Analysis of Algorithms

Graph Algorithms

More Definitions

Tree: **A subgraph is a tree if it is connected and removal of any one edge disconnects some pairs of vertices, i.e it is minimal connected graph.**

Forest: **A set of disjoint trees is called a forest.**

Breadth-First Tree

- For a graph $G = (V, E)$ with source s, the *predecessor subgraph* of G is $G_p = (V_p , E_p)$ where
  - $V_p = \{v \in V : \pi[v]\ ^{1}\ NIL\} \cup \{s\}$
  - $E_p = \{(\pi[v],v) \in E : v \in V_p - \{s\}\}$
- The predecessor subgraph G_p is a *breadth-first tree* if:
  - V_p consists of the vertices reachable from s and
  - for all $v \in V_p$, there is a unique simple path from s to v in G_p that is also a shortest path from s to v in G.
- The edges in E_p are called *tree edges*.
  $|E_p| = |V_p| - 1$

Intuition: Breadth-First Tree

- The predecessor pointers of the BFS define an inverted tree (an acyclic directed graph in which the source is the root, and every other node has a unique path to the root). If we make these edges bidirectional we get a rooted unordered tree called a BFS tree for G.

- There are potentially many BFS trees for a given graph, depending on where the search starts. These edges of G are called tree edges and the remaining edges of G are called cross edges.

Shortest Paths

- *Shortest-Path distance* $d(s, v)$ from s to v is the minimum number of edges in any path from vertex s to vertex v, or else ∞ if there is no path from s to v.

- A path of length $d(s, v)$ from s to v is said to be a *shortest path* from s to v.

Lemmas

- Let $G = (V, E)$ be a directed or undirected graph, and let $s \in V$ be an arbitrary vertex. Then, for any edge $(u,v) \in E$, $d(s, v) \le d(s, u) + 1$.
- Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on G from a given source vertex $s \in V$. Then upon termination, for each vertex $v \in V$, the value $d[v]$ computed by BFS satisfies $d[v] \ge d(s, v)$.
- Suppose that during the execution of BFS on a graph G, the queue Q contains vertices $(v_1, \ldots, v_r)$, where v_1 is the head of Q and v_r is the tail. Then, $d[v_r] \le d[v_1] + 1$ and $d[v_i] \le d[v_{i+1}]$ for $i = 1, 2, \ldots, r-1$.

Depth-First-Search (DFS)

- Explore edges out of the most recently discovered vertex v
- When all edges of v have been explored, backtrack to explore edges leaving the vertex from which v was discovered (its *predecessor*)
- "Search as deep as possible first"
- Whenever a vertex v is discovered during a scan of the adjacency list of an already discovered vertex u, DFS records this event by setting predecessor $\pi[v]$ to u.

Depth-First Trees

- Coloring scheme is the same as BFS. The predecessor subgraph of DFS is $G_p = (V, E_p)$ where $E_p = \{(\pi[v],v) : v \in V \text{ and } \pi[v]\ ^{1}\ NIL\}$. The predecessor subgraph G_p forms a *depth-first forest* composed of several *depth-first trees*. The edges in E_p are called *tree edges*.

- Each vertex u has 2 *timestamps*: $d[u]$ records when u is first discovered (grayed) and $f[u]$ records when the search finishes (blackens). For every vertex u, $d[u] < f[u]$.

