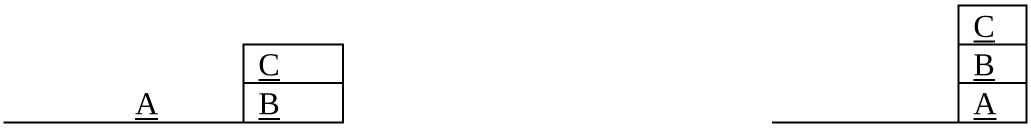
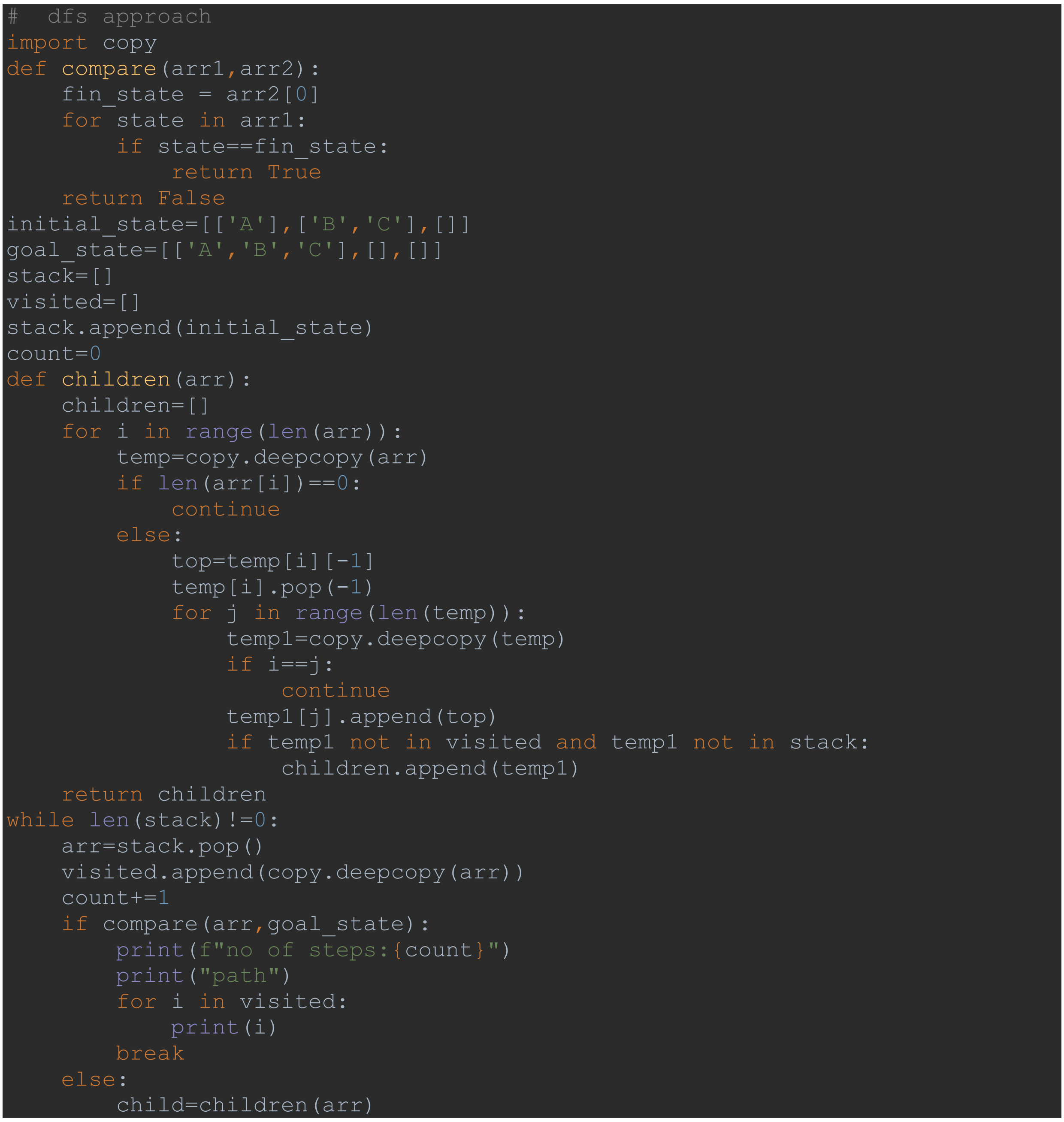
Computer Science and Engineering Department
Artificial Intelligence (UCS-521)
Lab Assignment-4

Q1. Solve the following blocks world problem using Depth First Search.

