Lab work – 1(Simple Intelligence Agent)

For this lab work, you must work with the given code using **matplotlib, random, heapq, numpy** library in Python. Create an instance of the environment**, not having less than 5 columns and 5 rows,** using the **WorldMap()** class. You also need to initiate a single vacuum cleaner agent and run your agent till it completes 25 random moves.

Try to understand the given Python code for the environment class **WorldMap()** and its different methods. Also, see how **matplotlib.pyplot** is used to create a visualization of the **WorldMap()** instance you create. Also, ensure the agent can not alter the obstacles (obs) and only the **dirt_blocks.**

Return the environment instance after the agent completes its 25 moves. Also, show how you get different results for different iterations.

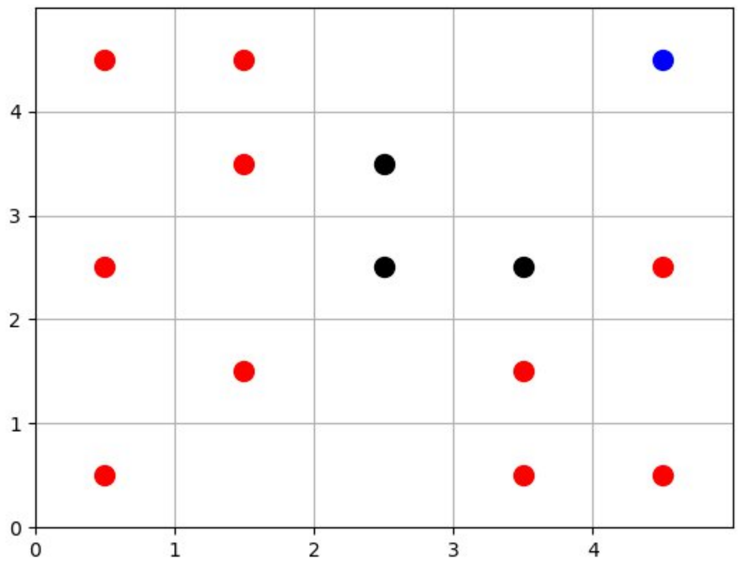
Program:

```
import matplotlib.pyplot as plt
import random
import heapq
import numpy as np



class WorldMap:
  def __init__(self, rows, cols, num_dirt_blocks, num_obs):
    self.rows = rows
    self.cols = cols
    self.num_dirt_blocks = num_dirt_blocks
    self.num_obs = num_obs
    self.world_map = [['clean' for _ in range(cols)] for _ in range(rows)]
    self.agent_positions = {}  # Dictionary to store agent position
    # Place dirt blocks randomly on the map
    for _ in range(num_dirt_blocks):
      row = random.randint(0, rows - 1)
      col = random.randint(0, cols - 1)
      while self.world_map[row][col] == 'dirt' or self.world_map[row][col] == 'agent':
        row = random.randint(0, rows - 1)
        col = random.randint(0, cols - 1)
      self.world_map[row][col] = 'dirt'

    # Place obstacles randomly on the map (excluding corners)
    for _ in range(num_obs):
      row = random.randint(1, rows - 2)  # Avoid corners
      col = random.randint(1, cols - 2)  # Avoid corners
  while self.world_map[row][col] == 'dirt' or self.world_map[row][col] == 'agent' or
self.world_map[row][col] == 'obs':
        row = random.randint(1, rows - 2)  # Avoid corners
        col = random.randint(1, cols - 2)  # Avoid corners
```

```python
        self.world_map[row][col] = 'obs'

    def add_agent(self, agent_id):
        while True:
            row = random.randint(0, self.rows - 1)
            col = random.randint(0, self.cols - 1)
            if self.world_map[row][col] == 'clean':
                self.world_map[row][col] = 'agent'
                self.agent_positions[agent_id] = (col,row)
                break


    def getAgentPos(self, agent_id):
        if agent_id in self.agent_positions:
            return self.agent_positions[agent_id]
        else:
            return None  # Agent not found

    def move_agent(self, agent_id, new_position):
        if agent_id in self.agent_positions:
            current_position = self.agent_positions[agent_id]
            if self.world_map[current_position[1]][current_position[0]] == 'agent':
                self.world_map[current_position[1]][current_position[0]] = 'clean'  # Clear the current cell
            self.world_map[new_position[1]][new_position[0]] = 'agent'  # Place the agent in the new cel
            self.agent_positions[agent_id] = new_position  # Update the agent's position


    def display_map(self):
        fig,ax = plt.subplots() # Clear the current plot
        for row in range(self.rows):
            for col in range(self.cols):
                if self.world_map[row][col] == 'dirt':
                    ax.plot(col + 0.5,  row + 0.5, 'ro', markersize=10)  # Display dirt as red dots
                elif self.world_map[row][col] == 'agent':
                    ax.plot(col + 0.5, row + 0.5, 'bo', markersize=10)  # Display agents as blue dots
                elif self.world_map[row][col] == 'obs':
                    ax.plot(col + 0.5, row + 0.5, 'ko', markersize=10)  # Display obstacles as black dots
        ax.set_xlim(0, self.cols)
        ax.set_ylim(0, self.rows)
        ax.set_xticks(range(self.cols))
        ax.set_yticks(range(self.rows))

  ax.grid()

    def is_valid_position(self, row, col):
        return 0 <= row < self.rows and 0 <= col < self.cols

world = WorldMap(5,5,10,3)
world.add_agent('A')
```

```
print(world.world_map)
world.display_map()
```

[['dirt', 'clean', 'clean', 'dirt', 'dirt'], ['clean', 'dirt', 'clean', 'dirt', 'clean'], ['dirt', 'clean', 'obs', 'obs', 'dirt'], ['clean', 'dirt', 'obs', 'clean', 'clean'], ['dirt', 'dirt', 'clean', 'clean', 'agent']]

