Assignment No. 1

DFS BFS

```
from collections import defaultdict, deque

class Graph:
    def __init__(self):
        self.graph = defaultdict(list)

    def add_node(self, nodeVal):
        if nodeVal not in self.graph:
            self.graph[nodeVal] = []

    def add_edge(self, u, v):
        self.graph[u].append(v)
        self.graph[v].append(u)

    def dfs(self, start, end, path=[], level=0):
        path = path + [start]
        print(f"Node {start} at level {level}")

        if start == end:
            return path

        for neighbour in self.graph[start]:
            if neighbour not in path:
                new_path = self.dfs(neighbour, end, path, level + 1)
                if new_path:
                    return new_path

        return None

    def bfs(self, start, end):
        queue = deque([(start, [start])])
        levels = {start: 0}

        while queue:
            current, path = queue.popleft()
            level = levels[current]
            print(f"Node {current} at level {level}")

            for neighbour in self.graph[current]:
                if neighbour not in path:
                    new_path = path + [neighbour]
                    levels[neighbour] = level + 1
                    if neighbour == end:
                        return new_path, levels
                    else:
                        queue.append((neighbour, new_path))
        return None, None

    def display_graph(self):
```

```python
        print("Current State of Graph: ")
        for node, neighbours in self.graph.items():
            print(f"\nNode {node} connects to: {neighbours}")


# Driver Code
network = Graph()

nodeNo = int(input("Enter the Number of Nodes: "))

for _ in range(nodeNo):
    nodeVal = int(input("Enter the Value of Node: "))
    network.add_node(nodeVal)

edgeNo = int(input("Enter the Number of Edges: "))

for _ in range(edgeNo):
    u, v = map(int, input("Enter the Edge (u v): ").split())
    network.add_edge(u, v)

start = int(input("Enter the Start Node: "))
end = int(input("Enter the End Node: "))

if start in network.graph and end in network.graph:
    dfs_path = network.dfs(start, end)
    print("DFS Path: ", dfs_path)

if start in network.graph and end in network.graph:
    bfs_path, levels = network.bfs(start, end)
    print("BFS Path: ", bfs_path)

network.display_graph()
```

Assignment No. 2

Implement A Star Algorithm for any game search problem.

```python
import copy
INFINITY: int = 1e10

class Puzzle:
  def __init__(self, elements: list[list[int]])-> None:
    self.board = elements #board: Represents the arrangement of elements (numbers from 1 to 8) in the puzzle.
    self.n_dims = len(elements) #n_dims: Represents the dimensions of the puzzle (number of rows or columns).
    for i in range(self.n_dims):
      for j in range(self.n_dims):
        if self.board[i][j] == 0:
```

```
                self.pos = (i, j) #pos: Represents the position of the empty cell (denoted by '0') in the puzzle.
                break

    def print(self):
        for row in self.board:
            print(" ".join(map(str, row)))

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

    def __hash__(self):
        return hash(str(self.board))


def move(puzzle: Puzzle)-> list[Puzzle]:
```
#This function generates all possible states (nodes) that can be reached from a given puzzle state by moving the empty cell in all four directions (up, down, left, right).
#It returns a list of new puzzle states (nodes) generated by the possible moves.
```
    x, y = puzzle.pos[0], puzzle.pos[1]
    dim = puzzle.n_dims
    possible_pos = [
        (x + 1, y),
        (x, y + 1),
        (x- 1, y),
        (x, y- 1),
    ]
    new_states = []
    for pos_x, pos_y in possible_pos:
        if 0 <= pos_x < dim and 0 <= pos_y < dim:
            new_elements = copy.deepcopy(puzzle.board)
            new_elements[pos_x][pos_y], new_elements[x][y] = (
                new_elements[x][y],
                new_elements[pos_x][pos_y],
            )
            new_states.append(Puzzle(new_elements))
    return new_states


def heuristic(init_puzzle: Puzzle, goal_puzzle: Puzzle)-> int:
```
#This function calculates a heuristic value for the given puzzle state, representing the estimated cost from the initial puzzle state to the goal puzzle state.
#The heuristic used here is the count of misplaced elements compared to the goal puzzle state.
```
    return sum(
        1 for i in range(init_puzzle.n_dims)
        for j in range(init_puzzle.n_dims)
        if init_puzzle.board[i][j] != goal_puzzle.board[i][j]
    )
```

```
def is_goal(curr_puzzle: Puzzle, goal_puzzle: Puzzle)-> bool:
#This function checks whether the current puzzle state matches the goal puzzle state, indicating whether the goal has been reached.
    return curr_puzzle == goal_puzzle


init_puzzle = Puzzle([[1, 2, 3], [0, 4, 6], [7, 5, 8]])
print("Initial state:")
init_puzzle.print()

goal_puzzle = Puzzle([[1, 2, 3], [4, 5, 6], [7, 8, 0]])
print("Goal state:")
goal_puzzle.print()

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

open_set = [init_puzzle]
visited = set()
visited.add(init_puzzle)

num_step = 1
while open_set:
    print(f">> Step {num_step}: ")

    current = min(open_set, key=lambda x: heuristic(x, goal_puzzle))
    current.print()

    num_step += 1

    if is_goal(current, goal_puzzle):
        print("Done")
        break

    print(">> Possible states: ")
    open_set.remove(current)
    for neighbor in move(current):
        if neighbor not in visited:
            visited.add(neighbor)
            open_set.append(neighbor)
            neighbor.print()
            print(f"h(n) for above state is {heuristic(neighbor, goal_puzzle)}")
            print(f"g(n) for above state is {num_step}")
            print(f"f(n) for above state is {num_step + heuristic(neighbor, goal_puzzle)}")

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

Assignment No. 3 and Assignment No. 4

Implement Greedy Search algorithm on some application for :

I. Minimum Spanning Tree OR

II. Single–Source Shortest Path Problem

Implement Greedy Search algorithm on some application for :

I. Job Scheduling Problem OR

II. Prim's Minimal Spanning Tree algorithm

