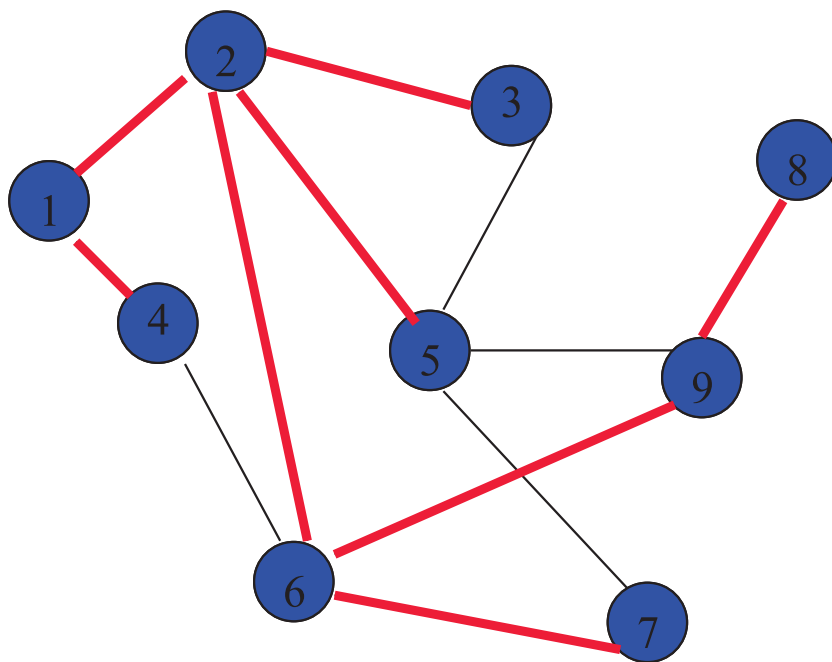


Spanning Tree



Breadth-first search from vertex **1**.

Breadth-first spanning tree.

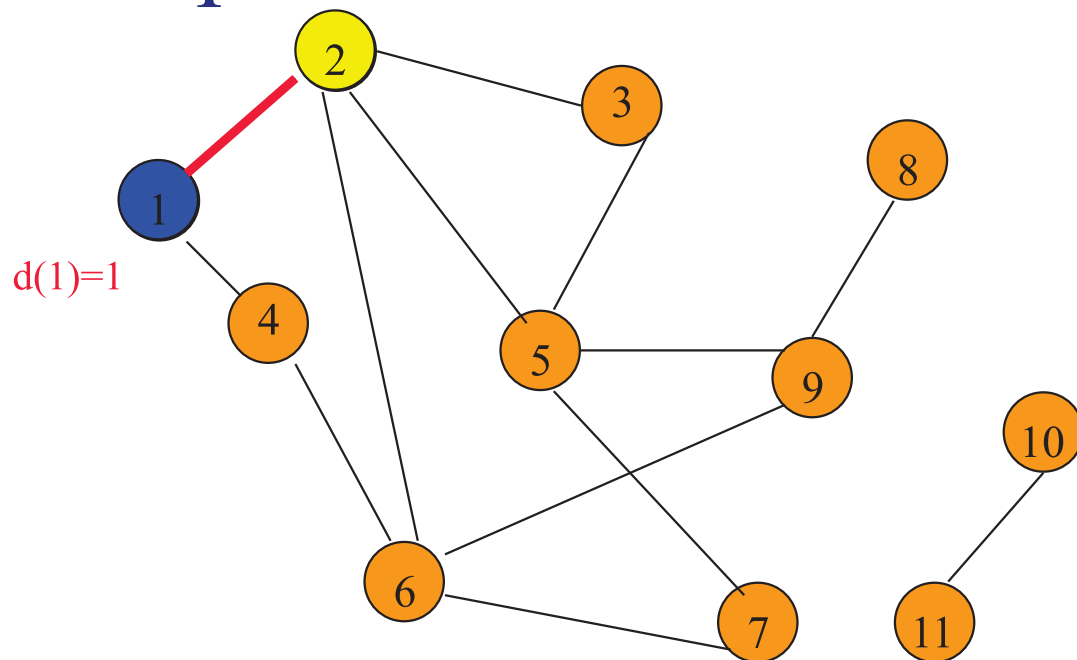
Spanning Tree

- Start a breadth-first search at any vertex of the graph.
- If graph is connected, the $n-1$ edges used to get to unvisited vertices define a spanning tree (breadth-first spanning tree).
- Time
 - $O(n^2)$ when adjacency matrix used
 - $O(n+e)$ when adjacency lists used (e is number of edges)

Depth-First Search

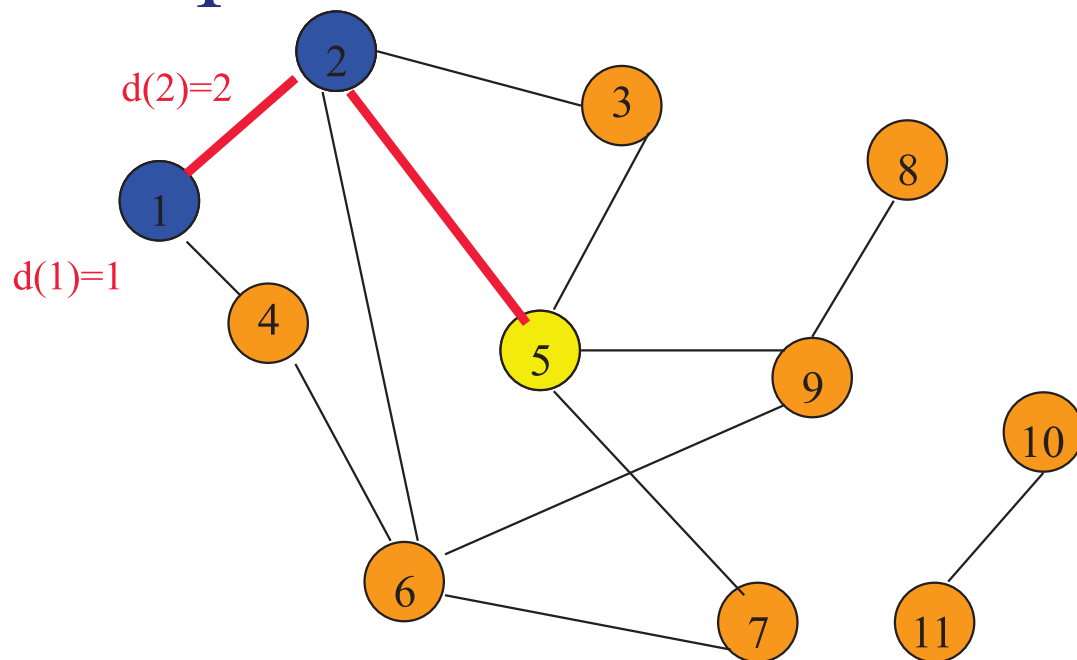
```
DFS(G)
1  for each vertex u ∈ V [G]
2      do color[u] ← WHITE
3      π[u] ← NIL
4  time ← 0
5  for each vertex u ∈ V [G]
6      do if color[u] = WHITE
7          then DFS-VISIT(u)
DFS-VISIT(u)
1  color[u] ← GRAY      ▷ White vertex u has just been discovered
2  time ← time + 1
3  d[u] ← time
4  for each v ∈ Adj[u]  ▷ Explore edge(u, v).
5      do if color[v] = WHITE
6          then π[v] ← u
7              DFS-VISIT(v)
8  color[u] ← BLACK      ▷ Blacken u; it is finished.
9  f [u] ▷ time ← time + 1
```

Depth-First Search Example



Start search at vertex 1.
Label vertex 1 and do a depth first search
from either 2 or 4.
Suppose that vertex 2 is selected.

