# CMSC 351: Breadth-First Traverse

Justin Wyss-Gallifent

November 6, 2023

# 1 Introduction

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

# 2 Intuition

One classic way to go about this is a breadth-first traverse. The idea is that starting with $s$ we check all vertices connected to $s$ first, and then all vertices connected to those, and so on. In this sense we're covering the graph in "layers of increasing distance from $s$". This is the idea of a breadth-first traverse.

**Note 2.0.1.** The shortest path algorithm basically basically follows a breadth-first approach.

**Note 2.0.2.** Breadth-first traversing is more useful if there's a target and we suspect that the target is close to the starting vertex.

**Note 2.0.3.** Breadth-first traversing is more useful for things like web-crawling when we might want to find all of the closer vertices first and the algorithm may truncate early.

**Note 2.0.4.** Breadth-first traversing is more useful when we are trying to explore a strongly connected part of a graph, the idea being that we want to explore close to home before venturing too far away.

# 3 Algorithm

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

We first set up:

- A queue $Q = [s]$.
- A boolean list $VISITED$ of length $V$ which indicates whether a vertex has been visited or not and fill it full of $F$, or 0, except $VISITED[s] = T$.
- A list $VORDER = [s]$ which will contain the vertices in the order we visit them.

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

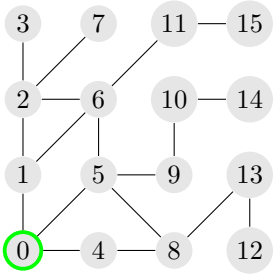
1. $x = Q.dequeue$
2. Find all vertices adjacent to $x$ which have not been visited. For each, put them on $Q$ and update $VISITED$ and $VORDER$.

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

# 4 Working Through an Example

In this example we won't show every single iteration. Instead we'll only show those as noted.

**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:
$Q = [0]$
$VISITED = [T, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F]$
$VORDER = [0]$

Iterate! We dequeue $x = 0$. We find all connected and undiscovered vertices

$\{1, 4, 5\}$ those get put onto $Q$ and we update $VISITED$ and $VORDER$:
$Q = [1, 4, 5]$
$VISITED = [T, T, F, F, T, T, F, F, F, F, F, F, F, F, F]$:
$VORDER = [0, 1, 4, 5]$

Iterate! We dequeue $x = 1$, we find all connected and undiscovered vertices $\{2, 6\}$ so those get put onto $Q$ and we update $VISITED$ and $VORDER$:
$Q = [4, 5, 2, 6]$
$VISITED = [T, T, T, F, T, T, T, F, F, F, F, F, F, F, F]$:
$VORDER = [0, 1, 4, 5, 2, 6]$

Iterate! We dequeue $x = 4$, we find all connected and undiscovered vertices $\{8\}$ so those get put onto $Q$ and we update $VISITED$ and $VORDER$:
$Q = [5, 2, 6, 8]$
$VISITED = [T, T, T, F, T, T, T, F, T, F, F, F, F, F, F]$:
$VORDER = [0, 1, 4, 5, 2, 6, 8]$

Iterate! We dequeue $x = 5$, we find all connected and undiscovered vertices $\{9\}$ so those get put onto $Q$ and we update $VISITED$ and $VORDER$:
$Q = [2, 6, 8, 9]$
$VISITED = [T, T, T, F, T, T, T, F, T, T, F, F, F, F, F]$:
$VORDER = [0, 1, 4, 5, 2, 6, 8, 9]$

Iterate! We dequeue $u = 2$, it gives us undiscovered vertices $\{3, 7\}$ so those get put onto $Q$ and we update $VISITED$ and $VORDER$:
$Q = [6, 8, 9, 3, 7]$
$VISITED = [T, T, T, T, T, T, T, T, T, T, F, F, F, F, F]$:
$VORDER = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7]$

