Algorithm Design & Analysis (CSE222)

Lecture-13

Recap

- Graph Notations

- DFS Revisit

Outline

- DFS Revisit

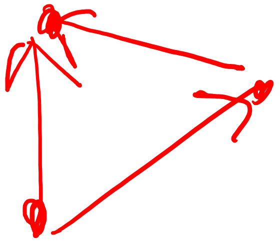
- Applications of DFS

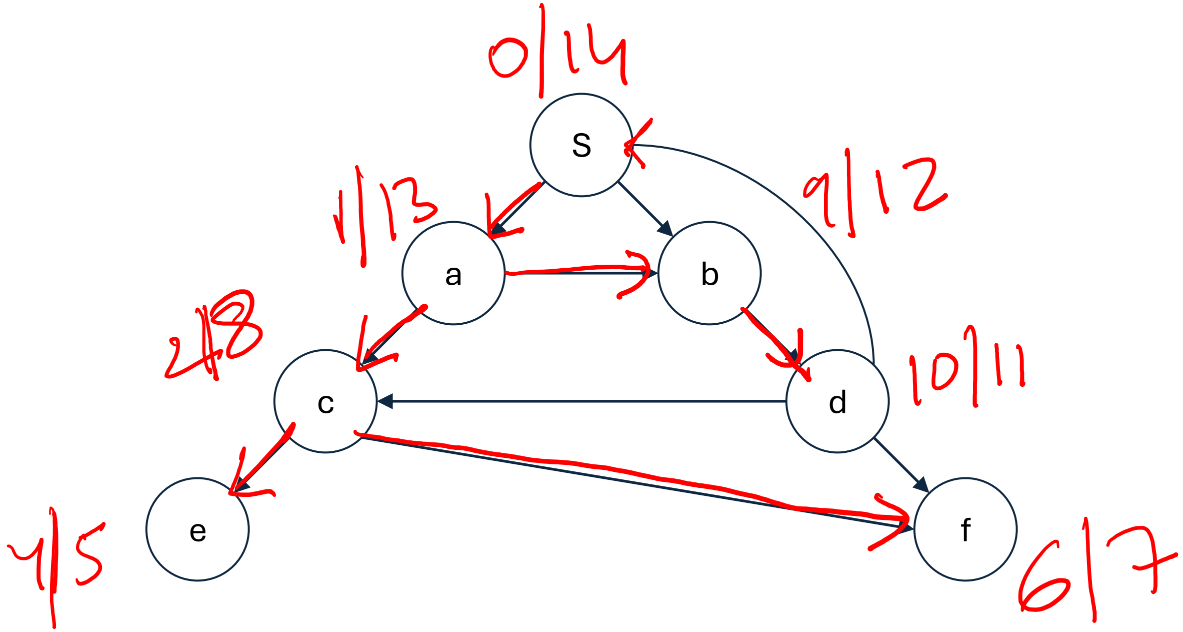
Depth First Search

```
DFS(G)
for each u ∈ G.V
    u.color = WHITE
time = 0
for each u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY                    // discover u
for each v ∈ G.Adj[u]             // explore (u, v)
    if v.color == WHITE
        DFS-VISIT(v)
u.color = BLACK
time = time + 1
u.f = time                        // finish u
```

Entry and Exit time





DFS Tree Edge Types

Let DFS starts at node s and returns a tree.