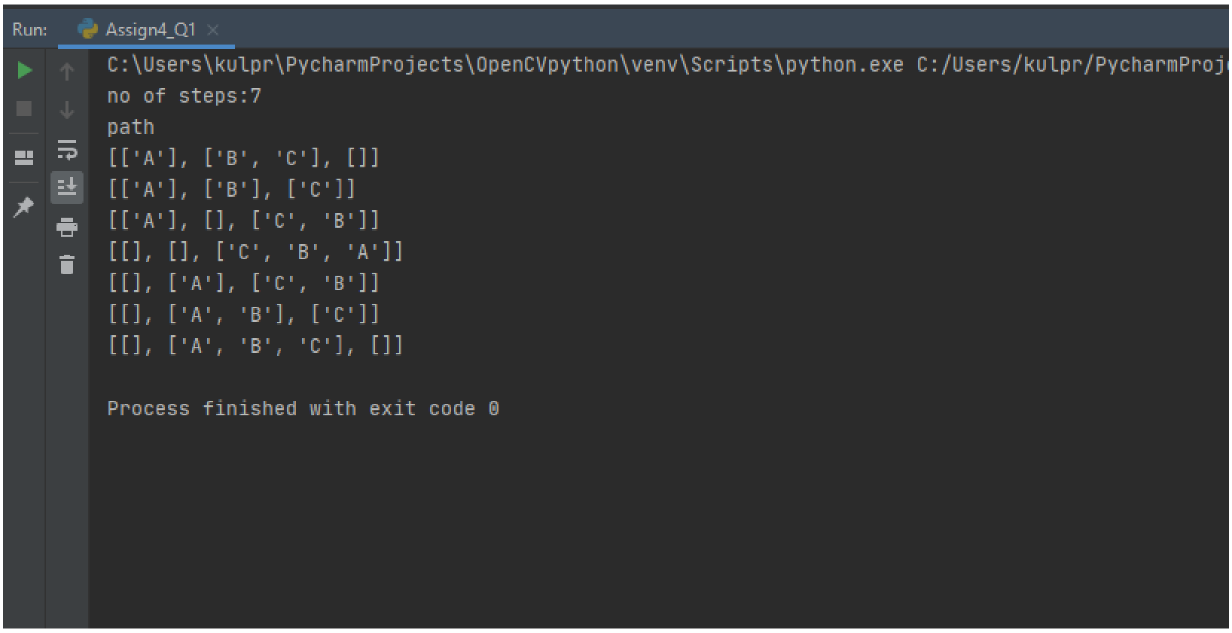


CODE:



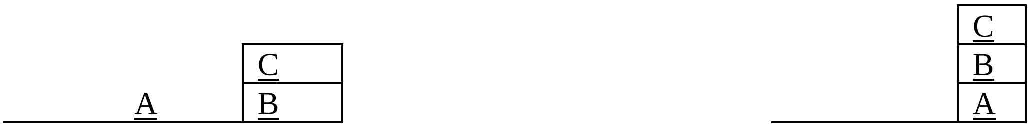
```
#  dfs approach
import copy
def compare(arr1,arr2):
    fin_state = arr2[0]
    for state in arr1:
        if state==fin_state:
            return True
    return False
initial_state=[['A'],['B','C'],[]]
goal_state=[['A','B','C'],[],[]]
stack=[]
visited=[]
stack.append(initial_state)
count=0
def children(arr):
    children=[]
    for i in range(len(arr)):
        temp=copy.deepcopy(arr)
        if len(arr[i])==0:
            continue
        else:
            top=temp[i][-1]
            temp[i].pop(-1)
            for j in range(len(temp)):
                temp1=copy.deepcopy(temp)
                if i==j:
                    continue
                temp1[j].append(top)
                if temp1 not in visited and temp1 not in stack:
                    children.append(temp1)
    return children
while len(stack)!=0:
    arr=stack.pop()
    visited.append(copy.deepcopy(arr))
    count+=1
    if compare(arr,goal_state):
        print(f"no of steps:{count}")
        print("path")
        for i in visited:
            print(i)
        break
    else:
        child=children(arr)
        for c in child:
            stack.append(copy.deepcopy(c))
```

OUTPUT:



