DFS and Applications

1 DFS

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

Algo DFS

```
1. Input: Directed Graph $G = (V, E)$.
2. Set $Discover[u]$ to $F$ for every $u \in V$.
3. For every $v \in V$
    • if $Discover[v]$ is $F$, Call $DFS(G, v)$.
```

Where the procedure

Procedure $DFS(G, v)$

```
1. $Discover[v] = T$
2. For every $u$ such that $\langle v, u\rangle \in E$
    • if $Discover[u]$ is $F$, $DFS(G, u)$.
```

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. $DFS(G, A)$: Discover 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) $DFS(G, B)$: Discover B and output B.
   (b) B has only one outgoing edge to D and D is not Discovered So Make a call to $DFS(G, D)$.
      i. $DFS(G, D)$: Discover 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 Discovered. $DFS(G, B)$ is completed.
3. Look at the other outgoing edge from A to C. Since C is not Discovered, 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 Discovered, $DFS(G, C)$ 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. Set $Discover[u]$ to $F$ for every $u \in V$.
3. counter = 0;
4. DFSTree[v] = null for every $v \in V$.
5. DFSNum[v] = -1 for every $v \in V$.
6. For every $uinV$
     • if $Discover$
```

Where the procedure

Procedure $DFS(G, v)$

```
1. $Discover[v] = T$; counter++;
2. DFSNum[v] = counter;
3. For every $u$ such that $\langle v, u \rangle \in E$
     • if $Discover[u]$ is $F$
         – 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.

We make a few observations about the DFS algorithm.

Observation 1: On every vertex v, the above algorithm invokes the call $DFS(G, v)$ exactly once. This is because the loop in **DFSAlgo** looks at every undiscovered vertex and makes the call $DFS(G, v)$ if v is undiscovered . Since all the vertices are initially undiscovered , this ensures that the algorithm calls $DFS(G, v)$ for every v at least once. Look at the procedure, $DFS(G, v)$. The first step in the algorithm is to discover v. Also note that a call $DFS(G, v)$ is made only when v is undiscovered . This ensures that $DFS(G, v)$ is called exactly once for each vertex.

Observation 2: Time taken my the above algorithm is $O(m+n)$ where m is the number of edges and n is number of vertices. This is because the algorithm considers each edge (in the loop in **Procedure DFS** exactly once.

Observation 3: Suppose that there is an edge from u to v in a graph. If the call $DFS(G,u)$ is made before $DFS(G,v)$, then $DFS(v)$ is completed before $DFS(G,u)$ is completed.

Observation 4: Suppose that there is no path from u to v in G. Then $DFS(G,u)$ will not make a recursive call to $DFS(G,v)$.

2 Start and End Times

The order in which recursive DFS calls are made and the order in which recursive DFS calls are completed are critical in developing some graph algorithms. We introduce these notions.

Start and End Time Algo

```
1. Input $G = (V, E)$ which is a DAG.
2. Set $Discover[u]$ to $F$ for every $u \in V$..
3. Global Variables: Startcounter = 0; EndCounter = 0;
    • if $Discover[v]$ is $F$, Call $DFS(G,v)$.
```

Procedure DF

```
1. StartCounter++;
2. StartTime[v] = StartCounter;
3. $Discover[v] = T$;
4. For every $u$ such that $\langle v, u\rangle \in E$
    (a) if $Discover[v]$ is $F$, $DFS(G,u)$.
5. Endcounter++;
6. EndTime(v) = Endcounter;
```

For example, if we run the above algorithm from from the following graph: Vertex set is A, B, C, D. The edges are $\langle A, B\rangle$, $\langle A, C\rangle$, $\langle C, B\rangle$, $\langle B, D\rangle$, $\langle C, D\rangle$. Then $StartTime(A) = 1$, $StartTime(B) = 2$, $StartTime(D) = 3$, $StartTime(C) = 4$, and $EndTime(A) = 4$, $EndTime(B) = 2$, $EndTime(C) = 3$, and $EndTime(D) = 4$.

Sometimes we use only one global variable to keep track of start and end times.

