Foundations of computing: Algorithms and Data Structures

Rasmus Pagh

September 29, 2011

1 Exercises on search trees

1. It is possible to sort by doing n insertions into a balanced binary search tree. What is the total (asymptotic) cost of these insertions?

2. One way of augmenting search trees (with integer keys) is to maintain in each node the sum of all keys in the subtree below. Another is simply to maintain the number of keys in each subtree. Now suppose we are interested in the average value, e.g., "What is the average salary among those who make more than x kr. a year?" How would you extend a search tree (with salaries as keys) to accommodate queries like this? What would be the worst-case complexity of a query?

2 Exercises on hashing

1. Give an algorithm that takes an input of n words (each of length at most 50) and outputs the k most frequent words (breaking ties arbitrarily). Analyze your algorithm in terms of n and k. (There exists an algorithm running in expected time $O(n)$.)

2. Java string objects are by default hashed by the following algorithm that produces 32 bit integers.

```
public static int hashCode(String input) {
    int h = 0;
    int len = input.length();
    for (int i = 0; i < len; i++) {
        h = 31 * h + input.charAt(i); // Only 32 least significant bits stored
    }
    return h;
}
```

(a) What is the running time of the algorithm, when used on a string of length n?

(b) The two strings `"ABCDEa123abc"` and `"ABCDFB123abc"` have the same `hashCode()` value. What are the implications for the hashing methods we discussed: Chaining, linear probing, and cuckoo hashing?

(c) How could the choice of hash function possibly be randomized, to avoid sure collisions like the one above? Can you identify a pitfall that might result in only *even* hash values?

3. *Oyster of the week:* Show that if n is large and we hash n keys to a hash table of size n it is unlikely that there will exist $\ell = \lceil \log n \rceil$ items that hash to the same bucket. You should assume that the hash function is fully random. **Hint:** There are less than $(ne/\ell)^\ell$ sets of ℓ keys. What is the probability that one *particular* set of ℓ keys will hash into one bucket?

2.1 Programming exercises

1. Write a program that reads a text file, and counts the number of different words in the text. We define a word to be a sequence of consecutive alphabetic characters, i.e., c satisfying $'a' \leq c \leq 'z'$ or $'A' \leq c \leq 'Z'$. Use a hashMap (in Java) to keep track of what words have been counted. You may test your program on the King James bible, which can be downloaded at `http://www.itu.dk/courses/SPT/E2008/bible.txt`. The actual text starts after the line containing the string:

   `*END*THE SMALL PRINT! FOR PUBLIC DOMAIN ETEXTS*Ver.04.29.93*END*`

2. Modify the program above to count the number of occurrences of each word, and output the k most frequent words. Use your solution from the theoretical question above, or a simpler alternative.

3. Implement cuckoo hashing, where each hash table entry should contain a String (the key) and an integer. You only need to implement this for a fixed-size hash table (i.e., r should be defined as a constant in the program), and you do not need to implement rehashing. Use the built-in hash funtion `hashCode`, which is defined for all objects, to extract two hash values:

   - `hashCode % r` (modulo r) and
   - (`hashCode() / r) % r` (integer division by r followed by a modulo r).

   This can only work with hash table size up to around 2^{16}*!* (Why?)