CSC4120 Spring 2024 - Written Homework 5

Yohandi 120040025
Andrew Nathanael 120040007

March 13, 2024

Hashing

Problem 1.

For each of the proposed hash functions given, determine whether they can be implemented and used as actual hash functions for a hash table with N slots. Indicate the reason.

(a) $h(k) = 0$

(b) $h(k) = \text{random}(0, N)$ (where $\text{random}(0, N)$ gives a uniformly generated random number from 0 to $N - 1$).

(c) $h(k) = k \mod N$.

(d) $h(k) = k \mod (N - 2)$ (Assume $N \geq 3$).

(e) $h(k) = f(k)$ (where $f(N)$ is a deterministic function that runs in $\Theta(N^2)$ time and returns a value from 0 to $N - 1$).

(a) No. It only hashes to one spot; consequently, every item collides in the same slot. This results in a poor performance $\Theta(n)$ for search, insert, and delete operations.

(b) No. The function is non-deterministic; consequently, it is not possible to guarantee the same item retrieval from the hash table.

(c) Yes. The function is deterministic, easy to compute, and distributes the keys well.

(d) Yes. The function can be used with the same reason in part (d), though with two spots being wasted.

(e) Yes. However, the function is simply impractical to be applied because the $\Theta(N^2)$ performance is highly inefficient.

Problem 2.

Consider the hash function $h(k) = k \mod m$, where $m = 2^p - 1$ and k is a character string interpreted in radix (base) 2^p. Show that if we can derive string x from string y by permuting its characters, then x and y hash to the same value.

Let x and y be strings of length n, and denote the characters in x and y as $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_n$ such that each character represents a value in the range $[0, 2^p - 1]$, respectively. Notice that the sets $\{x_1, x_2, \ldots, x_n\}$ and $\{y_1, y_2, \ldots, y_n\}$ are identical as x is permutable from y (and vice versa).

$$
\begin{aligned}
h(x) &= \left(\sum_{i=1}^{n} x_i \cdot (2^p)^{i-1}\right) \mod m \\
&= \left(\sum_{i=1}^{n} x_i \cdot (2^p)^{i-1}\right) \mod (2^p - 1) \\
&= \left(\sum_{i=1}^{n} x_i \cdot (2^p \mod (2^p - 1))^{i-1}\right) \mod (2^p - 1) \\
&= \left(\sum_{i=1}^{n} x_i \cdot 1^{i-1}\right) \mod (2^p - 1) \\
&= \left(\sum_{i=1}^{n} x_i\right) \mod (2^p - 1)
\end{aligned}
$$

Similarly, $h(y)$ is $\sum_{i=1}^{n} y_i \mod (2^p - 1)$. As the sets $\{x_1, x_2, \ldots, x_n\}$ and $\{y_1, y_2, \ldots, y_n\}$ are identical, so are the summations. This implies that:

$$
\begin{aligned}
h(x) &= \left(\sum_{i=1}^{n} x_i\right) \mod (2^p - 1) \\
&= \left(\sum_{i=1}^{n} y_i\right) \mod (2^p - 1) \\
&= h(y)
\end{aligned}
$$

This shows that x and y hash to the same value by function $h(x)$ given that x can be derived by permuting characters from y.

Problem 3.

Comparing open addressing to traditional chaining implementations, are the following statements true under the same loading factor? Indicate the reason.

(a) In open addressing it takes on the average less time to do an unsuccessful search.

(b) In open addressing insertions and deletions are faster.

(c) In open addressing we save storage space since we don't need pointers, so we can, for the same storage cost, build larger hash tables.

(d) Collisions (i.e., that when we insert a key we don't find an empty slot) are equally probable (under same loading factor, assuming the uniform hashing assumption)

(a) False. Unsuccessful search is calculated by $\frac{1}{1-\alpha}$, while the constraint of α is only set to be smaller than 1, the average time takes by open addressing requires huge steps when α is getting closer to 1. Unlike the traditional chaining implementation that takes only $\mathcal{O}(\alpha)$ steps.

(b) False. The same number of probes has been proved in part (a); for the traditional chaining implementation, insertion only takes $\mathcal{O}(1)$ as it can be append to the end of the chain.

