Acknowledgement

- Some of the contents are adapted from the work:

- Winter 2021: CPSC 319: Data Structures, Algorithms and their Applications by Jörg Denzinger

- Winter 2020: CPSC 319: Data Structures, Algorithms and their Applications by Dr. Jonathan Hudson

Graph Traversals

- Many problems can be described using graphs, where the solution to the problem requires that **we search the graph**, **looking for nodes (or paths) with a certain property**.

- Two important graph exploration techniques are

  - **breadth-first search**
  - **depth-first search**

Graph Traversals

- **Depth-First Search (DFS)**: like depth-first search in a tree, **we search as deeply as possible** by visiting a node, and then recursively performing depth-first search on each adjacent node.

https://www.youtube.com/watch?v=NUgMa5coCoE

- **Breadth-First Search (BFS)**: like breadth-first search in a tree, **we search as broadly as possible** by visiting a node, and then immediately visiting all nodes adjacent to that node.

https://www.youtube.com/watch?v=x-VTfcmrLEQ

DFS or BFS?

UNIVERSITY OF CALGARY

- **DFS traversal:**
  - **Complex connectivity questions**
    (Example: determine if every pair of vertices in a graph can be connected by two disjoint paths)
- **BFS traversal:**
  - **Finding shortest paths in a graph**
    (where distance is measured by the number of edges)

Depth-First Traversals

- Also referred as to *backtracking*
- Very similar to preorder traversal of a binary tree (node, left, right)
- **The traversal moves away from a current vertex as soon as possible**
- **Can be applied to directed and undirected graphs**

```
Depth_First_Search (VERTEX V)
{
Visit V;

Set the visit flag for the vertex V to TRUE;

For all adjacent vertices Vi (i = 1, 2, ….., n) of V
    if (Vi has not been previously visited)
        Depth_First_Search (Vi)
}
```

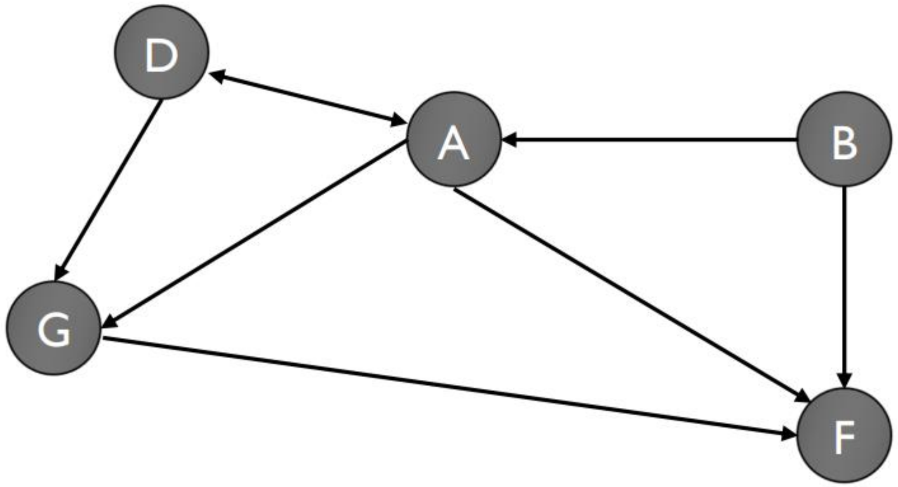
Depth-First Search/Traversal
(Adjacency Matrix initialized with 0)

```
int V; // total number of vertices

// weighted, undirect graph
int G[V][V];

// keep track of vertex visiting
int visited[V];

// visited is initialized to zero
for (int i  = 0; i < n; i++)
     visited[i] = 0; // i.e., false

// first call to DFS
// starting at vertex 0
DFS_AdjMat (0);
```

```
void DFS_AdjMat (int i)
{
       int j;
       visited[i] = 1; // i.e., true

       for(j = 0; j < V; j++)
       {
              if (!visited[j] && G[i][j] > 0)
                  DFS_AdjMat (j);
       }
}
```

