AI Assignment 2

-------------------------

Name: Yash Oswal

Div: B Roll no.: 38

SRN: 201901226

--------------------------

1. Water jug problem using BFS

Code:

from collections import deque

def BFS(a, b, target):

m = {}

isSolvable = False

path = []

q = deque()

q.append((0, 0))

while (len(q) > 0):

u = q.popleft()

if ((u[0], u[1]) in m):

continue

if ((u[0] > a or u[1] > b or

u[0] < 0 or u[1] < 0)):

continue

path.append([u[0], u[1]])

m[(u[0], u[1])] = 1

if (u[0] == target or u[1] == target):

isSolvable = True

if (u[0] == target):

if (u[1] != 0):

path.append([u[0], 0])

else:

if (u[0] != 0):

path.append([0, u[1]])

sz = len(path)

for i in range(sz):

print("(", path[i][0], ",",

path[i][1], ")")

break

q.append([u[0], b]) # Fill Jug2

q.append([a, u[1]]) # Fill Jug1

for ap in range(max(a, b) + 1):

c = u[0] + ap

d = u[1] - ap

if (c == a or (d == 0 and d >= 0)):

q.append([c, d])

c = u[0] - ap

d = u[1] + ap

if ((c == 0 and c >= 0) or d == b):

q.append([c, d])

q.append([a, 0])

q.append([0, b])

if (not isSolvable):

print ("No solution")

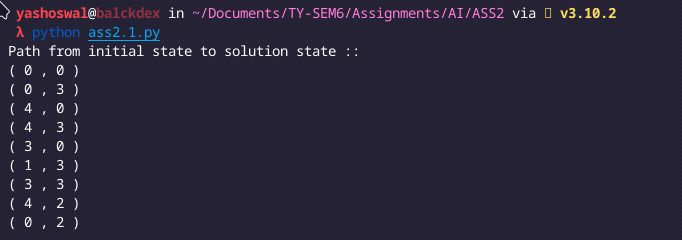
if \_\_name\_\_ == '\_\_main\_\_':

Jug1, Jug2, target = 4, 3, 2

print("Path from initial state to solution state ::")

BFS(Jug1, Jug2, target)

Output:



1. Water jug problem using DFS

Code:

#include <algorithm>

#include <cstdio>

#include <map>

#include <stack>

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:

