# EE 360C - ALGORITHMS

## Lecture 10

## Graphs 3

**Vallath Nandakumar**

**University of Texas at Austin**

If he [Thomas Edison] had a needle to find in a haystack, he would not stop to reason where it was most likely to be, but would proceed at once with the feverish diligence of a bee, to examine straw after straw until he found the object of his search. … [J]ust a little theory and calculation would have saved him ninety percent of his labor.

— Nikola Tesla

# Summary of last class

☞ Last class:
- Bipartite graphs
- DAG
- Topological ordering

# This class

- ☞ HW 2 solutions
- ☞ DFS review
- ☞ Using DFS to generate topological order in Java

# The Depth-First Search algorithm

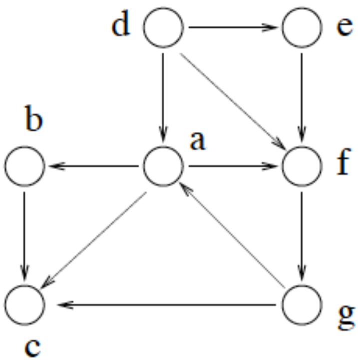
- ☞ Given for digraphs but can easily be modified for undirected graphs
- ☞ After processing a vertex it recursively processes *all* of its descendants
- ☞ Running time analysis of DFS

# DFS

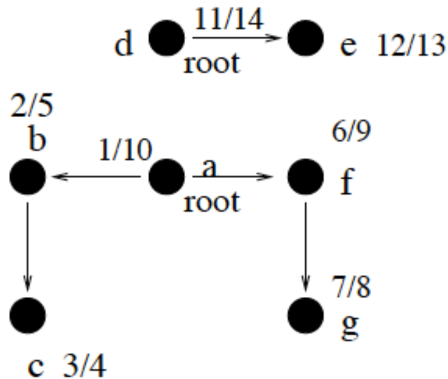
- ☞ Graph is  $G = (V, E)$. The algorithm works in discrete time steps.  Each vertex $v$ is given a “discovery” time $d[v]$ when it is first processed and a “finish” time $f[v]$, when all of its descendants are finished.
- ☞ The output is a collection of  trees. As well as $d[v]$ and $f[v]$, each node points  to *pred[v]*,  its parent in the forest.

# DFS Forest

- DFS creates a forest $F = (V, E_f)$, a collection of rooted trees, where
  $$E_f = \{(pred[v], v) \mid \text{where DFS calls are made}\}$$
- Forest can be stored in an array of predecessors, e.g.




original graph




Two source vertices a, d

# Idea of the DFS Algorithm

- ☞ In DFS, edges are explored out of the most recently discovered vertex *v*. Only edges to unexplored vertices are explored.
- ☞ When all of *v*'s edges have been explored, the search "backtracks" to explore edges leaving the vertex from which was discovered.
- ☞ The process continues until we have discovered all the vertices that are reachable from the original source vertex.
- ☞ If any undiscovered vertices remain, then one of them is selected as a new source vertex, and the search is repeated from that source vertex.
- ☞ This process is repeated until all vertices are discovered.

# iClicker – Shortest Path

Which of DFS and BFS yields the shortest path between two vertices for an undirected graph? Pick the *best* answer.

A. BFS always, DFS can’t tell beforehand
B. BFS sometimes, DFS never
C. BFS sometimes, DFS for tree graphs
D. Both BFS and DFS for trees only
E. BFS always, DFS for trees always

```
DFS()
     Create empty set Visited of visited vertices.
     For each v in G, mark v unvisited.
     For each v in Visited, mark v's predecessor null.
     For each v in G
          Call DFS(v, Visited)
DFS(v, Visited)
     Add v to Visited
     For each Edge e in v's adjacency list
          If e's destination dest is not visited
               Mark dest's predecessor as v.
               Call DFS(dest, Visited)
```

```
DFS()
    Create empty set Visited of visited vertices.          1
    For each v in G, mark v's predecessor null.        n
    For each v in G
        if (v is not in Visited)                             n tests
            Call DFS(v, Visited)
DFS(v, Visited)
    Add v to Visited                                       1
    For each Edge e in v's adjacency list
        If e's destination dest is not visited  outdeg(v) tests
            Mark dest's predecessor as v.             <= outdeg(v)
            Call DFS(dest, Visited)
```

$$\text{const.n} + \text{const} + \sum(\text{outdeg}(v)) \le T \le \text{const.n} + \text{const.} + \sum(2^*\text{outdeg}(v))$$