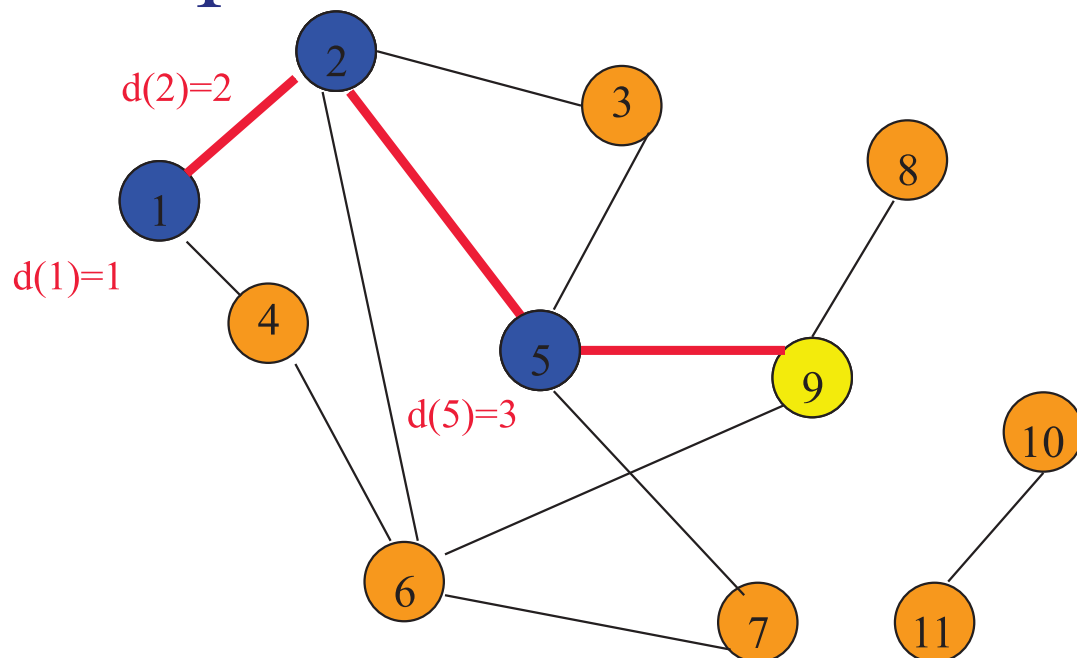
Depth-First Search Example



Label vertex **2** and do a depth first search from either **3**, **5**, or **6**.

Suppose that vertex **5** is selected.

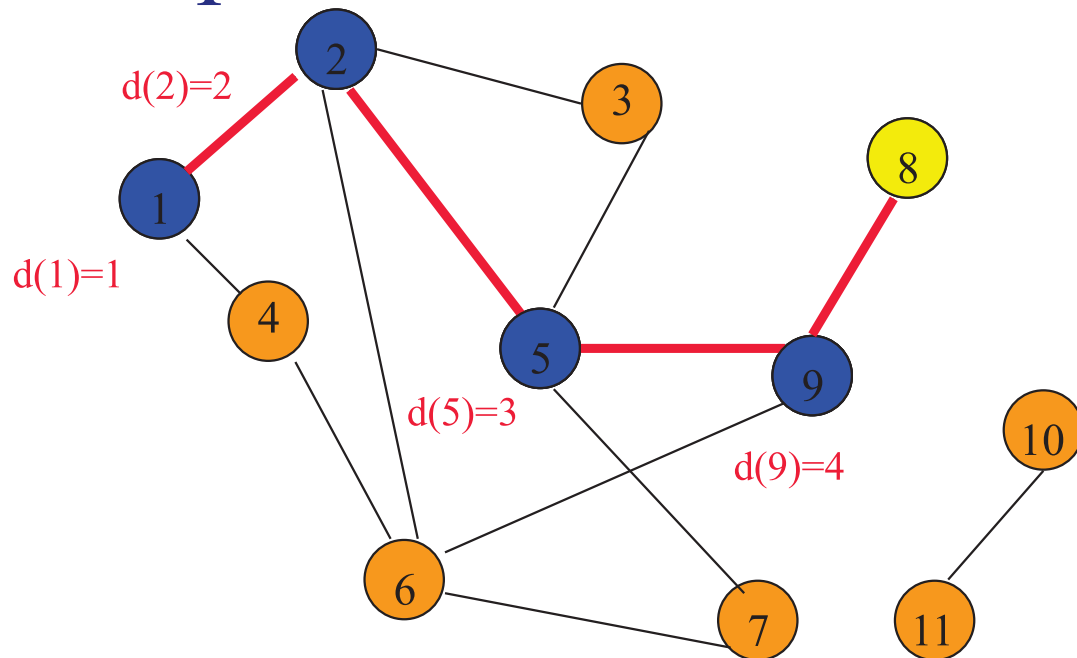
Depth-First Search Example



Label vertex **5** and do a depth first search from either **3**, **7**, or **9**.

Suppose that vertex **9** is selected.

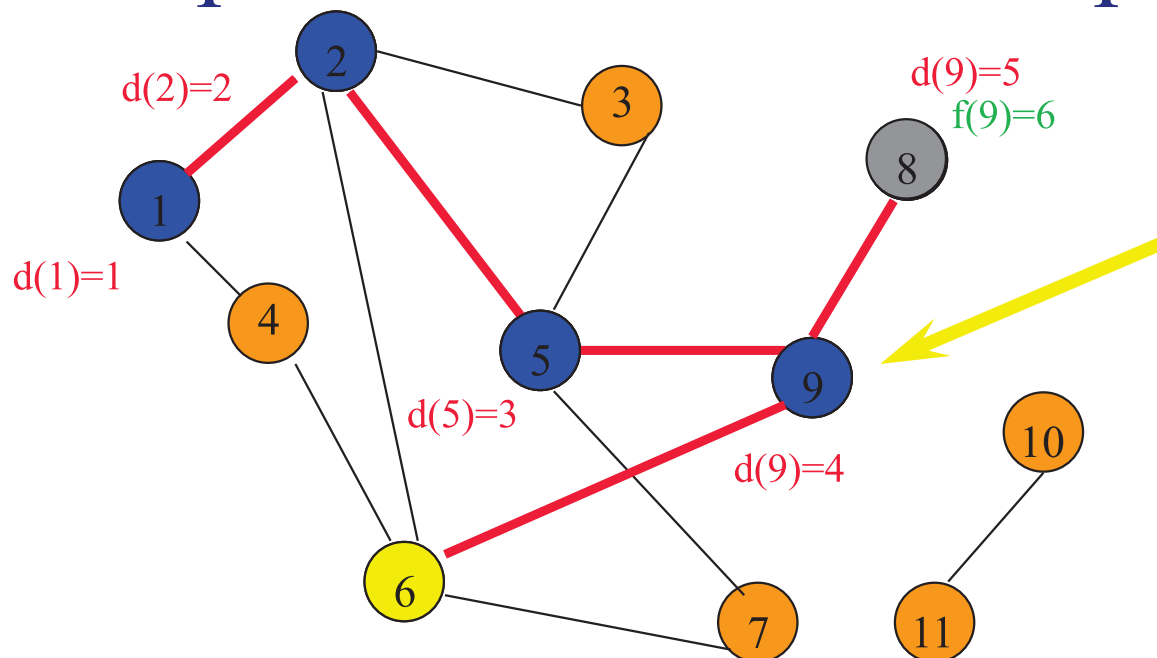
Depth-First Search Example



Label vertex **9** and do a depth first search from either **6** or **8**.

Suppose that vertex **8** is selected.

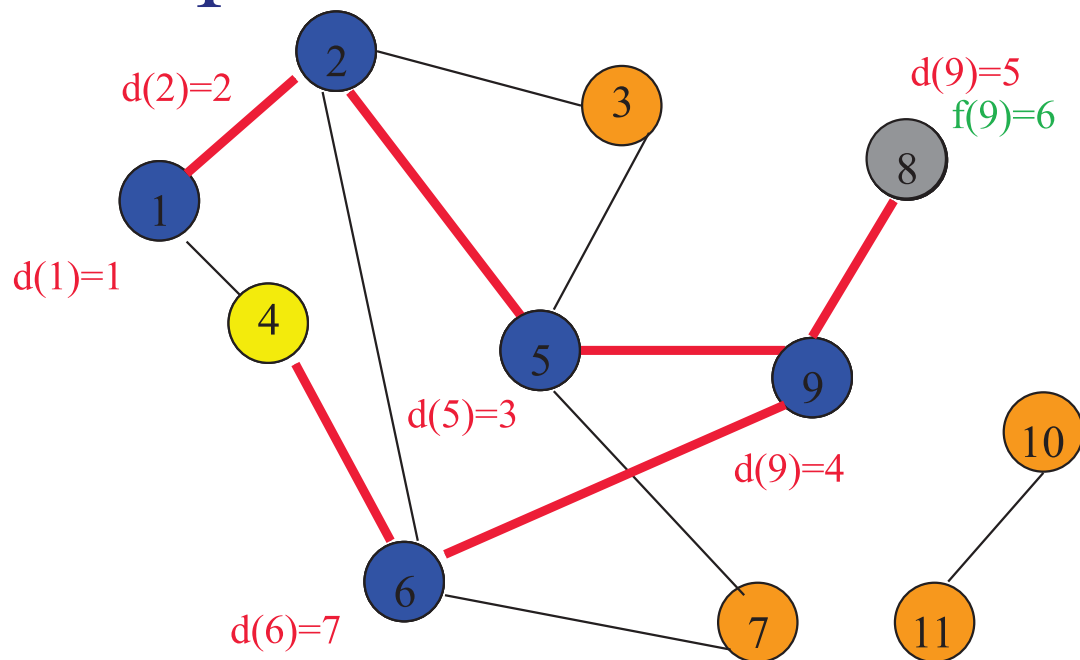
Depth-First Search Example



Label vertex **8** and return to vertex **9**.