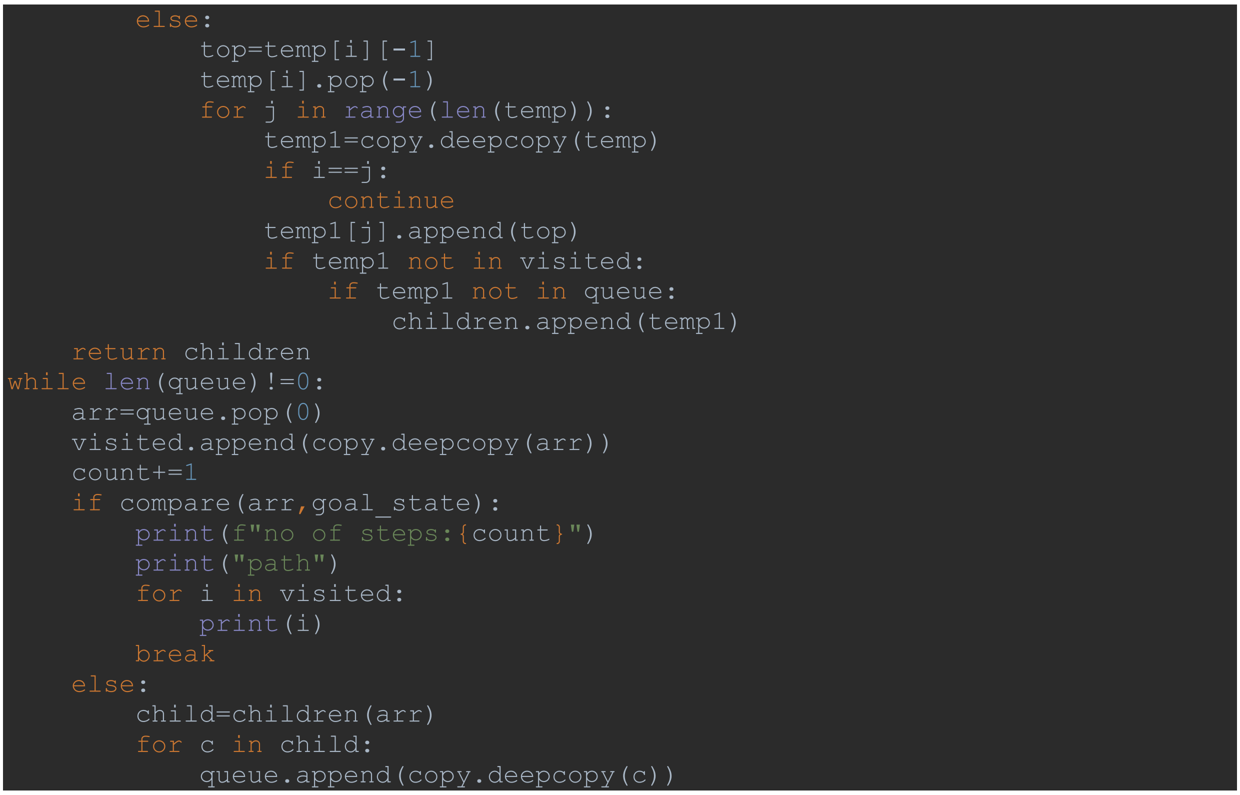
```
C:\Users\kulpr\PycharmProjects\OpenCVpython\venv\Scripts\python.exe C:/Users/kulpr/PycharmProj
no of steps:7
path
[['A'], ['B', 'C'], []]
[['A'], ['B'], ['C']]
[['A'], [], ['C', 'B']]
[[], [], ['C', 'B', 'A']]
[[], ['A'], ['C', 'B']]
[[], ['A', 'B'], ['C']]
[[], ['A', 'B', 'C'], []]

Process finished with exit code 0
```

Q2. Solve the following blocks world problem using Breadth First Search. Compare the results with the question 1.



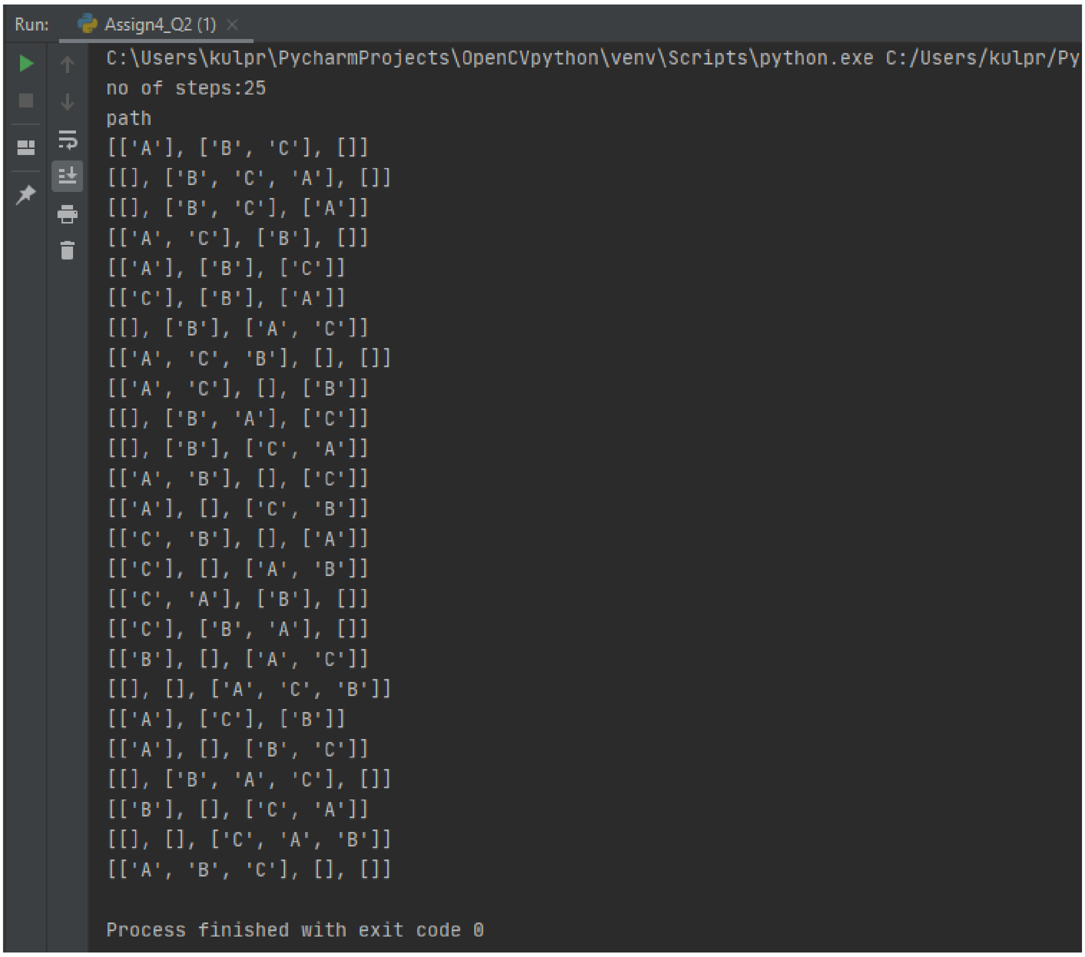
CODE:

```
# bfs approach
import copy
def compare(arr1,arr2):
    fin_state = arr2[0]
    for state in arr1:
        if state==fin_state:
            return True
    return False
initial_state=[['A'],['B','C'],[]]
goal_state=[['A','B','C'],[],[]]
queue=[]
visited=[]
queue.append(initial_state)
count=0
def children(arr):
    children=[]
    for i in range(len(arr)):
        temp=copy.deepcopy(arr)
        if len(arr[i])==0:
            continue
```



