CMSC 351: Depth-First Search

Justin Wyss-Gallifent

April 27, 2021

1 Introduction:

Suppose we are given a graph G and a starting vertex s. Suppose we wish to simply search 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 search process.

2 Intuition

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

If this concept is unclear we'll see it unfold with an example after we give the pseudocode.

3 Pseudocode

The following algorithm simply visits all vertices. It doesn't do anything with them except assigns a global list in the order in which they are visited.

Here we present the recursive version. It's fairly straightforward to write a non-recursive version but there are some nuances which need to be dealt with. Here the phrasing of the pseudocode assumes that `G` is stored as an adjacency list.

```
// These are global.
VISITORDER = []
VISITED = list of FALSE of length V
function depthfirstsearch(G,x):
    VISITORDER.append(x)
    VISITED[x] = TRUE
    for all edges from x
        y = the vertex at the other end
        if VISITED[Y] == FALSE
            depthfirstsearch(y)
        end
    end
end
depthfirstsearch(G,s)
```

Consider that each recursive call is on an undiscovered vertex. It follows that the maximum depth of the recursion will be equal to the longest path extending from the starting vertex. Along with the fact that the recursion is only called on a vertex which is undiscovered gives insight into the operation of the algorithm and why the recursive process actually terminates.

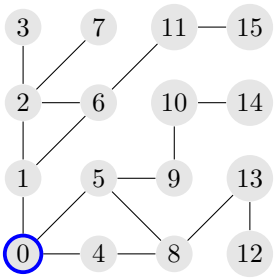
Note 3.0.1. Note that `for` loop will always encounter at least one previously discovered vertex, the algorithm-parent of `x`, except when `x` is the root. If at any point the algorithm enounters a previously discovered vertex which is not the algorithm-parent of `x` then the algorithm has discovered a cycle.

Note 3.0.2. If we wish to exit when we find a node then we'll need to manage the recursion properly so that when the node is found we progressively drop out of all the recursive layers.

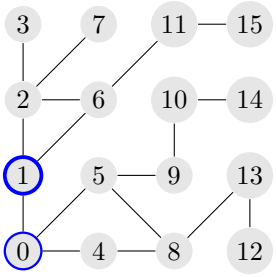
Note 3.0.3. Depth-first searching is more useful when we suspect that the target is far from the starting vertex.

Note 3.0.4. Depth-first searching is more useful for puzzle-like problems which involve making a decision and carrying it through to completion (this is a recursive process).

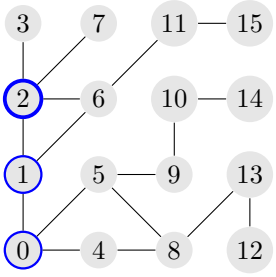
Example 3.1. Consider the following graph. This is almost identical to the graph in the breadth-first search except one edge has been removed because it makes the progress more clear.
Here $x = 0$ and $D = [T, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F]$.



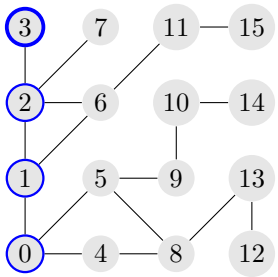
There are undiscovered vertices $\{1, 4, 5\}$.
Here $x = 1$ and $D = [T, T, F, F, F, F, F, F, F, F, F, F, F, F, F, F]$.



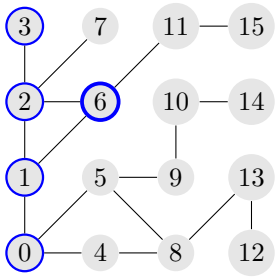
There are undiscovered vertices $\{2, 6\}$.
Here $x = 2$ and $D = [T, T, T, F, F, F, F, F, F, F, F, F, F, F, F, F]$:



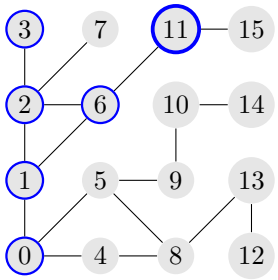
There are undiscovered vertices $\{3, 6, 7\}$.
Here $x = 3$ and $D = [T, T, T, T, F, F, F, F, F, F, F, F, F, F, F, F]$:



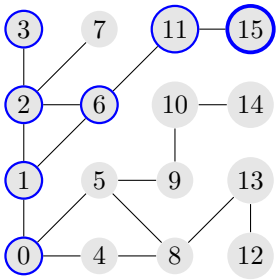
There are undiscovered vertices {}.
Here $x = 6$ and $D = [T, T, T, T, F, F, T, F, F, F, F, F, F, F, F, F]$:



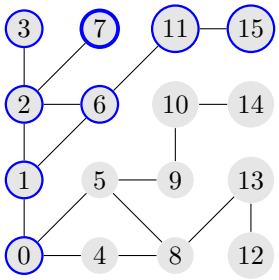
There are undiscovered vertices {11}.
Here $x = 11$ and $D = [T, T, T, T, F, F, T, F, F, F, F, T, F, F, F, F]$.



