2. Depth first search

Depth first search is a method for recursively traversing a graph that takes $\Theta(m + n)$ time. Whenever we visit a vertex, we visit its first neighbor. But before we visit its other neighbors, we visit the neighbor's neighbor, and so on. Effectively, we go "deeper" into the graph before finishing up business at the beginning.

Imagine you are reading the Wikipedia article on depth-first search.

Depth-first search

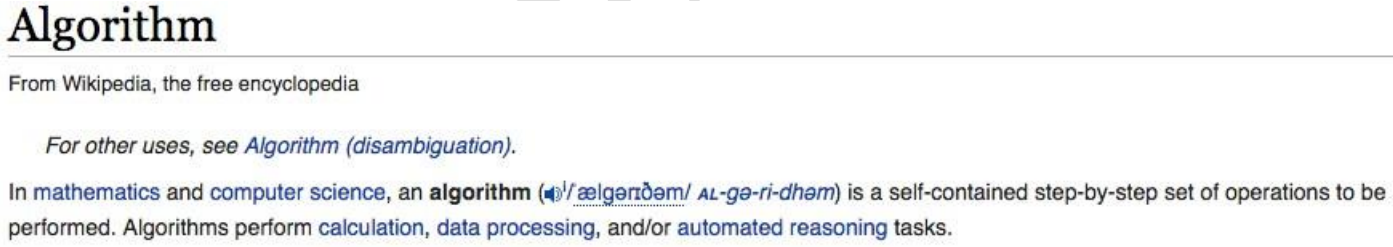
From Wikipedia, the free encyclopedia

This article **needs additional citations for verification**. Please help improve this article by ad
Unsourced material may be challenged and removed. *(July 2010) (Learn how and when to remove thi*

Depth-first search (**DFS**) is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking.

A version of depth-first search was investigated in the 19th century by French mathematician Charles Pierre Trémaux[1] as a strategy for solving mazes.[2][3]

If you wanted to traverse the link structure in a depth-first manner, you would get to the first link, which says "algorithm," and then click on it.



Algorithm

From Wikipedia, the free encyclopedia

For other uses, see Algorithm (disambiguation).

In mathematics and computer science, an **algorithm** (/ˈælgərɪðəm/ *AL-gə-ri-dhəm*) is a self-contained step-by-step set of operations to be performed. Algorithms perform calculation, data processing, and/or automated reasoning tasks.

Now the first link on this page is "mathematics," so you would immediately click on that to get to the "mathematics" page. You would keep doing this until there were no more unexplored links. (If you got to a link that you'd seen before, you'd just ignore it.)

Once there are no more unexplored links, you would go back to the "algorithm" page and then click on "computer science," and explore that part of the graph. Then you would click on "calculation," etc.

When you were done with the links on the "algorithm" page, you would go back to the "depth-first-search" page and click on "tree," and perform depth-first search on the "tree" page.

2.1 Algorithm

When we write this algorithm, we want to do it recursively, so that when we call depthfirst-search on a node, we call depth-first-search on each of its unvisited neighbors. Here we give each node a color, depending on its current state: white (the node is

unexplored), grey (which means the node was seen, but we are not finished processing its descendants yet), or black (which means we have processed the node and all of its descendants).

If we wanted to write pseudocode for this algorithm, it might look like

```
Let G be a graph with vertices that are all white DepthFirstSearch(graph G, node u):
    u.color = grey for each neighbor
    in Adj[u]:
        if neighbor.color == white:
            DepthFirstSearch(G, neighbor)
    u.color = black
```

You'll notice this DepthFirstSearch routine doesn't really do anything except color the vertices. If you wanted to use depth-first-search in real life, you would probably want to do something to a vertex every time you processed it. For instance, if you were using depthfirst-search to traverse the Wikipedia graph, you might want to save each article to your hard drive every time you colored that article black.

It might seem kind of useless to have separate colors for grey and black, because they are treated identically in the algorithm, but the vertices can fundamentally be in three different states, and having different names for different states helps us prove theorems about the algorithm.

