

PRACTICAL FILE

**ARTIFICIAL INTELLIGENCE AND DATA SCIENCE
(ITMDC02)**

**Master of Technology
in
Mobile Communication and Network Technology**

By

**Ineesh Raina
2023PMN4207**

**Sakshi Sharma
2023PMN4205**

**Ruchi Dayal
2023PMN4210**

**Anshika Rai
2023PMN4201**



Submitted to :

Dr. Meena Jha

INDEX

S. No.	Title	Concept
1	Tic Tac Toe	Implementation of Tic-Tac-Toe game
2	Tile Slide Puzzle	Minimize heuristic
3	Water Jug Problem	Production Rules
4	Generate and Test	Rat in a Maze
5	Systematic Generate and Test	Rat in a Maze
6	Hill Climbing	Minimize cost of Living
7	Steepest Ascent Hill Climbing	Minimize cost of travel
8	Best First Search	Shortest path
9	A* Algorithm	Shortest path
10	AO* Algorithm	Shortest path
11	NLP	Sentiment Analysis
12	Tic Tac Toe	Minimax with α - β pruning

Exp 1- Tic-Tac-Toe

Code:

```
def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 9)
def check_winner(board, player):
    for row in board:
        if all(cell == player for cell in row):
            return True
    for col in range(3):
        if all(board[row][col] == player for row in range(3)):
            return True
        if all(board[i][i] == player for i in range(3)) or all(board[i][2 - i] == player for i in range(3)):
            return True
    return False

def is_board_full(board):
```

```
return all(cell != " " for row in board for cell
in row)

def main():
    board = [[" " for _ in range(3)] for _ in range(3)]
    current_player = "X"
    print("Welcome to Tic Tac Toe!")
    print_board(board)

    while True:
        row = int(input(f"Player {current_player},\nenter row (0-2): "))
        col = int(input(f"Player {current_player},\nenter column (0-2): "))

        if row < 0 or row > 2 or col < 0 or col > 2
        or board[row][col] != " ":
            print("Invalid move. Try again.")
            continue
```

```
board[row][col] = current_player  
print_board(board)
```

```
if check_winner(board, current_player):  
    print(f"Player {current_player} wins!")  
    break
```

```
elif is_board_full(board):  
    print("It's a tie!")  
    break
```

```
    current_player = "O" if current_player ==  
    "X" else "X"
```

```
if __name__ == "__main__":  
    main()
```

Output:

```
Welcome to TIC Tac Toe!
|   |
-----
|   |
-----
|   |
-----
Player X, enter row (0-2): 0
Player X, enter column (0-2): 0
X |   |
-----
|   |
-----
|   |
-----
Player O, enter row (0-2): 2
Player O, enter column (0-2): 2
X |   |
-----
|   |
-----
|   | O
-----
Player X, enter row (0-2): 1
Player X, enter column (0-2): 0
X |   |
-----
X |   |
```

```
Player X, enter row (0-2): 1  
Player X, enter column (0-2): 0
```

```
X |   |
```

```
-----
```

```
X |   |
```

```
-----
```

```
 |   | 0
```

```
-----
```

```
Player O, enter row (0-2): 1
```

```
Player O, enter column (0-2): 2
```

```
X |   |
```

```
-----
```

```
X |   | 0
```

```
-----
```

```
 |   | 0
```

```
-----
```

```
Player X, enter row (0-2): 2
```

```
Player X, enter column (0-2): 0
```

```
X |   |
```

```
-----
```

```
X |   | 0
```

```
-----
```

```
X |   | 0
```

```
-----
```

```
Player X wins!
```

```
...Program finished with exit code 0
```

```
Press ENTER to exit console.
```

Implementation of Tile Slide Puzzle using Minimize heuristic

Code:

```
class Node:
    def __init__(self,data,level,fval):
        """ Initialize the node with the data, level of the node and the calculated
fvalue """
        self.data = data
        self.level = level
        self.fval = fval

    def generate_child(self):
        """ Generate child nodes from the given node by moving the blank space
either in the four directions {up,down,left,right} """
        x,y = self.find(self.data,'_')
        """ val_list contains position values for moving the blank space in either of
the 4 directions [up,down,left,right] respectively. """
        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):
        """ Move the blank space in the given direction and if the position value are out
of limits the return None """
        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):
        """ Copy function to create a similar matrix of the given node"""
        pass
```

```

temp = []
for i in root:
    t = []
    for j in i:
        t.append(j)
    temp.append(t)
return temp

def find(self,puz,x):
    """ Specifically used to find the position of the blank space """
    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):
        """ Initialize the puzzle size by the specified size,open and closed lists to
empty """
        self.n = size
        self.open = []
        self.closed = []

    def accept(self):
        """ Accepts the puzzle from the user """
        puz = []
        for i in range(0,self.n):
            temp = input().split(" ")
            puz.append(temp)
        return puz

    def f(self,start,goal):
        """ Heuristic Function to calculate hueristic value f(x) = h(x) + g(x) """
        return self.h(start.data,goal)+start.level

    def h(self,start,goal):
        """ Calculates the different between the given puzzles """
        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):
    """ Accept Start and Goal Puzzle state"""
    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)
    """ Put the start node in the open list"""
    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 the difference between current and goal node is 0 we have reached the
goal node"""
        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]

    """ sort the opne list based on f value """
    self.open.sort(key = lambda x:x.fval,reverse=False)

puz = Puzzle(3)
puz.process()

```

Output:

Shell

Clear

Enter the start state matrix

1 2 3
_ 4 5
6 7 8

Enter the goal state matrix

1 2 3
4 7 5
6 8 _
|
|
\\'

1 2 3
_ 4 5
6 7 8

|
|
\\'

1 2 3
4 _ 5
6 7 8

|
|
\\'

