

## Sender–Receiver Exercise 2: Reading for Senders

Harvard SEAS - Fall 2023

2023-10-10

The goals of this exercise are:

1. To develop your skills at understanding, distilling, and communicating proofs and the conceptual ideas in them, especially for proofs in graph theory
2. To deepen your understanding of breadth-first search and its efficiency

Sections 1 and 3, as well as the statement of Theorem 2.1, are also in the reading for receivers. Your goal will be to communicate the *proof* of Theorem 2.1 to the receivers.

## 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 prof.

**Theorem 1.1.** *Every undirected graph  $G = (V, E)$  can be partitioned into connected components. That is, there are sets  $V_0, \dots, V_{c-1} \subseteq V$  of vertices such that:*

1.  $V_0, \dots, V_{c-1}$  are disjoint, nonempty, and  $V_0 \cup V_1 \cup \dots \cup V_{c-1} = V$ . (This is what it means for  $V_0, \dots, V_{c-1}$  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  $V_0, \dots, V_{c-1}$  are unique (up to ordering), and are called the connected components of  $V$ .

In case you are interested, we include a proof of Theorem 1.1 below in Section 1, 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.

## 2 Finding Connected Components via BFS

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

**Theorem 2.1.** *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  $V - V_0$ , we pick an arbitrary  $s_1 \in V - V_0$ , 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.

```

1 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
2  $S[s] = \ell$ ;
3  $F = \{s\}$  ;                               /* the frontier vertices */
4  $d = 0$ ;
5 /* loop invariant:  $S[v] = \ell$  iff  $v$  has distance  $\leq d$  from  $s$ ,  $F$  = vertices at
   distance  $d$  from  $s$  */
6 while  $F \neq \emptyset$  do
7    $F = \{v \in V : \exists u \in F \text{ s.t. } (u, v) \in E \text{ and } S[v] \neq \ell\}$  ;
8   foreach  $v \in F$  do  $S[v] = \ell$ ;
9    $d = d + 1$ ;

```

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

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

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  $\sum_{u \in R} d_{out}(u) = 2m_s$  (since each of the  $m_s$  undirected edges contributes to  $d_{out}$  for two vertices). Thus, the run time of  $\text{BFSlabel}(G, s, S, \ell)$  is  $O(n_s + m_s)$ .

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

```

1 BFS-CC( $G$ )
   Input   : An undirected graph  $G = (V, E)$ 
   Output  : The number  $\ell$  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  $[\ell]$ 
2 Initialize  $S[v] = \star$  for all  $v \in V$ ;
3  $\ell = 0$ ;
4 foreach  $s \in V$  do
5   | if  $S[s] = \star$  then
6   | |    $\text{BFSlabel}(G, s, S, \ell)$ ;
7   | |    $\ell = \ell + 1$ ;
8 return  $(\ell, S)$ 

```

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

**Claim 2.2.** *At the start of each loop iteration,  $S$  has entries from  $\{\star, 0, 1, \dots, \ell - 1\}$  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  $\{\star, 0, 1, \dots, \ell - 1\}$  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 6 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  $\text{BFSlabel}(G, s, S, \ell)$  will label the entire connected component of  $s$  with label  $\ell$  and leave the rest of the array  $S$  unchanged. Thus, after Line 6,  $S$  has entries from  $\{\star, 0, 1, \dots, \ell\}$  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  $k + 1$ .  $\square$

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 2 to 5 take time  $O(n)$  in total. Each time we run Line 6, we execute  $\text{BFSlabel}(G, s, S, \ell)$ , 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  $\text{BFSlabel}$  on vertices  $s = s_0, \dots, s_{c-1}$  that are all in different connected components, the cost of all of these executions is

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

$\square$

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.

### 3 Proof of Theorem 1.1

*Proof.* For every vertex  $u$ , define

$$\llbracket u \rrbracket = \{v : \text{there is a path from } u \text{ to } v \text{ in } G\}.$$

Observe that  $u \in \llbracket u \rrbracket$ ; in particular, the set  $\llbracket u \rrbracket$  is nonempty.

Now let's show that for every two vertices  $u$  and  $w$ , we have either that  $\llbracket u \rrbracket$  and  $\llbracket w \rrbracket$  are disjoint or equal. Suppose they are not disjoint, i.e. there is a vertex  $v \in \llbracket u \rrbracket \cap \llbracket w \rrbracket$ . This means that there is a path  $p_{uv}$  from  $u$  to  $v$  and a path  $p_{wv}$  from  $w$  to  $v$ . Now we argue that  $\llbracket u \rrbracket \subseteq \llbracket w \rrbracket$ . Let  $a$  be any vertex in  $\llbracket u \rrbracket$ , so there is a path  $p_{ua}$  from  $u$  to  $a$ . Then we can get a path from  $w$  to  $a$  by first following the path  $p_{wv}$  to get from  $w$  to  $v$ , then reversing the edges in  $p_{uv}$  to get from  $v$  to  $u$ , and then following the path  $p_{ua}$  to get from  $u$  to  $a$ . Thus,  $a \in \llbracket w \rrbracket$ . Since we showed that this holds for every  $a \in \llbracket u \rrbracket$ , we conclude that  $\llbracket u \rrbracket \subseteq \llbracket w \rrbracket$ . The reverse inclusion  $\llbracket w \rrbracket \subseteq \llbracket u \rrbracket$  is proved in a similar manner.

So now we take  $V_0, \dots, V_{c-1}$  to be all of the distinct sets that occur among those of the form  $\llbracket u \rrbracket$ . Since every vertex  $u \in V$  is in the set  $\llbracket u \rrbracket$ , the sets  $V_0, \dots, V_{c-1}$  will cover all of  $V$ , and by what we just showed, any two distinct sets will be disjoint from each other. This establishes Item 1. Now if a vertex  $u$  is in component  $V_i$ , this means that  $\llbracket u \rrbracket = V_i$  (else  $\llbracket u \rrbracket$  and  $V_i$  would be distinct but not disjoint, contradicting what we showed above). So  $V_i$  contains exactly the vertices  $v$  that are reachable from  $u$ , establishing Item 2.

We omit the proof of uniqueness of the connected components. □

If you have seen equivalence relations, you may recognize some similarity with the above proof. Indeed, the above proof amounts to showing that “ $v$  is reachable from  $u$ ” is an equivalence relation on  $V$ , and then taking the connected components to be the equivalence classes under that relation.