Writing Algorithm Descriptions and Proofs

Introduction

In the upcoming homework assignment (and the midterm and the final and other courses) you will be asked to give an algorithm that does such and such at $O(\text{something})$. You will also have to prove that something is true (or false). Here is what I expect and some guidelines, tips and tricks.

Describing an Algorithm

An algorithm description can be verbal (preferably as an itemized list) or in the form of a pseudocode (pronounced sue-dough-code), which is a code-like description. Here is an example of the pseudocode for the Breadth-first search (BFS) algorithm described in class:

Algorithm 1 BFS($G, s$)

1: for $u \in V[G] \setminus s$ do
2: \hspace{1em} Color[$u$] $\leftarrow$ White
3: \hspace{1em} dist[$u$] $\leftarrow$ $\infty$
4: \hspace{1em} edge-to[$u$] $\leftarrow$ null
5: end for
6: Color[$s$] $\leftarrow$ Gray
7: dist[$s$] $\leftarrow$ 0
8: edge-to[$s$] $\leftarrow$ null
9: $Q$ $\leftarrow$ $\emptyset$
10: Enqueue($Q, s$)
11: while $Q \neq \emptyset$ do
12: \hspace{1em} $u$ $\leftarrow$ Dequeue($Q$)
13: \hspace{1em} for each $v \in Adj[u]$ do
14: \hspace{2em} if Color[$v$] $==$ White then
15: \hspace{2em} \hspace{1em} Color[$v$] $\leftarrow$ Gray
16: \hspace{2em} \hspace{1em} dist[$v$] $\leftarrow$ dist[$u$] $+ 1$
17: \hspace{2em} \hspace{1em} edge-to[$v$] $\leftarrow$ $u$
18: \hspace{2em} \hspace{1em} Mark the edge from edge-to[$v$] to $u$ as a “tree edge”.
19: \hspace{2em} \hspace{1em} Enqueue($Q, v$)
20: \hspace{2em} end if
21: \hspace{1em} end for
22: \hspace{1em} Color[$u$] $\leftarrow$ Black
23: end while

A verbal description should be as follows (like a recipe):
Input – a graph $G = (V, E)$ and a starting node $s$

1. Mark all nodes except $s$ as undiscovered, their distance infinity and their predecessor null.
2. Mark $s$ as discovered, its distance 0 and predecessor null.
3. Initialize a simple queue \( Q \) and enqueue \( s \)

4. While \( Q \) not empty:
   
   (a) Dequeue \( Q \), call the node \( u \)
   
   (b) For each of \( u \)'s neighbors \( v \):
      
      i. If \( v \) is undiscovered:
         
         • mark it discovered
         • Set \( \text{dist}[v] \) to be \( \text{dist}[u]+1 \)
         • Set \( v \)'s predecessor to \( u \)
         • Enqueue \( v \) in \( Q \)
   
   (c) Mark \( u \) as finished

That’s it. Please avoid long verbal description. If I don’t understand your description, then you’re doing it wrong.

**Remember:** If we described an algorithm in class you can use it as a "black box". That is, no need to re-describe it. For example, an algorithm to find if a graph is bipartite can be described as follows:

**Input:** Graph = \( G(V, E) \)

1. Run BFS on \( G \) starting from an arbitrary vertex \( s \).

2. Check during the run whether any two vertices with the same distance are connected by an edge.

3. If so – stop and return False.

4. If you finished running BFS without returning False, return True.

**Runtime analysis**

If you are asked to analyze the runtime of an algorithm (or give an algorithm that has a certain runtime), you should provide a short proof by counting the number of steps, provide a recurrence formula (if recursive), or use any result we showed in class.

For example: To show that BFS runs in \( O(m + n) \), we see that there are two loops. The external loop (while the queue is not empty...) runs at most \( n \) times, where \( n \) is the number of vertices, since every vertex is put in the queue at most once. The internal for loop (looking at neighbors) runs \( \text{degree}[v] \) times for each vertex \( v \). Since the sum of degrees in an undirected graph is \( 2m \) where \( m \) is the number of edges (since every edge contributes exactly two to the total degree count) the overall run time is \( O(n + m) \) (another correct answer is \( O(m) \)) since we run BFS for a connected component, and the minimum number of edges in this case is \( n - 1 \), so the \( n \) component is swallowed in \( m \).

