Hash Tables!

CS112 Recitation

Ashwin Haridas
ah1058@rutgers.edu

Original slides by: Hoai-An Nguyen

Let’s Review

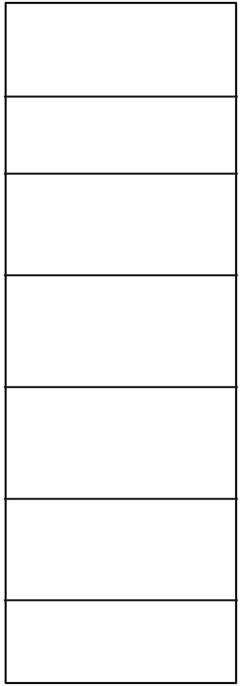
- Hash tables save items (key value pairs) in a key-indexed table (an array)
    - If we apply a hash function to a key (i.e. S), we get an index (2)
- When keys have the same hashed index, we resolve the collision with separate chaining or linear probing

insert(“S”, 0)
“S” hashes to 2

insert(“A”, 1)
“A” hashes to 0

insert(“QRS”, 2)
“QRS” hashes to 2

Hash Table

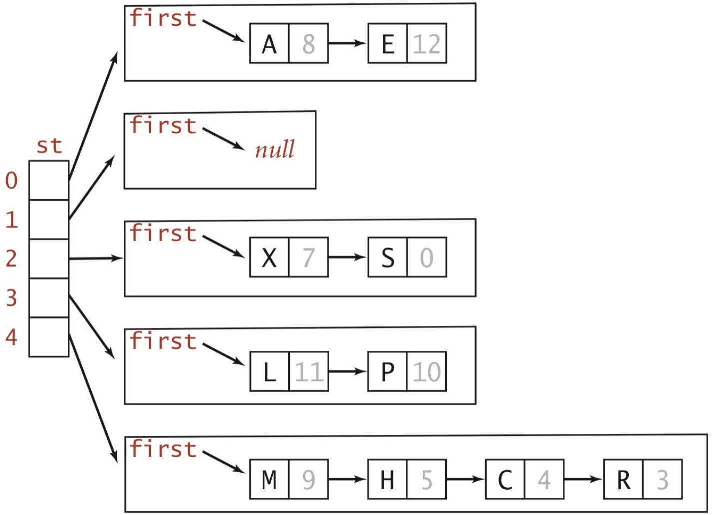


Let’s Review - Resolving by Chaining

- When a collision occurs, insert the item into the **front** of a linked list at that index
- Initially, before anything is inserted, we have empty linked lists at each index

Hash Table with chaining for collision resolving

key	hash	value
S	2	0
E	0	1
A	0	2
R	4	3
C	4	4
H	4	5
E	0	6
X	2	7
A	0	8
M	4	9
P	3	10
L	3	11
E	0	12



Let’s Review - Probing

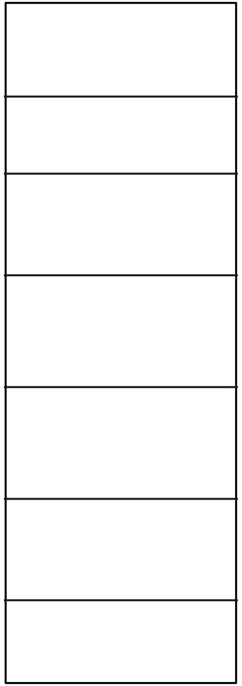
- When a collision occurs, insert the item at the **next available index** of the hash table

insert(“S”, 0)
“S” hashes to 2

insert(“A”, 1)
“A” hashes to 0

insert(“QRS”, 13)
“QRS” hashes to 2

Hash Table



Warm-Up

1. Is the following implementation of hashCode() legal?
```
public int hashCode() {
    return 17;
}
```
2. What is the worst case runtime for searching, inserting, and deleting when we implement separate chaining or linear probing?
3. Assuming uniform hashing, what is the average case runtime for searching, inserting, and deleting when we implement separate chaining?

Warm-Up

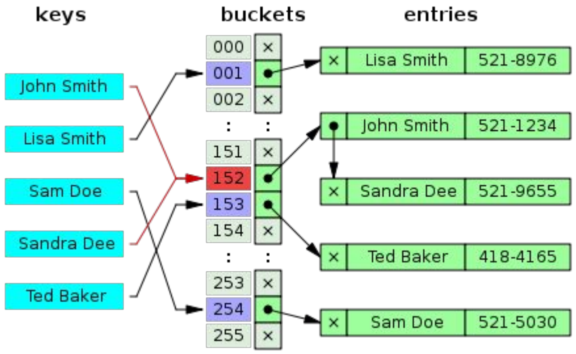
1. Is the following implementation of hashCode() legal?
```
public int hashCode() {
    return 17;
}
```
   Yes, as objects that are equal() will have the same hash code. However, it is ineffective because *all* keys will have the same hash code and will be inserted into the same index in the hash table.
2. What is the worst case runtime for searching, inserting, and deleting when we implement separate chaining? **O(n).**
3. Assuming uniform hashing, what is the average case runtime for searching, inserting, and deleting when we implement separate chaining? **O(1).**

