CMSC 351: Depth-First Traverse (Stack Version)

Justin Wyss-Gallifent

December 12, 2022

1 Introduction:

Suppose we are given a graph G and a starting node s. Suppose we wish to simply traverse the graph in some way looking for a particular value associated with a node. We're not interested in minimizing distance or cost or any such thing, we're just interested in the traverse process.

2 Intuition

One classic way to go about this is a depth-first traverse. The idea is that starting with a starting node s we follow one brance (typically recursively) as far as possible before backtracking. When we backtrack we only do so as little as possible until we can go deeper again.

3 Algorithm

The algorithm for depth-first traverse starting at a vertex s proceeds as follows:

We first set up:

- A stack with just s on it, so $S = [s]$.
- A boolean list D of length V called the *visited array* which indicates whether a vertex has been visited or not and fill it full of F, or 0.

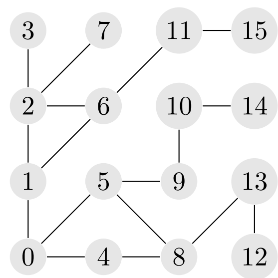
We then repeat the following steps until the stack is empty:

1. $x = S.pop$. If x has been visited, ignore it.
2. $D[x] = T$
3. Find all vertices adjacent to x which have not been visited. For each, push it on the stack.

By convention and consistency when we "find all vertices" we'll do it in increasing numerical order.

4 Working Through an Example

Example 4.1. Consider the following graph



Suppose we wish to traverse the graph starting at the node $s = 0$.

Thus for our above example we start with $S = [0]$ and $D = [F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F]$

Iterate! We pop $x = 0$. It is unvisited. We mark it as visited and note that it is adjacent to unvisited $1, 4, 5$. We push them onto the stack.
$S = [1, 4, 5]$
$D = [T, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F]$

Iterate! We pop $x = 5$. It is unvisited. We mark it as visited and note that it is adjacent to unvisited $8, 9$. We push them onto the stack.
$S = [1, 4, 8, 9]$
$D = [T, F, F, F, F, T, F, F, F, F, F, F, F, F, F, F]$

Iterate! We pop $x = 9$. It is unvisited. We mark it as visited and note that it is adjacent to unvisited 10. We push that onto the stack.
$S = [1, 4, 8, 10]$
$D = [T, F, F, F, F, T, F, F, F, T, F, F, F, F, F, F]$

Iterate! We pop $x = 10$. It is unvisited. We mark it as visited and note that it is adjacent to unvisited 14. We push that onto the stack.
$S = [1, 4, 8, 14]$
$D = [T, F, F, F, F, T, F, F, F, T, T, F, F, F, F, F]$

Iterate! We pop $x = 14$. It is unvisited. We mark it as visited and note that it is not adjacent to any unvisited vertices.
$S = [1, 4, 8]$
$D = [T, F, F, F, F, T, F, F, F, T, T, F, F, F, T, F]$

Iterate! We pop $x = 8$. It is unvisited. We mark it as visited and note that it is adjacent to unvisited $4, 13$. We push them onto the stack.
$S = [1, 4, 4, 13]$
$D = [T, F, F, F, F, T, F, F, T, T, T, F, F, F, T, F]$

Iterate! We pop $x = 13$. It is unvisited. We mark it as visited and note that it is adjacent to unvisited 12. We push that onto the stack.
$S = [1, 4, 4, 12]$
$D = [T, F, F, F, F, T, F, F, T, T, T, F, F, T, T, F]$

Iterate! We pop $x = 12$. It is unvisited. We mark it as visited and note that it is not adjacent to any unvisited vertices.

$S = [1, 4]$
$D = [T, F, F, F, F, T, F, F, T, T, T, F, T, T, T, F]$

Iterate! We pop $x = 4$. It is unvisited. We mark it as visited and note that it is not adjacent to any unvisited vertices.
$S = [1]$
$D = [T, F, F, F, T, T, F, F, T, T, T, F, T, T, T, F]$

Iterate! We pop $x = 4$. It has been visited so we ignore it.
$S = [1]$
$D = [T, F, F, F, T, T, F, F, T, T, T, F, T, T, T, F]$

Iterate! We pop $x = 1$. It is unvisited. We mark it as visited and note that it is adjacent to unvisited $2, 6$. We push them onto the stack.
$S = [2, 6]$
$D = [T, T, F, F, T, T, F, F, T, T, T, F, T, T, T, F]$

Iterate! We pop $x = 6$. It is unvisited. We mark it as visited and note that it is adjacent to unvisited $2, 11$. We push them onto the stack.
$S = [2, 2, 11]$
$D = [T, T, F, F, T, T, T, F, T, T, T, F, T, T, T, F]$

Iterate! We pop $x = 11$. It is unvisited. We mark it as visited and note that it is adjacent to unvisited 15. We push that onto the stack.
$S = [2, 2, 15]$
$D = [T, T, F, F, T, T, T, F, T, T, T, T, T, T, T, F]$

Iterate! We pop $x = 15$. It is unvisited. We mark it as visited and note that it is not adjacent to any unvisited vertices.
$S = [2, 2]$
$D = [T, T, F, F, T, T, T, F, T, T, T, T, T, T, T, T]$

Iterate! We pop $x = 2$. It is unvisited. We mark it as visited and note that it is adjacent to unvisited $3, 7$. We push them onto the stack.
$S = [2, 3, 7]$
$D = [T, T, T, F, T, T, T, F, T, T, T, T, T, T, T, T]$

Iterate! We pop $x = 7$. It is unvisited. We mark it as visited and note that it not adjacent to any unvisited vertices.
$S = [2, 3]$
$D = [T, T, T, F, T, T, T, T, T, T, T, T, T, T, T, T]$

Iterate! We pop $x = 3$. It is unvisited. We mark it as visited and note that it is not adjacent to any unvisited vertices.
$S = [2]$
$D = [T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T]$

Iterate! We pop $x = 2$. It has been visited so we ignore it.
$S = [3, 7]$
$D = [T, T, T, F, T, T, T, F, T, T, T, T, T, T, T, T]$

Now we are done because the stack is empty.

The order in which we traverse is the order in which the nodes were popped and marked as visited:

$$0, 5, 9, 10, 14, 8, 13, 12, 4, 1, 6, 11, 15, 2, 7, 3$$

5 Pseudocode

Here is the pseudocode for the above implementation.

```
function dftstack(G,x):
    D = list of FALSE of length V
    S = [x]
    while S is nonempty
        x = S.pop
        if D[x] == FALSE
            D[x] = TRUE
            for all y adjacent to x
                if D[y] == FALSE:
                    S.push(y)
                end if
            end for
        end if
    end while
end function
```

Note 5.0.1. Depth-first traversing is more useful when we suspect that the target is far from the starting node.

Note 5.0.2. Depth-first traversing is more useful for puzzle-like problems which involve making a decision and carrying it through to completion (this is a recursive process).

6 Pseudocode Time Complexity

Needs finishing, sorry!

7 Modifying to Search

Depth-first traverse can be tweaked if there is a target node in mind. How would you tweak the pseudocode to exit as soon as the target was found and how would that change the time complexity?