CS / MCS 401 Homework 6 grader solutions

assignment due July 25, 2016
written by Jānis Lazovskis
maximum points: 14

Some questions from CLRS. Questions marked with an asterisk $*$ were not graded.

1. *In section 11.3.2 of the textbook (page 263-264) the multiplication method is given by:*

$$h(k) = \lfloor m(kA \mod 1) \rfloor \quad \textit{with} \quad 0 < A < 1. \tag{1}$$

The CLRS slides for hash functions (used in lectures) describe the multiplication method for the special case where the table size is an integer power of 2. The hash function is given as: Assume all keys are integers, $m = 2^r$, and our computer has w bit words. Then

$$h(k) = (kA' \mod 2^w) \cdot \text{rsh}(w - r) \quad \textit{with} \quad 2^{w-1} < A' < 2^w. \tag{2}$$

Show that version (1) *is the same as version* (2) *under the given conditions.*

Solution: To show that (1) is the same as (2), we use notation from the book and slides, with $s = A2^w$ being a w-bit integer and rsh the "bitwise right-shift" operation (recall that shifting to the left by 1 bit is multiplication by 2 and shifting to the right by 1 bit is division by 2 (rounding down to the nearest integer)). Also recall that "$a = b \mod c$" is equivalent to saying "there exists an integer d such that $a = cd + b$." Hence we may equivalently say

$$\begin{aligned} x = kA \mod 1 \quad &\iff \quad \text{there exists an integer } d \text{ such that } x = d + kA \\ &\iff \quad \text{there exists an integer } d \text{ such that } 2^w x = 2^w d + 2^w kA \\ &\iff \quad 2^w x = 2^w kA \mod 2^w. \end{aligned}$$

Since $0 < A < 1$, we have that $0 < A2^w < 2^w$, which is not quite what we want, for $A' = A2^w$. Hence we must choose $1/2 < A < 1$, because $1/2 = 2^{-1}$, so then multiplying all sides by 2^w, we get $2^{w-1} < A2^w < 2^w$. This makes $A' = A2^w = s$ a fitting choice. Note that the book assumes that $s = A2^w$ is an integer, which we will have to use below. Now we have that

$$\begin{aligned} h(k) &= \lfloor m(kA \mod 1) \rfloor && \text{(given)} \\ &= \lfloor 2^r (kA \mod 1) \rfloor && \text{(assumption)} \\ &= \lfloor 2^r \tfrac{1}{2^w} (2^w kA \mod 2^w) \rfloor && \text{(equivalences above)} \\ &= \lfloor 2^{r-w} (ks \mod 2^w) \rfloor && \text{(definition)} \\ &= \lfloor 2^{r-w} \rfloor (kA' \mod 2^w) && \text{(since } k, s, w \text{ are integers)} \\ &= \text{rsh}(w - r) \cdot (kA' \mod 2^w). && \text{(definition)} \end{aligned}$$

Therefore the two definitions of $h(k)$ are the same. ■

2. *Are there any advantages to using the second version? Explain.*

Solution: The second version uses the bitwise right-shift operation instead of taking the floor of a number. On a computer operations in base-2 are faster than operations in base-10, so the right-shift operation makes the seond approach faster. ■

3. *Donald Knuth, who has been called the "father of the analysis of algorithms," suggests that* $A = (\sqrt{5}-1)/2 \approx 0.6180339887\ldots$ *is likely to work reasonably well in version* (1). *Do Exercise 11.3-4 assuming version* (1).

11.3-4 *(p.269)* *Consider a hash table of size* $m = 1000$ *and a corresponding hash function* $h(k) = \lfloor m(kA \bmod 1) \rfloor$ *for* $A = (\sqrt{5}-1)/2$. *Compute the locations to which the keys* $61, 62, 63, 64,$ *and* 65 *are mapped.*

Solution: This question involves computing $h(k)$ for the given values of k, which is more a computational question than a mathematical question. Below are some (not all, and not the best) ways of implementing this function in two common (mathematical) languages:

Mathematica:
```
h[k_] := Floor[1000*Mod[k*(Sqrt[5]-1)/2, 1]]
```

Sage / Python:
```
import math
def h(k):
    return math.floor(1000*(k*(math.sqrt(5)-1)/2 % 1))
```

With this, we can easily find the desired hash values to be as below.

k	61	62	63	64	65
$h(k)$	700	318	936	554	172

■

4. *Repeat Exercise 11.3-4 using a table size that is the next power of 2 greater than 1000.*

Solution: Note that $\log_2(1000) \approx 9.966$, so 10 is the next power of 2 greater than 1000. Define the function $H(k)$ which is just $h(k)$ but with 1024 instead of 1000. Repeating the computations for the same inputs gives the hash values as below.

