Assignment 1 (bfs dfs)

from collections import deque

class Graph:

def \_\_init\_\_(self,edges,n\_vertices):

self.adjList = [[] for i in range(n\_vertices)]

for (src,dest) in edges:

self.adjList[src].append(dest)

self.adjList[dest].append(src)

def BFS(graph,q,visited):

if not q:

return

v = q.popleft()

print(v,end = " ")

for i in graph.adjList[v]:

if not visited[i]:

q.append(i)

visited[i] = True

BFS(graph, q, visited)

def DFS(graph,i,visited):

visited[i] = True

print(i, end=" ")

for j in graph.adjList[i]:

if not visited[j]:

DFS(graph, j, visited)

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

edges = [(0,1),(0,2),(0,3),(1,2),(2,4)]

n\_vertices = 5

q = deque()

visited = [False]\*n\_vertices

graph = Graph(edges,n\_vertices)

#BFS

for i in range(n\_vertices):

if not visited[i]:

q.append(i)

visited[i] = True

BFS(graph,q,visited)

#DFS

for i in range(n\_vertices):

if not visited[i]:

DFS(graph,i,visited)

Assignment 2 (a\*)

class Node:

def \_\_init\_\_(self,data,level,fval):

self.data = data

self.level = level

self.fval = fval

def generate\_child(self):

x,y = self.find(self.data,'\_')

val\_list = [[x,y-1],[x,y+1],[x-1,y],[x+1,y]]

children = []

for i in val\_list:

child = self.shuffle(self.data,x,y,i[0],i[1])

if child is not None:

child\_node = Node(child,self.level+1,0)

children.append(child\_node)

return children

def shuffle(self,puz,x1,y1,x2,y2):

if x2 >= 0 and x2 < len(self.data) and y2 >= 0 and y2 < len(self.data):

temp\_puz = []

temp\_puz = self.copy(puz)

temp = temp\_puz[x2][y2]

temp\_puz[x2][y2] = temp\_puz[x1][y1]

temp\_puz[x1][y1] = temp

return temp\_puz

else:

return None

def copy(self,root):

temp = []

for i in root:

t = []

for j in i:

t.append(j)

temp.append(t)

return temp

def find(self,puz,x):

for i in range(0,len(self.data)):

for j in range(0,len(self.data)):

if puz[i][j] == x:

return i,j

class Puzzle:

def \_\_init\_\_(self,size):

self.n = size

self.open = []

self.closed = []

def accept(self):

puz = []

for i in range(0,self.n):

temp = input().split(" ")

puz.append(temp)

return puz

def f(self,start,goal):

return self.h(start.data,goal)+start.level

def h(self,start,goal):

temp = 0

for i in range(0,self.n):

for j in range(0,self.n):

if start[i][j] != goal[i][j] and start[i][j] != '\_':

temp += 1

return temp

def process(self):

print("Enter the start state matrix \n")

start = self.accept()

print("Enter the goal state matrix \n")

goal = self.accept()

start = Node(start,0,0)

start.fval = self.f(start,goal)

self.open.append(start)

print("\n\n")

while True:

cur = self.open[0]

print("")

print(" | ")

print(" | ")

print(" \\\'/ \n")

for i in cur.data:

for j in i:

print(j,end=" ")

print("")

if(self.h(cur.data,goal) == 0):

break

for i in cur.generate\_child():

i.fval = self.f(i,goal)

self.open.append(i)

self.closed.append(cur)

del self.open[0]

self.open.sort(key = lambda x:x.fval,reverse=False)

puz = Puzzle(3)

puz.process()