1 2 3
4 7 5
6 _ 8

|
|
\\'

1 2 3
4 7 5
6 8 _
> |

Implementation of Water Jug Problem

Code:

```
from collections import defaultdict

jug1 = int(input("Water in jug1: "))

jug2 = int(input("Water in jug2: "))

aim = int(input("Required Water in jug: "))

print("Steps:", jug1, "L", " ", jug2, "L")

visited = defaultdict(lambda: False)

def waterJugSolver(amt1, amt2):

    if (amt1 == aim and amt2 == 0) or (amt2 == aim and amt1 == 0):
        print(amt1, amt2)
        return True

    if visited[(amt1, amt2)] == False:
        print(amt1, amt2)
        visited[(amt1, amt2)] = True
        return (waterJugSolver(0, amt2) or
                waterJugSolver(amt1, 0) or
                waterJugSolver(jug1, amt2) or
                waterJugSolver(amt1, jug2) or
                waterJugSolver(amt1 + min(amt2, (jug1-amt1)),
                               amt2 - min(amt2, (jug1-amt1))) or
                waterJugSolver(amt1 - min(amt1, (jug2-amt2)),
                               amt2 + min(amt1, (jug2-amt2)))))

    else:
        return False
```

```
if jug1 >= jug2:  
    waterJugSolver(0, 0)  
elif jug1 <= jug2:  
    jug1, jug2 = jug2, jug1  
    waterJugSolver(0, 0)  
waterJugSolver(0, 0)
```

Output:

Shell

Clear

```
Water in jug1: 5  
Water in jug2: 4  
Required Water in jug: 2  
Steps: 5 L 4 L  
0 0  
5 0  
5 4  
0 4  
4 0  
4 4  
5 3  
0 3  
3 0  
3 4  
5 2  
0 2  
> |
```

EXPERIMENT 4: Rat in a Maze

CODE:

```
def isValid(n, maze, x, y, res):  
    if 0 <= x < n and 0 <= y < n and maze[x][y] == 1 and res[x][y] == 0:  
        return True  
    return False  
  
def RatMaze(n, maze, move_x, move_y, x, y, res):  
    if x == n - 1 and y == n - 1:  
        return True  
  
    for i in range(4):  
        x_new = x + move_x[i]  
        y_new = y + move_y[i]  
        if isValid(n, maze, x_new, y_new, res):  
            res[x_new][y_new] = 1  
            if RatMaze(n, maze, move_x, move_y, x_new, y_new, res):  
                return True  
            res[x_new][y_new] = 0  
    return False  
  
def solveMaze(maze):  
    n = len(maze)  
    res = [[0 for _ in range(n)] for _ in range(n)]  
    res[0][0] = 1  
  
    move_x = [-1, 1, 0, 0]  
    move_y = [0, 0, -1, 1]
```

```

if RatMaze(n, maze, move_x, move_y, 0, 0, res):
    for i in range(n):
        for j in range(n):
            print(res[i][j], end=' ')
    print()
else:
    print('Solution does not exist')

if __name__ == "__main__":
    n = int(input("Enter the number of rows and columns for the maze: "))
    maze = []
    print("Enter the maze elements:")
    for _ in range(n):
        row = list(map(int, input().split()))
        maze.append(row)
    print("Initial Stage :")
    for row in maze:
        for col in row:
            print(col, end=" ")
        print()
    print()
    print("Final Stage :")
    solveMaze(maze)

```

Output:

```
Enter the number of rows and columns for the maze: 4
Enter the maze elements:
```

```
1 0 0 0
```

```
1 1 0 1
```

```
0 1 0 0
```

```
1 1 1 1
```

```
Initial Stage :
```

```
1 0 0 0
```

```
1 1 0 1
```

```
0 1 0 0
```

```
1 1 1 1
```

```
Final Stage :
```

```
1 0 0 0
```

```
1 1 0 0
```

```
0 1 0 0
```

```
0 1 1 1
```

```
> □
```

Experiment 5

Implementation of Rat in a maze using Systematic Generate and Test

Code :

```
class Graph:
```

```
    def __init__(self, graph_dict=None, directed=True):  
        self.graph_dict = graph_dict or {}  
        self.directed = directed  
        if not directed:  
            self.make_undirected()
```

```
    def make_undirected(self):  
        for a in list(self.graph_dict.keys()):  
            for (b, dist) in self.graph_dict[a].items():  
                self.graph_dict.setdefault(b, {})[a] = dist
```

```
    def connect(self, A, B, distance=1):  
        self.graph_dict.setdefault(A, {})[B] = distance  
        if not self.directed:  
            self.graph_dict.setdefault(B, {})[A] = distance
```

```
    def get(self, a, b=None):  
        links = self.graph_dict.setdefault(a, {})  
        if b is None:  
            return links  
        else:  
            return links.get(b)
```

```
    def nodes(self):  
        s1 = set([k for k in self.graph_dict.keys()])
```

```

s2 = set([k2 for v in self.graph_dict.values() for k2, v2 in v.items()])
nodes = s1.union(s2)
return list(nodes)

class Node:

    def __init__(self, name:str, parent:str):
        self.name = name
        self.parent = parent
        self.g = 0 # Distance to the start node
        self.h = 0 # Distance to the goal node
        self.f = 0 # Total cost

    def __eq__(self, other):
        return self.name == other.name

    def __lt__(self, other):
        return self.f < other.f

    def __repr__(self):
        return '{0},{1}'.format(self.name, self.f)

def astar_search(graph, heuristics, start, end):
    open_nodes = []
    closed_nodes = []
    start_node = Node(start, None)
    goal_node = Node(end, None)
    open_nodes.append(start_node)
    while len(open_nodes) > 0:
        open_nodes.sort()
        current_node = open_nodes.pop(0)

