**Computer Science and Engineering Department**

**Artificial Intelligence (UCS-521)**

**Lab Assignment-4**

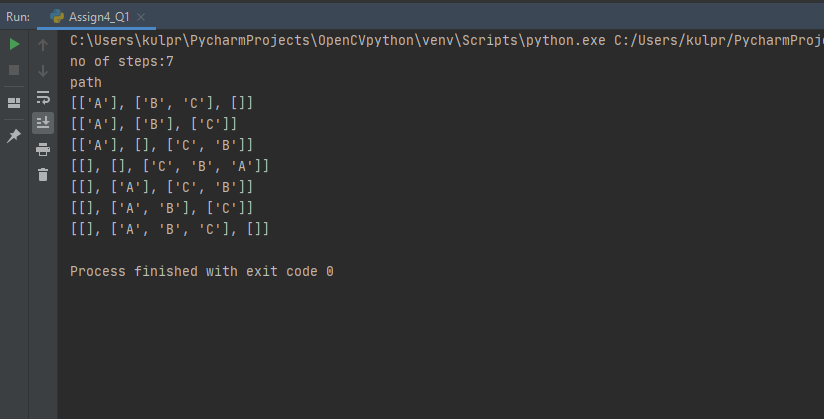
Q1. Solve the following blocks world problem using Depth First Search.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  | C |
|  | C |  |  | B |
| A | B |  |  | A |

**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:**



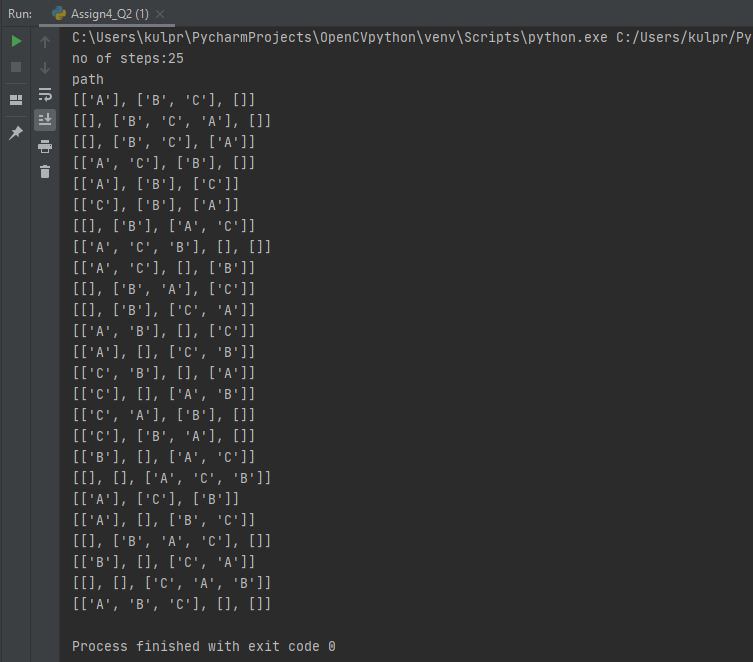
Q2. Solve the following blocks world problem using Breadth First Search. Compare the results with the question 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  | C |
|  | C |  |  | B |
| A | B |  |  | A |

**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:**



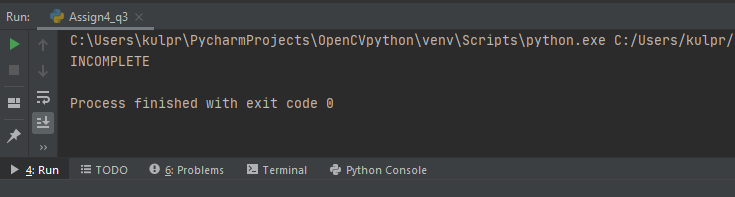
Q3. Write a python program to solve the following blocks world problem using Depth Limited Search (D=1). Check if it is complete or incomplete for depth = 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  | C |
|  | C |  |  | B |
| B | A |  |  | A |

**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:**



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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  | C |
|  | C |  |  | B |
| A | B |  |  | A |

**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

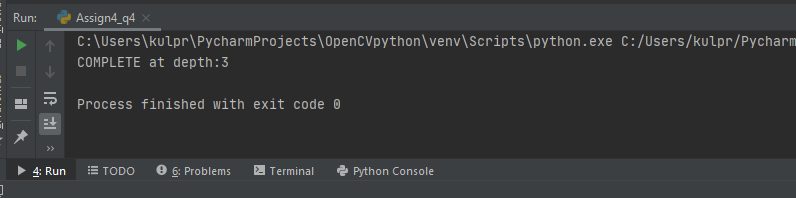
def children(arr,inner\_queue):  
 children=[]  
 for i in range(len(arr)):  
 temp=copy.deepcopy(arr)  
 if len(arr[i])==0:  
 continue  
 else:  
 top=temp[i][-1]  
 temp[i].pop(-1)  
 for j in range(len(temp)):  
 temp1=copy.deepcopy(temp)  
 if i==j:  
 continue  
 temp1[j].append(top)  
 if temp1 not in inner\_queue and temp1 not in visited:  
 children.append(temp1)  
 return children

depth\_count=0  
flag\_outer=False  
flag=False

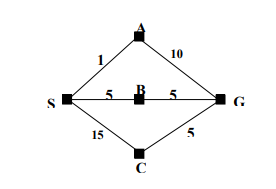
while True:  
 inner\_queue=[]  
 while len(queue)!=0:  
 inner\_queue.append(queue.pop(0))  
 while(len(inner\_queue)!=0):  
 arr=inner\_queue.pop(0)  
 visited.append(copy.deepcopy(arr))  
 if compare(arr,goal\_state):  
 print(f"COMPLETE at depth:{depth\_count}")  
 flag=True  
 flag\_outer=True  
 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:**



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:**

