U23AI021

AI LAB 1

Harshil Andhariya

Harshil Andhariya

U23AI021

AI-LAB 1

Sardar Vallabhbhai National Institute of Technology, Surat

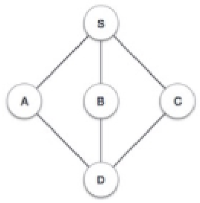
Department of Artificial Intelligence

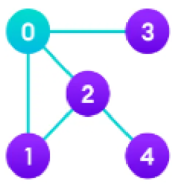
Artificial Intelligence(AI202)

B.tech II Division H

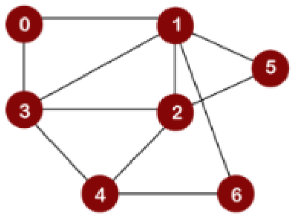
Assignment 1

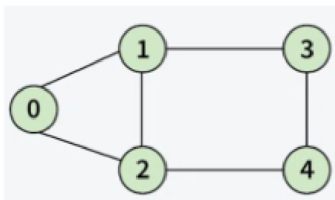
1) Implement the Depth First Search (DFS) Algorithm for the following graphs. Determine where this algorithm will be used in Artificial Intelligence.

a) 

b) 

2) Implement the Breadth First Search (BFS) Algorithm for the following graphs. Determine where this algorithm will be used in Artificial Intelligence.

a) 

b) 

Harshil Andhariya

U23AI021

AI-LAB 1

```
#include <iostream>
#include <vector>
using namespace std;

void dfs(int node, vector<vector<int>>& adjMatrix, vector<bool>& visited) {
    visited[node] = true;
    cout << node << " ";

    for (int i = 0; i < adjMatrix[node].size(); i++) {
        if (adjMatrix[node][i] == 1 && !visited[i]) {
            dfs(i, adjMatrix, visited);
        }
    }
}

int main() {
    int numNodes;

    cout << "Enter the number of nodes in the graph: ";
    cin >> numNodes;



    vector<vector<int>> adjMatrix(numNodes, vector<int>(numNodes, 0));

    cout << "Enter the adjacency matrix:\n";
    for (int i = 0; i < numNodes; i++) {
        for (int j = 0; j < numNodes; j++) {
            cin >> adjMatrix[i][j];
        }
    }

    int startNode;
    cout << "Enter the starting node: ";
    cin >> startNode;

    vector<bool> visited(numNodes, false);
```

```
    cout << "DFS Traversal: ";
    dfs(startNode, adjMatrix, visited);
    cout << endl;

    return 0;
}
```

a)

```
#include <iostream>
#include <vector>
using namespace std;

void dfs(int node, vector<vector<int>>& adjMatrix, vector<bool>& visited) {

    visited[node] = true;
    cout << node << " ";

    for (int i = 0; i < adjMatrix[node].size(); i++) {

        if (adjMatrix[node][i] == 1 && !visited[i]) {
            dfs(i, adjMatrix, visited);
        }
    }
}

int main() {

    vector<vector<int>> adjMatrix = {
        {0, 0, 0, 1, 1},
        {0, 0, 0, 1, 1},
        {0, 0, 0, 1, 1},
        {1, 1, 1, 0, 0},
        {1, 1, 1, 0, 0}
    };
```

Harshil Andhariya

U23AI021

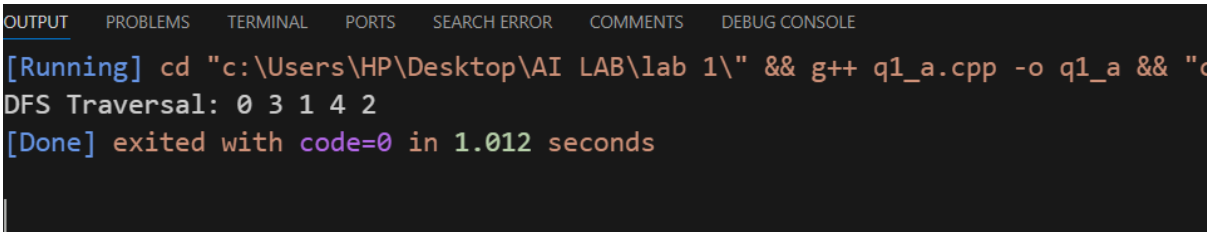
AI-LAB 1

```
    int numNodes = adjMatrix.size();
    vector<bool> visited(numNodes, false);

    cout << "DFS Traversal: ";

    for (int i = 0; i < numNodes; i++) {
        if (!visited[i]) {
            dfs(i, adjMatrix, visited);
        }
    }

    return 0;
}
```