```
#include<bits/stdc++.h> // Include necessary libraries

#define rep(i,a,b)  for(int i = a; i < b ; i++) // Define macro for loop iteration

using namespace std;

int minindex(vector<int> key, vector<bool> visited)
{
   int min = INT_MAX; // Initialize minimum key value
   int ind =-1; // Initialize index of minimum key value

   // Loop through all vertices
   rep(i,0,key.size())
   {
      if(key[i]<min && !visited[i]) // If key is smaller than current minimum and vertex is not visited
      {
         min = key[i]; // Update minimum key value
         ind = i; // Update index of minimum key value
      }
   }
   return ind; // Return index of minimum key value
}

void printMST(vector<int> parent, vector<vector<int>> graph,int n)
{
   cout << "Edge \tWeight\n"; // Print column headers
   for (int i = 1; i < n; i++) // Loop through all vertices except the first one
      cout << parent[i] << "- " << i << " \t"<< graph[i][parent[i]] << " \n"; // Print each edge and its weight



}

void minimumSpanningTree(vector<vector<int>> graph, int n)
{
   vector<int> parent(n,-1); // Array to store parent of each vertex in MST
   vector<int> key(n, INT_MAX); // Array to store key values used to determine MST
   vector<bool> visited(n,false); // Array to track visited vertices
```

```
    key[0] = 0; // Initialize key value of first vertex to 0

    rep(i,0,n-1) // Loop through all vertices except the last one
    {
        int u = minindex(key, visited); // Find vertex with minimum key value among unvisited vertices
        visited[u] = true; // Mark vertex as visited
        rep(v,0,n) // Loop through all vertices
        {
            if(graph[u][v] && !visited[v] && graph[u][v] < key[v] ) // If there's an edge between u and v, v is unvisited, and weight of edge is less than key of v
            {
                parent[v] = u; // Set u as parent of v
                key[v] = graph[u][v]; // Update key of v with weight of edge (u, v)
            }
        }
    }
    printMST(parent, graph, n); // Print the MST
}

int main()
{
    int v;
    cout << "enter the number of vertices: ";
    cin >> v; // Input number of vertices

    vector<vector<int>> graph(v); // Initialize adjacency matrix for the graph

    rep(i, 0, v)
    {
        graph[i].resize(v); // Resize each row of the matrix to v
    }

    int n;
    cout << "enter the number of edges: ";
    cin >> n; // Input number of edges

    rep(i, 0, n)
    {
        int u, v, wt;
        cin >> u >> v >> wt; // Input edge (u, v) and its weight

        graph[u][v] = graph[v][u]=wt; // Update adjacency matrix with edge weight
    }

    minimumSpanningTree(graph, v); // Find and print the MST

    return 0;
}
```

Assignment No. 5

Implement Greedy Search algorithm on some application for :

I. Kruskal’s Minimum Spanning Tree algorithm OR

II. Dijkstra’s Minimum Spanning Tree algorithm

