CSE12 - Lecture 17 - B00

Wednesday, November 2, 2022 9:00 AM

PA5 hard deadline → tonight
PA6 released today
PA5 Late / Resubmit → due Tuesday

Map and HashTable

Hash Function (same as previous)

```
int getIndex(String k) {
    return k.length;
}
```

# of buckets – 4
(i.e. the size of the array)

expandCapacity() called in set()

LoadFactor – 0.67

```
set("Smith", 1);
set("Johnson", 2);
set("Williams", 3);
set("Brown", 4);
set("Jones", 5);
set("Garcia", 6);
set("Miller", 7);
set("Davis", 8);
set("Rodriguez", 9);
set("Martinez", 10);
```

hash	index	LF
5	1 → 5	0/4
7	3 → 7	1/4
8	0 → 8	2/4
5	5	3/4
3	5 → 5	4/8
6	6	5/8
6	6	6/8
5	5	7/16
9	9	8/16
8	8	9/16

Draw the picture of the HashTable using Linear Probing (using expandCapacity)

What is the run-time for this HashTable (do picture first):

set()
Worst Case: $\Theta(N^2)$
Best Case: $\Theta(1)$

What conditions make up the best case for set()?
No collisions NO expand capacity

get()
Worst Case: $\Theta(N)$
Best Case: $\Theta(1)$

What conditions make up the best case for get()?
No collisions

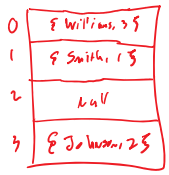
What happens if we remove something, then try to find something that collided it?

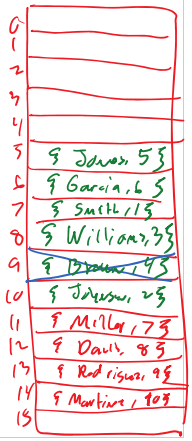
```
remove("Brown");
get("Davis");
```

What happens if we add something else?

```
set("Miranda", 11);
```

Key Value Pair < String, Integer > [ ]





tombstone → key → null
→ in hash from key value pair, equals() return false
→ expand capacity
↳ remove during re hash

get()

1

1

1

Amortized Analysis

What is the run-time for ArrayList add()?

Worst Case: $O(1) * \theta(N) \rightarrow O(N)$

Best Case: $\Omega(1)$

Average Case: $\Theta(1)$ per add

find()

wc N

bc 1

ac $\frac{N}{2}$

LF = .75

get()

O(1)

What is the run-time for HashTable set() using Separate Chaining and a good hash function?

Worst Case: $O(1) * \theta(N)$

O(1) Best Case: $\Omega(1)$

O(1) Average Case: $\Theta(1)$ per add

LF = .67

get()

What is the run-time for HashTable set() using Linear Probing and a good hash function?

O(1) Worst Case: $O(N)$

O(1) Best Case: $\Omega(1)$

O(1) Average Case: $\Theta(1)$ per add

Why is the hash function important?

even distribution

↳ less collisions

String → 4 billion

int ~ 4 billion

array index

4

8

16

32

Object

↳ int hashCode()

String

↳ int hashCode()

↳ override

2 4 8 16 32 64 128 256 512

2 4 8 16 32

1000

IIII IIII

ec called 9 times

log4

512
256
128
64
32
16
8
4
2

average

$\Theta(1)$ → per add