From vertex **9** do a **dfs(6)**.

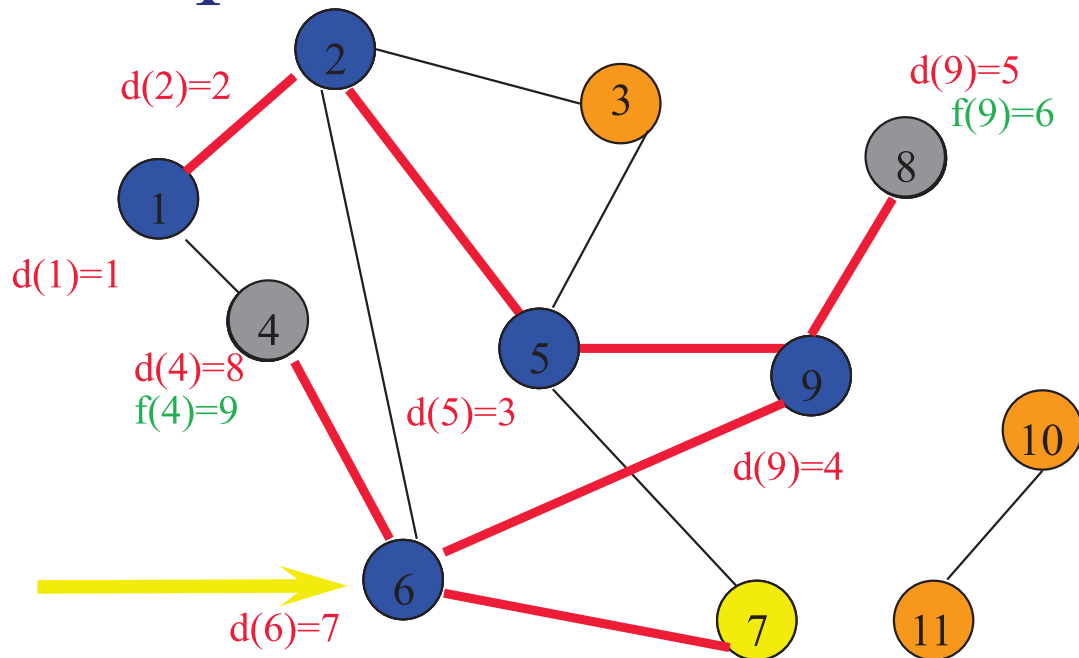
Depth-First Search Example



Label vertex **6** and do a depth first search from either **4** or **7**.

Suppose that vertex **4** is selected.

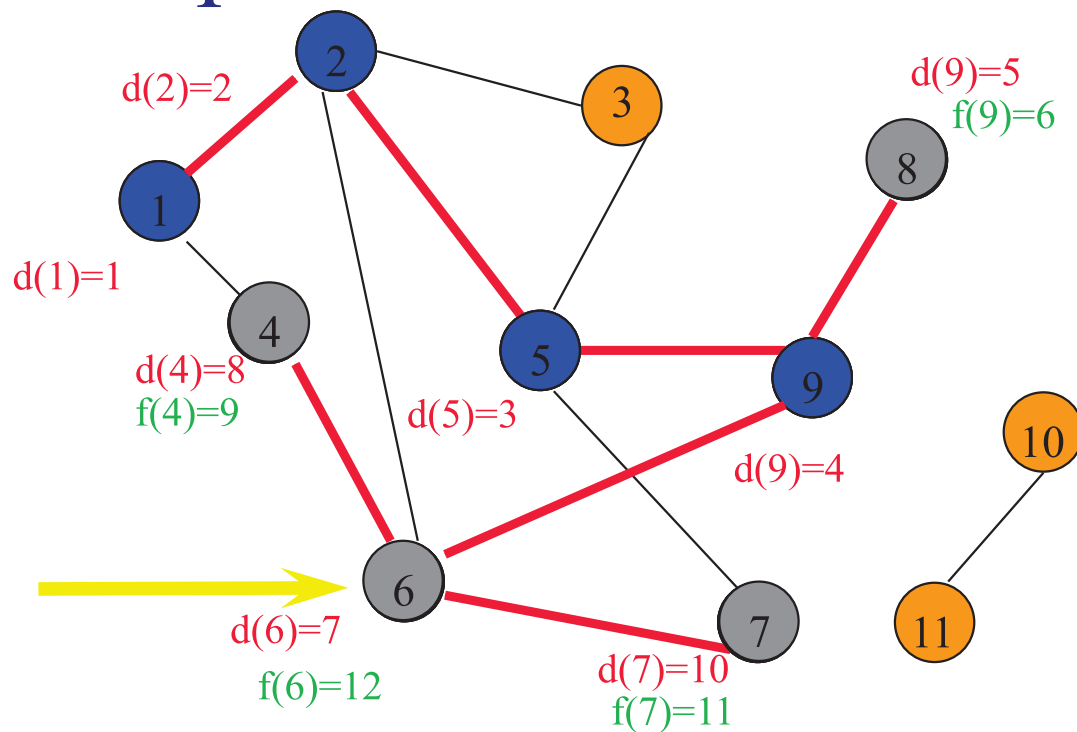
Depth-First Search Example



Label vertex **4** and return to **6**.

From vertex **6** do a **dfs(7)**.

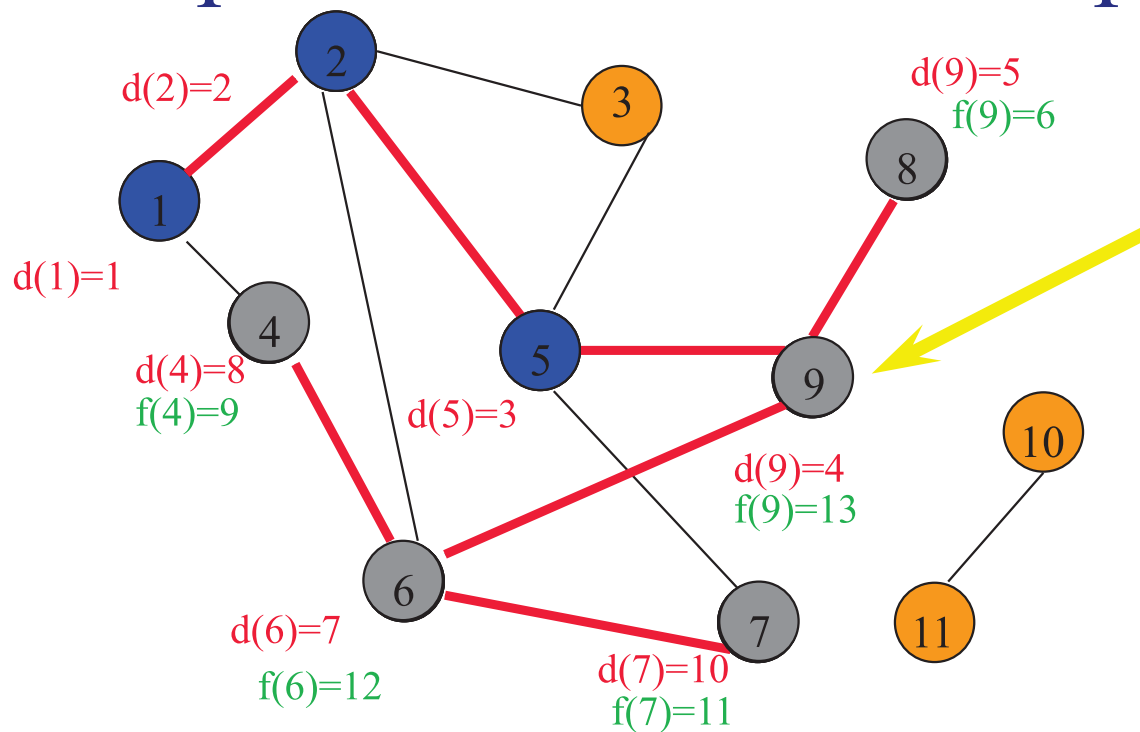
Depth-First Search Example



Label vertex **7** and return to **6**.