```

```

closed_nodes.append(current_node)

if current_node == goal_node:
    path = []
    while current_node != start_node:
        path.append(current_node.name + ':' + str(current_node.g))
        current_node = current_node.parent
    path.append(start_node.name + ':' + str(start_node.g))

    return path[::-1]

neighbors = graph.get(current_node.name)

for key, value in neighbors.items():
    neighbor = Node(key, current_node)
    if neighbor in closed_nodes:
        continue
    neighbor.g = current_node.g + graph.get(current_node.name, neighbor.name)
    neighbor.h = heuristics.get(neighbor.name)
    neighbor.f = neighbor.g + neighbor.h
    if add_to_open(open_nodes, neighbor):
        open_nodes.append(neighbor)
return None

def add_to_open(open_nodes, neighbor):
    for node in open_nodes:
        if neighbor.name == node.name and neighbor.f > node.f:
            return False
    return True

def main():

```

```

graph = Graph()

while True:

    print("Add connections in Graph? Y/N (Enter your Choice)")

    choice = input()

    if choice == 'Y':

        graph_node_connection = input("Enter Connection node Like this Node1 Node2 EdgeWeight :")

        connection = graph_node_connection.split()

        if len(connection) == 3:

            graph.connect(connection[0], connection[1], int(connection[2]))

        else:

            print("Enter a Valid Connection between Edges")

    elif choice == 'N':

        break

    else:

        print("Enter a valid choice")

print("Is your graph Undirected? Y/N")

choice = input("Enter your choice: ")

if choice == 'Y':

    graph.make_undirected()

heuristics = {}

print("Enter Your Heuristics for Each Node")

nodes = int(input("Enter No. of Nodes in the Graph: "))

for i in range(0, nodes):

    heuristic_value = input("Enter your Heuristic value like this Node HeuristicValue: ")

    heuristic = heuristic_value.split()

    heuristics[heuristic[0]] = int(heuristic[1])

```

```

start_and_goal_node = input("Enter your Start Node and Goal Node (StartNode:GoalNode) - ")

start_and_goal = start_and_goal_node.split(sep=':')

path = astar_search(graph, heuristics, start_and_goal[0], start_and_goal[1])

print("\nBelow is the optimal path from the graph:")

for i in range(len(path)):

    if i == len(path) - 1:

        print(path[i])

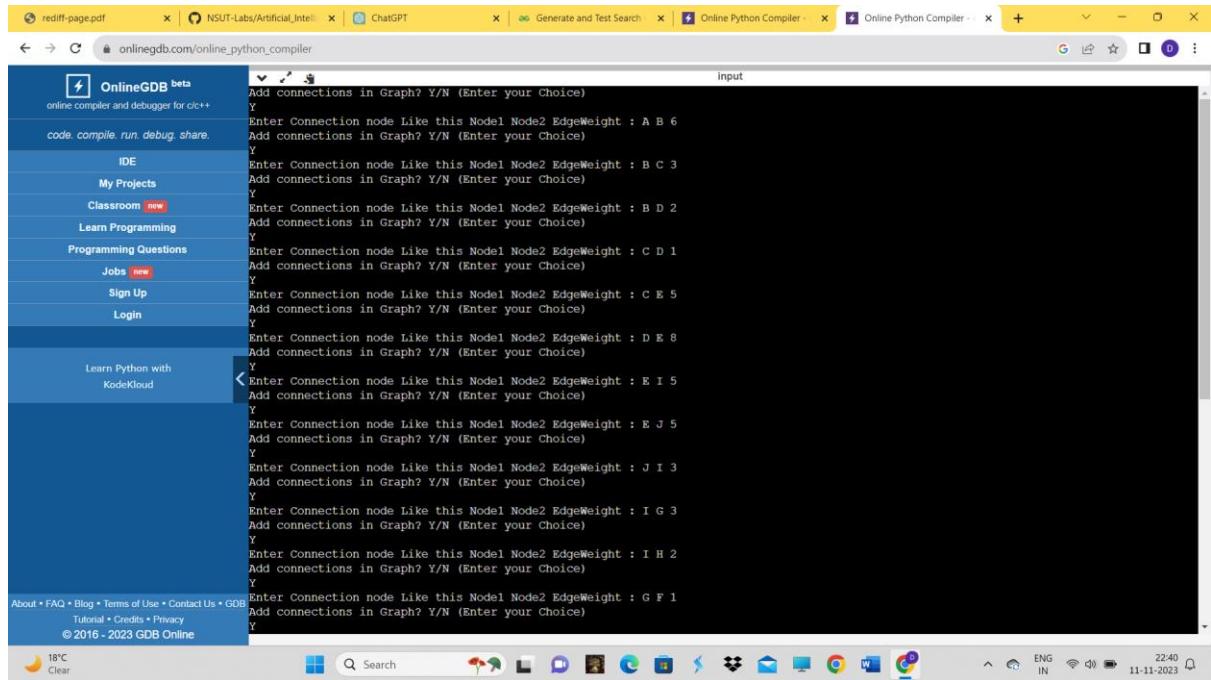
    else:

        print(path[i], end='-->')

if __name__ == "__main__":

    main()

```



reddiff-page.pdf NSUT-Labs/Artificial_Intel ChatGPT Generate and Test Search Online Python Compiler Online Python Compiler

onlinegdb.com/online_python_compiler

OnlineGDB beta
online compiler and debugger for c/c++

IDE
My Projects
Classroom new
Learn Programming
Programming Questions
Jobs now
Sign Up
Login

