طراحی الگوریتم

۱۲ آبان ۹۸

ملکی مجد

Topic	Reference
Recursion and Backtracking	Ch.1 and Ch.2 JeffE
Dynamic Programming	Ch.3 JeffE and Ch.15 CLRS
Greedy Algorithms	Ch.4 JeffE and Ch.16 CLRS
Amortized Analysis	Ch.17 CLRS
Elementary Graph algorithms	Ch.6 JeffE and Ch.22 CLRS
Minimum Spanning Trees	Ch.7 JeffE and Ch.23 CLRS
Single-Source Shortest Paths	Ch.8 JeffE and Ch.24 CLRS
All-Pairs Shortest Paths	Ch.9 JeffE and Ch.25 CLRS
Maximum Flow	Ch.10 JeffE and Ch.26 CLRS
String Matching	Ch.32 CLRS
NP-Completeness	Ch.12 JeffE and Ch.34 CLRS

topics

- DAG
- topological sort
- Strongly connected components

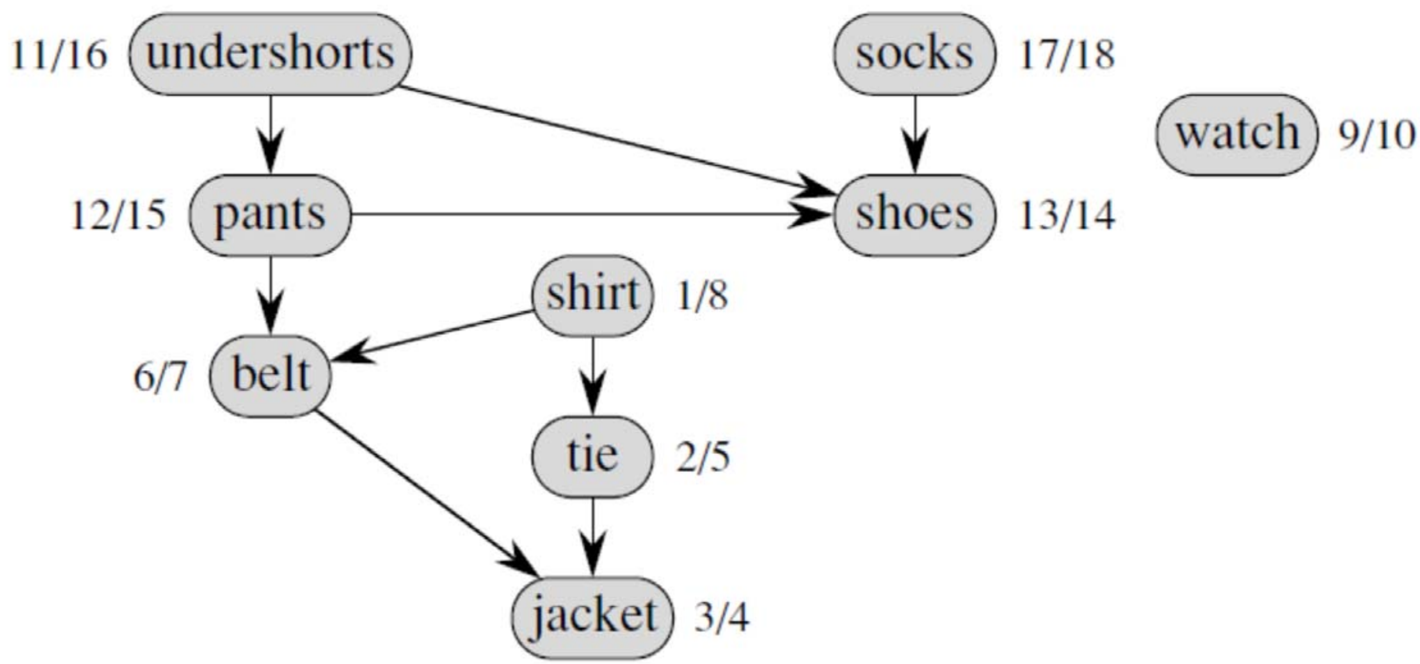
DAG

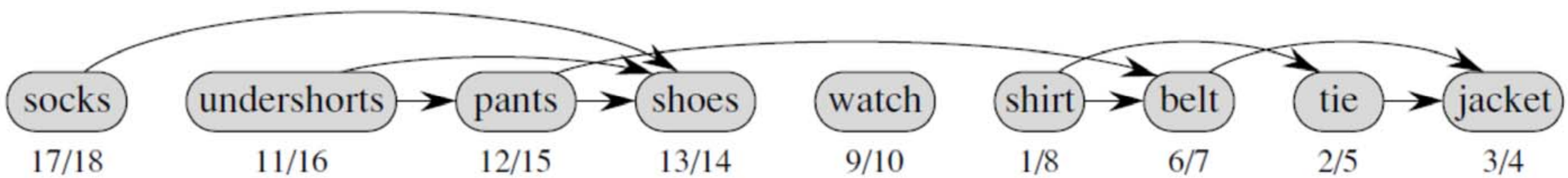
- Directed Acyclic Graph

- Directed acyclic graphs are used in many applications to indicate precedences among events

