Parma University, Math. Department

# Algorithms and Data Structures
# Depth First Search
# in directed graphs

Marco Pellegrini



## Summary

- Depth first search on Directed Graphs
- Depth first forest
- Edge classification
- topological sort
- material covered in CLR 23.3-4



## Depth first search (1)

- DFS is a method for visiting all nodes and edges in a graph.
- The main idea is to get as far from the source as possible and then back-track.
- At the end of DFS we have time-stamps associated with the nodes and an array p representing a spanning forest of the graph G.
- Color codes:
  white: Node not visited yet
  grey: Node seen once, but we might see it again.
  black: the visit of this node is completed.
- Timestamp d[v] = time when we see v for the first time.
  Timestamp f[v] = time when we see v for the last time.



## Code for DFS

- Input G=(V,E) in adj list representation.
- Output: arrays d[1..n], f[1..n], and p[1..n].
- Auxiliary data: color[1..n], time.
-
```
DFS(G)
  FOR each vertex u in V(G) DO
     color[u] := white
     p[u] := NIL;
  END FOR
  time := 0
  FOR each vertex u in V(G) DO
    IF color[u] = white
       THEN DFS-vist(u)
    END IF
  END FOR
```

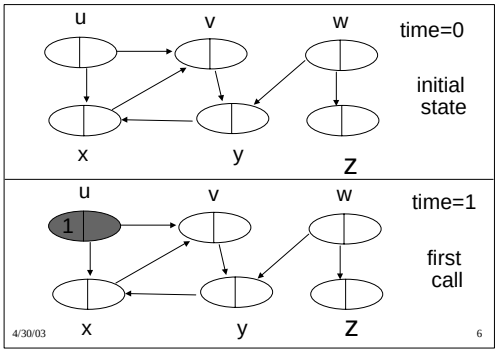


## DFS-visit

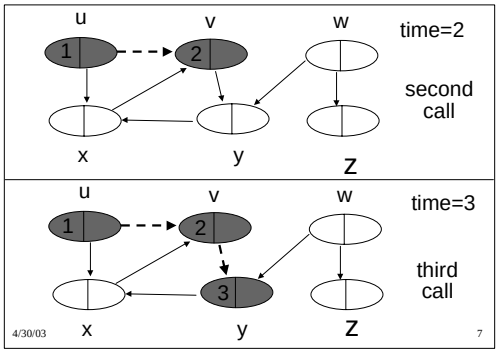
```
DFS-visit(u)
        color[u] := grey
        time := time + 1
        d[u] := time
        FOR each v in adiacency-list(u) DO
          IF color[v] = white
             THEN  p[v] := u
                   DFS-vist(v)
          END IF
        END FOR
        color[u] := black
        time := time +1
        f[u] := time
```



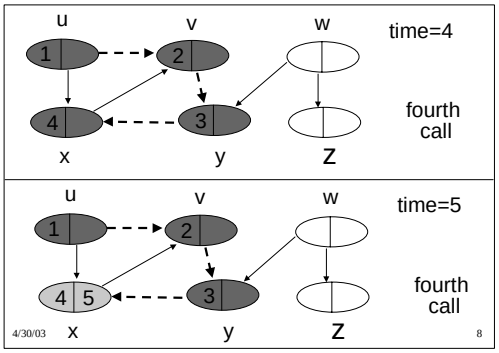
## Example (1)



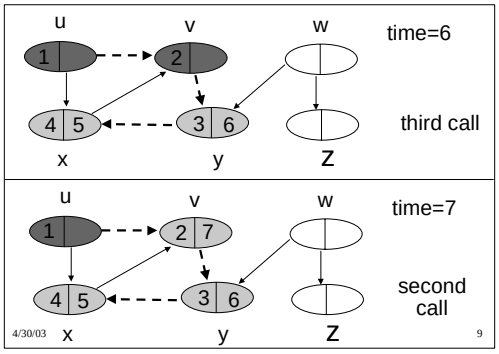
## Example (2)



