DFS and BFS Algorithms using Stacks and Queues

Lawrence L. Larmore UNLV

1 The Visitation Problem

Given a directed graph G and a *start vertex* $s \in V[G]$, a *visitation* algorithm for (G, s) is an algorithm which *visits* all vertices of G that are reachable from s. In principle, the *visitation* of a vertex u could consist of any kind of action (such as computing the credit score of the customer represented by the vertex u).

Every vertex reachable from s must be visited exactly once. In order to enforce this rule, we use a Boolean array *visited* on $V[G]$. Initially, *visited*$[u]$ is FALSE for all $u \in V[G]$, and at the end of the algorithm, *visited*$[u]$ is TRUE for all vertices reachable from s, and FALSE for all vertices unreachable from s. The actual act of visiting a vertex u is represented by the assignment statement *visited*$[u] :=$ TRUE.

Let n be the number of vertices of G and m the number of edges. We write $\mathrm{Adj}[v]$ to be the set of out-neighbors of v. A vertex u is an out-neighbor of v if there is a directed edge from v to u. We write $\mathrm{In}[v]$ to be the set of in-neighbors of v. A vertex u is an in-neighbor of v if there is a directed edge from u to v, that is, if v is an out-neighbor of u.

2 Depth First Search

We say that a visitation algorithm is a *depth first search* or DFS, algorithm, if vertices are visited in *depth first* order. The requirements of depth first search are as follows:

1. s is visited first.
2. If v is reachable from s, and $v \neq s$, there is some vertex $u \in \mathrm{In}[v]$ such that u is visited before v is visited. (In some algorithms, one such vertex is labeled the *predecessor* of v.)
3. If $u, v \in V[G]$ are both reachable from s and if v is not reachable from u, and if u is visited before v is visited, then all vertices reachable from u are visited before v is visited.

For a given directed graph and start vertex s, the order of DFS visitation is not necessarily unique.

2.1 Depth First Search Using a Stack

All DFS algorithms, as far as I know, use a stack. It is possible to write a DFS algorithm without an explicit stack data structure by using recursion, but that's "cheating," since you are actually

making use of the run-time stack. Since the purpose of this section is to show how to use a stack for DFS search, we will not allow any of our algorithms to use recursion.

Each of our algorithms uses a stack S of vertices. A vertex can be pushed onto the stack more than once, in fact, the same vertex could be on the stack in several places simultaneously. Being pushed onto the stack does not necessarily imply visitation; we must indicate the visitation of v by explicitly writing $visited[v] :=$ TRUE, and that assignment may not occur more than once.

Here is a stack algorithm for depth first search, which we call DFS-A(G, s).

```
DFS-A(G,s)
   for all v in V[G] do
      visited[v] := false
   end for
   S := EmptyStack
   Push(S,s)
   while not Empty(S) do
      u := Pop(S)
      if not visited[u] then
         visted[u] := true
         for all w in Adj[u] do
            if not visited[w] then
               Push(S,w)
            end if
         end for
      end if
   end while
```

DFS-A can have multiple copies on the stack at the same time. However, the total number of iterations of the innermost loop of DFS-A cannot exceed the number of edges of G, and thus the size of S cannot exceed m. The running time of DFS-A is $O(n + m)$.

It is possible to write a DFS algorithm where no vertex is ever in the stack in more than one place, but it is somewhat trickier. We give such an algorithm, DFS-B, below.

```
DFS-B(G,s)
   for all v in V[G] do
      visited[v] := false
   end for
   S := EmptyStack
   visited[s] := true
   Push(S,s)
   while not Empty(S) do
      u := Pop(S)
      if there is at least one unvisited vertex in Adj[u] then
         Pick w to be any unvisited vertex in Adj[u]
         Push(S,u)
         visited[w] := true
         Push(S,w)
```

```
      end if
   end while
```

No vertex can be on the stack in more than one place. The size of S is thus not more than n. The time complexity of DFS-B is $O(n + m)$.

Questions about DFS-B:

1. Why do we sometimes push u back on the stack after popping it?
2. Prove that the number of times Push is executed is $O(n)$. In fact, Push is executed *exactly* $2n - 1$ times if all vertices are reachable from s. Hint: use amortized analysis.
3. All the statements of the code of DFS-B are executed $O(n)$ times. Why can't we say that the time complexity of DFS-B is $O(n)$? Hint: Execution of Line 10 or 11 takes more than one time step. How would you actually implement those lines?

3 Breadth First Search

We say that a visitation algorithm is a *breadth first search* or BFS, algorithm, if vertices are visited in *breadth first* order. If v is reachable from s, let $d[v]$ be the length of the shortest path from s to v. Breadth first search must satisfy the following requirements:

1. If v is reachable from s, and $v \neq s$, there is some vertex $u \in \text{In}[v]$ such that u is visited before v is visited. (In some algorithms, one such vertex is labeled the *predecessor* of v.)
2. If u and v are both reachable from s, and if $d[u] < d[v]$, then u must be visited before v is visited.

For a given directed graph and start vertex s, the order of a BFS is not necessarily unique.

3.1 Breadth First Search Using a Queue

We will use a queue of vertices in our breadth first search algorithms. Here is one such algorithm.

```
BFS-A(G,s)
   for all v in V[G] do
      visited[v] := false
   end for
   Q := EmptyQueue
   Enqueue(Q,s)
   while not Empty(Q) do
      u := Dequeue(Q)
      if not visited[u] then
         visted[u] := true
         for all w in Adj[u] do
            if not visited[w] then
```

```
            Enqueue(Q,w)
         end if
      end for
   end if
end while
```

In BFS-A, a vertex could be in Q in several places simultaneously. The number of iterations of the innermost loop of BFS-A does not exceed m, and therefore the size of Q never exceeds m The running time of BFS-A is $O(n + m)$.

On the other hand, algorithm BFS-B never has multiple copies of a vertex in Q.

```
BFS-B(G,s)
   for all v in V[G] do
      visited[v] := false
   end for
   Q := EmptyQueue
   visited[s] := true
   Enqueue(Q,s)
   while not Empty(Q) do
      u := Dequeue(Q)
      for all w in Adj[u] do
      if not visited[w] then
         visited[w] := true
         Enqueue(Q,w)
      end if
   end while
```

In BFS-B, the size of Q is never larger than n. The running time of BFS-B is $O(n + m)$.