Department of
Computer Science and Engineering

Title: Implement Bread-First Search Traversal

Algorithms Lab
CSE 206

Green University of Bangladesh

1 Objective(s)

- To understand how to represent a graph using adjacency matrix.
- To understand how Bread-First Search (BFS) works.

2 Problem analysis

Every graph is a set of points referred to as vertices or nodes which are connected using lines called edges. The vertices represent entities in a graph. Edges, on the other hand, express relationships between entities. Hence, while nodes model entities, edges model relationships in a network graph. A graph G with a set of V vertices together with a set of E edges is represented as G= (V, E). Both vertices and edges can have additional attributes that are used to describe the entities and relationships. Figure 1 depicts a simple graph with five nodes and seven edges.

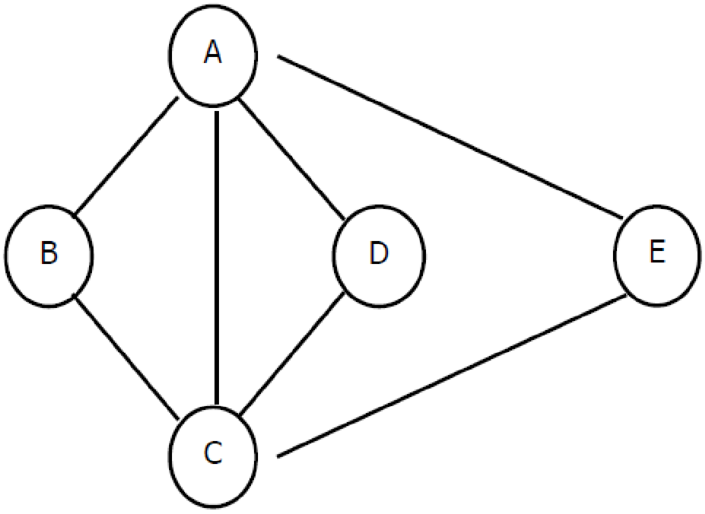
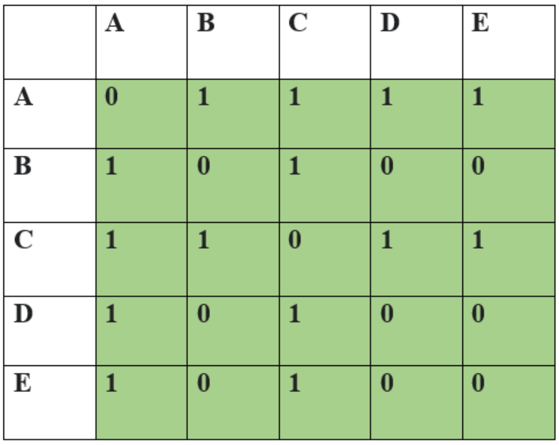


Figure 1: A simple graph

Adjacency Matrix:

Vertices are labelled (or re-labelled) with integers from 0 to V (G) - 1. A two-dimensional array “matrix” with dimensions V (G) * V (G) contains a 1 at matrix [j] [k] if there is an edge from the vertex labelled j to the vertex labelled k, and a 0 otherwise.Table:1 represents the graph of figure:1;



