Depth-First Search (DFS)

- another archetype for many important graph algorithms
- methodically explore *every* vertex and *every* edge
- Input: Given $G = (V, E)$

  Output: (1) Two timestamps for every $v \in V$
  $d[v] =$ when v is first discovered.
  $f[v] =$ when v is finished.
  and (2) classification of edges

Depth-First Search (DFS)

- DFS idea: *go as far as possible, then "back up"* :
  - edges are explored out of the most recently discovered vertex v that still have unexplored edges leaving
  - when all of v's edges have been explored, the search "backtracks" to explore edges leaving the vertex from which v was discoverd.
- Three-color code for search status of vertices
  - **White** = a vertex is undiscovered
  - **Gray** = a vertex is discovered, but its processing is incomplete
  - **Black** = a vertex is discovered, and its processing is complete

Review: Queue and Stack

- Queues and stacks are dynamic sets in which the elements removed from the set by the delete operation is prescribed.
- The queue implements a First-In-First-Out (FIFO) policy.
  The stack implements a Last-In-First-Out (LIFO) policy.
- Queue supports the following operations:
  Enqueue(Q,v): insert element v into the queue Q
  Dequeue(Q,v): delete element v from the queue Q
- There are several way efficient ways to implement queues and stacks.
  Section 10.1 describes an implementation by using arrays.

DFS

Pseudocode

```
DFS(G)    // main routine
for each vertex u in V
   color[u] = ‘‘white’’
endfor
time = 0
for each vertex u in V
   if color[u] = ‘‘white’’
      DFS-Visit(u)
   endif
endfor
// end of main routine
```

```
DFS-Visit(u)    // subroutine
color[u] = ‘‘gray’’
time = time + 1
d[u] = time
for each v in Adj[u]
   if color[v] = ‘‘white’’
      DFS-visit(v)
   endif
end for
color[u] = ‘‘black’’
time = time + 1
f[u] = time
// end of subroutine
```

DFS

- Vertices, from which exploration is incomplete, are proceessed in a LIFO stack.
- Running time: $\Theta(|V| + |E|)$

  not big-O since guaranteed to examine every vertex and edge.
- More properties of DFS, see pp.606-608 of [CLRS,3rd ed.]

DFS

Classification of edges

- **T** = Tree edge = encounter new vertex (gray to white)
- **B** = Back edge = from descendant to ancestor (gray to gray)
- **F** = Forward edge = from ancestor to descendant (gray to black)
- **C** = Cross edge = any other edges (between trees and subtrees): (gray to black)

Note: In an undirected graph, there may be some ambiguity since edge (u,v) and (v,u) are the same edge. Classify by the first type that matches.

DFS vs. BFS

1. DFS: vertices from which the exploring is incomplete are processed in a LIFO order (stack)

   BFS: vertices to be explored are organized in a FIFO order (queue)

2. DFS contains two processing opportunities for each vertex v, when it is "discovered" and when it is "finished"

   BFS contains only one processing opportunity for each vertex v, and then it is dequeued

Applications

1. For a undirected graph,
   (a) a DFS produces only Tree and Back edges
   (b) acyclic (tree) iff a DFS yeilds no back edges