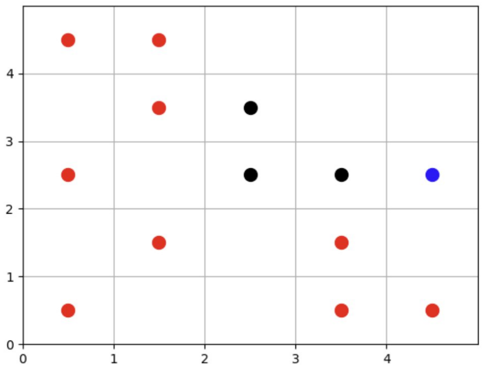



```
path=[world.getAgentPos('A')]
count = 0
while count < 25:
 a = random.randint(0,4)
 if a == 0:
  pos = world.getAgentPos('A')
  right = (pos[0],pos[1]+1)
  if world.is_valid_position(right[1],right[0]) and world.world_map[right[1]][right[0]] != 'obs': #and right not in path:
    world.move_agent('A',right)
    path.append(right)
    count += 1
    world.display_map()

 elif a == 1:
  pos = world.getAgentPos('A')
  left = (pos[0],pos[1]-1)
  if world.is_valid_position(left[1],left[0]) and world.world_map[left[1]][left[0]] != 'obs':#and left not in path:
    world.move_agent('A',left)
    path.append(left)
    count += 1
    world.display_map()

 elif a == 2:
  pos = world.getAgentPos('A')
  up = (pos[0]+1,pos[1])
  if world.is_valid_position(up[1],up[0]) and world.world_map[up[1]][up[0]] != 'obs':#and up not in
```

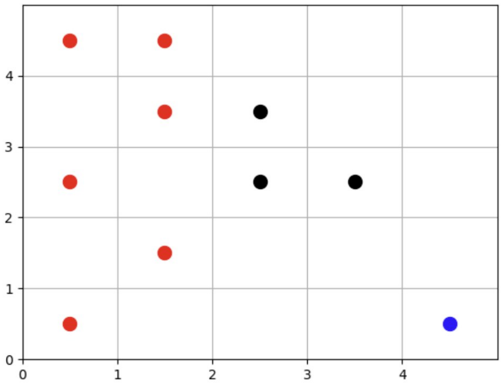
```
path:
        world.move_agent('A',up)
        path.append(up)
        count += 1
        world.display_map()

  elif a == 3:
    pos = world.getAgentPos('A')
    down = (pos[0]-1,pos[1])
    if world.is_valid_position(down[1],down[0]) and world.world_map[down[1]][down[0]] !=
'obs':#and down not in path:
        world.move_agent('A',down)
        path.append(down)
        count += 1
        world.display_map()

print(world.world_map)

print(path)
```

OUTPUT : Path

[['dirt', 'clean', 'clean', 'clean', 'agent'], ['clean', 'dirt', 'clean', 'clean', 'clean'], ['dirt', 'clean', 'obs', 'obs', 'clean'], ['clean', 'dirt', 'obs', 'clean', 'clean'], ['dirt', 'dirt', 'clean', 'clean', 'clean']]
[(4, 4), (3, 4), (3, 3), (4, 3), (4, 4), (4, 3), (4, 2), (4, 1), (4, 0), (4, 1), (3, 1), (4, 1), (4, 2), (4, 1), (4, 0), (3, 0), (3, 1), (4, 1), (3, 1), (3, 0), (4, 0)]



final state:



Lab work – 2 (DFS and BFS implementation)

For this lab work, you are required to implement the Breadth First Search (BFS) and Depth First Search (DFS) algorithms to solve the 8-puzzle game. To see how the 8-puzzle game works you can go to this link - http://www.puzzlopia.com/puzzles/puzzle 8/play. You can refer to the lecture slides for the pseudocode of the BFS and DFS algorithms.

Note that the implementations are similar but the only difference is in the implementation of the **frontier (BFS – Queue (FIFO), DFS – Stack (LIFO)).**

The starter code has already been provided to you. It consists of the following: Queue class – implementation of the queue data structure

Stack class – implementation of the stack data structure

State class – this class models the 8 puzzle game states, an instance of this class can be thought of as a state in the state space or a tree node in the search space.

Comments have been provided to you in the code, so be sure to read them and get familiar with what the code is doing. You will specifically find the get_children() function helpful. Your task is to fill in the code for the bfs() and dfs() functions respectively. Both functions should return the set of moves required to solve the puzzle as well as the total number of moves required given some initial starting state.Running times will vary depending on your implementation. Improper implementation may result in longer running times.

Also, you will notice the different solutions returned by DFS and BFS. Analyze the time complexity of the solutions and conclude it using the time library.

