CSCE 411: Design and Analysis of Algorithms

Lecture 12: Graph Algorithms: DFS

Date: February 27 *Nate Veldt, updated by Samson Zhou*

1 Depth First Search Algorithm

Unlike in a BFS, a depth-first search (DFS):

- Explores the *most recently discovered vertex* before backtracking and exploring other previously discovered vertices
- All nodes in the graph are explored (rather than just a DFS for a single node s)
- We keep track of a global *time*, and each node is associated with two timestamps for when it is *discovered* and *explored*.

Each node $u \in V$ is associated with the following attributes

Attribute	Explanation	Initialization
u.status	tells us whether a node has been *undiscovered*, *discovered*, and *explored*	u.status $= U$
u.D	timestamp when u is first discovered	NIL
u.F	timestamp when u is finished being explored	NIL
u.parent	predecessor/"discoverer" of u	NIL

DFS(G)

for $v \in V$ **do**
  v.parent $= NIL$
  v.status = U
end for
time $= 0$
for $u \in V$ **do**
  if u.status $== U$ **then**
    DFS-Visit(G, u)
  end if
end for

DFS-Visit(G, u)

time = time + 1
$u.D$ = time
u.status $= D$
for $v \in$ Adj$[u]$ **do**
  if v.status $== U$ **then**
    v.parent $= u$
    DFS-Visit(G, v)
  end if
end for
u.status = E
time = time + 1
$u.F$ = time

1.1 Runtime Analysis

Question 1. *What is the runtime of a depth first search, assuming that we store the graph in an adjacency list, and assuming that* $|E| = \Omega(|V|)$*?*

A $O(|V|)$

B $O(|E|)$

C $O(|V| \times |E|)$

D $O(|V|^2)$

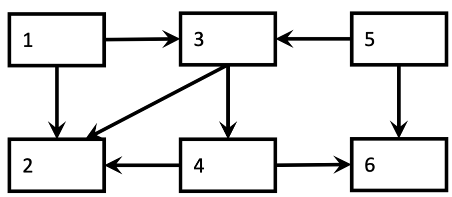
E $O(|E|^2)$

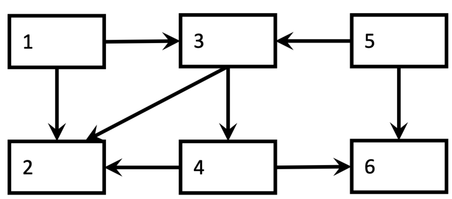
1.2 Properties of DFS

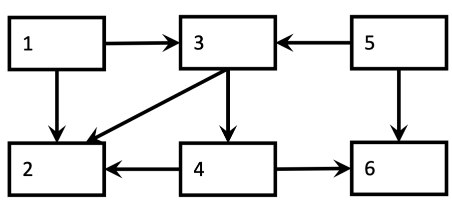
Theorem 1.1. *In any depth-first search of a graph $G = (V, E)$, for any pair of vertices u and v, exactly one of the following conditions holds:*

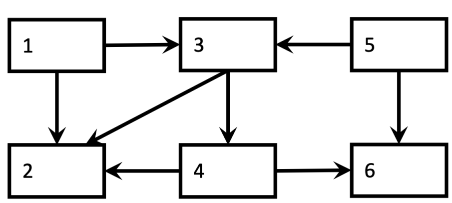
- $[u.D, u.F]$ *and* $[v.D, v.F]$ *are disjoint;* ____________________________________
- $[v.D, v.F]$ *contains* $[u.D, u.F]$ *and* ____________________________________
- $[u.D, u.F]$ *contains* $[v.D, v.F]$ *and* ____________________________________

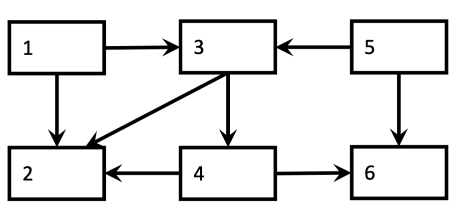
We will not prove this, but we'll give a quick illustration











1	1	2	3	4	5	6	7	8	9	10	11	12
2	1	2	3	4	5	6	7	8	9	10	11	12
3	1	2	3	4	5	6	7	8	9	10	11	12
4	1	2	3	4	5	6	7	8	9	10	11	12
5	1	2	3	4	5	6	7	8	9	10	11	12
6	1	2	3	4	5	6	7	8	9	10	11	12

Corollary 1.2. *v is a descendant of u* $\iff$

1.3 Classification of Edges

Given a graph $G = (V, E)$ performing a DFS on G produces a graph $\hat{G} = (V, \hat{E})$ where