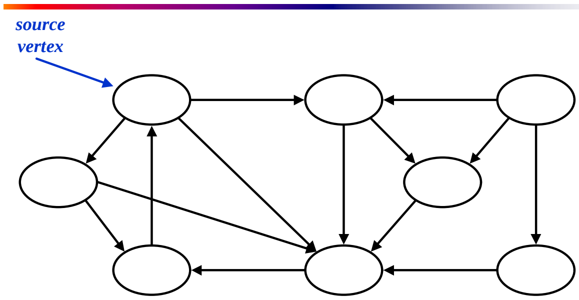
DFS(G)

1. for each vertex *u* Î *V*[*G*]
2. do *color*[*u*] ← WHITE
3. π[*u*] ← NIL
4. *time* ← 0
5. for each vertex *u* Î *V*[*G*]
6. do if *color*[*u*] = WHITE
7. then DFS-Visit(*u*)

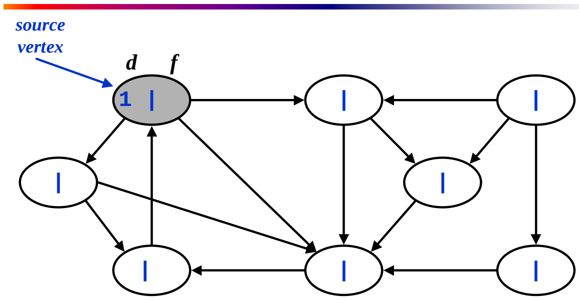
DFS-Visit(*u*)

1. *color*[*u*] ← GRAY
   ∇ White vertex *u* has been discovered
2. *d*[*u*] ← ++*time*
3. for each vertex *v* ∈ *Adj*[*u*]
4. do if *color*[*v*] = WHITE
5. then π[*v*] ← *u*
6. DFS-Visit(*v*)
7. *color*[*u*] ← BLACK
   ∇ Blacken *u*; it is finished.
8. *f*[*u*] ← ++ *time*

