CHAPTER 3: SEARCHING IN GRAPHS

Discrete Mathematics 2

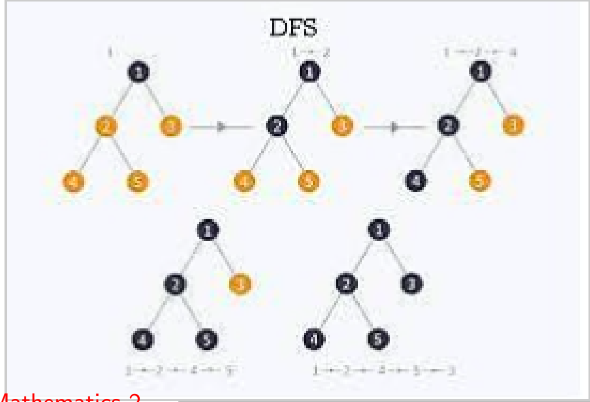
Lecturer: Nguyen Kieu Linh
Posts and Telecommunications Institute of Technology

Hanoi, 2023
http://www.ptit.edu.vn

Contents

Depth-First Search (DFS)

Input: The information of a matrix (adjacency matrix, edge list, adjacency list)
Output: Traverse all nodes in the graph.



Depth-First Search (DFS)

The algorithm starts at an arbitrary node of a graph and explores as far as possible along each branch before backtracking. The DFS algorithm consists of the following steps:

- Mark the current node as visited.
- Traverse the neighboring nodes that aren't visited and recursively call the DFS function for that node. (https://www.cs.usfca.edu/ galles/visualization/DFS.html)

Depth-First Search (DFS)

Input: The information of a matrix (adjacency matrix, edge list, adjacency list)
Output: Traverse all nodes in the graph.

```
DFS(u){ //u is the starting vertex
        <Visit u>;
        unTraverse[u] = false; // u has been traversed
        for(v ∈ Adj(u)){
                if( unTraverse[v]) //if v has not been traversed
                        DFS(v); //DFS from v
        }
}
```

DFS using Stack

```
DFS(u){
        Step 1: Initialize
        stack = ∅; // stack is empty
        push(stack, u);  //push vertex u to stack
        <Visit u>; //traverse vertex u
        unTraverse[u] = false; //u has been traversed
        Step 2: Loop
        while(stack ≠ ∅){
                s = pop(stack);  //get vertex at the top of stack
                for(t ∈ Adj(s)){
                        if( unTraverse[t]){  //if t has not been traversed
                                <Visit t>; //traverse vertex t
                                unTraverse [t] = false; //t has been
traversed

                                push(stack, s); //push s to stack
                                push(stack, t); //push t to stack
                                beak;  //get only one vertex t
                        }
                }
        }
        Step 3: Return results
```

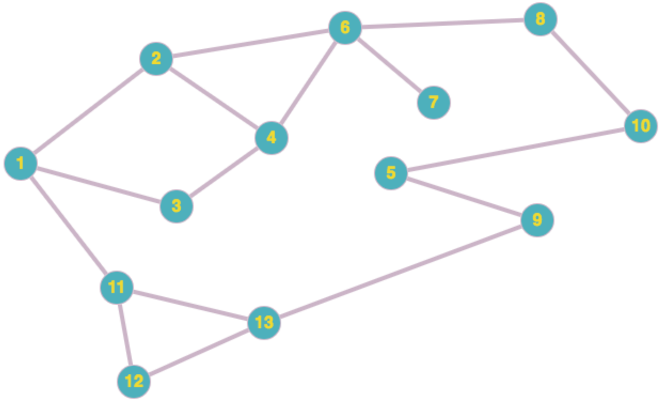
Computational Complexity of DFS

The computational complexity of DFS(u) depends on representation methods

- Graph representation using adjacency matrix: $O(n^2)$, n is the number of vertices
- Graph representation using edge list: $O(nm)$, n is the number of vertices, m is the number of edges
- Graph representation using adjacency list: $O(\max(n, m))$, n is the number of vertices, m is the number of edges.

