Green University of Bangladesh
Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering
Semester: (Spring, Year:2024), B.Sc. in CSE (Day)

Lab Report NO #02
Course Title: Artificial Intelligence Lab
Course Code: CSE 316 **Section:** 213 D7

Lab Experiment Name: Implement Iterative Deepening depth-first search(IDDFS),Graph Coloring Algorithm and Solve N-Queen Problem Using Backtracking Algorithm.

Student Details

Name		ID
1.	Md.Manzurul Alam	202902003

Lab Date : 10-03-2024
Submission Date : 21-05-2024
Course Teacher's Name : Sakhaouth Hossan

Lab Report Status

Marks: …… **Signature:.....**
Comments:..... **Date:.....**

1. TITLE OF THE LAB EXPERIMENT

This lab report covers the implementation and analysis of three essential algorithms: Iterative Deepening Depth-First Search (IDDFS), the Graph Coloring Algorithm, and the N-Queen Problem using Backtracking.

IDDFS merges the benefits of depth-first search (DFS) and breadth-first search (BFS), making it ideal for situations with unknown solution depths. We will explore IDDFS's properties, its advantages over BFS and DFS, and its implementation.

Constraint Satisfaction Problems (CSPs) involve variables that must satisfy specific constraints. By modeling problems as CSPs, we can apply systematic strategies to find solutions efficiently. This report examines the Graph Coloring and N-Queen problems as CSPs, discussing different solution strategies and their implications.

The Graph Coloring section aims to assign colors to vertices such that no two adjacent vertices share the same color, highlighting the total number of possible solutions. The N-Queen problem involves placing N queens on an N×N chessboard without threats, using backtracking to find and display all valid configurations.

This report demonstrates the practical applications and importance of these algorithms in solving complex problems efficiently, enhancing our understanding of these fundamental computational techniques.

2. OBJECTIVES

Understand and Implement Iterative Deepening Depth-First Search (IDDFS):

- Learn the fundamental properties of IDDFS.
- Implement IDDFS and compare its efficiency and advantages over BFS and DFS.

Explore Constraint Satisfaction Problems (CSPs):

- Define and understand the concept of CSPs.
- Learn how to model problems as CSPs.

Solve the Graph Coloring Problem:

- Implement the Graph Coloring Algorithm.
- Determine and discuss the total number of valid solutions for the graph coloring problem.
- Print all possible solutions for the problem.

Solve the N-Queen Problem Using Backtracking:

- Implement the Backtracking Algorithm for the N-Queen problem.
- Calculate and discuss the total number of solutions for different values of N.
- Print all valid configurations of the N-Queen problem.

Compare and Analyze Different Algorithms:

- Compare the performance and applications of IDDFS, Graph Coloring, and Backtracking algorithms.
- Discuss the practical significance and efficiency of these algorithms in solving complex problems.

3. PROCEDURE

Procedure for IDDFS Topological Sorting

1. Define the Graph Class:
   - Initialize an adjacency list to represent the graph.
   - Implement a method to add edges between vertices.

2. Implement Depth-Limited Search (DLS):
   - Define a recursive function that explores nodes up to a specified depth.
   - Mark nodes as visited and append them to a temporary stack once all their neighbors have been explored.

3. Implement Iterative Deepening Depth-First Search (IDDFS):
   - Iteratively call the DLS function with increasing depth limits.
   - Collect the nodes in topological order when a valid configuration is found.

4. Run IDDFS:
   - Create an instance of the Graph class.
   - Add edges to the graph.
   - Call the IDDFS function to get the topological order of the graph.

Procedure for Graph Coloring

1. Define the Graph Class:
   - Initialize an adjacency list to represent the graph.
   - Implement a method to add edges between vertices.

2. Read the Graph from a File:

- Define a function to read edges from a text file and construct the graph.

3. Implement Greedy Coloring Algorithm:
   - Define a function that assigns colors to vertices using a greedy approach.
   - Ensure no two adjacent vertices share the same color.

4. Run the Greedy Coloring Algorithm:
   - Read the graph from the input file.
   - Call the greedy coloring function to color the graph.
   - Print the color assigned to each vertex.

Procedure for N-Queens Problem using Backtracking

1. Define the is_safe Function:
   - Check if it's safe to place a queen at a given position by verifying columns and diagonals.

2. Implement the solve_n_queens_util Function:
   - Recursively attempt to place queens row by row.
   - Backtrack if placing a queen leads to an invalid configuration.

3. Implement the solve_n_queens Function:
   - Initialize the board and call the recursive utility function.
   - Collect all valid solutions.

4. Print the Solutions:
   - Define a function to print all solutions in a readable format.
   - Call this function after solving the N-Queens problem.

4. IMPLEMENTATION

Problem 1: Write a program to perform topological search using IDDFS.