b)

```
#include <iostream>
#include <vector>
using namespace std;

void dfs(int node, vector<vector<int>>& adjMatrix, vector<bool>& visited) {

    visited[node] = true;
    cout << node << " ";

    for (int i = 0; i < adjMatrix[node].size(); i++) {

        if (adjMatrix[node][i] == 1 && !visited[i]) {
            dfs(i, adjMatrix, visited);
        }
    }
}
```

Harshil Andhariya

U23AI021

AI-LAB 1

```
int main() {

    vector<vector<int>> adjMatrix = {
        {0, 1, 1, 1, 0},
        {1, 0, 1, 0, 0},
        {1, 1, 0, 0, 1},
        {1, 0, 0, 0, 0},
        {0, 0, 1, 0, 0}
    };

    int numNodes = adjMatrix.size();
    vector<bool> visited(numNodes, false);

    cout << "DFS Traversal: ";

    for (int i = 0; i < numNodes; i++) {
        if (!visited[i]) {
            dfs(i, adjMatrix, visited);
        }
    }

    return 0;
}
```

```
[Running] cd "c:\Users\HP\Desktop\AI LAB\lab 1\" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "c:\Users\HP\Desktop\AI LAB\lab 1\"tempCodeRunnerFile
DFS Traversal: 0 1 2 4 3
[Done] exited with code=0 in 1.131 seconds
```

Harshil Andhariya

U23AI021

AI-LAB 1

2)

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

void bfs(int startNode, vector<vector<int>>& adjMatrix, vector<bool>& visited) {
    queue<int> q;
    visited[startNode] = true;
    q.push(startNode);

    while (!q.empty()) {
        int node = q.front();
        q.pop();
        cout << node << " ";

        for (int i = 0; i < adjMatrix[node].size(); i++) {
            if (adjMatrix[node][i] == 1 && !visited[i]) {
                visited[i] = true;
                q.push(i);
            }
        }
    }
}

int main() {
    int numNodes;

    cout << "Enter the number of nodes in the graph: ";
    cin >> numNodes;

    vector<vector<int>> adjMatrix(numNodes, vector<int>(numNodes, 0));

    cout << "Enter the adjacency matrix:\n";
    for (int i = 0; i < numNodes; i++) {
```

Harshil Andhariya

U23AI021

AI-LAB 1

```
        for (int j = 0; j < numNodes; j++) {
            cin >> adjMatrix[i][j];
        }
    }

    int startNode;
    cout << "Enter the starting node: ";
    cin >> startNode;

    vector<bool> visited(numNodes, false);
    cout << "BFS Traversal: ";
    bfs(startNode, adjMatrix, visited);
    cout << endl;

    return 0;
}
```

a)

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

void bfs(int startNode, vector<vector<int>> &adjMatrix, vector<bool> &visited)
{
    queue<int> q;
    visited[startNode] = true;
    q.push(startNode);

    while (!q.empty())
    {
        int node = q.front();
        q.pop();
        cout << node << " ";

        for (int i = 0; i < adjMatrix[node].size(); i++)
```

```
        {
            if (adjMatrix[node][i] == 1 && !visited[i])
            {
                visited[i] = true;
                q.push(i);
            }
        }
    }
}

int main()
{

    vector<vector<int>> adjMatrix = {
        {0, 1, 0, 1, 0, 0, 0},
        {1, 1, 1, 1, 0, 1, 1},
        {1, 1, 0, 1, 1, 1, 0},
        {1, 1, 1, 0, 1, 0, 0},
        {0, 0, 1, 1, 0, 0, 0},
        {0, 0, 1, 1, 0, 1, 0},
        {0, 1, 0, 1, 0, 0, 0}
    };

    int numNodes = adjMatrix.size();
    vector<bool> visited(numNodes, false);

    cout << "BFS Traversal: ";

    for (int i = 0; i < numNodes; i++)
    {
        if (!visited[i])
        {
            bfs(i, adjMatrix, visited);
        }
    }
```

```
    return 0;
}
```

```
[Running] cd "c:\Users\HP\Desktop\AI LAB\lab 1\" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "c:\Users\HP\Desktop\AI LAB\lab 1\"tempCodeRunnerFile
BFS Traversal: 0 1 3 2 5 6 4
[Done] exited with code=0 in 1.24 seconds
```