DFS Verification

Example 1: Verify DFS(1) for the graph below



DFS Verification

#	Stack	Traversed Vertices
1	1	1
2	1, 2	1, 2
3	1, 2, 4	1, 2, 4
4	1, 2, 4, 3	1, 2, 4, 3
5	1, 2, 4	1, 2, 4, 3
6	1, 2, 4, 6	1, 2, 4, 3, 6
7	1, 2, 4, 6, 7	1, 2, 4, 3, 6, 7
8	1, 2, 4, 6	1, 2, 4, 3, 6, 7
9	1, 2, 4, 6, 8	1, 2, 4, 3, 6, 7, 8
10	1, 2, 4, 6, 8, 10	1, 2, 4, 3, 6, 7, 8, 10
11	1, 2, 4, 6, 8, 10, 5	1, 2, 4, 3, 6, 7, 8, 10, 5
12	1, 2, 4, 6, 8, 10, 5, 9	1, 2, 4, 3, 6, 7, 8, 10, 5, 9
13	1, 2, 4, 6, 8, 10, 5, 9, 13	1, 2, 4, 3, 6, 7, 8, 10, 5, 9, 13
14	1, 2, 4, 6, 8, 10, 5, 9, 13, 11	1, 2, 4, 3, 6, 7, 8, 10, 5, 9, 13, 11
15	1, 2, 4, 6, 8, 10, 5, 9, 13, 11, 12	1, 2, 4, 3, 6, 7, 8, 10, 5, 9, 13, 11, 12
16-	Pop vertices out of the stack	

Result: 1, 2, 4, 3, 6, 7, 8, 10, 5, 9, 13, 11, 12

DFS Verification

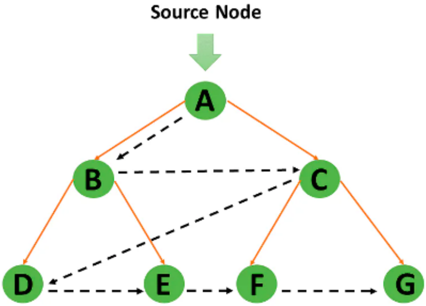
Given a graph with 13 vertices represented by the adjacency matrix below. Traverse the graph by applying DFS(1). Show the state of the stack and traversed vertices at each step.

$$
\begin{vmatrix}
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0
\end{vmatrix}
$$

Contents

Breadth-First Search (BFS)

Input: The information of a matrix (adjacency matrix, edge list, adjacency list)
Output: Traverse all nodes in the graph.



Breadth-First Search (BFS)

The breadth-first search (BFS) algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at a node of a graph and searches/visits all nodes at the current depth level before moving on to the nodes at the next depth level. The BFS algorithm consists of the following steps:

- To begin, move horizontally and visit all the current layer's nodes.
- Continue to the next layer.
  (https://www.cs.usfca.edu/ galles/visualization/BFS.html)

Breadth-First Search (BFS)

BFS(u){
    Step 1: Initialize
    $queue = \emptyset$; $push(queue, u)$; $unTraverse[u] = false$;
    Step 2: Loop
    while$(queue \neq \emptyset)${
        s = $pop(queue)$; <Visit s>;
        for$(t \in Adj(s))${
            if$(unTraverse[t])${
                $push(queue, t)$;$unTraverse[t] = false$;
            }
        }
    }
    Step 3: Return results
    return < set of traversed vertices >;
}

Computational Complexity of BFS

The computational complexity of BFS(u) depends on representation methods

- Graph representation using adjacency matrix: $O(n^2)$, n is the number of vertices
- Graph representation using edge list: $O(nm)$, n is the number of vertices, m is the number of edges
- Graph representation using adjacency list: $O(\max(n, m))$, n is the number of vertices, m is the number of edges.

BFS Verification

Given a graph with 13 vertices represented by the adjacency matrix below. Traverse the graph by applying BFS(1). Show the state of the queue and traversed vertices at each step.

$$
\begin{vmatrix}
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0
\end{vmatrix}
$$

BFS Verification

S#	Queue	Traversed Vertices
1	1	∅
2	2, 3, 4	1
3	3, 4, 6	1, 2
4	4, 6, 5	1, 2, 3
5	6, 5, 7	1, 2, 3, 4
6	5, 7, 12	1, 2, 3, 4, 6
7	7, 12, 8	1, 2, 3, 4, 6, 5
8	12, 8	1, 2, 3, 4, 6, 5, 7
9	8, 10	1, 2, 3, 4, 6, 5, 7, 12
10	10	1, 2, 3, 4, 6, 5, 7, 12, 8
11	9, 11, 13	1, 2, 3, 4, 6, 5, 7, 12, 8, 10
12	11, 13	1, 2, 3, 4, 6, 5, 7, 12, 8, 10, 9
13	13	1, 2, 3, 4, 6, 5, 7, 12, 8, 10, 9, 11
14	∅	1, 2, 3, 4, 6, 5, 7, 12, 8, 10, 9, 11, 13

