Name - Srushti Bhivaji Salgar
PRN - B24CE1079
Class - SE 2 Batch - A
Subject - Data Structures
Assignment 7

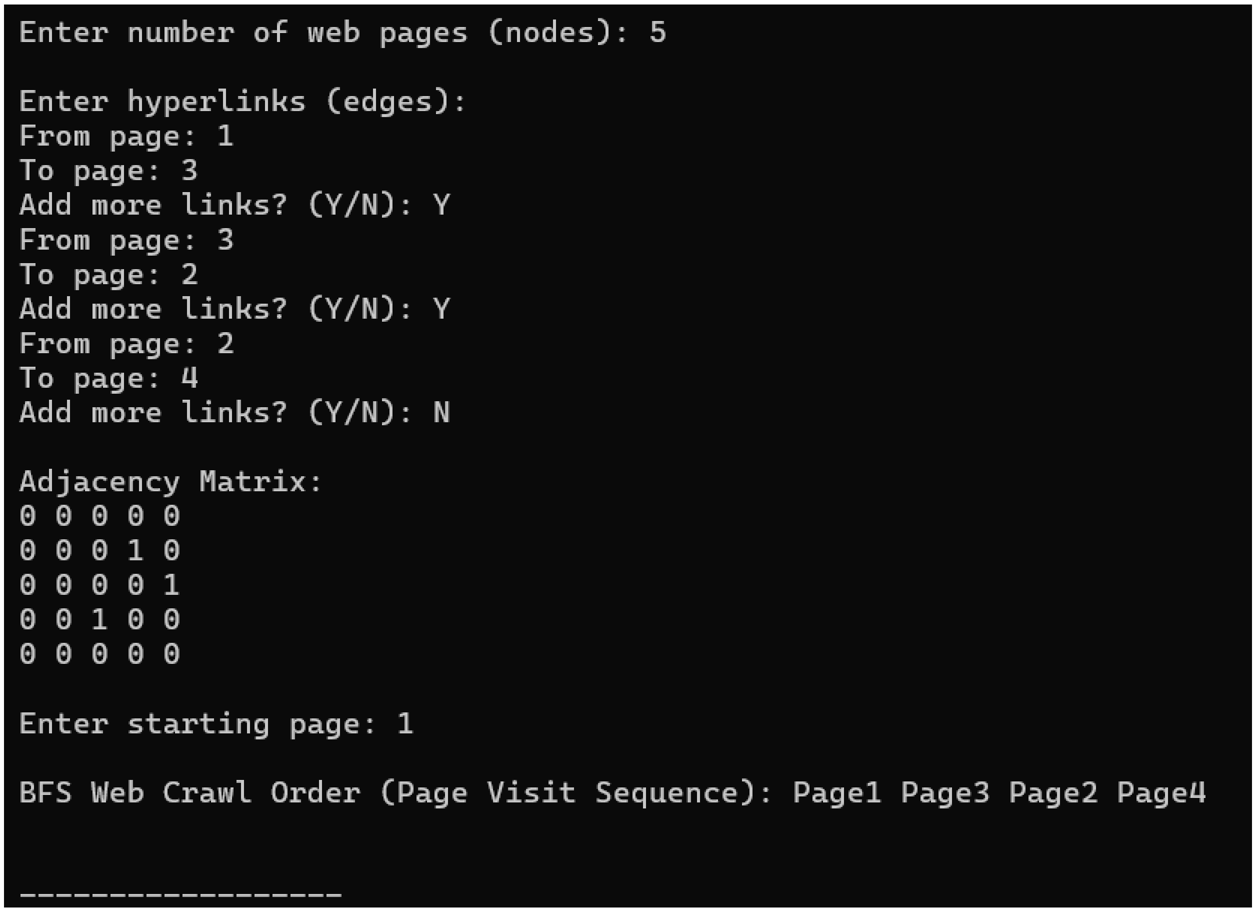
/***Web Crawling: Breadth First Search (BFS):**
Application:Indexing web pages for search engines.
Example: A web crawler uses BFS to visit web pages systematically, starting from a seed URL and exploring links level by level. Nodes represent web pages. Edges represent hyperlinks. BFS ensures that pages at the same "depth" (distance from the starting page) are visited before moving to deeper levels. Write a program to simulate the indexing of web pages for a search engine using a Breadth-First Search (BFS) algorithm.*/

```
#include <iostream>
using namespace std;
const int MAX = 10;
int graph[MAX][MAX];
bool visited[MAX];
int q[MAX]; // queue array
int front = -1, rear = -1;
void enqueue(int x) {
if (rear == MAX - 1) {
cout << "\nQueue Full!";
return;
}
if (front == -1) front = 0;
q[++rear] = x;
}
int dequeue() {
if (front == -1 || front > rear)
return -1;
return q[front++];
}
bool isEmpty() {
return (front == -1 || front > rear);
}
void BFS(int start, int n) {
enqueue(start);
visited[start] = true;
cout << "\nBFS Web Crawl Order (Page Visit Sequence): ";
```

```
while (!isEmpty()) {
int node = dequeue();
cout << "Page" << node << " ";
for (int i = 0; i < n; i++) {
if (graph[node][i] == 1 && !visited[i]) {
visited[i] = true;
enqueue(i);
}
}
}
}
int main() {
int n, u, v;
char more;
cout << "Enter number of web pages (nodes): ";
cin >> n;
// Initialize
for (int i = 0; i < n; i++) {
visited[i] = false;
for (int j = 0; j < n; j++)
graph[i][j] = 0;
}
cout << "\nEnter hyperlinks (edges):\n";
do {
cout << "From page: ";
cin >> u;
cout << "To page: ";
cin >> v;
if (u >= 0 && u < n && v >= 0 && v < n)
graph[u][v] = 1;
else
cout << "Invalid page numbers!\n";
cout << "Add more links? (Y/N): ";
cin >> more;
} while (more == 'Y' || more == 'y');
cout << "\nAdjacency Matrix:\n";
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++)
cout << graph[i][j] << " ";
cout << endl;
}
int start;
cout << "\nEnter starting page: ";
cin >> start;
```

```
BFS(start, n);
cout << endl;
return 0;
}
```