Application of Hash Tables

- Software that tracks amount of times a song is played
- Hash Table stores song name as key, and the frequency as the value
- We can access the song with its frequency in O(1) average time
- We can update the frequency in O(1) average time
- Example:
    - "A" hashes to 0
    - "B" hashes to 3
    - "C" hashes to 4
    - Insert A, B, C
    - Increment A
    - Increment C

Question 1: Array Resizing w/ Linear Probing

Give the contents of a linear-probing hash table that results when you insert the keys E A S Y Q U T I in that order into an initially empty table of initial size m = 4 that is expanded with doubling whenever half full.



Use the hash function (11 * k) % m to transform the kth letter of the alphabet into a table index.

Note: when inserting the item, the key is the letter and the value is the count so far (ex: inserting the first item has the value 0, second item has value 1)

Also: when you double the size of the table, you have to rehash / reinsert all items

Insert E, A

Question 1: Solution

Each letter forms a key-value pair.
The value = the order it was inserted.
Hash = the value from passing the letter into the hash function.

State of HT

Insert

key hash value
E 3 0

0 null 2 null
1 null 3 E0

key hash value
A 3 1

0 A1 2 null
1 null 3 E0

Question 1: Solution

Now, the hash table is half-full so we resize.

Doubles hash table size and M becomes 8

0 null	4 null
1 null	5 null
2 null	6 null
3 null	7 null

Reinsert E, A

A - 1	G - 7	M - 13	S - 19	Y - 25
B - 2	H - 8	N - 14	T - 20	Z - 26
C - 3	I - 9	O - 15	U - 21	
D- 4	J - 10	P - 16	V - 22	
E - 5	K - 11	Q - 17	W - 23	
F - 6	L - 12	R - 18	X - 24	

Question 1: Solution

We re-insert the keys from our previous hash table.

Reinsert A1

key	hash	value
A	3	1

0	null	4	null
1	null	5	null
2	null	6	null
3	A1	7	null

Reinsert E0

key	hash	value
E	7	0

0	null	4	null
1	null	5	null
2	null	6	null
3	A1	7	E0

Insert S, Y

A - 1	G - 7	M - 13	S - 19	Y - 25
B - 2	H - 8	N - 14	T - 20	Z - 26
C - 3	I - 9	O - 15	U - 21	
D- 4	J - 10	P - 16	V - 22	
E - 5	K - 11	Q - 17	W - 23	
F - 6	L - 12	R - 18	X - 24	

Question 1: Solution

key hash value
S 1 2

0 null	4 null
1 S2	5 null
2 null	6 null
3 A1	7 E0

key hash value
Y 3 3

0 null	4 Y3
1 S2	5 null
2 null	6 null
3 A1	7 E0

Question 1: Solution

Doubles hash table size and M becomes 16

0 null	8 null
1 null	9 null
2 null	10 null
3 null	11 null
4 null	12 null
5 null	13 null
6 null	14 null
7 null	15 null

Reinsert E, A, S, Y

A - 1	G - 7	M - 13	S - 19	Y - 25
B - 2	H - 8	N - 14	T - 20	Z - 26
C - 3	I - 9	O - 15	U - 21	
D- 4	J - 10	P - 16	V - 22	
E - 5	K - 11	Q - 17	W - 23	
F - 6	L - 12	R - 18	X - 24	

Question 1: Solution

Reinsert A1

key	hash	value
A	11	1

0 null	8 null
1 null	9 null
2 null	10 null
3 null	11 A1
4 null	12 null
5 null	13 null
6 null	14 null
7 null	15 null

Reinsert E0

key	hash	value
E	7	0

0 null	8 null
1 null	9 null
2 null	10 null
3 null	11 A1
4 null	12 null
5 null	13 null
6 null	14 null
7 E0	15 null

Reinsert S2

key	hash	value
S	1	2

0 null	8 null
1 S2	9 null
2 null	10 null
3 null	11 A1
4 null	12 null
5 null	13 null
6 null	14 null
7 E0	15 null

Reinsert Y3

key	hash	value
Y	3	3

0 null	8 null
1 S2	9 null
2 null	10 null
3 Y3	11 A1
4 null	12 null
5 null	13 null
6 null	14 null
7 E0	15 null

Insert Q, U, T, I

A - 1	G - 7	M - 13	S - 19	Y - 25
B - 2	H - 8	N - 14	T - 20	Z - 26
C - 3	I - 9	O - 15	U - 21	
D- 4	J - 10	P - 16	V - 22	
E - 5	K - 11	Q - 17	W - 23	
F - 6	L - 12	R - 18	X - 24	

