Tutorial 2: Lists and Hashing

Please note that the questions on hashing will need to be delayed for a later tutorial. It is given here so that you have the chance to look at it in time for the next ADS Thread tutorial.

1. (a) Give pseudo-code for an algorithm that reverses a singly linked list. The algorithm should run in $\Theta(n)$ time for any input that has $n$ items with the exception of the empty list when it runs in time $\Theta(1)$. (Note that as $n = 0$ for the empty list it would be incorrect to claim a runtime of $\Theta(n)$ here, as this is just 0 and of course the algorithm does take a small amount of time no matter what the input. We could avoid the special case by expressing the runtime as $\Theta(n+1)$ for all $n$, this is just a matter of taste.)

(b) Suppose we have an application that works with lists and needs to reverse them frequently. Explain how you would implement lists in such a way that reversing them takes $\Theta(1)$ time. What price do you pay for this improvement?

2. (a) Consider the expression

$$a \ b - c \times d \ a + /$$

given in reverse Polish notation (i.e., we write the arguments of a function first and then the name of the function, so we write $3 \ d$ – rather than $3 \ a \ -$). This has the advantage that there is no need for brackets. Use a stack to produce an equivalent form of this expression in standard bracketed form.

(b) Suppose we have an expression consisting of $n$ symbols altogether (counting each argument and each operand as one symbol every time it occurs). Is it correct to claim that we use $O(n)$ stack operations to evaluate the expression? You may assume that each operator is one of $+,-, \times$ or $/$ (though pretty much the same reasoning applies for arbitrary operators). Justify your answer.

(c) Suppose the input expression is well formed can you improve the estimate for the number of stack operations to a $\Theta(\cdot)$ one? (An expression is well formed if there is no error during processing, by trying to do a pop operation on the empty stack, and at the end the stack has exactly one element.)

3. Let $H$ be a bucket-array hash table with hash function $h(n) = (2n+5) \mod 11$. Draw the contents of $H$ after inserting the following keys:


4. This question is a little harder than the others but does represent a realistic scenario in the use of hashing. In fact the solutions are quite straightforward.

Suppose we wish to maintain a dictionary of everyday English words using a hash table, with collisions resolved by chaining. One possible hash function is

$$h(w) = \text{value}(w) \mod m,$$

where $m$ is the size of the hash table and $\text{value}(w)$ is the integer value of the character string $w$ interpreted as a radix-128 integer, i.e., if $w = w_0 w_1 \ldots w_{w-1}$, then $\text{value}(w) = \sum_{i=0}^{w-1} \text{ord}(w_i) \times 128^{w-i-1}$, where $\text{ord}(w_i)$ is the ASCII value of $w_i$.

(a) For any word $w$, the hash value $h(w)$ is less than the table size $m$ and so can be represented using the same number of bits (6, say) as $m$ itself. The quantity $\text{value}(w)$, however, will typically be so large as to cause integer overflow, even for short words $w$. Suggest a way of computing $h$ which, for any input string $w$, requires as storage only $b$-bit integers, where $b$ exceeds $b$ by only a small additive constant.

(b) If a table size of around 50000 were required, we might choose $m = 2^{16}$, which is convenient from the point of view of performing ‘mod’ operations. Explain why this is a poor choice in the current application, and suggest an alternative which is likely to improve the performance of the hash function.