Result: 1, 2, 3, 4, 6, 5, 7, 12, 8, 10, 9, 11, 13

BFS Verification

Given a graph with 13 vertices represented by the adjacency matrix below. Traverse the graph by applying BFS(1). Show the state of the stack and traversed vertices at each step.

$$
\begin{vmatrix}
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0
\end{vmatrix}
$$

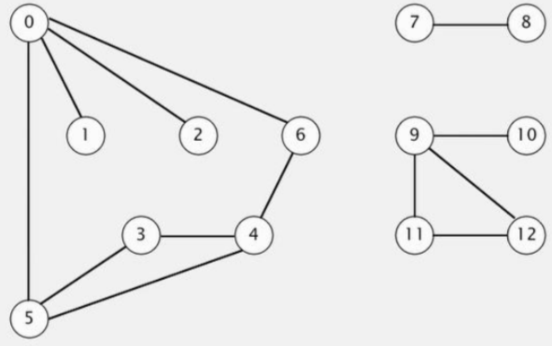
NOTES

- Undirected graph: If DFS(u) $= V$ or BFS(u) $= V$, the graph is connected
- Directed graph: If DFS(u) $= V$ or BFS(u) $= V$, the graph is weakly connected

Contents

Determine the number of connected components

- **Problem statement**: Given an undirected graph $G =< V, E >$, with the set of vertices V, and the set of edges E. Determine connected components of G?
- **Example**:

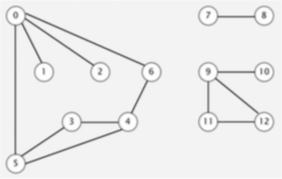


Determine the number of connected components

Algorithm:

```
ConComp (){
        Step 1: Initialize
        count = 0; //number of connected components equals to 0
        Step 2: Loop
        for(u ∈ V){ //for each vertex
                if( unTraverse[u]){
                        count = count + 1;//a connected component
                        BFS(u); // or DFS(u)
                        <store vertices of the connected component>;
                }
        }
        Step 3: Return result
        return <connected components>;
}
```

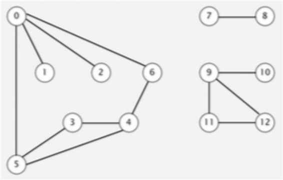
Determine the number of connected components




```
So dinh do thi: 13
 0  1  1  0  0  1  1  0  0  0  0  0  0
 1  0  0  0  0  0  0  0  0  0  0  0  0
 1  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  1  1  0  0  0  0  0  0  0
 0  0  0  1  0  1  1  0  0  0  0  0  0
 1  0  0  1  1  0  0  0  0  0  0  0  0
 1  0  0  0  1  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  1  0  0  0  0
 0  0  0  0  0  0  0  1  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  1  1  1
 0  0  0  0  0  0  0  0  0  1  0  0  0
 0  0  0  0  0  0  0  0  0  1  0  0  1
 0  0  0  0  0  0  0  0  0  1  0  1  0
TP. lien thong theo DFS 1:  0  1  2  5  3  4  6
TP. lien thong theo DFS 2:  7  8
TP. lien thong theo DFS 3:  9 10 11 12
```

DFS

Determine the number of connected components




```
So dinh do thi: 13
 0  1  1  0  0  1  1  0  0  0  0  0  0
 1  0  0  0  0  0  0  0  0  0  0  0  0
 1  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  1  1  0  0  0  0  0  0  0
 0  0  0  1  0  1  1  0  0  0  0  0  0
 1  0  0  1  1  0  0  0  0  0  0  0  0
 1  0  0  0  1  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  1  0  0  0  0
 0  0  0  0  0  0  0  1  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  1  1  1
 0  0  0  0  0  0  0  0  0  1  0  0  0
 0  0  0  0  0  0  0  0  0  1  0  0  1
 0  0  0  0  0  0  0  0  0  1  0  1  0
TP. lien thong theo BFS 1:  0  1  2  5  6  3  4
TP. lien thong theo BFS 2:  7  8
TP. lien thong theo BFS 3:  9 10 11 12
```

BFS

Determine the number of connected components

Exercise: Given an undirected graph represented by the adjacency matrix below. Determine connected components of the graph?

$$
\begin{vmatrix}
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0
\end{vmatrix}
$$

Finding paths between vertices

Exercise: Given an undirected graph represented by the adjacency matrix below. Determine connected components of the graph?

$$
\begin{vmatrix}
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0
\end{vmatrix}
$$

Finding paths between vertices

Problem statement: Given graph $G = \langle V, E \rangle$ (undirected or directed), with the set of vertices V and the set of edges E. Find a path from $s \in V$ to $t \in V$?