Code:

```
from collections import defaultdict, deque

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

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

```
    def dls(self, node, depth, visited, temp_stack):
        if depth < 0:
            return False
        visited.add(node)
        cycle_found = False
        for neighbor in self.graph[node]:
            if neighbor not in visited:
                if self.dls(neighbor, depth - 1, visited, temp_stack):
                    cycle_found = True
            elif neighbor in temp_stack:
                cycle_found = True
        temp_stack.appendleft(node)
        return cycle_found

    def iddfs(self, max_depth):
        for depth in range(max_depth):
            visited = set()
            temp_stack = deque()
            cycle_found = False
            for node in self.graph:
                if node not in visited:
                    if self.dls(node, depth, visited, temp_stack):
                        cycle_found = True
            if not cycle_found:
                return list(temp_stack)
        return []


g = Graph()
g.add_edge('A', 'C')
g.add_edge('B', 'C')
g.add_edge('B', 'D')
g.add_edge('C', 'E')
g.add_edge('D', 'F')
g.add_edge('E', 'F')

max_depth = len(g.graph) * len(g.graph)  # A rough upper bound for the depth
print("Topological order using IDDFS:", g.iddfs(max_depth))
```

Problem 2: Write a program to perform graph coloring algorithm which take input as text file from computer.

Code:

```
from collections import defaultdict
```

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

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

    def greedy_coloring(self):
        result = {}
        for u in sorted(self.vertices):  # Sorting to ensure deterministic output
            available_colors = [True] * len(self.vertices)
            for neighbor in self.graph[u]:
                if neighbor in result:
                    color = result[neighbor]
                    available_colors[color] = False
            for color, available in enumerate(available_colors):
                if available:
                    result[u] = color
                    break
        return result

def read_graph_from_file(filename):
    graph = Graph()
    with open(filename, 'r') as file:
        for line in file:
            u, v = line.strip().split()
            graph.add_edge(u, v)
    return graph

def main():
    filename = 'graph.txt'
    graph = read_graph_from_file(filename)
    coloring = graph.greedy_coloring()

    print("Vertex Coloring:")
```

```
    for vertex, color in coloring.items():
        print(f"Vertex {vertex}: Color {color}")

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

Problem 3: N queen Puzzle Problem

Code:

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

def solve_n_queens_util(board, row, N, solutions):
    if row >= N:
        solutions.append([row[:] for row in board])
        return
    for col in range(N):
        if is_safe(board, row, col, N):
            board[row][col] = 1
            solve_n_queens_util(board, row + 1, N, solutions)
            board[row][col] = 0

def solve_n_queens(N):
    board = [[0 for _ in range(N)] for _ in range(N)]
    solutions = []
    solve_n_queens_util(board, 0, N, solutions)
    return solutions

def print_solutions(solutions):
    for idx, solution in enumerate(solutions):
        print(f"Solution {idx + 1}:")
        for row in solution:
```

```
            print(" ".join(str(cell) for cell in row))
        print()

N = 4
solutions = solve_n_queens(N)
print_solutions(solutions)
```

5. TEST RESULT / OUTPUT

Problem 1: Write a program to perform topological search using IDDFS.

```
/home/niloy/PycharmProjects/pythonProject/.venv/bin/python /
Topological order using IDDFS: ['E', 'D', 'C', 'B', 'A']

Process finished with exit code 0
```

Problem 2: Write a program to perform graph coloring algorithm which take input as text file from computer.

```
/home/niloy/PycharmProjects/pythonProj
Vertex Coloring:
Vertex A: Color 0
Vertex B: Color 1
Vertex C: Color 2
Vertex D: Color 0
Vertex E: Color 1

Process finished with exit code 0
```

Problem 3: N queen puzzle problem

```
/home/niloy/PycharmProjects/pythonPr
Solution 1:
0 1 0 0
0 0 0 1
1 0 0 0
0 0 1 0

Solution 2:
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
```

6. ANALYSIS AND DISCUSSION

1. Successes:
   - IDDFS algorithm accurately found the topological order of the graph, demonstrating its versatility beyond traditional search problems.
   - Graph coloring algorithm efficiently colored vertices without conflicts, showcasing an effective solution for constraint satisfaction problems.
   - N-Queens solver successfully generated all distinct solutions using backtracking, highlighting its effectiveness in solving combinatorial problems.

2. Challenges:
   - Understanding and applying IDDFS for topological sorting posed a challenge due to its unconventional use.
   - Ensuring the graph coloring algorithm produced optimal colorings required careful management of adjacent vertices.
   - Implementing backtracking for the N-Queens problem involved managing recursion and backtracking effectively.

3. Learning:
   - Enhanced understanding of IDDFS and its applications.
   - Improved skills in solving constraint satisfaction problems using graph coloring.
   - Deepened understanding of recursive algorithms through backtracking implementation.

Objective Achievement:

1. IDDFS Topological Sorting:
   - Successfully applied IDDFS to find the topological order, showcasing its versatility in unconventional applications.

2. Graph Coloring:
   - Implemented a greedy algorithm to efficiently color graph vertices, meeting the objective of solving constraint satisfaction problems.

3. N-Queens Problem:
   - Backtracking implementation efficiently solved the N-Queens problem, achieving the objective of generating all valid configurations.