**CS1200: Intro. to Algorithms and their Limitations** Prof. Salil Vadhan

# Sender–Receiver Exercise 3: Reading for Senders

*Harvard SEAS - Fall 2025* *2025-10-09*

## 1 Connected Components

The reading for receivers contains Section 1.1 below, as well as the statement of Theorem .2. Your goal will be to communicate the *proof* of Theorem .2 below to the receivers.

### 1.1 Connected Components

We begin by defining the *connected components* of an undirected graph. To gain intuition, you may find it useful to draw some pictures of graphs with multiple connected components and use them to help you follow along the proof.

**Theorem .1.** *Every* undirected *graph $G = (V, E)$ can be partitioned into connected components. That is, there are sets $V_0, \ldots, V_{c-1} \subseteq V$ of vertices such that:*

1. *$V_0, \ldots, V_{c-1}$ are disjoint, nonempty, and $V_0 \cup V_1 \cup \cdots \cup V_{c-1} = V$. (This is what it means for $V_0, \ldots, V_{c-1}$ to be a* partition *of $V$.)*
2. *For every two vertices $u, v \in V$, $u$ and $v$ are in the same component $V_i$ if and only if there is a path from $u$ to $v$.*

*Moreover the sets $V_0, \ldots, V_{c-1}$ are unique (up to ordering), and are called the* connected components *of $V$.*

In case you are interested, we include a proof of this theorem in the Hesterberg-Vadhan textbook (Section 12.5.3), but studying that proof is not required for this exercise.

We remark that for *directed* graphs, one can consider *weakly connected components*, where we ignore the directions of edges, and *strongly connected components*, where two vertices $u, v$ are the same component if and only if there is a directed path from $u$ to $v$ *and* a directed path from $v$ to $u$. Strongly connected components are more useful, but more complicated. In particular, unlike in undirected graphs (or weakly connected components), there can be edges crossing between strongly connected components.

### 1.2 Finding Connected Components via BFS

The main result of this exercise is an efficient algorithm for finding connected components:

**Theorem .2.** *There is an algorithm that, given an undirected graph $G = (V, E)$ with $n$ vertices and $m$ edges, partitions $V$ into connected components in time $O(n + m)$.*

*Proof.* The idea is to do BFS from an arbitrary start vertex $s_0 \in V$, and let our first connected component $V_0$ consist of all the vertices that BFS finds. Then, if there are any vertices in $V \setminus V_0$, we pick an arbitary $s_1 \in V \setminus V_0$, and do BFS from $s_1$ to identify a second component $V_1$, and so on. Naively, this will give a runtime bound $O(c \cdot (n+m))$ where $c$ is the number of connected components, because we do $c$ executions of BFS, each of which could potentially take time $O(n+m)$.

We speed this up by showing that we can implement BFS from a start vertex $s$ in time $O(n_s + m_s)$, where $n_s$ and $m_s$ are the number of vertices and edges, respectively, in the connected component containing $s$. Since we do BFS on distinct connected components (whose sets of vertices and edges are disjoint), our total runtime will just be $O(n+m)$.

However, it's not immediate that BFS from a start vertex $s$ can be implemented in time $O(n_s + m_s)$.

Recall that our implementation of BFS maintained the set $S$ of vertices already visited as an array of $n$ bits (so that membership in $S$ can be tested in constant time). If we re-initialize the array each time we run BFS, our run time will be at least $n$ per BFS execution, regardless of how small the connected component containing $s$ is.

Thus, we modify our description of BFS so that the bit-array $S$ keeping track of the vertices we visit is already initialized as part of the input. It will also be convenient to allow us to use an arbitrary label $\ell$ to indicate which vertices we have visited in the current BFS execution rather than marking them with the bit 1; this will allow us to assign different labels for different connected components.

`BFSlabel`$(G, s, S, \ell)$:
**Input** : A directed graph $G = (V, E)$, a vertex $s \in V$, a label $\ell \in \mathbb{N}$, and an array $S$ of length $n = |V|$ where for every vertex $v$, $S[v] \neq \ell$
**Output** : The array is updated so that $S[v] = \ell$ for every $v$ reachable from $s$, and the other entries of $S$ are unchanged

```
0 S[s] = ℓ;
1 F = {s} ;                                          /* the frontier vertices */
2 d = 0;
3 /* loop invariant:  S[v] = ℓ iff v has distance ≤ d from s, F = vertices at
     distance d from s */
4 while F ≠ ∅ do
5 |   F = {v ∈ V : ∃u ∈ F s.t. (u, v) ∈ E and S[v] ≠ ℓ} ;
6 |   foreach v ∈ F do S[v] = ℓ;
7 |   d = d + 1;
```

**Algorithm .1:** `BFSlabel()`

Similarly to the runtime analysis we did in the previous lecture, the runtime of `BFSlabel`$(G, s, S, \ell)$ can be bounded as

$$O\left(\sum_{d=0}^{\infty} \sum_{u \in F_d} (1 + \deg_{out}(u))\right) \leq O\left(\sum_{u \in R} (1 + \deg_{out}(u))\right).$$

where $F_d$ is the set of vertices $u$ such that $\text{dist}_G(s, u) = d$, and $R = \bigcup_{d=0}^{\infty} F_d$ is the set of vertices reachable from $s$.

Now the key point is that, in an undirected graph $G$, $R$ is exactly the connected component containing $s$, so $|R| = n_s$ and $\sum_{u \in R} \deg_{out}(u) = 2m_s$ (since each of the $m_s$ undirected edges

contributes to $\deg_{out}$ for two vertices). Thus, the run time of `BFSlabel`$(G, s, S, \ell)$ is $O(n_s + m_s)$.