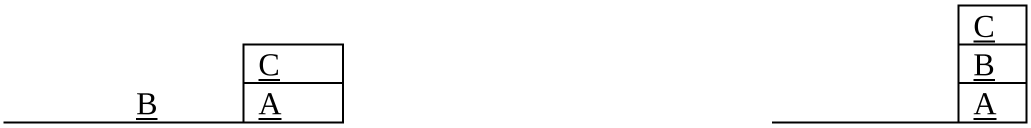
```
        else:
            top=temp[i][-1]
            temp[i].pop(-1)
            for j in range(len(temp)):
                temp1=copy.deepcopy(temp)
                if i==j:
                    continue
                temp1[j].append(top)
                if temp1 not in visited:
                    if temp1 not in queue:
                        children.append(temp1)
    return children
while len(queue)!=0:
    arr=queue.pop(0)
    visited.append(copy.deepcopy(arr))
    count+=1
    if compare(arr,goal_state):
        print(f"no of steps:{count}")
        print("path")
        for i in visited:
            print(i)
        break
    else:
        child=children(arr)
        for c in child:
            queue.append(copy.deepcopy(c))
```

OUTPUT:



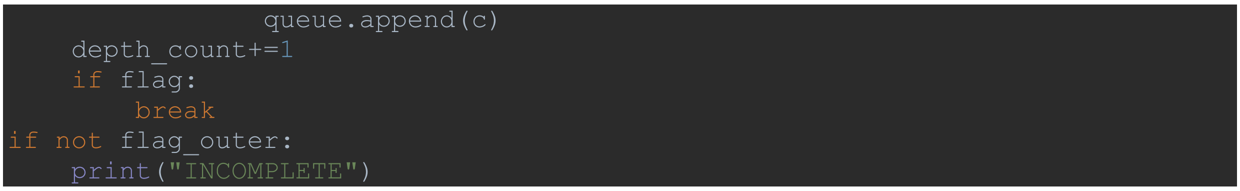
```
C:\Users\kulpr\PycharmProjects\OpenCVpython\venv\Scripts\python.exe C:/Users/kulpr/Py
no of steps:25
path
[['A'], ['B', 'C'], []]
[[], ['B', 'C', 'A'], []]
[[], ['B', 'C'], ['A']]
[['A', 'C'], ['B'], []]
[['A'], ['B'], ['C']]
[['C'], ['B'], ['A']]
[[], ['B'], ['A', 'C']]
[['A', 'C', 'B'], [], []]
[['A', 'C'], [], ['B']]
[[], ['B', 'A'], ['C']]
[[], ['B'], ['C', 'A']]
[['A', 'B'], [], ['C']]
[['A'], [], ['C', 'B']]
[['C', 'B'], [], ['A']]
[['C'], [], ['A', 'B']]
[['C', 'A'], ['B'], []]
[['C'], ['B', 'A'], []]
[['B'], [], ['A', 'C']]
[[], [], ['A', 'C', 'B']]
[['A'], ['C'], ['B']]
[['A'], [], ['B', 'C']]
[[], ['B', 'A', 'C'], []]
[['B'], [], ['C', 'A']]
[[], [], ['C', 'A', 'B']]
[['A', 'B', 'C'], [], []]

Process finished with exit code 0
```

Q3. Write a python program to solve the following blocks world problem using Depth Limited Search (D=1). Check if it is complete or incomplete for depth = 1.



