CS 395 – Analysis of Algorithms

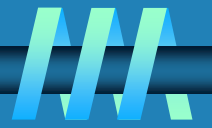
Chapter 4 – Decrease-and-Conquer

Read 4.2 & 4.3

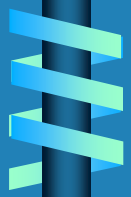
- Shellsort (reading assignment)

- Directed Acyclic Graphs (DAGs)
- Topological Sorting
  - Using DFS
  - Using Source-Removal
- Weekly Assignments 8-11 - sorting

Shellsort (Shell's Sort) 4.1

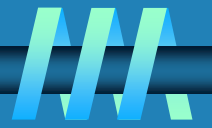


- **(Is basically…) An improved version of Insertion Sort**
  - The elements are divided into *segments* or *increments*
  - Corresponding elements in the segments form *sub-lists*
  - These sub-lists are sorted in each pass
- In each pass, increment value is decreased
  - In last pass increment value is 1
- **Time efficiency depends on *increments***
  - **with halving increments: $O(n \log n)$**
- **Space efficiency: $O(1)$**
- **Stability: yes**



Shell sort where interval is cut in half

Shellsort (Shell's Sort)

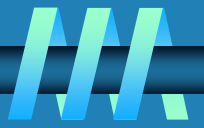


- **Key is to choose *increments***

What is the next number in this series?

- **(…, 121, 40, 13, 4, 1) – one of the best**

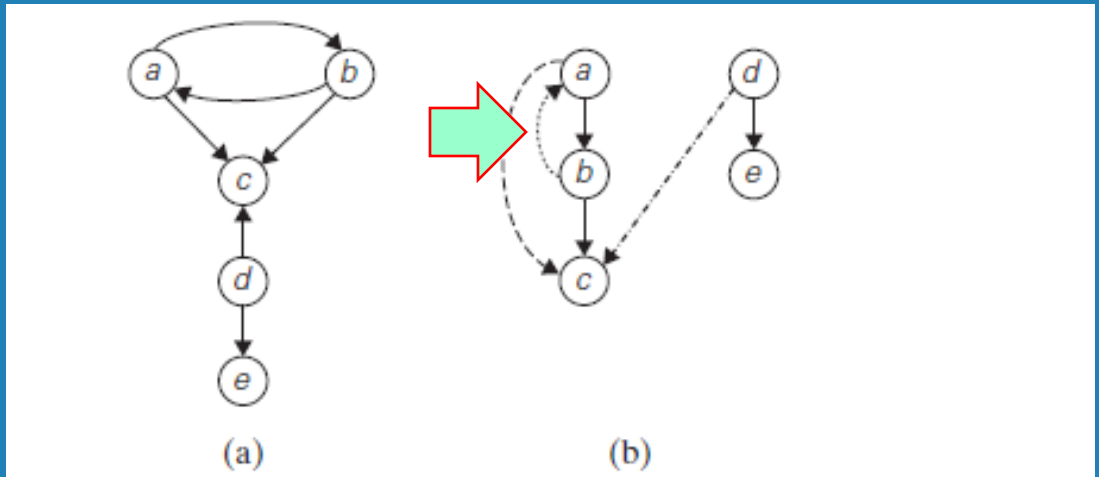
Pseudocode for Shell Sort



```
ALGORITHM ShellSort(A[0..n - 1])
//Sorts a given array by insertion sort
//Input: An array A[0..n - 1]) of n orderable elements
//Output: Array A[0..n - 1]) sorted in nondecreasing order
interval ← 1
while interval < n /3 do
    interval ← interval * 3 + 1
while interval > 0 do
    for i ← interval to n - 1 do
        v ← A[i]
        j ← i - 1
        while j > interval - 1 and A[j - interval] > v do
            A[j] ← A[j - interval]
            j ← j - interval
        A[j] ← v
    interval ← (interval - 1) /3
```

Digraphs 4.2

- Digraph = directed graph

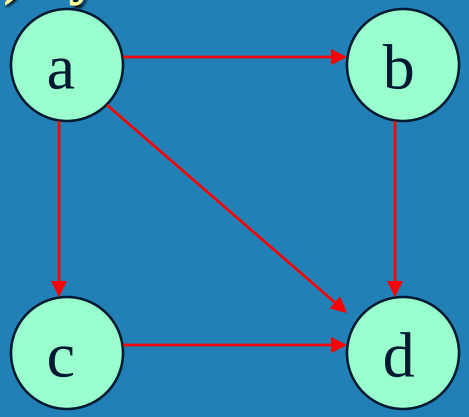


(a) Digraph. (b) DFS forest of the digraph for the DFS traversal started at *a*.