Algorithm description:

- If $t \in DFS(s)$ or $t \in BFS(s)$, there exists a path from s to t, otherwise, there is no path.

- To restore the path we use array $previous[]$ consisting of n elements ($n = |V|$).
  - Initialize $previous[u] = 0$ for all u
  - When push $v \in Adj(u)$ to the stack (DFS) or queue (BFS) we set $previous[v] = u$
  - If DFS and BFS cannot reach to t, $previous[t] = 0$, there is no path from s to t

Finding paths between vertices using DFS

```
DFS(s){
        Step 1: Initialize
        stack = ∅; push(stack, s); unTraverse[s] = false;
        Step 2: Loop
        while(stack ≠ ∅){
                u = pop(stack);
                for(v ∈ Adj(u)){
                        if( unTraverse[v]){ //v has not been traversed
                                unTraverse [v] = false; //v has been
traversed
                                push(stack, u); //push u to the stack
                                push(stack, v); //push v to the stack
                                previous[v] = u;
                                beak; //process one vertex only
                        }
                }
        }
        Step 3: Return result
        return <set of traversed vertices>;
```

Finding paths between vertices using BFS

```
BFS(s){
        Step 1: Initialize
        queue = ∅; push(queue, s); unTraverse[s] = false;
        Step 2: Loop
        while(queue ≠ ∅){
                u = pop(queue);
                for(v ∈ Adj(u)){
                        if( unTraverse[v]){
                                push(queue, v);
                                unTraverse[v] = false;
                                previous[v] = u;
                        }
                }
        }
        Step 3: Return result
        return <set of traversed vertices>;
}
```

Finding paths between vertices

Path restore

```
PathRestore(s, t){
        if( previous[t] == 0){
                <No path from s to t>;
        }
        else{
                <print out vertex t>;
                u = previous[t]; // u is the previous of t
                while(u ≠ s){
                        <print out vertex u>;
                        u = previous[u]; // trace back to the previous of u
                }
                <print out vertex s>;
        }
}
```

Finding paths between vertices

Exercise: Given a graph with 13 vertices represented by the adjacency matrix below. Find a path from vertex 1 to vertex 13?

$$
\begin{vmatrix}
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0
\end{vmatrix}
$$

Strongly Connected Property of Directed Graph

Problem statement: Directed graph $G =< V, E >$ is strongly connected if there exists a path between two every vertices. Given directed graph $G =< V, E >$, check whether G is strongly connected or not?

Algorithm:

```
bool Strongly_Connected (G =< V, E >){ //check strongly connected property of G
        ReInit( ); // ∀u ∈ V: unTraverse[u] = true;
        for(u ∈ V){ //loop for every vertices
                if(BFS(u) ≠ V) // or DFS(u) ≠ V
                        return false; // not strongly connected
                else
                        ReInit(); // reinitialize array unTraverse[]
        }
        return true; // strongly connected
}
```

Strongly Connected Property of Directed Graph

Exercise: Given graph $G =< V, E >$ with 13 vertices represented by the adjacency matrix below. Check whether G is strongly connected or not?

Algorithm:

$$\begin{vmatrix}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0
\end{vmatrix}$$

Finding Cut Vertices

Problem statement: Vertex $u \in V$ of an undirected graph $G = \langle V, E >$ is a cut vertex if its deletion (with its boundary edges) increases the number of connected components of the graph. Given (connected) directed graph $G =< V, E >$, find all cut vertices of G ?

Algorithm:

```
Finding_Cut_Vertices (G =< V, E >){
        ReInit( ); // ∀u ∈ V: unTraverse[u] = true;
        for(u ∈ V){ //for each vertex u
                unTraverse[u] = false; //prohibit BFS or DFS reaching to u
                if(BFS(v) ≠ V\{u}) // or DFS(v) ≠ V\{u}
                        <u is a cut vertex>;
                ReInit(); //reinitialize array unTraverse[]
        }
}
```

Finding Cut Vertices

Exercise: Given graph $G =< V, E >$ with 13 vertices represented by the adjacency matrix below. Find all cut vertices of G?
Algorithm:

$$\begin{vmatrix}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0
\end{vmatrix}$$

Finding Bridges

Problem statement: Edge $e \in E$ of undirected graph $G =< V, E >$ is a bridge if its deletion increases the number of connected components G. Given (connected) undirected graph $G = \langle V, E \rangle$, finding all bridges of G?