# Time Complexity bound of DFS

- ☞ $f(n + m) <= T <= f(n + 2*m)$
- ☞ T is $O(n+m)$
- ☞ T is $\Omega(n+m)$
- ☞ T is therefore $\Theta(n+m)$

# DFS for topological sort

- ☞ If *G = (V, E)* is a DAG then a topological sorting of *V* is a linear ordering *V* of such that for each edge *(u, v)* in the DAG, *u* appears *v* before in the linear ordering.
- ☞ Idea of Topological Sorting: Run the DFS on the DAG and output the vertices in reverse order of finishing time.
  - In an edge (u, v) of the DFS tree, vertex u starts first and it finishes last.
  - In a DAG, there are no edges that go back to an already discovered vertex, because then there will be a cycle.
  - So the vertices may be ordered as above, and no edge will go from a higher number to a lower number.

# The Algorithm

- ☞ Run DFS on DAG G, starting from vertices with no incoming edges
- ☞ As each vertex *v*'s DFS(*v*) finishes, insert it into the front of a list
- ☞ Output the list
- ☞ Running time: $\Theta(n+m)$, the same as DFS
- ☞ Example:

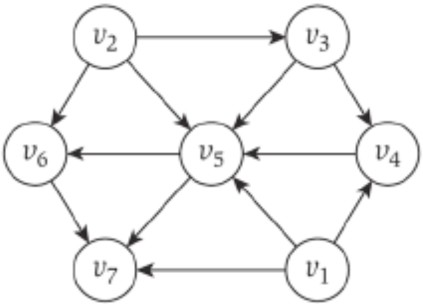
https://www.cs.usfca.edu/~galles/visualization/TopoSortDFS.html

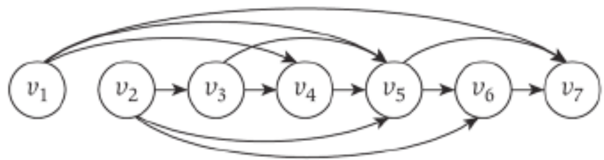
# iClicker

☞ What happens if you run DFS as shown previously on a directed graph with cycles?

A. You won't get a DFS tree at all, because you will miss some vertices
B. You will get a graph with cycles instead of a tree
C. You will get a DFS tree, but order of vertices will never be topological
D. You will get a DFS tree, and might get a topological order

# Topological order example



```
public void genDFSForest () {
    this.resetAll();
    List<Vertex> sorted = new LinkedList<Vertex>();
    Set<Vertex> visited = new HashSet<Vertex>();
    // Find all vertices that have no incoming edges
    Set<Vertex> startVertices = new
HashSet<Vertex>(vertices.values());
    for (Vertex v: vertices.values()) {
        for (Edge e: v.adjacency)
            startVertices.remove(e.dest);
    }
    for (Vertex startVertex: startVertices) {

        genDFSForest(sorted, visited, startVertex);
    }
    System.out.println(sorted);
}
```

```java
    private void genDFSForest(List<Vertex> sorted, Set<Vertex>
visited, Vertex startVertex) {
        visited.add(startVertex);
        for (Edge e: startVertex.adjacency) {
            Vertex v = e.dest;
            if (!visited.contains(v)) {
                genDFSForest(sorted, visited, e.dest);
            }
        }
        sorted.add(0, startVertex);
    }
```

# Summary

- ☞ We looked at DFS algorithm
- ☞ Topological sort using DFS
- ☞ Java program to do topological sort using DFS