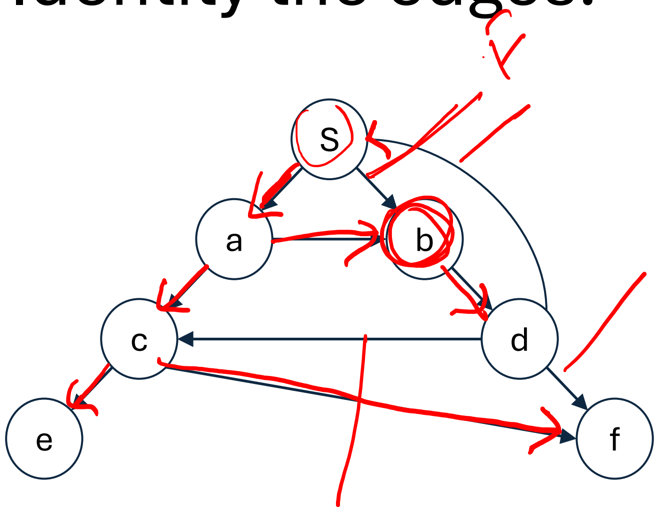
We call an edge (u, v) a **<u>tree edge</u>** if it is present it the returned tree.

The rest of the edges in the graph, which are non-tree edges, can further be classified as back edges, forward edges, and cross edges.

An edge $(u, v) \in E$ is called **<u>back edge</u>**, if v is an ancestor of u in DFS tree.
An edge $(u, v) \in E$ is called **<u>forward edge</u>**, if v is decentend of u in DFS tree.
An edge $(u, v) \in E$ is called **<u>cross edge</u>**, if v is neither ancestor nor a decentend of u in DFS tree.

Example

Identity the edges.



u, v

(S, B) – F

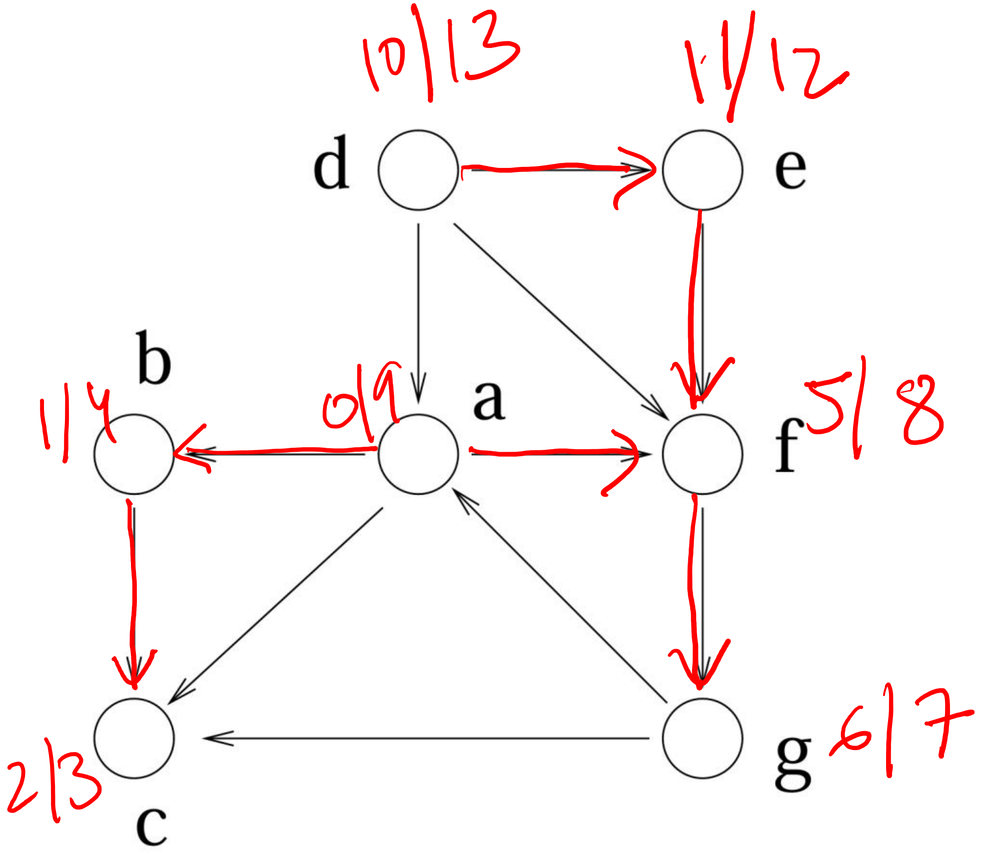
(d, c) – C

(d, f) – C

(d, S) – B

An edge $(u, v) \in E$ is called **back edge**, if v is an ancestor of u in DFS tree.
An edge $(u, v) \in E$ is called **forward edge**, if v is decentend of u in DFS tree.
An edge $(u, v) \in E$ is called **cross edge**, if v is neither ancestor nor a decentend of u in DFS tree.