Example

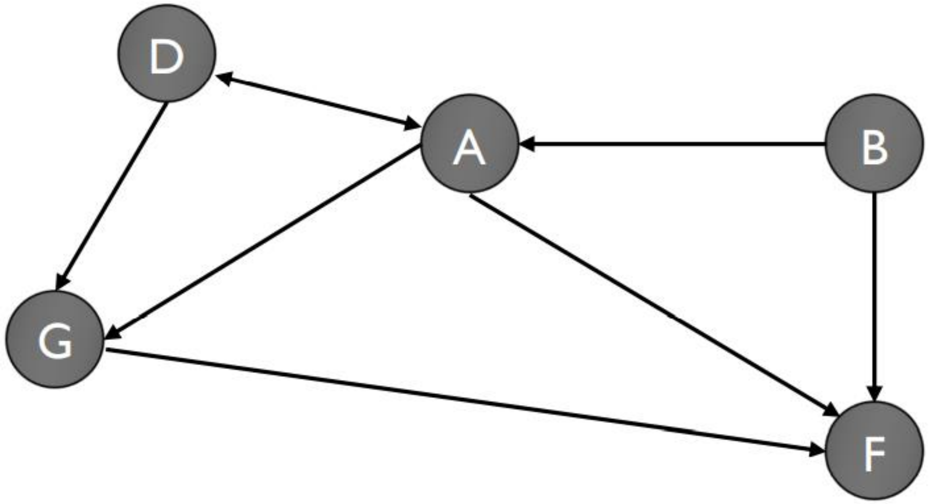
depthFirstSearch(A):



A-D-G-F

Example

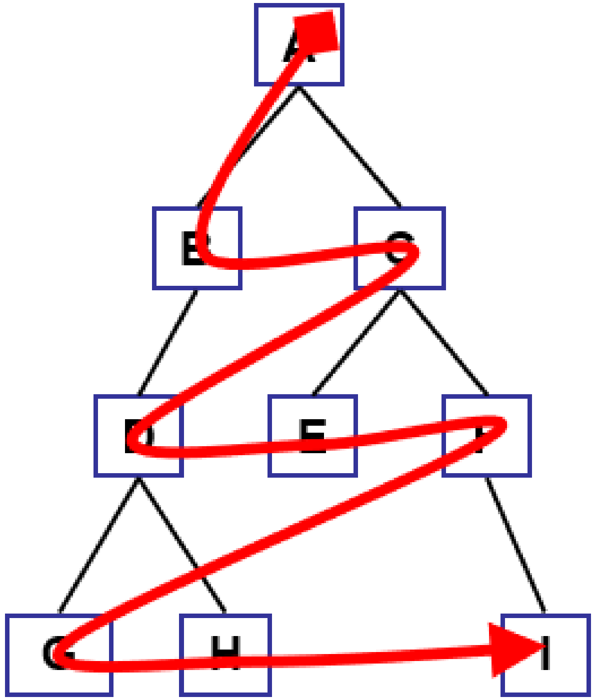
depthFirstSearch(B):



B-A-D-G-F

Breadth-First Traversals

- Very **similar** to **level-order** traversal of a binary tree



- Use a *queue* to track unvisited nodes
- For each node that is dequeued,
  - enqueue each of its children
  - until queue empty
- Also called: breadth first traversal

Breadth-First Traversals

- Very similar to level-order traversal of a binary tree

- **Visits a current vertex, say A, and all its adjacent vertices Ai (i = 1, …, p) if not visited previously, before visiting any adjacent vertices of Ai (i = 1, …, p)**

Algorithm for Breadth-First Traversal :

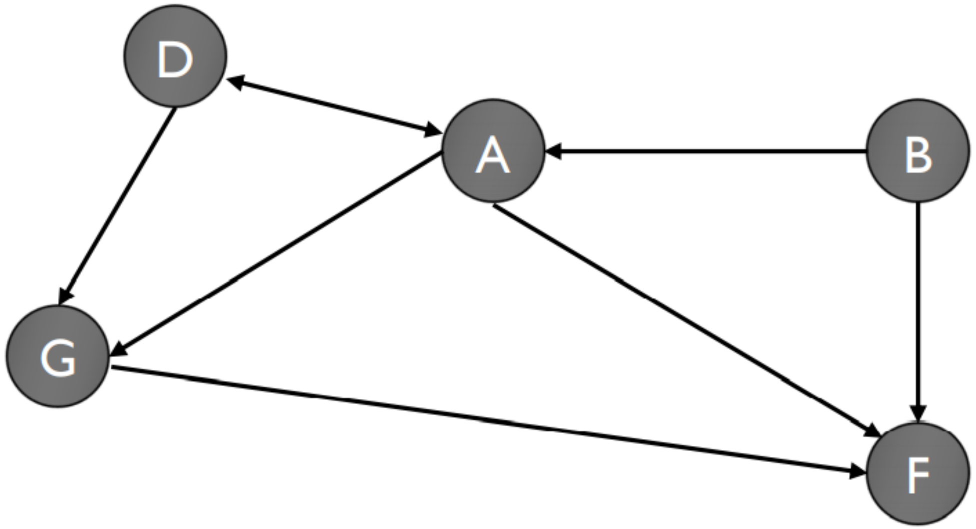
Starts with a vertex A, which has p number of adjacent vertices