OUTPUT



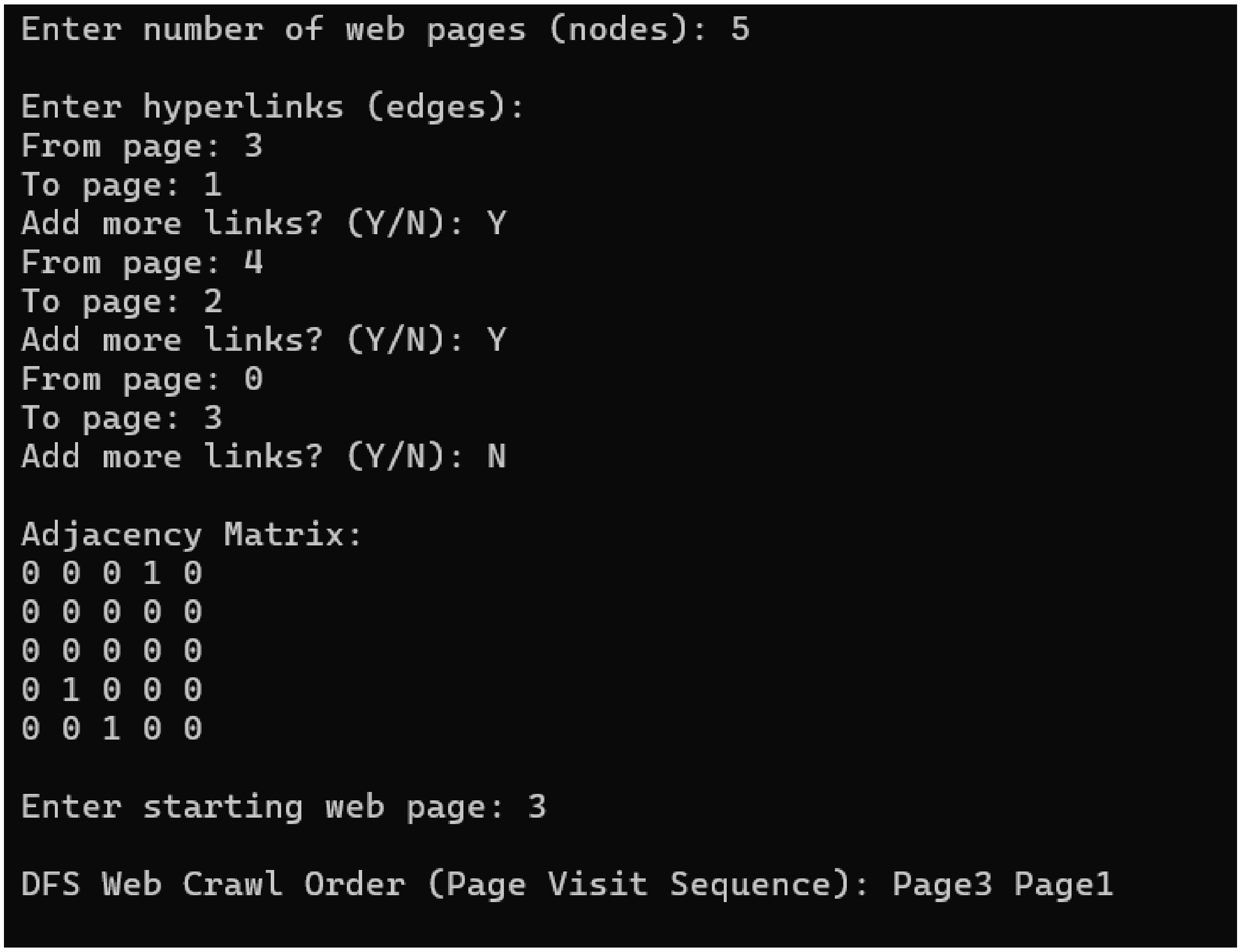
```
Enter number of web pages (nodes): 5

Enter hyperlinks (edges):
From page: 1
To page: 3
Add more links? (Y/N): Y
From page: 3
To page: 2
Add more links? (Y/N): Y
From page: 2
To page: 4
Add more links? (Y/N): N

Adjacency Matrix:
0 0 0 0 0
0 0 0 1 0
0 0 0 0 1
0 0 1 0 0
0 0 0 0 0

Enter starting page: 1

BFS Web Crawl Order (Page Visit Sequence): Page1 Page3 Page2 Page4


------------------
```