Learn Python with
KodeKloud

input

```
Enter Connection node Like this Node1 Node2 EdgeWeight : J I 3
Add connections in Graph? Y/N (Enter your Choice)
Y
Enter Connection node Like this Node1 Node2 EdgeWeight : I G 3
Add connections in Graph? Y/N (Enter your Choice)
Y
Enter Connection node Like this Node1 Node2 EdgeWeight : I H 2
Add connections in Graph? Y/N (Enter your Choice)
Y
Enter Connection node Like this Node1 Node2 EdgeWeight : G F 1
Add connections in Graph? Y/N (Enter your Choice)
Y
Enter Connection node Like this Node1 Node2 EdgeWeight : H F 7
Add connections in Graph? Y/N (Enter your Choice)
Y
Enter Connection node Like this Node1 Node2 EdgeWeight : F A 3
Add connections in Graph? Y/N (Enter your Choice)
N
Is your graph Undirected? Y/N
Enter your choice: Y
Enter Your Heuristics for Each Node
Enter No. of Nodes in the Graph: 10
Enter your Heuristic value like this Node HeuristicValue: A 10
Enter your Heuristic value like this Node HeuristicValue: B 8
Enter your Heuristic value like this Node HeuristicValue: C 5
Enter your Heuristic value like this Node HeuristicValue: D 7
Enter your Heuristic value like this Node HeuristicValue: E 3
Enter your Heuristic value like this Node HeuristicValue: J 0
Enter your Heuristic value like this Node HeuristicValue: G 5
Enter your Heuristic value like this Node HeuristicValue: I 1
Enter your Heuristic value like this Node HeuristicValue: H 3
Enter your Heuristic value like this Node HeuristicValue: F 6
Enter your Start Node and Goal Node (StartNode:GoalNode) - A:J
```

Below is the optimal path from the graph:
A: 0-->F: 3-->G: 4-->I: 7-->J: 10

About • FAQ • Blog • Terms of Use • Contact Us • GDB
Tutorial • Credits • Privacy
© 2016 - 2023 GDB Online

18°C Clear

Search

Windows Taskbar icons

ENG IN 22:40 11-11-2023

Experiment 6

Implementation of Minimize cost of Living using Hill Climbing

Code:

```
def cost_of_living(city_costs, current_city):
    return city_costs[current_city]

def get_neighbors(city_connections, current_city):
    return city_connections[current_city]

def hill_climbing(initial_city, city_costs, city_connections):
    current_city = initial_city
    current_cost = cost_of_living(city_costs, current_city)

    while True:
        neighbors = get_neighbors(city_connections, current_city)

        if not neighbors:
            break # No more neighbors to explore

        next_city = min(neighbors, key=lambda city: cost_of_living(city_costs, city))

        if cost_of_living(city_costs, next_city) >= current_cost:
            break

        current_city = next_city
        current_cost = cost_of_living(city_costs, current_city)

    return current_city, current_cost
```

```
def main():
    print("Enter the number of cities:")
    num_cities = int(input())

    city_costs = {}
    for i in range(num_cities):
        city = input(f"Enter the cost of living for city {chr(65 + i)}: ")
        city_costs[chr(65 + i)] = int(city)

    city_connections = {}
    for city in city_costs.keys():
        neighbors = input(f"Enter the neighbors of city {city} (separated by space): ").split()
        city_connections[city] = neighbors

    print("Enter the initial city (A, B, ..., Z):")
    initial_city = input().upper()

    if initial_city not in city_costs:
        print("Invalid city. Please enter a valid city.")
        return

    final_city, final_cost = hill_climbing(initial_city, city_costs, city_connections)

    print(f"The optimal city to minimize the cost of living is {final_city} with a cost of {final_cost}.")

if __name__ == "__main__":
    main()
```

The screenshot shows a Windows desktop environment with multiple windows open. The main window is the OnlineGDB IDE, which has a dark theme. On the left, there's a sidebar with links like 'IDE', 'My Projects', 'Classroom', 'Learn Programming', 'Programming Questions', 'Jobs', 'Sign Up', and 'Login'. The central area shows a Python script named 'main.py' with the following code:

```
27 def main():
28     print("Enter the number of cities:")
29     num_cities = int(input())
30
31     city_costs = {}
32     for i in range(num_cities):
33         city = input(f"Enter the cost of living for city {chr(65 + i)}: ")
34         city_costs[chr(65 + i)] = int(city)
35
36     city_connections = {}
37     for city in city_costs.keys():
38         neighbors = input(f"Enter the neighbors of city {city} (separated by space): ").split()
39         city_connections[city] = neighbors
```

Below the code editor, there's a terminal-like interface where the user has entered data:

```
Enter the number of cities:
7
Enter the cost of living for city A: 10
Enter the cost of living for city B: 5
Enter the cost of living for city C: 4
Enter the cost of living for city D: 12
Enter the cost of living for city E: 8
Enter the cost of living for city F: 9
Enter the cost of living for city G: 12
Enter the neighbors of city A (separated by space): C
Enter the neighbors of city B (separated by space): G
Enter the neighbors of city C (separated by space): D
Enter the neighbors of city D (separated by space): B
Enter the neighbors of city E (separated by space): E
Enter the neighbors of city F (separated by space): F
Enter the neighbors of city G (separated by space): A
Enter the initial city (A, B, ..., Z):
A
The optimal city to minimize the cost of living is C with a cost of 4.
```

At the bottom of the terminal, it says "...Program finished with exit code 0" and "Press ENTER to exit console."

The status bar at the bottom right shows the date and time: 11-11-2023 23:03. The taskbar icons include File Explorer, Edge browser, File Manager, Task View, Taskbar settings, and a weather icon showing 17°C.