Question 1: Solution

key hash value
Q 11 4

0 null	8 null
1 S2	9 null
2 null	10 null
3 Y3	11 A1
4 null	12 Q4
5 null	13 null
6 null	14 null
7 E0	15 null

key hash value
U 7 5

0 null	8 U5
1 S2	9 null
2 null	10 null
3 Y3	11 A1
4 null	12 Q4
5 null	13 null
6 null	14 null
7 E0	15 null

key hash value
T 12 6

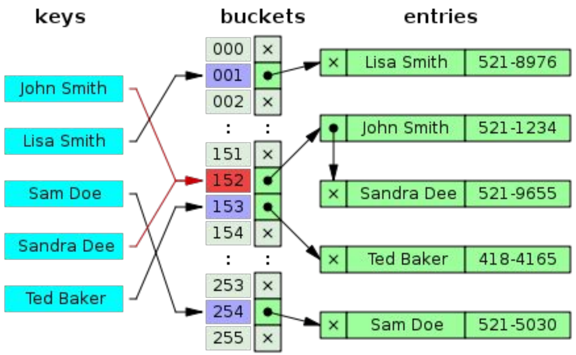
0 null	8 U5
1 S2	9 null
2 null	10 null
3 Y3	11 A1
4 null	12 Q4
5 null	13 T6
6 null	14 null
7 E0	15 null

key hash value
I 3 7

0 null	8 U5
1 S2	9 null
2 null	10 null
3 Y3	11 A1
4 I7	12 Q4
5 null	13 T6
6 null	14 null
7 E0	15 null

Question 2: Insert keys using separate chaining

Insert the keys E A S Y Q U T I O N in that order into an initially empty table of m = 5 lists, using separate chaining. Use the hash function 11 * k % m to transform the kth letter of the alphabet into a table index



EASYQUTION

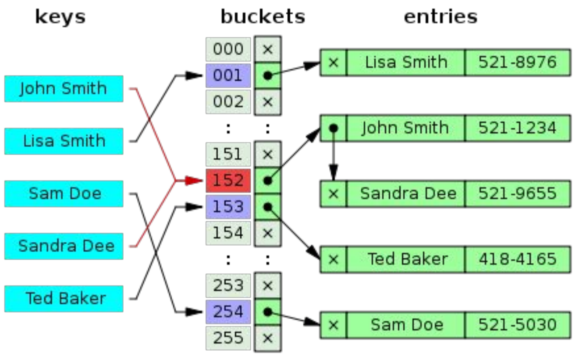
A - 1	G - 7	M - 13	S - 19	Y - 25
B - 2	H - 8	N - 14	T - 20	Z - 26
C - 3	I - 9	O - 15	U - 21	
D- 4	J - 10	P - 16	V - 22	
E - 5	K - 11	Q - 17	W - 23	
F - 6	L - 12	R - 18	X - 24	

E hashes to 0
A hashes to 1
S hashes to 4
Y hashes to 0
Q hashes to 2
U hashes to 1
T hashes to 0
I hashes to 4
O hashes to 0
N hashes to 4

Question 3: Comparisons

How many compares could it take, in the worst case, to insert n keys into an initially empty table, using linear probing with array resizing?

Assume that the initial hash table size is n and its size is expanded with doubling whenever half full.



Question 2: Solution Part 1

In the worst case all keys hash to the same index.

The number of compares per insert is:

1 for the first insert, 2 for the second insert, 3 for the third insert and so on, until the (n/2)th insert.

When the table is half full it is resized to 2n and the keys are reinserted, with 1, 2, 3, ..., n/2 compares.

Question 2: Solution Part 2

Then, for the insert of the other keys there are n/2 + 1, n/2 + 2, ..., n compares per insert.
This is equal to:
number of compares = (1 + 2 + 3 + ... + n/2) + 1 + 2 + 3 + ... + n/2 + (n/2 + 1) + (n/2 + 2) + ... + n
number of compares = (n/2 + 1) * n / 2 / 2 + (n + 1) * n / 2
number of compares = (n^2/2 + n) / 4 + (n^2 + n) / 2
number of compares = (n^2/2 + n) / 4 + (2n^2 + 2n) / 4
number of compares = (n^2/2 + (2n / 2)) / 4 + (2n^2 + 2n) / 4
number of compares = (n^2 + 2n) / 8 + (4n^2 + 4n) / 8
number of compares = (5n^2 + 6n) / 8

In the worst case, to insert n keys into an initially empty table, using linear probing with array resizing it would take (5n^2 + 6n) / 8 compares.

Good Work!

Go to https://dynrec.cs.rutgers.edu/live/

Enter the Quiz Code: