ASSIGNMENT: 02

Name: Abhijeet Biswas
SRN: 201900400
Roll No: 05
Div: B

Question 2_a. Implement the Water jug problem using Breadth First Search.

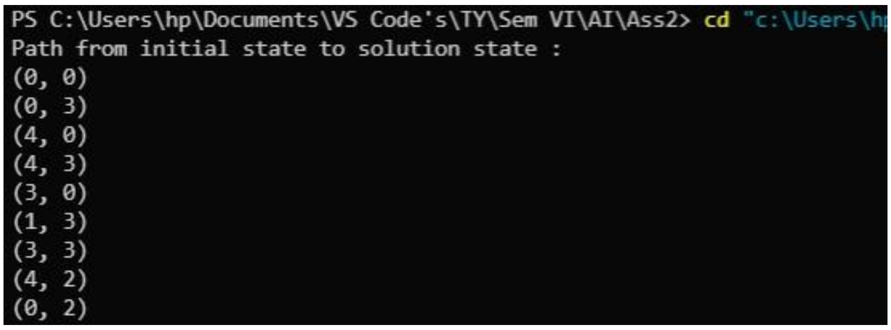
Code:

```
/*2_a. Implement the Water jug problem using Breadth First Search*/

#include <bits/stdc++.h>

#define pii pair<int, int>

#define mp make_pair

using namespace std;

void BFS(int a, int b, int target)

{

    map<pii, int> m;

    bool isSolvable = false;

    vector<pii> path;

    queue<pii> q;

    q.push({0, 0});

    while (!q.empty())

    {

        pii u = q.front();
```

```
q.pop();
if (m[{u.first, u.second}] == 1)
    continue;
if ((u.first > a || u.second > b ||
     u.first < 0 || u.second < 0))
    continue;
path.push_back({u.first, u.second});
m[{u.first, u.second}] = 1;
if (u.first == target || u.second == target)
{
    isSolvable = true;
    if (u.first == target)
    {
        if (u.second != 0)
            path.push_back({u.first, 0});
    }
    else
    {
        if (u.first != 0)
            path.push_back({0, u.second});
    }
    int sz = path.size();
    for (int i = 0; i < sz; i++)
        cout << "(" << path[i].first
             << ", " << path[i].second << ")\n";
    break;
```

```
        }
        q.push({u.first, b});
        q.push({a, u.second});
        for (int ap = 0; ap <= max(a, b); ap++)
        {
            int c = u.first + ap;
            int d = u.second - ap;
            if (c == a || (d == 0 && d >= 0))
                q.push({c, d});
            c = u.first - ap;
            d = u.second + ap;
            if ((c == 0 && c >= 0) || d == b)
                q.push({c, d});
        }
        q.push({a, 0});
        q.push({0, b});
    }
    if (!isSolvable)
        cout << "No solution";
}
int main()
{
    int Jug1 = 4, Jug2 = 3, target = 2;
    cout << "Path from initial state "
            "to solution state :\n";
    BFS(Jug1, Jug2, target);
    return 0;
}
```

Output:



```
PS C:\Users\hp\Documents\VS Code's\TY\Sem VI\AI\Ass2> cd "c:\Users\h
Path from initial state to solution state :
(0, 0)
(0, 3)
(4, 0)
(4, 3)
(3, 0)
(1, 3)
(3, 3)
(4, 2)
(0, 2)
```

Question 2_b. Implement the Water jug problem using Depth First Search.