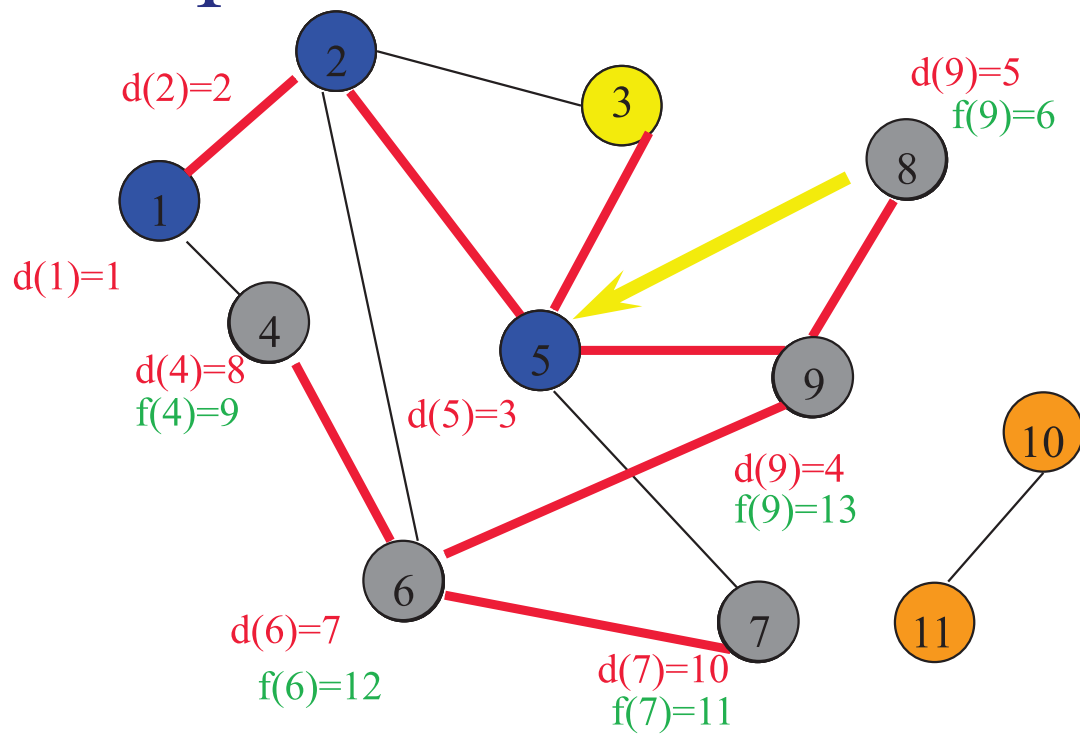
Return to **9**.

Depth-First Search Example



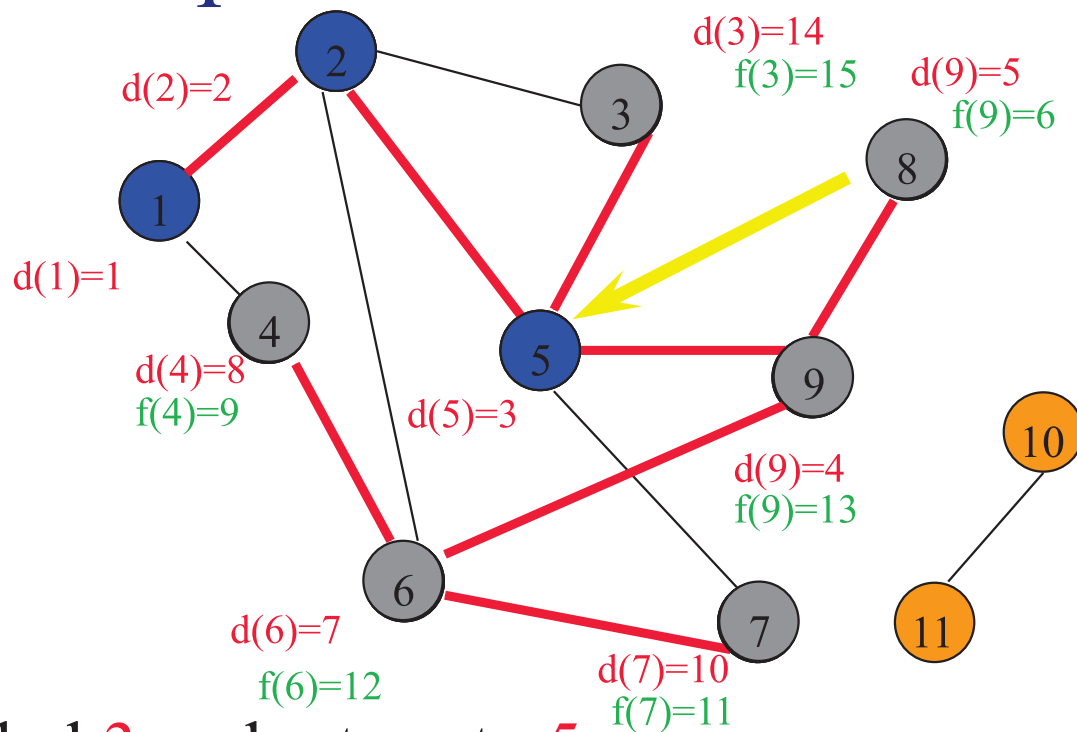
Return to **5**.

Depth-First Search Example



Do a $\text{dfs}(3)$.

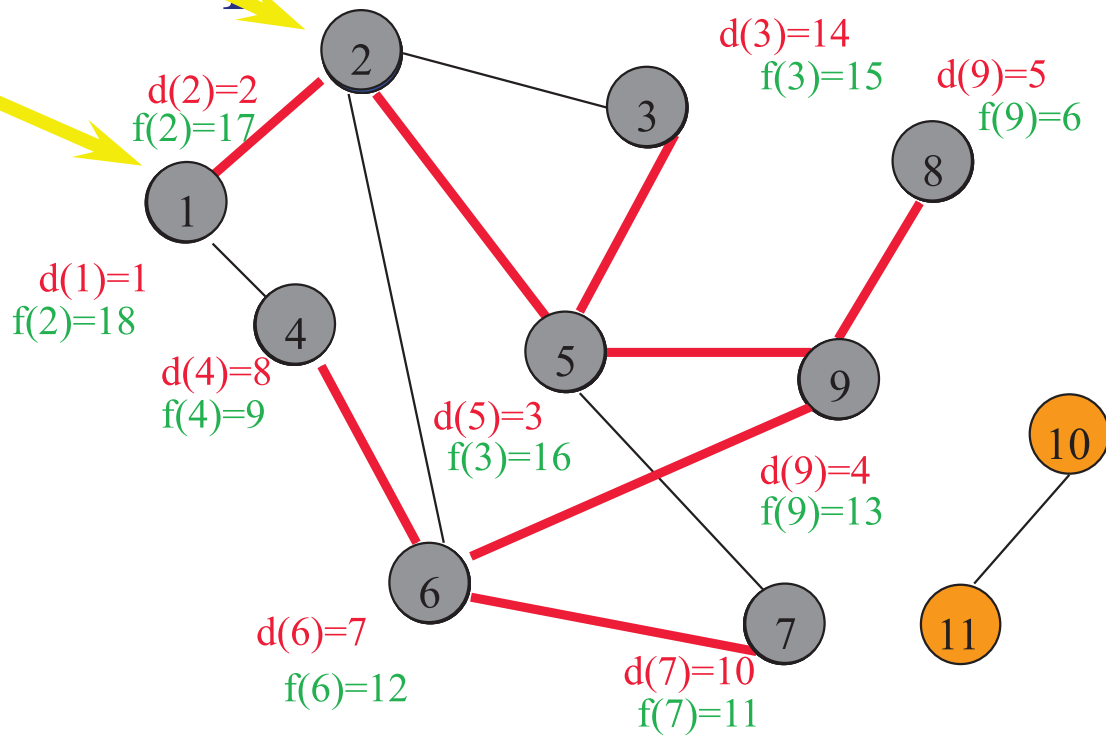
Depth-First Search Example



Label **3** and return to **5**.

