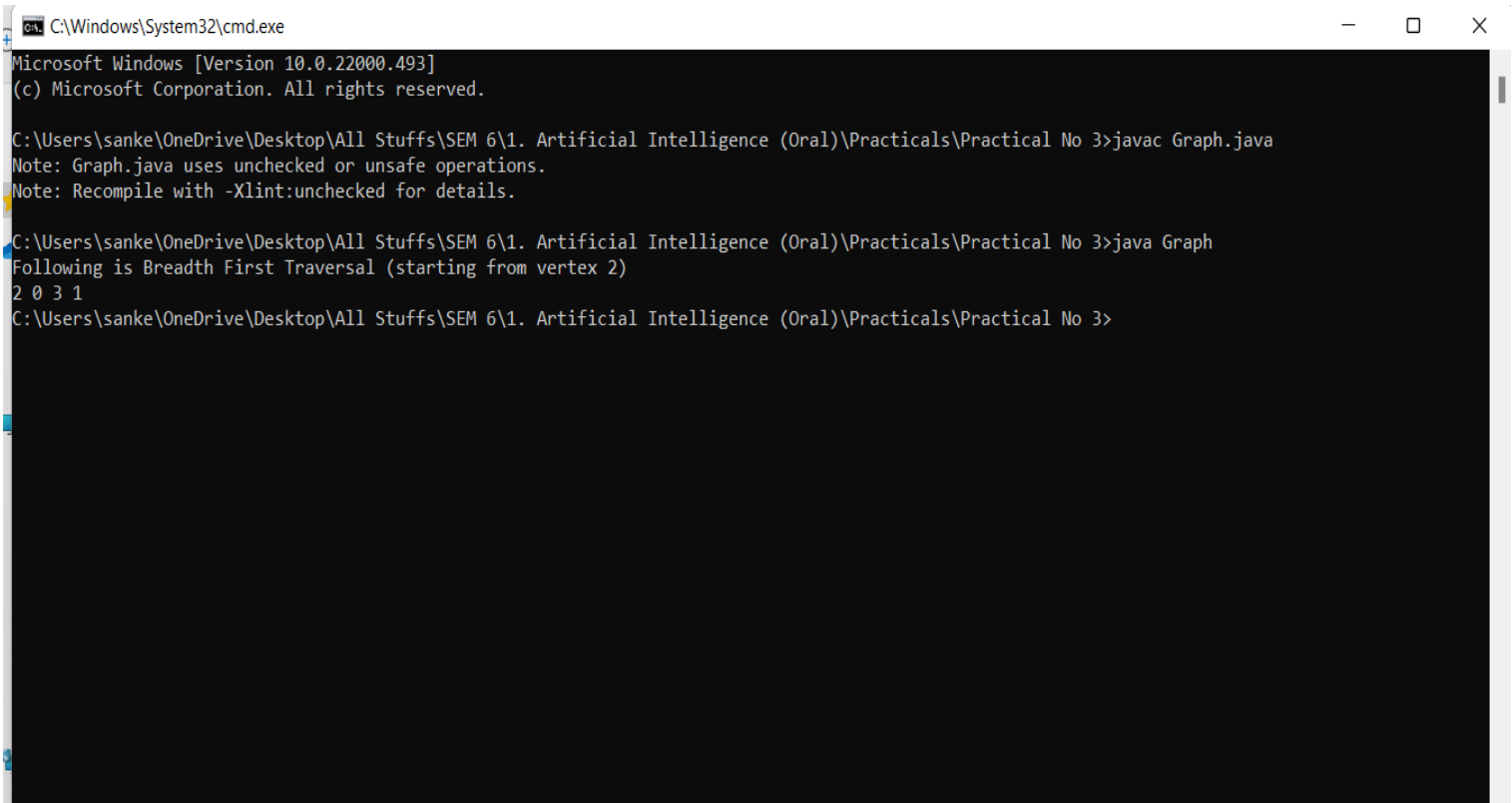
// Driver method to

```

```
public static void main(String args[])
{
    Graph g = new Graph(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    System.out.println("Following is Breadth First Traversal "+
        "(starting from vertex 2)");
    g.BFS(2);
}
}
```

Output :



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22000.493]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sanke\OneDrive\Desktop\All Stuffs\SEM 6\1. Artificial Intelligence (Oral)\Practicals\Practical No 3>javac Graph.java
Note: Graph.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

C:\Users\sanke\OneDrive\Desktop\All Stuffs\SEM 6\1. Artificial Intelligence (Oral)\Practicals\Practical No 3>java Graph
Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1
C:\Users\sanke\OneDrive\Desktop\All Stuffs\SEM 6\1. Artificial Intelligence (Oral)\Practicals\Practical No 3>
```