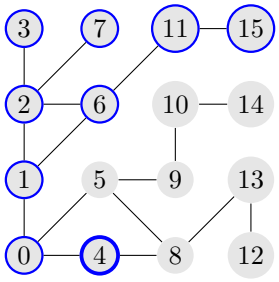
There are undiscovered vertices {15}.
Here $x = 15$ and $D = [T, T, T, T, F, F, T, F, F, F, F, T, F, F, F, T]$:



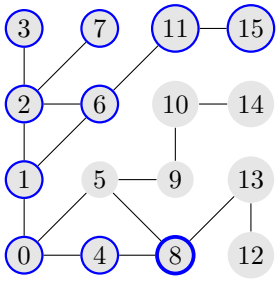
There are undiscovered vertices {}.
Here $x = 7$ and $D = [T, T, T, T, F, F, T, T, F, F, F, T, F, F, F, T]$:



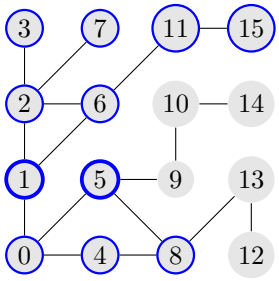
There are undiscovered vertices {}.
Here $x = 4$ and $D = [T, T, T, T, T, F, T, T, F, F, F, T, F, F, F, T]$:



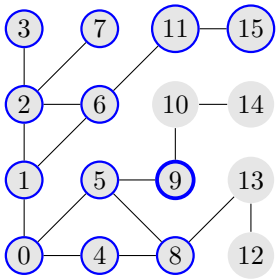
There are undiscovered vertices {8}.
Here $x = 8$ and $D = [T, T, T, T, T, F, T, T, T, F, F, T, F, F, F, T]$:



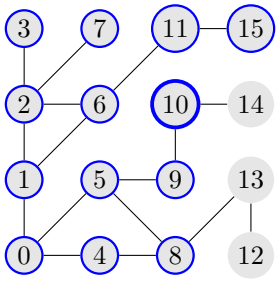
There are undiscovered vertices {5, 13}.
Here $x = 5$ and $D = [T, T, T, T, T, T, T, T, T, F, F, T, F, F, F, T]$:



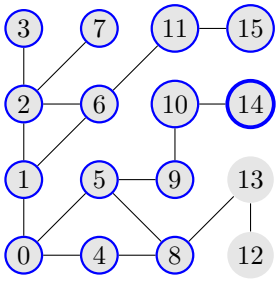
There are undiscovered vertices $\{9\}$.
Here $x = 9$ and $D = [T, T, T, T, T, T, T, T, T, T, F, T, F, F, F, T]$:



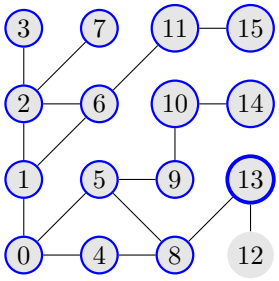
There are undiscovered vertices $\{10\}$.
Here $x = 10$ and $D = [T, T, T, T, T, T, T, T, T, T, T, T, F, F, F, T]$:



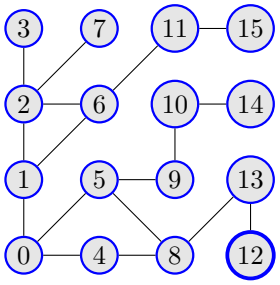
There are undiscovered vertices $\{14\}$.
Here $x = 14$ and $D = [T, T, T, T, T, T, T, T, T, T, T, T, F, F, T, T]$:



There are undiscovered vertices $\{\}$.
Here $x = 13$ and $D = [T, T, T, T, T, T, T, T, T, T, T, T, F, T, T, T]$:



There are undiscovered vertices $\{12\}$.
Here $x = 12$ and $D = [T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T]$:



4 Pseudocode Time Complexity

Suppose V is the number of vertices and E is the number of edges. What follows is exactly the same as breadth-first search.

- The initialization takes $\mathcal{O}(V)$. This could in fact take $\Theta(1)$ depending on the architecture but the choice has no effect on the result.
- Each vertex gets pushed onto and popped off the queue exactly once so this is $2\Theta(1)$ each for a total of $\Theta(V)$.
- Since each edge is attached to two vertices the `for` loop will iterate a total of $2E$ times over the course of the entire algorithm. This gives a total of $\Theta(2E) = \Theta(E)$.

The time complexity is therefore $\mathcal{O}(V) + \Theta(V) + \Theta(E) = \mathcal{O}(V + E)$. If initialization is actually $\Theta(1)$ then this becomes $\Theta(V + E)$.