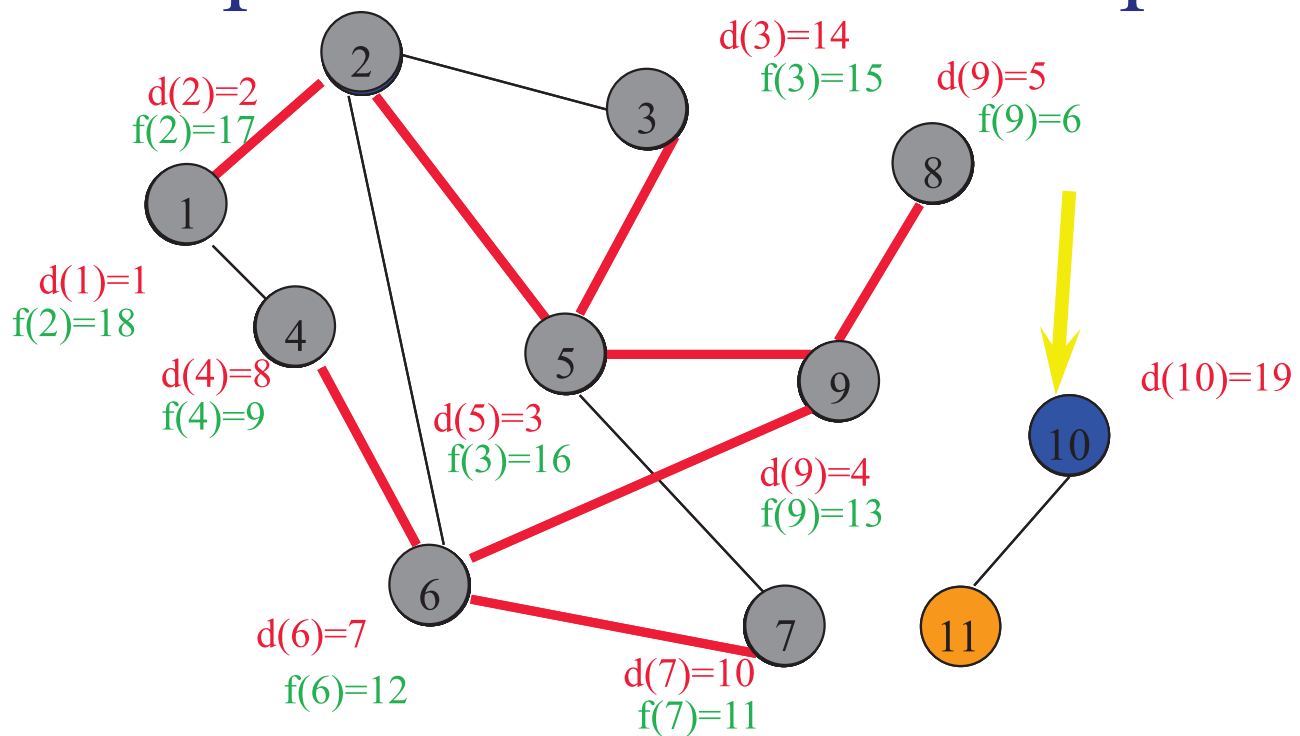
Return to **2**.

Depth-First Search Example



Return to 1.

Depth-First Search Example



Return to invoking method.

Depth-First Search Properties

- Same complexity as BFS.
- Same properties with respect to path finding, connected components, and spanning trees.
- Edges used to reach unlabeled vertices define a depth-first spanning tree when the graph is connected.
- There are problems for which BFS is better than DFS and vice versa.

Depth-First Search Properties

Theorem : (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:

- the intervals $[d[u], f[u]]$ and $[d[v], f[v]]$ are entirely disjoint, and neither u nor v is a descendant of the other in the depth-first forest,
- the interval $[d[u], f[u]]$ is contained entirely within the interval $[d[v], f[v]]$, and u is a descendant of v in a 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 a depth-first tree.

Depth-First Search Properties

Corollary : (Nesting of Descendants' Intervals)

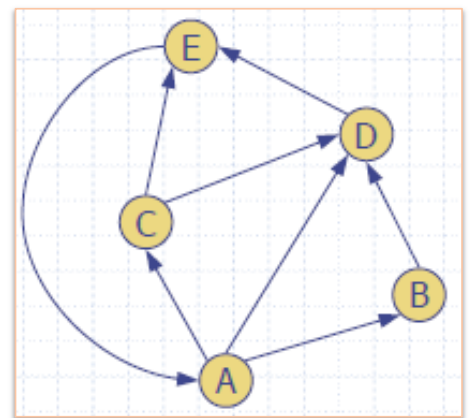
Vertex v is a proper descendant of vertex u in the depth-first forest for a (directed or undirected) graph G if and only if $d[u] < d[v] < f[v] < f[u]$.

Theorem : (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 $d[u]$ that the search discovers u , vertex v can be reached from u along a path consisting entirely of white vertices.

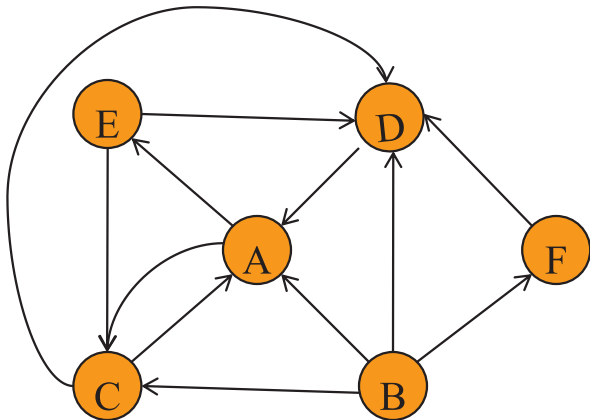
Directed Graphs (Digraphs)

- A **digraph** is a graph
 - whose edges are all directed.
- Short for “directed graph”
- Application
 - Scheduling: edge (A,B) means task **A** must be completed before **B** can be started.



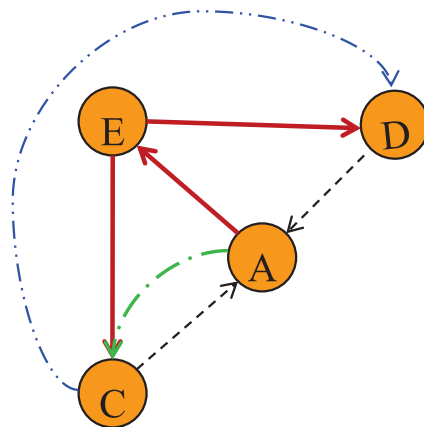
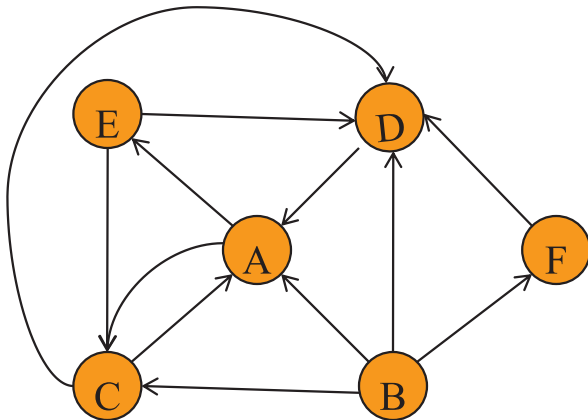
Example: DFS on Directed Graphs

- DFS starts at A, then B,...



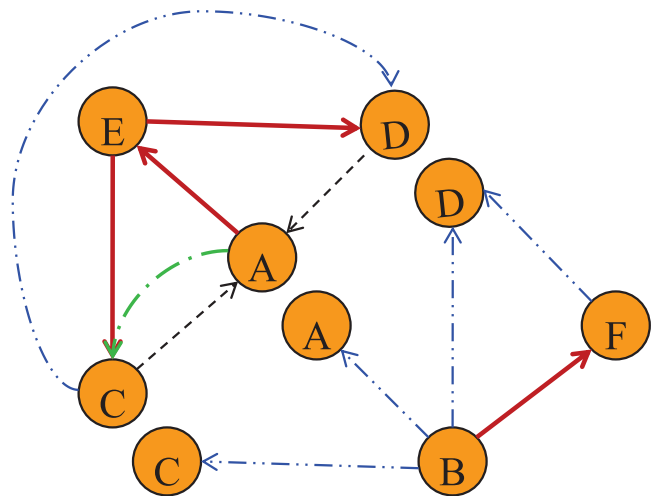
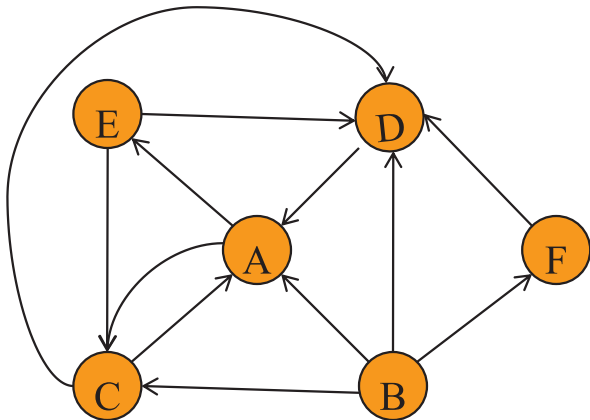
Example: DFS on Directed Graphs

- DFS starts at A, then B,...