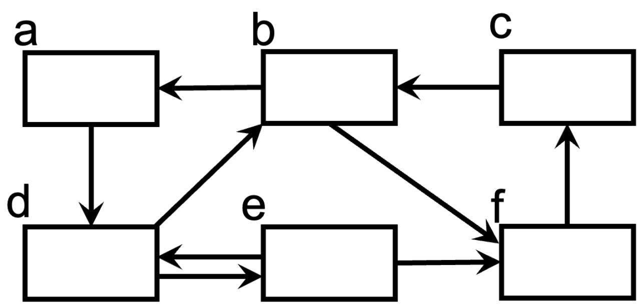
$$\hat{E} = \{(u.\text{parent}, u) \colon v \in V \text{ and } v.\text{parent} \neq NIL\}$$

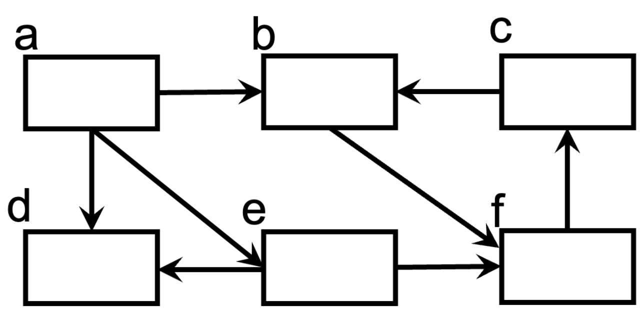
This is called a *depth-first* forest of G.

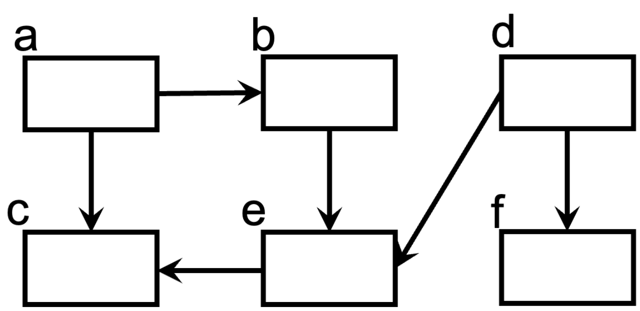
Given any edge $(u, v) \in E$, we can classify it based on the status of node v when we are performing the DFS:

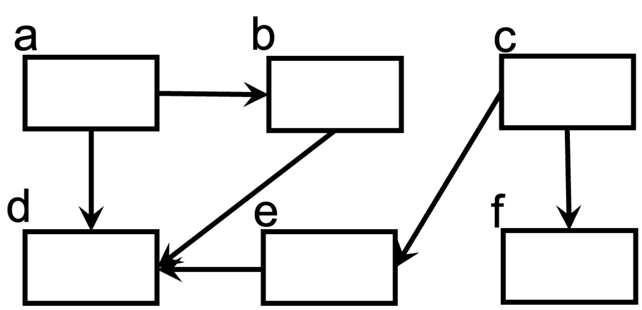
Edge	Explanation	How to tell when exploring (u, v)?
Tree edge	edge in $\hat{E}$	
Back edge	connects u to ancestor v	
Forward edge	connects vertex u to descendant v	*and* $u.D < v.D$
Cross edge	either (a) connects two different trees or (b) crosses between siblings/cousins in same tree	*and* $u.D > v.D$

1.4 Practice









Question 2. *How many of the above graphs were directed acyclic graphs?*

A *1*

B *2*

C *3*

D *4*

E *none of them*

2 Application 1: Checking if G is a DAG

Theorem 2.1. *G is a DAG $\iff$ a DFS yields no back edges. Equivalently:*

Proof First, ($\implies$) we show that if DFS yields a back edge, G is not a DAG.

Next ($\Leftarrow$) we show that if G is not a DAG there will be a back edge.

3 Application 2: Topological Sort

Given a directed acyclic graph $G = (V, E)$, a topological sort of G is an ordering of nodes such that for any $(u, v) \in E$, u comes before v in the ordering.

We can use the following procedure to solve the topological sort problem:

1.

2.

Theorem 3.1. *Ordering nodes in a directed acyclic graph $G = (V, E)$ by reversed finish times will produce a topological sort of G.*

Proof. 1. Let (u, v) be an edge in G

2. Our goal is to show that

   ____

3. When (u, v) is explored, there are three different possibilities for the status of v:

   - **Case 1**: v.status $== U$. This means v becomes a descendant of u.

     Thus, $v.F < u.F$. Reason: _______________________

   - **Case 2**: v.status $== E$, then we also have $v.F < u.F$.

     Reason:

   - **Case 3**: v.status $== D$, this means that v is an ancestor of u, so (u, v) is a back edge.

     But this is impossible. Reason: _______________________

4. In all cases that are possible, ____________________

$\square$