b)

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

void bfs(int startNode, vector<vector<int>>& adjMatrix, vector<bool>& visited) {
    queue<int> q;
    visited[startNode] = true;
    q.push(startNode);

    while (!q.empty()) {
        int node = q.front();
        q.pop();
        cout << node << " ";


        for (int i = 0; i < adjMatrix[node].size(); i++) {
            if (adjMatrix[node][i] == 1 && !visited[i]) {
                visited[i] = true;
                q.push(i);
            }
        }
    }
}

int main() {
```

Harshil Andhariya

U23AI021

AI-LAB 1

```
    vector<vector<int>> adjMatrix = {
        {0, 1, 1, 0, 0},
        {1, 0, 1, 1, 0},
        {1, 1, 0, 1, 0},
        {0, 1, 0, 0, 1},
        {0, 0, 1, 1, 0}
    };

    int numNodes = adjMatrix.size();
    vector<bool> visited(numNodes, false);

    cout << "BFS Traversal: ";

    for (int i = 0; i < numNodes; i++) {
        if (!visited[i]) {
            bfs(i, adjMatrix, visited);
        }
    }

    return 0;
}
```

```
[Running] cd "c:\Users\HP\Desktop\AI LAB\lab 1\" && g++ q2_b.cpp -o q2_b && "c:\Users\HP\Desktop\AI LAB\lab 1\"q2_b
BFS Traversal: 0 1 2 3 4
[Done] exited with code=0 in 1.054 seconds
```

Applications of DFS and BFS in Artificial Intelligence

U23AI021

AI LAB 1

Harshil Andhariya

Harshil Andhariya

U23AI021

AI-LAB 1

Applications of DFS in AI

1. Pathfinding in Game AI

- **Maze Solving**: DFS can be used to find paths or solutions in a maze or labyrinth-like structure.
- **Level Generation**: In procedural content generation, DFS is used to create intricate and connected maps, such as game levels.

2. Search Problems

- **Problem Solving**: DFS is used to search for solutions in state spaces, such as puzzles or configuration problems (e.g., the 8-puzzle problem).
- **Tree Search**: For problems like the Tower of Hanoi, DFS explores possible configurations to find the goal state.

3. Planning and Decision Making

- **Decision Trees**: DFS is used to explore all possible decision paths in a decision tree to determine the optimal choice.
- **Backtracking Algorithms**: DFS is central to backtracking, used in solving constraint satisfaction problems like Sudoku, n-Queens, and CSPs in AI.

4. Knowledge Representation

- **Inference in Logical Systems**: In Prolog and other logic programming environments, DFS-like methods are used to perform depth-based reasoning.
- **Ontology Reasoning**: DFS helps navigate and reason over hierarchical knowledge structures, such as semantic networks.

5. Constraint Satisfaction Problems (CSPs)

- DFS is instrumental in exploring the solution space for CSPs by systematically assigning values to variables and backtracking when constraints are violated.

6. Path Planning in Robotics

- **Navigation**: Robots use DFS for simple navigation tasks in graph-represented environments.
- **Exploration**: DFS helps explore unknown environments in robotic systems, such as autonomous mapping.

7. Natural Language Processing (NLP)

- **Parsing Trees**: DFS traverses syntax trees or dependency trees in linguistic analysis.
- **Semantic Role Labeling**: DFS is used to explore the relationships between words in a sentence.

8. Artificial Neural Networks (ANNs)

- **Neural Network Search**: DFS can be applied to search through the architecture space or hyperparameter configurations for optimizing neural networks.
- **Decision-Making Models**: In AI models resembling decision trees, DFS aids in inference and prediction.

9. AI in Automated Systems

- **Game Theory**: In Minimax algorithms for games like chess, DFS explores game tree paths to calculate the optimal moves.
- **Expert Systems**: DFS-like strategies are employed for searching knowledge bases to derive conclusions or diagnoses.

10. Web Crawling and Search Engines

