VE 281 Homework 2 (Due 10/31 11:59PM)

Name: 朱闻沛 ID: 5183709l0226 .

- **Exercise 1. Open Addressing**

  Consider inserting the keys $10, 22, 31, 4, 15, 28, 17, 88, 59$ into a hash table of length $m = 11$ using open addressing with the auxiliary hash function $h(k) = k$. Illustrate the result of inserting these keys using linear probing, using quadratic probing with $c_1 = 1$ and $c_2 = 3$, and using double hashing with $h_1(k) = k$ and $h_2(k) = 1 + (k \mod (m-1))$.

  Solution:

  We use T_t to represent each time stamp t starting with $i = 0$, and if encountering a collision, then we iterate i from $i = 1$ to $i = m - 1 = 10$ until there is no collision.

  Linear probing:

$h(k,i) = (k+i) \mod 11$	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
0 mod 11		22	22	22	22	22	22	22	22
1 mod 11								88	88
2 mod 11									
3 mod 11									
4 mod 11				4	4	4	4	4	4
5 mod 11					15	15	15	15	15
6 mod 11						28	28	28	28
7 mod 11							17	17	17
8 mod 11									59
9 mod 11			31	31	31	31	31	31	31
10 mod 11	10	10	10	10	10	10	10	10	10

  Quadradic probing:

$h(k,i) = (k+i+3i^2)$	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
0 mod 11		22	22	22	22	22	22	22	22
1 mod 11									59
2 mod 11									
3 mod 11							17	17	17
4 mod 11				4	4	4	4	4	4
5 mod 11									
6 mod 11						28	28	28	28
7 mod 11								88	88
8 mod 11					15	15	15	15	15
9 mod 11			31	31	31	31	31	31	31
10 mod 11	10	10	10	10	10	10	10	10	10

  Double hashing:

$h(k,i) = (k+i(1+k \mod 10)) \mod 11$	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
0 mod 11		22	22	22	22	22	22	22	22
1 mod 11									
2 mod 11									59
3 mod 11							17	17	17
4 mod 11					4	4	4	4	4
5 mod 11						15	15	15	15
6 mod 11							28	28	28
7 mod 11								88	88
8 mod 11									
9 mod 11				31	31	31	31	31	31
10 mod 11	10	10	10	10	10	10	10	10	10

- **Exercise 2. Slot-Size Bound for Chaining**

  Suppose that we have a hash table with n slots, with collisions resolved by chaining, and suppose that n keys are inserted into the table. Each key is equally likely to be hashed to each slot. Let M be the maximum number of keys in any slot after all the keys have been inserted. Your mission is to prove an $O(\lg n/\lg\lg n)$ upper bound on $\mathrm{E}[M]$, the expected value of M

  a. Argue that the probability Q_k that exactly k keys hash to a particular slot is given by

$$Q_k = \left(\frac{1}{n}\right)^k \left(1-\frac{1}{n}\right)^{n-k} \binom{n}{k}$$

  b. Let P_k be the probability that $M = k$, that is, the probability that the slot containing the most keys contains k keys. Show that $P_k \le nQ_k$.

  c. Use Stirling's approximation to show that $Q_k < e^k/k^k$.

  d. Show that there exists a constant $c > 1$ such that $Q_{k_0} < 1/n^3$ for $k_0 = c\lg n/\lg\lg n$. Conclude that $P_k < 1/n^2$ for $k \ge k_0 = c\lg n/\lg\lg n$

  e. Argue that

$$\mathrm{E}[M] \le \Pr\left\{M > \frac{c\lg n}{\lg\lg n}\right\}\cdot n + \Pr\left\{M \le \frac{c\lg n}{\lg\lg n}\right\}\cdot\frac{c\lg n}{\lg\lg n}$$

  Conclude that $\mathrm{E}[M] = O(\lg n/\lg\lg n)$

  Solution:

a. for each key into a slot, $P=\frac{1}{n}$, not into that slot $P=1-\frac{1}{n}$,
we choose C_n^k keys into that particular slot.
$Q_k=\left(\frac{1}{n}\right)^k\left(1-\frac{1}{n}\right)^{n-k}C_n^k=\left(\frac{1}{n}\right)^k\left(1-\frac{1}{n}\right)^{n-k}\binom{n}{k}$.

b. Q_k is exactly k keys hash to particular slot.
let Y_i denotes number of keys hash to slot i.
Ai denotes number of keys hash to slot i is k.
$P_k=P\{k=(\max_{1\le i\le n} Y_i)\}\le P\{\text{there exists i s.t. } Y_i=k\}$
$=P\{A_1\cup A_2\cdots\cup A_n\}\le P\{A_1\}+P\{A_2\}+\cdots+P\{A_n\}=nQ_k$

$\therefore P_k\le nQ_k$.

c. $Q_k=\left(\frac{1}{n}\right)^k\left(1-\frac{1}{n}\right)^{n-k}\binom{n}{k}=\left(\frac{1}{n}\right)^k\left(1-\frac{1}{n}\right)^{n-k}\frac{n!}{k!(n-k)!}$
$<\left(\frac{1}{n}\right)^k\frac{n!}{k!(n-k)!}=\frac{1}{k!}\cdot\frac{n!}{(n-k)!}\cdot\frac{1}{n^k}$
$=\frac{n(n-1)\cdots(n-k+1)}{n!}\cdot\frac{1}{k!}<\frac{1}{k!}\simeq\frac{1}{\sqrt{2\pi k}\left(\frac{k}{e}\right)^k}<\left(\frac{e}{k}\right)^k$

$\therefore Q_k<\left(\frac{e}{k}\right)^k$

$k_0 = c\lg n / \lg\lg n$

d. since $\lg\lg n > 0$, $n \geq 3$.