Iterate! We dequeue $u = 6$, we find all connected and undiscovered vertices $\{11\}$ so those get put onto $Q$ and we update $VISITED$ and $VORDER$:
$Q = [8, 9, 3, 7, 11]$
$VISITED = [T, T, T, T, T, T, T, T, T, T, F, T, F, F, F]$:
$VORDER = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11]$

Iterate! We dequeue $u = 8$, we find all connected and undiscovered vertices $\{13\}$ so those get put onto $Q$ and we update $VISITED$ and $VORDER$:
$Q = [9, 3, 7, 11, 13]$
$VISITED = [T, T, T, T, T, T, T, T, T, T, F, T, F, T, F]$:
$VORDER = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11]$

Iterate! We dequeue $u = 9$, we find all connected and undiscovered vertices $\{10\}$ so those get put onto $Q$ and we update $VISITED$ and $VORDER$:
$Q = [3, 7, 11, 13, 10]$
$VISITED = [T, T, T, T, T, T, T, T, T, T, T, T, F, T, F]$:
$VORDER = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 9]$

Iterate! We dequeue $u = 3$, we find all connected and undiscovered vertices $\{\}$.
$Q = [7, 11, 13, 10]$
$VISITED = [T, T, T, T, T, T, T, T, T, T, T, T, F, T, F, F]$:
$VORDER = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 9]$

Iterate! We dequeue $u = 7$, we find all connected and undiscovered vertices $\{\}$.
$Q = [11, 13, 10]$
$VISITED = [T, T, T, T, T, T, T, T, T, T, T, T, F, T, F, F]$:
$VORDER = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 9]$

Iterate! We dequeue $u = 11$, we find all connected and undiscovered vertices $\{15\}$ so those get put onto $Q$ and we update $VISITED$ and $VORDER$:
$Q = [13, 10, 15]$
$VISITED = [T, T, T, T, T, T, T, T, T, T, T, T, F, T, F, T]$:
$VORDER = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 9, 15]$

Iterate! We dequeue $u = 13$, we find all connected and undiscovered vertices $\{12\}$ so those get put onto $Q$ and we update $VISITED$ and $VORDER$:
$Q = [10, 15, 12]$
$VISITED = [T, T, T, T, T, T, T, T, T, T, T, T, T, F, T]$:
$VORDER = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 9, 15, 12]$

Iterate! We dequeue $u = 10$, we find all connected and undiscovered vertices $\{14\}$ so those get put onto $Q$ and we update $VISITED$ and $VORDER$:
$Q = [15, 12, 14]$
$VISITED = [T, T, T, T, T, T, T, T, T, T, T, T, T, T, T]$:
$VORDER = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 9, 15, 12, 14]$

Iterate! We dequeue $u = 15$, we find all connected and undiscovered vertices $\{\}$.
$Q = [12, 14]$
$VISITED = [T, T, T, T, T, T, T, T, T, T, T, T, T, T, T]$:
$VORDER = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 9, 15, 12, 14]$

Iterate! We dequeue $u = 12$, we find all connected and undiscovered vertices $\{\}$.
$Q = [14]$
$VISITED = [T, T, T, T, T, T, T, T, T, T, T, T, T, T, T]$:
$VORDER = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 9, 15, 12, 14]$

Iterate! We dequeue $u = 14$, we find all connected and undiscovered vertices $\{\}$.
$Q = []$
$VISITED = [T, T, T, T, T, T, T, T, T, T, T, T, T, T, T]$:

$VORDER = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 9, 15, 12, 14]$

Since $Q$ is empty we're done.

We return $VORDER = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 13, 10, 15, 12, 14]$.

# 5 Pseudocode

Here is the pseudocode:

```
function bft(G,s)
    QUEUE = [s]
    VISITED = list of FALSE of length V
    VISITED[s] = TRUE
    VORDER = [s]
    while QUEUE is not empty
        x = QUEUE.dequeue
        for all y adjacent to x
            if VISITED[y] == FALSE
                QUEUE.enqueue(y)
                VISITED[y] = TRUE
                VORDER.append(y)
            end
        end
    end
    return(VORDER)
end
```

# 6 Pseudocode Time Complexity

Suppose $V$ is the number of vertices and $E$ is the number of edges.