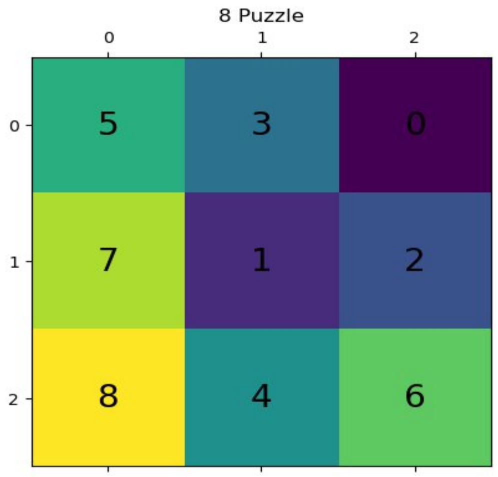
CODE:

```
import matplotlib.pyplot as plt
import numpy as np

class Queue():
  def__init__(self, initial):
    self.items = [initial]
  def isEmpty(self):
    return self.items == []
  def enqueue(self, item):
    self.items.insert(0,item)
  def dequeue(self):
    return self.items.pop()
  def size(self):
    return len(self.items)

class Stack():
  def__init__(self, initial):
    self.items = [initial]
  def isEmpty(self):
```

```
        return self.items == []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        return self.items.pop()
    def size(self):
        return len(self.items)

class State():
    right = {0, 1, 3, 4, 6, 7}
    left = {1, 2, 4, 5, 7, 8}
    up = {3, 4, 5, 6, 7, 8}
    down = {0, 1, 2, 3, 4, 5}

    def__init__(self, board_config, parent, move):
        self.board_config = board_config  # board configuration of the current state in a string
        self.board_config_list = list(map(int,board_config.split(',')))  # board configuration of the current
state in a list
        #print(self.board_config_list)
        self.i = self.board_config_list.index(0)  # index of empty space in board (index of 0 in this case)
        self.parent = parent  # parent state (node) of the present state
        self.move = move  # the move (Up,Down,Left,Right) made in parent state that results in the
present state
    def get_children(self):
        """returns the list of all possible states reachable from the current state,
        each child in the list is a State object"""
        children = []
        if self.i in State.up:
            new_board_config = self.board_config_list[:]
            new_board_config[self.i], new_board_config[self.i-3] = new_board_config[self.i-3],
new_board_config[self.i]
            children.append(State(','.join(map(str,new_board_config)), self.board_config,'Up'))

        if self.i in State.down:
            new_board_config = self.board_config_list[:]
            new_board_config[self.i], new_board_config[self.i+3] = new_board_config[self.i+3],
new_board_config[self.i]
            children.append(State(','.join(map(str,new_board_config)), self.board_config,'Down'))

        if self.i in State.left:
            new_board_config = self.board_config_list[:]
            new_board_config[self.i], new_board_config[self.i-1] = new_board_config[self.i-1],
new_board_config[self.i]
            children.append(State(','.join(map(str,new_board_config)), self.board_config,'Left'))

        if self.i in State.right:
            new_board_config = self.board_config_list[:]
            new_board_config[self.i], new_board_config[self.i+1] = new_board_config[self.i+1],
```

```
new_board_config[self.i]
            children.append(State(','.join(map(str,new_board_config)), self.board_config,'Right'))
        return children
    def plot_8_puzzle(self):
        board = np.array([int(x) for x in self.board_config.split(',')]).reshape(3, 3)

        fig, ax = plt.subplots()
        ax.matshow(board)

        for i in range(3):
            for j in range(3):
                ax.text(j, i, str(board[i, j]), va='center', ha='center', fontsize=20, color='black')

        plt.title('8 Puzzle')
        plt.show()
    def __str__(self):
        return self.board_config

def dfs(initial,goal):
    frontier = Stack(initial)
    frontier_set = set()
    count = 0
    while not frontier.isEmpty():
        count +=1
        state = frontier.pop()
        frontier_set.add(state.board_config)
        if state.board_config == goal:
            return "success"
        else:
            for child in state.get_children():
             if child.board_config not in frontier_set:
                frontier.push(child)
    return 'failure'

def bfs(initaial, goal):
    frontier = Queue(initaial)
    frontier_set = set()
    count = 0
    while not frontier.isEmpty():
        count +=1
        state = frontier.dequeue()
        frontier_set.add(state.board_config)
        if state.board_config == goal:
            return "success"
        else:
            for child in state.get_children():
             if child.board_config not in frontier_set:
                frontier.enqueue(child)
    return 'failure'
```

```
start = '5,3,0,7,1,2,8,4,6'
goal = '0,1,2,3,4,5,6,7,8'
initial_state = State(start, None, None)
initial_state.plot_8_puzzle()
```




```
import time
s=time.time()
print(dfs(initial_state, goal))
f=time.time()
e=f-s
print(e)
```

Output:

success
4.385337352752685 5

```
import time
s=time.time()
print(bfs(initial_state, goal))
f=time.time()
e=f-s
print(e)
```

output:

success
12.166506290435791

LAB :3 (A* Search)

This lab work is a continuation of lab work 1. You are required to implement the A* search algorithm to solve the 8 puzzle game. To see how the 8 puzzle game works you can go to this link - http://www.puzzlopia.com/puzzles/puzzle-8/play. You can refer to the lecture slides for the pseudocode of the algorithms. Note that the implementation of the frontier for A* is a priority queue The starter code has already been provided to you. It consists of the following: You are required to implement the priority queue for A* search.

State class – this class models the 8 puzzle game states, an instance of this class can be thought of as a state in the state space or a tree node in the search space. Comments have been provided to you in the code, so be sure to read them and get familiar with what the code is doing. You will specifically find the get_children() function helpful.

Your task is to fill in code for the ast() functions respectively. All functions should return the set of moves required to solve the puzzle as well as the total number of moves required given some initial starting state. Running times will vary depending on your implementation. Improper implementation may result in longer running times. However, if your code is written fairly well, the program should find the solution for any random starting state in less than 10 seconds.

Also, you will notice the different solutions returned A*. Analyze the solutions and conclude it. Also, analyze their running times.

For A* search to work properly a good heuristic function should be used. As discussed in the lecture, you can use the Manhattan distance as the heuristic which determines how close a current state is to the goal.

To calculate the Manhattan distance between 2 points, use the following formula:

if a = (x1, y1) and b = (x2, y2),

then the manhattan distance between a and b will be given by

d = |x1 − x2| + |y1 − y2|

You can think of the different positions in the board as different points in a 2d space. So each board position will have the following coordinates (x,y)

0,0	**0,1**	**0,2**
1,0	**1,1**	**1,2**
2,0	**2,1**	**2,2**

So the Manhattan distance between the board positions 2,2 and 0,1 will be:

x1, y1 = 2,2; x2, y2 = 0,1;

d = |2 − 0| + |2 − 1| = 3

The overall cost function for A* search will be the following:

f(n) = g(n) + h(n)

g(n) − cost for reaching a node n from the start node

h(n) − heuristic function (in this case,the sum of manhattan distances)

You can use g(n) as simply the number of steps required to reach a node from the start.For calculating this the attribute depth in the state class will be helpful.

