UNIT 2:
Decrease and Conquer

Breadth First Search

Breadth First Search (BFS)

If depth-first search is a traversal for the brave,

breadth-first search is a traversal for the cautious.

Breadth First Search (BFS)

- Graph traversal algorithm.
- Uses decrease and conquer (decrease-by-one) strategy.
- Invented in 1945 by Konrad Zuse, in his (rejected) Ph.D. thesis on the Plankalkül programming language.
- Reinvented in 1959 by Edward F. Moore, who used it to find the shortest path out of a maze, and later developed by C. Y. Lee into a wire routing algorithm.

Working of BFS

- BFS proceeds in a concentric manner by visiting first all the vertices that are adjacent to a starting vertex
- Then all unvisited vertices two edges apart from it, and so on.
- It stops when all the vertices in the same connected component as the starting vertex are visited.
- If there still remain unvisited vertices, the algorithm has to be restarted at an arbitrary vertex of another connected component of the graph

Note: tie can be resolved arbitrarily, may also depends on data structure representing the graph

BFS uses Queue to trace the operation!

- Queue is initialized with the traversal's starting vertex, which is marked as visited.
- On each iteration, the algorithm identifies all unvisited vertices that are adjacent to the front vertex, marks them as visited, and adds them to the queue; after that, the front vertex is removed from the queue.

NOTE:
Unlike DFS, BFS has single ordering of vertices: i.e., Insertion order is same as the deletion order (FIFO)

BFS forest

- BFS traversal's starting vertex serves as the root of the first tree in BFS forest.

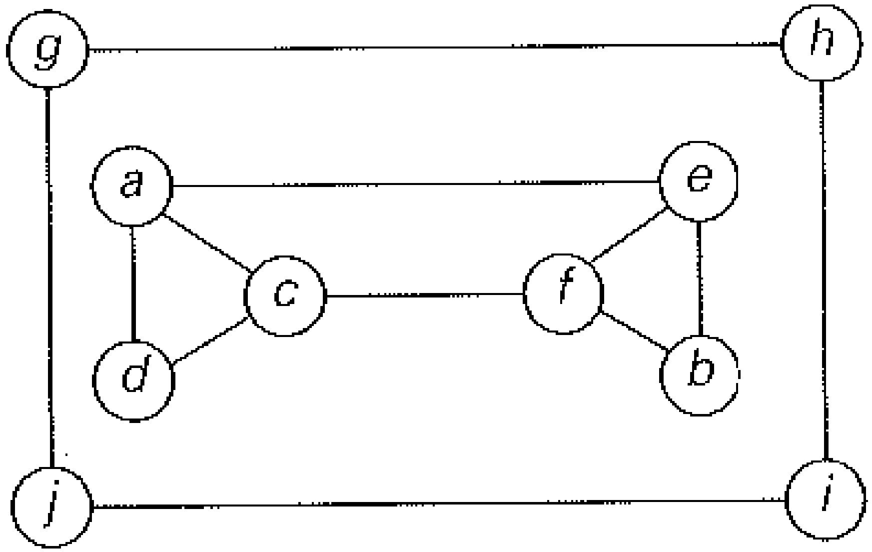
- **Tree edge**: Whenever a new unvisited vertex is reached for the first time, it is attached as a child to the vertex (using tree edge) from which it is being reached.

- **Cross edge**: Edge leading to a previously visited vertex other than its immediate predecessor (i.e., its parent in the tree).

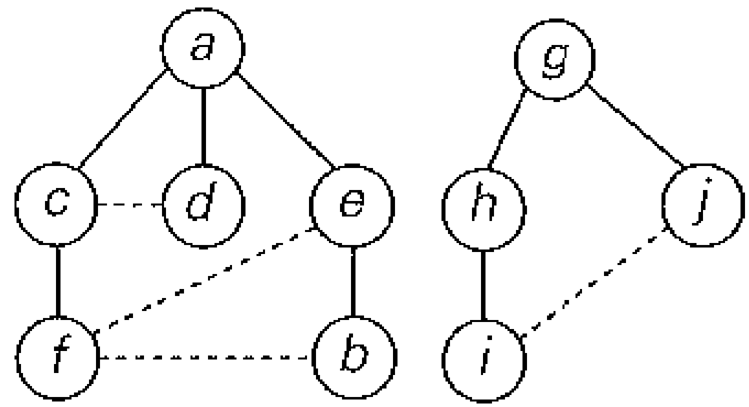
Breadth First Search

```
ALGORITHM  BFS(G)
    //Implements a breadth-first search traversal of a given graph
    //Input: Graph G = ⟨V, E⟩
    //Output: Graph G with its vertices marked with consecutive integers
    //in the order they have been visited by the BFS traversal
    mark each vertex in V with 0 as a mark of being "unvisited"
    count ← 0
    for each vertex v in V do
        if v is marked with 0
            bfs(v)

    bfs(v)
    //visits all the unvisited vertices connected to vertex v by a path
    //and assigns them the numbers in the order they are visited
    //via global variable count
    count ← count + 1;  mark v with count and initialize a queue with v
    while the queue is not empty do
        for each vertex w in V adjacent to the front vertex do
            if w is marked with 0
                count ← count + 1;  mark w with count
                add w to the queue
        remove the front vertex from the queue
```



(a) Graph.