- 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 enqueued and dequeued exactly once so this is $V\,\Theta(1)$ each for a total of $\Theta(V)$ for just the `dequeue`, not the `for` loop.
- The body of the `for` loop iterates $2E$ times over the course of the entire algorithm, once for each vertex at each end of the edge. The body takes constant time $\Theta(1)$. so overall this is $\Theta(2E) = \Theta(E)$.

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 6.0.1.** 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 $\Theta(V^2)$.

**Note 6.0.2.** There are breadth-first traverses which run in $\mathcal{O}(E \lg V)$ but they requires a radically different pseudocode based upon a heap structure instead of a simple list $S$. This heap structure is what leads to the $\lg V$ factor.

# 7 Modifying to Search

Breadth-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?

# 8 Thoughts, Problems, Ideas

1. Suppose node $i$ is a structure with properties $i.height$, $i.weight$ and $i.volume$. Modify the pseudocode to return the weight of the first node encountered whose weight is more than 100. You may assume such a node exists.
2. Same as above but no such assumption. Return $NULL$ if no such node is found.
3. When $s$ is dequeued all of the vertices connected to $s$ will be newly discovered. This is not true for any other vertex $x$ because the algorithm-parent of $x$ will already be discovered. Under what circumstances, for every other vertex $x$, would the algorithm-parent be the only previously discovered vertex?
4. Let $q_i$ be the length of $Q$ after the $i$th iteration of the `while` loop. We'll say $q_0 = 1$ to be comprehensive since $Q = [s]$ when no iterations have completed. So in the example in the notes $q_1 = 3$, $q_3 = 4$, and so on. Of course there is some $k$ such that $q_k = 0$ as this is when the algorithm ends. Moreover in the example $q_i$ initially (nonstrictly) increases and then (nonstrictly) decreases. Must the $q_i$ always follow this pattern? Explain.
5. Building off the previous problem what is the maximum that $k$ might be? How about the minimum? What would a graph look like (qualitatively) if its $k$-value were somewhere in the middle? Explain using specific examples of graphs.
6. Describe the impact if the graph were given with the adjacency matrix rather than the adjacency list. How would that impact the pseudocode and the time complexity?
7. Modify the pseudocode so that we may pass a `maxdepth` and so that the algorithm will go no further than that depth from the starting vertex.

# 9 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 discovered.

Code:

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

```
s = 0
visited = bfs(EL,n,s)
print('Discovered = ' + str(visited))
```

Output:

```
Q = [0]
V = [0]
Q = [1, 4, 5]
V = [0, 1, 4, 5]
Q = [4, 5, 2, 6]
V = [0, 1, 4, 5, 2, 6]
Q = [5, 2, 6, 8]
V = [0, 1, 4, 5, 2, 6, 8]
Q = [2, 6, 8, 9]
V = [0, 1, 4, 5, 2, 6, 8, 9]
Q = [6, 8, 9, 3, 7]
V = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7]
Q = [8, 9, 3, 7, 11]
V = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11]
Q = [9, 3, 7, 11, 13]
V = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 13]
Q = [3, 7, 11, 13, 10]
V = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 13, 10]
Q = [7, 11, 13, 10]
V = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 13, 10]
Q = [11, 13, 10]
V = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 13, 10]
Q = [13, 10, 15]
V = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 13, 10, 15]
Q = [10, 15, 12]
V = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 13, 10, 15, 12]
Q = [15, 12, 14]
V = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 13, 10, 15, 12, 14]
Q = [12, 14]
V = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 13, 10, 15, 12, 14]
Q = [14]
V = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 13, 10, 15, 12, 14]
Q = []
V = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 13, 10, 15, 12, 14]
Discovered = [0, 1, 4, 5, 2, 6, 8, 9, 3, 7, 11, 13, 10, 15, 12, 14]
```