Now we can obtain our algorithm for connected components as follows:

```
BFS-CC(G):
Input   : An undirected graph G = (V, E)
Output  : The number ℓ of connected components in G and a partition of G into
          those components, specified by an array S of length n = |V| with entries
          from [ℓ]
0 Initialize S[v] = ⋆ for all v ∈ V;
1 ℓ = 0;
2 foreach s ∈ V do
3 |   if S[s] = ⋆ then
4 |   |   BFSlabel(G, s, S, ℓ);
5 |   |   ℓ = ℓ + 1;
6 return (ℓ, S)
```

**Algorithm .2:** `BFS-CC()`

To get some intuition for this algorithm, consider how it runs on some examples. For example, if we take the graph in the textbook's Shortest Walks Example 12.2 and Figure 1 (This is the same as figure 12.1 in the textbook).

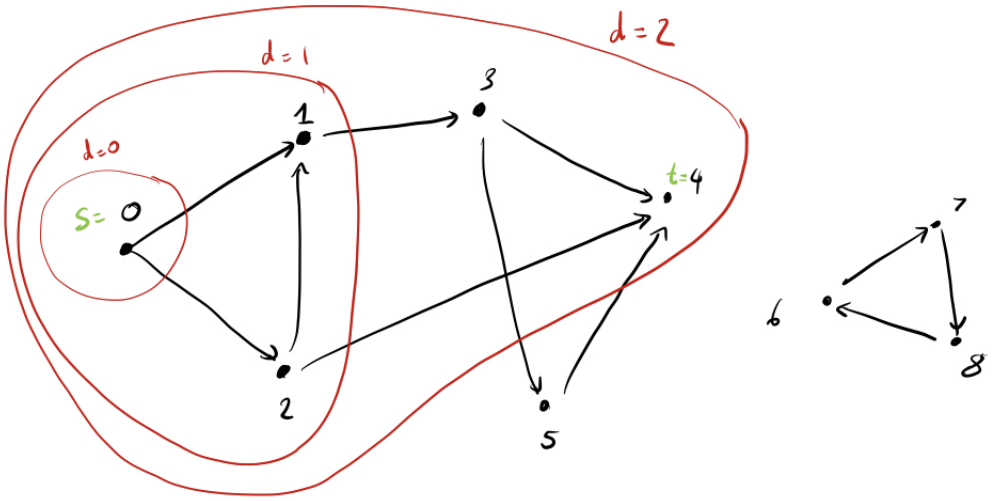


Figure 1: `BFS` on the graph from Example 12.2 with $s = 0, t = 4$

and undirect all of its edges, one intermediate state of the array $S$ would be $S = [0, 0, 0, 0, 0, 0, 1, \star, \star]$, when the algorithm has entirely explored one component and just started exploring the other. If we did the following swaps on vertex numbering $1 \leftrightarrow 7$ and $3 \leftrightarrow 8$ before running the algorithm, an intermediate state of the array would be $[0, 1, 0, \star, 0, 0, \star, 0, 0]$.

For the correctness of this algorithm, we prove the following loop invariant.

**Claim .3.** *At the start of each loop iteration, $S$ has entries from $\{\star, 0, 1, \ldots, \ell-1\}$ with the vertices*

*of each label $i \neq \star$ corresponding to a distinct connected component of $G$.*

*Proof of claim.* We use induction on the number $k$ of loop iterations that have been completed.

The base case ($k = 0$) follows because we initialize $S$ to all $\star$'s.

For the induction step, assume that the claim is true at the start of a loop iteration $k$ and we will argue that it is true at the start of loop iteration $k+1$. The induction hypothesis tells us that at the start of loop iteration $k$, $S$ has entries from $\{\star, 0, 1, \ldots, \ell - 1\}$ with the vertices of each label $i \neq \star$ corresponding to a distinct connected component of $G$.

If $S[s] \neq \star$, then neither $S$ nor $\ell$ change during loop iteration $k$, so the claim also holds at the start of loop iteration $k+1$. If $S[s] = \star$, then $S$ changes in Line 4 and we increment $\ell$ by 1 during loop iteration $k$. Since $S[s] = \star$, we know that $s$ is not in any of the previously labelled connected components and thus `BFSlabel`$(G, s, S, \ell)$ will label the entire connected component of $s$ with label $\ell$ and leave the rest of the array $S$ unchanged. Thus, after Line 4, $S$ has entries from $\{\star, 0, 1, \ldots, \ell\}$ with the vertices of each label $i \neq \star$ corresponding to a distinct connected component of $G$. Since we increment $\ell$, the claim will also hold at the start of loop iteration $k+1$. □

We also observe that due to the loop over $s \in V$, we will be sure to assign every vertex in $V$ to some connected component.

For the runtime, observe all of the executions of Lines 0 to 3 take time $O(n)$ in total. Each time we run Line 4, we execute `BFSlabel`$(G, s, S, \ell)$, which runs in time $O(n_s + m_s)$, where $n_s$ and $m_s$ are the number of vertices and edges in the connected component of $s$. Since we run `BFSlabel` on vertices $s = s_0, \ldots, s_{c-1}$ that are all in different connected components, the cost of all of these executions is

$$O\left(\sum_{i=0}^{c-1}(n_{s_i} + m_{s_i})\right) = O(n + m).$$

□

In the above algorithm, BFS could have easily been replaced with another search strategy like depth-first search (DFS), since we don't care about finding *shortest* paths. It turns out that DFS can be used in a more sophisticated, two-pass fashion, to find the strongly connected components of a directed graph. That algorithm is covered in CS124.