Graph Traversal Algorithms: DFS and BFS

CMSC 142: Design and Analysis of Algorithms

Department of Mathematics and Computer Science
College of Science
University of the Philippines Baguio

Overview

Graph Traversals

Depth-First Search (DFS)

1. DFS starts at an arbitrary vertex v of graph by marking it as visited.
2. On each iteration, DFS proceeds to an unvisited vertex that is adjacent to v. *If there are several such vertices, a tie can be resolved by the alphabetical order of the vertices.*

   The process continues until a dead end - a vertex with no adjacent unvisited vertices is encountered.
3. At a dead end, DFS backs up one edge to the vertex it came from and tries to continue visiting unvisited vertices from here.
4. The algorithm halts after backing up to the starting vertex, with the latter being a dead end.

   If unvisited vertices remain, DFS must be restarted at any one of them.

Graph Traversal: Depth-First Search

Implementation

A **stack** is used to trace the operation of DFS.

- Push a vertex onto stack when vertex is reached the first time.
- Pop a vertex off stack when it becomes a dead end.

Constructing DFS Forest

- The traversal's starting vertex serves as the root of the first tree in such a forest
- Whenever a new unvisited vertex is reached for the first time, it is attached as a child to the vertex (*tree edge*) from which it is being reached.

  An edge leading to a previously visited vertex other than its parent in the tree is a *back edge*.

Depth-First Search: Example

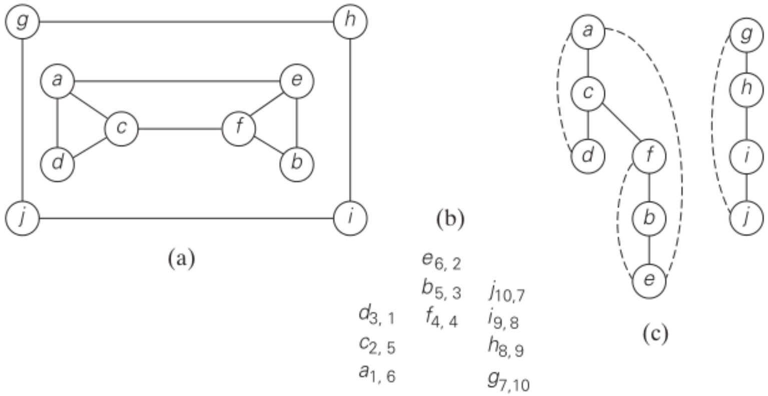
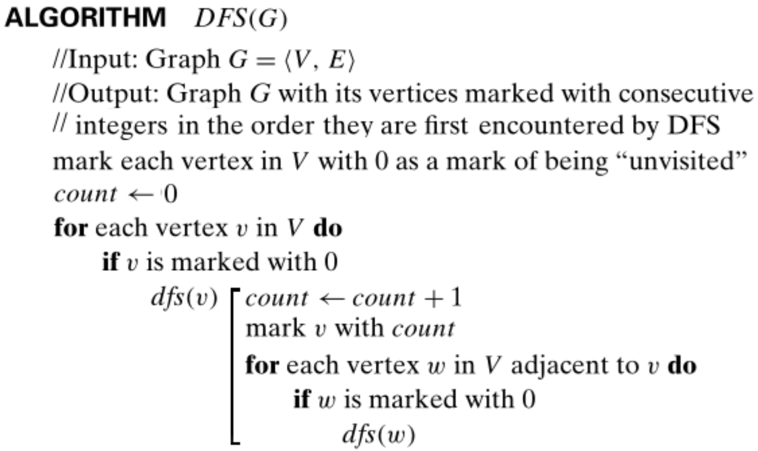


Figure: (a) Graph. (b) Traversal's stack (the first subscript indicates order in which vertex was pushed onto stack; second one indicates the order in which vertex was popped off the stack). (c) DFS forest (tree edges- solid lines; back edges-dashed lines).

Depth-First Search: Pseudocode



```
ALGORITHM  $DFS(G)$
    //Input: Graph $G = \langle V, E \rangle$
    //Output: Graph $G$ with its vertices marked with consecutive
    // integers in the order they are first encountered by DFS
    mark each vertex in $V$ with 0 as a mark of being “unvisited”
    $count \leftarrow 0$
    for each vertex $v$ in $V$ do
        if $v$ is marked with 0
            $dfs(v)$

$dfs(v)$
    $count \leftarrow count + 1$
    mark $v$ with $count$
    for each vertex $w$ in $V$ adjacent to $v$ do
        if $w$ is marked with 0
            $dfs(w)$
```

Breadth-First Search

1. BFS proceeds by visiting all the vertices that are adjacent to a starting vertex v
2. All unvisited vertices two edges apart from v are visited, followed by unvisited vertices three edges apart from v and so on are visited until all the vertices in the same connected component as the starting vertex are visited.

   If there are still remaining unvisited vertices, BFS has to be restarted at an arbitrary vertex of another connected component of the graph.

Graph Traversal: Breadth-First Search

Implementation

A **queue** is used to trace the operation of BFS.

- queue is initialized with starting vertex (marked as visited)
- On each iteration, BFS identifies all unvisited vertices that are adjacent to the *front vertex*, marks them as visited and adds them to the queue; after that front vertex is removed from the queue.

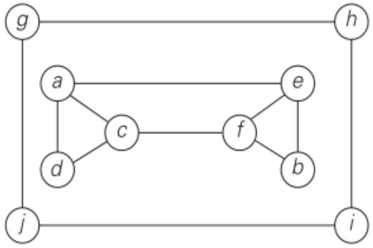
Graph Traversal: Breadth-First Search

Constructing BFS Forest

- The traversal's starting vertex v serves as the root of the first tree in such a forest.
- When a new unvisited vertex v_u is reached the first time, v_u is attached as child to vertex it is being reached from (*tree edge*).

  An edge leading to a previously visited vertex other than its parent is a *cross edge*.

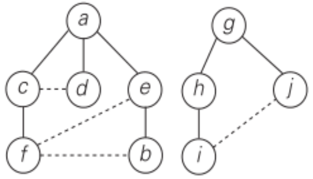
Breadth-First Search: Example



(a)

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

(b)



(c)

Figure: (a) Graph. (b) Traversal's queue, with the numbers indicating the order in which the vertices were visited. (c) BFS forest (with the tree edges shown with solid lines and the cross edges shown with dotted lines).

Breadth-First Search: Pseudocode

```
ALGORITHM  $BFS(G)$
//Input: Graph $G = \langle V, E \rangle$
//Output: Graph $G$ with its vertices marked with consecutive integers
//            in the order they are visited by the BFS traversal
mark each vertex in $V$ with 0 as a mark of being "unvisited"
$count \leftarrow 0$
for each vertex $v$ in $V$ do
    if $v$ is marked with 0
        $bfs(v)$
bfs(v)
    $count \leftarrow count + 1$;
    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
                $count \leftarrow count + 1$;   mark $w$ with $count$
                add $w$ to the queue
        remove the front vertex from the queue
```

Efficiency of DFS and BFS Traversals

- The efficiency of DFS and BFS algorithms is $\Theta(|V|^2)$ when graph is represented using its adjacency matrix representation.
- The efficiency of DFS and BFS is $\Theta(|V| + |E|)$ for adjacency list representation of graph.

Remark

(i) Important elementary applications of DFS and BFS include checking connectivity and checking acyclicity of a graph.

(ii) BFS can be used for finding a path with the fewest number of edges between two given vertices.

End of Lecture