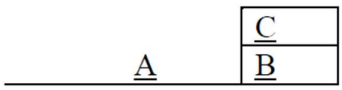
UCS411: ARTIFICIAL INTELLIGENCE

Assignment-4 (Lab Assignment)

Submitted To: - Ms. Anupam Garg

***QUES1*. Solve the following blocks world problem using Depth First Search**

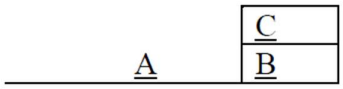


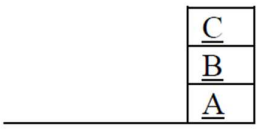
ANS.

```
import copy
initial=[['A'],['B','C'],[]]
goal1=[['A','B','C'],[],[]]
visit=[]
stack=[]
visit.append(initial)
stack.append(initial)
i=0
def child(curr,visit):
   val1=[]
   for j in range(len(curr)):
      if (len(curr[j])>0):
         p = copy.deepcopy(curr)
         val = curr[j][-1]
         p[j].pop(-1)
         k = 0
         while (k < len(curr)):
            p1 = copy.deepcopy(curr)
            p1[j].pop(-1)
            if (k != j):
               p1[k].append(val)
               if (p1 not in visit):
                  visit.append(p1)
                  val1.append(p1)
            k += 1

   return val1

def solve(initial,goal):
   if(goal in initial):
```

```
      visit.append(goal)
      return(visit)
   else:
      while(goal not in visit):
         a=stack[-1]
         stack.pop(-1)
         x=child(a,visit)
         if(goal in x):
            break
         elif(len(x)==0):
            stack.pop()
         else:
            for i in x:
               stack.append(i)

solve(initial,goal1)
print(visit)
```

QUES2. **Solve the following blocks world problem using Breadth First Search. Compare the results with the question 1.**



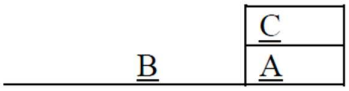


ANS.

```
import copy
initial=[['A'],['B','C'],[]]
goal1=[['A','B','C'],[],[]]
visit=[]
stack=[]
visit.append(initial)
stack.append(initial)
i=0
def child(curr,visit):
    val1=[]
    for j in range(len(curr)):
        if (len(curr[j])>0):
            p = copy.deepcopy(curr)
            val = curr[j][-1]
            p[j].pop(-1)
            k = 0
            while (k < len(curr)):
                p1 = copy.deepcopy(curr)
                p1[j].pop(-1)
                if (k != j):
                    p1[k].append(val)
```

```
                if (p1 not in visit):
                    visit.append(p1)
                    val1.append(p1)
            k += 1

    return val1

def solve(initial,goal):
    if(goal in initial):
        visit.append(goal)
        return(visit)
    else:
        while(goal not in visit):
            for i in stack:
                x=child(i,visit)
                if(goal in x):
                    break
                else:
                    stack.pop(0)
                    for i in x:
                        stack.append(i)

solve(initial,goal1)
print(visit)
```

