Bansilal Ramnath Agarwal Charitable Trust's

Vishwakarma Institute of Technology

(An Autonomous Institute affiliated to Savitribai Phule Pune University)

**Assignment -4**

|  |  |
| --- | --- |
| **Subject** | ADS |
| **Name** | Parth Petkar |
| **Class** | CS-C |
| **Roll No.** | 76 |

* **Graphs**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX\_NODES 10

struct Node {

    int data;

    struct Node\* next;

};

struct Node\* adjacencyList[MAX\_NODES] = {NULL};

bool visitedList[MAX\_NODES] = {false};

int colors[MAX\_NODES] = {0}; // 0: Uncolored, 1: Color 1, 2: Color 2

bool adjacencyMatrix[MAX\_NODES][MAX\_NODES] = {false};

bool visitedMatrix[MAX\_NODES] = {false};

void addEdgeToList(int node1, int node2) {

    struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

    newNode->data = node2;

    newNode->next = adjacencyList[node1];

    adjacencyList[node1] = newNode;

}

void dfsListRec(int node) {

    visitedList[node] = true;

    printf("%d ", node);

    struct Node\* current = adjacencyList[node];

    while (current != NULL) {

        if (!visitedList[current->data]) {

            dfsListRec(current->data);

        }

        current = current->next;

    }

}

void dfsListNoRec(int start, int nodes) {

    int stack[MAX\_NODES];

    int top = -1;

    stack[++top] = start;

    while (top != -1) {

        int current = stack[top--];

        if (!visitedList[current]) {

            visitedList[current] = true;

            printf("%d ", current);

        }

        struct Node\* temp = adjacencyList[current];

        while (temp != NULL) {

            if (!visitedList[temp->data]) {

                stack[++top] = temp->data;

            }

            temp = temp->next;

        }

    }

}

void dfsMatrixRec(int node, int nodes) {

    visitedMatrix[node] = true;

    printf("%d ", node);

    for (int i = 0; i < nodes; ++i) {

        if (adjacencyMatrix[node][i] && !visitedMatrix[i]) {

            dfsMatrixRec(i, nodes);

        }

    }

}

void dfsMatrixNoRec(int start, int nodes) {

    int stack[MAX\_NODES];

    int top = -1;

    stack[++top] = start;

    while (top != -1) {

        int current = stack[top--];

        if (!visitedMatrix[current]) {

            visitedMatrix[current] = true;

            printf("%d ", current);

        }

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

            if (adjacencyMatrix[current][i] && !visitedMatrix[i]) {

                stack[++top] = i;

            }

        }

    }

}

void bfsListNoRec(int start, int nodes) {

    bool queue[MAX\_NODES] = {false};

    int front = -1, rear = -1;

    queue[++rear] = start;

    visitedList[start] = true;

    printf("BFS traversal using adjacency list starting from node %d: ", start);

    while (front != rear) {

        int current = queue[++front];

        printf("%d ", current);

        struct Node\* temp = adjacencyList[current];

        while (temp != NULL) {

            if (!visitedList[temp->data]) {

                queue[++rear] = temp->data;

                visitedList[temp->data] = true;

            }

            temp = temp->next;

        }

    }

}

void bfsMatrixNoRec(int start, int nodes) {

    bool queue[MAX\_NODES] = {false};

    int front = -1, rear = -1;

    queue[++rear] = start;

    visitedMatrix[start] = true;

    printf("BFS traversal using adjacency matrix starting from node %d: ", start);

    while (front != rear) {

        int current = queue[++front];

        printf("%d ", current);

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

            if (adjacencyMatrix[current][i] && !visitedMatrix[i]) {

                queue[++rear] = i;

                visitedMatrix[i] = true;

            }

        }

    }

}

bool isBipartiteList(int node, int color) {

    visitedList[node] = true;

    colors[node] = color;

    struct Node\* current = adjacencyList[node];

    while (current != NULL) {

        if (!visitedList[current->data]) {

            if (!isBipartiteList(current->data, 3 - color)) {

                return false;

            }

        } else if (colors[current->data] == color) {

            return false;

        }

        current = current->next;

    }

    return true;