Code:

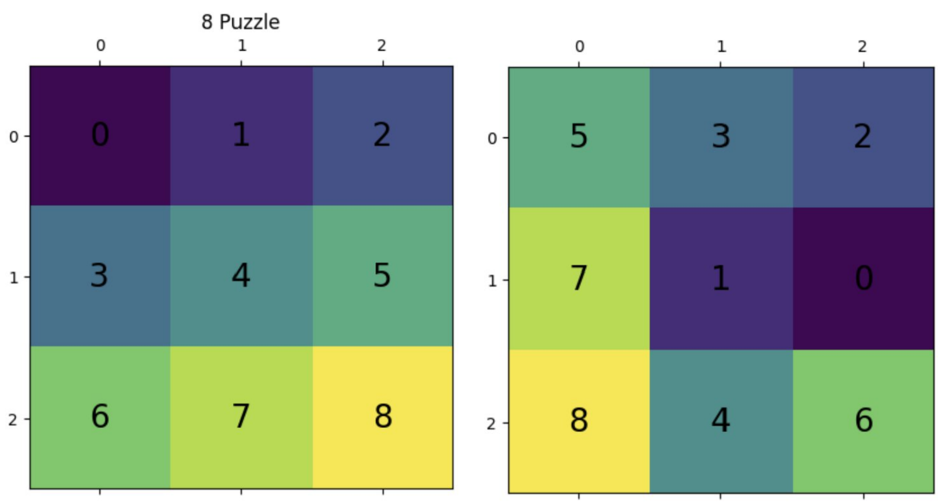
```
import matplotlib.pyplot as plt
import numpy as np

class PriorityQueue():
    def __init__(self,item,cost):
        self.items = {cost:[item]}
        self.costs = {cost}

    def isEmpty(self):
        return self.items == {}

    def dequeue(self):
        least_cost = sorted(self.costs)[0]
        item = self.items[least_cost].pop(0)

        if len(self.items[least_cost]) == 0:
            self.costs.remove(least_cost)
            del self.items[least_cost]

        return item
    def enqueue(self,item,cost):
        if cost in self.costs:
            self.items[cost].append(item)

        else:
            self.items[cost] = [item]
            self.costs.add(cost)

    def update_cost(self,item,old_cost,new_cost):
        #print(old_cost)
        for i in self.items[old_cost]:
            if i.board_config == item.board_config:
                self.items[old_cost].remove(i)
                break
        if len(self.items[old_cost]) == 0:
            self.costs.remove(old_cost)
            del self.items[old_cost]
        if new_cost in self.costs:
            self.items[new_cost].append(item)
```

```
        else:
            self.items[new_cost] = [item]
            self.costs.add(new_cost)

class State():
    right=(0,1,3,4,6,7)
    left=(1,2,4,5,7,8)
    up=(3,4,5,6,7,8,)
    down=(0,1,2,3,4,5)

    def __init__(self,board_config,parent,move,depth):
        self.board_config = board_config
        self.board_config_list = list(map(int,board_config.split(',')))
        self.i = self.board_config_list.index(0)
        self.parents = parent
        self.move = move
        self.depth = depth
       # print('Constructor called')

    def get_children(self):
        children = []
        if self.i in State.right:
            new_config = self.board_config_list[:]
            new_config[self.i],new_config[self.i+1]=new_config[self.i+1], new_config[self.i]
            children.append(State(','.join(map(str,new_config)),self.board_config,'Right',self.depth+1))
        if self.i in State.left:
            new_config = self.board_config_list[:]
            new_config[self.i],new_config[self.i-1]=new_config[self.i-1], new_config[self.i]
            children.append(State(','.join(map(str,new_config)),self.board_config,'Left',self.depth+1))
        if self.i in State.up:
            new_config = self.board_config_list[:]
            new_config[self.i],new_config[self.i-3]=new_config[self.i-3], new_config[self.i]
            children.append(State(','.join(map(str,new_config)),self.board_config,'Up',self.depth+1))
        if self.i in State.down:
            new_config = self.board_config_list[:]
            new_config[self.i],new_config[self.i+3]=new_config[self.i+3], new_config[self.i]
            children.append(State(','.join(map(str,new_config)),self.board_config,'Down',self.depth+1))
          # print(children)
        return children
    def plot_8_puzzle(self):
        board = np.array([int(x) for x in self.board_config.split(',')]).reshape(3, 3)

        fig, ax = plt.subplots()
        ax.matshow(board)

        for i in range(3):
            for j in range(3):
                ax.text(j, i, str(board[i, j]), va='center', ha='center', fontsize=20, color='black')
```

```
        plt.title('8 Puzzle')
        plt.show()
    def __str__(self):
        return self.board_config

def manhattan_dist(x,y):
  # print('j')
  return abs(x[0]-y[0])+abs(x[1]-y[1])

indexes = {0:(0,0), 1:(0,1),2:(0,2),3:(1,0),4:(1,1),5:(1,2), 6:(2,0), 7:(2,1),8:(2,2)}

def h(s):
  s = s.split(',')
  dist=0
  # print('k')
  for each in s :
    i = s.index(each)
    x = indexes[i]
   #  print('l')
    y = indexes[int(each)]
    dist = dist + manhattan_dist(x,y)
  return dist

h("0,5,3,8,2,1,7,4,6")

16

def ast(inp,goal):
    frontier = PriorityQueue(inp,h(inp.board_config))
    frontier_dict = {inp.board_config:h(inp.board_config)}
    graph = {}

    explored = set()

    while not frontier.isEmpty():
        state = frontier.dequeue()
        explored.add(state.board_config)
        del frontier_dict[state.board_config]
        graph[state.board_config] = state
        if state.board_config == goal:
          path = []
          current_state = state
          print(h(goal))
          while not current_state.parents == None:
              current_state.plot_8_puzzle()
              path.append(current_state.move)
              current_state =graph[current_state.parents]
           #   print('n')
          return print (path, len(path))
```

```
        else:
          # print('st')
           a = state.get_children()
           for children in a:
           #     print(children.board_config)
                #print (frontier)
                cost= h(children.board_config)+ children.depth
                if children.board_config not in frontier_dict and children.board_config not in explored:
                   frontier.enqueue(children,cost)
                   #print('o')
                   frontier_dict[children.board_config]= cost
                   #print('v')
                   if children.board_config in frontier_dict:
                   # new_cost = cost
                   #old_cost = frontier_dict[children.board_config]
                    if cost < frontier_dict[children.board_config]:
                       frontier.update_cost(children,frontier_dict[children.board_config],cost)
                    #    print('p')
    return' failure'

inp= State('5,3,0,7,1,2,8,4,6',None,None,0)
import time
s=time.time()
print(ast(inp,'0,1,2,3,4,5,6,7,8'))
e=time.time()
t=e-s
print(t)
```

