Breadth-First Search and Shortest Paths

Suhas Arehalli

COMP221 - Spring 2024

1 Introductions

- When you finally get free of sorting, the next most common Algorithms topic is Graphs!
- Why? Because, as you saw in COMP128, graphs are really good at representing a variety of problems, and the solutions for those problems can often be found by analyzing the corresponding graph using a standard graph algorithm.
- **Note:** Skiena does a very in-depth job introducing you to the various ways that graph traversals (today's topic) can be useful for a variety of problems. But that's just one pillar of the course! I'll spend today's notes preparing you for one other pillar: Analyzing graph algorithms for correctness!
- Since BFS/DFS are algorithms you've seen and talked about before, I'll be light on the exposition and dive into the trickiness of the working with graphs formally.

2 Shortest Paths in an Unweighted Graph

- Our goal for today is to prove formally that BFS finds us the shortest path between the start node and a target node. Pay attention to how the proof is structured, and where the intutions come from, because working with graphs is a bit tricker than arrays!
- A Graph G is a pair (V, E), where V is a set consisting of vertices and E is a set of edges, where each edge is a pair $(v_1, v_2) \in E$, where $v_1, v_2 \in V$.
  - Today we'll consider *undirected* graphs, where $(v_1, v_2) \in E$ implies $(v_2, v_1) \in E$. That is, the order of vertices in an edges doesn't matter!
  - We often visually represent them as circles connected by lines, but to analyze algorithms over graphs, it's helpful to build an intuition for the correspondence between our visual representations of data structures, their implementation (in adjacency lists or adjacency matrices!), and their mathematical structure (A pair (V, E)).
- A path from $v \in V$ to $v' \in V$ is a sequence of edges

$$(v, v_1), (v_1, v_2), \ldots, (v_{k-2}, v_{k-1}), (v_{k-1}, v') \in E$$

  We call the number of edges in the path (here k) it's length. Denote this by $d(P) = k$.

- Equivalently, we can represent a path from v to v' as a sequences of vertices

$$v = v_0, v_1, v_2, \ldots, v_{k-1}, v_k = v'$$

  where every adjacent pair of vertices $(v_i, v_{i+1}) \in E$. These are equivalent!

- With that terminology out of the way, we want to prove that BFS will find the *shortest path* from a start vertex $s \in V$ to target vertex $t \in V$.

- In practice, we will find that looking for the distance from s to some t is asymptotically the same as finding the distance from s to *every* $v \in V$. We call this the **Single Source Shortest Path** Problem!

- For simplicity, the pseudocode for BFS in Alg. **??** does not explicitly construct the path from s to t, but instead returns the *length* of the shortest path. Maintaining the path is not hard (just maintain a representation of the path, either in the graph's data structure or in the tuple in the queue!), but will clutter our pseudocode — consider it an exercise for the reader!

Algorithm 1 A Breadth First Search that returns the length of the shortest path from s to v for all $v \in V$.

```
function BFS(G = (V, E), s)
    Let Q be a Queue.
    Let d be an array s.t. d[v] ← ∞ for v ∈ V.
    DISCOVER(s)
    d[s] ← 0
    Q.enqueue(s)
    while Q is nonempty do
        v ← Q.dequeue()
        for (v, v') ∈ E do
            if v' is undiscovered then
                DISCOVER(v')
                d[v'] ← d[v] + 1
                Q.enqueue(v')
            end if
        end for
    end while
    return d
end function
```

2.1 Proof of Correctness

We'll prove the correctness of this algorithm in a sequence of smaller statements, called lemmas. These will prove useful facts about BFS that will eventually lead to an argument for the fact that BFS finds the shortest paths!

Note that I've taken this presentation of correctness from CLRS, the alternate, optional textbook linked in the Syllabus, and adapted it for our purposes. Feel free to take a look if you want the raw presentation!

First, some notation and a useful fact about shortest path lengths:

Def.. For a graph $G = (V, E)$, we define $\delta(s, v)$ as the length of the shortest path from s to v.

Lemma 1. *Let $G = (V, E)$ and $s \in V$. For all $(v, v') \in V$, we have that*

$$\delta(s, v') \leq \delta(s, v) + 1$$

Proof. Proceed by cases:

Case 1: If s and v' are disconnected, then s and v must be disconnected, since we know the edge $(v, v') \in E$. Then both $\delta(s, v) = \infty$ and $\delta(s, v') = \infty$ and the inequality is satisfied.

Case 2: The shortest path from s to v' consists of the shortest path from s to v followed by the edge (v, v'). Then $\delta(s, v') = \delta(s, v) + 1$.

