

## Notes 6 for CS 170

### 1 Breadth-First Search

*Breadth-first search (BFS)* is the variant of search that is guided by a *queue*, instead of the stack that is implicitly used in DFS's recursion. In preparation for the presentation of BFS, let us first see what an iterative implementation of DFS looks like.

```
procedure i-DFS(u: vertex)
  initialize empty stack S
  push(u,S)
  while not empty(S)
    v=pop(S)
    visited(v)=true
    for each edge (v,w) out of v do
      if not visited(w) then push(w)

algorithm dfs(G = (V,E): graph)
  for each v in V do visited(v) := false
  for each v in V do
    if not visited(v) then i-DFS(v)
```

There is one stylistic difference between DFS and BFS: One does not restart BFS, because BFS only makes sense in the context of exploring the part of the graph that is reachable from a particular node ( $s$  in the algorithm below). Also, although BFS does not have the wonderful and subtle properties of DFS, it does provide useful information: Because it tries to be “fair” in its choice of the next node, it visits nodes in order of increasing distance from  $s$ . In fact, our BFS algorithm below labels each node with the shortest distance from  $s$ , that is, the number of edges in the shortest path from  $s$  to the node. The algorithm is this:

```
Algorithm BFS(G=(V,E): graph, s: node);
  initialize empty queue Q
  for all  $v \in V$  do  $\text{dist}[v]=\infty$ 
  insert(s,Q)
   $\text{dist}[s]:=0$ 
  while Q is not empty do
    v:= remove(Q),
    for all edges (v,w) out of v do
      if  $\text{dist}[w] = \infty$  then
        insert(w,Q)
         $\text{dist}[w]:= \text{dist}[v]+1$ 
```

For example, applied to the graph in Figure 1, this algorithm labels the nodes (by the array `dist`) as shown. We would like to show that the values of *dist* are exactly the distances

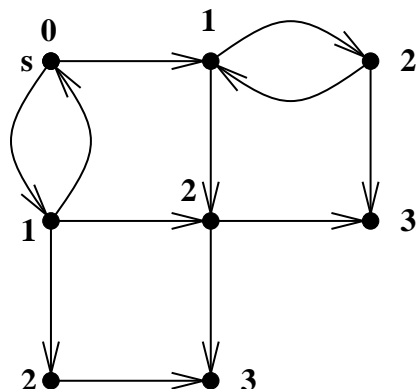


Figure 1: BFS of a directed graph

of each vertex from  $s$ . While this may be intuitively clear, it is a bit complicated to prove it formally (although it does not have to be as complicated as in CLR/CLRS). We first need to observe the following fact.

## LEMMA 1

*In a BFS, the order in which vertices are removed from the queue is always such that if  $u$  is removed before  $v$ , then  $dist[u] \leq dist[v]$ .*

PROOF: Let us first argue that, at any given time in the algorithm, the following invariant remains true:

$$\text{if } v_1, \dots, v_r \text{ are the vertices in the queue then } dist[v_1] \leq \dots \leq dist[v_r] \leq dist[v_1] + 1.$$

At the first step, the condition is trivially true because there is only one element in the queue. Let now the queue be  $(v_1, \dots, v_r)$  at some step, and let us see what happens at the following step. The element  $v_1$  is removed from the queue, and its non-visited neighbors  $w_1, \dots, w_i$  (possibly,  $i = 0$ ) are added to queue, and the vector  $dist$  is updated so that  $dist[w_1] = dist[w_2] = \dots = dist[w_i] = dist[v_1] + 1$ , while the new queue is  $(v_2, \dots, v_r, w_1, \dots, w_i)$  and we can see that the invariant is satisfied.

Let us now prove that if  $u$  is removed from the queue in the step before  $v$  is removed from the queue, then  $dist[u] \leq dist[v]$ . There are two cases: either  $u$  is removed from the queue at a time when  $v$  is immediately after  $u$  in the queue, and then we can use the invariant to say that  $dist[u] \leq dist[v]$ , or  $u$  was removed at a time when it was the only element in the queue. Then, if  $v$  is removed at the following step, it must be the case that  $v$  has been added to queue while processing  $u$ , which means  $dist[v] = dist[u] + 1$ .

The lemma now follows by observing that if  $u$  is removed before  $v$ , we can call  $w_1, \dots, w_i$  the vertices removed between  $u$  and  $v$ , and see that  $dist[u] \leq dist[w_1] \leq \dots \leq dist[w_i] \leq dist[v]$ .  $\square$

We are now ready to prove that the  $dist$  values are indeed the lengths of the shortest paths from  $s$  to the other vertices.

## LEMMA 2

At the end of BFS, for each vertex  $v$  reachable from  $s$ , the value  $dist[v]$  equals the length of the shortest path from  $s$  to  $v$ .

PROOF: By induction on the value of  $dist[v]$ . The only vertex for which  $dist$  is zero is  $s$ , and zero is the correct value for  $s$ .

Suppose by inductive hypothesis that for all vertices  $u$  such that  $dist[u] \leq k$  then  $dist[u]$  is the true distance from  $s$  to  $u$ , and let us consider a vertex  $w$  for which  $dist[w] = k + 1$ . By the way the algorithm works, if  $dist[w] = k + 1$  then  $w$  was first discovered from a vertex  $v$  such that the edge  $(v, w)$  exists and such that  $dist[v] = k$ . Then, there is a path of length  $k$  from  $s$  to  $v$ , and so there is a path of length  $k + 1$  from  $s$  to  $w$ . It remains to prove that this is the shortest path. Suppose by contradiction that there is a path  $(s, \dots, v', w)$  of length  $\leq k$ . Then the vertex  $v'$  is reachable from  $s$  via a path of length  $\leq k - 1$ , and so  $dist[v'] \leq k - 1$ . But this implies that  $v'$  was removed from the queue before  $v$  (because of Lemma 1), and when processing  $v'$  we would have discovered  $w$ , and assigned to  $dist[w]$  the smaller value  $dist[v'] + 1$ . We reached a contradiction, so indeed  $k + 1$  is the length of the shortest path from  $s$  to  $w$ , and this completes the inductive step and the proof of the lemma.  $\square$

Breadth-first search runs, of course, in linear time  $O(|V| + |E|)$ . The reason is the same as with DFS: BFS visits each edge exactly once, and does a constant amount of work per edge.