Experiment 7

Implementation of Minimize cost of travel using Steepest Ascent Hill Climbing

Code:

```
def cost_of_travel(city_costs, current_city, next_city):
    return city_costs[current_city][next_city]

def get_neighbors(city_connections, current_city):
    return city_connections[current_city]

def steepest_ascent_hill_climbing(initial_city, city_costs, city_connections):
    current_city = initial_city
    current_cost = float('inf') # Initialize with infinity to ensure the first neighbor is chosen

    while True:
        neighbors = get_neighbors(city_connections, current_city)

        if not neighbors:
            break # No more neighbors to explore

        # Find the neighbor with the minimum cost
        next_city = min(neighbors, key=lambda city: cost_of_travel(city_costs, current_city, city))

        # Calculate the cost of traveling to the next city
        next_cost = cost_of_travel(city_costs, current_city, next_city)

        # Break if no improvement in cost
        if next_cost >= current_cost:
            break
```

```
current_city = next_city
current_cost = next_cost

return current_city, current_cost

def main():
    print("Enter the number of cities:")
    num_cities = int(input())

    city_costs = {}
    for i in range(num_cities):
        city_costs[chr(65 + i)] = {}
        for j in range(num_cities):
            cost = int(input(f"Enter the cost of travel from city {chr(65 + i)} to {chr(65 + j)}: "))
            city_costs[chr(65 + i)][chr(65 + j)] = cost

    city_connections = {}
    for city in city_costs.keys():
        neighbors = input(f"Enter the neighbors of city {city} (separated by space): ").split()
        city_connections[city] = neighbors

    print("Enter the initial city (A, B, ..., Z):")
    initial_city = input().upper()

    if initial_city not in city_costs:
        print("Invalid city. Please enter a valid city.")
        return

    final_city, final_cost = steepest_ascent_hill_climbing(initial_city, city_costs, city_connections)
```

```
print(f"The optimal city to minimize the cost of travel is {final_city} with a cost of {final_cost}.")
```

```
if __name__ == "__main__":
    main()
```

The screenshot shows a web-based IDE interface for OnlineGDB. The code in the editor is:

```
print(f"The optimal city to minimize the cost of travel is {final_city} with a cost of {final_cost}.")
```

```
if __name__ == "__main__":
    main()
```

The terminal window shows the execution of the program:

```
Enter the number of cities:
4
Enter the cost of travel from city A to A: 1
Enter the cost of travel from city A to B: 4
Enter the cost of travel from city A to C: 7
Enter the cost of travel from city A to D: 2
Enter the cost of travel from city B to A: 8
Enter the cost of travel from city B to B: 9
Enter the cost of travel from city B to C: 3
Enter the cost of travel from city B to D: 5
Enter the cost of travel from city C to A: 6
Enter the cost of travel from city C to B: 7
Enter the cost of travel from city C to C: 2
Enter the cost of travel from city C to D: 8
Enter the cost of travel from city D to A: 1
Enter the cost of travel from city D to B: 3
Enter the cost of travel from city D to C: 9
Enter the cost of travel from city D to D: 7
Enter the neighbors of city A (separated by space): D
Enter the neighbors of city B (separated by space): C
Enter the neighbors of city C (separated by space): A
Enter the neighbors of city D (separated by space): B
Enter the initial city (A, B, ..., Z):
D
The optimal city to minimize the cost of travel is B with a cost of 3.

...Program finished with exit code 0
Press ENTER to exit console.
```

The browser tab bar includes: rediff-page.pdf, NSUT-Labs/Artificial..., Rat Maze Solution, solve hill climbing us, Online Python Comp, WhatsApp, New Tab. The status bar at the bottom shows: 17°C Clear, Search, various icons, ENG IN, 23:35, 11-11-2023.

Experiment 8

Implementation of Best First Search using Shortest path

Code:

```
import heapq

def create_graph():
    graph = {}
    num_edges = int(input("Enter the number of edges: "))

    for _ in range(num_edges):
        start, end, weight = map(int, input("Enter edge (startnode, end node & weight): ").split())
        if start not in graph:
            graph[start] = []
        graph[start].append((end, weight))

    return graph

def heuristic(current, goal):
    # You can define your own heuristic function based on the problem requirements.
    return abs(current - goal)

def best_first_search(graph, start, goal):
    visited = set()
    priority_queue = [(0, start)] # (heuristic value, node)

    while priority_queue:
        cost, current_node = heapq.heappop(priority_queue)
        if current_node in visited:
            continue

        visited.add(current_node)
        if current_node == goal:
            break

        for neighbor, weight in graph[current_node]:
            new_cost = cost + weight
            heapq.heappush(priority_queue, (new_cost, neighbor))

    return visited
```

```
print(f"Visit Node: {current_node}")

visited.add(current_node)

if current_node == goal:
    print("Goal reached!")
    break

neighbors = graph.get(current_node, [])

for neighbor, weight in neighbors:
    if neighbor not in visited:
        heuristic_value = heuristic(neighbor, goal)
        heapq.heappush(priority_queue, (heuristic_value, neighbor))

# Example Usage
graph = create_graph()
start_node = int(input("Enter the start node: "))
goal_node = int(input("Enter the goal node: "))

best_first_search(graph, start_node, goal_node)
```