topological sort

- A ***topological sort*** of a dag $G = (V, E)$ is a linear ordering of all its vertices such that if G contains an edge (u, v), then u appears before v in the ordering.
  - ordering of its vertices along a horizontal line so that all directed edges go from left to right
- If the graph is not acyclic, then no linear ordering is possible.


11/16 undershorts
socks 17/18
watch 9/10
12/15 pants
shoes 13/14
shirt 1/8
6/7 belt
tie 2/5
jacket 3/4


socks
17/18
undershorts
11/16
pants
12/15
shoes
13/14
watch
9/10
shirt
1/8
belt
6/7
tie
2/5
jacket
3/4

TOPOLOGICAL-SORT*(G)*

1 call DFS*(G)* to compute finishing times *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

- We can perform a topological sort in time $\Theta(V + E)$, since depth-first search takes $\Theta(V + E)$ time and it takes $O(1)$ time to insert each of the $|V|$ vertices onto the front of the linked list

Lemma 22.11

- A directed graph *G* is acyclic if and only if a depth-first search of *G* yields no back edges

Theorem 22.12

- TOPOLOGICAL-SORT*(G)* produces a topological sort of a directed acyclic graph *G*.

Decomposing a directed graph

- a classic application of depth-first search:
  - decomposing a directed graph into its strongly connected components.
- Do this decomposition using two depth-first searches
- Many algorithms that work with directed graphs begin with such a decomposition. After decomposition, the algorithm is run separately on each strongly connected component. The solutions are then combined according to the structure of connections between components

Strongly connected components

- A strongly connected component of a directed graph $G = (V, E)$ is a maximal set of vertices $C \subseteq V$ such that for every pair of vertices u and v in C, we have both $u \rightsquigarrow v$ and $v \rightsquigarrow u$; that is, vertices u and v are reachable from each other

idea

- The algorithm uses the transpose of *G*,
- G^T = (V, E^T), where $E^T = \{(u, v) : (v, u) \in E\}$.
- The time to create G^T is $O(V + E)$.
- G and G^T have exactly the same strongly connected components

STRONGLY-CONNECTED-COMPONENTS*(G)*

1 call $DFS(G)$ to compute finishing times $f\ [u]$ for each vertex u

2 compute G^T

3 call $DFS(G^T)$, but in the main loop of DFS, consider the vertices in order of decreasing $f\ [u]$ (as computed in line 1)

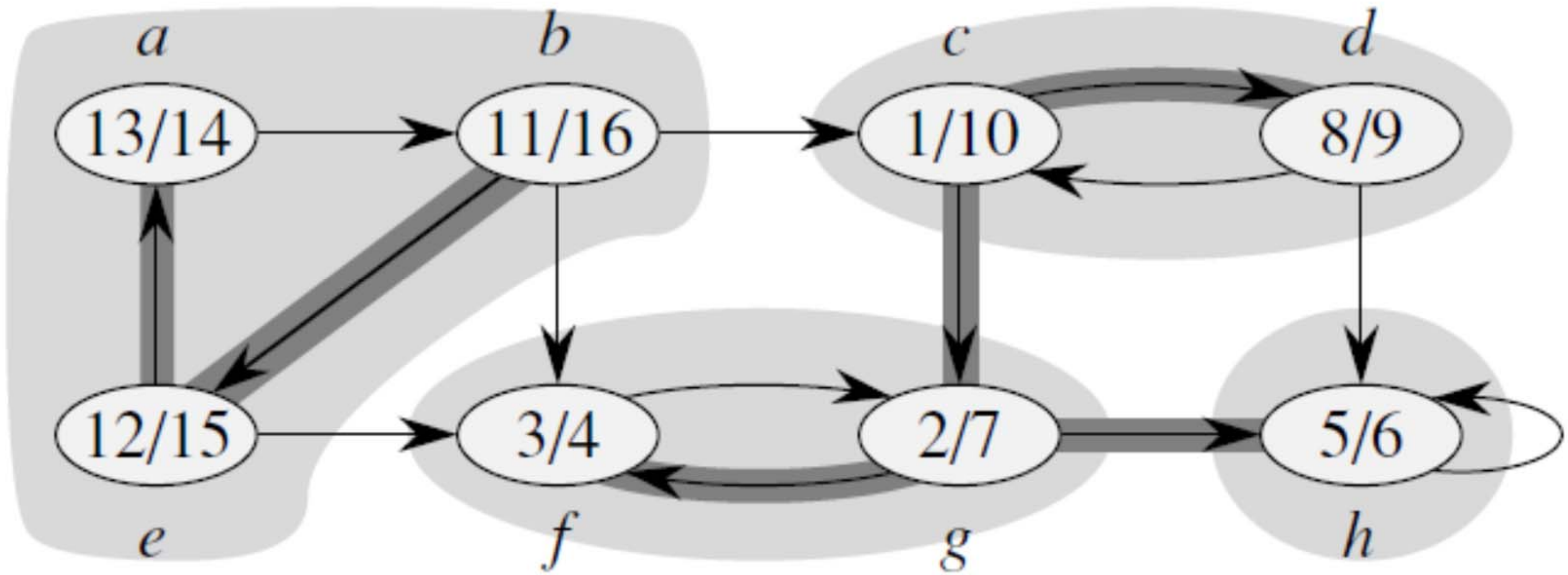
4 output the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component

