CMSC 28100-1 / MATH 28100-1
Introduction to Complexity Theory
Fall 2017 – Homework 2

October 7, 2017

Exercise 1. The *Knapsack Problem* is defined as follows. The input is a set S of $|S| = n$ items, each with a positive integer weight w_i and a non-negative value v_i ($i \in S$), a positive integer knapsack capacity W, and a positive integer target total value V. The output should be "Yes" if there is a subset $S' \subseteq S$ of items such that $\sum_{i \in S'} w_i \leqslant W$ and $\sum_{i \in S'} v_i \geqslant V$ and should be "No" otherwise.

(a) Give an algorithm for the Knapsack Problem. If you give a dynamic programming algorithm, state the precise "English" definitions of your sub-problems, state base cases, and the recurrence. Try to make it as efficient as you can. Prove its correctness and analyze its running time. Does it run in polynomial time?

(b) Define the εW-*Knapsack Problem* for a fixed constant ε as the Knapsack Problem above, except that you have the guarantee that, in the input, we have $\varepsilon W \leqslant w_i$ for every $i \in S$ (i.e., there are no arbitrarily small weights). Can you come up with a polynomial time algorithm for this problem? Or is your algorithm for part (a) already polynomial time for this problem? Why? Analyze. (Note: the expected answer is short and ε is *not* part of the input.)

(c) Define the εV-*Knapsack Problem* for a fixed constant ε as the Knapsack Problem above, except that you have the guarantee that, in the input, we have $\varepsilon V \leqslant v_i$ for every $i \in S$ (i.e., there are no arbitrarily small values). Can you come up with a polynomial time algorithm for this problem? Why? Analyze. (Note: the expected answer is short and ε is *not* part of the input.)

Exercise 2. We are given an array $A[1..n]$ of n rational numbers with $n \geqslant 3$ and the special property that $A[1] \geqslant A[2]$ and $A[n-1] \leqslant A[n]$. We say that an element $A[x]$ is a *local minimum of* A if $A[x-1] \geqslant A[x]$ and $A[x] \leqslant A[x+1]$. First, note that, given the condition $A[1] \geqslant A[2]$ and $A[n-1] \leqslant A[n]$, the array A must have at least one local minimum, and it is trivial to find it in $O(b \cdot n)$ time, where b is the largest bitlength of an element of A. Give an algorithm to find and return a local minimum of A in $O(b \cdot (\log n)^2)$ time. Analyze the running time of your algorithm. Prove the correctness of your algorithm.

Exercise 3. A *cycle* in an undirected simple graph $G = (V, E)$ is a sequence of vertices

$$(v_0, v_1, \ldots, v_k)$$

with $v_0 = v_k$, $k \geqslant 3$ and $\{v_i, v_{i+1}\} \in E(G)$, $v_i \neq v_{i+1}$ and $v_i \in V(G)$ for every $i \in \{0, 1, \ldots, k-1\}$.

In this exercise, we consider that access to a graph G with vertex set $V(G) = \{1, \ldots, n\}$ is given by the following subroutines.

- The subroutine vertices() returns the number of vertices n in time $O(\log n)$ (which is the bitlength of n).
- The subroutine nextneighbor(i, j) returns the smallest $k \geqslant j$ such that $\{i, k\} \in E(G)$ and returns "NONE" if no such k exists. This subroutine also takes time $O(\log n)$.

Design an algorithm that, using the subroutines above, runs in time $O(n \log n)$ and returns a cycle of the graph G if one exists and returns "ACYCLIC" if G does not contain any cycle. Prove the correctness of your algorithm.

Observation: the graph may have much more than $O(n)$ edges (it can have up to $\Omega(n^2)$ edges), so your algorithm cannot inspect all edges of the graph. Note also that your algorithm can only make at most $O(n)$ calls to the subroutines above.