2. Depth First Search (DFS)

Program :

```
// Java program to print DFS
// mtraversal from a given given
// graph
import java.io.*;
import java.util.*;

// This class represents a
// directed graph using adjacency
// list representation
public class DGraph
{
    private int V; // No. of vertices

    // Array of lists for
    // Adjacency List Representation
    private LinkedList<Integer> adj[];

    // Constructor
    @SuppressWarnings("unchecked") DGraph(int v)
    {
        V = v;
        adj = new LinkedList[V];
        for (int i = 0; i < v; ++i)
            adj[i] = new LinkedList();
    }

    // Function to add an edge into the graph
    void addEdge(int v, int w)
    {
        adj[v].add(w); // Add w to v's list.
    }

    // A function used by DFS
    void DFSUtil(int v, boolean visited[])
    {
        // Mark the current node as visited and print it
```

```

visited[v] = true;
System.out.print(v + " ");

// Recur for all the vertices adjacent to this
// vertex
Iterator<Integer> i = adj[v].listIterator();
while (i.hasNext()) {
    int n = i.next();
    if (!visited[n])
        DFSUtil(n, visited);
}
}

// The function to do DFS traversal.
// It uses recursive
// DFSUtil()
void DFS(int v)
{
    // Mark all the vertices as
    // not visited(set as
    // false by default in java)
    boolean visited[] = new boolean[V];

    // Call the recursive helper
    // function to print DFS
    // traversal
    DFSUtil(v, visited);
}

// Driver Code
public static void main(String args[])
{
    DGraph g = new DGraph(4);

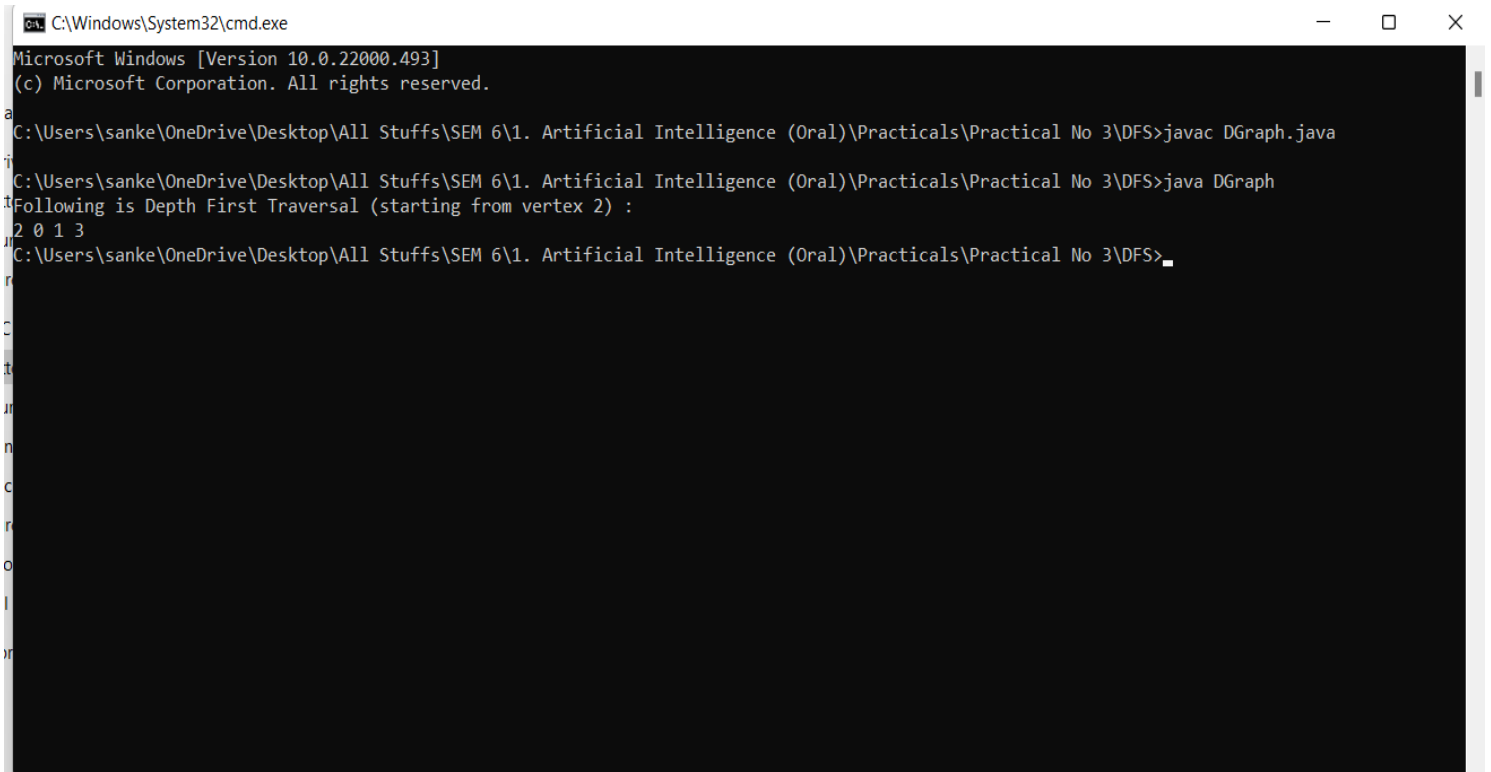
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);
}