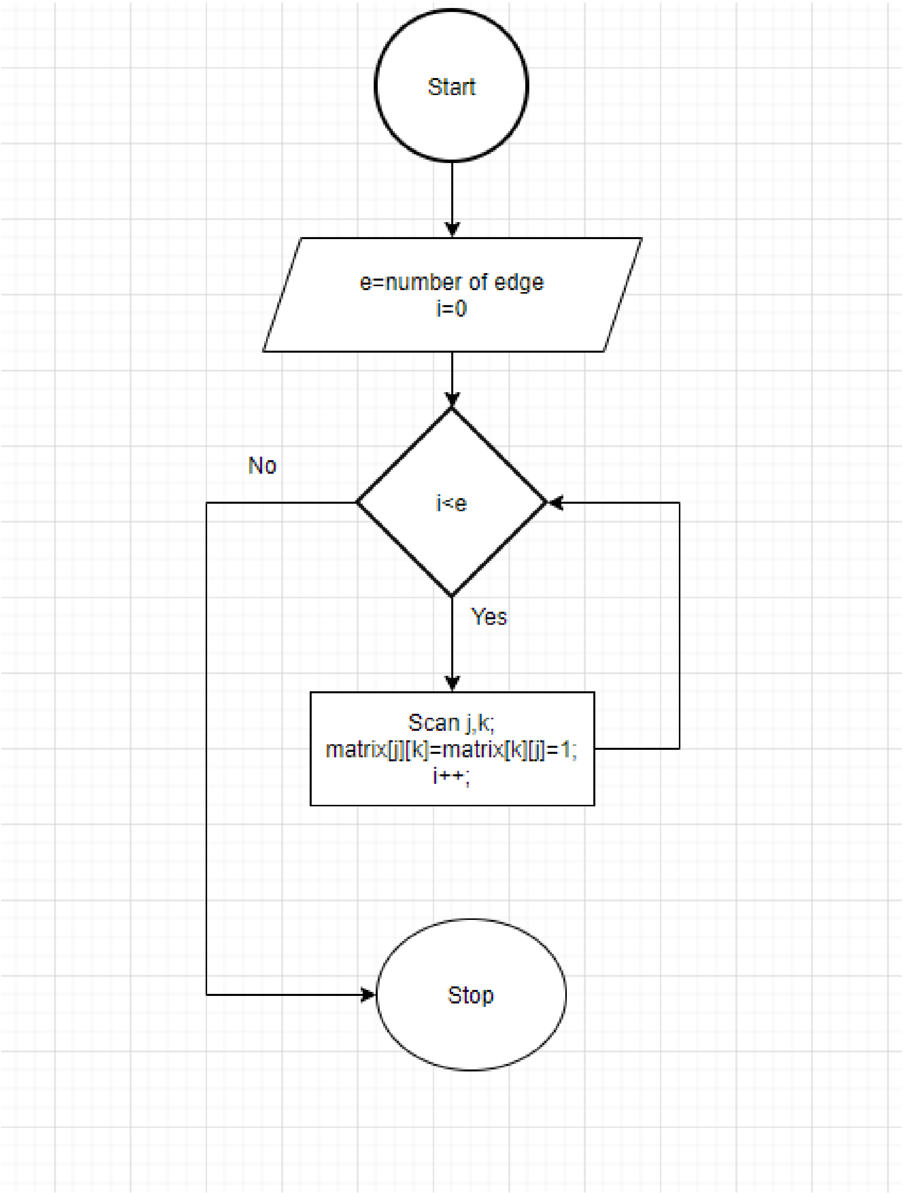
	A	**B**	**C**	**D**	**E**
A	**0**	**1**	**1**	**1**	**1**
B	**1**	**0**	**1**	**0**	**0**
C	**1**	**1**	**0**	**1**	**1**
D	**1**	**0**	**1**	**0**	**0**
E	**1**	**0**	**1**	**0**	**0**

Table: 1

3 Algorithm (Adjacency Matrix)

Step 1. Set i=0, e = Number of edges.
Step 2. e (number of edge) < i (Decision). • if no - continue with the step 7.
Step 3. Take the values of edge by giving the adjacency nodes [j], [k] (A, B, C, D, E=0,1,2,3,4).
Step 4. matrix[j][k] =matrix[k][j] =1.
Step 5. Increment i (i++).
Step 6. continue with the step 2.
Step 7. Stop.