b)Depth First Search (DFS):

Application: Web crawlers use DFS to explore web pages systematically, following links and indexing content for search engines. Write a simple program to index web pages using Depth First Search (DFS). The program should simulate a web graph where pages are represented as nodes and hyperlinks as edges.

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

const int MAX = 10;
int graph[MAX][MAX];
bool visited[MAX];
int stackArr[MAX];
int top = -1;

void push(int x) {
    if (top == MAX - 1) {
        cout << "Stack Overflow\n";
        return;
    }
    stackArr[++top] = x;
}

int pop() {
    if (top == -1)
        return -1;
    return stackArr[top--];
}

bool isEmpty() {
    return top == -1;
}

void DFS_Stack(int start, int n) {
    push(start);
    cout << "\nDFS Web Crawl Order (Page Visit Sequence): ";

    while (!isEmpty()) {
        int node = pop();
        if (!visited[node]) {
            cout << "Page" << node << " ";
            visited[node] = true;
        }
        // Push all unvisited adjacent nodes to stack
        // (reverse order so smaller index gets visited first)
        for (int i = n - 1; i >= 0; i--) {
            if (graph[node][i] == 1 && !visited[i]) {
                push(i);
            }
        }
    }
}

int main() {
    int n, u, v;
    char more;
    cout << "Enter number of web pages (nodes): ";
    cin >> n;

    // Initialize graph and visited array
    for (int i = 0; i < n; i++) {
        visited[i] = false;
        for (int j = 0; j < n; j++)
            graph[i][j] = 0;
    }

    cout << "\nEnter hyperlinks (edges):\n";
    do {
        cout << "From page: ";
        cin >> u;
        cout << "To page: ";
        cin >> v;
        if (u >= 0 && u < n && v >= 0 && v < n)
            graph[u][v] = 1;
        else
            cout << "Invalid page numbers!\n";
        cout << "Add more links? (Y/N): ";
        cin >> more;
    } while (more == 'Y' || more == 'y');

    cout << "\nAdjacency Matrix:\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            cout << graph[i][j] << " ";
        cout << endl;
    }
    int start;
    cout << "\nEnter starting web page: ";
    cin >> start;

    DFS_Stack(start, n);

    cout << endl;
    return 0;
}
```

OUTPUT



```
Enter number of web pages (nodes): 5

Enter hyperlinks (edges):
From page: 3
To page: 1
Add more links? (Y/N): Y
From page: 4
To page: 2
Add more links? (Y/N): Y
From page: 0
To page: 3
Add more links? (Y/N): N

Adjacency Matrix:
0 0 0 1 0
0 0 0 0 0
0 0 0 0 0
0 1 0 0 0
0 0 1 0 0

Enter starting web page: 3

DFS Web Crawl Order (Page Visit Sequence): Page3 Page1
```