```

```
System.out.println(
    "Following is Depth First Traversal "
    + "(starting from vertex 2) : ");

    g.DFS(2);
}
}
```

Output:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22000.493]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sanke\OneDrive\Desktop\All Stuffs\SEM 6\1. Artificial Intelligence (Oral)\Practicals\Practical No 3\DFS>javac DGraph.java

C:\Users\sanke\OneDrive\Desktop\All Stuffs\SEM 6\1. Artificial Intelligence (Oral)\Practicals\Practical No 3\DFS>java DGraph
Following is Depth First Traversal (starting from vertex 2) :
2 0 1 3
C:\Users\sanke\OneDrive\Desktop\All Stuffs\SEM 6\1. Artificial Intelligence (Oral)\Practicals\Practical No 3\DFS>
```

Program and Output

Program :

```
#include<iostream>
#include<cmath>
using namespace std;
#define MAX 10
class nqueen
{
    public: void placequeen(int);
           int place(int[MAX],int);
};
int nqueen :: place(int x[MAX],int k)
{
    int i;
    for(i=1;i<k;i++)
        if(x[i]==x[k] || abs(x[i]-x[k])==abs(i-k)) return 0;
    return 1;
}
void nqueen :: placequeen(int n)
{
    int k,count,x[MAX],i;
    k=1; count=0;x[k]=0;
    cout<< "\n The different solutions are as follows";
    cout<< "\n\nEach solution indicates the column in which the Queen";
    cout<< "\n Is to be placed in different rows";
    while(k!=0)
    {
        x[k]=x[k]+1;
        while((x[k]<=n) && (!place(x,k)))
            x[k]=x[k]+1;
        if(x[k]<=n)
        {
            if(k==n)
            {
                count=count+1;
                cout<<endl<<endl;
            }
        }
    }
}
```

```

        for(i=1;i<=n;i++)
            cout<<x[i]<<"\t";//getch

    }
    else
    {
        k++;
        x[k]=0;
    }
}
else
k--;
}
}
int main()
{
    int n;
    nqueen nq;
    cout<<"\nEnter the number of Queen :"; cin>>n;
    nq.placequeen(n);return 0;
}

```

Output :

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22000.556]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sanke\OneDrive\Desktop\All Stuffs\8. SEM 6\1. Artificial Intelligence (Oral)\Practicals\Practical No 4>g++ nQueen.cpp
C:\Users\sanke\OneDrive\Desktop\All Stuffs\8. SEM 6\1. Artificial Intelligence (Oral)\Practicals\Practical No 4>a
Enter the number of Queen :5

The different solutions are as follows

Each solution indicates the column in which the Queen
Is to be placed in different rows
1      3      5      2      4
1      4      2      5      3
2      4      1      3      5
2      5      3      1      4
3      1      4      2      5
3      5      2      4      1
4      1      3      5      2
4      2      5      3      1
5      2      4      1      3
5      3      1      4      2
C:\Users\sanke\OneDrive\Desktop\All Stuffs\8. SEM 6\1. Artificial Intelligence (Oral)\Practicals\Practical No 4>_
```