***QUES3*. 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.**

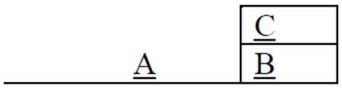


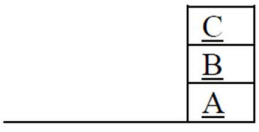
ANS.

```
import copy
initial=[['B'],['A','C'],[]]
goal1=[['A','B','C'],[],[]]
visit=[]
stack=[]
visit.append(initial)
stack.append(initial)
i=0
def child(curr,visit,sum_level,level):
    nik=[]
    if(sum_level<level):
        for j in range(len(curr)):
            if (len(curr[j])>0):
                p = copy.deepcopy(curr)
                val = curr[j][-1]
                p[j].pop(-1)
                k = 0
```

```
            while (k < len(curr)):
               p1 = copy.deepcopy(curr)
               p1[j].pop(-1)
               if (k != j):
                  p1[k].append(val)
                  if (p1 not in visit):
                     visit.append(p1)
                     nik.append(p1)
               k += 1
   return nik


def solve(initial,goal,level):
   sum_level=0
   if(goal in initial):
      visit.append(goal)
      return(visit)
   else:
      while(goal not in visit):
         a=stack[-1]
         stack.pop(-1)
         x=child(a,visit,sum_level,level)
         if(goal in x):
            print('Done')
            break
         elif (len(x) == 0):
            if (len(stack) == 0):
               print("Can't Done")
               break
            stack.pop()
            sum_level-=1
         else:
            for i in x:
               stack.append(i)
            sum_level+=1

solve(initial,goal1,1)
print(visit)
```

QUES4. **Find the depth at which the goal is achieved using Iterative Deepening for the following problem**





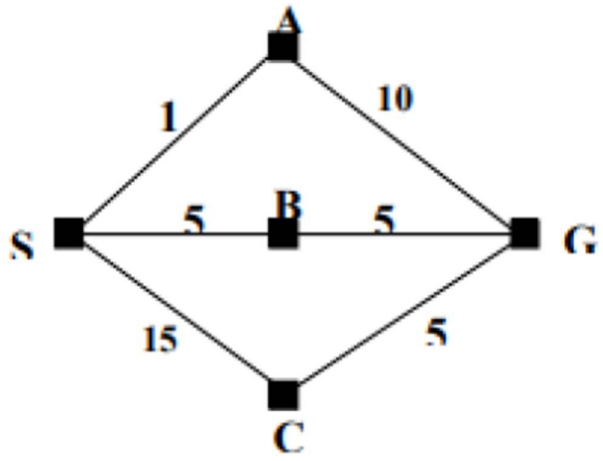
ANS.

```
import copy
initial=[['A'],['B','C'],[]]
```

```
goal1=[['A','B','C'],[],[]]
visit=[]
stack=[]
visit.append(initial)
stack.append(initial)
i=0
def child(curr,visit):
    nik=[]
    for j in range(len(curr)):
        if (len(curr[j])>0):
            p = copy.deepcopy(curr)
            val = curr[j][-1]
            p[j].pop(-1)
            k = 0
            while (k < len(curr)):
                p1 = copy.deepcopy(curr)
                p1[j].pop(-1)
                if (k != j):
                    p1[k].append(val)
                    if (p1 not in visit):
                        visit.append(p1)
                        nik.append(p1)
                k += 1
    return nik


def solve(initial,goal):
    sum_level=0
    if(goal in initial):
        visit.append(goal)
        return(visit)
    else:
        while(goal not in visit):
            a=stack[-1]
            stack.pop(-1)
            x=child(a,visit)
            if(goal in x):
                sum_level+=1
                print('Done')
                print(sum_level)
                break
            elif (len(x) == 0):
                stack.pop()
                sum_level-=1
            else:
                for i in x:
                    stack.append(i)
```

```
            sum_level+=1

solve(initial,goal1)
```

QUES5. **Solve this given problem using Uniform Cost search.**



ANS.

```
path={'S':[[1,'A'],[5,'B'],[15,'C']], 'A':[[10,'G']],'B':[[5,'G']],'C':[[15,'G']]}
stack=[]
def child(curr):
   if(curr in path.keys()):
      x=path[curr]
      return x
def solve(source,goal):
   stack.append([0,source])
   sum=999999
   total=[]
   while(len(stack)>0):
      a=stack[-1]
      stack.pop(-1)
      if(a[-1]==goal):
         total.append(a[0])
         sum=min(sum,a[0])
      else:
         y=child(a[-1])
         y.sort(reverse=True)
         for i in y:
            i[0]+=a[0]
            stack.append(i)
   print(total)
   print(sum)
solve('S','G')
```