Computer Science and Engineering
UG SEM~5

Rutazeet Ritik Rout
1BM18CS151
13/11/20

Course :- Artificial Intelligence
Course code :- 20CS5PCAIP

2)
```
def main ():

starting node = [[0, 0]]
jugs = get_jugs()
goal_amount = get_goal (jugs)
 check_dict = {}
is_depth = True
search (starting_node, jugs, goal_amount, check_dict,
            is_depth)

def get_index (node):
    return pow (7, node [0]) * pow (5, node [1])

def get_jugs ():
    print ("Receiving the volume of the jugs")
     jugs=[]
    temp = int (input ("Enter first jug volume (>1): "))
   while temp < 1:
         temp = int (input ("Enter a valid amount(>1): "))
   jugs.append (temp)

   temp = int (input ("Enter second jug volume(>1): "))
    while temp < 1:
          temp = int (input ("Enter a valid amount (>1): "))
   jugs.append (temp)

   return jugs.
```

Rutazeet

2

1BM18CS151

```
def get_goal(jugs):

print ("Recieve the desired amount of water")
max_amount = max (jugs [0], jugs [1])
S = "Enter the desired amount of water (1 - {0}): ".
                                   format(max_amount)

goal_amount = int (input(S))
while goal_amount < 1 or goal_amount > max_amount:
goal_amount = int (input ("Enter a valid amount (1-{0}): ".
                                   format (max_amount)))

return goal_amount.

def is_goal (path, goal_amount):
print ("Check it the goal is achieved ")
return path [-1][0] == goal_amount or path [-1][1] ==
                          goal_amount.

def been_there (node, check_dict):
 print ("Check if {0} is visited before". format(node))
 return check_dict.get (get_index(node), False)

def next transitions( jugs, path, check_dict):
   print ("Finds next transitions and check for loops")
   result = [ ]
   next_nods = []
   node = []
   a_max = jugs [0]
   b_max = jugs [1]

   a = path [-1] [0]
   b = path [-1][1]
```

```
node. append (a_max)
node. append (b)
if not been_ there (node, check _dict):
        next_node. append (node)
node = [].

node . append (a)
node. append (b_max)
if not been_there (node, check_dict):
    next_node. append (node).
node = []

node. append (min (a_max , a+b))
node. append (b- (node [0] _a))
if not been there (node, check_dict):
    next_node. append (node).
node = []

node. append (min (a+b , b_max)).
node. insert (0, a - (node [0]_ b)).
if not been _ there (node , check_dict):
    next_node. append (node)
node = [].

node. append (0)
node, append (b).
if not been_ there (node, check_ dict):
    next _node .append (node)
node = []
node . append (a)
 node . append (0)
 if not been_there (node, check dict):
     next_node . append (node).

for  i in range (0, len (next_nodes)):
     temp = list (path).
     temp . append (next_node [i])
     result . append (temp)

if len (next _node) == 0:
   Print ("No more unvisitide nods In Backtracking")
else: 
     print ("possible transitions: ")
      for mode in next_node:
         Print (mode).
 return  result

def transition (old, new, jugs):

a= old [0]
b= old [1]
a_prime = new [0]
b-prime = new [1]
a - max = jugs [0]
b - max  =  jugs [1]

if  a> a-prime:
     if   b= b_prime :
     return "Clear {0}- litter jug:\t ". format (a_max)
else:
    return "Pour {0} - litter jug into {1} - litter jug: \t" Rutarget
                        .format (a_max, b_max)
```

```
else:
        if b> b_prime:
            if a= a-prime.
                return "Clear {0}-litter jug \t". format(b-max)
        else
            return "pour {0}-litter jug into {1}-litter jug\t"
                        . format (b-max, a-max)

~~es~~     else.
            if a==a-prime:
                return "Fill {0}-litter jug: \t". format (b-max)
            else
                return "Fill {0}- litter jug: \t". format (a-max)

def print_path (path,jugs):

    print ("Starting from: \t" path[0])
     for i in range (0, len (path) -1):
        print (i+1, ":", transition(path[i], path [i+1], jugs),
                        path [i+1])

def search (starting_node, jugs, goal_amount, check_dict,
            * is depth):

if is_depth:
        print ("Implement DFS")

goal []
accomplished = False
q= collection. deque()
  q.append left (starting_node).
```

6

1BM18CS151

```
while len(q) != 0:
        path = q.popleft()
        check_dict [get_index (path [-1])] = True
        if lent (path)>=2:
            print (transition(path [-2], path[-1], jugs), path[-1])
        if  is_goal (pathy goal_amount):
          accomplished= True.
          goal = path
          break

   next_moves= next_transition(jugs, path, check_dict)
        for  i in next_moves:
          if   is_depth:
                q. append left(i).
           else:
                q. append (i).

    if accomplised:
        print(" the goal is achied" \n printing the sequence \n").
         print_ path (goal, jugs)
   else:
        print ( ~~pdat~~ "problem cant be solved").

  if _name_ == '_main_':
    main ().
```