1.25 Ordenar longituds variables

[CLRS capítol 8 «*Sorting in linear time*», problema 8.3]

1.25. Com ordenar eficientment elements de longitud variable:

(a) Donada una taula d'enters, on els enters emmagatzemats poden tenir diferent nombre de dígits. Però sabem que el nombre total de dígits sobre tots els enters de la matriu és n. Mostreu com ordenar la matriu en $O(n)$ passos.

(b) Se us proporciona una sèrie de cadenes de caràcters, on les diferents cadenes poden tenir diferent nombre de caràcters. Com en al cas previ, el nombre total de caràcters sobre totes les cadenes és n. Mostreu com ordenar les cadenes en ordre alfabètic fent servir $O(n)$ passos. (Tingueu en compte que l'ordre desitjat és l'ordre alfabètic estàndard, per exemple, $a < ab < b$.)

a. The usual, unadorned radix sort algorithm will not solve this problem in the required time bound. The number of passes, d, would have to be the number of digits in the largest integer. Suppose that there are m integers; we always have $m \le n$. In the worst case, we would have one integer with $n/2$ digits and $n/2$ integers with one digit each. We assume that the range of a single digit is constant. Therefore, we would have $d = n/2$ and $m = n/2 + 1$, and so the running time would be $\Theta(dm) = \Theta(n^2)$.

Let us assume without loss of generality that all the integers are positive and have no leading zeros. (If there are negative integers or 0, deal with the positive numbers, negative numbers, and 0 separately.) Under this assumption, we can observe that integers with more digits are always greater than integers with fewer digits. Thus, we can first sort the integers by number of digits (using counting sort), and then use radix sort to sort each group of integers with the same length. Noting that each integer has between 1 and n digits, let m_i be the number of integers with i digits, for $i = 1, 2, \ldots, n$. Since there are n digits altogether, we have $\sum_{i=1}^{n} i \cdot m_i = n$.

It takes $O(n)$ time to compute how many digits all the integers have and, once the numbers of digits have been computed, it takes $O(m + n) = O(n)$ time to group the integers by number of digits. To sort the group with m_i digits by radix sort takes $\Theta(i \cdot m_i)$ time. The time to sort all groups, therefore, is

$$\begin{aligned}\sum_{i=1}^{n} \Theta(i \cdot m_i) &= \Theta\left(\sum_{i=1}^{n} i \cdot m_i\right) \\ &= \Theta(n)\,.\end{aligned}$$

b. One way to solve this problem is by a radix sort from right to left. Since the strings have varying lengths, however, we have to pad out all strings that are shorter than the longest string. The padding is on the right end of the string, and it's with a special character that is lexicographically less than any other character (e.g., in C, the character `'\0'` with ASCII value 0). Of course, we

don't have to actually change any string; if we want to know the jth character of a string whose length is k, then if $j > k$, the jth character is the pad character.

Unfortunately, this scheme does not always run in the required time bound. Suppose that there are m strings and that the longest string has d characters. In the worst case, one string has $n/2$ characters and, before padding, $n/2$ strings have one character each. As in part (a), we would have $d = n/2$ and $m = n/2 + 1$. We still have to examine the pad characters in each pass of radix sort, even if we don't actually create them in the strings. Assuming that the range of a single character is constant, the running time of radix sort would be $\Theta(dm) = \Theta(n^2)$.

To solve the problem in $O(n)$ time, we use the property that, if the first letter of string x is lexicographically less that the first letter of string y, then x is lexicographically less than y, regardless of the lengths of the two strings. We take advantage of this property by sorting the strings on the first letter, using counting sort. We take an empty string as a special case and put it first. We gather together all strings with the same first letter as a group. Then we recurse, *within each group*, based on each string with the first letter removed.

The correctness of this algorithm is straightforward. Analyzing the running time is a bit trickier. Let us count the number of times that each string is sorted by a call of counting sort. Suppose that the ith string, s_i, has length l_i. Then s_i is sorted by at most $l_i + 1$ counting sorts. (The "+1" is because it may have to be sorted as an empty string at some point; for example, `ab` and `a` end up in the same group in the first pass and are then ordered based on `b` and the empty string in the second pass. The string `a` is sorted its length, 1, time plus one more time.) A call of counting sort on t strings takes $\Theta(t)$ time (remembering that the number of different characters on which we are sorting is a constant.) Thus, the total time for all calls of counting sort is

$$\begin{aligned} O\left(\sum_{i=1}^{m}(l_i + 1)\right) &= O\left(\sum_{i=1}^{m} l_i + m\right) \\ &= O(n + m) \\ &= O(n)\ , \end{aligned}$$

where the second line follows from $\sum_{i=1}^{m} l_i = n$, and the last line is because $m \le n$.