Illinois Institute of Technology
Department of Computer Science

Solutions to Third Examination

CS 430 Introduction to Algorithms
Spring, 2018

Wednesday, April 25, 2018
10am–11:15am & 11:25am–12:40pm
111 Robert A. Pritzker Science Center

Exam Statistics

105 students took the exam; 2 students were no-shows. Excluding the no-shows, the range of scores was 3–98, with a mean of 27.94, a median of 25, and a standard deviation of 18.48; undergraduates generally did much better than graduate students. The exam scores were generally disappointing, and very roughly speaking, if I had to assign final grades on the basis of this exam only, 60 and above would be an A (6), 40–59 a B (17), 20–39 a C (41), 10–19 a D (29), below 10 an E (12). Every student should have been able to get at least half credit on each problem; thus no score should have been below 50.

Problem Solutions

1. A tree with a root and n singleton-node children can occur as a Fibonacci heap only for $n \leq 2$.

   For $n = 0$, we get a single node tree when a new item is inserted by FIB-HEAP-INSERT on page 510 of CLRS3. For $n = 1$, we get a root node with a singleton-node child when two heaps of root degree 0 are combined by CONSOLIDATE on page 516. For $n = 2$, we get such a tree (a root node with two singleton-node children) when two heaps with root degree 1 (the previous case) are combined by CONSOLIDATE and the grandchild node is deleted by FIB-HEAP-DELETE on page 522.

   According to Lemma 19.1 we cannot get a root with $n \geq 3$ singleton-node children because the nth child to be acquired, y_n, must have $y_n.\mathit{degree} \geq n - 1 \geq 2$; that is, it cannot be a singleton node.

2. (a) Any edge except a self-loop can be classfied as a tree edge if the DFS starts at the base of that edge. A self-loop is always a back edge.

   (b) No, an edge can be classified as a back edge only if the two vertices connected lie on a cycle—that is, the endpoints of the edge are in the same strongly connected component.

   (c) & (d) No, an edge can be classified as a forward edge or a cross edge only if it connects vertices in two different strongly connected components. In such a case, if the edge is not classified as a tree edge, the particular classification depends on the order in which the strongly connected components are encountered.

3. (a) In fact, *any* spanning tree of G must contain the useful edges. Suppose some useful edge e from vertex u to vertex v is not in some spanning tree T. Because e is not on any cycle in G, adding it to T cannot yield a cycle so that there can be no path from u to v in T; that means that T is disconnected and hence not a spanning tree.

   (b) The proof is by contradiction. Suppose some dangerous edge e is in a minimum spanning tree T. Consider the cycle C on which e is the longest edge. Because no two edge weights are equal, every edge on C except e must have less weight than e; furthermore, since T does not contain a cycle (it is a tree, after all!) some edge x on the cycle C is not in T. Deleting edge e from T and adding edge x therefore gives a spanning of less weight than T—impossible if T is a minimum spanning tree.

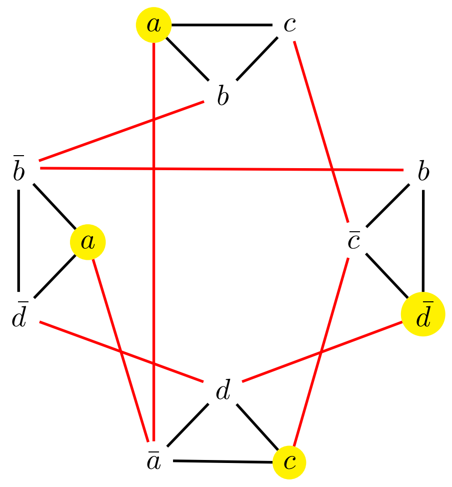
4. No, because $O(n|S|)$ is exponential in the *length* of the input, which is $O(\log |S|)$.

5. (a) No, it cannot be in NP because it is not a decision problem: it asks for the *largest* independent set in a given graph.

   (b) To prove NP-hardness, we give a reduction from 3-SAT. Given a 3-CNF expression, we need a graph that has an independent set of a certain size if and only if the formula is satisfiable. The graph is a variation on Figure 34.14 on page 1088 of CLRS3—a clause is a triangle of 3 edges, connecting nodes that are the literals of the clause. Nodes in different triangles are connected by an edge if they correspond to a variable and its negation. For example, the 3-CNF expression

$$(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$$

becomes the graph



in which black edges form the triangles joining literals in the same clause; red edges join literals that cannot be simultaneously true (contradictory literals). The yellow vertices are an independent set.

If the graph has an independent set of k vertices, there are k (black) triangles and each vertex in the independent set must come from a different triangle (clause). To obtain a satisfying assignment, we assign the value T to each literal in the independent set. Since contradictory literals are connected by (red) edges, this assignment is consistent. There may be variables that have no literal in the independent set; we can set these to any value we like. The resulting assignment satisfies the original 3-CNF formula.

On the other hand, if we have a satisfying assignment, then we can choose one literal per clause that is T. Those literals form an independent set in the graph.