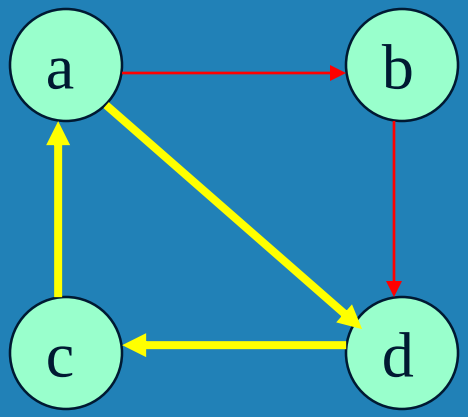
- four types of edges in a DFS forest of a directed graph:
  - tree edges (ab, bc, de),
  - back edges (ba) from vertices to their ancestors,
  - forward edges (ac) from vertices to their descendants in the tree other than their children,
  - cross edges (dc)

DAGs

DAG: a directed acyclic graph, i.e. a directed graph with no (directed) cycles



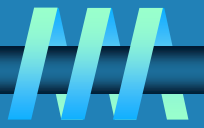
a dag



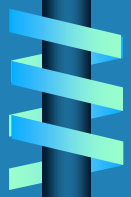
not a dag

Can you spot another cycle?

Topological Sorting

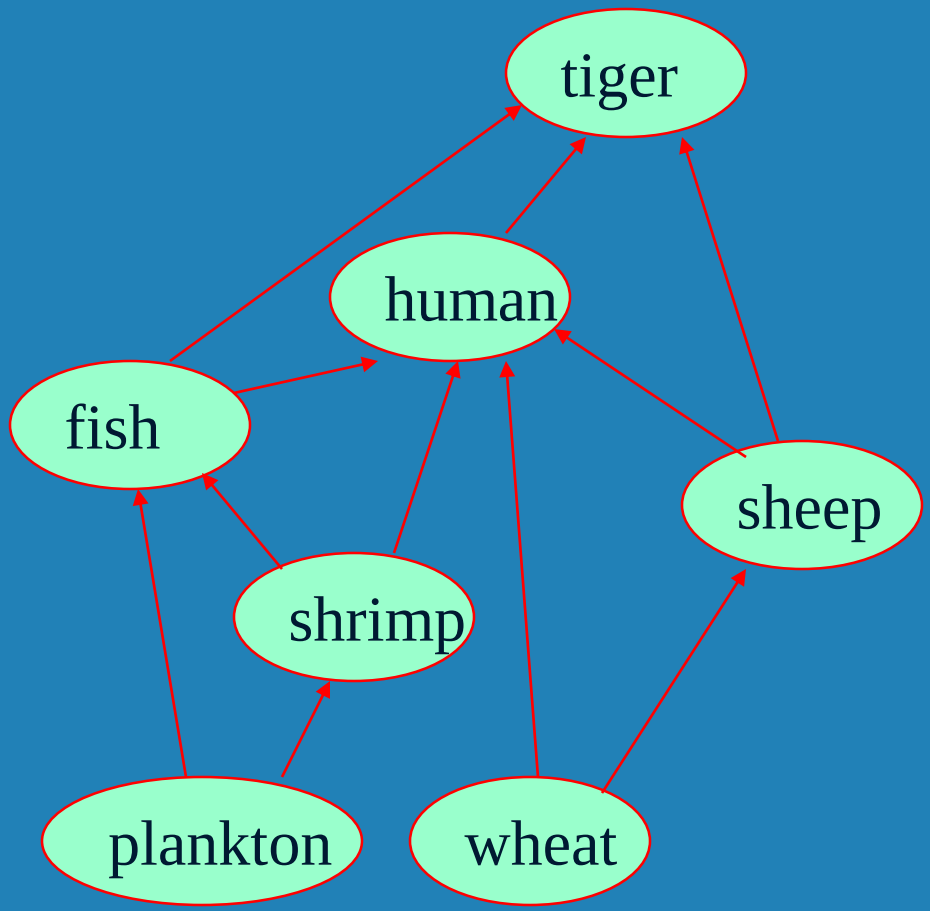


- A topological sort of a DAG G = (V, E) is a linear ordering of all its vertices such that if G contains an edge (u, v), then u appears before v in the ordering.
- Being a DAG is a necessary condition for topological sorting to be possible.
- Arise in modeling many problems that involve prerequisite constraints (construction projects, document version control) – Remember PERT diagrams from CS 383?



