15-750: Graduate Algorithms February 6, 2017

Lecture 9: Graph Algorithms: Depth-First Search

Lecturer: David Witmer *Scribe: Klas Leino*

1 Depth-first search basics

Depth-first search (DFS) is an algorithm for searching the vertices of a graph, that works by exploring as far into the graph as possible before backtracking. In the following sections, we consider the problem of searching a directed graph using DFS. An example of a possible DFS is shown in figure 1. We will periodically refer to this example, a variant of an example from [CLRS09] Section 22.3, throughout the lecture.

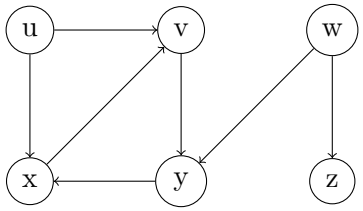

Figure 1: An example directed graph, G. A possible DFS of this graph would be $(u \to v \to y \to x), (w \to z)$. As we go as far as we can without before backtracking, we would not search an order such as $(u \to v \to x)$, as we would in some searching algorithms such as breadth-first search.

1.1 Input

Depth-first search takes as input a directed graph, $G = (V, E)$, where V is the vertex set and E is the edge set. For convenience, we suppose we are given some representation of adjacent vertices, e.g., an adjacency map, that can give us a list representation of the adjacent vertices, $adj(v)$ to a given vertex, v.

1.2 Colors

It is useful to define the following "colors" for vertices in order to keep track of the state of the algorithm:

- **White**: vertices are white before they are discovered.
- **Gray**: vertices are gray when they are discovered but their outgoing edges are still in the process of being explored.
- **Black**: vertices are black when the entire subtree starting at the vertex has been explored.

Let $color(u)$ be the color of vertex u.

1.3 Events

We also find it useful to maintain a timeline of the search. We keep track of a time at each step, and the following events:

- $disc(u)$: the time at which we discover vertex u, i.e., the time it becomes gray.
- $finish(u)$: the time at which we finish exploring all edges out of vertex u, i.e., the time it becomes black.

In our example DFS from figure 1, the events would be as follows:

$disc(u) = 1$	$finish(v) = 7$
$disc(v) = 2$	$finish(u) = 8$
$disc(y) = 3$	$disc(w) = 9$
$disc(x) = 4$	$disc(z) = 10$
$finish(x) = 5$	$finish(z) = 11$
$finish(y) = 6$	$finish(w) = 12$

1.4 Algorithm

We define the depth-first search algorithm as follows:

Algorithm 1 Depth-first search

Input: $G = (V, E)$
Output: $DFS(G)$
  // Initialize the colors to white
  $\forall u \in V : color(u) := white$
  $time := 0$
  for all $u \in V$ **do**
    if $color(u) = white$ **then**
      $DFS\_visit(u)$
    end if
  end for

Input: $u \in V$
Output: $DFS\_visit(u)$
  $color(u) := gray$
  $time := time + 1$
  disc(u) := time
  for all $v \in adj(u)$ **do**
    if $color(v) = white$ **then**
      $DFS\_visit(v)$
    end if
  end for
  $color(u) := black$
  $time = time + 1$
  $finish(u) = time$

1.5 Run-time

Let $|V| = n$ and $|E| = m$. We see that we must loop through each vertex in the DFS routine, which takes at least $O(n)$ time. In DFS_visit, we also loop through all the out edges of u. We do this for each $u \in V$, thus the total time for this is

$$O\left(\sum_{v \in V} deg(v)\right) = O(|E|) = O(m),$$

where $deg(V)$ is the out-degree of v in G. Thus, the total cost must be

$$O(n + m).$$

2 Properties of DFS

Now we present some observations about DFS that we will find useful when developing algorithms based on DFS.

2.1 DFS forest

We note that a DFS of a graph forms a forest. The forest formed depends on the ordering of the vertices when we search them. The forest formed by the example DFS from our example in figure 1 is shown in figure 2. We say that a vertex, v, is a descendant of vertex u, with respect to some DFS search, written v *desc* u if v is a child of u in the corresponding DFS forest. We note the following claim:

Claim 2.1. (v *desc* u) $\iff$ (v *is discovered when* u *is gray*).

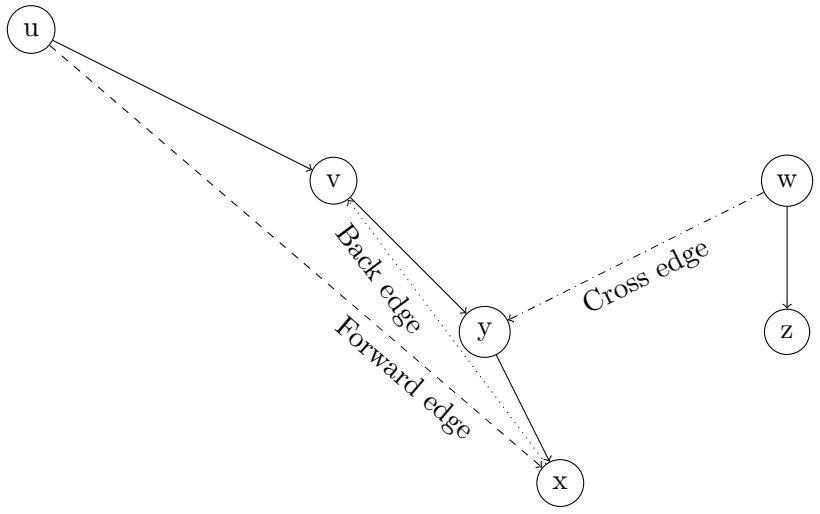

Figure 2: The DFS forest for the example search in figure 1. The edges in the forest are shown as solid lines — the dotted lines are other edges of the graph that are not part of the forest.

