Math 182 Lecture 15

§ 8.3 Depth-first search

Recall: Breadth-first search (BFS) visits vertices in broad, shallow order.

Today: DFS (depth-first search).

DFS will search as deep as possible, then backtrack.

Like BFS, in DFS we will give all vertices a predecessor $v.\pi$. Unlike BFS, in DFS we will visit all vertices. Thus DFS tree will be a forest.

Define predecessor subgraph to be

$$G_\pi = (V, E_\pi) \text{ w/ } E_\pi = \{(v.\pi, v) : v \in V \text{ and } v.\pi \neq NIL\}$$

Predecessor subgraph forms depth-first forest comprised of depth-first trees. Edges in E_π called tree edges.

In DFS we will use colors White, Gray, Black

in the same way. Additionally, we will timestamp all the vertices with
v.d = discovery time
v.f = finishing time.

DFS(G) (main algorithm)

```
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)
```

lines 1–3: initialize all vertices.
line 4: initializes time (global variable)
lines 5–7: Will perform DFS-Visit on each undiscovered vertex until each vertex is discovered.

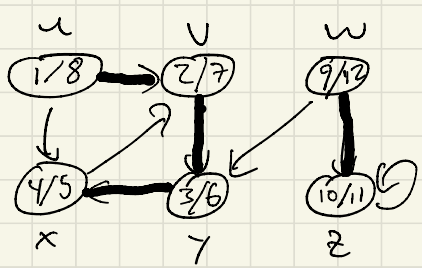
source vertex

DFS-VISIT(G, u) (algorithm which does the DFSing)

```
1  time = time + 1          // while vertex u has just been discovered
2  u.d = time               ← timestamp
3  u.color = GRAY           ← change color
4  for each v ∈ G.Adj[u]    ← considers all edges adjacent to u
5      if v.color == WHITE  ← discover white neighbors
6          v.π = u          ← set pred.
7          DFS-VISIT(G, v)  ← recursively call DFS on v.
8  u.color = BLACK          ← Now done w/ u, so set color and time stamp.
9  time = time + 1
10 u.f = time
```

Note: $u.d < u.f$ for all u.

DFS-visit(a)



Running Time

Every vertex gets processed once and every edge gets considered once or twice (depending on directed graph or not) so running time is $\Theta(V+E)$.

Properties of DFS

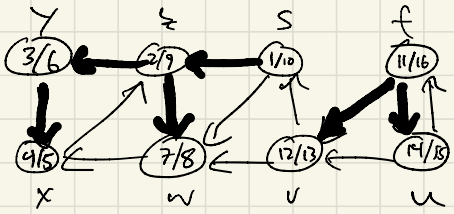
- G_π is indeed a forest (recall structure of G_π mirrors structure of recursive calls to DFS-Visit: $u = v.\pi$ iff DFS-Visit(G,v) called during search of u's neighbors.
- More generally, v descendent of u iff DFS-Visit(G,v) called while u gray.

Theorem 8.3.1 (Parenthesis theorem). *In any depth-first search of a (directed or undirected) graph $G = (V, E)$, for any two vertices u and v, exactly one of the following three conditions holds:*

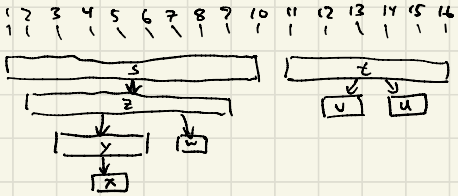
(1) the intervals $[u.d, u.f]$ and $[v.d, v.f]$ are entirely disjoint, and neither u nor v is a descendent of the other in the depth-first forest,

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

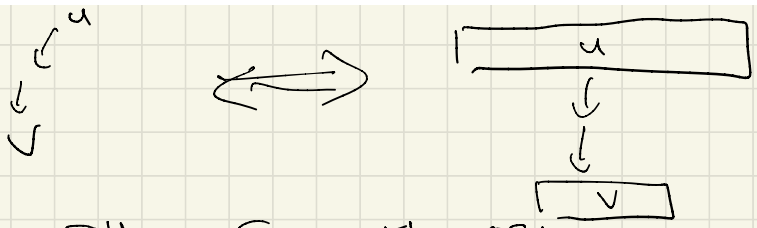
(3) the interval $[v.d, v.f]$ is contained entirely within the interval $[u.d, u.f]$, and v is a descendant of u in a depth-first tree.



Proof: See book/notes.