2. A directed graph is acyclic iff a DFS yields no back edges

3. Topological sort of a dag (= directed acyclic graph)

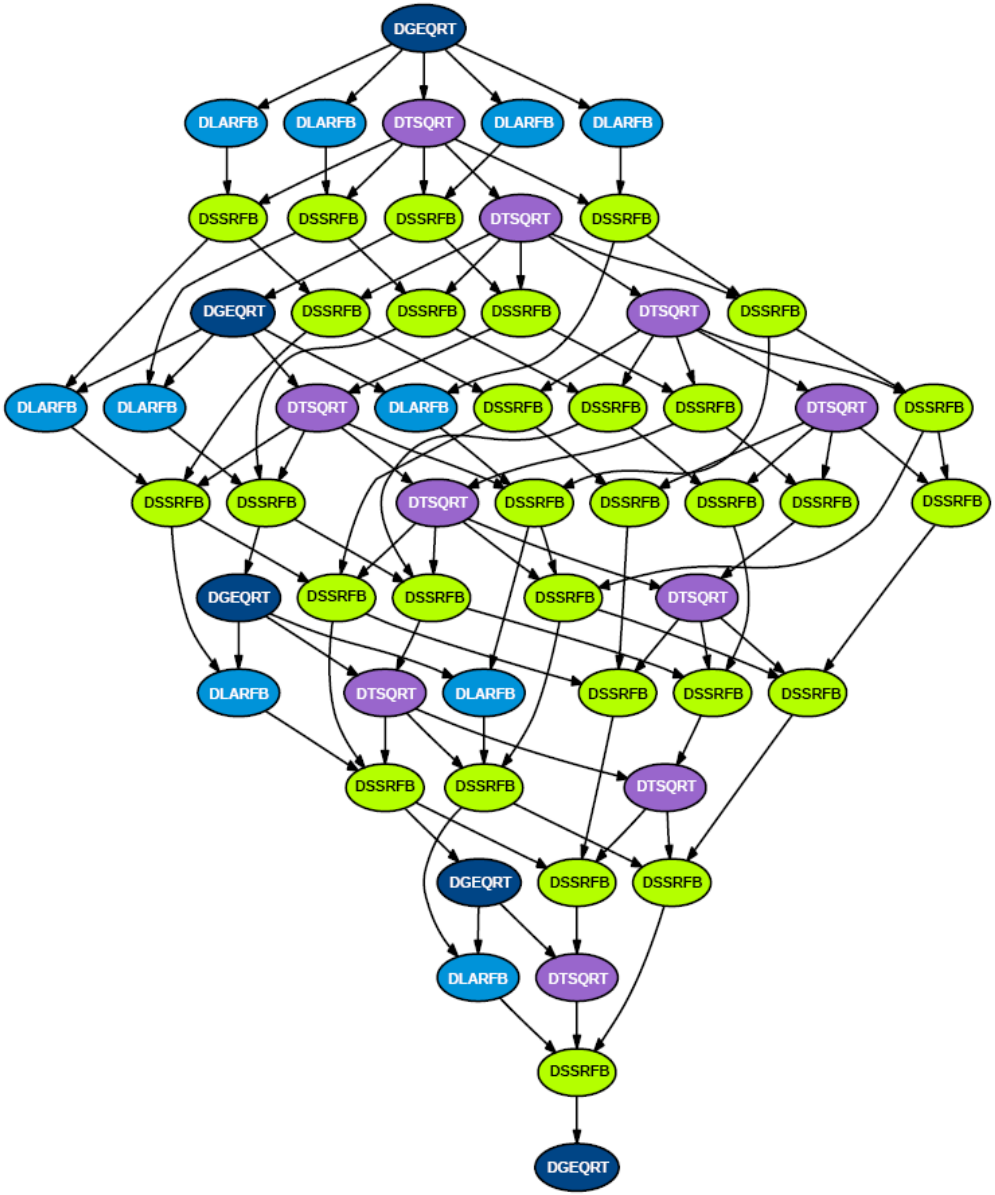
4. Strongly connected components, see Sec.22.5 of [CLRS,3rd ed.]

Topological sort

- A topological sort (TS) of a dag $G = (V, E)$ is a linear ordering of all its vertices such that if $(u, v) \in E$, then u appears before v.
- A TS is not possible if G has a cycle.
- The ordering is not necessarily unique.

