GRAPH THEORY

6 - Graph Traversal Algorithms
Breadth-First-Search

Documents are here:

https://www-l2ti.univ-paris13.fr/~viennet/ens/2024-USTH-Graphs

Emmanuel Viennet
emmanuel.viennet@univ-paris13.fr

Graph : *review*

- *Number of nodes : N, number of edges (links) : L*
- *Neighbors of a node*
- *Incident link, indegree, outdegree*
- *path*
- *distance*
- *cycle*
- *connected*
- *tree*

Graph Traversal techniques

The previous connectivity problem, as well as many other graph problems, can be solved using *graph traversal techniques*

There are two standard graph traversal techniques:

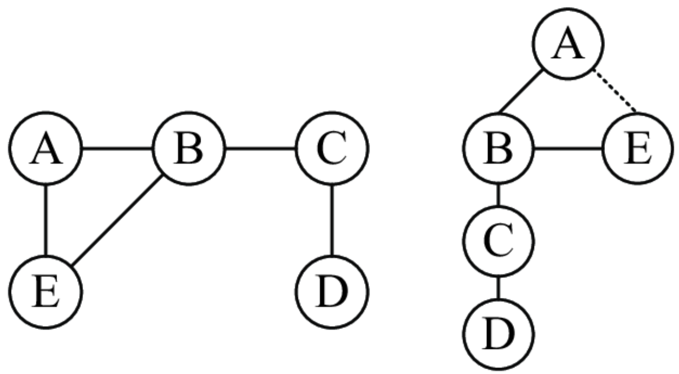
- *Depth-First Search* (DFS)
- *Breadth-First Search* (BFS)

Graph Traversal techniques (2)

In both DFS and BFS, the nodes of the undirected graph are visited in a systematic manner so that **every node is visited exactly one**.

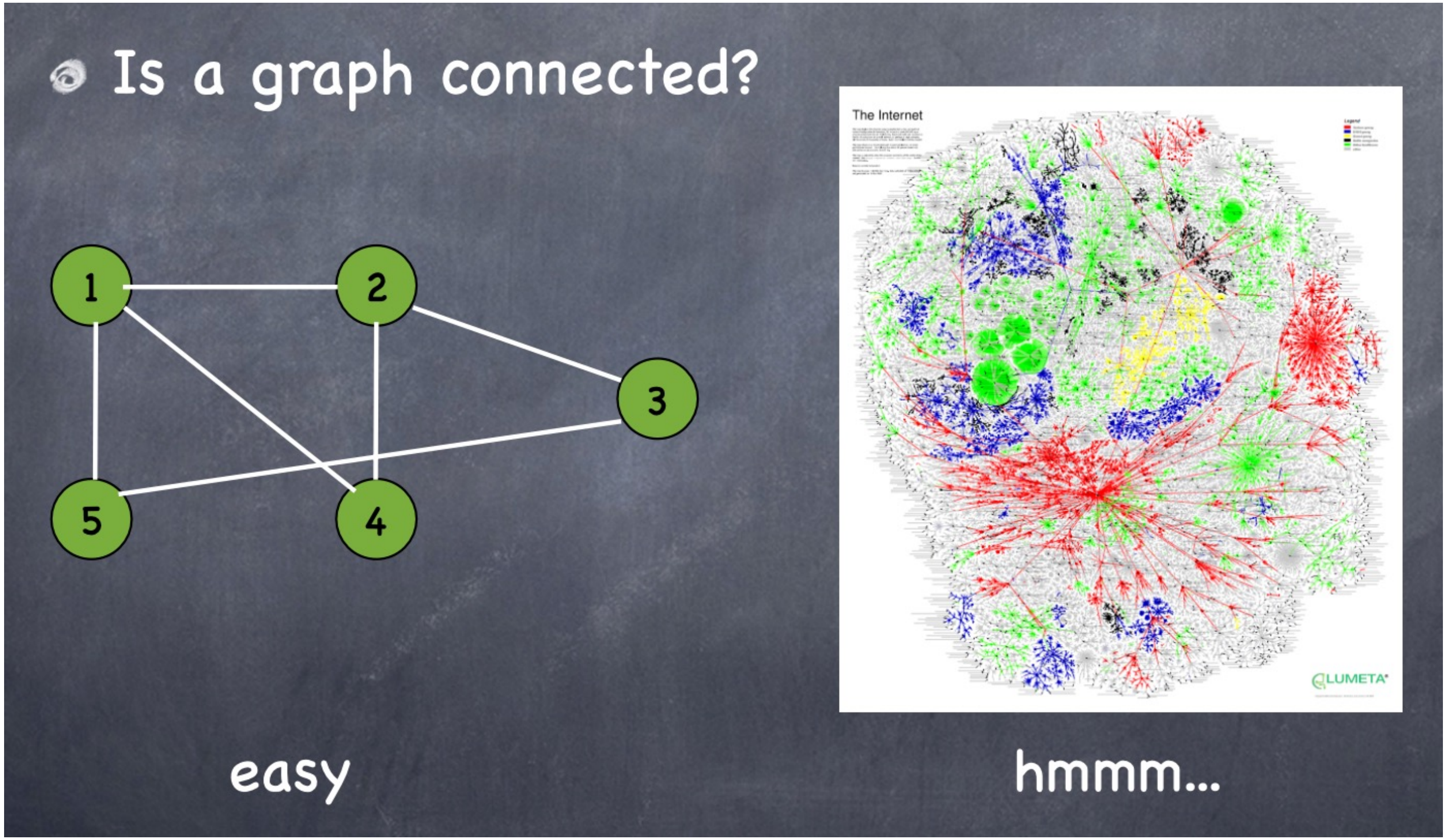
Both BFS and DFS give rise to a **tree**:

- When a node x is visited, it is labeled as visited, and it is added to the tree
- If the traversal got to node x from node y,
  y is viewed as the parent of x,
  and x a child of y



(a) undirected graph **(b)** DFS spanning tree

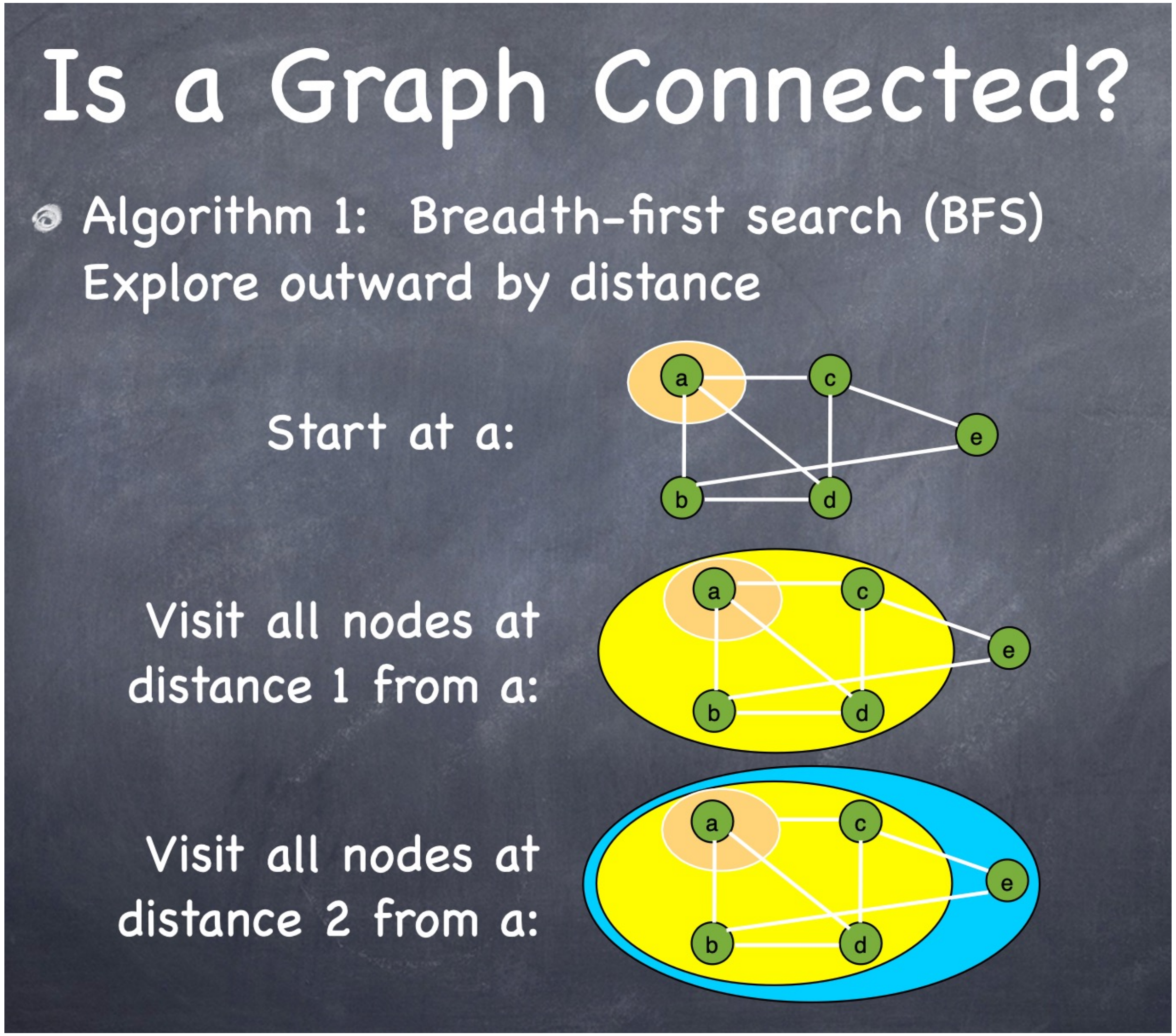
Graph Traversal



Graph Traversal

- Is a graph connected?
- Approach: explore outward from arbitrary starting node s to find all nodes reachable from s (connected component)

Graph Traversal



Breadth-First-Search (BFS)

Layers

- $L_0 = \{ s \}$.
- L_1 = all neighbors of L_0
- L_2 = nodes with edge to L_1 that do not belong to L_0 or L_1
- ...
- L_{i+1} = nodes with edge to L_i that do not belong to an earlier layer

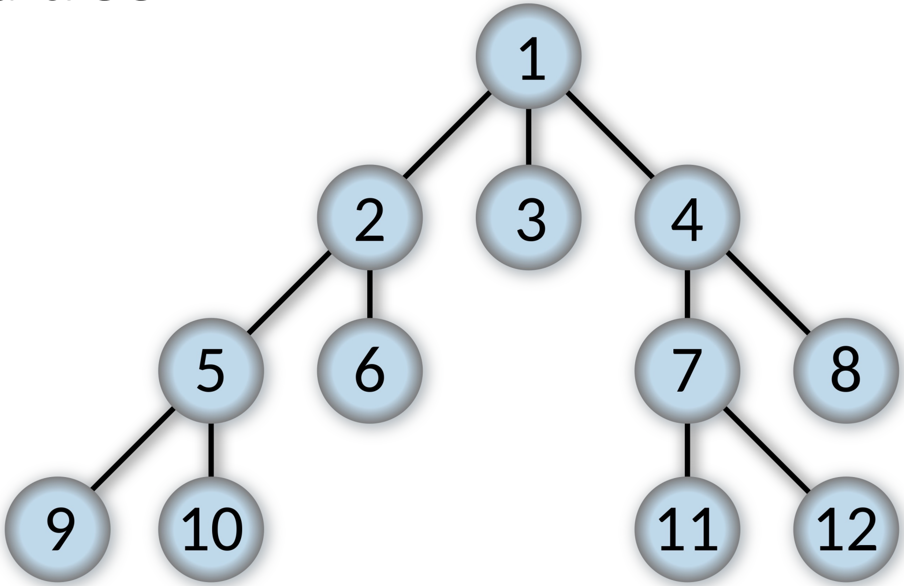
$$L_{i+1} = \{ v : \exists (u,v) \in E, u \in L_i, v \notin L_0 \cup \ldots \cup L_i \}$$

Observation:

- L_i consists of all nodes at distance exactly i from s. There is a path from s to t iff t appears in some layer.

BFS Tree

If we keep only the edges traversed while doing a breadth-first-search, we will have a tree



https://en.wikipedia.org/wiki/Breadth-first_search

BFS pseudo-code

very similar to DFS, but use a Queue

```
procedure BFS(G, root) is
    let Q be a queue
    label root as explored
    Q.enqueue(root)
    while Q is not empty do
        v := Q.dequeue()
        if v is the goal then
            return v
        for all edges from v to w in G.adjacentEdges(v) do
            if w is not labeled as explored then
                label w as explored
                w.parent := v
                Q.enqueue(w)
```

Depth-First Search (DFS)

Play: https://visualgo.net/en/dfsbfs