(c) True. In open addressing, each slot directly contains the key-value pair, and no pointers are necessary since the probing sequence determines where to find or place entries. Therefore, the storage for each entry is just s bytes. The total storage required for n entries is simply ns bytes (assuming perfect utilization). However, if the average size of a key-value pair is s bytes and the size of a pointer is p bytes, the total storage for an entry in a chaining scheme is $s + p$ bytes. If we have n entries, the simplistic total storage needed would be $n(s + p)$ bytes, not accounting for any overhead for list headers or empty list pointers in the table slots.

(d) False. In the traditional chaining implementation, the collision will only happen at most once as the added element proceeds into the chain immediately. In the open addressing, however, after probing into the next index, collision might still happen.

Problem 4.

Consider a hash table of size $m = \sqrt{n}$. Then under the uniform hashing assumption, as $n \to \infty$, the number of empty slots tends to (i) a constant, (ii) zero, or (iii) infinity? Why?

(b) If $m = cn$, $c > 0$, then as $n \to \infty$, the number of empty slots tends to (i) a constant, (ii) zero, or (iii) infinity? Why?

(c) If $m = n^a$, $a > 1$, then as $n \to \infty$, the number of empty slots tends to (i) a constant, (ii) zero, or (iii) infinity? Why?

(a) Zero. Each table is expected to consist of $\frac{n}{\sqrt{n}} = \sqrt{n}$ values and as $\sqrt{n}$ tends to ∞ when $n \to \infty$, it's getting further away from 0. Equivalently, it is unlikely to have any empty slot.

(b) Infinity. Consider the following derivation:

$$\lim_{n\to\infty} m(1 - \frac{1}{m})^n = \lim_{n\to\infty} cn(1 - \frac{1}{cn})^n = \lim_{n\to\infty} cne^{-\frac{1}{c}} = \lim_{n\to\infty} c'n$$

, for $c' = ce^{-\frac{1}{c}}$. The number of empty slots is going to infinity.

(c) Infinity. By pigeonhole principle, the number of empty slots is at least $n^a - n$. Consider the following derivation:

$$\lim_{n\to\infty} n^a - n \to \infty \text{ when } a > 1.$$

This derivation implies that the number of empty slots is going to infinity.

Problem 5.

Let H be an open address hash table, initialized with 4 slots. Assume that the load factor cannot exceed 0.75, and assume that re-hashing is done by doubling. During each insertion of a new element into H, suppose that the hash function used to generate a probe sequence is obtained via linear probing, based on the auxiliary hash function $h(k) = k \mod m$, where m equals the number of slots in the hash table. Draw the final open address hash table obtained after inserting into H the following integers 12, 43, 13, 88, 23, 26, 11, 5 in this given order.

Problem 6.

Collision Resolution Assume simple uniform hashing in the entire problem.

(a) Consider a hash table with m slots that uses chaining for collision resolution. The table is initially empty. What is the probability that, after 4 keys are inserted, there is a chain of size exactly 3?

(b) Consider a hash table with m slots that uses open addressing with linear probing. The table is initially empty. A key k_1 is inserted into the table, followed by key k_2, and then key k_3. What is the probability that the total number of probes while inserting these keys is at least 4?

(c) Suppose you have a hash table where the load-factor a is related to the number n of elements in the table by the following formula:

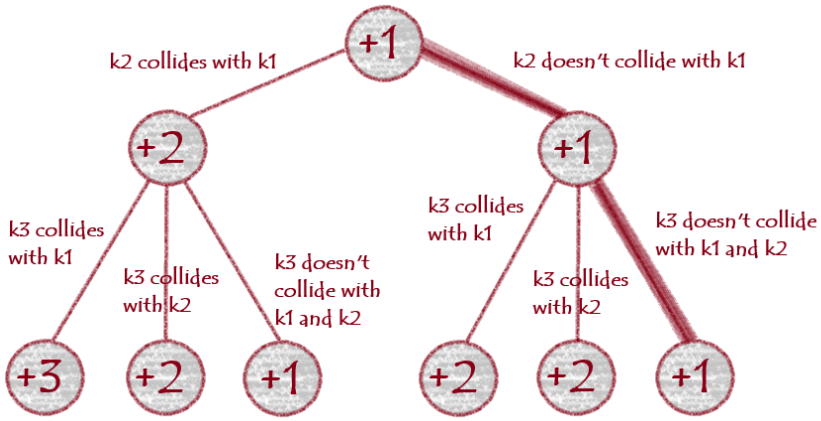
$$a = 1 - \frac{1}{\sqrt{n}}$$

If you resolve collisions by open addressing, what is the expected time for an unsuccessful search in terms of n?