- DFS can simulate how search engines crawl websites by exploring links deeply before moving to other parts of the web.

11. Planning in AI

- **Task Scheduling**: DFS is useful in analyzing dependencies in Directed Acyclic Graphs (DAGs) for task scheduling or execution order in AI systems.
- **Recursive Problem Solving**: Many recursive algorithms in AI are based on DFS principles.

Harshil Andhariya

U23AI021

AI-LAB 1

Applications of BFS in AI

1. Pathfinding in Game AI

- **Shortest Path**: BFS is ideal for finding the shortest path in an unweighted graph (e.g., in games or navigation systems).
- **Exploration in Grids**: Used to explore game maps or terrains level-by-level, ensuring comprehensive coverage of accessible areas.

2. Search Problems

- **State Space Search**: BFS explores all possible states layer by layer, ensuring optimal solutions when the cost per step is uniform.
- **Uninformed Search Algorithms**: BFS is a cornerstone of uninformed (blind) search, ensuring completeness and optimality in suitable cases.

3. Planning and Scheduling

- **Planning in AI Systems**: BFS is used to explore potential actions and their consequences to find the best sequence of actions.
- **Task Scheduling**: BFS explores dependencies in Directed Acyclic Graphs (DAGs) to determine execution order.

4. Natural Language Processing (NLP)

- **Word Ladder Problems**: BFS can find the shortest transformation sequence from one word to another.
- **Syntax and Semantic Analysis**: It traverses parse trees or dependency trees level by level, useful in language understanding.

5. Social Network Analysis

- **Friend Recommendations**: BFS helps explore connections in social graphs to suggest friends based on mutual relationships.
- **Influence Spread**: It identifies users within a certain distance (in terms of hops) in a network, critical for viral marketing campaigns.

6. Robotics and Navigation

- **Robot Navigation**: BFS helps robots navigate unweighted graphs, ensuring the shortest route to a target location.
- **Exploration in Unknown Environments**: BFS provides a structured approach to explore an area systematically.

Harshil Andhariya

U23AI021

AI-LAB 1

7. Web Crawling and Search Engines

- BFS mimics how web crawlers explore websites layer by layer, discovering all accessible pages in increasing depth.

8. Artificial Neural Networks (ANNs)

- **Layer-Wise Computation**: BFS is conceptually similar to the way feedforward neural networks compute outputs layer by layer.

9. Game Theory and AI

- **Game Tree Analysis**: BFS ensures all moves at the current depth are analyzed before progressing deeper, often combined with heuristics in strategies like Monte Carlo Tree Search.
- **Solving Puzzles**: For puzzles like sliding tiles, BFS guarantees finding the shortest sequence of moves.

10. AI in Automated Systems

- **Knowledge Representation**: BFS is used to query hierarchical data structures like ontologies or decision trees to find specific information.
- **Expert Systems**: BFS aids in exhaustive search for reasoning over knowledge bases.

11. Graph Analysis

- **Connected Components**: BFS is used to identify connected components in undirected graphs.
- **Shortest Paths in Unweighted Graphs**: BFS ensures the shortest path is found in graphs with equal edge weights.

12. Collaborative Filtering

- **Recommender Systems**: BFS is used in recommendation systems to explore user-item graphs layer by layer for suggesting items based on proximity.

13. AI Planning Problems

- **Level Order Exploration**: BFS explores actions systematically, ensuring completeness and optimality in uniform-cost domains.
- **Dependency Analysis**: In planning graphs, BFS identifies task dependencies and order.

14. Simulation and Modeling

- BFS is used in modeling the spread of information, diseases, or other phenomena across networks.