Note 4.0.1. Note that our pseudocode and analysis assumes we have direct access to a vertex's edges using something like an adjacency list. If we use an adjacency matrix then the inner loop becomes $\Theta(V)$ and the entire pseucode becomes $\Theta(v^2)$.

5 Thoughts, Problems, Ideas

1. Suppose G is stored by its adjacency matrix AM. Adjust the depth-first pseudocode to make this clear and calculate the resulting Θ time complexity. Call this new function `depthfirstsearch(AM,x)`.

2. Modify the depth-first pseudocode to add a second list `PARENT` which keeps track of the parent of each node visited. That is, `PARENT(z)` should contain the parent of vertex `z`. The root node should have `NULL` assigned. How does this affect the time complexity?

3. Modify the depth-first search pseudocode to detect and return `TRUE` if there is a cycle in the graph and `FALSE` if not. How does this affect the time complexity?

6 Python Test and Output

The following code is applied to the graph above. This follows the model of the pseudocode and in addition creates and returns a list of the vertices in the order in which they were visited.
Code:

```
def dfs(EL,n,x,depth):
    print('_'*depth + 'Recursive depth = ' + str(depth))
    D[x] = True
    V.append(x)
    print('_'*depth + 'V = ' + str(V))
    print('_'*depth + 'D = ' + str(D).replace('True','T').
        replace('False','F'))
    for y in EL[x]:
        if not D[y]:
            dfs(EL,n,y,depth+1)
EL = [
    [1, 4, 5],
    [0, 2, 6],
    [1, 3, 6, 7],
    [2],
    [0, 8],
    [0, 8, 9],
    [1, 2, 11],
    [2],
    [4, 5, 13],
    [5, 10],
    [9, 14],
    [6, 15],
    [13],
    [8, 12],
    [10],
    [11]
]
n = 16
s = 0
D = [False] * n
V = []
dfs(EL,n,s,0)
```

Output:

```
V = [0]
D = [T, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F]
_Recursive depth = 1
_V = [0, 1]
_D = [T, T, F, F, F, F, F, F, F, F, F, F, F, F, F, F]
__Recursive depth = 2
__V = [0, 1, 2]
__D = [T, T, T, F, F, F, F, F, F, F, F, F, F, F, F, F]
___Recursive depth = 3
___V = [0, 1, 2, 3]
___D = [T, T, T, T, F, F, F, F, F, F, F, F, F, F, F, F]
___Recursive depth = 3
___V = [0, 1, 2, 3, 6]
___D = [T, T, T, T, F, F, T, F, F, F, F, F, F, F, F, F]
____Recursive depth = 4
____V = [0, 1, 2, 3, 6, 11]
____D = [T, T, T, T, F, F, T, F, F, F, F, T, F, F, F, F]
_____Recursive depth = 5
_____V = [0, 1, 2, 3, 6, 11, 15]
_____D = [T, T, T, T, F, F, T, F, F, F, F, T, F, F, F, T]
___Recursive depth = 3
___V = [0, 1, 2, 3, 6, 11, 15, 7]
___D = [T, T, T, T, F, F, T, T, F, F, F, T, F, F, F, T]
_Recursive depth = 1
_V = [0, 1, 2, 3, 6, 11, 15, 7, 4]
_D = [T, T, T, T, T, F, T, T, F, F, F, T, F, F, F, T]
__Recursive depth = 2
__V = [0, 1, 2, 3, 6, 11, 15, 7, 4, 8]
__D = [T, T, T, T, T, F, T, T, T, F, F, T, F, F, F, T]
___Recursive depth = 3
___V = [0, 1, 2, 3, 6, 11, 15, 7, 4, 8, 5]
___D = [T, T, T, T, T, T, T, T, T, F, F, T, F, F, F, T]
____Recursive depth = 4
____V = [0, 1, 2, 3, 6, 11, 15, 7, 4, 8, 5, 9]
____D = [T, T, T, T, T, T, T, T, T, T, F, T, F, F, F, T]
_____Recursive depth = 5
_____V = [0, 1, 2, 3, 6, 11, 15, 7, 4, 8, 5, 9, 10]
_____D = [T, T, T, T, T, T, T, T, T, T, T, T, F, F, F, T]
______Recursive depth = 6
______V = [0, 1, 2, 3, 6, 11, 15, 7, 4, 8, 5, 9, 10, 14]
______D = [T, T, T, T, T, T, T, T, T, T, T, T, F, F, T, T]
___Recursive depth = 3
___V = [0, 1, 2, 3, 6, 11, 15, 7, 4, 8, 5, 9, 10, 14, 13]
___D = [T, T, T, T, T, T, T, T, T, T, T, T, F, T, T, T]
____Recursive depth = 4
____V = [0, 1, 2, 3, 6, 11, 15, 7, 4, 8, 5, 9, 10, 14, 13,
    12]
____D = [T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T]
```