ChatGPT | Online Python Compiler - online | Online Python Compiler - online | +

onlinegdb.com/online_python_compiler

OnlineGDB beta
online compiler and debugger for c/c++
code, compile, run, debug, share.
IDE
My Projects
Classroom new
Learn Programming
Programming Questions
Jobs new
Sign Up
Login

main.py

```
1 import heapq
2
3 def create_graph():
4     graph = {}
5     num_edges = int(input("Enter the number of edges: "))
6
7     for _ in range(num_edges):
8         start, end, weight = map(int, input("Enter edge (startnode, end node & weight): ").split())
9         if start not in graph:
10             graph[start] = []
11         graph[start].append((end, weight))
12
13     return graph
14
15 def heuristic(current, goal):
16     # You can define your own heuristic function based on the problem requirements.
17     return abs(current - goal)
```

input

```
Enter the number of edges: 7
Enter edge (startnode, end node & weight): 1 2 5
Enter edge (startnode, end node & weight): 2 3 6
Enter edge (startnode, end node & weight): 4 6 10
Enter edge (startnode, end node & weight): 3 4 6
Enter edge (startnode, end node & weight): 4 5 2
Enter edge (startnode, end node & weight): 5 6 7
Enter edge (startnode, end node & weight): 6 1 5
Enter the start node: 1
Enter the goal node: 6
Visit Node: 1
Visit Node: 2
Visit Node: 3
Visit Node: 4
Visit Node: 6
Goal reached!
```

About • FAQ • Blog • Terms of Use • Contact Us • GDB
Tutorial • Credits • Privacy
© 2016 - 2023 GDB Online

...Program finished with exit code 0

This screenshot shows a web-based Python compiler interface. The main window displays a Python script named 'main.py' which defines a function to create a graph from a series of edges provided by the user. It also includes a heuristic function for a search algorithm. Below the code editor is a terminal window showing the execution of the script. The user enters 7 edges: (1, 2, 5), (2, 3, 6), (4, 6, 10), (3, 4, 6), (4, 5, 2), (5, 6, 7), and (6, 1, 5). The terminal then prompts for the start and goal nodes, both of which are set to 1. The script then performs a search, visiting nodes 1, 2, 3, 4, and 6 in sequence, and finally reaches the goal node, outputting "Goal reached!". The status bar at the bottom right indicates the date and time as 12-11-2023 17:05.

Experiment 9

Implementation of A* Algorithm using Shortest path

Code:

```
import heapq
```

```
class Node:
```

```
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.g = float('inf') # Initial cost from start node
        self.h = 0 # Heuristic cost to goal node
        self.parent = None # Parent node in the path
```

```
    def __lt__(self, other):
        return (self.g + self.h) < (other.g + other.h)
```

```
def __eq__(self, other):
    return self.x == other.x and self.y == other.y
```

```
def heuristic(node, goal):
```

```
    # Manhattan distance heuristic
    return abs(node.x - goal.x) + abs(node.y - goal.y)
```

```
def get_neighbors(node, grid):
```

```
    neighbors = []
    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)] # Possible movement directions (up, right, down, left)
```

```
    for dx, dy in directions:
```

```
        nx, ny = node.x + dx, node.y + dy
```

```
    if 0 <= nx < len(grid) and 0 <= ny < len(grid[0]) and grid[nx][ny] != 1:  
        neighbors.append(Node(nx, ny))  
  
    return neighbors
```

```
def a_star(start, goal, grid):  
    open_set = [start]  
    closed_set = set()  
  
    while open_set:  
        current_node = heapq.heappop(open_set)  
        if current_node == goal:  
            path = []  
            while current_node:  
                path.insert(0, (current_node.x, current_node.y))  
                current_node = current_node.parent  
            return path  
  
        closed_set.add((current_node.x, current_node.y))  
  
        for neighbor in get_neighbors(current_node, grid):  
            if (neighbor.x, neighbor.y) in closed_set:  
                continue  
  
            tentative_g = current_node.g + 1 # Assuming uniform cost for simplicity  
  
            if tentative_g < neighbor.g:  
                neighbor.g = tentative_g  
                neighbor.h = heuristic(neighbor, goal)  
                neighbor.parent = current_node
```

```

        if neighbor not in open_set:
            heapq.heappush(open_set, neighbor)

    return None # No path found

def print_grid_with_path(grid, path):
    for i in range(len(grid)):
        for j in range(len(grid[0])):
            if (i, j) in path:
                print("P", end=" ")
            elif grid[i][j] == 1:
                print("X", end=" ")
            else:
                print(".", end=" ")
        print()

def get_user_input():
    rows = int(input("Enter the number of rows in the grid: "))
    cols = int(input("Enter the number of columns in the grid: "))

    grid = [[0] * cols for _ in range(rows)]

    # Take obstacle input
    num_obstacles = int(input("Enter the number of obstacles: "))
    for _ in range(num_obstacles):
        obstacle_row = int(input("Enter obstacle row: "))
        obstacle_col = int(input("Enter obstacle column: "))
        grid[obstacle_row][obstacle_col] = 1

    # Take start and goal input
    start_row = int(input("Enter the starting row: "))

```

```
start_col = int(input("Enter the starting column: "))

start_node = Node(start_row, start_col)

goal_row = int(input("Enter the goal row: "))

goal_col = int(input("Enter the goal column: "))

goal_node = Node(goal_row, goal_col)

return grid, start_node, goal_node

def main():

    grid, start_node, goal_node = get_user_input()

    path = a_star(start_node, goal_node, grid)

    if path:

        print("Shortest Path:")

        print_grid_with_path(grid, path)

    else:

        print("No path found!")

if __name__ == "__main__":

    main()
```

