CMPT 280

Topic 15: Hash Tables

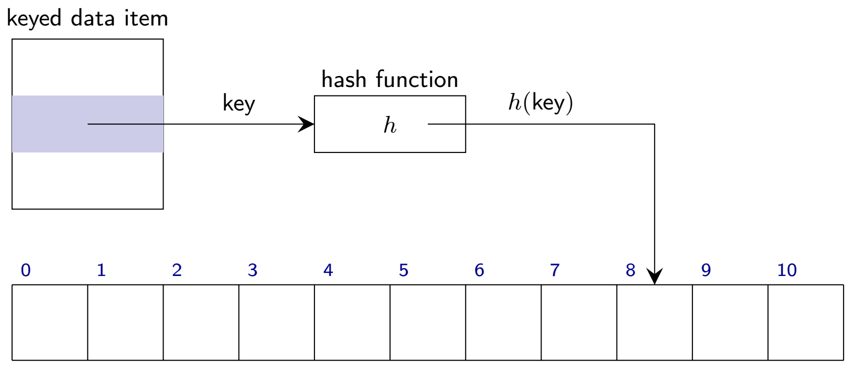
Mark G. Eramian

References

- Textbook, Chapter 15

Hash Tables

Basic idea:



Hashing Functions

- For integers: $h_{\text{int}}(k) = k \bmod N$
- For floats: $h_{\text{float}}(k) = h_{\text{int}}(\text{round}(k))$
- For strings: $h_{\text{string}}(k) = \left(\sum_i k[i]\right) \bmod N$

Exercise 1

Let $N = 12$. Using the appropriate hash functions on the previous slide,

(a) Determine the hash values of the keys: 7, 9, 26, 48

(b) Determine the hash values of the keys: 9.7, 13.7

(c) Determine the hash values of the keys: "ABC", "123".(the ASCII value of 'A' is 65; the ASCII value of '1' is 49)

Solution

(a) The hash values are 7, 9, 2, and 0, respectively.

(b) The hash values are 10 and 2, respectively.

(c) The for "ABC" the hash value is $(65 + 66 + 67) \bmod 12 = 6$ respectively. The ASCII character values for characters 'A', 'B', and 'C' are 65, 66, and 67 respectively. We add those together and get 198, and 198 mod 12 is 6. For "123" the ASCII character values are 49, 50, and 51, so the hash value for "123" $(49 + 50 + 51) \bmod 12 = 6$.

Exercise 2
Using the appropriate hash function from the earlier slide, insert items with keys 42, 27, 69, 78, and 7 into the following hash table array:

0	1	2	3	4	5	6	7	8

(a) using linear probing for collision resolution; $p(j) = j$

(b) using quadratic probing for collision resolution; $p(j) = (-1)^{i-1}\left(\frac{i+1}{2}\right)^2$

(c) using double hashing; $p(j) = h_2(k) \cdot j$, $h_2(k) = k \bmod 7 + 1$

Solution

(a) To hash integer keys we just have to take the keys modulo N (in this case $N = 9$). The keys 42, 27, 69, 78, and 7 hash to indices 6, 0, 6, 6, and 7, respectively. The first two keys hash without collision:

0	1	2	3	4	5	6	7	8
27						42		

The next key, 69, hashes to index 6, so we have to linear probe with an increment of 1, so it ends up at index 7. Then 78 also hashes to index 6, so we have to linear probe to index 7 again, but this time we find it occupied, so we move to index 8, which we find is empty, so the item with key 78 ends up there.

0	1	2	3	4	5	6	7	8
27						42	69	78

Finally, key 7 hashes to index 7, where we have a collision, even though no other key hashed to index 7! We have in fact collided with an alternate index for index 6. This makes no difference, it's still a collision, and we resolve it in the same way. We linear probe with a probe increment of 1 to offset 8, which is occupied. So we move on to the next one which would be offset 9, but since we always "mod N", this becomes 0 because 9 mod 0 is 0. But index 0 is also occupied! So we probe again to offset 1, and find an empty index for the item with key 7:

0	1	2	3	4	5	6	7	8
27	7					42	69	78

(b) Using quadratic probing, again, the first two keys hash without collisions:

0	1	2	3	4	5	6	7	8
27						42		

Then 69 hashes to index 6 as before, but this time we use quadratic probing to find the alternate index. $p(1) = 1$, so our first alternate index is $6 + 1 \mathrm{mod} 9 = 7$. This index is available, so the item with key 69 is placed at index 7.

Then 78 hashes to index 6. It is full, so we probe the first alternative index which we previously found to be 7; however it is full too. So we compute $p(2) = -1$, and our second alternate index is $6 - 1 \mathrm{mod} 9 = 5$, and the item with key 78 goes there:

0	1	2	3	4	5	6	7	8
27	7				78	42	69	

Finally, the key 7 hashes to index 7, which is occupied. So we try the first alternate index at $7 + p(1) \bmod 9 = 8$. This index is available so the item with key 7 goes at index 8.

0	1	2	3	4	5	6	7	8
27					78	42	69	7

Note how overall we used fewer probes to insert all the items compared to linear probing.