Topological Sorting Example 1

- Order the following items in a food chain



Topological Sorting Example 2

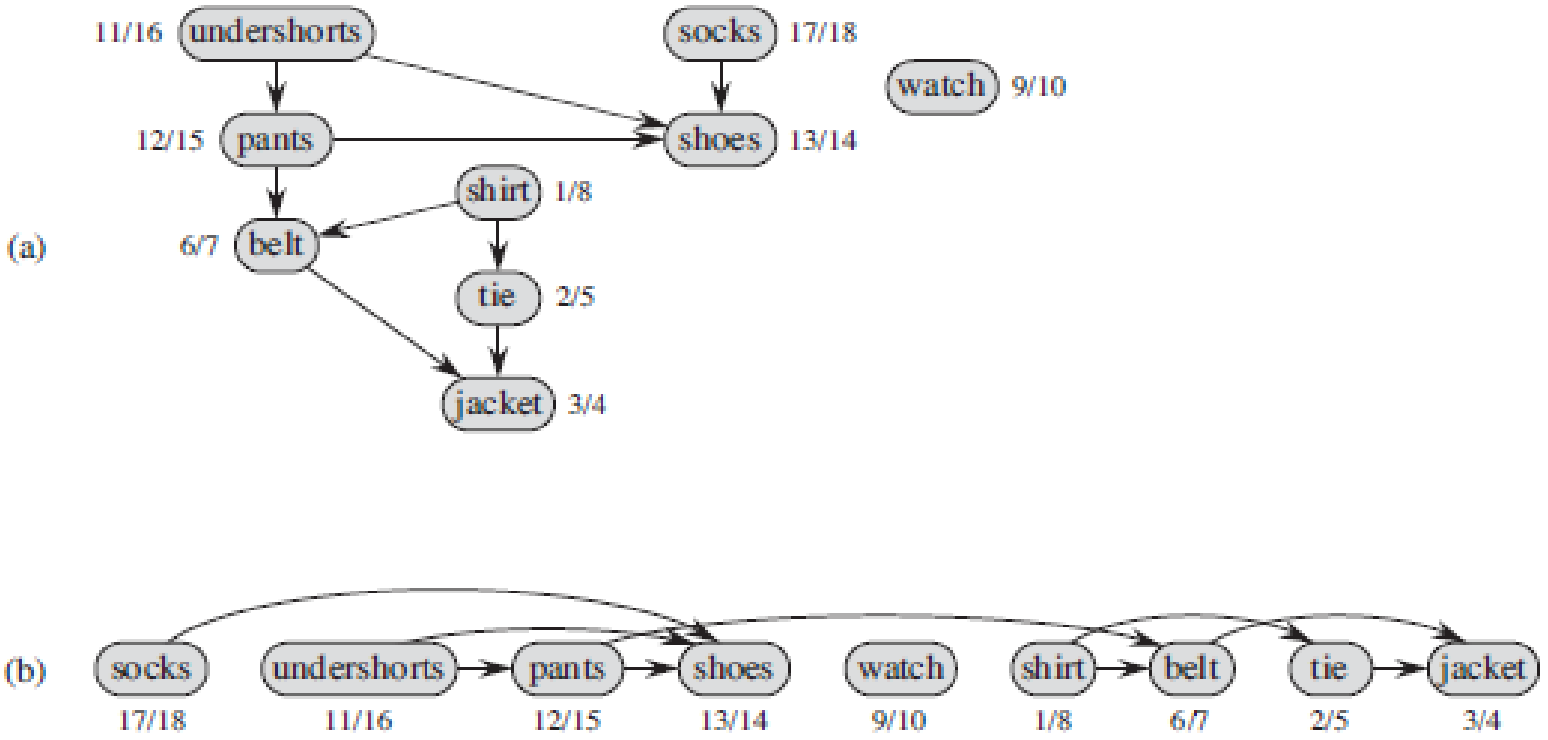
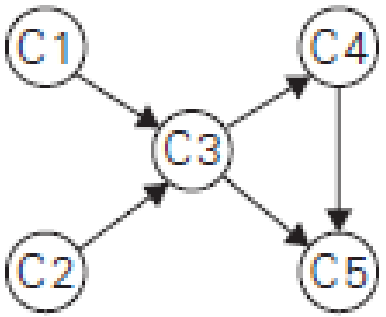


Figure 22.7 **(a)** Professor Bumstead topologically sorts his clothing when getting dressed. Each directed edge (u, v) means that garment u must be put on before garment v. The discovery and finishing times from a depth-first search are shown next to each vertex. **(b)** The same graph shown topologically sorted, with its vertices arranged from left to right in order of decreasing finishing time. All directed edges go from left to right.

