Assignment 5

Fu Guanshujie, Ge Yuhao, Lou Haina, Qiu Xiaomin

March 2021

1

Next page.

2

See the code.

3

Assume the Bernoulli-distributed random variables $X_0, \cdots, X_{m-1}$ represent the possibility of the entries to be empty, $X_i = 1$ if $A[i]$ is empty.

As the hash function is random, it maps e to each $h(e) \in \{0, \cdots, m-1\}$ with the same probability, independent of $h(k)$, i.e. $P(h(e) = h(k)) = \frac{1}{m}$ and $E(X) = \frac{n}{m}$.

For $n > m$, assume there are k empty entries, there might be C_m^k kinds of situations. And in each case, the possibility is about $(\frac{m-k}{m})^n$. Then the possibility of finding k empty entries is

$$P(k, m, n) = \frac{C_m^k \, (m-k)^n}{m^n}$$

Then we could determine the expected number of empty entries in the hash table by equation:

$$E(m, n) = \frac{\sum_{k=1}^{m-1} \, k \, C_m^k \, (m-k)^n}{m^m} \tag{1}$$

4

See the code.

(i) when the new value doesn't break the rules, only need to change the value. which need constant time in $O(1)$

when the new value breaks the rules:

① when it is greater than it's children. Cut all it's children to the root list and cut itself to the root list with new value. The parent of this node should be marked, if it was unmarked or it is also cut ont. unmarked and inserted into the root list. This is cascaded upwards. Adjust H.min if needed

$\Phi(H) = t(H) + 2m(H)$

$\Phi(H') = (t(H) + \underbrace{D(n)}_{\text{it's children}} + \underbrace{1}_{\text{itself}} + \underbrace{C}_{\text{cut marked}}) + 2(m(H) - C + 2)$

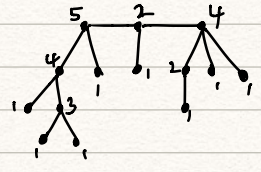
$a\Phi = D(n) + 1 + C - 2C + 4 = D(n) - C + 5 \leq \log n - C + 5$

Operations like cut. insert increase need constant time C.

So amortised time complexity is in $O(\log n)$

② when it is smaller than its parent, the procedure is the same as decrease which is in $O(1)$

(ii) we can add another fild in the class "tot"; which stores the total number of nodes below this node. For example:



when we need to prune r nodes. We find the node on root that has the largest "tot" that is smaller or equal to r, detete this node with all the children and $r = r - tot$. loop this procedure till there is no such root.

Then go to one of the remaining root that has largest "tot" and consider its children as new root list and redo the above procudure until $r = 0$.

As the complexity is porpotional to the length of root, the amortised time complexity is in $O(\log n)$