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
 1  time = time + 1            // white vertex u has just been discovered
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]      // explore edge (u, v)
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK            // blacken u; it is finished
 9  time = time + 1
10  u.f = time
```

The full version of depth-first search keeps track of a few additional things. First, it keeps track of the parent node π of each vertex. The parent node of v is the node that triggers a visit to v (e.g. in the Wikipedia example, "depth-first-search" is the parent node of

"algorithm"). Knowing all the parent nodes allows us to reconstruct a "depth-first-search tree," where there is an edge from *u* to *v* if *u* is the parent of *v*.

It also keeps track of the "discovery" and "finishing" times of each vertex. The discovery time is the moment we color the vertex grey and start iterating through all of its neighbors. The finishing time is when we finish iterating through all the neighbors and color the vertex black.

Finally, it accounts for the possibility that not all vertices can be reached from the start node. If we finish exploring from the start node and there are still unexplored vertices left, we just repeat the process at one of the unexplored vertices, and we keep doing this until the graph is exhausted. Now the depth-first search tree becomes a depth-first search forest.

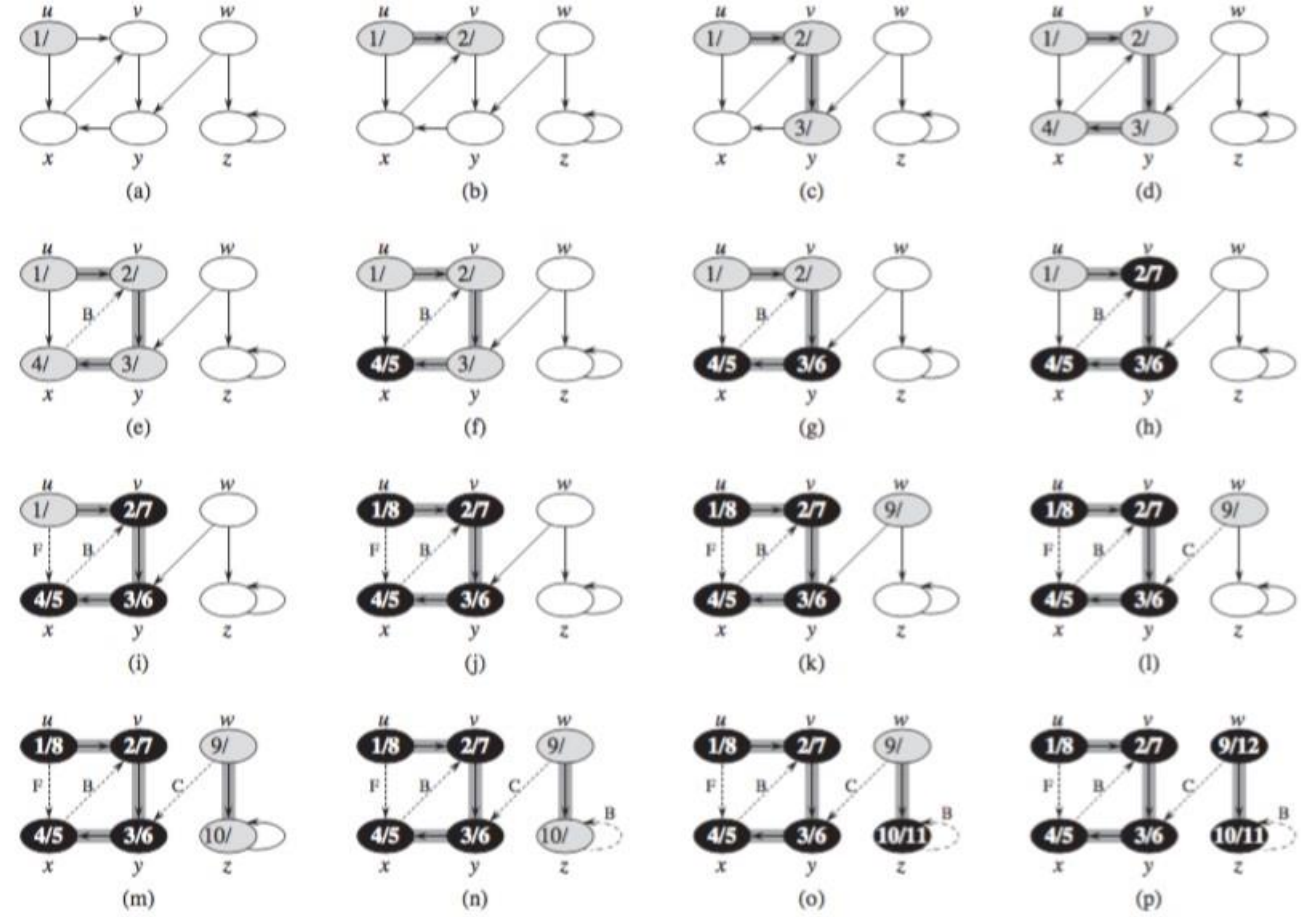


Figure 22.4 The progress of the depth-first-search algorithm DFS on a directed graph. As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise). Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges. Timestamps within vertices indicate discovery time/finishing times.

2.2 Runtime