$Q_{k_0} < \frac{e^{k_0}}{k_0^{k_0}} < \frac{1}{n^3} \Rightarrow$ we need to prove $\frac{k_0^{k_0}}{e^{k_0}} > n^3$.

take $\lg$ on both side.

$3\lg n < (\lg k_0 - \lg e)\, k_0$

$= (\lg\lg n + \lg c - \lg e - \lg\lg\lg n)\ \frac{c\lg n}{\lg\lg n}$

$\Rightarrow 3 < c\left(1 + \frac{\lg c - \lg e}{\lg\lg n} - \frac{\lg\lg\lg n}{\lg\lg n}\right)$

since $\lim_{n\to\infty}\left(1 + \frac{\lg c - \lg e}{\lg\lg n} - \frac{\lg\lg\lg n}{\lg\lg n}\right) = 1$

let $x = \left(1 + \frac{\lg c - \lg e}{\lg\lg n} - \frac{\lg\lg\lg n}{\lg\lg n}\right)$

there exists value n_0 s.t. $x \geq \frac{1}{2}$, Thus, $c > 6$ for $n \geq n_0$

we have $3 < cx$.

Thus, $Q_{k_0} < \frac{1}{n^3}$.

by conclusion of (b)

for $k = k_0$, $P_{k_0} \leq nQ_{k_0} < n\left(\frac{1}{n^3}\right)$

$\therefore P_{k_0} < \frac{1}{n^2}$ then, for $k \geq k_0$, $P_k \leq P_{k_0} < \frac{1}{n^2}$.

so we prove $P_k < \frac{1}{n}$ for $k \geq k_0$.

e. $E[M] = \sum_{k=1}^{n} k \cdot P\{M = k\}$

$= \sum_{k=1}^{c\lg n/\lg\lg n} k\, P\{M = k\} + \sum_{k=c\lg n/\lg\lg n}^{n} k \cdot P\{M = k\}$

$\leq (c\lg n/\lg\lg n)\, P\{M \leq \frac{c\lg n}{\lg\lg n}\} + P\{M > \frac{c\lg n}{\lg\lg n}\} \cdot n$

$\Rightarrow E[M] \leq c\lg n/\lg\lg n + \left\{\frac{1}{n^2} \cdot n\right\} \cdot n$

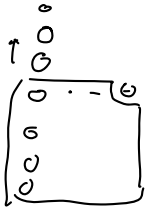
$\leq c\lg n/\lg\lg n + 1$

$\therefore E[M] = O(\lg n/\lg\lg n)$.

- **Exercise 3. D-select**

  In the deterministic linear-time selection algorithm, instead of partitioning all the inputs into a number of groups of size 5, we partition the inputs into a number of groups of size 7. The rest steps of the algorithm are similar to the original version. Will the runtime of this new algorithm still be O(n), where n is the input size? Prove your claim.

  Solution:



at least $\frac{4}{7} \cdot \frac{1}{2} = \frac{2}{7}$ elements smaller/larger than $x_{k/2}$.

$T(n) \le cn + T(\frac{n}{7}) + T(\frac{5n}{7})$.

choose $a = 7c$

$T(1) \le c$.

$T(n) \le cn + T(\frac{n}{7}) + T(\frac{5n}{7})$

$T(n) \le 7cn$.

Proof by induction:

Base case: $T(1) \le 7c$.

$T(k) \le 10ck$. $\forall k < n$ then

$T(n) \le cn + T(\frac{n}{7}) + T(\frac{5n}{7}) \le cn + cn + 5cn = 7cn$

$\therefore T(n) = O(n)$. still be $O(n)$.

- **Exercise 4. Silly Blue Tiger**

  1. Imagine that an algorithm requires us to hash strings containing English phrases. Knowing that strings are stored as sequences of characters, Blue Tiger decides to simply use the sum of those character values (modulo the size of its hash table) as the string's hash. Will the performance of the implementation match the expected value shown in lecture? (Yes/No and the brief reasoning are what we expect.)
  2. Blue Tiger decides to implement both collision resolution and rehashing for its hash table. However, it does not want to do more work than necessary, so it wonders if it needs both to maintain the correctness and performance it expects. After all, if it has rehashing, it can resize to avoid collisions; and if it has collision resolution, collisions ~~don't cause~~ correctness issues. Which statement about these two properties true? (Choose one or two of the statements and briefly explain your choice.)
  3. Blue Tiger wants to implement rehashing. Assuming it is enlarging a table of size m into a table of size m', and the table contains n elements, what is the best time complexity to achieve this rehashing step (expanding a m-slot table with n elements into a m'-slot table)? Answer in big theta notation in terms of the variables in the context. Intermediate results of your complexity analysis should be displayed.
  4. In lecture, we discussed approximately doubling the size of our hash table. Blue Tiger begins to implement this approach (that is, it lets m' be a prime close to but greater than $2m$) but stops when it thinks that it might be able to avoid wasting half of the memory the table occupies on empty space by letting m' be a prime number close to $(m + k)$ instead, where k is some constant. Does this work equally well comparing to picking a prime number close to $2m$? Explain your answer.