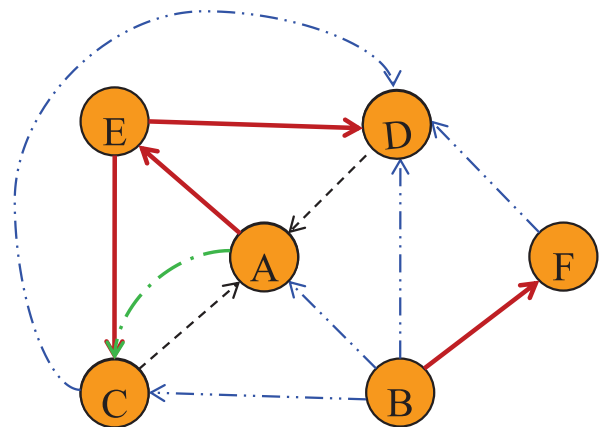
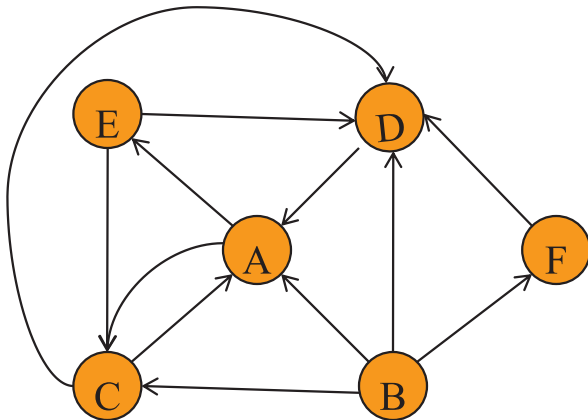
Example: DFS on Directed Graphs

- DFS starts at A, then B,...



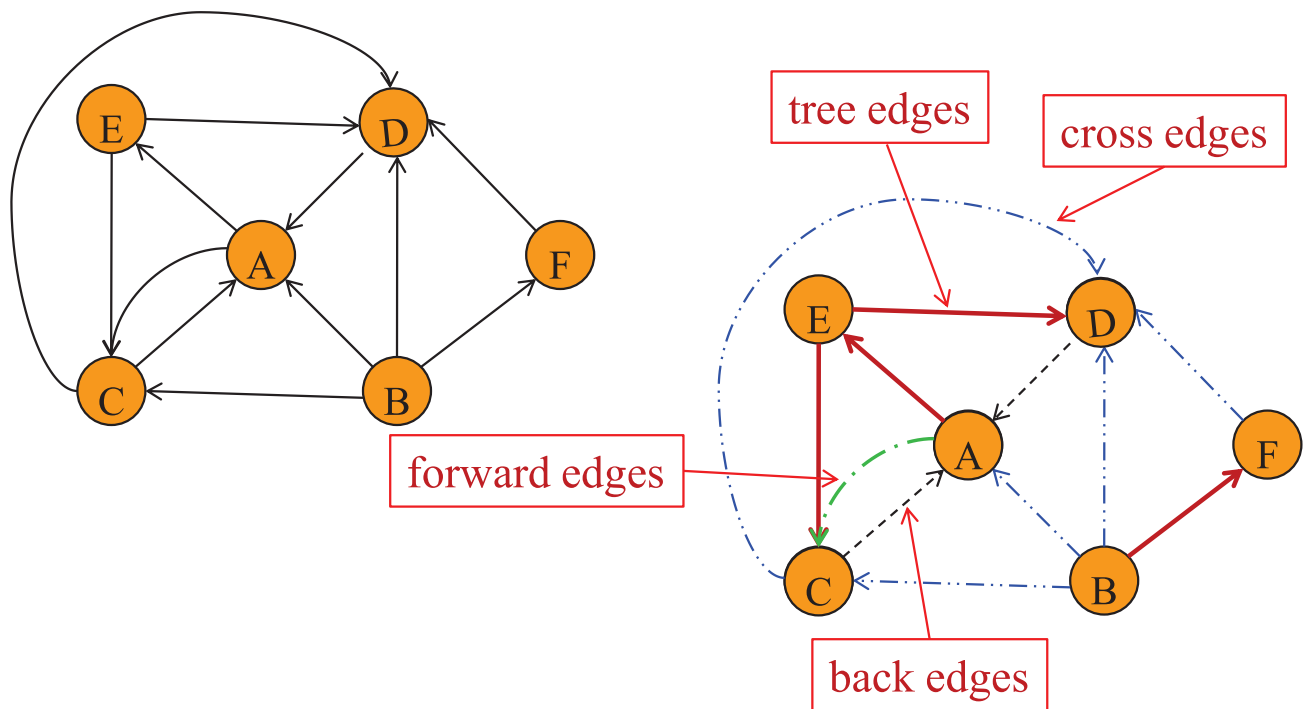
Example: DFS on Directed Graphs

- DFS starts at A, then B,...



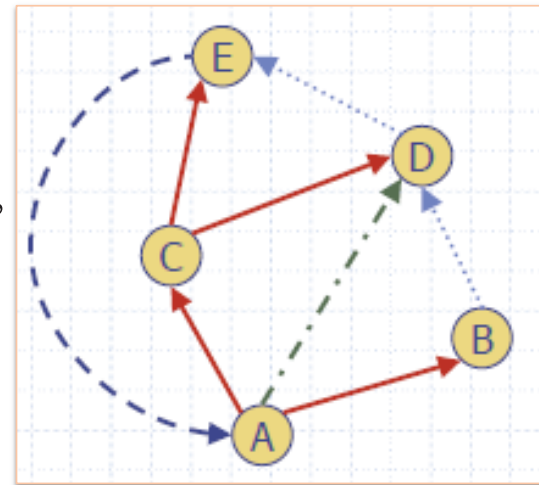
Example: DFS on Directed Graphs

- DFS starts at A, then B,...