OUTPUT :




```
['Up', 'Up', 'Left', 'Left', 'Down', 'Right', 'Right', 'Up', 'Left', 'Left', 'Down', 'Right', 'Up', 'Right', 'Down', 'Left',
'Down', 'Right', 'Up', 'Up', 'Left', 'Left', 'Down', 'Down'] 24
None
9.857656717300415
```

LAB :4 (Production_rule_based_system)

For this lab work, you need to write a program that can use the production rules of the water jug problem and find the solution for any input of the full capacity of jugs and the targeted volume.

Capacity of first jug =X

Capacity of second jug=Y

Targeted Volume = T

The program should return not possible if the given targeted volume can not be achieved using the chosen values of X and Y. Use the following information to make the general solution.

A pair of Jugs can only find targeted volume if

G modulo T = 0 [: where G= Highest Common Factor of X and Y]

CODE:

```
print("Solution for water jug problem")
x_capacity = input("Enter Jug 1 capacity:")
y_capacity = input("Enter Jug 2 capacity:")
end = input("Enter target volume:")

def Production_Rule_Implementation(start, end, x_capacity, y_capacity):
 path = []
 front = []
 front.append(start)
 visited = []
#visited.append(start)
  while front:
       current = front.pop()
       x = current[0]
       y = current[1]
       path.append(current)
       if x == end or y == end:
         print("Found!")
         return path
                  # rule 1 filling x
if current[0] < x_capacity and ([x_capacity, current[1]] not in visited):
       front.append([x_capacity, current[1]])
       visited.append([x_capacity, current[1]])
                  # rule 2 filling y
if current[1]< y_capacity and ([current[0],y_capacity] not in visited):
        front.append([current[0], y_capacity])
        visited.append([current[0], y_capacity])
                  # rule 3 emptying x
if current[0] > x_capacity and ([0, current[1]] not in visited):
         front.append([0, current[1]])
```

```
        visited.append([0, current[1]])
      # rule 4 emptying y
      if current[1] > y_capacity and ([x_capacity, 0] not in visited):
        front.append([x_capacity, 0])
        visited.append([x_capacity, 0])
# rule 5 pouring y into x, both in case y is empty or y is left with some#(x, y) -> (min(x + y,
x_capacity), max(0, x + y - x_capacity)) if y > 0
if current[1] > 0 and ([min(x + y, x_capacity), max(0, x + y - x_capacity)] not in visited):
       front.append([min(x + y, x_capacity), max(0, x + y - x_capacity)])
       visited.append([min(x + y, x_capacity), max(0, x + y - x_capacity)])
# rule 6 pouring x into y, both in case x is empty or x is left with some
# (x, y) -> (max(0, x + y - y_capacity), min(x + y, y_capacity)) if x > 0
if current[0] > 0  and ([max(0, x + y - y_capacity), min(x + y, y_capacity)] not in visited):
       front.append([max(0, x + y - y_capacity), min(x + y, y_capacity)])
       visited.append([max(0, x + y - y_capacity), min(x + y, y_capacity)])
      return "Not found"
def gcd(a, b):
      if a == 0:
        return b
      return gcd(b%a, a)
start = [0, 0]
if int(end) % gcd(int(x_capacity),int(y_capacity)) == 0:
      print (Production_Rule_Implementation(start, int(end),int( x_capacity),int( y_capacity)))
else:
      print ("No solution possible for this combination.")
```

OUTPUT:

Solution for water jug problem
Enter Jug 1 capacity:5
Enter Jug 2 capacity:7
Enter target volume:3
Found!
[[0, 0], [0, 7], [5, 2], [5, 7], [5, 0], [0, 5], [5, 5], [3, 7]]

Lab :5 (Naïve Bayes Classifier)

For this lab work, you will be working with the **Naïve Bayes Classifier** for spam detection. Given below are 2 tables containing training data and test data respectively.

Training examples consist of text (sms) labeled as spam or not spam. Use the examples to build the vocabulary for the classifier. Then using the ***bag of words*** approach, transform the texts into feature vectors.

Then following the algorithm for the Naïve Bayes Classifier, classify the 2 texts in the test data table as spam or not spam.

The algorithm for the Naïve Bayes Classifier is as follows (also refer to the lecture slides):

Learning: Based on the frequency counts in the dataset:

1. Estimate all $p(y)$, $\forall y \in Y$
2. Estimate all $p(a_j|y)$, $\forall y \in Y$, $\forall a_j$

Classification: For a new example, use:

$$y_{new} = argmax_{y \in Y}\ p(y) \prod_{j=1}^{d} p(a_j|y)$$

You are required to submit your work in form of a simple report showing all the calculations that you have done and your final result. As the process will require lots of redundant work, you can write code to speed up the process.