(a) The probability is exactly the same with three out of the four keys being hashed to the same slot. Let the first three keys being hashed to the same slot, that is slot i, we have $\frac{1}{m^3}$. Then, we must have the fourth key to be hashed into different slot, we want to multiply our probability by $1 - \frac{1}{m} = \frac{m-1}{m}$. Now, we only need to get the number of arrangement to choose which key that should be hashed into different slot, there are $\binom{4}{3}$ arrangements. As there are m possible slots, we multiply our probability by m. Put all those together we have:

$$p = \binom{4}{3} \cdot m \cdot \frac{1}{m^3} \cdot \frac{(m-1)}{m} = \frac{4(m-1)}{m^3}$$

(b) The below figure shows all possibilities for the insertion.



It can be seen that the only possibility that results in less than 4 probes is when:

- k_2 doesn't collide with k_1, and
- k_3 doesn't collide with k_1 and k_2.

The first point has the probability of $\frac{m-1}{m}$ while the second point has the probability of $\frac{m-2}{m}$. So, the probability of inserting k_1, k_2, and k_3 such that the total number of probes is at least 4 is $1 - \frac{m-1}{m} \cdot \frac{m-2}{m} = \frac{3m-2}{m^2}$.

(c) For open addressing, the expected time for an unsuccessful search is equivalent to the average attempts, r, to find first empty slot. r is obtained as follows:

$$\begin{aligned} r &\leq \frac{1}{1-\alpha} \\ &\leq \frac{1}{1-(1-\frac{1}{\sqrt{n}})} \\ &\leq \frac{1}{\frac{1}{\sqrt{n}}} \\ &\leq \sqrt{n} \end{aligned}$$

Armortized Analysis

Problem 7.

(a) For the case table doubling and halving, using the charging method (i.e., define the amortized cost by assigning appropriate additional cost to past operations) prove that the amortized cost of both of insertions and deletions is $O(1)$. Table doubling occurs when the number of stored elements n becomes $n = m$, in which case $m \to 2n$ with cost $\Theta(m)$. Table halving occurs when the table becomes 1/4 full, i.e., $n = m/4$, in which case $m \to 2n$, incurring cost $\Theta(m)$. Hint: the case of insertions is proved in the slides. Use the same idea to prove the case of deletions.

(b) For the case of 2-3 trees use the charging method to prove that we can use amortized costs of $O(\log(n))$ for insertion and $\emptyset$ for deletions, where n is the size of the set of keys in the tree at the time of the insertion.

(a) The amortized cost will be calculated using the charging method.

- When the table doubles from size m to size $2m$, the cost is $\Theta(2n)$, for which it will be charged to the previous $\frac{n}{2}$ insertions (distributing the big insert across the previous insertions evenly). Each insertion runs in $\frac{\Theta(2n)}{\frac{n}{2}} = \Theta(1)$.
- In table halving, when the table is down to 1/4 full, $n = m/4$, we shrink the table size from m to $m/2$ at $\Theta(m)$ cost. This way, the table is half full again after any resize (doubling or shrinking). Now each table doubling still has $\geq m/2$ insert operations to charge to, and each table halving has $\geq m/4$ delete operations to charge to. So the amortized cost per insert or delete is still $O(1)$.

(b) The amortized cost will be calculated using the charging method.

- When inserting, an $\mathcal{O}(\log n)$ cost is applied for the traversal (height of the tree). As it is already the same as what we intend to prove, we are done for insertion. We will utilize this chance to get charged for the deletion.
- When deleting, both merge and node redistribution are required, for which they incur additional costs. Those operations, which have $\mathcal{O}(\log n)$ cost, can be charged to the last made insertion. By definition, the last insertion can't be charged twice; hence, it proves the maintenance of $\mathcal{O}(\log n)$ cost.