Algorithms and Data Structues 2020/21

Coursework 2

This coursework is due by **4:00pm, on Friday, 20. November 2020**. This is a firm deadline. Solutions must be submitted by uploading them on the LEARN page (www.learn.ed.ac.uk) of ADS 2020/21, Left Menu item "Assessment", item on page "ADS Coursework 2". Multiple submissions are possible, but only the last submission counts.

This coursework 2 is **summative**. It counts for 50% of the overall course grade.

Late policy: http://www.inf.ed.ac.uk/student-services/teaching-organisation/for-taught-students/coursework-and-projects/late-coursework-submission

Conduct policy: http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct

1. Consider the problem of taking a set of n items with sizes $s_1, \ldots, s_n$, and values $v_1, \ldots, v_n$ respectively. We assume $s_i, v_i \in \mathbb{N}$ for all $1 \leq i \leq n$. Suppose we are also given a capacity $C \in \mathbb{N}$.

   The packing problem is the problem of finding a subset $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} s_i \leq C$ and such that $\sum_{i \in S} v_i$ is maximized subject to the first constraint.

   We write $P_{n,C}$ to denote the value $\sum_{i \in S} v_i$ of the maximum-value packing on the set of all items. For any $k \leq n$, and any $\widehat{C} \leq C, \widehat{C} \in \mathbb{N}$, we can consider the same problem on the first k items in regard to capacity $\widehat{C}$. We denote the maximum-value packing for such a subproblem by $P_{k,\widehat{C}}$.

   The goal is to develop a $\Theta(n \cdot C)$ dynamic programming algorithm to compute the optimal packing wrt. the original n items and capacity C.

   (a) Prove a suitable recurrence for $P_{k,\widehat{C}}$ that holds for all $k \leq n$ and $\widehat{C} \leq C$. [*25 marks*]

   (b) Use your recurrence above to develop a $\Theta(n \cdot C)$ dynamic programming algorithm to compute the optimal packing wrt. the original n items and capacity C. Formally justify the $\Theta(n \cdot C)$ runtime of your algorithm. [*25 marks*]

2. King Arthur has problems to administer his realm. His court contains n knights and he rules over m counties. The knights differ in their abilities and local popularity: Each knight i can oversee at most q_i counties, and each county j will revolt unless it is overseen by some knight in a given subset $S_j \subseteq \{1, \ldots, n\}$ of the knights. Only 1 knight can oversee a county, to prevent conflicts between the knights.

   The king discusses the problem with his court magician Merlin.

   (a) Show how Merlin can use the Max-Flow algorithm to efficiently compute an assignment of the counties to the knights that prevents a revolt, provided that one exists.

   How can he determine whether the algorithm was successful?

   Prove the complexity of this algorithm, *in terms of some function* $F(v, e)$, where $F(v, e)$ denotes the running time of a Max-Flow algorithm on a graph with v vertices and e edges. [*30 marks*]

   (b) Suppose that Merlin runs the Max-Flow algorithm on the encoded instance, but it does not produce a solution that fits the constraints and prevents all counties from revolting.

   King Arthur demands an explanation. While he believes that the algorithm/encoding (and proof) is correct, he doubts that Merlin has executed the algorithm correctly on the given large instance. Merlin needs to convince the king that no suitable assignment is possible under the given constraints. Re-running the algorithm step-by-step in front of the king is not an option, because the king does not have that much time.

   How can Merlin *quickly* convince the king that there is no solution, based on the structure of the Max-Flow problem? (Note that the argument must work for every instance where there is no solution, not just a particular instance.) [*20 marks*]