```
#include <bits/stdc++.h>
#define rep(i,a,b)     for(int i = a; i < b; i ++)
using namespace std;

int mindist(vector<int> distance, vector<bool> visited)
{
   int min = INT_MAX;int ind =-1;
   rep(i,0,distance.size())
   {
      if(distance[i]!=INT_MAX && distance[i] < min && visited[i] == false)
      {
         ind = i;
         min = distance[i];
      }
   }
   return ind;
}

void printSolution(vector<int> dist)
{
   cout << "Vertex \t Distance from Source" << endl;
   for (int i = 0; i < dist.size(); i++)
      cout << i << " \t\t" << dist[i] << endl;
}

void djikstras(vector<vector<int>> graph, int src)
{
   vector<int> distance(graph.size(), INT_MAX);
   vector<bool> visited(graph.size(), false);

   distance[src] = 0;

   rep(i,1,graph.size()-1)
   {
      int u = mindist(distance, visited);
      visited[u] = true;

      rep(v,0,graph.size())
      {
```

```
            if(graph[u][v] && !visited[v] && distance[u]!=INT_MAX && graph[u][v] + distance[u] < distance[v])
            {
               distance[v] = graph[u][v] + distance[u] ;
            }

         }
   }
   printSolution(distance);

}

int main()
{
   int v;
   cout << "enter the number of vertices: "; cin >> v;
   vector<vector<int>> graph(v);

   rep(i,0,v)
   {
      graph[i].resize(v);
   }
   cout << "enter the number of edges present inside the graph: "; int n; cin >> n;
   rep(i,0,n)
   {
      int x,y,wt;
      cin >> x >>y >> wt;
      graph[x][y] = wt;
   }

   djikstras(graph,0);

}
```

Assignment No. 6

Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem.

```
// C++ program to solve N Queen Problem using backtracking
#include <bits/stdc++.h>
#define rep(i,a,b)      for(int i = a ; i < b ; i++)
using namespace std;

bool issafe(int** board, int x, int y, int n)
{
   rep(i,0,x)
```

```
    {
        if(board[i][y] == 1)
            return false;
    }


    int row = x; int col = y;

    while(row >= 0 && col >= 0)
    {
        if(board[row][col] == 1)
        {
            return false;
        }
        row-- ;
        col--;
    }

    row = x;
    col = y;
    while(row >= 0 && col < n)
    {
        if(board[row][col] == 1)
        {
            return false;
        }
        row--;
        col ++;
    }
    return true;
}

bool nqueen(int** arr, int x, int n)
{
    if(x >= n)
        return true;

    for(int col = 0; col < n; col ++)
    {
        if(issafe(arr, x, col, n))
        {
            arr[x][col] = 1;
            if(nqueen(arr, x+1, n))
            {
                return true;
            }
            arr[x][col] = 0 ;//backtracking
        }
    }
```

```
    return false;
}

int main()
{
    int n;
    cin >> n;

    int** board = new int*[n];
    rep(i,0,n)
    {
        board[i] = new int[n];
        rep(j,0,n)
        {
            board[i][j] = 0;
        }
    }

   if(nqueen(board, 0, n))
    rep(i,0,n)
    {
        rep(j,0,n)
        {
            cout << board[i][j] << " ";
        }cout << endl;
    }

}
```

Assignment No. 7

Chatbot

```
# Building a chatbot with hiearchical text-search

search_tree = {
    "admission": {
        "cutoff": "The cuttoff for is > 99 percentile for MHTCET. For more info, refer https://pict.edu/cutoff-FE/",
        "fee": "The fees is 93K for open-category candidates. For more info, refer https://pict.edu/fee-structure/",
        "branch": "Branches are COMP, IT, EnTC, ECE, AIDS",
    },
    "hostel": "The campus has a girls and boys hostel in-premises. For more info, refer https://pict.edu/hostel/",
    "placement": {
        "statistic": "For this year, 706 students are placed. For detailed statistics, refer https://pict.edu/placement/index.php#statistics",
        "activit": "https://pict.edu/placement/index.php#trainingactivities",
    },
```

```
    "library": "PICT central library being information hub provides access to full text, digital and printed resources to support the scholarly and informational needs of the students, faculty, researchers, and other users.",
    "timetable": "https://pict.edu/time_table_syllabus/",
    "syllabus": "https://pict.edu/time_table_syllabus/",
}

def search(query: str, subtree):
    if type(subtree) == str:
        print(f"Reply: {subtree}")
        return
    children = list(subtree.keys())
    for child in children:
        if child in query:
            search(query, subtree[child])
            break
    else:
        print("Could not understand context, available options are: ")
        print(children)

while True:
    question = input("Enter your query: ")
    search(question, search_tree)
```