Cse-334

Artificial intelligence Lab

Submitted To

Mosaddek Adnan Sikdar

Lecturer, Dept. of CSE

Primeasia University

Submitted By

Md.soykot

181-041-042

**1. Fibonacci sequence without recursion**

n=int(input("Enter number:"))

fiboPrev=0

fibo=1

print("Fibonacci sequence for {0} number:".format(n))

for i in range(n):

print(fiboPrev,end=" ")

fiboNext=fiboPrev+fibo

fiboPrev=fibo

fibo=fiboNext

**2. Fibonacci Sequence Using Recursion**

def fibo(n):

if n<=1:

return n

else:

return fibo(n-1)+fibo(n-2)

n=int(input("Enter number:"))

print("Fibonacci sequence for {0} number:".format(n))

for i in range(n):

print(fibo(i),end=" ")

**3. Armstrong Number in an Interval**

print("Enter two number lower and upper range:")

lower=int(input())

upper=int(input())

for num in range(lower,upper+1):

total=0

temp=num

while temp>0:

singleDigit=temp%10

total +=singleDigit\*\*3

temp //=10 #it's give only integer value

if num==total:

print(num)

**4. Convert Binary to Decimal Using Recursion.**

def binToDec(num):

if num ==1 or num == 0:

return num

length=len(str(num))

firstDigit=num//pow(10,length-1)

return (pow(2,length-1)\*firstDigit)+binToDec(num%pow(10,length-1))

binary=int(input('Enter a binary number: '))

decimal=binToDec(binary)

print('Deccimal of {0} is {1}'.format(binary,decimal))

**5. Convert Decimal to Binary, Octal and Hexadecimal.**

decimal=int(input('Enter a decimal number: '))

print(decimal, "in binary",bin(decimal).replace("0b", "") )

print(decimal, "in Octal : ", oct(decimal).replace("0o", ""))

print(decimal, " in Hexadecimal : ", hex(decimal).replace("0x", ""))

**6. Multiply Two Matrices.**

def matrics(r,c):

matrix=[]

for i in range(r):

a=[]

for j in range(c):

a.append(int(input()))

matrix.append(a)

return matrix

print("=========Enter first matrix======== ")

r1 = int(input("Enter the number of rows:"))

c1 = int(input("Enter the number of columns:"))

firstMatrix=matrics(r1,c1)

print("First matrix:")

for i in range(r1):

for j in range(c1):

print(firstMatrix[i][j],end=" ")

print()

print("==========Enter second matrix========= ")

r2 = int(input("Enter the number of rows:"))

c2 = int(input("Enter the number of columns:"))

secondMatrix=matrics(r2,c2)

print("Second matrix:")

for i in range(r2):

for j in range(c2):

print(secondMatrix[i][j],end=" ")

print()

result=[[0 for x in range(r1)] for y in range(c2)]

print("Multiplication matrix:")

if r1==c2:

for i in range(len(firstMatrix)):

for j in range(len(secondMatrix[0])):

for k in range(len(secondMatrix)):

result[i][j] +=firstMatrix[i][k]\*secondMatrix[k][j]

for r in result:

print(" ".join(map(str,r)))

else:

print("Invalid input")

**7. Transpose a Matrix.**

def matrics(r,c):

matrix=[]

for i in range(r):

a=[]

for j in range(c):

a.append(int(input()))

matrix.append(a)

return matrix

r = int(input("Enter the number of rows:"))

c = int(input("Enter the number of columns:"))

firstMatrix=matrics(r,c)

print("matrix:")

for i in range(r):

for j in range(c):

print(firstMatrix[i][j],end=" ")

print()

result=[[ 0 for x in range(r)] for y in range(c)]

for i in range(len(firstMatrix)):

for j in range(len(firstMatrix[0])):

result[j][i]=firstMatrix[i][j]

print("Transpose matrix:")

for r in result:

print(r)

**8. Count the Number of Each Vowel on a given sentence**

sen=input("Enter a sentence: ")

lowerCase=sen.lower()

vowelCount={}

for vowel in "aeiou":

count=lowerCase.count(vowel)

vowelCount[vowel]=count

print(vowelCount)

