Assignment 11
Published on 2020-12-05,

Estimated Time: 30 minutes,
Max.grade 10‰

1 Introduction

If you know how to do topological sorting using the DFS traversal (recording their discovery and final visit times), you can immediately start with the items in Section 2. You can also read an inspirational slide deck on TOpological Sorting from University of California, Davis:
`https://www.cs.ucdavis.edu/~bai/ECS122A/Notes/DFSapps.pdf`. Or (Goodrich2011, p.633), *Section 13.4.3 Directed Acyclic Graphs.*
In the example below we show the following:

- Draw the visual representation from its adjancency list representation.
- Add two more edges to the graph.
- Run the DFS to check, if it is still a DAG (directed acyclic graph). We do that DFS in a repeatable order (always pick the lexicographically smallest vertex, if there is a choice).
- Use the DFS visiting order ("final" timestamp) to create the topological sorting.

1.1 Sample DAG

Mr. Bunbury dresses every morning according to certain rules: He always puts on his Trousers after Underpants, his Sweater after his Belt. He also puts on his Face-mask after the Sweater (otherwise it would displace the Face-mask as it is pulled over the head), glasses are put on after the Face-mask, since otherwise they get foggy, and so on.
All the rules are shown in graph G Figure 1 (it is also the adjecency list representation; it shows all the before-after binary relationships). Square cells that are crossed diagonally show null-pointers (i.e. they denote where the adjacency lists end). Mr. Bunbury has figured how he can get dressed according to these rules. To make it even more intuitive, he made a visual representation of the rules as in Figure 2.
Somebody tells Mr. Bunbury that his Coat and Boots may be contaminated with viruses, but hands should be clean as he applies the Face-mask (this adds two red arrows to the diagram in Figure 2). At this point Mr. Bunbury needs advice – one feasible way to get dressed by *topologically sorting* the clothing items (or an evidence of a contradiction: a loop of arrows that prove that the updated graph is not a *Directed Acyclic Graph* anymore and it cannot be sorted).

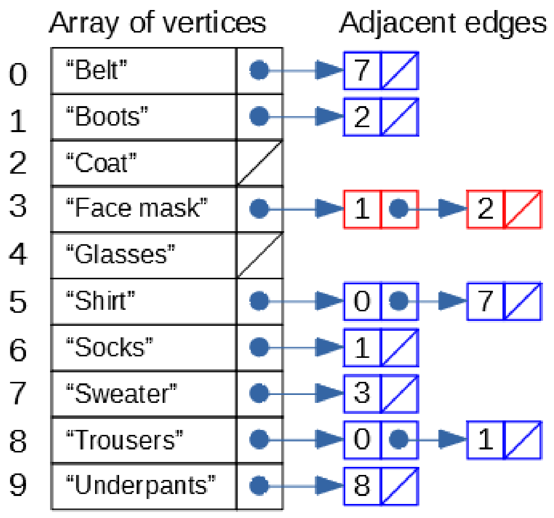

Figure 1: Adjacency List Representation

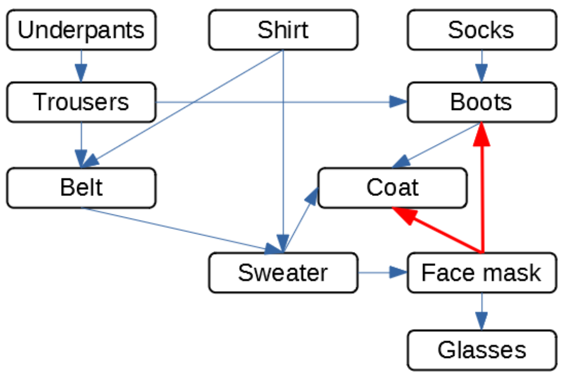

Figure 2: Graph G visualized

1.2 DFS Traversal Example

The Depth-First-Search traversal is described by a recursive algorithm having this pseudocode:

```
DFS($G$)
1  for each vertex $u \in G.V$ :
2      $u.color$ = WHITE
3      $u.parent$ = NIL
4  $time = 0$
5  for each vertex $u \in G.V$ :
6      if $u.color$ == WHITE :
7          DFS_VISIT($G, u$)
```