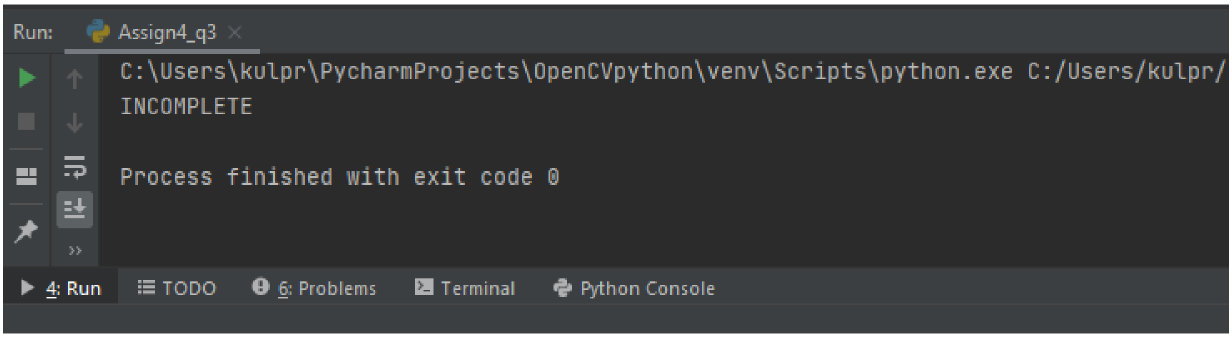
CODE:

```
#  depth limited search
import copy
def compare(arr1,arr2):
    fin_state = arr2[0]
    for state in arr1:
        if state==fin_state:
            return True
    return False
initial_state=[['b'],['a','c'],[]]
goal_state=[['a','b','c'],[],[]]
queue=[]
visited=[]
depth=1
queue.append(initial_state)
count=0
def children(arr,stack):
    children=[]
    for i in range(len(arr)):
        temp=copy.deepcopy(arr)
        if len(arr[i])==0:
            continue
        else:
            top=temp[i][-1]
            temp[i].pop(-1)
            for j in range(len(temp)):
                temp1=copy.deepcopy(temp)
                if i==j:
                    continue
                temp1[j].append(top)
                if temp1 not in stack and temp1 not in visited:
                    children.append(temp1)
    return children
depth_count=0
flag_outer=False
flag=False
while depth_count<=depth:
    stack=[]
    while len(queue)!=0:
        stack.append(queue.pop(0))
    while(len(stack)!=0):
        arr=stack.pop()
        visited.append(copy.deepcopy(arr))
        if compare(arr,goal_state):
            print(f"COMPLETE at depth:{depth_count}")
            flag=True
            flag_outer=True
            break
        else:
            child=children(arr,stack)
            for c in child:
```



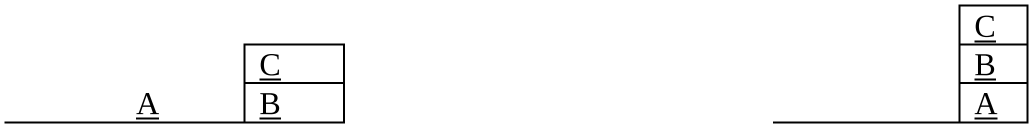
```
                queue.append(c)
    depth_count+=1
    if flag:
        break
if not flag_outer:
    print("INCOMPLETE")
```

OUTPUT:

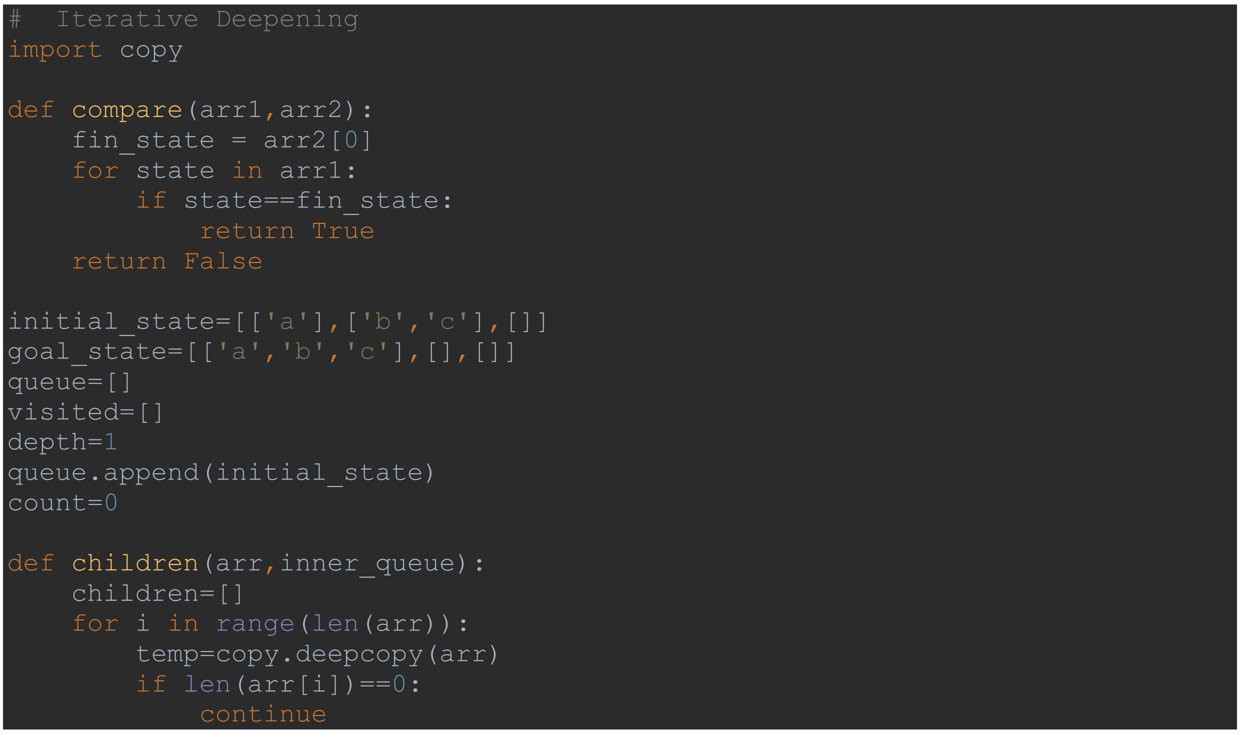


```
C:\Users\kulpr\PycharmProjects\OpenCVpython\venv\Scripts\python.exe C:/Users/kulpr/
INCOMPLETE

Process finished with exit code 0
```