Solution

(c) For double hashing, once again, the first two keys hash without collisions:

0	1	2	3	4	5	6	7	8
27						42		

Then 69 hashes to 6 and we have a collision. The probe increment is the determined by the second hasing function: $h_2(69) = 69 \bmod 7 + 1 = 7$. So our first probe location is $(6+7) \bmod 9 = 13 \bmod 9 = 4$. Offset 4 is open, so 69 hashes there:

0	1	2	3	4	5	6	7	8
27				69		42		

Next, 78 hashes to index 6 as before and we have another collision. This time the probe increment $h_2(78) = 78 \bmod 7 + 1 = 2$. Thus we probe offset 6 and then 8, which is open:

0	1	2	3	4	5	6	7	8
27				69		42		78

Finally the key 7 hashes to offset 7, which is not occupied:

0	1	2	3	4	5	6	7	8
27				69		42	7	78

Exercise 3

0	1	2	3	4	5	6	7	8
27	7	11	12			42	69	78

How many probes are required to perform a search for each of the keys: 27, 78, 12, and 87 using linear probing?

Solution

- 27 hashes to 0, and is found right away with one probe
- 78 hashes to 6, and is found in 3 probes at offsets 6, 7, and 8.
- 12 hashes to 3, and is found in 1 probe.
- 87 hashes to 6, and eight probes (indices 6 thru 8, then 0 thru 4) are needed to determine that there is no item with key 87 in the hash table. We know it's not present when we reach an unused bin before finding the sought key.

Exercise 4

0	1	2	3	4	5	6	7	8
27		11	12	69		42	7	78

- What is the load factor of the hash table pictured above?
- How many probes are required to perform a search for each of the keys: 78, 11, and 87 using double hashing (again using $p(j) = h_2(k) \cdot j$, and $h_2(k) = k \bmod 7 + 1$).

Solution

The load factor is the number of elements in the hash table, divided by the number of entries in the table, in this case, $7/9 = 0.777\ldots.$

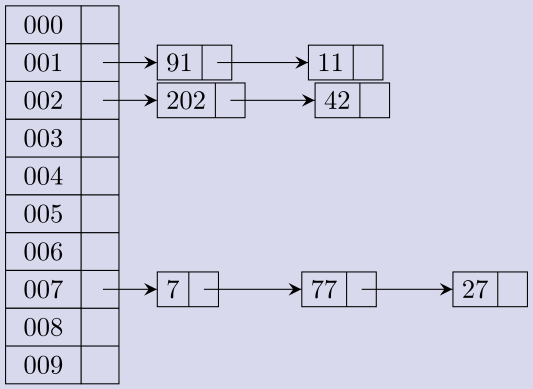
- 78 hashes to 6 (78/9 is 8 remainder 6) and is not found; $78 \bmod 7 + 1 = 2$ so the probe increment is 2. So we check entries 6, and 8 for a total of 2 probes.
- 11 hashes to 2 ($11/9 = 1$ remainder 2) and is found immediately in one probe.
- 87 hashes to 6 ($87/9 = 9$ remainder 6). The key there doesn't match; $87 \bmod 7 + 1 = 4$ so the probe increment is 4. Thus we probe 6, then 1, and we find an empty cell, so we can conclude it's not there. Two probes are made in total.

Exercise 5

- Draw a diagram representing the structure of a hash table with separate chaining with $N = 10$ after the insertion of items 27, 11, 91, 77, 42, 202, and 7.
- What is the load factor of the hash table?

Solution

We hash each item's key to an array index, then insert the item onto the front of the linked list for that index (because that's guaranteed to be $O(1)$ regardless of whether the list is singly or doubly linked).



Exercise 6

In a hash table that uses separate chaining, what is the average number of items that will be examined when searching for a key if the load factor of the hash table is:

(a) 2?

(b) 5?

(c) an integer n?

(d) What is the time complexity of searching this hash table in the average case?

(e) What is the time complexity of searching this hash table in the worst case?

(f) What is the time complexity of searching this hash table in the best case?

Solution

(a) If the load factor is 2, there must be 2 items on each linked list on average. We can find the list the sought item is on by hashing, so we only ever need to search one list. So what is the average number of items examined when searching a list of two items? On average, about half the time we'll examine 1 item, and the other half of the time we'll examine two items. So on average we'll examine 1.5 items.

(b) If the load factor is 5, there are, on average, 5 items on each linked list. To search one list, we will have to examine, with about equal probability, either one, two, three, four, or five items. Thus the average number of items examined on a list of average length will be $(1 + 2 + 3 + 4 + 5)/5 = 15/5 = 3$.

(c) In general if the load factor is $n > 2$, on average we'll need to examine $\sum_{i=1} n/n = n \cdot (n + 1)/2n = 1/2n + 1/2$ items.

(d) It is $O(n)$ where n is the load factor because the number of items examined, $1/2n + 1/2$ is $O(n)$.

(e) $O(k)$ where k is the number of items in the hash table (but this is very unlikely!). The worst case occurs when all k items in the hash table happened to hash to the same offset, and the item sought is at the end of the chain at that offset.

(f) $O(1)$. There's always the possibility that the first item we examine is the one we want.

Exercise 7
For a hash table that uses separate chaining:

(a) What is the time complexity for insertion in the worst case? best case?

(b) What is the time complexity for deletion in the average case? worst case? best case?

Solution

(a) For the worst case it is $O(1)$. Hashing the key is $O(1)$, this gets us the linked list for the index. Then insertion to the beginning of that list is $O(1)$. The best case is the same as the worst case.

(b) Once we find the item we want to delete, it takes only $O(1)$ to remove the item from the linked list. So the average case time complexity is the same as for searching: $O(LF)$ where LF is the load factor. The worst and best cases are also the same as for the search operation. The worst case is that all items are on the same list and we have to search through all of them: $O(k)$ where k is the number of items in the hash table. The best case is that we find the item to be deleted right away: $O(1)$.

Next Class

- *Next class reading:* Chapter 16: 2-3 trees.