1. Create and initialize queue object Que_obj
2. Initialize the queue object to be empty
3. For the set of n vertices, allocate an array object, visit[ ], of size n
4. Initialize visit[ ] by setting all its b entries to FALSE
5. Set visit[A] to TRUE
6. Visit the vertex A
7. Add the start vertex A to queue object, Que_obj
8. While “Que_obj is not empty”, do:
    1. Curr_start_vrtx = Dequeue(Que_obj)
    2. Get all vertices adjacent to Curr_start_vrtx
    3. For all vertices Ai adjacent to Curr_start_vrtx do:
        1. If visit[Ai] = FALSE
            1. visit[Ai] = TRUE
            2. Visit Ai
            3. Enqueue(Que_obj, Ai)
9. Deallocate and remove the queue object Que_obj
10. Deallocate and remove the array object visit[ ]
11. Return

Example

We order lexicographically the nodes in the out lists.

breadthFirstSearch(A):



queue:

A

Example

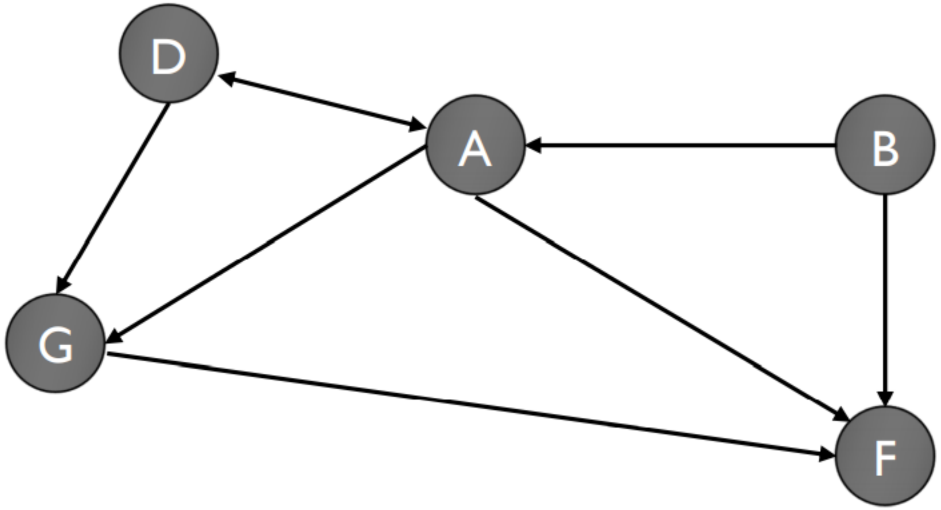
breadthFirstSearch(A):

queue:

D

F

G

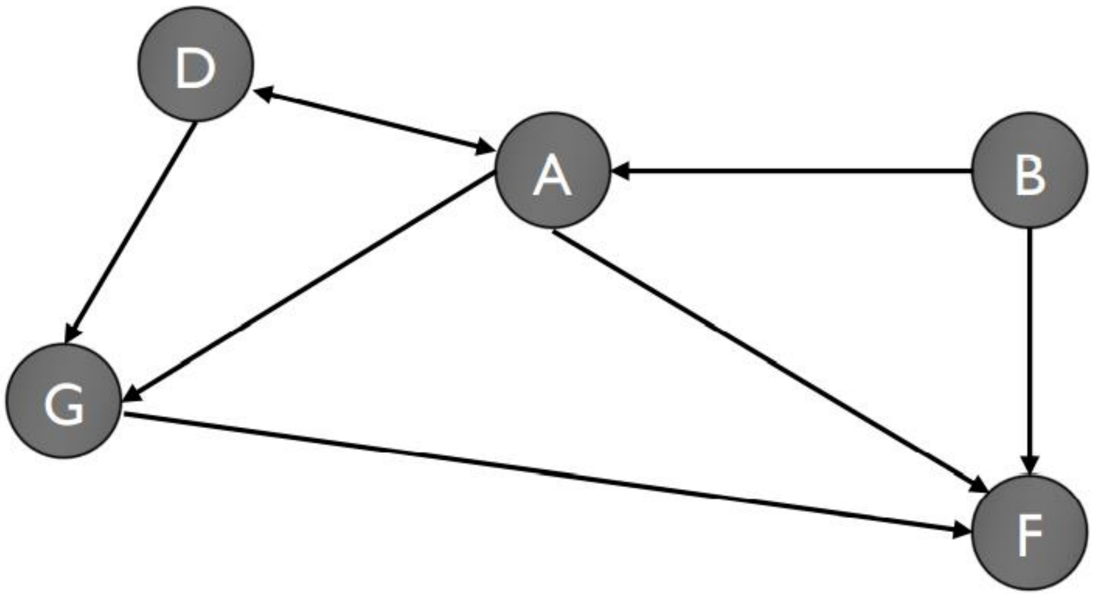


A

Example

breadthFirstSearch(A):

queue:

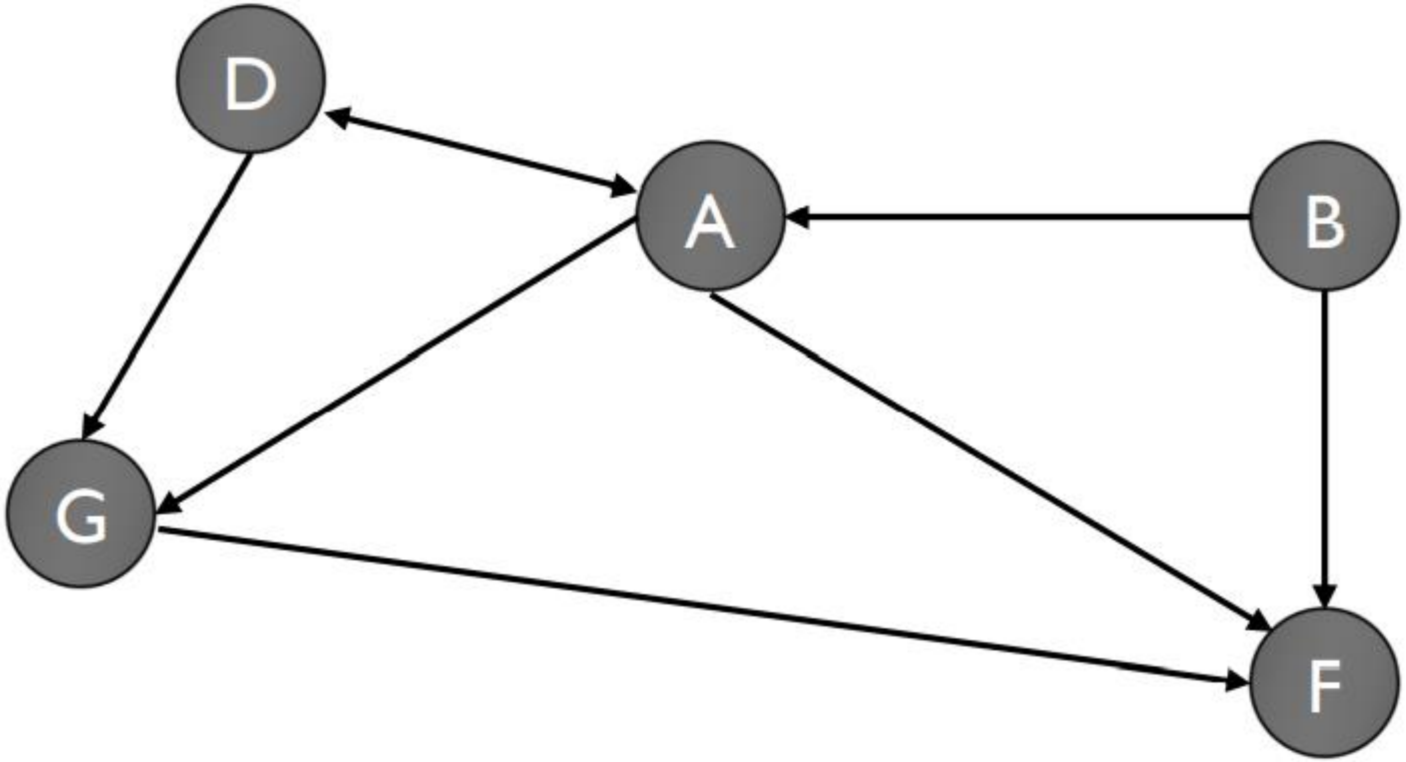


A-D-F-G

breadthFirstSearch(B):

queue:

B

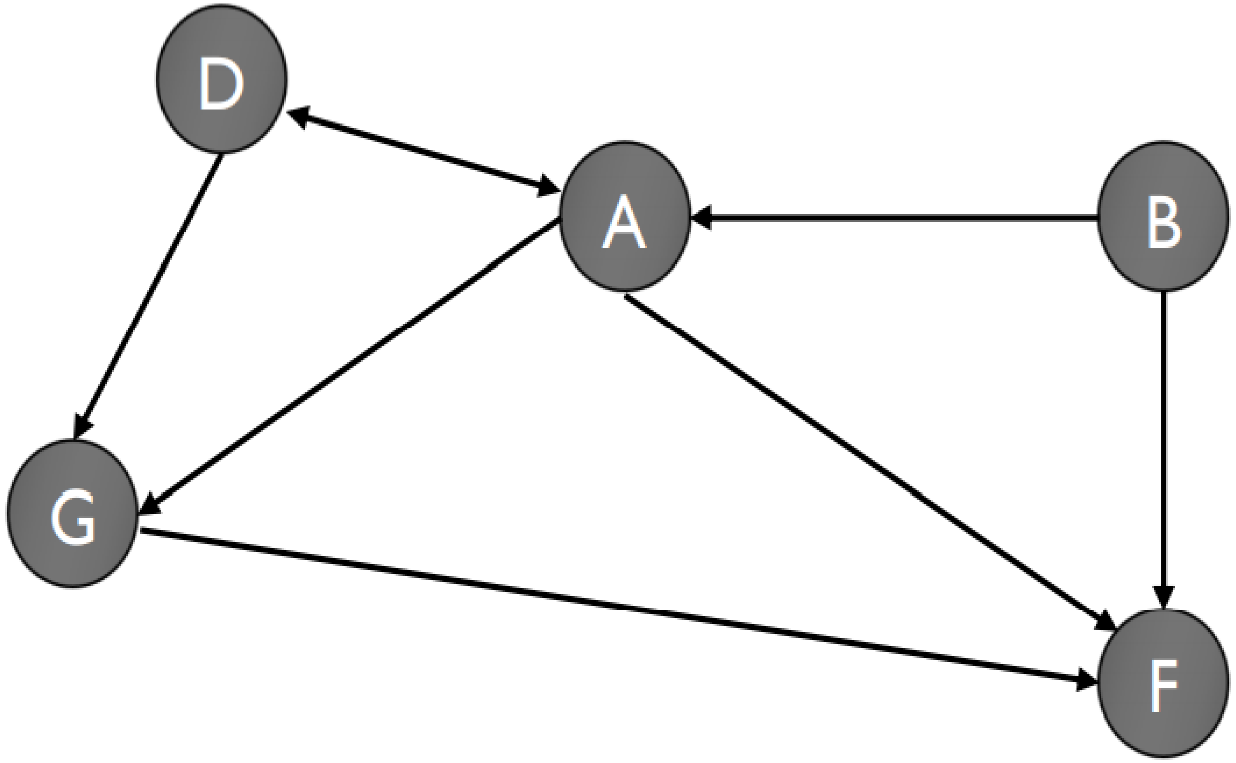


breadthFirstSearch(B):

queue:

A

F



B

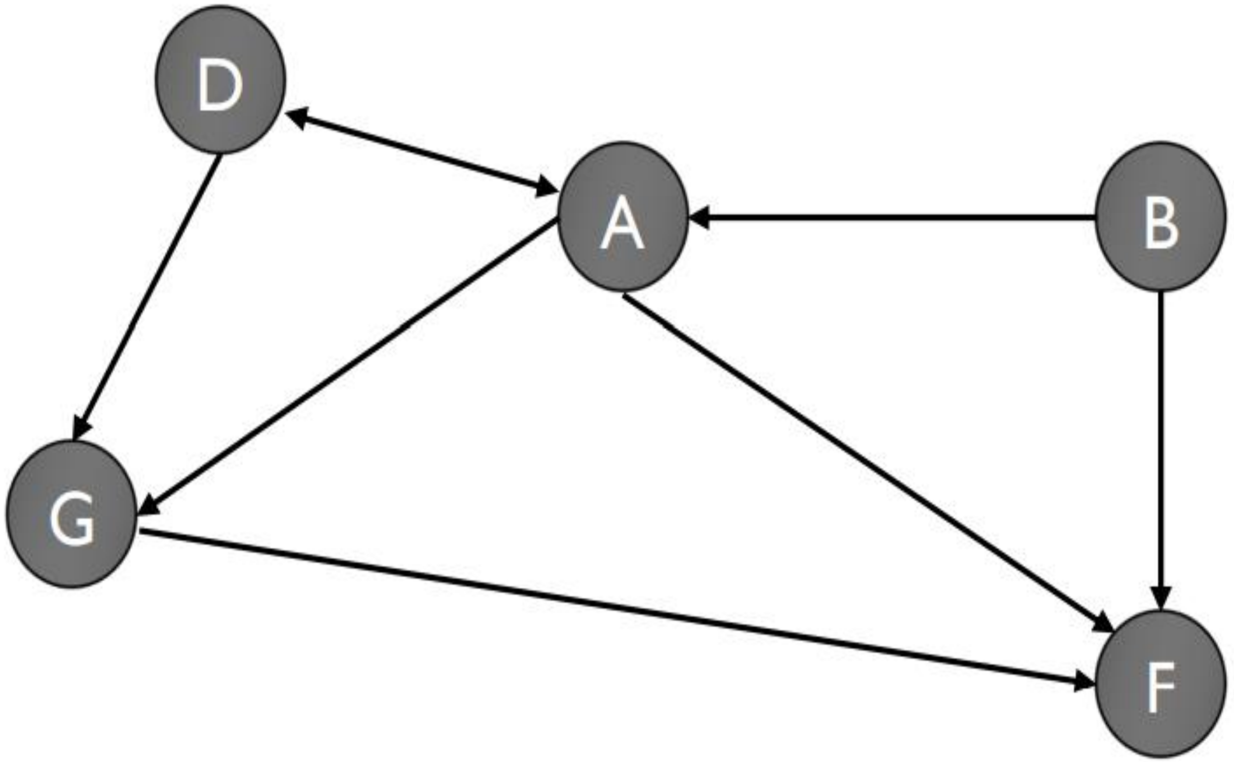
breadthFirstSearch(B):

queue:

F

D

G



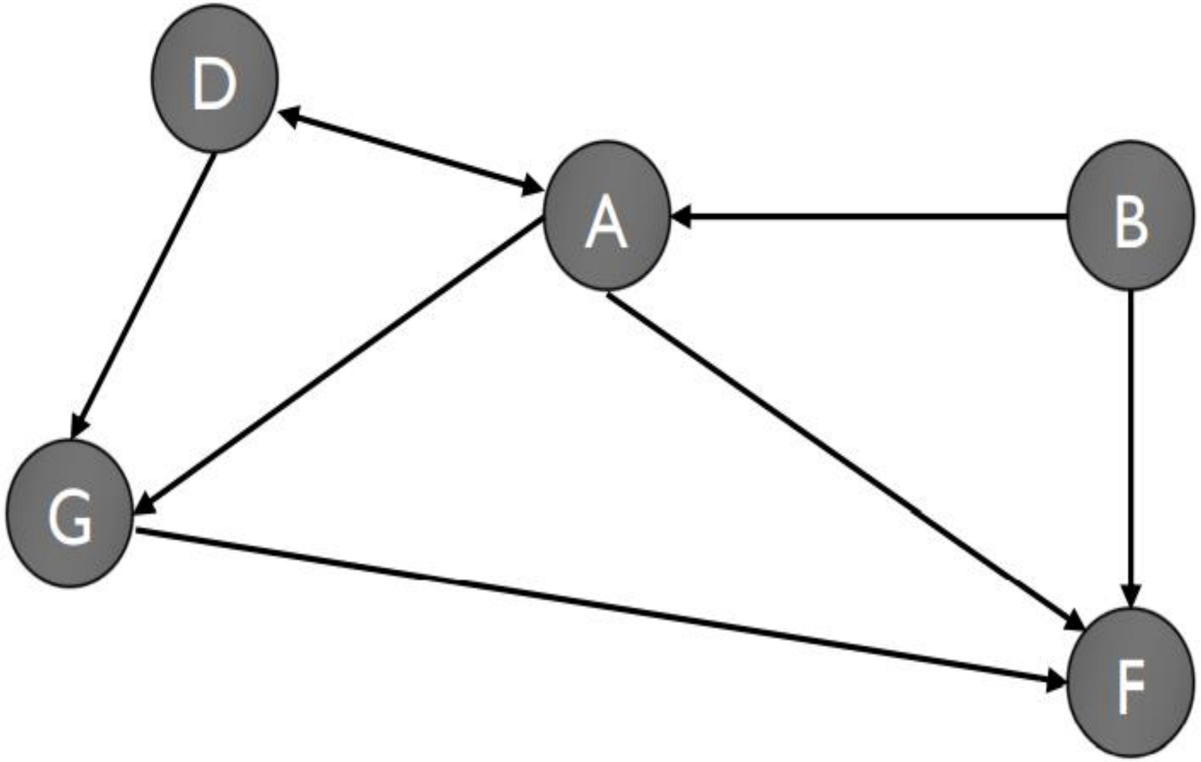
B - A

breadthFirstSearch(B):

queue:

D

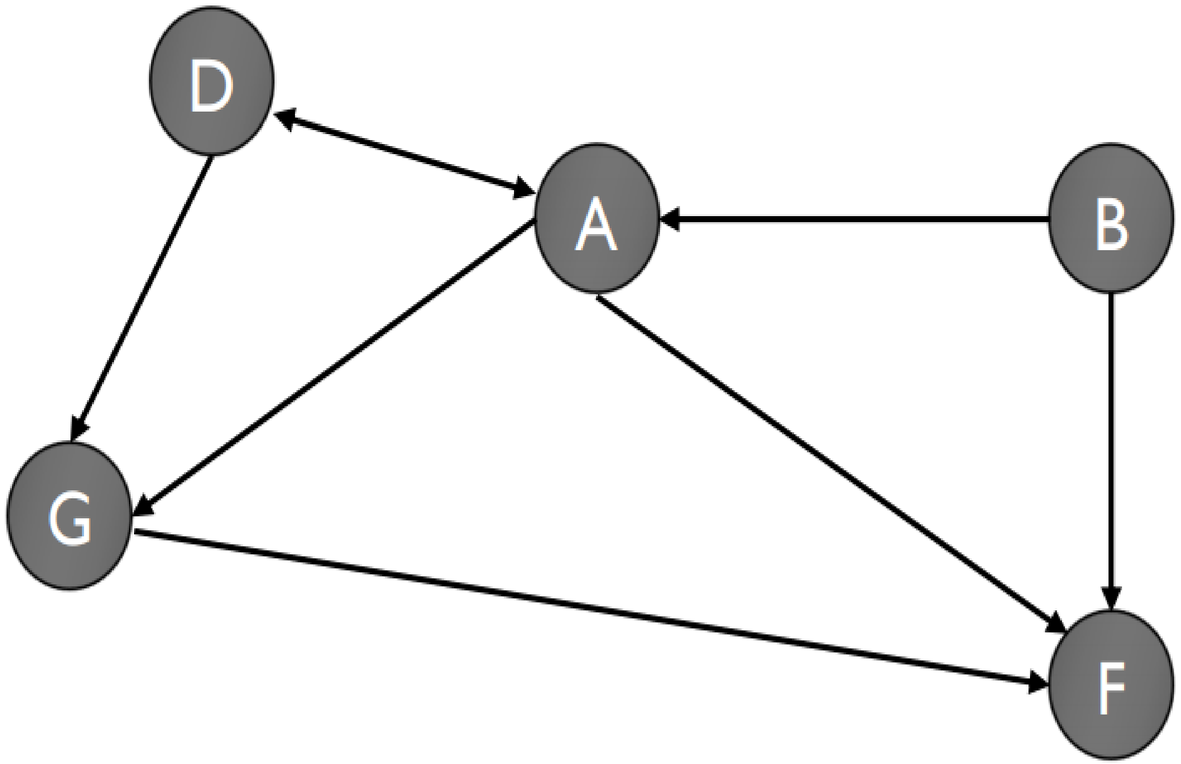
G



B - A - F

breadthFirstSearch(B):

queue:



B - A – F - D - G

Additional Resources

- https://www.cs.cornell.edu/courses/cs2112/2018fa/lectures/#traversals
- https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
- https://algs4.cs.princeton.edu/41graph/