Start and End Time Algo

```
1. Input $G = (V, E)$ which is a DAG.
2. Set $Discover[u]$ to $F$ for every $u \in V$..
3. Global Variable: Counter = 0;
     • if $Discover[v]$ is $F$, Call $DFS(G, v)$.
4. Output vertices in reverse Finish Time Order.
```

Procedure $DFS(G, v)$

```
1. Counter++;
2. StartTime[v] = Counter;
3. $Discover[v] = T$;
4. For every $u$ such that $\langle v, u\rangle \in E$.
   (a) if $Discover[v]$ is $F$, $DFS(G, u)$.
5. Counter++;
6. EndTime(v) = Coun
```

What are the start and end times on the graph from previous exam
Recall that along with the DFS algorithm, we can construct a DFS t G be a directed graph and T be its DFS tree. The edges of G that belo *dges.* Let us look at non-tree edges. We classify them into three classes

- *Back Edges.* These is an edge of the graph $\langle x, y\rangle$ such that y is an ancestor of x in the DFS tree T. An edge $\langle x, y\rangle$ is a *back edge* if and only if the following conditions hold
  - $StartTime(y) < StartTime(x)$
  - $EndTime(y) > EndTime(x)$.
- *Forward Edges.* This is an edge of the graph $\langle x, y\rangle$ such that x is an ancestor of y in the DFS tree T. An edge $\langle x, y\rangle$ is a *forward edge* if and only if the following conditions hold
  - $StartTime(x) < StartTime(y)$
  - $EndTime(x) > EndTime(y)$.
- *Cross Edges.* All non-trees edges that are neither back edges nor forward edges are classified as cross edges.

Using start and end times, we can quickly classify the non-tree edges.

3 Topological Sorting

We now apply the DFS algorithm to find a topological ordering of vertices in a Directed Acyclic Graph (DAG). A directed graph is called a *DAG* if it has no cycles. Let $G = (V, E)$ be a DAG and *topological sorting/order* of V is a sequence $x_1, x_2, \cdots, x_n$ such that if $\langle x, y\rangle$ is an edge in G, then x appears before y in the sequence. Ie., if $x = x_i$ and $y = y_j$, then $i < j$.

Given a graph we need to compute topological ordering of vertices. Let us start with an example; Consider the following "diamond" graph with four vertices A, B, C, and D with following edges: A to B, A to C, B to D and C to D. Suppose we perform DFS on this graph. The first DFS call that we make is arbitrary (it could be $DFS(G, A)$, or $DFS(G, B)$ etc).

Suppose we the first DFS call is $DFS(G, A)$: which DFS finishes first? Is it $DFS(G, A)$ or $DFS(G, B)$, or $DFS(G, C)$ or $DFS(G, D)$? That would be $DFS(G, D)$—which is the last vertex in the topological ordering. What if the first DFS call is $DFS(G, C)$? Even in this case also, the $DFS(G, D)$ finishes first. Order the vertices based on the finishing time of DFS. This is exactly the reverse topological order. This suggests the following algorithm:

Topological Order Algo

```
1. Input $G = (V, E)$ which is a DAG.
2. Set $Discover[u]$ to $F$ for every $u \in V$.
3. counter = 0;
     • if $Discover$
4. Output vertices in rev
```

Procedure $DFS(G, v)$

```
1. $Discover[v] = T$;
2. For every $u$ such that $\langle v, u\rangle \in E$
     (a) if $Discover[v]$ is $F$, $DFS(G, u)$.
3. counter++;
4. EndTime(v) = counter;
```

We now claim the the above algorithm produces a correct topological ordering. The following observation is obvious.

Observation 5. If $DFS(G, v)$ finishes before $DFS(G, u)$, then $EndTime(v) > EndTime(u)$.

Now suppose that $\langle u, v\rangle \in E$. We claim that $EndTime(u) > EndTime(v)$. We consider two cases.

Case 1: $DFS(G, u)$ is called before $DFS(G, v)$. By Observation 3, $DFS(G, v)$ finishes before $DFS(G, u)$. Thus by Observation 5, $EndTime(u) > EndTime(v)$.

Case 2: $DFS(G, v)$ is called before $DFS(G, u)$. Note that since G is a DAG, there is no path from u to v. Thus by Observation 4, $DFS(G, v)$ will not make a recursive call to $DFS(G, u)$. Thus $DFS(G, v)$ finishes before $DFS(G, u)$. Thus $EndTime(u) > EndTime(v)$.

Note that the time taken by the above algorithm is $O(m + n)$.