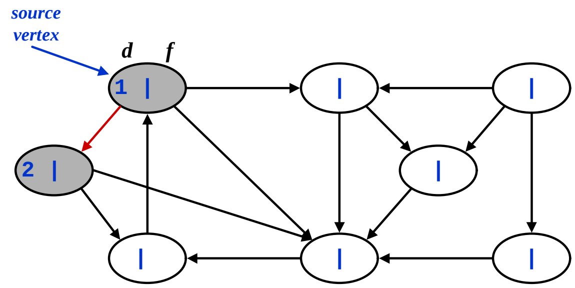
DFS Example



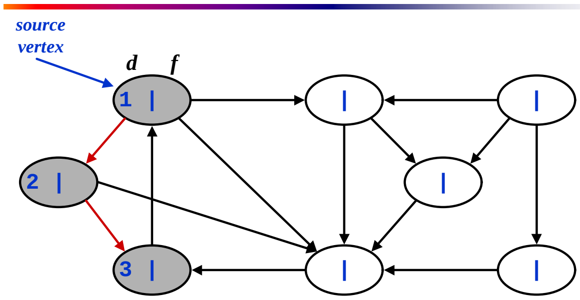
DFS Example



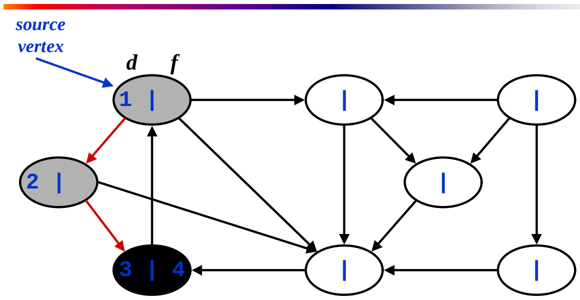
DFS Example



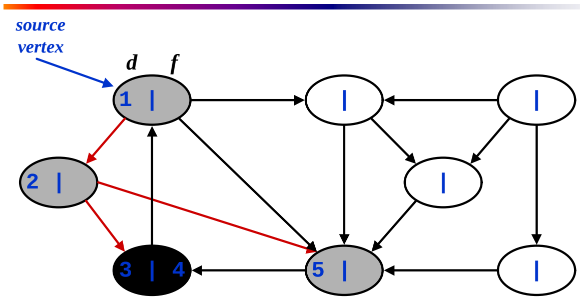
DFS Example



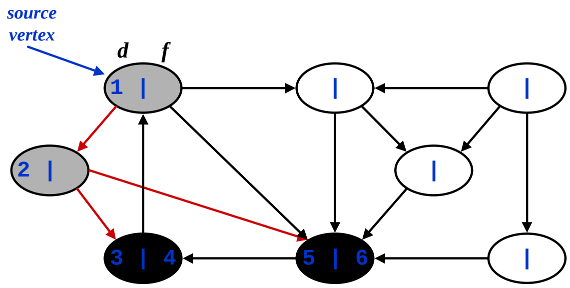
DFS Example



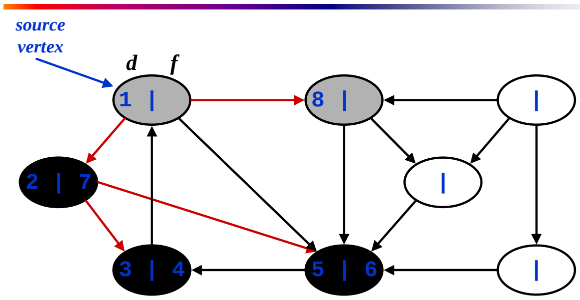
DFS Example



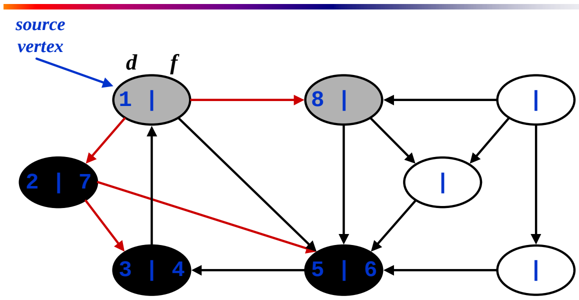
DFS Example



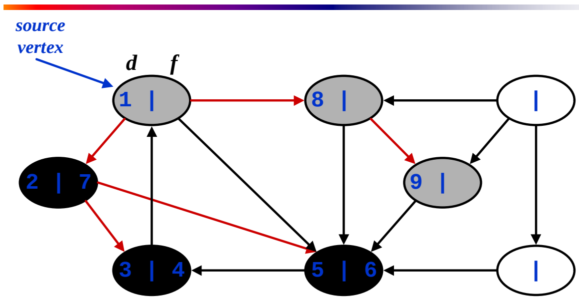
DFS Example



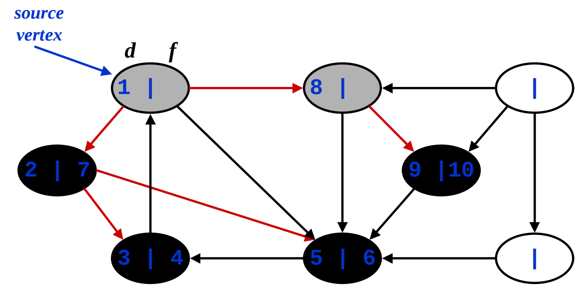
DFS Example



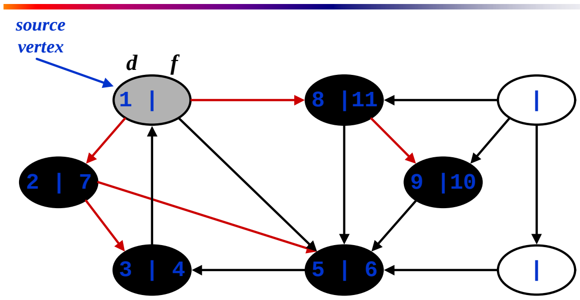
DFS Example



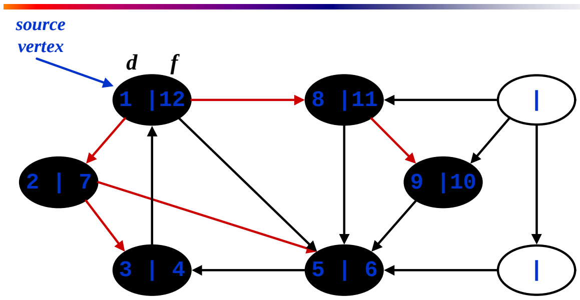
DFS Example



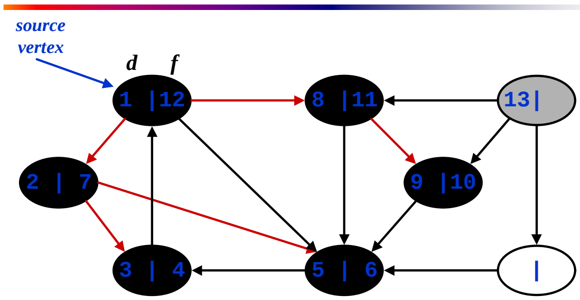