Given a DFS forest of G, we distinguish between four types of edges in the original graph; namely *tree edges* (shown as solid lines), *back edges* (shown as a dotted line), *forward edges* (shown as a dashed line), and *cross edges* (shown as a dashed dotted line). More formally,

Forest edges are those traversed in the DFS_visit routine.

Back edges are any edge, (u, v), such that u is a descendant of v.

Forward edges are any non-forest edge, (u, v), such that v is a descendant of u.

Cross edges are any other edge. These are the edges that go between subtrees or trees in the forest.

2.1.1 Color and edge types

When exploring edge, (u, v), u is gray. If v is white, then (u, v) is a tree edge as it will be explored in DFS_visit. If v is gray, we must still be searching in a subtree of v, thus u is a descendant of v, and (u, v) is a back edge. Finally, if v is black, then (u, v) is either a forward or cross edge.

2.2 Times and DFS forest structure

Let $int(u)$ of vertex u, be the interval $[disc(u), finish(u)]$, i.e., the time from the discovery of vertex u until it is colored black, for a given DFS.

Theorem 2.2. *$\forall u, v \in V$, exactly one of the following holds:*

(1) $int(u)$ and $int(v)$ are disjoint and neither of u and v is a descendant of the other, or

(2) $int(u) \subset int(v)$ and u is a descendent of v, or $int(v) \subset int(u)$ and v is a descendent of u. That is, the interval of u is nested in the interval of v and u is a descendent of v or vice versa.

Proof. WLOG, assume $disc(u) < disc(v)$.

Case: $disc(v) < finish(u)$

Thus, v was discovered when u was gray, as we know u was started but not finished. Therefore, v is a descendant of u by claim 2.1. But in our algorithm, the descendants must finish before their ancestors.

$\implies finish(v) < finish(u)$

$\implies int(v) \subset int(u)$

Case: $finish(u) < disc(v)$

Thus the intervals are disjoint, i.e., $int(u) \cap int(v) = \emptyset$. Each vertex is only gray during the time spanned by its interval, as it is white before it is discovered and black after it is finished. Thus neither vertex is discovered while the other was gray. Therefore, by claim 2.1, neither u nor v is the descendant of the other.

□

2.2.1 More time relationships

Consider edge, $(u, v) \in E$.

(u, v) is a forest or forward edge $\iff disc(u) < disc(v) < finish(v) < finish(u)$

(u, v) is a back edge $\iff disc(v) < disc(u) < finish(u) < finish(v)$

(u, v) is a cross edge $\iff disc(v) < finish(v) < disc(v) < finish(v)$

3 DFS and cycles

Theorem 3.1. *Graph, G, has a cycle $\iff$ every DFS has a back edge.*

Proof. We show this by proving both directions of the equivalence.

($\Leftarrow$) A back edge goes from a descendant in a DFS tree to its ancestor. As the descendant must be reachable from the ancestor, this forms a cycle. Therefore if a DFS has a back edge, G has a cycle.

($\Rightarrow$) Let $c = (v_1, v_2, ..., v_k)$, be a cycle in G. WLOG, we visit v_1 first, so $disc(v_1) < disc(v_k)$. As v_k is a descendant of v_1, it must also be the case that $finish(v_k) < finish(v_1)$, so we have that $disc(v_1) < disc(v_k) < finish(v_k) < finish(v_1)$.

$\implies (v_k, v_1)$ is a back edge

Therefore, if G has a cycle, every DFS will have a back edge.

□

Corollary 3.2. *If G is a DAG (directed, acyclic graph), and $(u, v) \in E$, then (u, v) cannot be a back edge, thus $finish(v) < finish(u)$.*

4 Topological sorting

Definition 4.1. On a DAG, $G = (V, E)$, an ordering on the vertices of V, $v_1, v_2, ..., v_n$, is a *topological sort* if

$$(v_i, v_j) \in E \implies i < j.$$

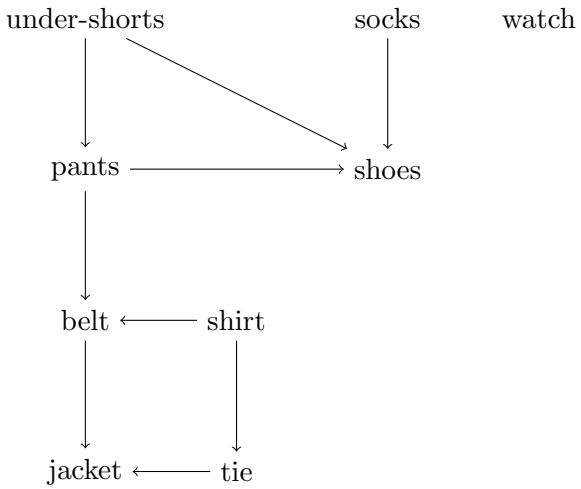

Figure 3: An example scenario (from [CLRS09] Section 22.4) involving topological sort. Consider the possible scenario of dependencies when getting dressed, shown in the figure by arrows, e.g., under-shorts must be put on before pants, and pants must be put on before shoes. A topological sort of this graph gives a valid ordering of putting on clothes that satisfies the dependencies. For example, (1) socks, (2) under-shorts, (3) pants, (4) shoes, (5) watch, (6) shirt, (7) belt, (8) tie, (9) jacket, would be a possible topological sort of the graph.

Theorem 4.2. *In a DAG, the reversed finish times of a DFS give a topological sort, i.e.,*

$$(u, v) \in E \implies f(u) > f(v).$$

Proof. This follows directly from corollary 3.2. □

Corollary 4.3. *Topological sort can be done in $O(n + m)$ time, as it only requires a DFS.*

References

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, third edition, 2009. 1, 3