```
#Assignment2
#Roll No : 3350
#Name : Rajiya Mulla
#Write a program for Water jug problem


max1=int(input("capacity of JUG1 : "))
max2=int(input("capacity of JUG2 : "))
fill=int(input("how much liters of water fill in JUG1 : "))

def pour(jug1,jug2):
    print("%d\t%d" % (jug1,jug2))
    if jug2==fill:
        return
    elif jug2==max2:
        pour(0,jug1)
    elif jug1 != 0 and jug2==0:
        pour(0,jug1)
    elif jug1==fill:
        pour(jug1,0)
    elif jug1 < max1:
        pour(max1,jug2)
    elif jug1 <(max2-jug2):
        pour(0, (jug1+jug2))
    else:
        pour(jug1-(max2-jug2), (max2-jug2)+jug2)

print("JUG1\tJUG2")
pour(0,0)
```

```
capacity of JUG1 : 3
capacity of JUG2 : 4
how much liters of water fill in JUG1 : 2
JUG1    JUG2
0       0
3       0
0       3
3       3
2       4
0       2
```

```
# Assignment No. 2
# Rollno.: 3308
# Name: Nikita Bankar
# Problem Statement: Implemennt N-Queens Problem as Constraint Statisfaction Problem


def print_board(board):
    for row in board:
        print(" ".join(row))

def is_safe(board, row, col):
    for i in range(col):
        if board[row][i]=="Q":
            return False
    for i, j in zip(range(row, -1, -1),range(col, -1, -1)):
        if board[i][j]=="Q":
            return False
    for i, j in zip(range(row, len(board), -1) ,range(col, -1, -1)):
        if board[i][j]=="Q":
            return False
    return True

def solve(board, col):
    if col>=len(board):
        return True
    for i in range(len(board)):
        if is_safe(board, i, col):
            board[i][col]="Q"
            if solve(board, col+1):
                return True
            board[i][col]="."
    return False

n=int(input("Enter the Number of Queens: "))
board=[["." for i in range(n)] for j in range(n)]

if solve(board, 0):
    print_board(board)
else:
    print("Solution not found")
```

```
Enter the Number of Queens: 4
. . Q .
Q . . .
. . . Q
. Q . .
```

```
# Assignment No. 4
# Rollno.: 3308
# Name: Nikita Bankar
# Problem Statement: Write a program for the Information Retrieval System
# using appropriate NLP tools (such as NLTK, Open NLP, ...)
# a. Text tokenization
# b. Count word frequency
# c. Remove stop words
# d. POS tagging


import nltk
nltk.download('punkt')
from nltk.corpus import stopwords
nltk.download('stopwords')
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
from nltk.tag import pos_tag
nltk.download('averaged_perceptron_tagger')

#Define the text to be analysed
text="This is a sample sentence. It contains multiple words and some of them repeat . We will analyze this text using NLTK"

#Tokenize the text into words
words=word_tokenize(text)
print("tokenized words:")
print(words)

#Convert all words to lowercase
words = [word.lower() for word in words]

#Count the frequency of each word
fdist = FreqDist(words)
print("Word Frequency:")
for word, freq in fdist.items():
    print(f"{word}: {freq}")

#Remove stopwords
stop_words = set(stopwords.words('english'))
filtered_words = [word for word in words if word.casefold() not in stop_words]
print("Filtered Words:")
print(filtered_words)

#Perform POS tagging
pos_tags = pos_tag(words)
print(pos_tags)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
tokenized words:
['This', 'is', 'a', 'sample', 'sentence', '.', 'It', 'contains', 'multiple', 'words', 'and', 'some', 'of', 'them', 'repeat', '.', '