The screenshot shows a web-based IDE interface for Python 3. The main window displays a file named 'main.py' containing Python code for an A* algorithm implementation. The code prompts the user for grid dimensions, obstacle locations, and start and goal coordinates. It then executes the program, which outputs the user's input and the algorithm's results, including a message indicating 'No path found!'. The interface includes a toolbar with 'Run', 'Debug', 'Stop', 'Share', 'Save', and 'Beautify' buttons, and a sidebar with various links related to programming and learning.

```
WhatsApp | A* Algorithm Implementation | Online Python Compiler - onlinegdb.com
language: Python 3 | settings

OnlineGDB beta
online compiler and debugger for c/c+++
code. compile. run. debug. share.

IDE
My Projects
Classroom new
Learn Programming
Programming Questions
Jobs new
Sign Up
Login

Learn Python with
KodeKloud

main.py
84     obstacle_col = int(input("Enter obstacle column: "))
85     grid[obstacle_row][obstacle_col] = 1
86
87     # Take start and goal input
88     start_row = int(input("Enter the starting row: "))
89     start_col = int(input("Enter the starting column: "))
90     start_node = Node(start_row, start_col)
91
92     goal_row = int(input("Enter the goal row: "))
93     goal_col = int(input("Enter the goal column: "))

Enter the number of rows in the grid: 5
Enter the number of columns in the grid: 5
Enter the number of obstacles: 6
Enter obstacle row: 1
Enter obstacle column: 1
Enter obstacle row: 1
Enter obstacle column: 2
Enter obstacle row: 2
Enter obstacle column: 3
Enter obstacle row: 3
Enter obstacle column: 3
Enter obstacle row: 1
Enter obstacle column: 2
Enter the starting row: 0
Enter the starting column: 0
Enter the goal row: 5
Enter the goal column: 5
No path found!

...Program finished with exit code 0
Press ENTER to exit console.

About • FAQ • Blog • Terms of Use • Contact Us • GDB
Tutorial • Credits • Privacy
Connecting...
25°C Haze
Search
ENG IN 12-11-2023
```

Experiment 10

Implementation of AO* Algorithm using Shortest path

Code :

```
import heapq

class Node:
    def __init__(self, state, parent=None, cost=0, heuristic=0):
        self.state = state
        self.parent = parent
        self.cost = cost
        self.heuristic = heuristic

    def __lt__(self, other):
        return (self.cost + self.heuristic) < (other.cost +
other.heuristic)

def astar_search(start, goal, neighbors_func, heuristic_func):
    open_set = [Node(start, None, 0, heuristic_func(start))]
    closed_set = set()

    while open_set:
        current_node = heapq.heappop(open_set)

        if current_node.state == goal:
            path = []
            while current_node:
                path.insert(0, current_node.state)
                current_node = current_node.parent
            return path

        closed_set.add(current_node.state)

        for neighbor in neighbors_func(current_node.state):
            if neighbor in closed_set:
                continue

            cost = current_node.cost + 1 # Assuming each step has
a cost of 1
            heuristic = heuristic_func(neighbor)
```

```

new_node = Node(neighbor, current_node, cost, heuristic)

if new_node not in open_set:
    heapq.heappush(open_set, new_node)

return None # No path found

# Example usage:
def neighbors(state):
    x, y = state
    return [(x+1, y), (x-1, y), (x, y+1), (x, y-1)]

def heuristic(state):
    goal = (5, 5) # Change this to your goal state
    return abs(state[0] - goal[0]) + abs(state[1] - goal[1])

start_state = (0, 0)
goal_state = (5, 5)
path = astar_search(start_state, goal_state, neighbors, heuristic)

if path:
    print("Shortest path:", path)
else:
    print("No path found.")

```

Output:

The screenshot shows the OnlineGDB IDE interface. On the left, there's a sidebar with links like 'OnlineGDB beta', 'code. compile. run. debug. share.', 'IDE', 'My Projects', 'Classroom new', 'Learn Programming', 'Programming Questions', 'Jobs new', 'Sign Up', and 'Login'. Below that is a link to 'Learn Python with KodeKloud'. At the bottom of the sidebar are links for 'About', 'FAQ', 'Blog', 'Terms of Use', 'Contact Us', 'GDB Tutorial', 'Credits', and 'Privacy', along with a copyright notice '© 2016 - 2023 GDB Online'.

The main area has tabs for 'main.py' and 'ChatGPT'. The 'main.py' tab contains the following Python code:

```
cost = current_node.cost + 1 # Assuming each step has a cost of 1
heuristic = heuristic_func(neighbor)
new_node = Node(neighbor, current_node, cost, heuristic)

if new_node not in open_set:
    heapq.heappush(open_set, new_node)

return None # No path found

# Example usage:
def neighbors(state):
    x, y = state
    return [(x+1, y), (x-1, y), (x, y+1), (x, y-1)]

def heuristic(state):
    goal = (5, 5) # Change this to your goal state
    return abs(state[0] - goal[0]) + abs(state[1] - goal[1])

start_state = (0, 0)
goal_state = (5, 5)
path = astar_search(start_state, goal_state, neighbors, heuristic)

if path:
    print("Shortest path:", path)
else:
    print("No path found.")


```

The output window shows the results of running the code:

```
Shortest path: [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (2, 3), (3, 3), (4, 3), (4, 4), (5, 4), (5, 5)]  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Experiment 11

Implementation of NLP using Sentiment Analysis

Code :

```
def analyze_sentiment(text):
    """
        Analyze sentiment of the given text using a simple rule-based
        approach.

    Parameters:
    - text (str): The input text for sentiment analysis.

    Returns:
    - str: The sentiment label ('Positive', 'Negative', 'Neutral').
    """
    # You can customize these words/phrases based on your
    # specific needs
    positive_keywords = ["good", "great", "excellent", "positive"]
    negative_keywords = ["bad", "poor", "negative"]

    # Tokenize the input text
    words = text.lower().split()

    # Check for positive and negative keywords
    positive_count = sum(word in positive_keywords for word in
                          words)
    negative_count = sum(word in negative_keywords for word in
                          words)

    # Define a threshold for sentiment classification
    sentiment_threshold = 2 # Adjust as needed

    if positive_count >= sentiment_threshold:
        return "Positive"
    elif negative_count >= sentiment_threshold:
        return "Negative"
    else:
        return "Neutral"

def get_user_input():
```

Get user input for sentiment analysis.

Returns:

- str: User input text.