Case 3: Otherwise, by definition, the shortest path must be shorter than the path constructed by appending the edge from (v, v') to the shortest path from s to v, which has length $\delta(s, v) + 1$. Thus, $\delta(s, v') < \delta(s, v) + 1$. □

Now let's prove a bonus property that we didn't see in class that secretly works it's way into the proof structure:

Lemma 2. *Let $G = (V, E)$ and $s \in V$. Let $P = s, v_1, \ldots, v_k, v$ be the shortest path from s to v. For any $0 \leq i \leq k$, $P_i = s, v_1, \ldots, v_i$ must be the shortest path from s to v_i.*

Proof. Assume for contradiction that there is a shorter path $P_i' = s, v_1', \ldots v_m', v_i$ from s to v_i, where $m < i - 1$. Now construct the new path from s to v, $P' = s, v_1', \ldots, v_m', v_i, v_{i+1}, \ldots v_k, v$. Since $d(P_i') < d(P_i)$, it follows that $d(P') < d(P)$ (P and P' share the other edges!), which contradicts that P is the shortest path from s to v. □

Now we get to a useful proof strategy for this kind of algorithm — the idea that BFS will *relax* a constraint over time to arrive at more precise estimates of distances in d. Thus, we'll begin by showing that $d[v]$ is always an *upper bound* on the true distance $\delta(s, v)$.

Lemma 3. *For BFS called on a graph $G = (V, E)$ and $s \in V$, at all times,*

$$d[v] \geq \delta(s, v)$$

Here, our proof technique will be a little more freeform than we've seen before: induction over operations that change the state of d!

Proof. Proceed by induction over assignments to d

Base Cases: At initialization, trivially true since all values are infinite.

After assignment to $d[s]$, we have that $d[s] \leftarrow 0$. We also know that $\delta(s, s) = 0$, so the inequality remains true.

Inductive Step: Assume by our IH that before this assignment, $d[v] \geq \delta(s, v)$. We will show after the assignment $d[v'] \leftarrow d[v] + 1$, this remains true.

Observe that by our IH, we know $d[v] + 1 \geq \delta(s, v) + 1$. By the previous lemma, we know that $\delta(s, v) + 1 \geq \delta(s, v')$, since our for loop finds the edge $(v, v') \in E$. Chain these two inequalities to arrive at the fact that $d[v] + 1 \geq \delta(s, v')$, and thus after assignment, $d[v'] \geq \delta(s, v')$ as desired. □

Now, a statement from your reading assignment, the key idea for BFS - our Queue is sorted by distance and the distance differs by at most 1!

Lemma 4. *Consider the queue Q in BFS, and let it's length be N. After every iteration of the while loop, we have that*

$$d[Q[1]] \leq \dots d[Q[N]] \leq d[Q[1]] + 1$$

Proof. By induction over queue operations!

Base Case(s) Two special cases: Upon initialization, the queue is empty, so this statement is satisfied trivially.

After s is added, the Queue has 1 element, and it's still true trivially!

Inductive Step: We assume as an IH that we begin with a Queue that satisfies the inequality, and then we will show that our Queue operations will preserve the inequality. Since we have two queue operations that are tied together within our while-loop, we'll handle them in sequence:

Dequeue: Before our dequeue, we will have, for a Queue Q of size N,

$$d[Q[1]] \leq d[Q[2]] \leq \cdots \leq d[Q[N]] \leq d[Q[1]] + 1$$

After our dequeue, keeping the same indexing despite the dequeue for clarity, we need to show

$$d[Q[2]] \leq \cdots \leq d[Q[N]] \leq d[Q[2]] + 1$$

Well, the only inequality that's not a part of our IH is that $d[Q[N]] \leq d[Q[2]] + 1$, but this follows from the IH as well:

$$\begin{aligned} d[Q[1]] &\leq d[Q[2]] \\ d[Q[1]] + 1 &\leq d[Q[2]] + 1 \end{aligned}$$

which we can combine with the last inequality of our IH to find

$$d[Q[N]] \leq d[Q[1]] + 1 \leq d[Q[2]] + 1$$

as desired.