To show that the bipartite check runs in \( O(m + n) \), notice that in the worst case we add another \( O(1) \) test to BFS’ inner for loop, when we check whether \( \text{dist}[u] \) is equal to \( \text{dist}[v] \). This doesn’t change the overall order of magnitude of the BFS runtime (yes, that’s the entire proof. Since we already showed that BFS is \( O(m + n) \) we can use it as-is).

**Writing Proofs**

Don’t be afraid of proofs! Most of them make a lot of sense and use a small subset of techniques that you should already be familiar with. In particular, two techniques are often used: Induction and proof by contradiction. I will give two examples that show how to write a complete and rigorous proof.
Induction

In BFS, all the vertices of layer $i$ (namely, at distance $i$ from $s$) enter the queue before all the vertices of layer $i + 1$.

**Proof:**

- **Base case** (don’t forget it!) The induction starts with $i = 0$. This is the vertex $s$ and by design, it is first in the queue.

- **Inductive hypothesis:** Assume the claim hold for $0$ to $i$. We shall see that all the $i + 1$ layer vertices, enter the queue before the $i + 2$ vertices.

- **Proof for $i+1$:** Let $u$ be a vertex in layer $i + 1$, and let $w$ be a vertex of layer $i + 2$. By definition, there are, respectively, vertices $u_1$ and $w_1$ in layers $i$ and $i + 1$, so that $(u_1, u), (w_1, w) \in E$. Note that by the induction hypothesis, $u_1$ will enter the queue before $w_1$ (or any other vertex in layer $i + 1$, that has $w$ as a neighbor). When we dequeue $u_1$, or even before, $u$ will enter the queue (both are at the same layer). This implies that $u$ enters the queue before $w$ (as no $w_1$ vertex with $(w_1, w) \in E$ can have been scanned yet, by the induction hypothesis). The claim follows.

Notice that you always have to use the inductive hypothesis.

**Proof by Contradiction**

At the termination of BFS, if BFS explores $v$, then $\text{dist}[v]$ is the distance from $s$ to $v$.

**Proof by contradiction:**

- Assume there is some vertex with $\text{dist}()$ not equal to the distance from $s$.

- Let $v$ be such a vertex which is the smallest distance from $s$ out of all the vertices with such property.

- Let $u$ be its predecessor on a REAL shortest path from $s$ to $v$.

- This means that $\text{dist}[u] = \text{actual shortest distance from s to u}$ (since $v$ is the closest to $s$ for which $\text{dist}[v] > \text{real distance from s}$).

- By our (contradiction assumption), $\text{dist}[v] > \text{distance from s to v} = (\text{distance from s to u}) + 1 = \text{dist}[u] + 1$

- Consider when $u$ was dequeued. We’ll contradict the above chain of inequalities in any of the three cases:

  - if $v$ wasn’t explored yet, $v$ would have been enqueued with $\text{dist}(v) = \text{dist}(u) + 1$. Contradiction to above inequalities.

  - if $v$ on the queue, then the node that discovered $v$ has lower or equal dist than $u$ does (since it was enqueued before $u$). Then $\text{dist}[v] = \text{dist}(\text{parent}(v)) + 1 \leq \text{dist}(u) + 1$, again contradicting the inequalities.

  - if $v$ has already been dequeued, then $\text{dist}[v] \leq \text{dist}[u]$. Contradiction again (since we said that the $v$ is the closest that violates the distance property, and that vertices are enqueued by order of discovery).

**Things to Remember**

- An example is not a proof! A counter-example can be a proof if you want to show that something is not true. But to show that something IS true, you have to show for the general case and not a specific example.

- An if and only if (iff) proof should include both sides.