DFS Example



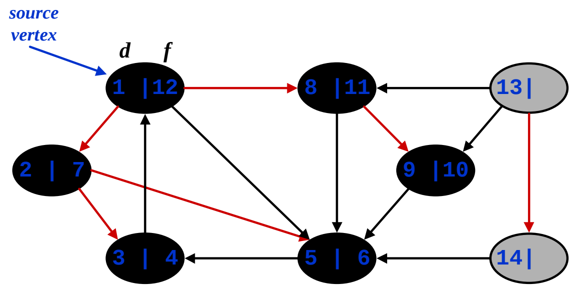
DFS Example



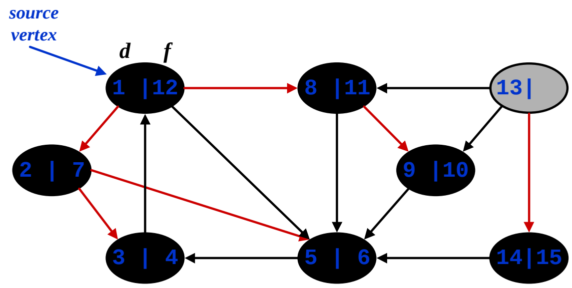
DFS Example



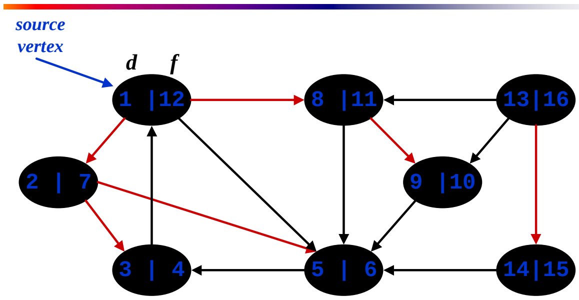
DFS Example



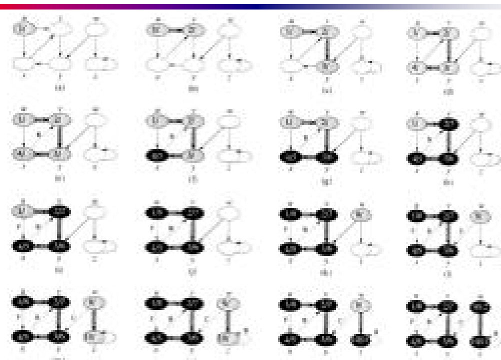
DFS Example



DFS Example



Operations of DFS



Analysis of DFS

- Loops on lines 1-2 & 5-7 take $\Theta(V)$ time, excluding time to execute DFS-Visit.
- DFS-Visit is called once for each white vertex $v \in V$ when it's painted gray the first time. Lines 3-6 of DFS-Visit is executed $|\text{Adj}[v]|$ times. The total cost of executing DFS-Visit is $\sum_{v \in V} |\text{Adj}[v]| = \Theta(E)$
- Total running time of DFS is $\Theta(V+E)$.

