Trevor Swan

CSDS233 ASSIGNMENT 4

Question 1) Collision Resolution Using Linear Probing and Double Hashing

The keys 14, 17, 18, 3, 8, 1, 18, 11, 13, and 20 are inserted into an initially empty hash table of size 10 using the hash function h(x) = x mod 10.

a) Hash: $h(x) = x \bmod 10$

Key	14	17	18	3	8	1	18	11	13	20
h(x)	4	7	8	3	8	1	8	1	3	0

Insert 14, 17, 18, 3 — all unique, no issue

0	1	2	3	4	5	6	7	8	9
			3	14			17	18	

Insert 8, already occupied so step

0	1	2	3	4	5	6	7	8	9
			3	14			17	18	8

Insert 1 then 18, Spot 8 occupied, so step twice to resolve collision

0	1	2	3	4	5	6	7	8	9
18	1		3	14			17	18	8

Insert 11, Spot 1 is occupied -> probe

0	1	2	3	4	5	6	7	8	9
18	1	11	3	14			17	18	8

Insert 13, Position 3 occupied probe

0	1	2	3	4	5	6	7	8	9
18	1	11	3	14	13		17	18	8

Insert 20, Position 0 occupied -> probe

0	1	2	3	4	5	6	7	8	9
18	1	11	3	14	13	20	17	18	8

b) $h1(x) = x \bmod 20$ and $h2(x) = 7 - (x \bmod 7)$

Key	14	17	18	3	8	1	18	11	13	20
h1(x)	14	17	18	3	8	1	18	11	13	0
h2(x)	7	4	3	4	6	6	3	3	1	1

Intermediate steps not required, so I just highlighted i=0 to start

To resolve collisions: $h1(x) + i \cdot h2(x)$, i++ while collision present

0	1	2	3	4	5	6	7	8	9
20	1		3	18				8	

10	11	12	13	14	15	16	17	18	19
	11		13	14			17	18	

(i) $18 + 1(3) = 21 \rightarrow 18 + 2(3) = 24$

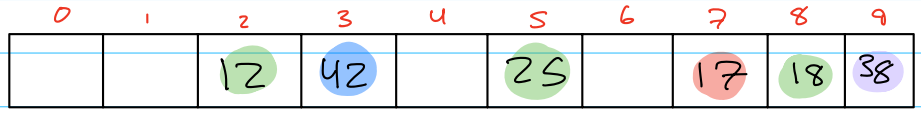
Question 2) Rehashing

You have a hash table that exceeds the load factor threshold = 0.7. After inserting the following keys [12, 18, 25, 42, 38, 17], the table size is doubled from 10 to 20.

a) Original Table before expansion and rehash $h(x) = x \bmod 10$

Only One hash function given, assume use of linear probing

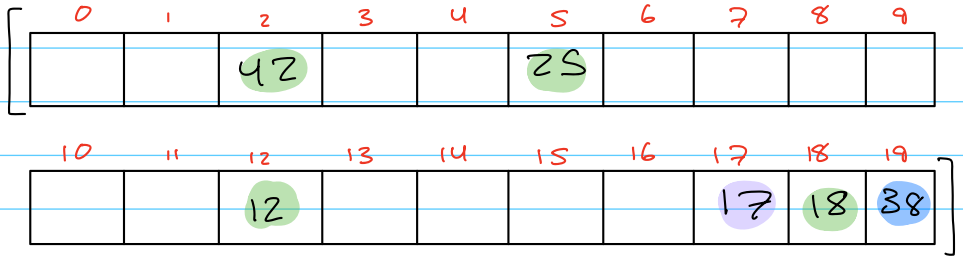
Key	12	18	25	42	38	17
h(x)	2	8	5	2	8	7



b) New rehashed table with doubled size and new hash $h(x) = x \bmod 20$

Only One hash function given, assume use of linear probing

Key	12	18	25	42	38	17
h(x)	12	18	5	2	18	17



Question 3) Comparison of Collision Resolution Techniques

Explain and compare the following collision resolution techniques:

- Linear Probing
- Quadratic Probing
- Separate Chaining

In your answer, discuss the pros and cons of each method, focusing on their performance under high load factors and how they handle clustering.

Linear Probing: Linear probing is an easy to implement, just checking sequential elements in the hash table until an open spot is found. The biggest problem of linear probing is that it groups elements together frequently. As spots are searched linearly, clusters of elements are created when there are collisions. As clusters are created, it increases the chance for more collisions to occur, which increases clustering further as the load factor increases. This causes the time it takes to find an open position to be much higher as the load factor increases.

Quadratic Probing: Quadratic probing is also very easy to understand and implement, simply checking spots in a quadratic manner instead of a linear one. Compared to linear probing, this method of collision handling means that there is much less clustering. The elements are spread out much more due to the quadratic stepping. Ultimately, this leads to a more even distribution of elements once the load factor increases. The problem with quadratic probing is that there are pseudo-clustered areas in the table. This occurs when hashing elements that map to the same position. In this case, the efficiency is identical to the linear probing method as the probing will check the same pattern of locations when given the same position. This means that this collision case is effectively the same as linear probing. Another big issue is that, even though performance under high load is better than linear probing, it still suffers from the fact that a spot may not always be found. With quadratic probing, it is not guaranteed to resolve a collision and insert an item if the table size is not prime and the load factor exceeds 50%. In general though, this is preferred to linear probing when considering clustering, as we can design our table and optimize the load factor for maximum performance.

Separate Chaining: Separate chaining is slightly more complex than the other two methods. Here, the table usually uses linked lists at each index which stores multiple elements. If a collision occurs, the element simply gets appended to the linked list at that index. Clustering is not a concern for this method as each slot can store multiple elements without issue. At higher load factors, this method is still incredibly efficient as the only thing that is changed is the size of the linked lists. It is also very easy to implement deletions due to the nature of linked lists, though deletion is not difficult for linear and quadratic probing. The biggest issue with this method is the performance and memory overhead of using linked lists. Since each position is its own linked list, there is much more memory used to store elements that were caught in a collision. Also, linked list traversal is an O(n) operation, which is not great considering the goal of hash tables is efficient search, addition, and deletion. As linked list size increase, there is much more memory used and item retrieval takes much longer. This method is the best for handling high load factor tables, while linear and quadratic probing should be limited to lower load factor scenarios.1