Example



(a, c) — F

(g, a) — B

(g, c) — C

(d f) — F

(d, a) — C

Properties

For a given directed graph G = (V, E), let DFS(G) returns a tree. For any two nodes u, v ∈ V.

- v is descented of u if and only if start(u) < start(v) < finish(v) < finish(u).
- u and v are not related to each other if finish(u) < start(v) or finish(v) < start(u)
- Forward edge is from a node of higher finish time to node of lower finish time.
- Back edge is from a node of lower finish time to node of higher finish time.

Questions

- What is the finish time relationship between cross edge?
- What is the relationship between u and v if start(u) < start(v) < finish(u) < finish(v)?

Outline

- DFS Revisit

- Applications of DFS

Questions

- How to compute if node v is reachable from node u?
- How can we identify the cycle in a digraph using these edges?

Claim

A digraph G = (V, E) is acyclic if and only if DFS(G) does not yield any back edge.

Proof:
(⇐) Suppose there is no back edge from DFS(G). It implies that all edges go from higher finish time node to node with a lower finish time. So there cannot be any cycle.
(⇒) Proof by contradiction – Let there is no cycle in G.
There is a back edge $(u, v) \in E$.

- So, v is an ancestor of u in the DFS tree.
- So, there is path from v to u in the DFS tree.

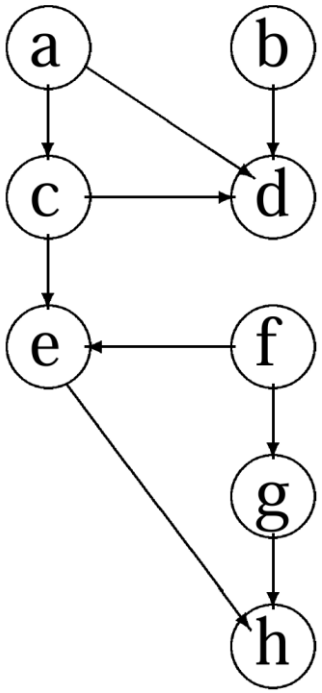
Path and back edge = cycle (Contradiction!)