**9. A\* search algorithm**

class Node():

def \_\_init\_\_(self, parent=None, position=None):

self.parent = parent

self.position = position

self.g = 0

self.h = 0

self.f = 0

def \_\_eq\_\_(self, other):

return self.position == other.position

def astar(maze, start, end):

start\_node = Node(None,start)

start\_node.g = start\_node.h = start\_node.f = 0

end\_node = Node(None, end)

end\_node.g = end\_node.h = end\_node.f = 0

open\_list = []

closed\_list = []

open\_list.append(start\_node)

while len(open\_list) > 0:

current\_node = open\_list[0]

current\_index = 0

for index, item in enumerate(open\_list):

if item.f < current\_node.f:

current\_node = item

current\_index = index

open\_list.pop(current\_index)

closed\_list.append(current\_node)

if current\_node == end\_node:

path = []

current = current\_node

while current is not None:

path.append(current.position)

current = current.parent

return path[::-1]

children = []

for new\_position in [(0, -1), (0, 1), (-1, 0), (1, 0), (-1, -1), (-1, 1), (1, -1), (1, 1)]: # Adjacent squares

node\_position = (current\_node.position[0] + new\_position[0], current\_node.position[1] + new\_position[1])

if node\_position[0] > (len(maze) - 1) or node\_position[0] < 0 or node\_position[1] > (len(maze[len(maze)-1]) -1) or node\_position[1] < 0:

continue

if maze[node\_position[0]][node\_position[1]] != 0:

continue

new\_node = Node(current\_node, node\_position)

children.append(new\_node)

for child in children:

for closed\_child in closed\_list:

if child == closed\_child:

continue

child.g = current\_node.g + 1

child.h = ((child.position[0] - end\_node.position[0]) \*\* 2) + ((child.position[1] - end\_node.position[1]) \*\* 2)

child.f = child.g + child.h

for open\_node in open\_list:

if child == open\_node and child.g > open\_node.g:

continue

open\_list.append(child)

def main():

maze = [[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 1, 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, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 1, 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]]

start = (0, 0)

end = (7, 6)

path = astar(maze, start, end)

print(path)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**10. Tic-tac-toe**

theBoard = {'7': ' ' , '8': ' ' , '9': ' ' ,

'4': ' ' , '5': ' ' , '6': ' ' ,

'1': ' ' , '2': ' ' , '3': ' ' }

board\_keys = []

for key in theBoard:

board\_keys.append(key)

def printBoard(board):

print(board['7'] + '|' + board['8'] + '|' + board['9'])

print('-+-+-')

print(board['4'] + '|' + board['5'] + '|' + board['6'])

print('-+-+-')

print(board['1'] + '|' + board['2'] + '|' + board['3'])

def game():

turn = 'X'

count = 0

for i in range(10):

printBoard(theBoard)

print("It's your turn," + turn + ".Move to which place?")

move = input()

if theBoard[move] == ' ':

theBoard[move] = turn

count += 1

else:

print("That place is already filled.\nMove to which place?")

continue

if count >= 5:

if theBoard['7'] == theBoard['8'] == theBoard['9'] != ' ':

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['4'] == theBoard['5'] == theBoard['6'] != ' ':

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['1'] == theBoard['2'] == theBoard['3'] != ' ':

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['1'] == theBoard['4'] == theBoard['7'] != ' ':

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['2'] == theBoard['5'] == theBoard['8'] != ' ':

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['3'] == theBoard['6'] == theBoard['9'] != ' ':

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['7'] == theBoard['5'] == theBoard['3'] != ' ':

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['1'] == theBoard['5'] == theBoard['9'] != ' ':

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

if count == 9:

print("\nGame Over.\n")

print("It's a Tie!!")

if turn =='X':

turn = 'O'

else:

turn = 'X'

restart = input("Do want to play Again?(y/n)")

if restart == "y" or restart == "Y":

for key in board\_keys:

theBoard[key] = " "

game()

if \_\_name\_\_ == "\_\_main\_\_":

game()

**11. BFS Implementation**

graph = {

1:[2,3,4],

2:[],

3:[5],

4:[6],

5:[],

6:[]

}

visited = []

queue = []

def bfs( node):

visited.append(node)

queue.append(node)

while queue:

s = queue.pop(0)

print (s, end = " ")

for neighbour in graph:

if neighbour not in visited:

visited.append(neighbour)

queue.append(neighbour)

bfs(1)

**12. DFS Implementation**

graph = {

1:[2,3],

2:[4,5],

3:[6],

4:[],

5:[6],

6:[]

}

visited =[]

print("Depth first search:")

def dfs(node):

if node not in visited:

print (node,end=" ")

visited.append(node)

for neighbour in graph[node]:

dfs(neighbour)

dfs(1)