Corollary 8.3.2 (Nesting of descendants' intervals). *Vertex v is a proper descendant of a vertex u in the depth-first forest for a (directed or undirected) graph G if and only if $u.d < v.d < v.f < u.f$.*



Follows from Thm 8.3.1.

Another characterization of descendantcy:

Theorem 8.3.3 (White-path theorem). *In a depth-first forest of a (directed or undirected) graph $G = (V, E)$, vertex v is a descendant of vertex u if and only if at the time $u.d$ that the search discovers u, there is a path from u to v consisting entirely of white vertices.*

Proof ($\Rightarrow$) If $v = u$, then $v = u$ is white at $u.d$ and u is a path from u to v. O/w, sps v proper descendant of u. By Cor, $u.d < v.d$, so at $u.d$, v is white. This is true for all descendants of u, so true for all vertices on unique simple path from u to v on G_π.

($\Leftarrow$) Sps towards contradiction $\exists$ path of white vertices from u to v at time $u.d$, but v is not descendant of u in G_π. Can assume v is first non-descendant on path from u to v in G_π.

Let w be pred. of v in this path. Then w is descendant of u. Thus $w.f \leq u.f$ bcs of Cor. Furthermore, v must be discovered after u is discovered, but before w is finished (bcs $\exists$ edge from w to v) thus $u.d < v.d < w.f \leq u.f$.

By Parenthesis Thm, must have $v.f < u.f$. Thus v proper descendant of u. $\#$

Classification of edges

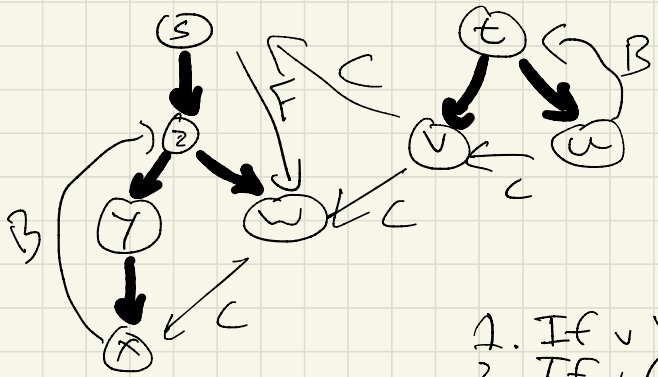
DFS can be used to classify the edges of $G = (V, E)$.

Four different types of edges:

(1) **Tree edges** are edges in the depth-first forest G_π. Edge (u, v) is a tree edge if v was discovered by exploring edge (u, v).
(2) **Back edges** are those edges (u, v) connecting a vertex u to an ancestor v in a depth-first tree. We consider self-loops, which may occur in directed graphs, to be back edges.
(3) **Forward edges** are those nontree edges (u, v) connecting a vertex u to a descendant v in a depth-first tree.
(4) **Cross edges** are all other edges. They can go between vertices in the same depth-first tree, as long as one vertex is not an ancestor of the other, or they can go between vertices in different depth-first trees.

Ex:

Note: Can determine



edge type while running DFS.
(u,v) has type one of the following depending on at time (u,v) is explored:

1. If v white, (u,v) tree edge.
2. If v Gray, then (u,v) back edge!
3. If v Black, then (u,v) cross or forward edge.

Theorem 8.3.4. *In a depth-first search of an undirected graph G, every edge of G is either a tree edge or a back edge.*

Proof: Sps (u,v) is an edge, and u.d < v.d. Then search must finish and discover v before it finishes u (while u is gray), since v is on u's adjacency list. Two cases: if first time we cross edge (u,v) is from u to v, then (u,v) is tree edge. If not, then it will be a back edge. ∎

§8.4 Minimum spanning trees

Setup: designing electronic circuit w/ pins want to connect w/ wires. Want to do this in a way that uses least amount of wire.

Consider undirected connected graph $G=(V,E)$

V = pins E = possible connections. For each $(u,v) \in E$, will have a cost $w(u,v) \in \mathbb{R}$.

Goal want to find spanning tree $T \subseteq E$ s.t. $w(T) = \sum_{(u,v)\in T} w(u,v)$ is minimized.

Spanning tree = (V,T) is connected.

Minimum-spanning-tree-problem

Idea: will use a greedy approach.

GENERIC-MST(G, w)

1 $A = \emptyset$
2 **while** A does not form a spanning tree
3 find an edge (u, v) that is safe for A
4 $A = A \cup \{(u, v)\}$
5 **return** A