Assignment 16, Data-Structures

Oleg Sivokon

<2016-05-25 Wed>

Contents

1 Problems

1.1 Problem 1

1. Given a hash-table with chaining of initial capacity m. What is the probability four elements inserted will end up in the same bucket?
2. Given a hash-table with open addressing and elements k_1, k_2 and k_3 inserted in that order, what is the chance of performing three checks when inserting the third element?
3. Given hash-table s.t. its density is $1-\frac{1}{\lg n}$. Provided the table uses open addressing, what is the expected time of failed search as a function of n?

1.1.1 Answer 1

Our simplifying assumption is that we draw hashing functions at random from a universe of hashing functions allows us to say that a probability of a key being hashed to a slot in the table of m slots is $\frac{1}{m}$. Using product law we can conclude that the probability of four keys being mapped to the same slot is $\frac{1}{m} \times \frac{1}{m} \times \frac{1}{m} = \frac{1}{m^3}$.

1.1.2 Answer 2

Using the same simplifying assumption as before, we see that for the element k_3 to be places only after three checks it has to first collide with k_1 and then with k_2. Using product law gives the probability of $\frac{1}{m} \times \frac{1}{m} = \frac{1}{m^2}$.

1.1.3 Answer 3

Recall that the average time needed for failed search in a hash table with open addressing is $\frac{1}{1-\alpha}$. Substituting $1 - \frac{1}{\lg n}$ in place of α obtains:

$$\begin{aligned}\frac{1}{1-\alpha} &= \frac{1}{1-\frac{1}{\lg n}} \\ &= \frac{\lg n}{\lg n - 1}\end{aligned}$$

1.2 Problem 2

Given a set of rational numbers S and a rational number z,

1. write an algorithm that finds two distinct summands of z with running time $\Theta(n)$.
2. Same as in (2), but for four summands and time $\Theta(n^2)$.

1.2.1 Answer 4

1.2.2 Answer 5

1.3 Problem 3

Given a binary search tree with n nodes there are $n + 1$ *left* and *right* nil-pointers. After performing the following on this tree: If `left[z]` = `nil`, then `left[z]` = `tree-predecessor(z)`, and if `right[z]` = `nil`, then `right[z]` = `tree-cussessor(z)`. The tree built in this way is called "frying pan" (WTF?), and the arcs are called "threads".

1. How can one distinguish between actual arcs and "threads"?
2. Write procedures for inserting and removing elements from this tree.
3. What is the benefit of using "threads"?

1.3.1 Answer 6

Search tree invariant implies that left pointer must point at a node with a value less than the node holding the pointer, but predecessor would have a value larger than the node holding the pointer. The situation for right node is symmetrical.

1.3.2 Answer 7

1.3.3 Answer 8

None what so ever. Whoever wrote this question is a brainless moron, who has no idea of how computers work. He seems to believe that it matters whether the node stores a null pointer or a pointer to some other node in terms of amount of memory used, which is absolute bullshit.

1.4 Problem 4

Given array $A[1 \dots n]$ s.t.

$$\begin{aligned} A[1] &> \cdots > A[p] \\ A[p+1] &> \cdots > A[q] \\ A[q+1] &> \cdots > A[n] \\ A[1] &< A[q] \\ A[p+1] &< A[n] \end{aligned}$$

insert it into binary tree.

1. What is the height of the resulting tree?
2. Erase $A[p+1]$ and insert it anew: how will the height and the shape of the tree change?

1.4.1 Answer 9

1.4.2 Answer 10