The loop on Line 5 (see the pseudocode of DFS(G) above) visits vertices $u \in G.V$ in the lexicographic order. (It may also happen that DFS_VISIT(G, u) is invoked just once, if a single DFS call is enough to visit every vertex and to turn it black.
Here is the code of DFS_VISIT(G, u) (the DFS-visit for vertex $u \in G.V$). Please pay attention to the two parameters $u.d$ (discovery time – when the DFS traversal enters u for the first time) and $u.d$ (finish time – when the DFS traversal leaves u forever, since u is visited along with all its freshly discovered child vertices).

```
DFS_VISIT($G, u$)
1   $time = time + 1$
    (white vertex u was just discovered)
2   $u.d = time$
3   $u.color$ = GRAY
    (explore $(u, v_1)$, $(u, v_2)$, $\ldots$)
4   for each $v \in G.Adj[u]$ :
5       if $v.color$ == WHITE :
6           $v.parent = u$
7           DFS_VISIT($G, u$)
    (vertex u now fully processed)
8   $u.color$ = BLACK
9   $time = time + 1$
10  $u.f = time$
```

Once again, we assume that the white neighbors of u (for example, v_1, v_2) are visited in their lexicographic order. These assumptions make DFS traversal repeatable (without such assumption there are typically multiple ways how to do DFS-style visits).
Let us run this DFS pseudocode on the clothing graph. We start by visiting the vertex "Belt" (as it is alphabetically first), so its discovery time $u.d = 1$. After that we visit all its neighbors (in their alphabetical order, if there are several). When we finish processing "Belt", we find the next vertex (alphabetically first among those unvisited or having color WHITE), and so on. The result of these activities is shown in Figure 3 (each vertex in G has a green pair (d/f) that shows the discovery/finish time).

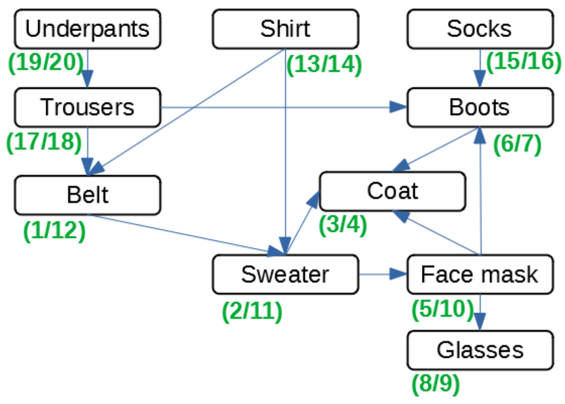

Figure 3: Graph G with DFS discovery/finish

1.3 Getting Topological Sorting

```
TOPOLOGICAL_SORT($G$)
    (result accumulates in reverse order)
1   $S$ = STACK.EMPTY()
2   call DFS($G$)
3   for each $u \in G.V$ :
4       as soon as $u.f$ (finish time) is assigned :
5       $S.push(u)$
6   return $S$
```

We can look at Figure 3 and list all the nodes in the reverse order of their finish times (their values are $20, 18, 16, 14, 12, 11, 10, 9, 7, 4$), we would get the following topological order (one feasible way how Mr. Bunbury can dress himself). For every clothing item we also specify the time value when that vertex was finished (painted black).

(20) Underpants; (18) Trousers;
(16) Socks; (14) Shirt;
(12) Belt; (11) Sweater;
(10) Face mask; (9) Glasses;
(7) Boots; (4) Coat.

The sequence shows that DFS (and the formal rule that always picks the lexicographically smallest item) causes a reasonable advice to get dressed.

1.4 Graphs that are Not DAGs

Topological sorting is not always doable – if there is any loop in the graph, it is no longer a DAG (a directed acyclical graph). During the DFS traversal it is possible to verify, if a graph is a DAG or not.
Let us remember, how all directed edges fall into 4 categories as we perform DFS:

Tree/Discovery edges
: All the edges (u, v) visited by DFS as the depth-first forest is built (see Line 4 in pseudocode DFS_VISIT(G, u)).

Back edges
: Edges connecting a vertex u with its ancestor v in some DFS tree. (If a cyclical graph contains a loop (v, v) from a vertex to itself, it is considered a back edge as well.)

Forward edges
: Non-tree edges (u, v) connecting u to a descendant v in some DFS tree.