Properties of DFS

- Predecessor subgraph G_p forms a forest of trees (the structure of a depth-first tree mirrors the structure of DFS-Visit)
- The discovery and finishing time have *parenthesis structure*, i.e. the parentheses are properly nested. (See the figures and next theorem)

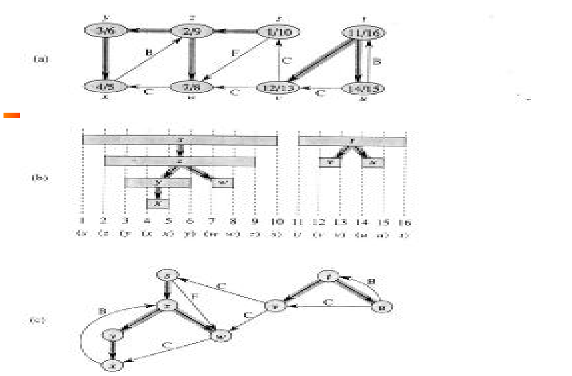
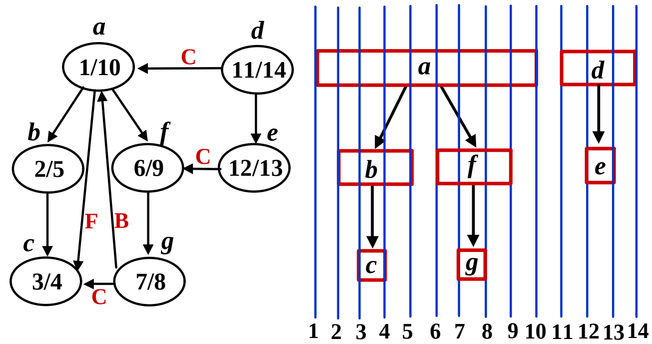


Figure 23.5 Properties of depth-first search. (a) The result of a depth-first search of a directed graph. Vertices are timestamped and edge types are indicated as in Figure 23.4. (b) Intervals for the discovery time and finishing time of each vertex correspond to the parenthesization shown. Each rectangle spans the interval given by the discovery and finishing times of the corresponding vertex. Tree edges are shown. If two intervals overlap, then one is nested within the other, and the vertex corresponding to the smaller interval is a descendant of the vertex corresponding to the larger. (c) The graph of part (a) redrawn with all tree and forward edges going down within a depth-first tree and all back edges going up from a descendant to an ancestor.

DFS & Parenthesis Lemma



Parenthesis Theorem

In any DFS of a graph $G = (V, E)$, for any two vertices u and v, exactly one of the followings holds:

- the interval $[d[u], f[u]]$ and $[d[v], f[v]]$ are entirely disjoint
- the interval $[d[u], f[u]]$ is contained entirely within the interval $[d[v], f[v]]$, and u is a descendant of v in the depth-first tree, or
- the interval $[d[v], f[v]]$ is contained entirely within the interval $[d[u], f[u]]$, and v is a descendant of u in the depth-first tree

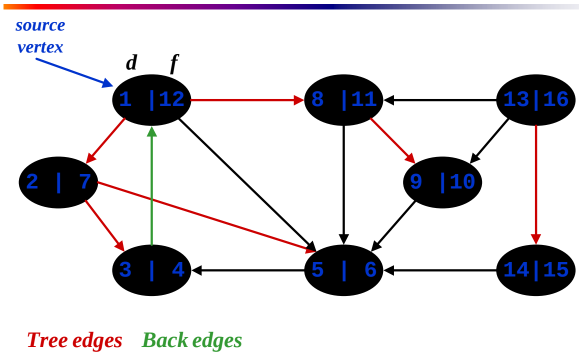
Nesting of Descendent' Intervals

- Vertex v is a proper descendant of vertex u in the depth-first forest for a (direct or undirected) graph G if and only if $d[u] < d[v] < f[v] < f[u]$