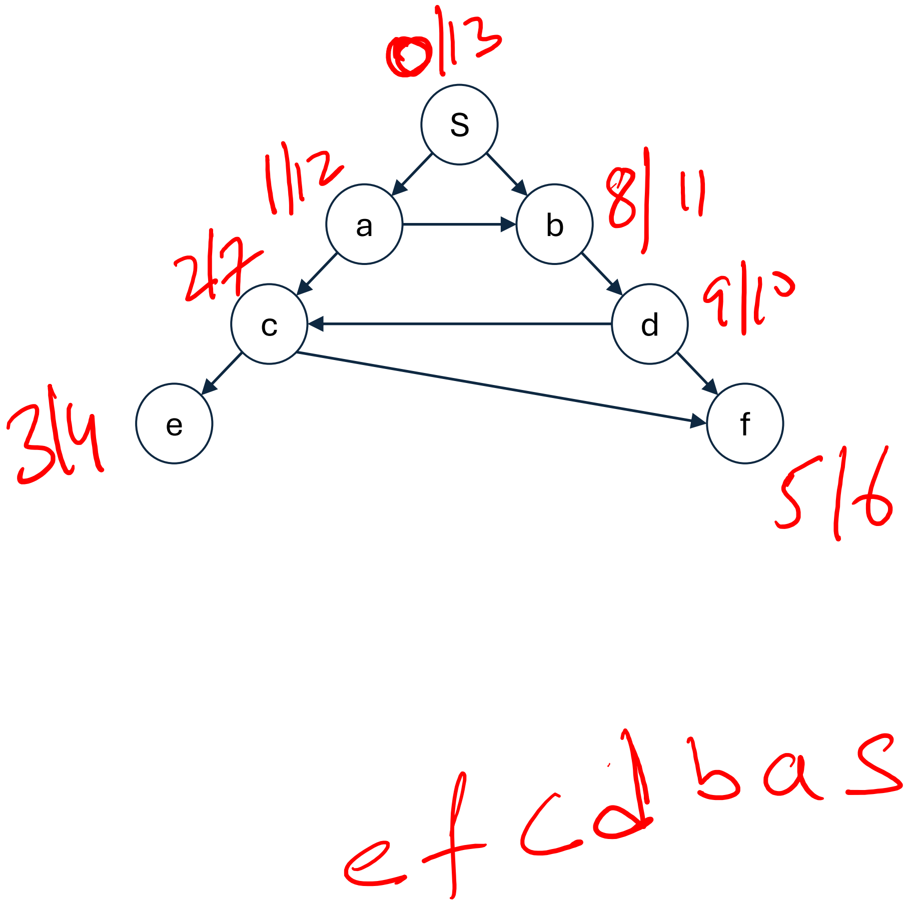
Topological Sorting

If G is DAG then there is an ordering of nodes V such that for each $(u, v) \in E$, u appears before v.

Topological Sort of a DAG is a total ordering $v_1 < v_2 < \ldots < v_n$ of vertices in V such that for any edge $(v_i, v_j) \in E$, $j > i$.

Example





<f, g, b, a, c, e, h, d>

<s, a, b, d, c, f, e>

Algorithm

Input: G = (V, E)
Output: Topological sort of V

TopSort(G):

1. Run DFS(G)
2. When computing the finish time for every vertex v, insert it at the front of a list
3. Return the list.

Running time?

Longest Path Weight

Input: Edge weighted directed graph G = (V, E, l). $l : E \to R^+$
Goal: Total weight of a longest path from v to t.

Let, LLP(v) computes the longest path in G from node v to node t.

If v = t then,
LLP(v) = 0
Else
max{l(v,w) + LLP(w) | (v, w) ∈ E}

Longest Path Weight

Input: Edge weighted directed graph G = (V, E, l).
Goal: Total weight of a longest path from v to t.

$$LLP(v) = \begin{cases} 0 & \text{if } v = t, \\ \max\left\{\ell(v \to w) + LLP(w) \;\middle|\; v \to w \in E\right\} & \text{otherwise,} \end{cases}$$

```
LongestPath(v, t):
  if v = t
      return 0
  if v.LLP is undefined
      v.LLP ← −∞
      for each edge v→w
          v.LLP ← max{v.LLP, ℓ(v→w) + LongestPath(w, t)}
      return v.LLP
```

Is there a possible DP?

Dynamic Programming

Compute postorder of G.
Subproblem: w.LLP: Computes the longest path in G from node w (appears after v) to node t.
Recurrence: If v = t then, v.LLP = 0
Else max{l(v,w) + w.LLP | w appears after v in the post order}

LONGESTPATH(s,t):
  for each node v in postorder
    if $v = t$
      $v.LLP \leftarrow 0$
    else
      $v.LLP \leftarrow -\infty$
      for each edge $v \rightarrow w$
        $v.LLP \leftarrow \max\left\{v.LLP,\ \ell(v \rightarrow w) + w.LLP\right\}$
  return $s.LLP$

Reference

Slides

Jeff Erickson Chp-6.1 & 6.3