Lab04-Hashing

VE281 - Data Structures and Algorithms, Xiaofeng Gao, TA: Qingmin Liu, Autumn 2019

∗ Please upload your assignment to website. Contact webmaster for any questions.

∗ Name:Jin Zhejian Student ID:517370910167 Email: jinzhejian@outlook.com

1. Given a sequence of inputs 192, 42, 142, 56, 39, 319, 14, insert them into a hash table of size 10. Suppose that the hash function is $h(x) = x\%10$. Show the result for the following implementations:

   (a) Hash table using separate chaining. Assume that the insertion is always at the beginning of each linked list.

   (b) Hash table using linear probing.

   (c) Hash table using quadratic probing.

   (d) Hash table using double hashing, with the second hash function as $h_2(x) = (x + 4)\%7$.

   Solution.

   (a) Hash table using separate chaining:

   [0] [ ]
   [1] [ ]
   [2] [ ] → [142] → [42] → [192]
   [3] [ ]
   [4] [ ] → [14]
   [5] [ ]
   [6] [ ] → [56]
   [7] [ ]
   [8] [ ]
   [9] [ ] → [319]→[39]

   (b) Hash table using linear probing:

319		192	42	142	14	56			39

   (c) Hash table using quadratic probing:

319		192	42	14		142	56		39

   (d) Hash table using double hashing, $h_2(x) = (x + 4)\%7$:

56	319	192		14		42		142	39

□

2. Show the result of rehashing the four hash tables in the Problem 1. Rehash using a new table size of 14, and a new hash function $h(x) = x\%14$. (Hint: The order in rehashing depends on the order stored in the old hash table, not on their initial inserting order.)

Solution.

(a) Hash table using separate chaining:
[0] [ ] → [56] → [14] → [42]
[1] [ ]
[2] [ ] → [142]
[3] [ ]
[4] [ ]
[5] [ ]
[6] [ ]
[7] [ ]
[8] [ ]
[9] [ ]
[10] [ ] → [192]
[11] [ ]→ [39] → [319]
[12] [ ]
[13] [ ]

(b) Hash table using linear probing:

42	14	142	56						192	319	39	

(c) Hash table using quadratic probing:

42	14	142		56					192	319	39	

(d) Hash table using double hashing, $h_2(x) = (x + 4)\%7$:

56		142		14				42		192	319	39	

□

3. Suppose we want to design a hash table containing at most 900 elements using linear probing. We require that an unsuccessful search needs no more than 8.5 compares and a successful search needs no more than 3 compares on average. Please determine a proper hash table size.

Solution.

$$U(L) = \frac{1}{2}\left[1 + \left(\frac{1}{1-L}\right)^2\right] \leq 8.5 \Rightarrow L \leq \frac{3}{5}$$
$$S(L) = \frac{1}{2}\left[1 + \frac{1}{1-L}\right] \leq 10 \Rightarrow L \leq \frac{18}{19}$$

Therefore, $L = \dfrac{|s|}{n} \leq \dfrac{3}{4} \Rightarrow n \geq \dfrac{4}{3} \cdot 900 = 1200$

We pick n as a prime number, i.e. $n = 1201$.

□

4. Implement queues with two stacks. We know that stacks are first in last out (FILO) and queues are first in first out (FIFO). We can implement queues with two stacks. The method is as follows:

- For **enqueue** operation, push the element into stack S_1.
- For **dequeue** operation, there are two cases:
  - $S_2 = \emptyset$**,** pop all elements in S_1, push these elements into S_2, pop S_2
  - $S_2 \neq \emptyset$**,** pop S_2

Using amortized analysis to calculate the complexity of **enqueue** and **dequeue** step.

Solution.
Aggregate analysis:

First, we calculate the cost of each operation, assuming that the cost for one **pop** or **push** operation is 1:

- For the **enqueue** operation,we push one element into S_1, so the cost is 1.
- For the **dequeue** operation, when $S_2 \neq \emptyset$, we only need to pop one element from S_2, so the cost is 1.
- For the **dequeue** operation, when $S_2 = \emptyset$, suppose there are n elements in S_1, we need to pop all of them from S_1, push all into S_2, and pop one element from the S_2, so, the cost is $2n + 1$.

Now, considering a sequence of n operations on two empty stack S_1, S_2. We can divide the problem into two cases:

1. S_1 has at least gone through 1 "all clear" operations, i.e. the **dequeue** when $S_2 = \emptyset$.
2. S_2 has never gone through the **dequeue** operation when $S_2 = \emptyset$.

Obviously, case 2 needs fewer costs than case 1. Also, case 1 happens rarely, so, by amortized analysis, we only need to calculate the total costs in case 1.

Assuming S_2 has done k times **dequeue** when $S_2 = \emptyset$, therefore, S_1 has been cleared (all pop out) for k times. For each state from S_1 is empty to S_1 is empty again, we denote it as Bucket i, with operation numbers n_i. For the state from the last time S_2 is empty to the last operation, it is denoted as remaining bucket with n_{rest} operations. The partition is shown in the following graph.

n_i	.....	n_i	.....	n_k	n_{rest}
Bucket 1	.....	Bucket i	.....	Bucket k	remaining Bucket

We have:

$$n = \sum_{i=1}^{k} n_i + n_{rest}$$

In bucket i, we can easily calculate the complexity:

$$\begin{aligned} T(n_i) &= Cost_{\text{enqueue}} + Cost_{\text{dequeue when empty}} + Cost_{\text{dequeue when not empty}} \\ &= \frac{n_i}{2} + \frac{n_i}{2} - 1 + 2 \cdot \frac{n_i}{2} + 1 \\ &= 2n_i \end{aligned}$$

Then, we sum up $T(n_i)$ and get:

$$\begin{aligned} T(n) &= \sum_{i=1}^{k} T(n_i) + T(n_{rest}) \\ &= \sum_{i=1}^{k} 2n_i + n_{rest} \\ &\leq \sum_{i=1}^{k} 2n_i + 2n_{rest} \\ &= 2n \end{aligned}$$

Therefore, on average, the **enqueue** and **dequeue** operation takes $O(1)$ time.

□