BFS and DFS

March 23, 2018

1 BFS

Let $G = (V, E)$ be a graph. The following is an algorithm for BFS.

Algo BFS

1. Input: Directed Graph $G = (V, E)$.
2. $Visited = \emptyset$, $Q = \emptyset$.
3. For every $v \in V$, if $v \notin Visited$
   (a) $BFS(G, v)$.

Procedure $BFS(G, v)$

1. Place v in $Visited$ and (at the end of) Q.
2. While $Q \neq emptyset$
   (a) $x =$ first element of Q. Remove x from Q.
   (b) Output x.
   (c) For every $\langle x, y \rangle \in E$
      i. if $y \notin Visited$, add y to visited and place at the end of Q.

Time Complexity of the BFS Algorithm: We can implement Q as a linked list. So, the time taken to remove the fist element of Q and add an element to the end of the Q takes $O(1)$ time. Let us assume that it takes $f(n)$ to place to check if $v \in Visited$ and $g(n)$ time to add v to $Visited$. We make following observations:

Observation 1: Every vertex x of the Graph is placed added to $Visited$ and to Q, and is never removed from $Visited$. A vertex x can be placed into $Visited$, during Step 2(c)i, if that does not happen, then the algorithm will make a call $BFS(G, x)$ in Step 3a, which will place x into $Visited$. We never remove a vertex from $Visited$. Whenever a vertex is placed in to $Visited$, it is also added to Q.

Observation 2: If a vertex $x \in Q$, then $x \in Visited$. This is because, whenever a vertex is added to Q, it is also added to $Visited$ (either in Step 2(c)i or in Step 1).

Observation 3: Once a vertex x is removed from Q, it is never placed in Q again. This is because if x is removed from Q it must have been in $Visited$ (by Observation 2) and is never removed from $Visited$ (by Observation 1). Note that a vertex is placed in Q, only when it is not in $Visited$. Thus x can not be placed back into Q again.

By combining all three observations, we can claim the following: For every $x \in G$, Step 2c is performed exactly once. Now let us fix a vertex x and analyze the time taken by Step 2c due to x. Note that the body of the loop is performed exactly $d(x)$ many times, where d is the outdegree of x. Each iteration takes atmost $O(f(n) + g(n))$ time. Thus the total time taken by Step 2c is $O(d(x)[f(n) + g(n)])$. Thus the total time taken by the algorithm is

$$O(\sum_{x \in V} [d(x)[f(n) + g(n)]]) = O(m(f(n) + g(n))$$

If we use array for $Visited$, then $f(n) = O(n)$ and $g(n) = O(1)$; If we use Balanced Search Tree for $Visited$, then both $f(n)$ and $g(n)$ are $O(\log n)$. If we used Hash table for $Visited$, then both $f(n)$ and $g(n)$ are $O(1)$ on average.

If $V = \{1, 2, \cdots, n\}$, then we can use a Bit Array to represent $Visited$. This can be done as follows: In Step 2, create an array $Visited$ of size n and initialize every $Visited[i]$ to 0. To add y to $Visited$, we set $Visited[y] = 1$; To check if $y \in Visited$, we check if $Visited[y]$ equals 1 or not. Thus both $f(n)$ and $g(n)$ are $O(1)$ in worst-case. Thus the time taken is $O(m + n)$.

We can use the BFS algorithm to assign BFS number to every vertex and to construct a BFS Tree. We can represent a BFS Tree with an array (lets call it BFSTree) of size n, where $BFSTree[i]$ store the parent of i.

Algo BFS Tree

1. Input: Directed Graph $G = (V, E)$. $V = \{1, \cdots, n\}$.
2. $Visited[i] = 0$ for every $1 \leq i \leq n$. $Q = \emptyset$.
3. $BFSTree[i] = 0$ for every $1 \leq i \leq n$.
4. $Counter = 0$.
5. $BFSNum[i] = 0$ for every $1 \leq i \leq n$.
6. For every $v \in V$, if $Visited[v] \neq 0$
   (a) $BFS(G, v)$.

Procedure $BFS(G, v)$

1. $counter++$; $BFSNum[v] = counter$.
2. Place v in $Visited$ (by Setting $Visited[v] = 1$) and (at the end of) Q.
3. While $Q \neq emptyset$
   (a) $x =$ first element of Q. Remove x from Q.
   (b) Output x.
   (c) For every $\langle x, y \rangle \in E$
      i. if $y \notin Visited$, add y to visited and place at the end of Q; $BFSTree[y] = x$.

2 DFS

Let $G = (V, E)$ be a directed graph. We now describe DFS algorithm.

Algo DFS

1. Input: Directed Graph $G = (V, E)$.
2. Unmark every vertex of V.
3. For every $v \in V$
   - if v is unmarked, Call $DFS(G, v)$.

Where the procedure $DFS(G, v)$ (which is a recursive procedure) is as follows:

Procedure $DFS(G, v)$

1. Mark v. Output v.
2. For every u such that $\langle v, u\rangle \in E$
   - if u is unmarked, $DFS(G, u)$.

Let us consider the following graph. Vertex set is A, B, C, D. The edges are $\langle A, B\rangle$, $\langle A, C\rangle$, $\langle C, B\rangle$, $\langle B, D\rangle$, $\langle C, D\rangle$

Consider the loop in **Algo DFS**. Suppose the first vertex that we picked is A and called $DFS(A)$. Then the recursive call works as follows:

1. $DFS(G, A)$: Mark A and output A.
2. A has two outgoing edges to B and to C. Suppose it picks B and calls $DFS(G, B)$.
   (a) $DFS(G, B)$: Mark B and output B.
   (b) B has only one outgoing edge to D and D is unmarked. So Make a call to $DFS(G, D)$.
      i. $DFS(G, D)$: Mark D and output D.
      ii. D does not have any outgoing edge. So $DFS(G, D)$ is completed.
   (c) Since B has only one outgoing edge to D, and D is already marked. $DFS(G, B)$ is completed.
3. Look at the other outgoing edge from A to C. Since C is unmarked, call $DFS(G, C)$.
   (a) $DFS(G, C)$: Mark C and output C.
   (b) C has two outgoing edges to B and D. Since both B and D are marked, $DFS(G, C)$ is completed.
4. $DFS(G, A)$ is completed

The output of the algorithm is A, B, D, C: This is a DFS order.

We can modify the above basic algorithm to assign a DFS number to each vertex and to build the DFS Tree/Forest.

Algo DFS

1. Input: Directed Graph $G = (V, E)$.
2. Unmark every vertex of V.
3. counter = 0;
4. DFSTree[v] = null for every $v \in V$.
5. DFSNum[v] = -1 for every $v \in V$.
   - if v is unmarked, Call $DFS(G, v)$.

Where the procedure $DFS(G, v)$ (which is a recursive procedure) is as follows:

Procedure $DFS(G, v)$

1. Mark v; counter++;
2. DFSNum[v] = counter;
3. For every u such that $\langle v, u \rangle \in E$
   - if u is unmarked
     - DFSTree[u] = v;
     - $DFS(G, u)$.

What is the DFS tree/forest constructed and the DFS numbers of vertices if we invoke the above algorithm on the following graph: The graph has six vertices named A, B, C, D, E, F. The edges are as follows: A to B, B to A, C to A, D to F,DtoB, E to D, F to D and E to F.