Training Data:

Text	Label
Congrats, You have won!! reply to our sms for a free nokia mobile + free camcorder.	spam
Congrats! 1 year special cinema pass for 2 is yours. reply to this sms to claim your prize.	spam
I am pleased to tell you that you are awarded with a 1500 Bonus Prize, reply to this sms to claim your prize.	spam
Dont worry. I guess he is busy.	not spam
Going for dinner. msg you later.	not spam
Ok, I will call you up when I get some cash.	not spam

Test Data:

Text	Label
I am busy. I will msg you later.	?
Congrats! You are awarded a free mobile.	?

CODE:

In [ ]:

```python
data ='''Congrats, You have won!! reply to our sms for a free nokia mobile + free camcorder. ---spam
Congrats! 1 year special cinema pass for 2 is yours. reply to this sms to claim your prize. ---spam
I am pleased to tell you that you are awarded with a 1500 Bonus Prize, reply to this sms to claim your prize.---spam
Dont worry. I guess he is busy. ---not spam
Going for dinner. msg you later.  ---not spam
Ok, I will call you up when I get some cash.---not spam'''
print(data)
data = data.split('\n')
data
```

```
Congrats, You have won!! reply to our sms for a free nokia mobile + free camcorder. ---spam
Congrats! 1 year special cinema pass for 2 is yours. reply to this sms to claim your prize. ---spam
I am pleased to tell you that you are awarded with a 1500 Bonus Prize, reply to this sms to claim your prize.---spam
Dont worry. I guess he is busy. ---not spam
Going for dinner. msg you later.  ---not spam
Ok, I will call you up when I get some cash.---not spam
```

Out[124]:

```
['Congrats, You have won!! reply to our sms for a free nokia mobile + free camcorder. ---spam',
 'Congrats! 1 year special cinema pass for 2 is yours. reply to this sms to claim your prize. ---spam',
 'I am pleased to tell you that you are awarded with a 1500 Bonus Prize, reply to this sms to claim your prize.---spam',
 'Dont worry. I guess he is busy. ---not spam',
 'Going for dinner. msg you later.  ---not spam',
 'Ok, I will call you up when I get some cash.---not spam']
```

In [ ]:

```python
len(data)
```

Out[114]: 1

```python
# data = [data.split("\n") ]
data = [d.split("---") for d in data]
# del(data[-1])
print(data)
```

```
[['Congrats, You have won!! reply to our sms for a free nokia mobile + free camcorder. ', 'spam'], ['Congrats! 1 year specia
l cinema pass for 2 is yours. reply to this sms to claim your prize. ', 'spam'], ['I am pleased to tell you that you are awa
rded with a 1500 Bonus Prize, reply to this sms to claim your prize.', 'spam'], ['Dont worry. I guess he is busy. ', 'not sp
am'], ['Going for dinner. msg you later.  ', 'not spam'], ['Ok, I will call you up when I get some cash.', 'not spam']]
```

In [ ]:

```
data = pd.DataFrame(data)
print (data)
data.columns = ["text","label"]
data
```

```
                                              0         1
0  Congrats, You have won!! reply to our sms for ...      spam
1  Congrats! 1 year special cinema pass for 2 is ...      spam
2  I am pleased to tell you that you are awarded ...      spam
3                      Dont worry. I guess he is busy.  not spam
4                       Going for dinner. msg you later.  not spam
5           Ok, I will call you up when I get some cash.  not spam
```

Out[120]:

	text	label
0	Congrats, You have won!! reply to our sms for ...	spam
1	Congrats! 1 year special cinema pass for 2 is ...	spam
2	I am pleased to tell you that you are awarded ...	spam
3	Dont worry. I guess he is busy.	not spam
4	Going for dinner. msg you later.	not spam
5	Ok, I will call you up when I get some cash.	not spam

```
vocab = []
text_vector = []
for each in data["text"]:
    vocab.extend(each.lower().replace(".","").replace(",","").replace("+","").replace("!","").split())
    text_vector.append(each.lower().replace(".","").replace(",","").replace("+","").replace("!","").split())

vocab = list(set(vocab))
print(vocab)
```

['camcorder', 'a', 'prize', 'pleased', 'busy', 'going', 'up', 'have', 'ok', '1500', 'this', 'will', 'to', 'awarded', 'sms', 'he', 'guess', 'reply', 'cinema', 'am', 'cash', 'bonus', '1', 'dont', 'i', 'that', 'congrats', 'your', 'worry', 'get', 'some', 'free', 'msg', 'is', 'for', 'pass', 'call', 'special', 'won', 'when', 'yours', '2', 'our', 'with', 'later', 'tell', 'claim', 'nokia', 'are', 'mobile', 'you', 'dinner', 'year']

```
print(text_vector)
```

[['congrats', 'you', 'have', 'won', 'reply', 'to', 'our', 'sms', 'for', 'a', 'free', 'nokia', 'mobile', 'free', 'camcorder'], ['congrats', '1', 'year', 'special', 'cinema', 'pass', 'for', '2', 'is', 'yours', 'reply', 'to', 'this', 'sms', 'to', 'claim', 'your', 'prize'], ['i', 'am', 'pleased', 'to', 'tell', 'you', 'that', 'you', 'are', 'awarded', 'with', 'a', '1500', 'bonus', 'prize', 'reply', 'to', 'this', 'sms', 'to', 'claim', 'your', 'prize'], ['dont', 'worry', 'i', 'guess', 'he', 'is', 'busy'], ['going', 'for', 'dinner', 'msg', 'you', 'later'], ['ok', 'i', 'will', 'call', 'you', 'up', 'when', 'i', 'get', 'some', 'cash']]