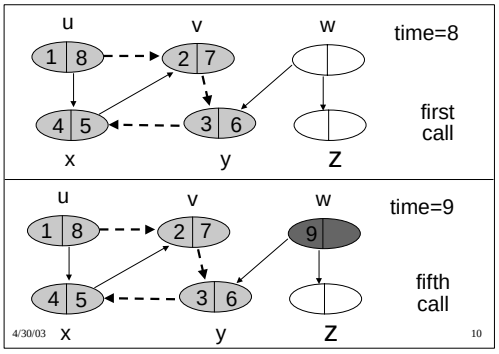

## Example (3)



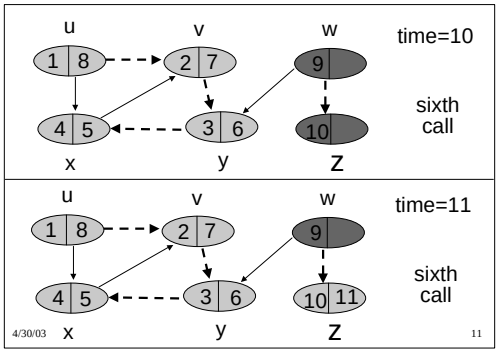

## Example (4)



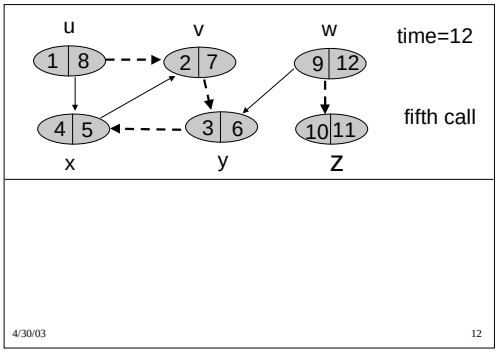

## Example (5)




## Example (6)




## Example (7)



# Exercise

- Simulate DFS on the graph of the previous slide supposing the order of the nodes is w, v, u, z, y, x when we execute the for loop on all the vertices or on adjacency lists. Show the time stamps d[] and f[] and the pointer p[] for each node.



# Observations:

- Time analysis. The initialization loop takes time O(n).
- We call DFS-visit on a node only if it is white. The first action of a call is to make a node grey.
- Since a node cannot become white again. DFS is called only once on each node.
- The for loop of DFS-visit scans the adjacency list of a node only once.
- The total time is therefore O(|V| + |E|).
- DFS-forest. The DFS forest records the order in which we visit the nodes for the first time.

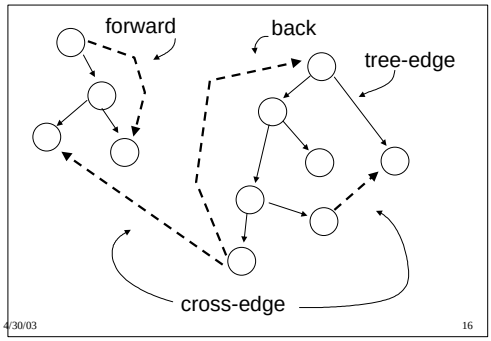


# Properties of time-stamps and DFS-forest

- Parenthesis theorem (CLR 23.6)
  Intervals A=[d[u],f[u]] and B=[d[v],f[v] are *disjoint* OR
  A is contained in B, in which case u is a *descendant* of v in a DFS-tree OR
  B is contained in A, in which case v is a *descendant* of u in a DFS-tree.
- Classification of edges.
  1) *Tree-edges*: edges belonging to a tree in the DFS-forest.
  2) *Back-edge*: edge from a note v to an ancestor of v in a tree of the DFS-forest
  3) *Forward-edge*: edge from a node v to a descendant of v in a tree of the DFS-forest
  4) *Cross-edge*: any other edge.



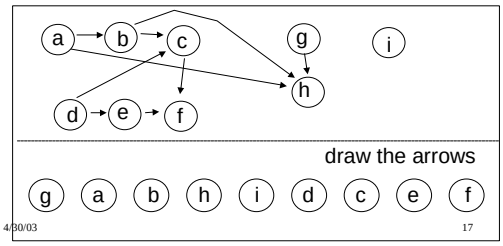
# Examples






# Topological sort

- Topological sort is the operation of taking an acyclic directed graph G and put all the nodes on a line so that the arrows go only to the right.






# Topological sort

- Toplogical sort is also the operation to turn a partial order into a total order.
- It is very useful for scheduling of tasks.
- Topological-sort(G)
  Run DFS(G),
  whenever f[u] is assigned also put u at the head of a list L.
  RETURN L
- When we assign the finish time f[u] at u we know DFS have already visited anything DFS could reach from u, so it is safe to put u in the output list L.
- What is the running time of Topological-sort?

# Topological sort, cntd.

- How do we know if G is acyclic?
- Lemma (CLR 23.10) G is acyclic if and only if DFS(G) does not detect back-edges.
- Now the idea to test whether a graph G is acyclic is to run DFS and when we find an edge from a grey node v to a grey node u, then this is a back-edge.
- So we can run Topological-sort on any graph provided we modify DFS to test the presence of back-edges. if we find one we stop and report that G is not a Directed Acyclic Graph.



# Homeworks

HW4.5 CLR 23.3-8 (singleton example):
Give an example of a node u in a directed graph G that has both incoming and exiting edges, but after the application of DFS is in a DFS-tree cointaining only u.

HW4.6 CLR 23.3-6 (descendant counter example)
Conjecture: If there is a path from u to v in a directed graph G and d[u]<d[v] in a DFS of G, then v is a descendant of u in a DFS-tree produced. Prove the Conjecture false with a counter-example.

HW4.7 Give an algorithm based on DFS that detects a cycle in a directed graph G.

HW4.8 Give an algorithm based on DFS that detects a cycle in an undirected graph G and works in time O(|V|).



# Conclusions

- DFS is a method of visiting each node and each edge of a graph once.
- DFS produces time stamps and a spanning forest of the graph G.
- We can detect using DFS if a directed graph is acyclic and in this case we can produce its topologically sorted sequence.