Word Frequency:
this: 2
is: 1
a: 1
sample: 1
sentence: 1
.: 2
it: 1
contains: 1
multiple: 1
words: 1
and: 1
some: 1
of: 1
them: 1
repeat: 1
we: 1
will: 1
analyze: 1
text: 1
using: 1
nltk: 1
Filtered Words:
['sample', 'sentence', '.', 'contains', 'multiple', 'words', 'repeat', '.', 'analyze', 'text', 'using', 'nltk']
[('this', 'DT'), ('is', 'VBZ'), ('a', 'DT'), ('sample', 'JJ'), ('sentence', 'NN'), ('.', '.'), ('it', 'PRP'), ('contains', 'VBZ'),
```

```
# Assignment No. 1
# Rollno.: 3308
# Name: Nikita Bankar
# Problem statement: Identify and Implement heuristic and search strategy for
#Travelling salesperson Problem

import sys
def nearest_neighbor(curr,unvisited,dist_matrix):
    """Returns the nearest neighbor to the current city"""
    nearest = 900
    #nearest=sys.maxsize
    neighbor=None
    for city in unvisited:
        if dist_matrix[curr][city]<nearest:
            nearest=dist_matrix[curr][city]
            neighbor=city
    return neighbor,nearest

def tsp_nn(dist_matrix):
    """Solves the travelling salesman problem using the nearest neighbor algorithm"""
    n=len(dist_matrix)
    tour=[0]*5 #Initialize the tour
    unvisited=set(range(1,n)) #Set of unvisited cities
    curr_city=0 #Starting city

    for i in range(1,n):
        next_city,dist=nearest_neighbor(curr_city,unvisited,dist_matrix)
        tour[i]=next_city
        curr_city=next_city
        unvisited.remove(next_city)

        #return to the strating city
        tour[0]=0

        #calculate the total cost of the tour
        cost=sum(dist_matrix[tour[i]][tour[i+1]] for i in range(n-1))
        cost+=dist_matrix[tour[n-1]][tour[0]]
        return tour,cost


Nodes:int(input("Enter the Nodes: "))
r=int(input("Enter the number of rows: "))
c=int(input("Enter the number of columns: "))
dist_matrix=[]
print("Enter elements row-wise: ")

for i in range(r):
    a=[]
    for j in range(c) :
        a.append(int(input()))
    dist_matrix.append(a)

for i in range(r):
    for j in range(c):
        print(dist_matrix[i][j],end=" ")
    print()

# dist_matrix=[
#     [0,5,15,4],
#     [5,0,35,25],
#     [15,35,0,30],
#     [4,25,30,0]
# ]

tour, cost = tsp_nn(dist_matrix)
print("Tour:",tour)
print("Total cost:",cost)


     Enter the Nodes: 4
     Enter the number of rows: 4
     Enter the number of columns: 4
     Enter elements row-wise:
     0
     1
     2
     3
     4
     5
     6
     7
     8
```

```
9
5
4
3
2
1
7
0 1 2 3
4 5 6 7
8 9 5 4
3 2 1 7
Tour: [0, 1, 0, 0, 0]
Total cost: 5
```

```
# Assignment No. 3
# Rollno.: 3308
# Name: Nikita Bankar
# Problem Statement: Implement Water-Jug Problem using Rule Based Reasoning Technique

def pour(jug1, jug2):
    max1, max2, fill = 3, 4, 2 #Change maximum capacity and final capacity
    print("%d\t%d" % (jug1, jug2))
    if jug2 == fill:
        return
    elif jug2 == max2:
        pour(0, jug1)
    elif jug1 != 0 and jug2 == 0:
        pour(0, jug1)
    elif jug1 == fill:
        pour(jug1, 0)
    elif jug1 < max1:
        pour(max1, jug2)
    elif jug1 < (max2-jug2):
        pour(0, (jug1+jug2))
    else:
        pour(jug1-(max2-jug2), (max2-jug2)+jug2)

print("JUG1\tJUG2")
pour(0, 0)
```

```
JUG1    JUG2
0       0
3       0
0       3
3       3
2       4
0       2
```