Asiignment3 (job scheduling)

from operator import itemgetter

print("==================================================")

num = int(input("Enter number of jobs : "))

list = [[] for i in range(num)]

for i in range(num):

print("-------------------------------")

for j in range(3):

if j == 0:

k = int(input("Enter job no: "))

list[i].append(k)

elif j == 1:

k = int(input("Enter Profit: "))

list[i].append(k)

else:

k = int(input("Enter deadline (days): "))

list[i].append(k)

print(list)

list = sorted(list, key= lambda x:x[1],reverse=True)

plan = ["No Jobs"]\*num

for i in range(num):

if plan[list[i][2]-1] == "No Jobs":

plan[list[i][2]-1] = list[i][0]

else:

for j in range(list[i][2]):

if plan[j] == "No Jobs":

plan[j] = list[i][0]

break

print("==================================================")

print("Jobs assigned are as follows:")

print(plan)

Assignment 3 (N Queens)

Part 1 backtracking

#define N 5

#include <stdbool.h>

#include <stdio.h>

void printSolution(int board[N][N])

{

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++)

printf(" %d ", board[i][j]);

printf("\n");

}

}

bool isSafe(int board[N][N], int row, int col)

{

int i, j;

/\* Check this row on left side \*/

for (i = 0; i < col; i++)

if (board[row][i])

return false;

/\* Check upper diagonal on left side \*/

for (i = row, j = col; i >= 0 && j >= 0; i--, j--)

if (board[i][j])

return false;

/\* Check lower diagonal on left side \*/

for (i = row, j = col; j >= 0 && i < N; i++, j--)

if (board[i][j])

return false;

return true;

}

bool solveNQUtil(int board[N][N], int col)

{

/\* base case: If all queens are placed

then return true \*/

if (col >= N)

return true;

/\* Consider this column and try placing

this queen in all rows one by one \*/

for (int i = 0; i < N; i++) {

if (isSafe(board, i, col)) {

/\* Place this queen in board[i][col] \*/

board[i][col] = 1;

/\* placing other queens \*/

if (solveNQUtil(board, col + 1))

return true;

board[i][col] = 0; // BACKTRACK

}

}

return false;

}

bool solveNQ()

{

int board[N][N] = { { 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 } };

if (solveNQUtil(board, 0) == false) {

printf("Solution does not exist");

return false;

}

printSolution(board);

return true;

}

int main()

{

solveNQ();

return 0;

}

Part 2 branch and bound

N = 4

def printBoard(board):

for i in range(N):

for j in range(N):

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

print()

def isSafe(row, col, slashCode, backslashCode,rowLookup, slashCodeLookup,backslashCodeLookup):

if (slashCodeLookup[slashCode[row][col]] or backslashCodeLookup[backslashCode[row][col]] or rowLookup[row]):

return False

return True

def solveNQueensUtil(board, col, slashCode, backslashCode, rowLookup, slashCodeLookup, backslashCodeLookup,):

if(col >= N):

return True

for i in range(N):

if(isSafe(i, col, slashCode, backslashCode, rowLookup, slashCodeLookup, backslashCodeLookup)):

board[i][col] = 1

rowLookup[i] = True

slashCodeLookup[slashCode[i][col]] = True

backslashCodeLookup[backslashCode[i][col]] = True

print()

printBoard(board)

print()

if(solveNQueensUtil(board, col + 1, slashCode, backslashCode, rowLookup, slashCodeLookup, backslashCodeLookup,)):

return True

board[i][col] = 0

rowLookup[i] = False

slashCodeLookup[slashCode[i][col]] = False

backslashCodeLookup[backslashCode[i][col]] = False

print('the column to be changed : ',end=" ")

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

print('col = ',col)

return False

def solveNQueens():

board = [[0 for i in range(N)] for j in range(N)]

slashCode = [[0 for i in range(N)] for j in range(N)]

backslashCode = [[0 for i in range(N)] for j in range(N)]

rowLookup = [False] \* N

x = 2 \* N - 1

slashCodeLookup = [False] \* x

backslashCodeLookup = [False] \* x

for rr in range(N):

for cc in range(N):

slashCode[rr][cc] = rr + cc

backslashCode[rr][cc] = rr - cc + (N-1)

if(solveNQueensUtil(board, 0, slashCode, backslashCode, rowLookup, slashCodeLookup, backslashCodeLookup,) == False):

print("Solution does not exist")

return False

printBoard(board)

return True

solveNQueens()