(c) BFS forest

$a_1\ c_2\ d_3\ e_4\ f_5\ b_6$
$g_7\ h_8\ j_9\ i_{10}$

(b) Traversal's queue,

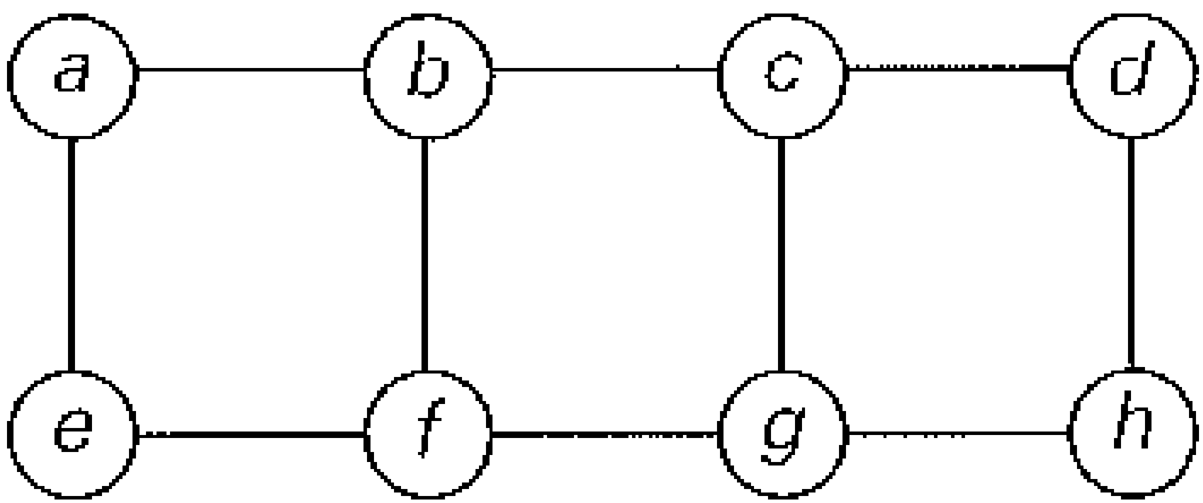
DFS/BFS: Visualization

https://visualgo.net/en/dfsbfs

https://www.cs.usfca.edu/~galles/visualization/BFS.html

Let's check our understanding...

Which algorithm (DFS/BFS) finds a path with the fewest number of edges between vertex **a** and **g**?



BFS algorithm analysis (same as DFS)

Running time is proportional to the size of the data structure used for representing the graph.

Efficiency for adjacent matrix	$\Theta(\lvert V^2 \rvert)$
Efficiency for adjacent lists	$\Theta(\lvert V \rvert + \lvert E \rvert)$

BFS: Applications

- checking connectivity, finding connected components
- checking acyclicity (if no cross edges)
- find paths from a vertex to all other vertices with the smallest number of edges

Algorithms that use breadth-first search as a building block:

- Copying garbage collection, Cheney's algorithm
- Finding the shortest path between two nodes u and v, with path length measured by number of edges (an advantage over depth-first search)
- (Reverse) Cuthill–McKee mesh numbering
- Ford–Fulkerson method for computing the maximum flow in a flow network
- Serialization/Deserialization of a binary tree vs serialization in sorted order, allows the tree to be re-constructed in an efficient manner.
- Construction of the failure function of the Aho-Corasick pattern matcher.
- Testing bipartiteness of a graph.

DFS vs BFS

	DFS	BFS
Data structure	stack	queue
No. of vertex orderings	2 orderings	1 ordering
Edge types (undirected graphs)	tree and back edges	tree and cross edges
Applications	connectivity, acyclicity, articulation points	connectivity, acyclicity, minimum-edge paths
Efficiency for adjacent matrix	$\Theta(\lvert V^2\rvert)$	$\Theta(\lvert V^2\rvert)$
Efficiency for adjacent lists	$\Theta(\lvert V\rvert + \lvert E\rvert)$	$\Theta(\lvert V\rvert + \lvert E\rvert)$

Next session…

Decrease and Conquer…

Topological sorting