The runtime of depth-first-search is $\Theta(m + n)$. The loops in DFS take $\Theta(n)$, plus the time taken to execute the calls to DFSVisit. DFSVisit is called exactly once for each vertex (taking $\Theta(n)$), since the vertex on which DFSVisit is invoked must be white, and the first thing DFSVisit does is paint that vertex grey. The loop inside DFSVisit gets executed $Adj[u]$ times for each vertex u, and the total number of entries in Adj (across all the vertices) is just the number of edges in the graph. So this adds $\Theta(m)$ to the runtime. **2.3 Properties of depth-first search**

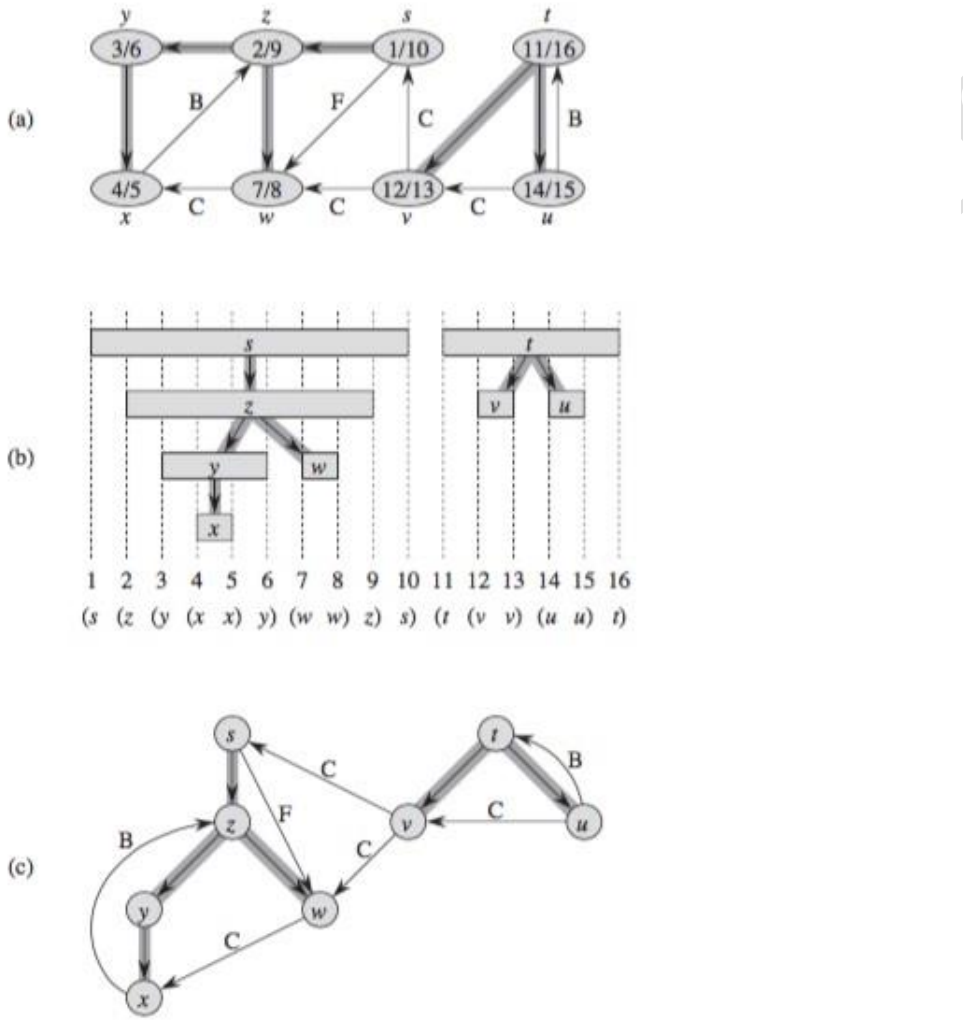


Figure 22.5 Properties of depth-first search. **(a)** The result of a depth-first search of a directed graph. Vertices are timestamped and edge types are indicated as in Figure 22.4. **(b)** Intervals for the discovery time and finishing time of each vertex correspond to the parenthesization shown. Each rectangle spans the interval given by the discovery and finishing times of the corresponding vertex. Only tree edges are shown. If two intervals overlap, then one is nested within the other, and the vertex corresponding to the smaller interval is a descendant of the vertex corresponding to the larger. **(c)** The graph of part (a) redrawn with all tree and forward edges going down within a depth-first tree and all back edges going up from a descendant to an ancestor.

1. Vertex v is a descendant of vertex u in the depth-first forest if and only if v is discovered during the time that u is grey.

2. **(Parenthesis structure)** If v is a descendant of u in the depth-first forest, then $[v.d,v.f]$ is contained entirely within $[u.d,u.f]$. If neither vertex is a descendant of the other, then the intervals are disjoint.
3. **(White path theorem)** In a depth-first forest, v is a descendant of u if and only if at the time $u.d$ that the search discovers u, there is a path from u to v consisting entirely of white vertices.

2.4 Correctness proofs for depth-first-search properties

2.4.1 v is a descendant of u if and only if v is discovered during the time that u is grey

This is because the structure of the depth-first forest precisely mirrors the structure of recursive calls of DFSVisit. Specifically, u is grey only during the time that we are processing the neighbors of u, and the neighbors of u's neighbors, etc.

