Depth First Search

Arnab Ganguly, Assistant Professor
Department of Computer Science, University of Wisconsin – Whitewater
Data Structures (CS 223)

Simulation Weblink

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

1 Depth First Search

We consider a graph (directed or undirected, but unweighted) with N vertices and M edges. Like BFS, Depth-First-Search (DFS) is another graph exploration algorithm.

DFS starting from vertex s

- Initialize an array $level[\,]$ of length N that stores the DFS distances of vertices from s
- Initialize an empty set *closed* that keeps track of the vertices that have been closed by DFS so far (i.e., the vertices which have been popped from the stack at least once)
- Initially all entries of *level* is ∞ and *closed* is empty
- Initialize a stack of size M and push s. Set $level[s] = 0$.
- Repeat the following steps until the stack is empty
  - Pop vertex v from the stack.
  - If $v \in closed$ then
    - add v to *closed*
    - For each adjacent vertex w, if $w \notin closed$ then
      - $level[w] = level[v] + 1$
      - $push(w)$

1.1 Simulation

We simulate the DFS algorithm on the graph in Figure 1, as shown in Table 1.

Important: In case of DFS, a vertex can be pushed multiple times onto the stack (at most the number of incident/incoming edges). Hence, $level[v]$ can also be updated multiple times. This is in contrast to BFS, where a vertex is enqeued once and $level[v]$ is updated once.

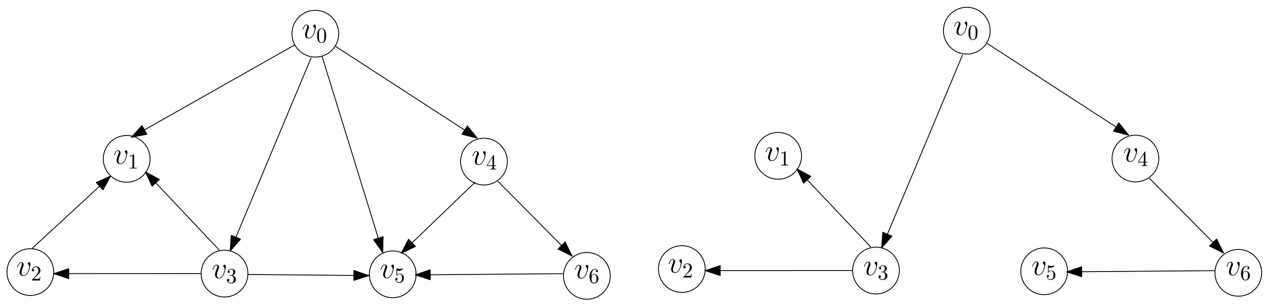

Figure 1: Graph used for DFS simulation and a DFS tree obtained

Table 1: Illustration of DFS

	closed	**stack**	**level array**						
			v_0	v_1	v_2	v_3	v_4	v_5	v_6
At Start	$\varnothing$	$[v_0]$	0	∞	∞	∞	∞	∞	∞
Pop (Relax v_0)	v_0	$[v_1, v_3, v_5, v_4]$	0	1	∞	1	1	1	∞
Pop (Relax v_4)	v_0, v_4	$[v_1, v_3, v_5, v_5, v_6]$	0	1	∞	1	1	2	2
Pop (Relax v_6)	v_0, v_4, v_6	$[v_1, v_3, v_5, v_5, v_5]$	0	1	∞	1	1	3	2
Pop (Relax v_5)	v_0, v_4, v_6, v_5	$[v_1, v_3, v_5, v_5]$	0	1	∞	1	1	3	2
Pop (Relax v_5)	v_0, v_4, v_6, v_5	$[v_1, v_3, v_5]$	0	1	∞	1	1	3	2
Pop (Relax v_5)	v_0, v_4, v_6, v_5	$[v_1, v_3]$	0	1	∞	1	1	3	2
Pop (Relax v_3)	v_0, v_4, v_6, v_5, v_3	$[v_1, v_1, v_2]$	0	2	2	1	1	3	2
Pop (Relax v_2)	$v_0, v_4, v_6, v_5, v_3, v_2$	$[v_1, v_1, v_1]$	0	3	2	1	1	3	2
Pop (Relax v_1)	$v_0, v_4, v_6, v_5, v_3, v_2, v_1$	$[v_1, v_1]$	0	3	2	1	1	3	2
Pop (Relax v_1)	$v_0, v_4, v_6, v_5, v_3, v_2, v_1$	$[v_1]$	0	3	2	1	1	3	2
Pop (Relax v_1)	$v_0, v_4, v_6, v_5, v_3, v_2, v_1$	$[\,]$	0	3	2	1	1	3	2

1.2 Depth First Tree

As in the case of BFS, if we execute the DFS algorithm on a graph $\mathcal{G}$ from a vertex s, then we obtain a tree on the same set of vertices of $\mathcal{G}$. This tree is called the DFS tree.

Obtaining the tree is achieved simply by modifying the DFS algorithm as follows:

DFS tree

- Start with only the set of vertices of $\mathcal{G}$, with no edge connecting any two vertices.
- Execute the DFS algorithm by making the following minor modification.
- Suppose a vertex v is popped from the stack, following which an unclosed vertex w is added to the stack. First remove any edge that is incident on w in the tree. Add an edge from v to w in the tree.

1.3 Complexity

In each of the algorithms discussed in the notes, an edge or a vertex is processed only a constant number of times. Hence the complexity on a graph with N vertices and M edges is $O(N + M)$.