}

bool isBipartiteMatrix(int node, int color, int nodes) {

    visitedMatrix[node] = true;

    colors[node] = color;

    for (int i = 0; i < nodes; ++i) {

        if (adjacencyMatrix[node][i]) {

            if (!visitedMatrix[i]) {

                if (!isBipartiteMatrix(i, 3 - color, nodes)) {

                    return false;

                }

            } else if (colors[i] == color) {

                return false;

            }

        }

    }

    return true;

}

int main() {

    int nodesList, edgesList;

    printf("Enter the number of nodes for adjacency list graph: ");

    scanf("%d", &nodesList);

    printf("Enter the number of edges for adjacency list graph: ");

    scanf("%d", &edgesList);

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

        int node1, node2;

        printf("Enter edge %d (node1 node2): ", i + 1);

        scanf("%d %d", &node1, &node2);

        addEdgeToList(node1, node2);

    }

    int nodesMatrix, edgesMatrix;

    printf("Enter the number of nodes for adjacency matrix graph: ");

    scanf("%d", &nodesMatrix);

    printf("Enter the number of edges for adjacency matrix graph: ");

    scanf("%d", &edgesMatrix);

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

        int node1, node2;

        printf("Enter edge %d (node1 node2): ", i + 1);

        scanf("%d %d", &node1, &node2);

        adjacencyMatrix[node1][node2] = true;

    }

    int choice;

    do {

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

            visitedList[i] = false;

            colors[i] = 0;

        }

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

            visitedMatrix[i] = false;

            colors[i] = 0;

        }

        printf("1. DFS using adjacency list with recursion\n");

        printf("2. DFS using adjacency list without recursion\n");

        printf("3. DFS using adjacency matrix with recursion\n");

        printf("4. DFS using adjacency matrix without recursion\n");

        printf("5. BFS using adjacency list without recursion\n");

        printf("6. BFS using adjacency matrix without recursion\n");

        printf("7. Check if adjacency list graph is bipartite\n");

        printf("8. Check if adjacency matrix graph is bipartite\n");

        printf("9. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("DFS traversal using adjacency list with recursion: ");

                dfsListRec(0);

                printf("\n");

                break;

            case 2:

                printf("DFS traversal using adjacency list without recursion: ");

                dfsListNoRec(0, nodesList);

                printf("\n");

                break;

            case 3:

                printf("DFS traversal using adjacency matrix with recursion: ");

                dfsMatrixRec(0, nodesMatrix);

                printf("\n");

                break;

            case 4:

                printf("DFS traversal using adjacency matrix without recursion: ");

                dfsMatrixNoRec(0, nodesMatrix);

                printf("\n");

                break;

            case 5:

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

                    visitedList[i] = false;

                }

                bfsListNoRec(0, nodesList);

                printf("\n");

                break;

            case 6:

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

                    visitedMatrix[i] = false;

                }

                bfsMatrixNoRec(0, nodesMatrix);

                printf("\n");

                break;

            case 7:

                if (isBipartiteList(0, 1)) {

                    printf("The adjacency list graph is bipartite.\n");

                } else {

                    printf("The adjacency list graph is not bipartite. Here's the evidence:\n");

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

                        if (colors[i] == 1) {

                            printf("Node %d is in Color 1.\n", i);

                        } else if (colors[i] == 2) {

                            printf("Node %d is in Color 2.\n", i);

                        }

                    }

                }

                break;

            case 8:

                if (isBipartiteMatrix(0, 1, nodesMatrix)) {

                    printf("The adjacency matrix graph is bipartite.\n");

                } else {

                    printf("The adjacency matrix graph is not bipartite. Here's the evidence:\n");

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

                        if (colors[i] == 1) {

                            printf("Node %d is in Color 1.\n", i);

                        } else if (colors[i] == 2) {

                            printf("Node %d is in Color 2.\n", i);

                        }

                    }

                }

                break;

            case 9:

                printf("Exiting the program.\n");

                break;

            default:

                printf("Invalid choice. Please try again.\n");

        }

    } while (choice != 9);

    return 0;

}

**Output:**

**A screenshot of a computer program

Description automatically generated** A screenshot of a computer program

Description automatically generated