Experiment No : 2

Aim : Write a C/C++/Java program to solve the water jug problem using Depth First Search Technique.

Theory :
Depth First Search
The DFS algorithm is a recursive algorithm that uses the idea of backtracking. It involves exhaustive searches of all the nodes by going ahead, if possible, else by backtracking.

Here, the word backtrack means that when you are moving forward and there are no more nodes along the current path, you move backwards on the same path to find nodes to traverse. All the nodes will be visited on the current path till all the unvisited nodes have been traversed after which the next path will be selected.

This recursive nature of DFS can be implemented using stacks. The basic idea is as follows:

Pick a starting node and push all its adjacent nodes into a stack.

Pop a node from stack to select the next node to visit and push all its adjacent nodes into a stack.

Repeat this process until the stack is empty. However, ensure that the nodes that are visited are marked. This will prevent you from visiting the same node more than once. If you do not mark the nodes that are visited and you visit the same node more than once, you may end up in an infinite loop.

Code :
```
#include<iostream>
#include<list>
using namespace std;

const int JUGA=3;
const int JUGB=4;
const int CAPACITY[]={JUGA,JUGB};
const int DESIRED_CAP=2;
bool visited[JUGA+1][JUGB+1];

typedef struct state{
int jug[2];
}state;
list<state> stack;


void push(int x,int y){
stack.push_front(state{{x,y}});
cout<<"\t Pushing "<<x<<" "<<y<<endl;
}

void empty(state a){
if(a.jug[0]!=0)
push(0,a.jug[1]);

if(a.jug[1]!=0)
push(a.jug[0],0);
```

```
}

void fill(state a){
if(a.jug[0]!=CAPACITY[0])
push(CAPACITY[0],a.jug[1]);
if(a.jug[1]!=CAPACITY[1])
push(a.jug[0],CAPACITY[1]);
}

void transfer(state a){
int x,y;
if(a.jug[0]!=CAPACITY[0] && a.jug[1]!=0){
x=a.jug[0];
y=a.jug[1];
x=x+y;
if(x>CAPACITY[0]){
y=x-CAPACITY[0];
x=CAPACITY[0];
}
else
y=0;
push(x,y);
}
if(a.jug[1]!=CAPACITY[1] && a.jug[0]!=0){
x=a.jug[0];
y=a.jug[1];
y=x+y;
if(y>CAPACITY[1]){
x=y-CAPACITY[1];
y=CAPACITY[1];
}
else
x=0;
push(x,y);
}
}
bool goalState(state a){
if(a.jug[0]==DESIRED_CAP)
return true;
return false;
}
void initializeVisited(){
int i,j;
for(i=0;i<CAPACITY[0];i++){
for(j=0;j<CAPACITY[1];j++){
visited[i][j]=false;
}
}
}
int main(){
initializeVisited();
```

```
state curr=state{{0,0}};
push(curr.jug[0],curr.jug[1]);
while(!stack.empty()){
curr=stack.front();
stack.pop_front();
if(!visited[curr.jug[0]][curr.jug[1]]){
visited[curr.jug[0]][curr.jug[1]]=true;
cout<<"Visiting "<<curr.jug[0]<<" "<<curr.jug[1]<<endl;
if(goalState(curr)){
cout<<"Goal state reached"<<endl;
break;
}
empty(curr);
fill(curr);
transfer(curr);
}
else{
// cout<<"Already visited "<<curr.jug[0]<<" "<<curr.jug[1]<<endl;
}
}
}
```

Output :

```
    Pushing 0 0
Visiting 0 0
    Pushing 3 0
    Pushing 0 4
Visiting 0 4
    Pushing 0 0
    Pushing 3 4
    Pushing 3 1
Visiting 3 1
    Pushing 0 1
    Pushing 3 0
    Pushing 3 4
    Pushing 0 4
Visiting 3 4
    Pushing 0 4
    Pushing 3 0
Visiting 3 0
    Pushing 0 0
    Pushing 3 4
    Pushing 0 3
Visiting 0 3
    Pushing 0 0
    Pushing 3 3
    Pushing 0 4
    Pushing 3 0
Visiting 3 3
    Pushing 0 3
    Pushing 3 0
    Pushing 3 4
    Pushing 2 4
Visiting 2 4
Goal state reached
```

Conclusion : This Experiment was successfully executed and the output was verified.

Deepraj Bhosale
181105016