DFS: Kinds of edges

- DFS introduces an important distinction among edges in the original graph:
  - *Tree edge*: encounter new (white) vertex
  - *Back edge*: from descendent to ancestor
    - Encounter a grey vertex (grey to grey)

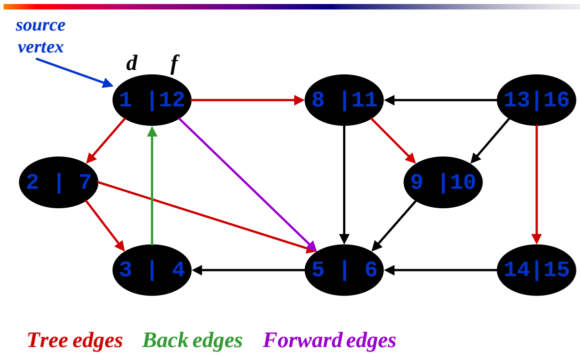
DFS Example



DFS: Kinds of edges

- DFS introduces an important distinction among edges in the original graph:
  - *Tree edge*: encounter new (white) vertex
  - *Back edge*: from descendent to ancestor
  - *Forward edge*: from ancestor to descendent
    - Not a tree edge, though
    - From grey node to black node

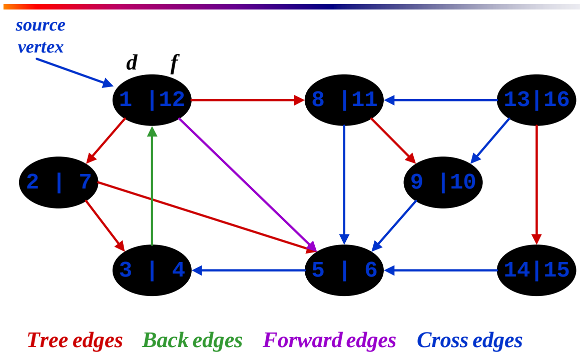
DFS Example



DFS: Kinds of edges

- DFS introduces an important distinction among edges in the original graph:
  - *Tree edge*: encounter new (white) vertex
  - *Back edge*: from descendent to ancestor
  - *Forward edge*: from ancestor to descendent
  - *Cross edge*: between a tree or subtrees
    - From a grey node to a black node

DFS Example

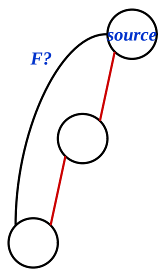


DFS: Kinds of edges

- DFS introduces an important distinction among edges in the original graph:
  - *Tree edge*: encounter new (white) vertex
  - *Back edge*: from descendent to ancestor
  - *Forward edge*: from ancestor to descendent
  - *Cross edge*: between a tree or subtrees
- Note: tree & back edges are important; most algorithms don't distinguish forward & cross

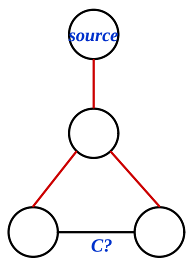
DFS: Kinds Of Edges

- Thm 23.9: If G is undirected, a DFS produces only tree and back edges
- Proof by contradiction:
  - Assume there's a forward edge
    - But F? edge must actually be a back edge (*why?*)



DFS: Kinds Of Edges

- Thm 23.9: If G is undirected, a DFS produces only tree and back edges
- Proof by contradiction:
  - Assume there's a cross edge
    - But C? edge cannot be cross:
    - must be explored from one of the vertices it connects, becoming a tree vertex, before other vertex is explored
    - So in fact the picture is wrong…both lower tree edges cannot in fact be tree edges



DFS And Graph Cycles

- Thm: An undirected graph is *acyclic* iff a DFS yields no back edges
  - If acyclic, no back edges (because a back edge implies a cycle
  - If no back edges, acyclic
    - No back edges implies only tree edges (*Why?*)
    - Only tree edges implies we have a tree or a forest
    - Which by definition is acyclic
- Thus, can run DFS to find whether a graph has a cycle