Solution:

1. No. sum of character may result in same hash key for different words. (post, stop). The collision will happen. Thus, performance of insert, find, remove will not be O(1) when there are many phrases.

2. ② is true.

① = rehashing improve hash table when load factor too big. but it cannot avoid collision. false

② : with collision resolution, collision will not cause correctness issue if using separate addressing. If use open addressing, L cannot be more than 1.

3. ① create new hash table O(1)

② insert all n elements into new table cost O(n)

∴ total time complexity $O(1)+O(n) = O(n)$.

4. No, the works not equally well. ($k<m$).

when number of elements is less than $L \times (m+k)$, using prime number bigger than $(m+k)$ will reduce memory cost. (the smallest)

when number of elements is larger and close to $L \times 2m$. use prime number close to $(m+k)$ will have lower time efficiency.

h(x)+i g(x)

9 6

- **Exercise 5. Rehashing**

Given a sequence of inputs 4371, 1323, 6173, 4199, 4344, 9679, 1989, insert them into a hash table of size 10. Suppose that the hash function is h(x) = x mod 10. Hash table uses the double hashing with the second hash function as g(x) = (7 - x) mod 7.(Be careful with this mod calculation)
If any given input cannot be inserted correctly, do rehashing to guarantee a successful insertion, using a new table size as introduced in class (the first prime number after doubling the original hash table size) and a new hash function h(x) = x mod new_size. Note that the order in rehashing depends on the order stored in the old hash table, not their previous inserting order.
Solve this question step by step and show intermediate results.

Solution:

The initial hash table:

we cannot find block to place 1989

	T_0	T_1	T_2	T_3	T_4	T_5	T_6
*entry*0							
*entry*1	4371	4371	4371	4371	4371	4371	4371
*entry*2							
*entry*3		1323	1323	1323	1323	1323	1323
*entry*4			6173	6173	6173	6173	6173
*entry*5						9679	9679
*entry*6							
*entry*7					4344	4344	4344
*entry*8							
entry 9				4199	4199	4199	4199

Does the hash table need rehashing? If it does, when and why (answer in 1-2 sentences)? Also finish the rehashing process by drawing your own progress table that is similar to the one above:

Then we cannot put 4199 in correct place.

10 x 2 = 20 nearest prime 23.

0							
1	4371	4371	4371	4371	4371	4371	4371
2							
3							
4							
5							
6							
7							
8							
9			6173	6173	6173	6173	6173
10							
11							1989
12		1323	1323	1323	1323	1323	1323
13				4199	4199	4199	4199
14							
15							
16							
17							
18						9679	9679
19							
20					4344	4344	4344
21							
22							

- **Exercise 6**

  Suppose we want to design a hash table containing at most 1162 elements using quadratic probing. We require it to guarantee successful insertions, S(L) < 2 and U(L) < 4. Please determine a proper hash table size and show all intermediate steps.

  $U(L) = \frac{1}{1-L} < 4 \Rightarrow L < \frac{3}{4}$

  $S(L) = \frac{1}{L}\ln\frac{1}{1-L} < 2 \Rightarrow L < 0.7968.$

  $\Rightarrow L < \frac{3}{4}.$

  1162 elements $1162 \cdot \frac{4}{3} = \frac{4648}{3} \approx 1550.$

  the nearest prime is 1553.

- **Exercise 7**

  In a few sentences (in an itemized fashion), explain why a hash table with open addressing and rehashing (doubling the size every time when the load factor exceeds a threshold) achieves insertion in $O(1)$ time.

  Answer the following questions: 1) When rehashing is not triggered, is the insertion time $O(1)$ (does it have an upper bound)? Why? 2) Show that at any given point, the amortized (distributed to all items in the hash table) cost of rehashing is $O(1)$ for each item in the hash table.

1) The insertion time is O(1). It has upper bound O(M).
If the hash table is properly designed, L is small to avoid too many comparisons.
So time complexity for insertion is O(1). Upper bound happens when comparisons is O(L·M). thus, upper bound is O(M).

2) for M+1 items, the time complexity for rehashing is
$2O(M)+2O(1) = O(M)$.
then distributed to all items, time complexity is $\frac{O(M)}{M+1} = O(1)$.

What if during rehashing, the hash table is expanded by a factor x with $x > 2$? Is the amortized cost of rehashing still $O(1)$? If no, explain. If yes, show your work. If it depends, show both.

Yes, creating new hash table is O(1). insertion takes O(M). so time complexity is still O(M).
Distribute to all elements it will still be O(1).