Algorithm:

```
Finding_Bridges (G =< V, E >){
        ReInit( ); //∀u ∈ V: unTraverse[u] = true;
        for(e ∈ E){ //for each vertex of graph
                E = E \ {e}; //remove edge e from the graph
                if(BFS(1) ≠ V) // or DFS(1) ≠ V
                        <e is a bridge>;
                E = E ∪ {e}; // retrun edge e to the graph
                ReInit(); //reinitialize array unTraverse[]
        }
}
```

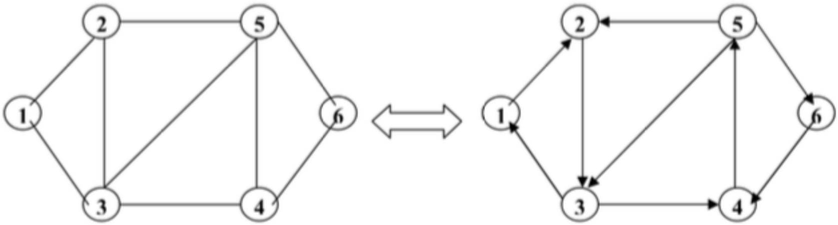
Finding Bridges

Exercise: Given graph $G =< V, E >$ with 13 vertices represented by the adjacency matrix below. Find all bridges of G?
Algorithm:

$$
\begin{vmatrix}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0
\end{vmatrix}
$$

Graph Orientation Problem

Problem statement: **Definition**: An **orientation** of an undirected graph is an assignment of a direction to each edge, turning the initial graph into a directed graph. A strong orientation is an orientation that results in a strongly connected graph. **Example**:



Graph Orientation Problem

Theorem: For any undirected graph $G = \langle V, E \rangle$, there exists a strong orientation on G if and only if all its edges are not bridge.

Some problems:

- Prove that there exists a strong orientation on an undirected graph
- Write a program to check whether exists a strong orientation on an undirected graph or not?
- Show a strong orientation on an undirected graph

Summary

- Depth first search algorithm from vertex $u \in V$, $DFS(u)$
- Breadth-first search algorithm form vertex $u \in V$, $BFS(u)$
- Applications of $DFS(u)$ and $BFS(u)$
  - Traverse all the vertices of a graph
  - Determine connected components of a graph
  - Find a path from vertex s to vertex t of a graph
  - Check the strongly connected property of a directed graph
  - Find all cut vertices of a graph
  - Find all bridges of a graph
  - Check whether exists a strong orientation on an undirected graph or not.

Exercises

Exercise 1. Given an undirected graph $G = \langle V, E \rangle$ consisting of 10 vertices is represented as an adjacency matrix as follows:

$$\begin{vmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{vmatrix}$$

a) Using the DFS algorithm to find the number of connected components of the graph G, specifying the result at each step of the algorithm?
b) Using the DFS algorithm find all the cut edges of the graph G, specifying the result at each step of the algorithm?

Exercise 2. Given an undirected graph $G = \langle V, E \rangle$ consisting of 10 vertices is represented as an adjacency matrix as follows:

$$\begin{vmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{vmatrix}$$

a) Use the BFS algorithm to find a path with the least number of edges from vertex 1 to vertex 7 of the graph G, specifying the result at each step performed by the algorithm?

b) Using the BFS algorithm find all the cut vertices of the graph G, specifying the result at each step of the algorithm?

Exercise 3. Given a directed graph $G = \langle V, E \rangle$ consisting of 10 vertices is represented as an adjacency list as follows:

$$
\begin{array}{ll}
\text{Adj}(1) = \{2, 3\} & \text{Adj}(6) = \{7, 8\} \\
\text{Adj}(2) = \{3, 4, 5\} & \text{Adj}(7) = \{4, 8\} \\
\text{Adj}(3) = \{9, 10\} & \text{Adj}(8) = \{1, 2\} \\
\text{Adj}(4) = \{6, 7\} & \text{Adj}(9) = \{6, 10\} \\
\text{Adj}(5) = \{6\} & \text{Adj}(10) = \{1, 2\}
\end{array}
$$

Use DFS to determine whether G is strongly connected, weakly connected, or disconnected? (Do not need to perform detailed steps of DFS algorithm, just write the results of execution)

Exercise 4. Given a directed graph $G = \langle V, E \rangle$ consisting of 10 vertices is represented as an adjacency matrix as follows:

$$\begin{vmatrix}
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{vmatrix}$$

Using breadth-first search to prove that G is strongly connected?