component graph $G^{SCC} = (V^{SCC}, E^{SCC})$

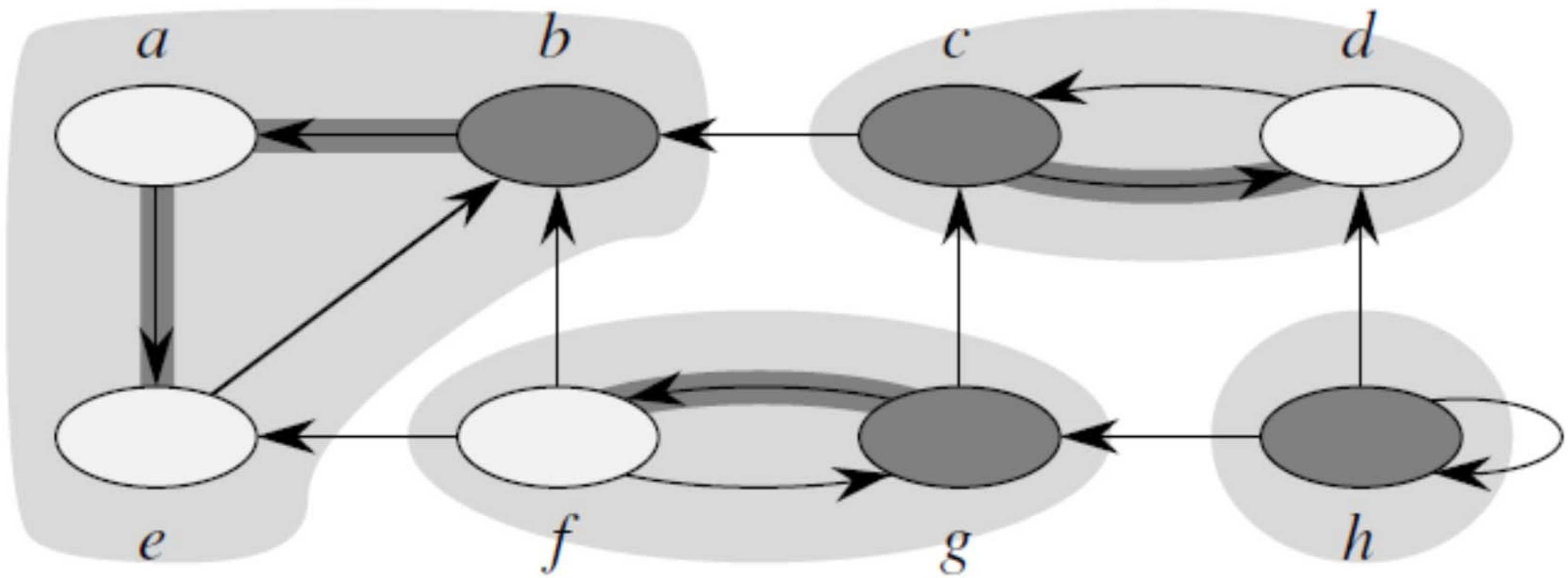
- Suppose that G has strongly connected components $C_1, C_2, \ldots, C_k$
- The vertex set V^{SCC} is $\{v_1, v_2, \ldots, v_k\}$, and it contains a vertex v_i for each strongly connected component C_i of G
- There is an edge $(v_i, v_j) \in E^{SCC}$ if *G* contains a directed edge (x, y) for some $x \in C_i$ and some $y \in C_j$
  - contracting all edges whose incident vertices are within the same strongly connected component of *G*

the component graph is a **dag**

strongly connected components
+labeled with its discovery and finishing times



the transpose of graph



The acyclic component graph G^{SCC}

It is a dag