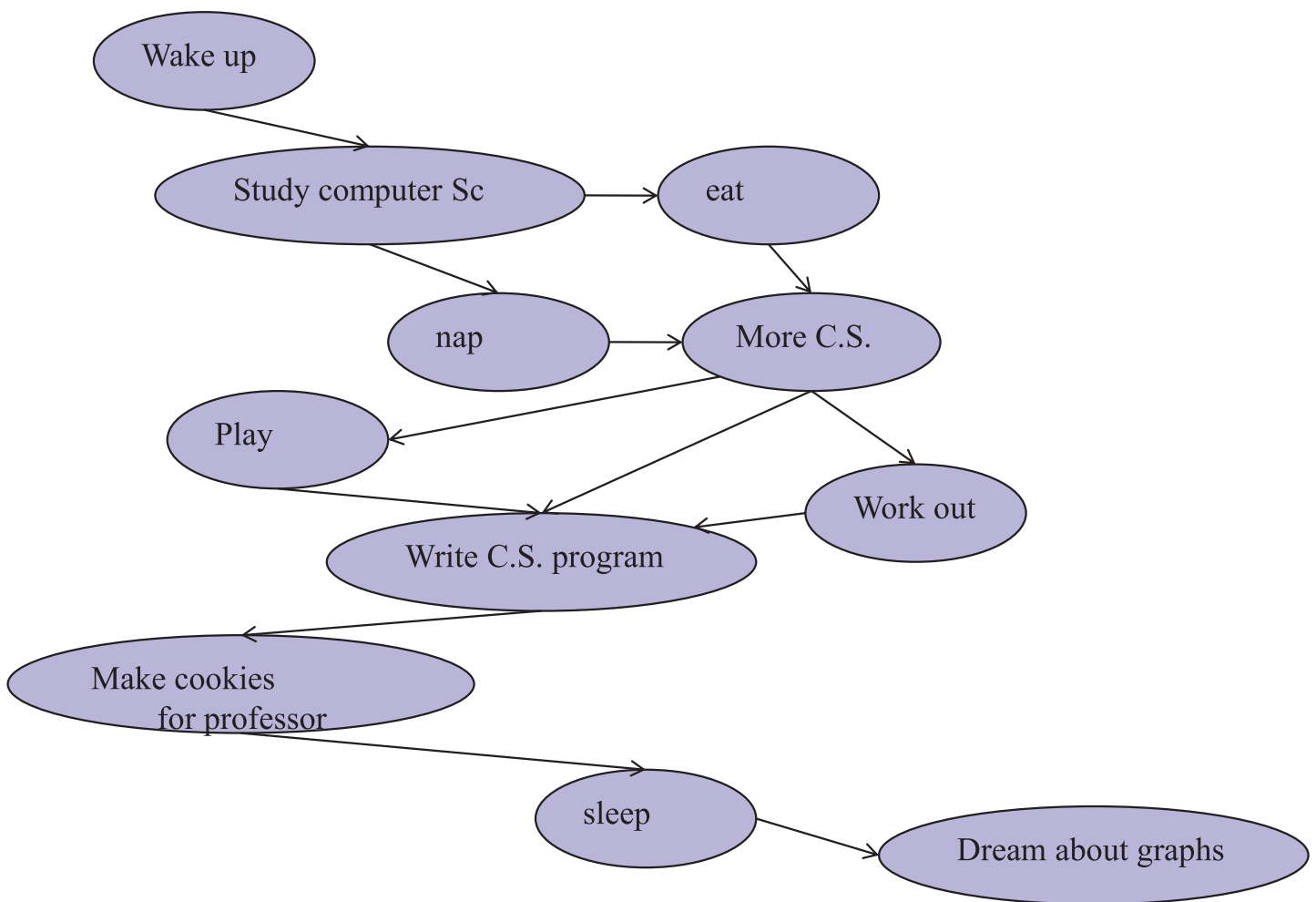


Classification of edges

- **Tree edges** are edges in the depth-first forest $G\pi$. Edge (u, v) is a tree edge if v was first discovered by exploring edge (u, v) .
- **Back edges** are those edges (u, v) connecting a vertex u to an ancestor v in a depthfirst tree. Self-loops, which may occur in directed graphs, are considered to be back edges.
- **Forward edges** are those nontree edges (u, v) connecting a vertex u to a descendant v in a depth-first tree.
- **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.

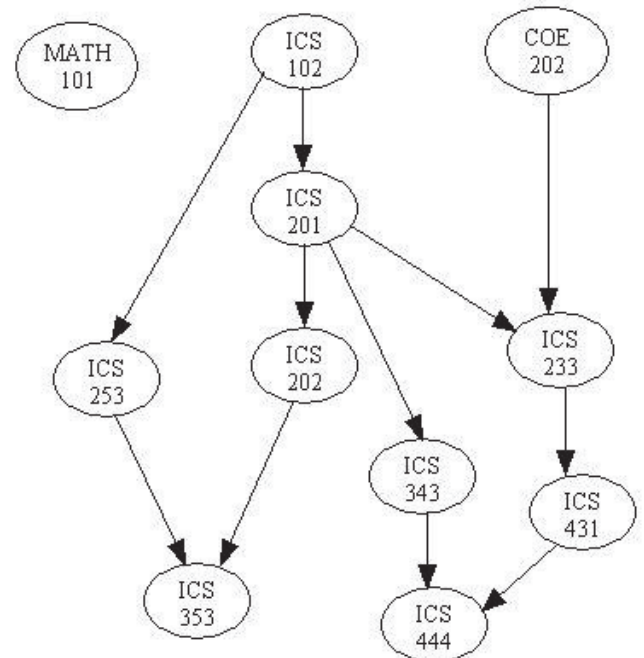


Example of digraph: A typical student day



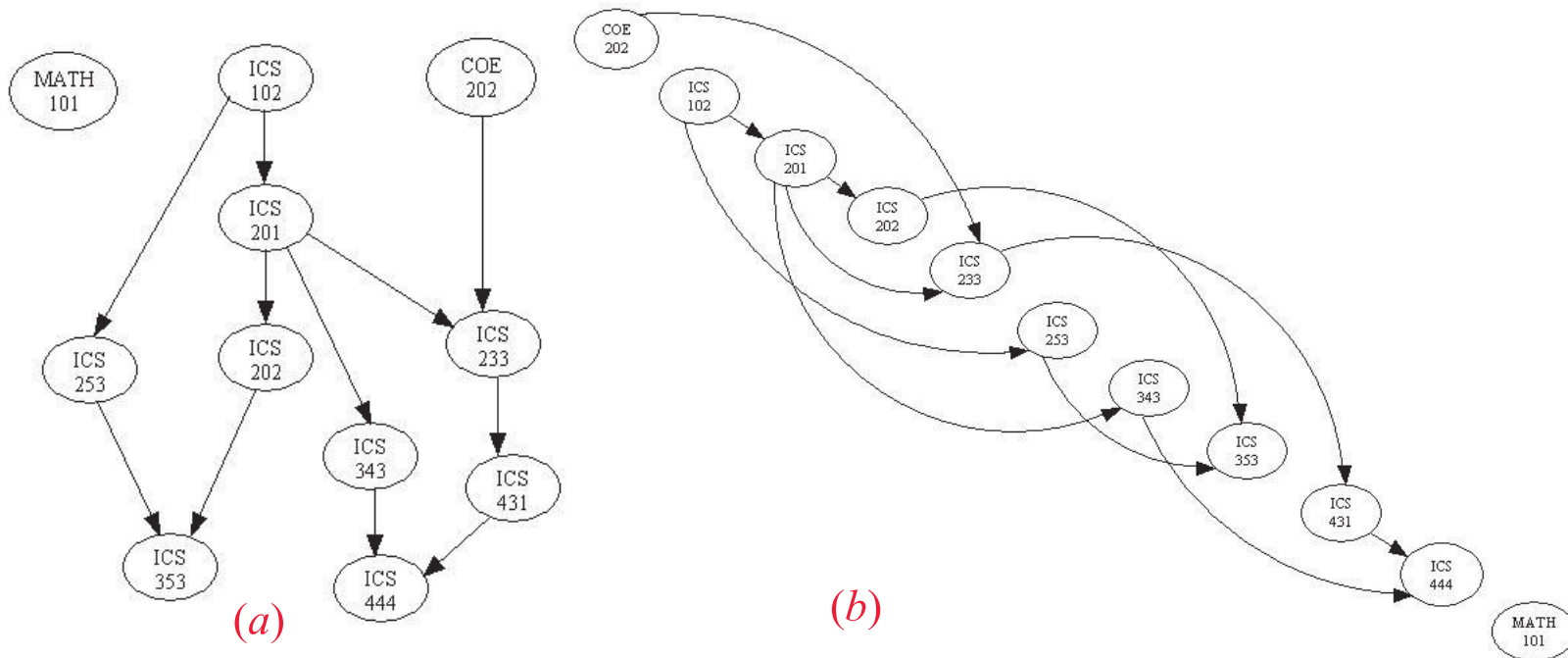
Linear Ordering

- There are many problems involving a set of tasks in which some of the tasks must be done before others.
- For example, consider the problem of taking a course only after taking its prerequisites.
- Is there any systematic way of linearly arranging the courses in the order that they should be taken ?
- Yes ! - Topological sort.