DFS-based Algorithm

- **DFS-based algorithm for topological sorting**
  - **Perform DFS traversal, noting the order vertices are popped off the traversal stack**
  - **Reverse order solves topological sorting problem**
  - **Back edges encountered?→ NOT a dag!**
- **Efficiency:**
  - **adjacency matrices: Θ(|V|2)**
  - **adjacency lists: Θ(|V|+|E|)**



(a)

$C5_1$
$C4_2$
$C3_3$
$C1_4$ $C2_5$

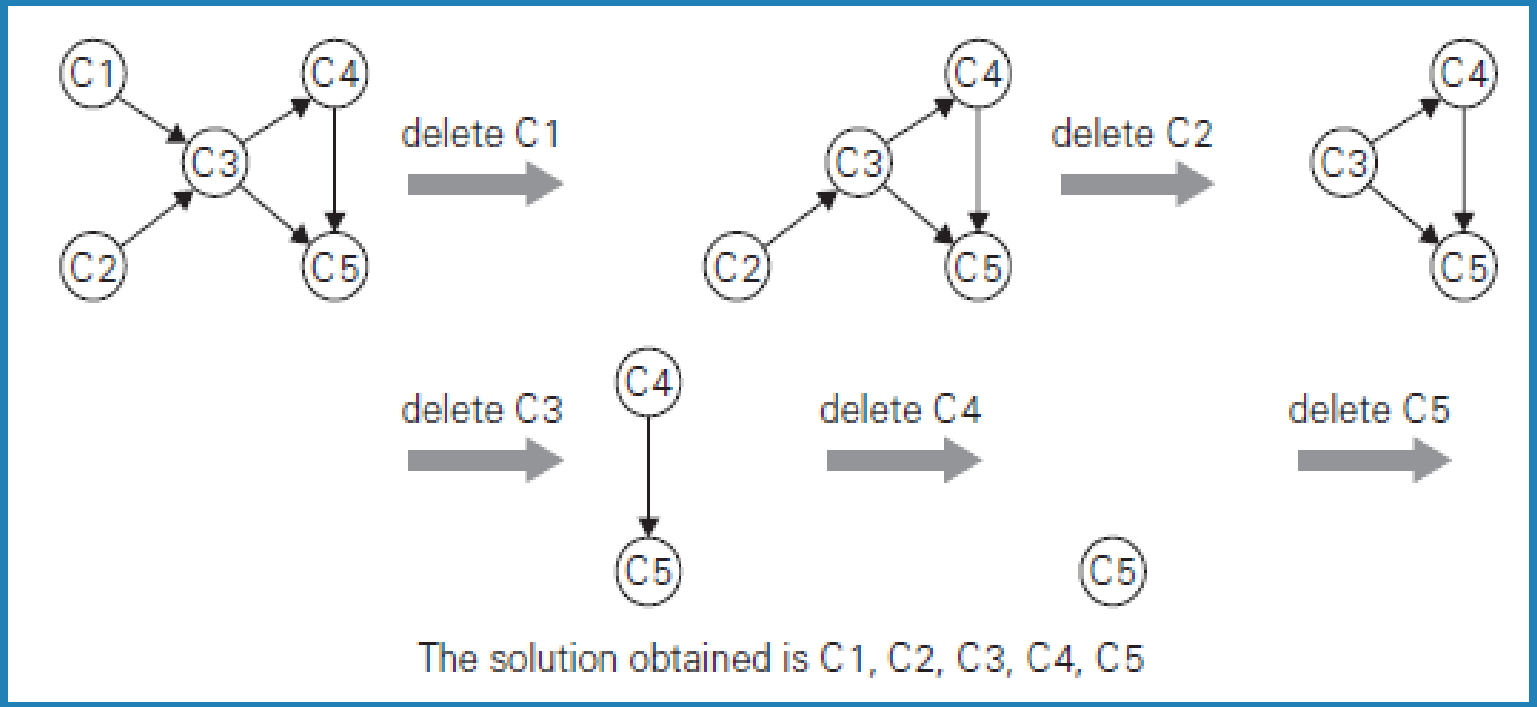
(b)

The popping-off order:
C5, C4, C3, C1, C2
The topologically sorted list:
C2 C1→C3→C4→C5

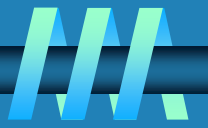
(c)

Source Removal Algorithm

- **Source removal algorithm**
  - **Repeatedly identify and remove a source (a vertex with no incoming edges) and all the edges incident to it until either no vertex is left (problem is solved) or there is no source among remaining vertices (not a DAG)**
- **Efficiency: same as efficiency of the DFS-based algorithm**



Weekly Assignments 8-11



- One per lecture you have to program:
  - Selection sort - today
  - Insertion sort – next class
  - Quicksort – two classes from now
- The 4th lecture you will write a program that combines the sorting algorithm and prints out a table with the results of the run time of each of the algorithms in milliseconds:

Number of Elements	Selection Sort	Insertion sort	Quicksort
100	50	60	30
1000	700	800	400
10000	8000	1200	6000