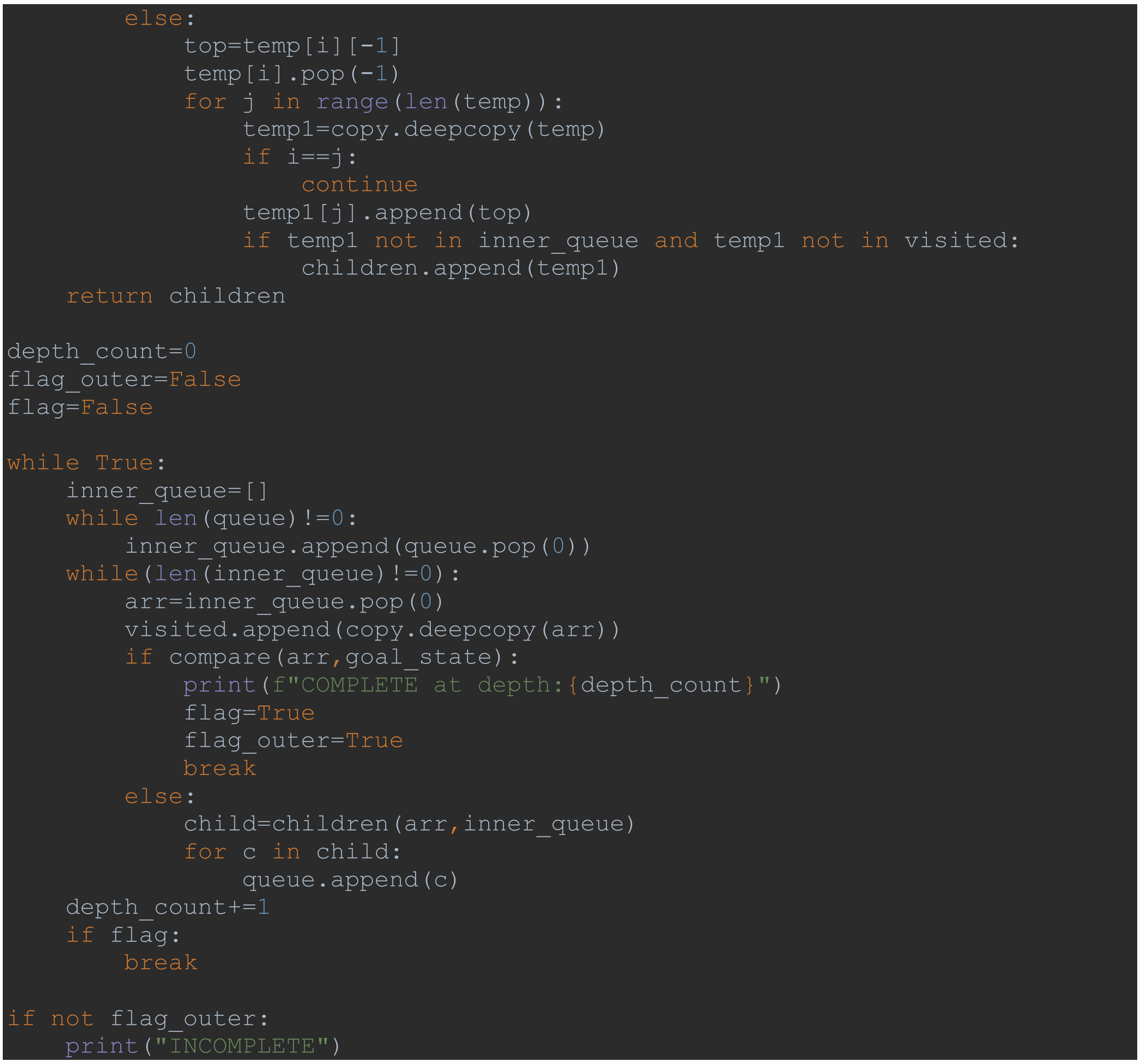
Q4. Find the depth at which the goal is achieved using Iterative Deepening for the following problem



CODE:

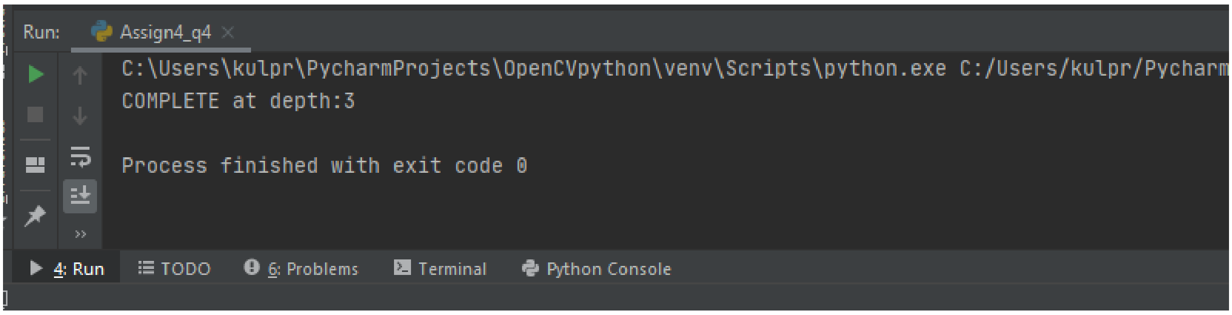


```
#  Iterative Deepening
import copy

def compare(arr1,arr2):
    fin_state = arr2[0]
    for state in arr1:
        if state==fin_state:
            return True
    return False

initial_state=[['a'],['b','c'],[]]
goal_state=[['a','b','c'],[],[]]
queue=[]
visited=[]
depth=1
queue.append(initial_state)
count=0

def children(arr,inner_queue):
    children=[]
    for i in range(len(arr)):
        temp=copy.deepcopy(arr)
        if len(arr[i])==0:
            continue
```



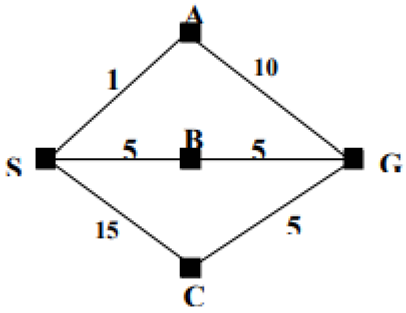
```
        else:
            top=temp[i][-1]
            temp[i].pop(-1)
            for j in range(len(temp)):
                temp1=copy.deepcopy(temp)
                if i==j:
                    continue
                temp1[j].append(top)
                if temp1 not in inner_queue and temp1 not in visited:
                    children.append(temp1)
    return children

depth_count=0
flag_outer=False
flag=False

while True:
    inner_queue=[]
    while len(queue)!=0:
        inner_queue.append(queue.pop(0))
    while(len(inner_queue)!=0):
        arr=inner_queue.pop(0)
        visited.append(copy.deepcopy(arr))
        if compare(arr,goal_state):
            print(f"COMPLETE at depth:{depth_count}")
            flag=True
            flag_outer=True
            break
        else:
            child=children(arr,inner_queue)
            for c in child:
                queue.append(c)
    depth_count+=1
    if flag:
        break

if not flag_outer:
    print("INCOMPLETE")
```

OUTPUT:



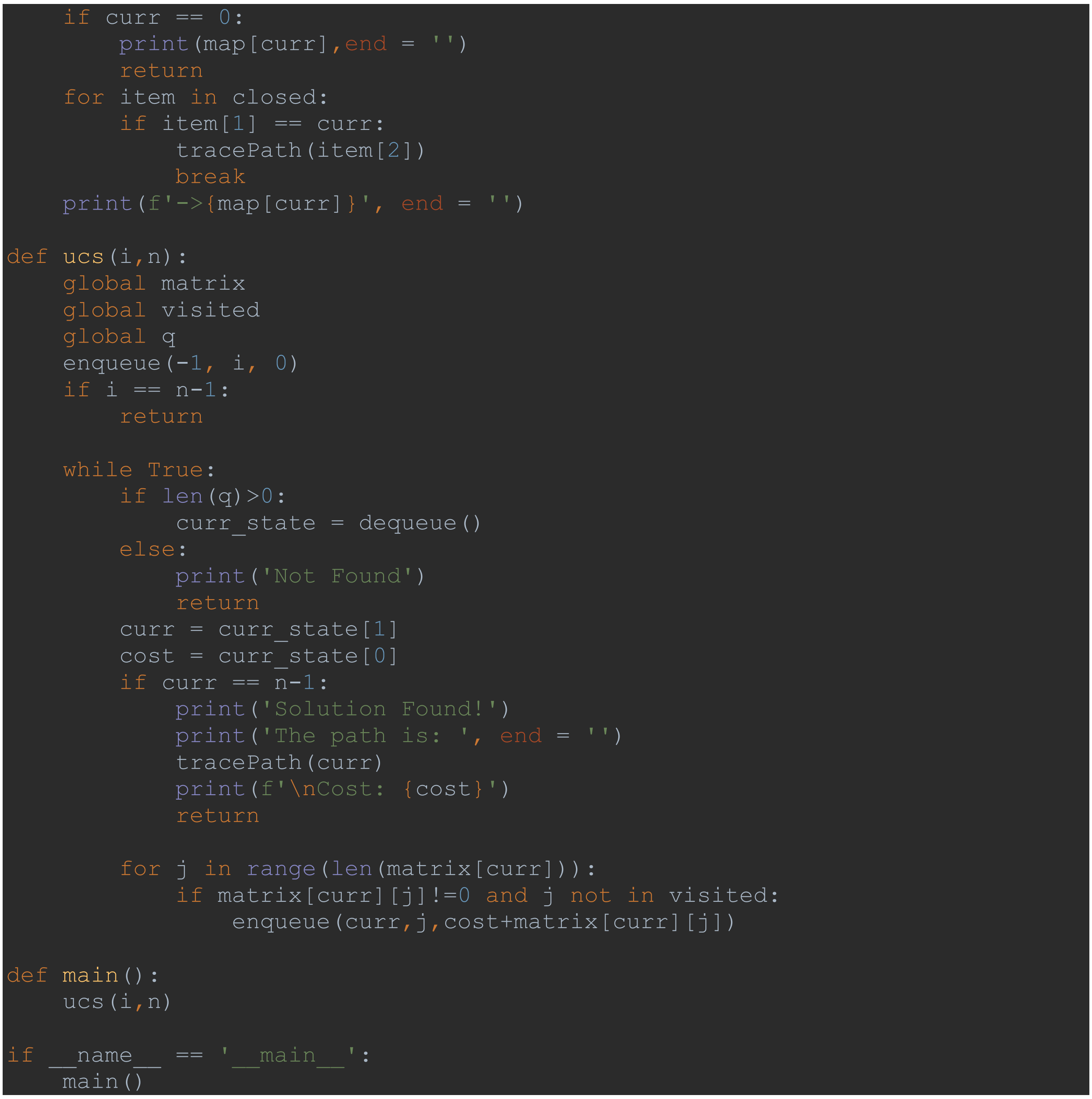
```
C:\Users\kulpr\PycharmProjects\OpenCVpython\venv\Scripts\python.exe C:/Users/kulpr/Pycharm
COMPLETE at depth:3

Process finished with exit code 0
```