In [ ]:

```
text_vector_num = []
for text in text_vector:
    vector = []
    for word in vocab:
        vector.append(text.count(word))
    text_vector_num.append(vector)
```

In [ ]:

```
for each in text_vector_num:
    print(each)
```

```
[1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 0, 0, 1, 0, 0, 0, 1, 0, 0,
0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 2, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1,
1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1]
[0, 1, 2, 1, 0, 0, 0, 0, 0, 1, 1, 0, 3, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 1, 1, 0, 1, 0, 2, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 2, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
```

In [ ]:
```python
print(len(text_vector_num[0]))
```

```
53
```

In [ ]:
```python
print(len(vocab))
```

```
53
```

In [ ]:
```python
spam = np.array(text_vector_num[:3])
ham = np.array(text_vector_num[3:])
print(spam)
print(ham)
```

```
[[1 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 2 0 0 1 0
  0 0 1 0 0 0 1 0 0 0 0 1 0 1 1 0 0]
 [0 0 1 0 0 0 0 0 0 0 1 0 2 0 1 0 0 1 1 0 0 0 1 0 0 0 1 1 0 0 0 0 0 1 1 1
  0 1 0 0 1 1 0 0 0 0 1 0 0 0 0 0 1]
 [0 1 2 1 0 0 0 0 0 1 1 0 3 1 1 0 0 1 0 1 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 1 0 1 1 0 1 0 2 0 0]]
[[0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
  0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0]
 [0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 2 0 0 0 0 1 1 0 0 0 0 0
  1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0]]
```

In [ ]:
```python
# add one in all the probabilities to omit zeros and add len(vocab to norma
probabilities_spam = (np.sum(spam,axis=0)+1)/(np.sum(spam)+len(vocab))
probabilities_spam
```

Out[95]:
```
array([0.01834862, 0.02752294, 0.03669725, 0.01834862, 0.00917431,
       0.00917431, 0.00917431, 0.01834862, 0.00917431, 0.01834862,
       0.02752294, 0.00917431, 0.06422018, 0.01834862, 0.03669725,
       0.00917431, 0.00917431, 0.03669725, 0.01834862, 0.01834862,
       0.00917431, 0.01834862, 0.01834862, 0.00917431, 0.01834862,
       0.01834862, 0.02752294, 0.02752294, 0.00917431, 0.00917431,
       0.00917431, 0.02752294, 0.00917431, 0.01834862, 0.02752294,
       0.01834862, 0.00917431, 0.01834862, 0.01834862, 0.00917431,
       0.01834862, 0.01834862, 0.01834862, 0.01834862, 0.00917431,
       0.01834862, 0.02752294, 0.01834862, 0.01834862, 0.01834862,
       0.03669725, 0.00917431, 0.01834862])
```

In [ ]:
```python
new = ["I am busy. I will msg you later.","Congrats! You are awarded a free mobile."]
new_words= []
for each in new:
    processed = each.lower().replace(".","").replace(",","").replace("+","").replace("!","").split()
    new_words.append(processed)
new_words
```

Out[101]:
```
[['i', 'am', 'busy', 'i', 'will', 'msg', 'you', 'later'],
 ['congrats', 'you', 'are', 'awarded', 'a', 'free', 'mobile']]
```

```
In [ ]: def predict(words,probability_spam,probability_ham,probabilities_spam,probabilities_ham,vocab):
            spam = probability_spam
            ham = probability_ham
            for word in words:
                spam *= probabilities_spam[vocab.index(word)]
                ham *= probabilities_ham[vocab.index(word)]
            print(spam,ham)
            if spam > ham:
                return " ".join(words),"spam"
            else:
                return " ".join(words),"ham"
```

```
In [ ]: print(predict(new_words[0],probability_spam,probability_ham,probabilities_spam,probabilities_ham,vocab))
        print(predict(new_words[1],probability_spam,probability_ham,probabilities_spam,probabilities_ham,vocab))
```

OUTPUT:

8.029860474764295e-16 3.107459112648254e-13
('i am busy i will msg you later', 'ham')
2.3631879377231325e-12 9.346654362262329e-14
('congrats you are awarded a free mobile', 'spam')

LAB :6 (Neural Network)

For this lab work, Student need to make a 2 layered artificial neural network model to replace a full adder circuit.

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sigmoid should be used as activation function.

$$S(x) = \frac{1}{1+e^{-x}}$$

Use backpropagation as the learning method for the model.

Derivative of Sigmoid function

$$y = \frac{1}{1+e^{-x}}$$

$$\frac{dy}{dx} = -\frac{1}{(1+e^{-x})^2}(-e^{-x}) = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{1}{1+e^{-x}}\left(1-\frac{1}{1+e^{-x}}\right) = y(1-y)$$