**13. Depth Limited Search Implementation**

graph = {

1:[2,3],

2:[4,5],

3:[6],

4:[],

5:[6],

6:[]

}

visited =[]

print("Depth limited search:")

def dfs(node,limit):

if not limit:

return 0

else:

limit -=1

if node not in visited:

print (node,end=" ")

visited.append(node)

for neighbour in graph[node]:

dfs(neighbour,limit)

dfs(1,2)

**14. Naive Bayes (Probability of Playing if Raining)**

weather\_tbl=[]

play\_tbl=[]

n=int(input("Enter number of dataset:"))

for i in range(n):

weather=input("Enter weather name:")

weather\_tbl.append(weather.lower())

play=input("Enter paly or not:")

play\_tbl.append(play.lower())

t\_rainy\_y,t\_rainy\_n=0,0

t\_y,t\_n=0,0

t\_rainy=0

for i in range(n):

if play\_tbl[i]=="yes":

t\_y +=1

else:

t\_n +=1

if weather\_tbl[i]=="rainy":

t\_rainy +=1

if play\_tbl[i]=="yes":

t\_rainy\_y +=1

else:

t\_rainy\_n +=1

p\_rainy\_y=(t\_rainy\_y/t\_y)

p\_y=(t\_y/n)

p\_rainy=(t\_rainy/n)

p\_rainy\_n=(t\_rainy\_n/t\_n)

p\_n=(t\_n/n)

p\_y\_rainy=(p\_rainy\_y\*p\_y)/p\_rainy

p\_n\_rainy=(p\_rainy\_n\*p\_n)/p\_rainy

if p\_n\_rainy>p\_y\_rainy:

print("Player will play,if it not rainy")

else:

print("player will not play if it rainy")

**15. Naive Bayes (Classifying Unknown Fruit)**

typ\_col=[]

lon\_col=[]

n\_lon\_col=[]

swt\_col=[]

n\_swt\_col=[]

yell\_col=[]

n\_yell\_col=[]

total\_col=[]

d\_set=int(input("Enter number of dataset:"))

n=int(input("Eneter total fruits:"))

for i in range(n):

typ=input("Enter fruits type:")

typ\_col.append(typ.lower())

lon=int(input("Enter long fruits:"))

lon\_col.append(lon)

n\_lon=int(input("Enter not long fruits:"))

n\_lon\_col.append(n\_lon)

swt=int(input("Enter sweet fruits:"))

swt\_col.append(swt)

n\_swt=int(input("Enter not sweet fruits:"))

n\_swt\_col.append(n\_swt)

yell=int(input("Enter yellow fruits:"))

yell\_col.append(yell)

n\_yell=int(input("Enter not yellow fruits:"))

n\_yell\_col.append(n\_yell)

total=int(input("Enter total fruits:"))

total\_col.append(total)

p\_banana,p\_orange,p\_other=0,0,0

for i in range(n):

if typ\_col[i]=="banana":

p\_banana=total\_col[i]/d\_set

p\_long\_b=lon\_col[i]/total\_col[i]

p\_sweet\_b=swt\_col[i]/total\_col[i]

p\_yell\_b=yell\_col[i]/total\_col[i]

elif typ\_col[i]=="orange":

p\_orange=total\_col[i]/d\_set

p\_long\_o=lon\_col[i]/total\_col[i]

p\_sweet\_o=swt\_col[i]/total\_col[i]

p\_yell\_o=yell\_col[i]/total\_col[i]

elif typ\_col[i]=="other":

p\_other=total\_col[i]/d\_set

p\_long\_other=lon\_col[i]/total\_col[i]

p\_sweet\_other=swt\_col[i]/total\_col[i]

p\_yell\_other=yell\_col[i]/total\_col[i]

p\_fruits\_b=p\_long\_b\*p\_sweet\_b\*p\_yell\_b

p\_fruits\_o=p\_long\_o\*p\_sweet\_o\*p\_yell\_o

p\_fruits\_other=p\_long\_other\*p\_sweet\_other\*p\_yell\_other

new\_fruit=max(p\_fruits\_b,p\_fruits\_o,p\_fruits\_other)

if new\_fruit==p\_fruits\_b:

print("========New fruit will be banana==========")

elif new\_fruit==p\_fruits\_o:

print("=========New fruit will be orange=======")

else:

print("======new fruits will be other fruits=======")