4 Flowchart



5 Implementation in Java

```java
public class AdjacencyMatrix {
static int[][]matrix= new int[20][20];// 2D array that will contain the graph
    public static void main(String[] args) {
        int e=7, n=5;//e is number of edges, n is number of vetices
```

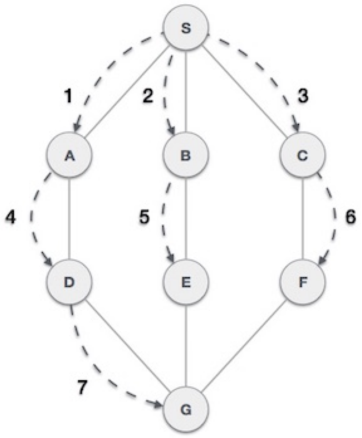
```
       Inmatrix(e);
      System.out.println("Output:");
       for(int i=0;i<n;i++){
          for(int j=0;j<n;j++){
          System.out.print(matrix[i][j]+" ");
         }
          System.out.println("");
       }


   }
   static void Inmatrix(int e){
       Scanner sn =new Scanner(System.in);
       System.out.println("Enter The Edges:");
      int i;
      char j,k;

for(i=0;i<e;i++){// this loop runs e times to take the all edges.
    j=sn.next().charAt(0);
    k=sn.next().charAt(0);
  matrix[(int)j-65][(int)k-65]=matrix[(int)k-65][(int)j-65]=1;
  // An undirected edge has both ways access between the nodes.
  //If  A to B has a way to go then B to A has the same way.

}

   }
```

6 Sample Input/Output (Compilation, Debugging & Testing)

Input:
Enter The Edges:
A B
A C
A D
A E
B C
C D
C E
Output:
0 1 1 1 1
1 0 1 0 0
1 1 0 1 1
1 0 1 0 0
1 0 1 0 0

7 Bread First Search

Breadth First Search (BFS) algorithm traverses a graph in a breadth Ward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.



As in the example given above, BFS algorithm traverses from A to B to E to F first then to C and G lastly to D. It employs the following rules.

8 Algorithm of BFS

A standard BFS implementation puts each vertex of the graph into one of two categories:
1.Visited
2.Not Visited
The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

The algorithm works as follows:
Step 1. Start by putting any one of the graph's vertices at the back of a queue.
Step 2. Take the front item of the queue and add it to the visited list.
Step 3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
Step 4. Keep repeating steps 2 and 3 until the queue is empty.

The graph might have two different disconnected parts so to make sure that we cover every vertex, we can also run the BFS algorithm on every node

9 Implementation in Java

```java
import java.util.*;
public class BreadFS {

  static char[] c={'A','B','C','D','E','F','G','S'};
  static int e[]={2,2,2,2,2,2,3,3};
  static int list[][]={{3,7},{4,7},{5,7},{0,6},{1,6},{2,6},{3,4,5},{0,1,2}};// adjacency list of graph figure 1.
  static int[] checked=new int[20];
  static int[] que=new int[20];
  static int first=0,last=0;
    public static void main(String[] args) {

        int i,j,n;
enq(7);
while(first<last){
n=dq();
    for(i=0;i<e[n];i++){
       if(notChecked(list[n][i])==1)
         enq(list[n][i]);
      }
    }



    }
     static int notChecked(int n){// this method checks the node visited or not
if(checked[n]==1)
    return 0;
return 1;
}

     static void enq(int n){//this methods is used to enqueue node
checked[n]=1;
que[last]=n;
last++;
}

static int  dq(){// this method is used to dequeue a node.
//printf("%c ",que[first]+65);
System.out.print(c[que[first]]+" ");
first++;
return que[first-1];
}
}
```

10 Sample Input/Output (Compilation, Debugging & Testing)

Output(BSF traversal sequence):
S A B C D E F G

11 Lab Task (Please implement yourself and show the output to the instructor)

1. Write a program to perform BSF traversal on a dynamic graph from user.
2. Write a program to find the level of each node using BFS.

12 Lab Exercise (Submit as a report)

- Write a program to detect the cycle in a graph using BFS.

13 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.