CS350 – Winter 2019

Homework 2 Key

Due Tuesday, 5^{th} February, on paper, at the start of class. This assignment will be graded.

1. There are different ways students dress up for school every day. Most of their routines can be depicted as a graph. The following are typical practices one might follow to dress up for school: A *t-shirt* should be worn before putting on a *bike helmet*. *Socks* and *pants* should be worn before putting on *shoes*. A *jacket* should be worn before putting on a *backpack*. *Underpants* should be worn before putting on *pants*. A *t-shirt* should be worn before putting on a *jacket*.

   Draw these ordering constraints as a directed graph. Include all the items mentioned above. 1pt for each correct arrow, should be 6pts overall

   Implement one of the topological sort algorithms described in Levitin in your favorite language, run it on the above data, and output a list of items in a valid order for getting dressed. (Note: there are many valid orders; your algorithm should pick one.) **Submit your code and your output for grading**. If you use a reference for pseudocode, be sure to cite your reference.

   Note: if you don't have a favourite language, we recommend Python: it's easy to read and is at about the right level of abstraction. C-like languages have too much boilerplate that obscures what is going on. Even Python has some high-level operations, such as comprehensions, whose time complexity is hard to guess; avoid them in this class.

   - some representation of the graph (matrix or adjacency list), as an algebraic data type or as an object (1pt)
   - initialization of the graph data structure with the given data (1pt)
   - a function/method topologicalSort applied to the data structure. I would probably write it recursively, but I suppose that it could be done iteratively. (2pts)
   - inside that, I expect to see a function/method that extracts a smaller graph from the input graph. (1pt)
   - to do the above, I expect to see a function/method that finds a sink (or source either will work) (1 pt)
   - construction of the solution to the original problem by adding the sink node to the sorted result. This will be a list append (or pre-pend), so trivial, and may not warrant a method/function to encapsulate it. (1pt)
   - correct output; 8 pts, deduct 1pt for each item in the wrong place

2. Any decrease by one algorithm can be viewed as having three parts:
   (a) Extracting a problem of size $(n-1)$ from a problem of size n by removing some element
   (b) Solving the problem of size $(n-1)$, usually by applying the same algorithm recursively, until the solution is trivial, and

(c) Recombining the solution to the problem of size $(n-1)$ with the removed element, thus obtaining a solution to the original problem of size n.

For example, in a decrease-by-one topological sort, part (a) involves removing a source or sink node, part (b) involves topologically sorting the remaining nodes, and part (c) involves attaching the source or sink note to the appropriate end of the sorted list.

Insertion sort and selection sort can both be viewed as "decrease by one" algorithms. This is true in spite of the fact that Levitin classifies selection sort as a brute force algorithm. This does not this mean that Levitin is wrong in his classification — the algorithm can be viewed both ways.

For **both** Insertion sort and Selection sort, do the following.

(a) Explain how a problem of size $(n-1)$ is extracted from a problem of size n.
insertion sort: reduce the bound of the array by 1, or remove last element from list (1pt)
selection sort: find the least (or greatest) element and remove it (1pt)

(b) State the asymptotic efficiency of the process of extracting the problem of size $(n-1)$.
insertion sort: $O(1)$ (1pt) **selection sort**: $O(n)$ (1pt)

(c) Explain how the solution from applying the algorithm recursively (for the problem of size $(n-1)$) is extended to construct a solution for the original problem of size n.
insertion sort: search through the sorted array (or list) of size $(n-1)$ and find the right spot to insert the element removed in (a), and insert it there (by sliding elements down the array, or inserting a node in the list) (1pt)
selection sort: concatenate the least element to the start (or greatest element to the end) of the sorted sub-array (or list). For array implementation, this means moving the array bound by 1. (1pt)

(d) Calculate the asymptotic efficiency of the process of extending the solution to the problem of size $(n-1)$ to the solution of the problem of size n.
insertion sort: $O(n)$ (1pt) **selection sort**: $O(1)$ (1pt)