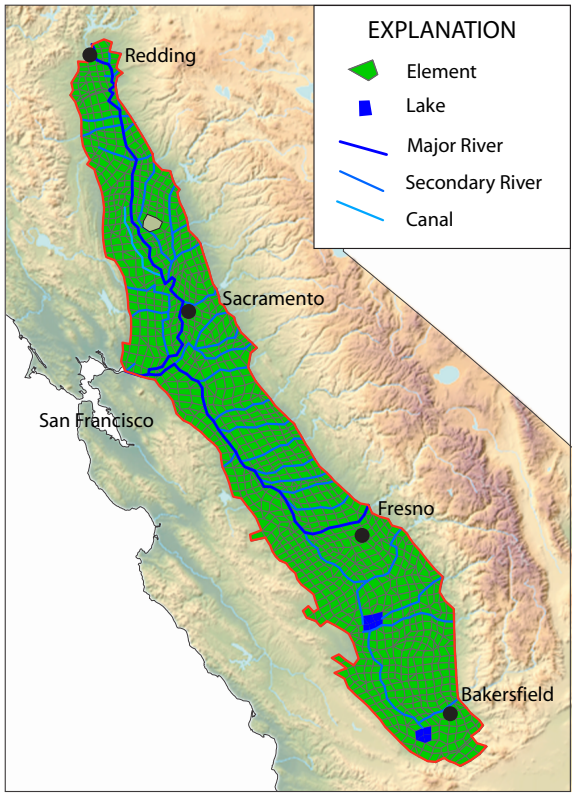
Topological sort

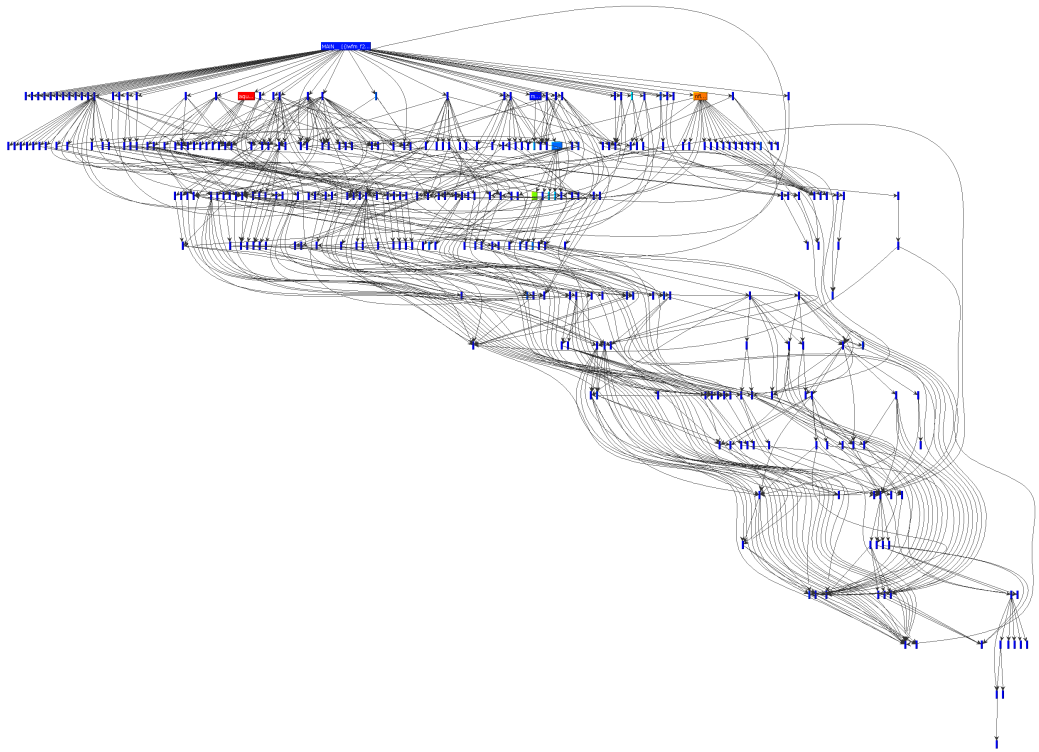
Application:



Topological sort

Application:





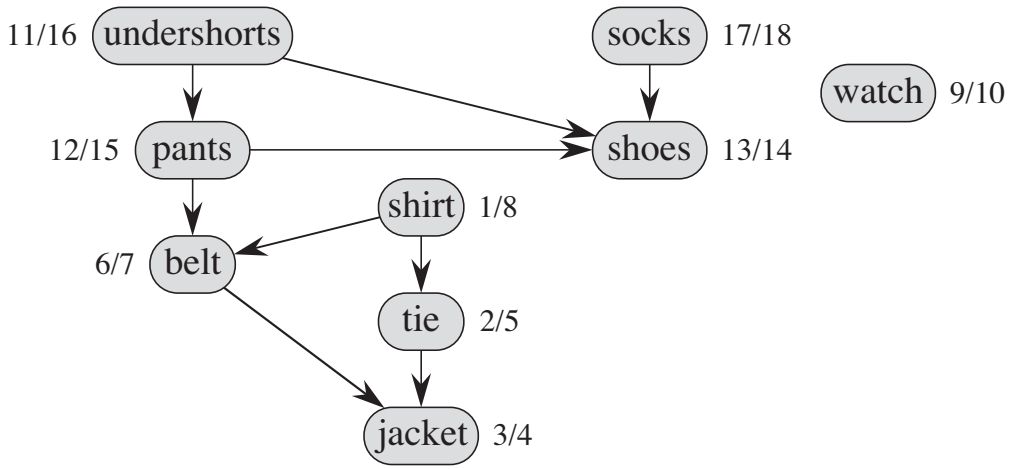
Topological sort

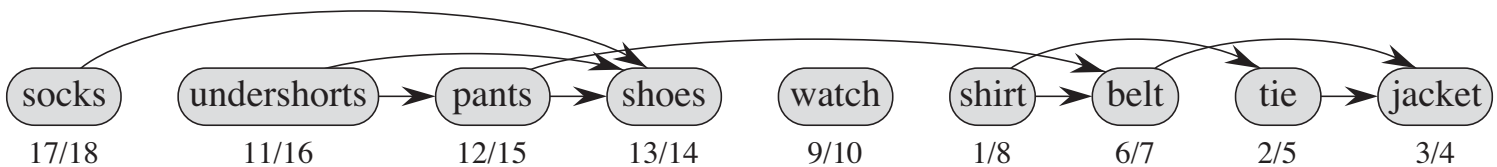
- TS Algorithm
  1. run DFS(G) to compute finishing times $f[v]$ for all $v \in V$
  2. output vertices in order of decreasing times

- Running time: $\Theta(|V| + |E|)$

Topological sort

Example:
Getting dressed and DFS





Topologically sorted

Topological sort

Theorem (correctness of the algorithm):
TS(G) produces a toplogical sort of a dag G.

Proof: *Just need to show that if* $(u, v) \in E$, *then* $f[v] < f[u]$.
When we explore edge (u, v), u is gray, what's the color of v?

- Is v gray too?
  no, because then v would be ancestor of u, edge (u, v) is a back edge, a contradiction of a dag.
- Is v white?
  yes, then v is descendant of u, by DFS, $d[u] < d[v] < f[v] < f[u]$
- Is v black?
  yes, then v is already finished. Since we're exploring (u, v), we have not yet finished u, therefore $f[v] < f[u]$