MC3305: Challenge 1

Due on Tuesday, June 9, 2015

Jesus P. Mena-Chalco

Rodrigo Martins de Oliveira

Problem 1

The challenge involved developing an algorithm that is able to sort an array of integers in any possible base using Radix Sort, without imposing limits to the base order, array length or number of digits of the array items.

The approach chosen here was to work with the binary number stored in system memory for the array items and cast get their representation in base 10. It is worth to mention that I believe that working directly with base 2 is better, but as the time for submission falls short I just implemented it with base 10 because I can ensure everything will be working as I learned Radix Sort in base 10 primary.

The sorting algorithm by itself is very straight as it follows the most basic definition of Radix Sort: taking an array of n elements the algorithm takes passes sorting just a certain significant position digit at each pass. Once all significant positions are sorted the algorithm finishes execution.

On the sorting by significant positions there are two traditional ways that Radix Sort can operate that: one is by starting by ordering the least significant digit, which is most used because the majority of integers representation today is little endian; the other is by starting from the most significant digit, which is specially useful when dealing with strings storing integers in a given base for obvious reasons.

The real challenge on the development of this algorithm was making it work for any base. As it is easy to think about base 2, base 8, base 10, base 16, base 26 or base 36, for example, when it comes to working with bases like 1000, 10000, etc. it is not obvious the approach one should use. Here I will describe my approach on that and then give a quick overview of the capabilities of my algorithm and how to use it.

In order to work with high order bases I took a step on virtual digit representation, i.e. representing one digit as a composition of other symbols. This decision solves the problem of running out of known symbols to represent new digits of high order bases, so now the user can virtually represent any radix.

As the above implies that for certain bases digits will be a composition of others, it is possible to constraint the set of primary symbols to the decimal digits (or even 0's and 1's), but for user convenience the algorithm permits the use of any printable character of ASCII table to represent a digit as long the "weight" of a given symbol corresponds to its number in ASCII table, so non traditional bases where $'x' < 'a'$ are not supported.

In order to receive an array of any data type, be it int, char, string, etc. the function works with a void pointer to get the array to be sorted. As it is permitted to use compositions of symbols to create new ones the passed array can be non linear, i.e. 2D arrays in case of strings (each element is a char array), 3D arrays, etc. To handle that it must be passed to the Radix Sort function what is called depth of the array, which is its dimension - 1. So an array of strings, which has dimension 2, will have a depth of 1 and an array of ints has depth 0.

Problem 1 continued on next page...

In order to know the length of the sub-arrays inside the main one an long unsigned int array m has to be passed to the function, its length must be equal to the depth of the array to be sorted and the number in a given index gives the length of the array at the same depth as the index on m. Also the algorithm doesn't impose any maximum depth, so the number of digits can be as big as the environment allows, but the algorithm works only with fixed length strings as the due date did not permit a more careful coding, so to represent numbers with fewer digits than the string length chosen in m the user has to fill the first positions of the string with zeros.

Also to permit working with both little endian and big endian items a flag *big_endian* must be passed to the function.

In order to facilitate usage of the function a macro, in association with a struct and a wrapper function was defined to permit easy calls for sorting one dimensional arrays.

The function also need to be supplied a function to get the maximum element of the array. Using an user supplied function was a decision to make the algorithm more flexible and independent (but it is questionable if it is really useful).

When dealing with multi-dimensional arrays the algorithm uses of recursive calls to access the primitive symbols

Said that, the developed algorithm can virtually sort an array of any size, with any number of digits per item, in any base. A known limitation of the algorithm is that for multi-dimensional arrays the primitive type they are based on must be chars or strings (which are arrays of chars in fact), this limitation is due to deadline falling too short...

To use this algorithm please read the source code which is better documented and comes with working examples of usage. But basically the user can call for radixSort() or radixSort_base() to use the algorithm and just pass the required parameters.

OBS.: This report is lacking content and polish, but making it better would require more time than I dispose.