2.4.2 Parenthesis structure

This mostly follows from the structure of the recursive calls.

Assume without loss of generality that $u.d < v.d$ (i.e. u is discovered before v). We consider two cases:

1. $v.d < u.f$ (i.e. v is discovered before u is finished). In this case v is discovered during the time u is grey, so v is a descendant of u. This also means that DFS-VISIT is called on v while we are still somewhere inside the DFS-VISIT call on u, which means we need to finish up v before finishing up u. Therefore $v.f < u.f$, and $[v.d,v.f]$ is entirely contained within $[u.d,u.f]$.
2. $u.f < v.d$ (i.e. u is finished before v is discovered). Because discovery always happens before finishing, $u.d < u.f < v.d < v.f$, and the intervals are disjoint. Since the intervals are disjoint, neither vertex was discovered while the other was grey, and so neither vertex is a descendant of the other.

2.4.3 White path theorem

If $v = u$, the statement is true, because u is white when we set the value of $u.d$.

($\Rightarrow$) If v is a proper descendant of u, then $u.d < v.d$, so v is white at time $u.d$. This applies to all the vertices on the path from u to v in the depth first forest, because all those vertices are descendants of u. Therefore, the path from u to v consists entirely of white vertices.

($\Leftarrow$) Now suppose there is a path from u to v consisting entirely of white vertices, but v is not a descendant of u in the depth first tree. Consider v^0, which is the first vertex along the path from u to v that is not a descendant of u. Let $\pi(v^0)$ be the predecessor of v^0. $\pi(v^0)$

must be a descendant of u, so by the parenthesis structure, $\pi(v^0).f \leq u.f$. Furthermore, because there is a path of white vertices from u to v^0, we know that v^0 is discovered after u is discovered, but before $\pi(v^0)$ is finished. So $u.d < v^0.d < \pi(v^0).f \leq u.f$. By the parenthesis structure, $[v^0.d, v^0.f]$ must be contained within $[u.d, u.f]$, so v^0 is a descendant of u, producing a contradiction.