CODE:

```
import numpy as np
X = np.array([[0,0,0],[0,0,1],[0,1,0],[0,1,1],[1,0,0],[1,0,1],[1,1,0],[1,1,1] ])
y = np.array([[0,1,1,0,1,0,0,1]]).T
z = np.array([[0,0,0,1,0,1,1,1]]).T
syn0 = 2*np.random.random((3,4)) - 1
print(syn0)
syn1 = 2*np.random.random((4,1)) - 1
print(syn1)
syn3 = 2*np.random.random((3,4)) - 1
print(syn0)
syn4 = 2*np.random.random((4,1)) - 1
print(syn1)
#Neural network with Z variable as target
```

```
for j in range(60000):
  #feed forward with sigmoid activation for first layer
  l1 = 1/(1+np.exp(-(np.dot(X,syn0))))
  #feed forward with sigmoid activation for first layer
  l2 = 1/(1+np.exp(-(np.dot(l1,syn1))))
  l2_delta = (y - l2)*(l2*(1-l2))
  l1_delta = l2_delta.dot(syn1.T) * (l1 * (1-l1))
  syn1 += l1.T.dot(l2_delta)
  syn0 += X.T.dot(l1_delta)
# Neural network with Z variable as target
for k in range(60000):
  #feed forward with sigmoid activation for first layer
  l3 = 1/(1+np.exp(-(np.dot(X,syn3))))
  #feed forward with sigmoid activation for first layer
  l4 = 1/(1+np.exp(-(np.dot(l3,syn4))))
  l4_delta = (z - l4)*(l4*(1-l4))
  l3_delta = l4_delta.dot(syn4.T) * (l3 * (1-l3))
  syn4 += l3.T.dot(l4_delta)
  syn3 += X.T.dot(l3_delta)

  #feed forward with sigmoid activation for first layer
  l1 = 1/(1+np.exp(-(np.dot(X,syn0))))
  #feed forward with sigmoid activation for Second layer
  l2 = 1/(1+np.exp(-(np.dot(l1,syn1))))
  #Backpropagation of error for second layer of weights with derivative of Sigmoid
  l2_delta = (y - l2)*(l2*(1-l2))
  #Backpropagation of error for first layer of weights with derivative of Sigmoid and the summation
  l1_delta = l2_delta.dot(syn1.T) * (l1 * (1-l1))
  #update of weights in both layers with respective errors from back propagation
  syn1 += l1.T.dot(l2_delta)
  syn0 += X.T.dot(l1_delta)
```

```
[[ 0.59748185  0.99880913 -0.86035202  0.11367176]
 [ 0.2285702   0.58677096  0.94756785 -0.32049195]
 [ 0.33771263 -0.66553232 -0.07699487  0.72668792]]
[[ 0.26880237]
 [-0.10724149]
 [ 0.86003988]
 [ 0.64647042]]
[[ 0.59748185  0.99880913 -0.86035202  0.11367176]
 [ 0.2285702   0.58677096  0.94756785 -0.32049195]
 [ 0.33771263 -0.66553232 -0.07699487  0.72668792]]
[[ 0.26880237]
 [-0.10724149]
 [ 0.86003988]
 [ 0.64647042]]
```

```
Z=[]
for a in range(8):
```

```
  Z.append([y[a],z[a]])
Z=np.array(Z)
np.reshape(Z,(8,2))
```

Output

```
array([[0, 0],
       [1, 0],
       [1, 0],
       [0, 1],
       [1, 0],
       [0, 1],
       [0, 1],
       [1, 1]])
```

```
Z

c = []
d = []
for a in  X:

     l5 = 1/(1+np.exp(-(np.dot(a,syn0))))
     l6 = 1/(1+np.exp(-(np.dot(l5,syn1))))
     l7 = 1/(1+np.exp(-(np.dot(a,syn3))))
     l8 = 1/(1+np.exp(-(np.dot(l7,syn4))))
     c.append(round(list(l6)[0]))
     d.append(round(list(l8)[0]))
print(c)
if c == list(y) and d==list(z):
  print ('test successful')
else:
  print ('failure')

import numpy as np

import numpy as np
X = np.array([[0,0,0],[0,0,1],[0,1,0],[0,1,1],[1,0,0],[1,0,1],[1,1,0],[1,1,1] ])
y = np.array([[0,1,1,0,1,0,0,1]]).T
z = np.array([[0,0,0,1,0,1,1,1]]).T
Z=[]
for a in range(8):# making the labels
  Z.append([y[a],z[a]])
Z=np.reshape(Z,(8,2))

syn0 = 2*np.random.random((3,4)) - 1
syn1 = 2*np.random.random((4,2)) - 1

#Neural network with Z variable as target
for j in range(60000):
  #feed forward with sigmoid activation for first layer
```

```
  l1 = 1/(1+np.exp(-(np.dot(X,syn0))))
  #feed forward with sigmoid activation for first layer
  l2 = 1/(1+np.exp(-(np.dot(l1,syn1))))
  l2_delta = (Z - l2)*(l2*(1-l2))
  l1_delta = l2_delta.dot(syn1.T) * (l1 * (1-l1))
  syn1 += l1.T.dot(l2_delta)
  syn0 += X.T.dot(l1_delta)
# Neural network with Z variable as target


l4 = 1/(1+np.exp(-(np.dot([1,1,0],syn0))))
l5 = 1/(1+np.exp(-(np.dot(l4,syn1))))
l5

import numpy as np
X = np.array([[0,0,0],[0,0,1],[0,1,0],[0,1,1],[1,0,0],[1,0,1],[1,1,0],[1,1,1] ])
y = np.array([[0,1,1,0,1,0,0,1]]).T
z = np.array([[0,0,0,1,0,1,1,1]]).T
labels=[]
for a in range(8):# making the labels
  labels.append([y[a],z[a]])
labels=np.reshape(labels,(8,2))
# initializing random neurons
syn0 = 2*np.random.random((3,4)) - 1
syn1 = 2*np.random.random((4,2)) - 1

#Neural network with Z variable as target
for j in range(90000):
  #feed forward with sigmoid activation for first layer
  b = np.dot(X,syn0)
  l1 = 1/(1+np.exp(-(b)))
  #feed forward with sigmoid activation for first layer
  c = np.dot(l1,syn1)
  l2 = 1/(1+np.exp(-(c)))
  sigmoid_derivatives2 = (l2*(1-l2))
  l2_delta = (labels - l2)* sigmoid_derivatives2
  sigmoid_derivatives1 =(l1 * (1-l1))
  l1_delta = l2_delta.dot(syn1.T) * sigmoid_derivatives1
  syn1 += l1.T.dot(l2_delta)
  syn0 += X.T.dot(l1_delta)
```