Q5. Solve this given problem using Uniform Cost search.



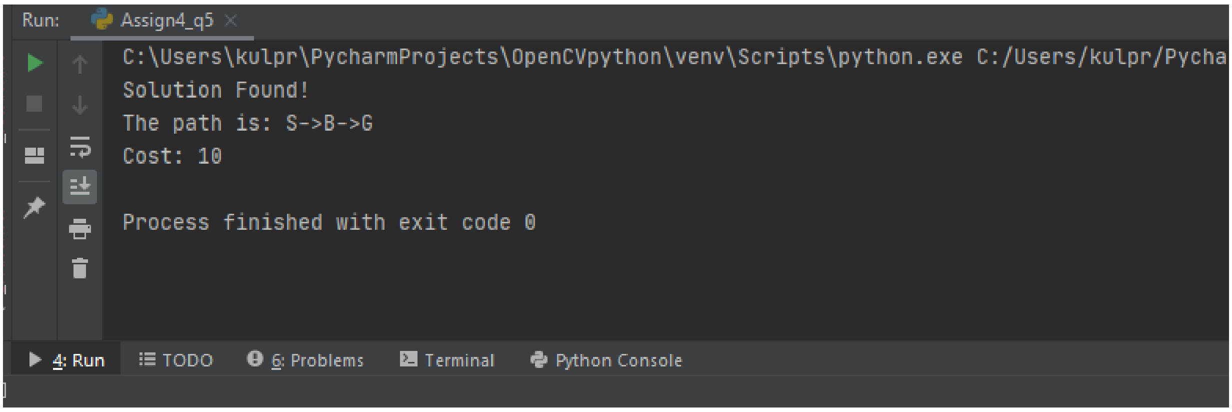
CODE:

```
import sys

matrix = [[0, 1, 5, 15, 0],
          [1, 0, 0, 0, 10],
          [5, 0, 0, 0, 5],
          [15, 0, 0, 0, 5],
          [0, 10, 5, 5, 0]
          ]
map = {0: 'S', 1: 'A', 2: 'B', 3: 'C', 4: 'G'}
visited = []
n = len(matrix)
i = 0
q = []
open = []
closed = []
def enqueue(parent,s,val):
    global q
    flag = 0
    idx = 0
    for item in q:
        if item[1] == s:
            flag = 1
            break
        idx += 1
    if flag == 1:
        if q[idx][0] > val:
            q[idx][0] = val
            q[idx][2] = parent
    else:
        q = q + [[val,s,parent]]

def dequeue():
    global q
    global visited
    global closed
    q.sort()
    visited = visited + [q[0][1]]
    closed = closed + [q[0]]
    temp = q[0]
    del q[0]
    return (temp)

def tracePath(curr):
    global closed
```



```
    if curr == 0:
        print(map[curr],end = '')
        return
    for item in closed:
        if item[1] == curr:
            tracePath(item[2])
            break
    print(f'->{map[curr]}', end = '')

def ucs(i,n):
    global matrix
    global visited
    global q
    enqueue(-1, i, 0)
    if i == n-1:
        return

    while True:
        if len(q)>0:
            curr_state = dequeue()
        else:
            print('Not Found')
            return
        curr = curr_state[1]
        cost = curr_state[0]
        if curr == n-1:
            print('Solution Found!')
            print('The path is: ', end = '')
            tracePath(curr)
            print(f'\nCost: {cost}')
            return

        for j in range(len(matrix[curr])):
            if matrix[curr][j]!=0 and j not in visited:
                enqueue(curr,j,cost+matrix[curr][j])

def main():
    ucs(i,n)

if __name__ == '__main__':
    main()
```

OUTPUT:



```
C:\Users\kulpr\PycharmProjects\OpenCVpython\venv\Scripts\python.exe C:/Users/kulpr/Pycha
Solution Found!
The path is: S->B->G
Cost: 10

Process finished with exit code 0
```