Assignment5 (chatbot)

from nltk.chat.util import Chat, reflections

pairs = [

[

r"hey|hellp|hi",

["Hi, how are you"]

],

[

r"what is your name?",

["Hi iam chatty"]

],

[

r"My name is (.\*)",

["Hi Thats a nice name"]

]

]

def chatty():

print("Hello Im chittyty chityy robo")

chat = Chat(pairs,reflections)

chat.converse()

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

chatty()

Assignment6 (Expert system)

print('Welcome to COVID-19 Expert system')

covidSuspisionCounter=0

severity=0

asym=0

oxylevel=0

temp=0

questions=['What is your body temparature','What is your oxygen level','How many vaccines have you taken','What is your age']

yesnoqs=['Do you have cough and cold','Are you able to recognize smell and taste','Are you suffering from sore throat','Are you suffering from headache','Are you suffering from BP/ diabetes','Have you come in a contact of a Covid suspicious person']

for i in range(6):

print(yesnoqs[i])

print()

ans=input()

if(i!=1 and ans=='yes'):

covidSuspisionCounter+=1

elif(i==1 and ans=='no'):

covidSuspisionCounter+=1

for i in range(4):

print(questions[i])

print()

if(i==0):

ans=float(input())

if(ans>=101.0):

severity+=2

covidSuspisionCounter+=1

temp=1

elif(ans<101.0 and ans>=99.6):

severity+=1

else:

severity+=0

if(i==1):

ans=int(input())

if(ans>=94):

severity+=0

elif(ans<94 and ans>87):

severity+=1

else:

severity+=2

covidSuspisionCounter+=1

oxylevel=1

if(i==2):

ans=int(input())

if(ans==0):

severity+=2

elif(ans==1):

severity+=1

else:

severity+=0

if(i==3):

ans=int(input())

if(ans>12 and ans<31):

severity+=0

elif(ans>31 and ans<51):

severity+=1

else:

severity+=2

if(covidSuspisionCounter>3):

print('The patient is probably covid positive')

print()

if(severity<3):

print('It looks like the symptoms are mild\nhome quarantine')

elif(severity>=3 and severity<6):

print('The patient can get an admission in the general ward')

else:

print('The patient looks critical')

else:

print('It looks like patient is not Covid positive')

print()

if(oxylevel==1):

print("Keep monitoring patient's oxygen level")

if(temp==1):

print("Keep monitoring patient's body temperature")

Assignment 7 (KVM)

egrep -c "(vmx|svm)" /proc/cpuinfo

sudo apt install qemu-kvm libvirt-clients libvirt-daemon-system bridge-utils

sudo adduser libvirt

sudo adduser kvm

sudo apt install virt-manager

sudo virt-manager

Assignment 8 (Sales Force)

Cal.apex file

public class Calculator {

public integer num1 {get;set;}

public integer num2 {get;set;}

public integer res {get;set;}

public void add(){

res = num1+num2;

}

public void sub(){

res = num1-num2;

}

public void mul(){

res = num1\*num2;

}

public void div(){

res = num1/num2;

}

}

Visual file

<apex:page controller="Calculator" >

<apex:form >

<apex:pageBlock title="Calculator">

Number one : <apex:inputText value="{!num1}"></apex:inputText><br/>

Number two : <apex:inputText value="{!num2}"></apex:inputText><br/>

<apex:pageBlockButtons >

<apex:commandButton value="ADD" action="{!add}"/>

<apex:commandButton value="SUB" action="{!sub}"/>

<apex:commandButton value="MUL" action="{!mul}"/>

<apex:commandButton value="DIV" action="{!div}"/>

</apex:pageBlockButtons>

RESULT : <apex:outputText value="{!res}"></apex:outputText>

</apex:pageBlock>

</apex:form>

</apex:page>