Enqueue: Keeping our indexing from above, we observe that in our code, $v = Q[1]$, and via our IH, we have that

$$d[Q[2]] \leq \cdots \leq d[Q[N]] \leq d[Q[2]] + 1$$

And since we have the same $v = Q[1]$ as in the dequeue section, we know $d[Q[N]] \leq d[Q[1]] + 1$. When we enqueue v', we assign $d[v'] \leftarrow d[v] + 1$, which means

$$\begin{aligned} d[v'] &= d[v] + 1 \\ &= d[Q[1]] + 1. \end{aligned}$$

Since we append a number of v's to our queue with exactly this distance, to preserve the desired property, we must show that

$$\begin{aligned} d[Q[2]] \leq \cdots \leq d[Q[N]] \leq d[v] \leq d[Q[2]] + 1 \\ d[Q[2]] \leq \cdots \leq d[Q[N]] \leq d[Q[1]] + 1 \leq d[Q[2]] + 1 \end{aligned}$$

but as we showed in the enqueue portion, we know that $d[Q[1]] \leq d[Q[2]]$, satisfying the rightmost inequality, and the remaining inequality follows from the identity of v being the element dequeued, as stated above. □

The trick to the above proof is wading through notation: The idea is simple, if we remove the first element, and only add elements of length exactly one longer, than this property must hold, but things get bogged down in notation, which makes things precises, but a little unwieldy.

Now let's put these together to prove that BFS will always find $d[v] = \delta(s, v)$

Statement (BFS is Correct)**.** *Let d be the array returned by BFS run on a graph $G = (V, E)$ with source vertex $s \in V$.*

$$d[v] = \delta(s, v)$$

for all $v \in V$

Before we get into the proof, let me outline our strategy: We will proceed by *contradiction*, and assume that we got $d[v]$ wrong for some $v \in V$. Our goal will be to isolate the first step we went wrong, and since we base our assignments to $d[v']$ based on other $d[v]$ for v closer to s, we will choose the $d[v']$ we got wrong where v' is closest to s! Then, we'll inspect where we went wrong: Along the way to v' via the shortest path, we will be at some other vertex immediately before — call it v — and then we follow the edge (v, v') to v'. Since v' is the vertex closest to s that has a wrong $d[v]$, and v is closer to s than v', we know that $d[v] = \delta(s, v)$, and we can inspect why our algorithm got $d[v]$ right and $d[v']$ wrong.

I recommend drawing out a diagram to accompany this argument!

Now let's get to the formal stuff

Proof. Suppose for contradiction that there exist some $v \in V$ such that $d[v] \neq \delta(s, v)$. Let $v' \in V$ be the vertex with the smallest $\delta(s, v')$ such that $d[v'] \neq \delta(s, v')$.

Let's make 3 observations: first that by Lemma **??**, we know that if $d[v'] \neq \delta(s, v')$, then $d[v'] > \delta(s, v')$ — BFS can only overestimate the distance!

Also observe that if we overestimate, we know $\delta(s, v) \neq \infty$, since we cannot overestimate infinity. This means that s, v are connected, which means we can inspect the true shortest path.

Finally, since we know we assign $d[s] \leftarrow 0$ and $\delta(s, s) = 0$, $\delta(s, v') \geq 1$, so we can talk about the vertex in the path prior to v'

So now let's look that the true shortest path P, labeling the intermediate vertices $v_1, \ldots v_k, v$. [1]

$$P = s, v_1, \ldots, v_k, v, v'$$

Since this is the shortest path from s to v', and the path $P' = s, v_1, \ldots, v_k, v$ is shorter. In fact, by lemma **??**, P' is the shortest path from s to v and thus $\delta(s, v') = \delta(s, v) + 1$. Since we picked v' to be the vertex with the shortest shortest path that was still labeled incorrectly, we know that $d[v] = \delta(s, v)$!

Let's look at when that assignment happened in our pseudocode: Whenever we assign a value to $d[v]$, v must be enqueued, which means it must then be dequeued at some point before termination, as that is required by our while loop. At that point, the for loop must consider the edge $(v, v') \in E$. On that iteration, consider two cases, based on the if-statement:

Case 1: We do not enter the if-statement because v' has been marked discovered. Note that we only mark things discovered if we have added them to the queue prior and assigned a distance to them. Since we just dequeued v, by applying Lemma **??** before the dequeue, we know that

[1] If P is of length one, let $v = s$ and see that $P = s, v'$

$d[v'] \leq d[v] + 1 = \delta(s, v) + 1$. Since we saw earlier that $\delta(s, v') = \delta(s, v) + 1$, this would get us to claim that $d[v'] \leq \delta(s, v')$, which contradicts our observation that we must have overestimated $d[v']$!

Case 2: We enter the if-statement. This means that we eventually assign $d[v'] \leftarrow d[v] + 1$, and since we know that $d[v] = \delta(s, v)$ and that $\delta(s, v') = \delta(s, v) + 1$, we know $d[v'] = \delta(s, v')$. Since we then mark v' as discovered, we never modify $d[v']$ again, so we terminate with $d[v'] = \delta(s, v')$. However, we assumed for contradiction that we didn't!

Thus, we arrive at a contradiction in both cases, and we know that BFS ends with $d[v] = \delta(s, v)$ for all $v \in V$.

□