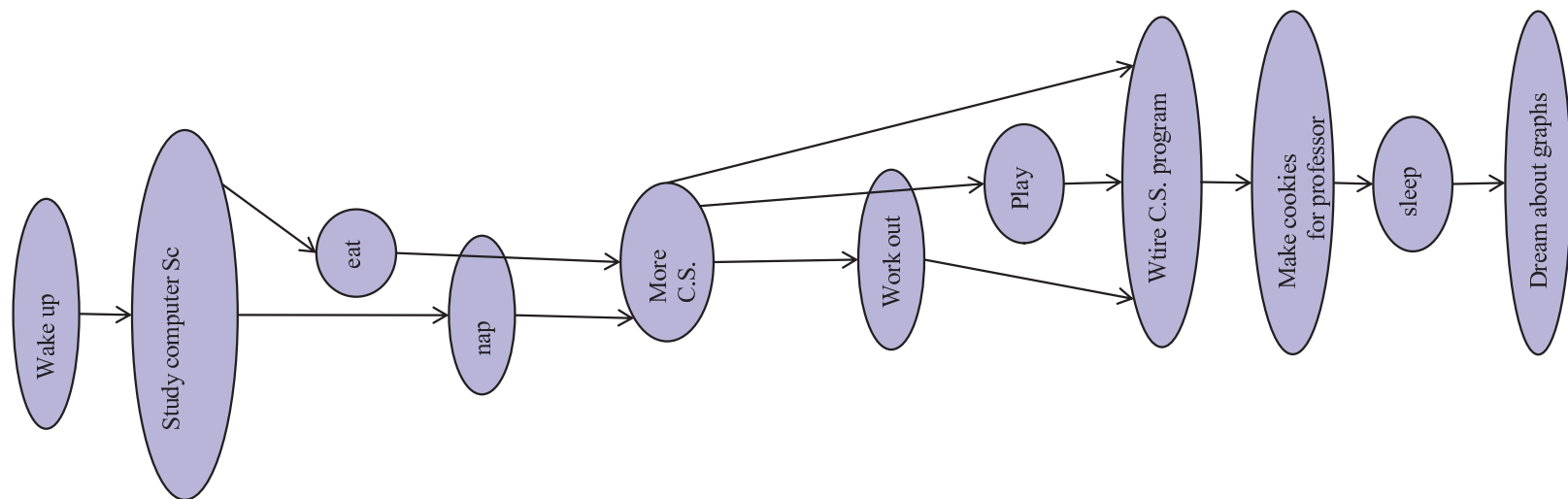
Definition of Topological Sort

- Topological sort is a method of arranging the vertices in a directed acyclic graph (DAG), as a sequence, such that no vertex appear in the sequence before its predecessor.
- The graph in (a) can be topologically sorted as in (b)



Topological Sorting

Number vertices, so that (u, v) in E implies $u < v$

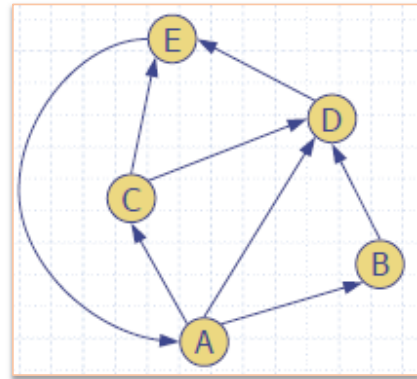


Topological Sorting

- A **directed acyclic graph (DAG)** is a digraph that has **no directed cycles**.
- A topological ordering of a digraph is a numbering v_1, \dots, v_n of the vertices such that for every edge (v_i, v_j) , we have $i < j$.

Theorem: A digraph admits a topological ordering if and only if it is a DAG.

Lemma: A digraph is DAG if and only if a DFS of G yields no back edges.

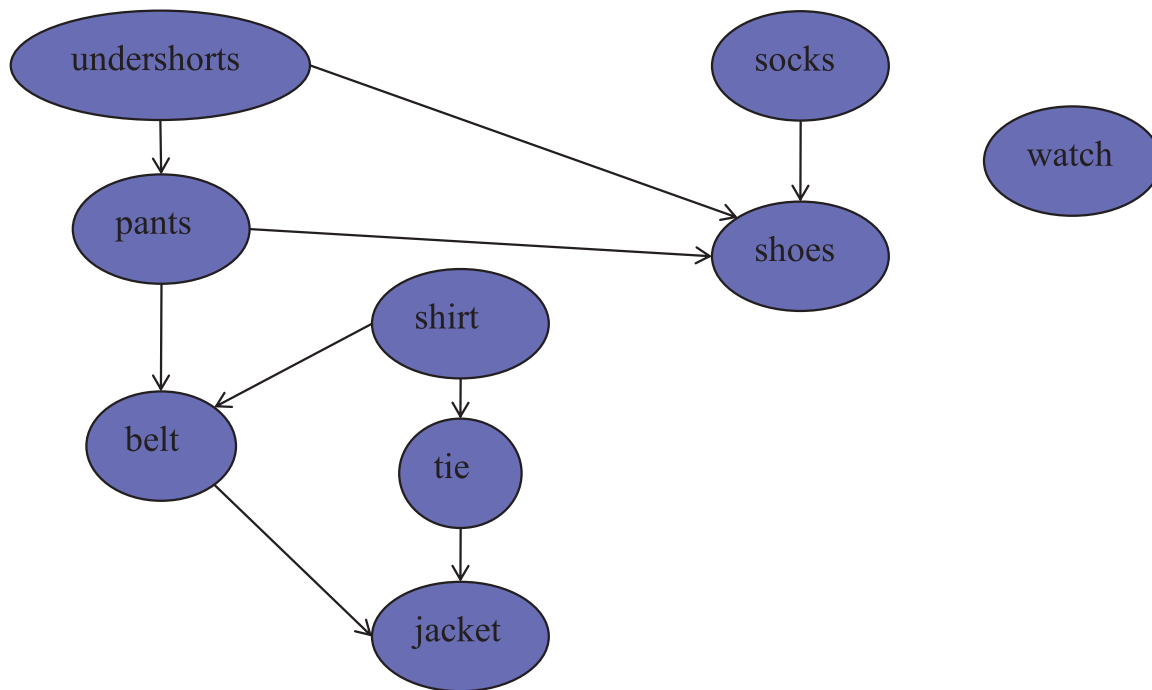


Algorithm for Topological Sorting

Topological Sort(G)

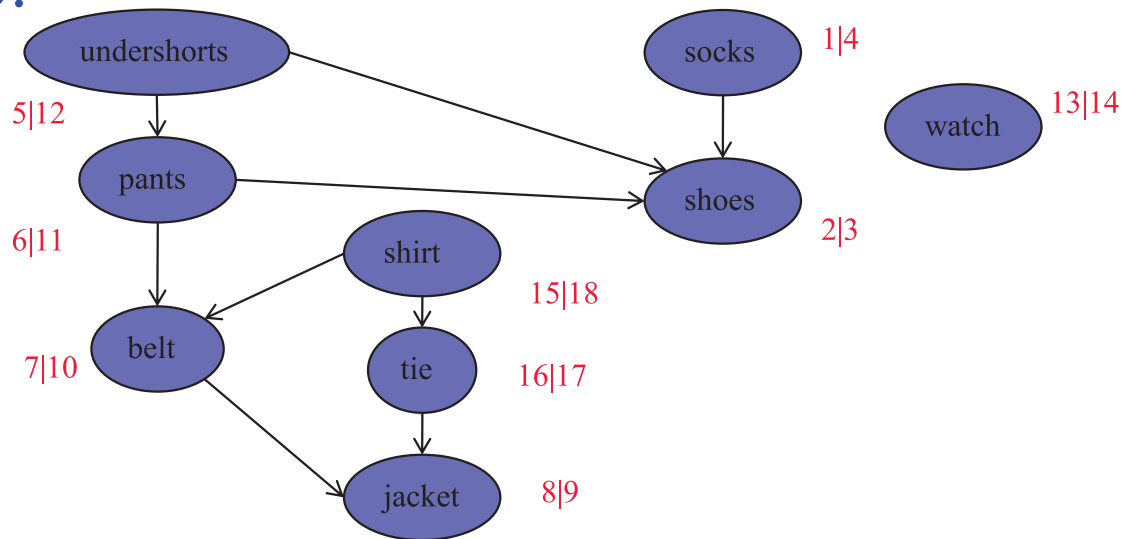
1. Call **DFS(G)** to compute finishing time $f[v]$ for each vertex v .
2. As each vertex is finished, insert it onto the front of a linked list.
3. **Return** the linked list of vertices.

Example: Digraph

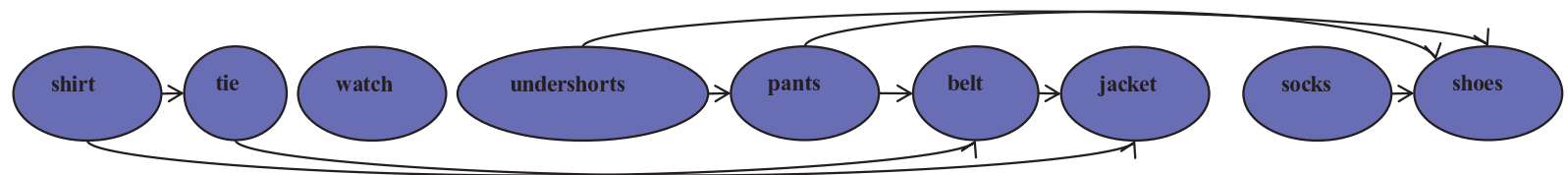


Example: Topological Sorting

DFS:



Events ordering after Topological Sort:



Time complexity for topological sort

- Runtime for a topological sort is $O(V+E)$
- Since DFS takes $O(V+E)$ time
- and $O(1)$ time to insert each of the $|V|$ vertices onto the front of the linked list.