Cross edges
: All the other edges – a cross edge connects a vertex to a vertex that is neither its ancestor nor its descendent

Statement. A graph is a DAG iff there is no back edge: all edges during the DFS traversal are either discovery edges, forward edges or cross edges.
Example. Consider the graph in Figure 4. It has a back-edge (D, A) which completes a cycle (in this example it is made from 3 edges: (A, B), (B, D), (D, A))
To identify directed graphs that are not acyclical, modify Line 4 in the pseudocode of DFS_VISIT(G, u): Before adding the new discovery/tree edge (u, v), check, if u has any neighbor v^* that currently has color GRAY; this would mean that v^* is an ancestor of u and (u, v^*) is a back-edge, which is forbidden. (Even in case when $v^* = u$, and the edge (u, v^*) is a loop to itself, the check would reveal that u has color GRAY.)

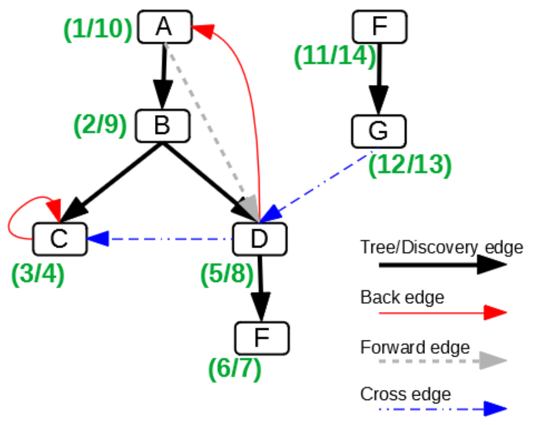

Figure 4: A graph with a cycle

2 Problem

We start with the graph shown in Figure 5.

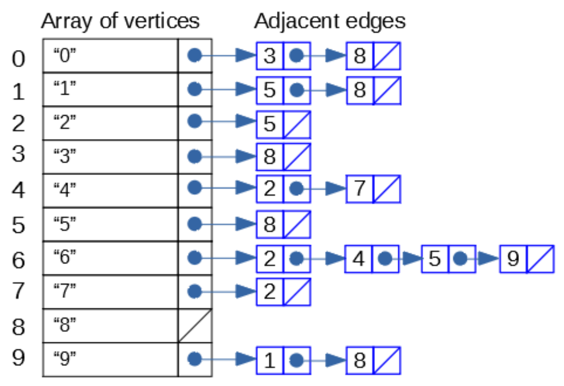

Figure 5: Adjacency list representation

(A) Draw the visual representation of the graph, each vertex is a circle (with string values `"0"` to `"9"`) inside. If there is a directed edge from one vertex to another, draw it as an arrow.
(Figures 1 and 2 show how this transformation was made for the clothing graph.)

(B) Compute the following 4 numbers from a, b, c (your last Student ID numbers):

$$\begin{cases} T = 3 \cdot ((a + b) \bmod 4) \\ U = 3 \cdot ((b + c) \bmod 3) + 1 \\ V = 3 \cdot ((c + a) \bmod 3) + 1 \\ W = 3 \cdot ((a + b + c) \bmod 3) + 2 \end{cases}$$

By $(x \bmod y)$ we denote the remainder as x is divided by y. Add to the original graph two new directed edges (T, U) and (V, W).

(For example, if $T = 5$ and $U = 7$, then there should be an arrow from vertex `"5"` to vertext `"7"`. If such an edge already exists, do not add anything.)
Draw the new graph; show the newly added edges in bold or colored differently.

(C) Run the DFS traversal algorithm on the graph just obtained in (B), mark each vertex with the pair of numbers `d/f`, where the first number `d` is the discovery time, and the second number `f` is the finishing time. (You will use all the numbers from 1 to 20: every number will be used exactly once.) It should take exactly 20 steps to do DFS enters/exits in a directed graph with 10 vertices.
If there are multiple ways how to pick a vertex to visit next, always pick the vertex with the smallest number. (Otherwise your results cannot be properly verified.)

(D) If the graph you got in (B) after adding new edges is not a DAG anymore, please say so in your answer; indicate what vertex you visited as you discovered that a back edge exists. And also show which is the cycle.
If the graph is a DAG, produce a topological sorting of its vertices (the algorithm is explained in Section 1.4 above).

Note. Your solution should contain two directed graphs ((A) and (B)), one directed graph with vertices anotated with `d/f` (in (C)) and EITHER information on the back edge+cycle OR topological sorting of the vertices.