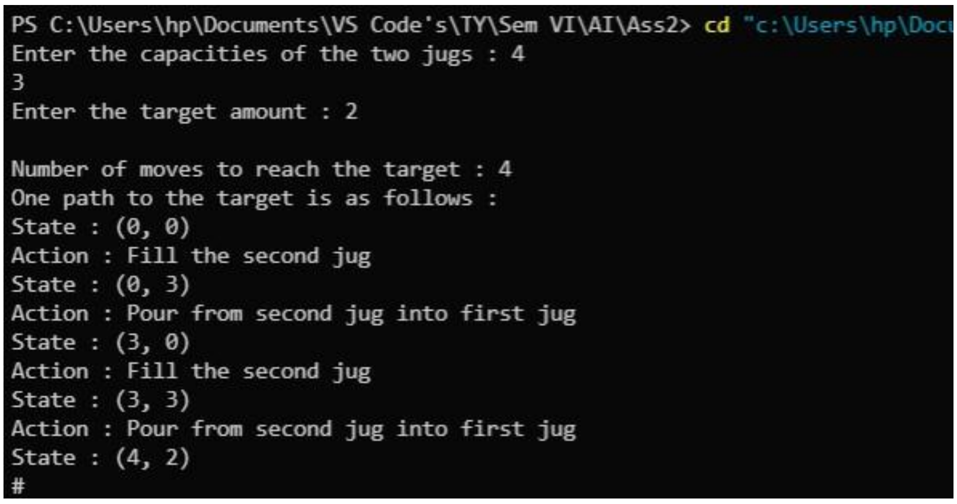
Code:

```
/*2_b. Implement the Water jug problem using Depth First Search*/

#include <cstdio>
#include <stack>
#include <map>
#include <algorithm>
using namespace std;
// x and y are the amounts of water in litres in the two jugs respectively
struct state
{
    int x, y;
    bool operator<(const state &that) const
    {
        if (x != that.x)
            return x < that.x;
        return y < that.y;
    }
};
int capacity_x, capacity_y, target;
```

```
void dfs(state start, stack<pair<state, int>> &path)
{
    stack<state> s;
    state goal = (state){-1, -1};
    map<state, pair<state, int>> parentOf;
    s.push(start);
    parentOf[start] = make_pair(start, 0);
    while (!s.empty())
    {
        state top = s.top();
        s.pop();
        if (top.x == target || top.y == target)
        {
            goal = top;
            break;
        }
        if (top.x < capacity_x)
        {
            state child = (state){capacity_x, top.y};
            if (parentOf.find(child) == parentOf.end())
            {
                s.push(child);
                parentOf[child] = make_pair(top, 1);
            }
        }
        if (top.y < capacity_y)
        {
            state child = (state){top.x, capacity_y};
            if (parentOf.find(child) == parentOf.end())
            {
                s.push(child);
                parentOf[child] = make_pair(top, 2);
            }
        }
        if (top.x > 0)
        {
            state child = (state){0, top.y};
            if (parentOf.find(child) == parentOf.end())
            {
                s.push(child);
                parentOf[child] = make_pair(top, 3);
            }
        }
        if (top.y > 0)
        {
            state child = (state){top.x, 0};
            if (parentOf.find(child) == parentOf.end())
            {
                s.push(child);
                parentOf[child] = make_pair(top, 4);
            }
        }
        if (top.y > 0)
        {
            state child = (state){min(top.x + top.y, capacity_x), max(0, top.x + top.y -
capacity_x)};
            if (parentOf.find(child) == parentOf.end())
            {
                s.push(child);
                parentOf[child] = make_pair(top, 5);
            }
        }
        if (top.x > 0)
        {
            state child = (state){max(0, top.x + top.y - capacity_y), min(top.x + top.y,
capacity_y)};
            if (parentOf.find(child) == parentOf.end())
            {
                s.push(child);
                parentOf[child] = make_pair(top, 6);
            }
        }
    }
    if (goal.x == -1 || goal.y == -1)
        return;
    path.push(make_pair(goal, 0));
    while (parentOf[path.top().first].second != 0)
        path.push(parentOf[path.top().first]);
}
int main()
```

```
{
   stack<pair<state, int>> path;
   printf("Enter the capacities of the two jugs : ");
   scanf("%d %d", &capacity_x, &capacity_y);
   printf("Enter the target amount : ");
   scanf("%d", &target);
   dfs((state){0, 0}, path);
   if (path.empty())
      printf("\nTarget cannot be reached.\n");
   else
   {
      printf("\nNumber of moves to reach the target : %d\nOne path to the
target is as follows :\n", path.size() - 1);
      while (!path.empty())
      {
         state top = path.top().first;
         int rule = path.top().second;
         path.pop();
         switch (rule)
         {
         case 0:
            printf("State : (%d, %d)\n#\n", top.x, top.y);
            break;
         case 1:
            printf("State : (%d, %d)\nAction : Fill the first jug\n", top.x, top.y);
            break;
         case 2:
            printf("State : (%d, %d)\nAction : Fill the second jug\n", top.x, top.y);
            break;
         case 3:
            printf("State : (%d, %d)\nAction : Empty the first jug\n", top.x, top.y);
            break;
         case 4:
            printf("State : (%d, %d)\nAction : Empty the second jug\n", top.x,
top.y);
            break;
         case 5:
            printf("State : (%d, %d)\nAction : Pour from second jug into first jug\n",
top.x, top.y);
            break;
            case 6:
                printf("State : (%d, %d)\nAction : Pour from first jug into second jug\n",
top.x, top.y);
                break;
            }
        }
    }
    return 0;
}
```

Output:



```
PS C:\Users\hp\Documents\VS Code's\TY\Sem VI\AI\Ass2> cd "c:\Users\hp\Docu
Enter the capacities of the two jugs : 4
3
Enter the target amount : 2

Number of moves to reach the target : 4
One path to the target is as follows :
State : (0, 0)
Action : Fill the second jug
State : (0, 3)
Action : Pour from second jug into first jug
State : (3, 0)
Action : Fill the second jug
State : (3, 3)
Action : Pour from second jug into first jug
State : (4, 2)
#
```