ECE-250 – Algorithms and Data Structures (Winter 2012)

Tutorial 3 (2012-02-02)

1 - Playing with binary arithmetic: Write (reasonably efficient) C++ functions operating on `unsigned int` values that:

(a) Check if a given number is a power of 2.
(b) Round a number up to the next power of 2 (e.g., values between 17 and 32 would be rounded to 32).
(c) Multiply times 34 (recall that multiplying times 2 or times a power of 2 is fast; also, increasing a value by 1 is fast — more so than adding any other arbitrary value).
(d) Swap the lower and upper bytes in a 16-bit value — that is, produce a value that is the given value with the swapped bytes (you may assume either an `unsigned short int` / `uint16_t` value, or simply assume that the value stored in an `unsigned int` is a 16-bit value).
(e) Reverse the bits.
(f) Given four `unsigned int` values in an array, obtain a single value that contains, at the corresponding position of the byte, the lower byte of the value at position 0, the second lower byte of the value at position 1, and so on for the four values.

Example: if the values are {`0x11111180, 0x22224022, 0x33203333, 0x10444444`}, the function should produce `0x10204080` as the result.

2 - Hash tables – linear probing: We want to store values between 0 and 9999 in a hash table of size 10.

The hash function operates as follows: given a value x, add the four digits of x and take the last (right-most) digit. For example, the hash of 3276 would be 8 $(3 + 2 + 7 + 6 = 18)$.

(a) Insert, in the given order, the values 3836, 7209, 2373, 9412, 6950, 471, 5569, 9703.

(b) Then, remove, in the given order, 3836, 9412, and 5569.

3 - Hash tables – double hashing (you may need a calculator for this one [1]): with a hash table of size 8 to store values between 0 and 9999, we use the following hash functions: given a value $x = d_3d_2d_1d_0$, we compute the value $(d_3 + 1) \times (d_2 + 1) \times (d_1 + 1) \times (d_0 + 1)$. This result takes no more than 13 or 14 bits (right? *why?*). So, the primary hash function (the one to determine the bin) is the value $b_4b_3b_2$, where b_n is the bit n of the result, with the convention that b_0 is the least-significant bit. The jump size (the secondary hash function) is given by the value b_6b_51.

(a) Write C++ functions (as efficient as possible) implementing these hash functions — you want to avoid repeating the product of the digits. You may assume that the given value is between 0 and 9999.

(b) Insert, in the given order, the values 3836, 7209, 2373, 9412, 6950.

[1] If you don't, good for you!