k	61	62	63	64	65
$H(k)$	716	325	958	567	176

■

5. *Give a function definition for the previous problem that implements*

$$h(k) = (kA \mod 2^w) \cdot \text{rsh}(w-r) \quad \textit{with} \quad 2^{w-1} < A < 2^w.$$

The values of $A, w,$ *and* r *must be defined in your function definition. Note, the examples used in class notes the hash functions were coded in the C programming language. Many of the hash functions posted on the internet are also given as C code. C code is often used as pseudocode in textbooks. Define your function in C. Do not use Java. If you do not understand C code then use pseudocode similar to that used in the textbook.*

Solution: A solution in pseudo-code to question 4 above, on a 64-bit computer, is given below.

```
1  define BASE(n, b) :
2      return base-b representation of n
3
4  define h(k) :
5      w = 64
6      r = 10
7      A = 2^w · (√5 − 1)/2
8      ans = k · A  mod 2^w
9      ans = (w − r bits removed on the right of BASE(ans, 2))
10     return BASE(ans, 10)
```

■

∗ 6. *In the previous problem k is an integer. Modify your function so the input is a key represented as a string. Your hash function should first convert the string to an int using Horners Method and then hash the resulting integer to an array index.*

7. *Using the division method define a hash function that will only use the lower 8 bits of the key. All of the higher bits will be ignored resulting in a hash function that does not use all of the information available in the key. This is not a good hash function but easy to code.*

Solution: A solution in pseudocode is given below.

```
1  define h(k) :
2      return k  mod 2^8
```

■

8. *Heapsort. On page 155 of the textbook it states that*

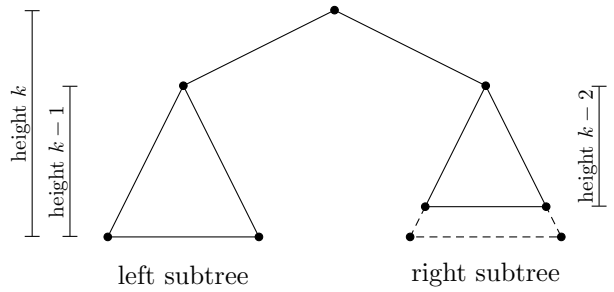
   (a) *"The chldren's subtrees each have size at most $2n/3$"*

   (b) *"the worst case occurs when the bottom level of the tree is exactly half full"*

   (c) *"therefore we can describe the running time of* MAX-HEAPIFY *by the recurrence $T(n) \leqslant T(2n/3) + \Theta(1)$"*

   Derive (a), show (b), (c) and finally solve (c).

Solution: Suppose the tree has size n. If the last line were completely full, then $n = 2^{k+1} - 1$, for k the height of the tree (the number of edges from the top to the deepest leaf). By the construction of the tree, it is immediate that the left subtree of any node is always larger than or equal in size to the right subtree. Hence to find an upper bound on the size of the subtrees, we only need to consider the size of the left subtree. Proportionally to the size of the whole tree, the left subtree will have the most nodes when its last line is full but the last line of the right subtree is completely empty (as in the diagram below).



This proves (b). To show this upper bound, observe that when the worst case happens,

$$\frac{\text{size of left subtree}}{\text{size of tree}} = \frac{2^k - 1}{2^{k+1} - 1 - 2^k/2} = \frac{2^{k+1} - 2}{2^{k+2} - 2^k - 2}. \tag{3}$$

This is not enlightening, but if we consider the limiting case as $k \to \infty$, we find that

$$\lim_{\text{size}\to\infty} \left[\frac{\text{size of left subtree}}{\text{size of tree}}\right] = \lim_{k\to\infty} \left[\frac{2^{k+1} - 2}{2^{k+2} - 2^k - 2}\right] = \lim_{k\to\infty} \left[\frac{2 - \frac{1}{2^{k-1}}}{4 - 1 - \frac{1}{2^{k-1}}}\right] = \frac{2}{3}.$$

Moreover, the limit is approached from below, because if we add nodes to the right subtree (when the left is full), then the denominator of (3) increases, making the expression smaller. If we take away nodes from the bottom of the left subtree, we are subtracting the same amount from the numerator and the denominator, and since the denominator is strictly larger, the value of the whole expression also decreases. This proves that the left subtree has at most $2/3$ the nodes of the whole tree. In other words, we have shown that the left subtree has size at most $2n/3$, proving (a).

By the second case of the master method, with $a = 1$ and $b = 3/2$, we see that $T(n) \leqslant \Theta(\log_2(n))$, implying that $T(n) = O(\log_2(n))$, giving part (c). ■