1999

p 195
LCP

Foundations of Computer Science (Paper 1)

We face the Year 2000 crisis because programmers did not apply the principles of data abstraction. Discuss. [4 marks]

Your employer asks you to implement a dictionary. The pattern of usage will consist of taking an empty dictionary and performing many insertions and lookups. You must choose one of three data structures. Each requires $O(\log n)$ time for the lookup and update operations, where n is the number of items in the dictionary. They are (1) binary search trees, which take $O(\log n)$ time in the average case; (2) balanced trees, which need complicated algorithms but take $O(\log n)$ time in the worst case; (3) self-adjusting trees, which take $O(\log n)$ amortized time in the worst case.

Explain the differences between the three notions of $O(\log n)$ time. Argue that any of the three data structures might turn out to be the best, depending upon further details of the application. If no further details are available, which of the three is the safest choice? [8 marks]

An algorithm requires $T(n)$ units of space given an input of size n, where the function T satisfies the recurrence

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(n/2) + n \qquad (n > 1). \end{aligned}$$

Express the algorithm's space requirement using O-notation, carefully justifying your answer. [8 marks]

Model Answer

(Much of this is covered in the notes.) Everything in a computer is represented ultimately as zeros and ones. Data abstraction means thinking about the map from abstraction objects, such as dates, to lower-level data, such as bits. The representation must take into account the operations that will be performed on the data. The programmer should make the operations available explicitly, e.g. as functions, rather than sprinkling low-level data and operations throughout the code.

Data abstraction does not guarantee that data will be represented perfectly, because that is not possible: computers are finite. In the case of dates, a programmer in the 1960s could reasonably decide to represent dates in a way that would not work beyond 1999. Early programming languages provided no support for data abstraction, making the programmer's task difficult (COBOL still does not even provide functions). If abstract data types were available, then changing programs would be quite easy, but there would still be the task of converting stored files from one representation to another.

Regarding the three data structures. (1) An *average case* cost refers to the cost of one operation averaged over all possible arguments to that operation. The worst case for binary search trees is terrible: $O(n)$. (2) The *worst case* cost refers to the worst possible cost of one operation considering all possible arguments. But since complicated algorithms are involved, we can expect the constant factor to be higher than that of binary search trees.

(3) Amortized cost refers to the average cost over a batch of operations, rather than one operation; the worst case amortized cost is the worst such average for all possible batches. A batch refers to a string of operations that could validly follow one another, e.g. the updates and lookups described in the question. The cost of any one operation may be high, but the average over the batch will be $O(\log n)$.

Data structure (1) is best if the worst case is highly unlikely and its constant factor is low. If the worst case is likely, then (3) is best if the cost of individual operations is unimportant and (2) is best otherwise, for example in a real-time application. So data structure (2) is the safest choice, guaranteeing that each operation is reasonably fast, provided we can live with the constant factor.

The function $T(n)$ is linear in n. One way to prove this is to consider the special case $n = 2^k$. A straightforward mathematical induction proves $T(2^k) = 2^{k+1} - 1$, which with a bit of handwaving yields the result that T is linear.

More convincing is to prove $T(n) \leq 2n$ by induction. For $n = 1$ we have $T(1) = 1 \leq 2 \times 1$. For $n > 1$ we have $T(n) = T(n/2) + n \leq 2(n/2) + n = 2n$; we can utilize the induction hypothesis because $n/2 < n$.

The course notes use $n/2$ to denote integer division; the pedantic notation $\lfloor n/2 \rfloor$ should not be necessary.