

PA5 die homepage
PA7 Late/Resubmit die homepage

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 = 0.75

```
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);
```

Draw the picture of the HashTable using Separate Chaining (using expandCapacity())

Does the run-time change with expandCapacity()?

What is the run-time for this HashTable (do picture first):
 $N \times 2^w$

set()

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

What conditions make up the best case for set()?

if k hash, empty \rightarrow empty bucket

get()

Worst Case: $\Theta(w)$ Best Case: $\Theta(1)$

What conditions make up the best case for get()?

empty, if already empty bucket

Why is the hash function important?

prevents clustering
even distribution

LoadFactor = 0.75

```
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);
```

size
Capacity

hash
0
1
2
3
4
5
6
7
8
9
10

index
0
1
2
3
4
5
6
7
8
9
10

LP
0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

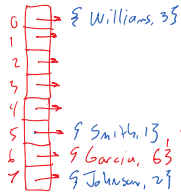
0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

0
1
2
3
4
5
6
7
8
9
10

re-hash \rightarrow re-hash \rightarrow 

Williams, 35
Smith, 15
Brown, 45
Jones, 55
Garcia, 65
Miller, 75
Davis, 85
Rodriguez, 95
Martinez, 105

clustering
bad hash function