```
user_input = input("Enter a sentence or paragraph for  
sentiment analysis: ")  
return user_input
```

```
def main():
```

```
    # Get user input
```

```
    user_text = get_user_input()
```

```
    # Perform sentiment analysis
```

```
    sentiment_result = analyze_sentiment(user_text)
```

```
    # Display the result
```

```
    print(f"Sentiment: {sentiment_result}")
```

```
if __name__ == "__main__":
```

```
    main()
```

Output:

The screenshot shows the OnlineGDB IDE interface. On the left is a sidebar with links like ChatGPT, OnlineGDB beta (selected), code. compile. run. debug. share., IDE, My Projects, Classroom, Learn Programming, Programming Questions, Jobs, Sign Up, and Login. Below that is a section for Learn Python with KodeKloud. The main workspace has tabs for ChatGPT and main.py. The main.py tab contains the provided Python code. The code defines a get_user_input function that returns a sentiment based on user input. It then defines a main function that gets user input, performs sentiment analysis, and prints the result. A conditional statement checks if the script is run directly (if __name__ == "__main__") and calls the main function. The bottom part of the interface shows the terminal window with the code and its execution results. The terminal output shows the user entering "I am not feeling well" and the program outputting "Sentiment: Neutral". It also shows the program finished with exit code 0 and a prompt to press ENTER to exit the console.

```
28     return "Negative"  
29 - else:  
30     return "Neutral"  
31  
32 def get_user_input():  
33     """  
34     Get user input for sentiment analysis.  
35     Returns:  
36     - str: User input text.  
37     """  
38     user_input = input("Enter a sentence or paragraph for sentiment analysis: ")  
39     return user_input  
40  
41  
42 def main():  
43     # Get user input  
44     user_text = get_user_input()  
45  
46     # Perform sentiment analysis  
47     sentiment_result = analyze_sentiment(user_text)  
48  
49     # Display the result  
50     print(f"Sentiment: {sentiment_result}")  
51  
52 if __name__ == "__main__":  
53     main()  
54  
Enter a sentence or paragraph for sentiment analysis: I am not feeling well  
Sentiment: Neutral  
...Program finished with exit code 0  
Press ENTER to exit console.
```


Experiment 12

Implementation of Tic Tac Toe using Minimax with α - β pruning

Code:

```
import math

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

def is_winner(board, player):
    # Check rows, columns, and diagonals for a win
    for i in range(3):
        if all(board[i][j] == player for j in range(3)) or
           all(board[j][i] == player for j in range(3)):
            return True
        if all(board[i][i] == player for i in range(3)) or
           all(board[i][2 - i] == player for i in range(3)):
            return True
    return False

def is_board_full(board):
    return all(board[i][j] != ' ' for i in range(3) for j in
range(3))

def evaluate_board(board):
    if is_winner(board, 'X'):
        return 1
    elif is_winner(board, 'O'):
        return -1
    elif is_board_full(board):
        return 0
    else:
        return None
```

```
def minimax(board, depth, maximizing_player, alpha,
beta):
    score = evaluate_board(board)

    if score is not None:
        return score

    if maximizing_player:
        max_eval = -math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
                    board[i][j] = 'X'
                    eval = minimax(board, depth + 1, False,
alpha, beta)
                    board[i][j] = ' ' # Undo the move
                    max_eval = max(max_eval, eval)
                    alpha = max(alpha, eval)
                if beta <= alpha:
                    break
        return max_eval
    else:
        min_eval = math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
                    board[i][j] = 'O'
                    eval = minimax(board, depth + 1, True,
alpha, beta)
                    board[i][j] = ' ' # Undo the move
                    min_eval = min(min_eval, eval)
                    beta = min(beta, eval)
                if beta <= alpha:
                    break
        return min_eval
```

```
def find_best_move(board):
    best_val = -math.inf
    best_move = (-1, -1)

    for i in range(3):
        for j in range(3):
            if board[i][j] == ' ':
                board[i][j] = 'X'
                move_val = minimax(board, 0, False, -math.inf,
math.inf)
                board[i][j] = ' ' # Undo the move

                if move_val > best_val:
                    best_move = (i, j)
                    best_val = move_val

    return best_move

def play_tic_tac_toe():
    board = [[' ' for _ in range(3)] for _ in range(3)]
    player_turn = True

    while True:
        print_board(board)

        if player_turn:
            row = int(input("Enter row (0, 1, or 2): "))
            col = int(input("Enter column (0, 1, or 2): "))
            if board[row][col] == ' ':
                board[row][col] = 'O'
                player_turn = False
            else:
                print("Invalid move. Try again.")
        else:
            print("Computer's turn:")
            move = find_best_move(board)
            board[move[0]][move[1]] = 'X'
```

```

player_turn = True

winner = evaluate_board(board)
if winner is not None:
    print_board(board)
    if winner == 1:
        print("Player wins!")
    elif winner == -1:
        print("Computer wins!")
    else:
        print("It's a tie!")
    break

if __name__ == "__main__":
    play_tic_tac_toe()

```

Output :

The screenshot shows a web-based IDE interface for OnlineGDB. The code in the editor is:

```

row = int(input("Enter row (0, 1, or 2): "))
col = int(input("Enter column (0, 1, or 2): "))
if board[row][col] == ' ':
    board[row][col] = 'O'

```

The terminal window shows the following interaction:

```

Enter row (0, 1, or 2): 1
Enter column (0, 1, or 2): 2
O
Computer's turn:
X
O
O
Enter row (0, 1, or 2): 2
Enter column (0, 1, or 2): 1
X
O
O
Computer's turn:
X X
O
O O
Player wins!
...Program finished with exit code 0
Press ENTER to exit console.

```

The interface includes a sidebar with various programming-related links and a footer with copyright information.