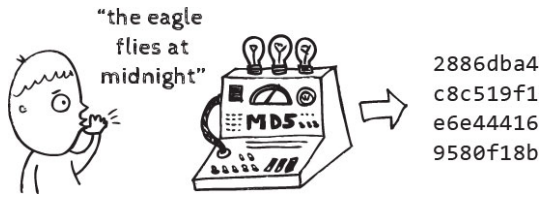
CSE 143

Computer Programming II

Hashing



IntSet 1

Today, we will consider multiple new implementations of `IntSet`:

```
1 public interface IntSet {
2    public void add(int value);
3    public void remove(int value);
4    public boolean contains(int value);
5 }
```

A New Data Structure 2

Design a class `RangeSet` that represents a set which only allows numbers inside a **fixed range**.

You should have a constructor:

`RangeSet(max)`	This constructor initializes a new `RangeSet` which only allows elements between 0 (inclusive) and **max** (exclusive).

And the following **public** methods:

`add(val)`	Adds **val** to the `RangeSet` if it is a valid value and throws an `IllegalArgumentException` otherwise.
`remove(val)`	Removes **val** to the `RangeSet` if it is a valid value in the set and does nothing otherwise.
`contains(val)`	Returns true if **val** is in the `RangeSet` and false otherwise.

`add`, `remove`, and `contains` must all be $\mathcal{O}(1)$

RangeSet 3

```
1  public class RangeSet implements IntSet {
2     private boolean[] data;
3
4     public RangeSet(int max) { this.data = new boolean[max]; }
5
6     public void add(int value) {
7        if (value >= this.data.length || value < 0) {
8           throw new IllegalArgumentException();
9        }
10       this.data[value] = true;
11    }
12
13    public boolean contains(int value) {
14       if (value >= this.data.length || value < 0) {
15          return false;
16       }
17       return this.data[value];
18    }
19
20    public void remove(int value) {
21       if (value < this.data.length && value >= 0) {
22          this.data[value] = false;
23       }
24    }
25 }
```

Generalizing to an Arbitrary Range 4

In `RangeSet`, when we got the number n, we mapped it to the index n. What if we had a function that took an input and mapped it to an index?

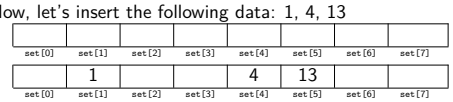
Definition (HashCode)

A **hash code** is a function that takes in a piece of data and maps it to an array index.

If we have an array of size 8, consider the following hashcode:

```
1 public int hashCode(int value) {
2    return value % 8;
3 }
```

Now, let's insert the following data: 1, 4, 13



HashSet Attempt #1


```
1  public class IntHashSet implements IntSet {
2     public final int DEFAULT_SIZE = 20;
3     public Integer[] data;
4
5     public IntHashSet() {
6        this.data = new Integer[DEFAULT_SIZE];
7     }
8
9     private int hashCode(int value) {
10       return value % data.length;
11
12    public void add(int value) {
13       this.data[hashCode(value)] = value;
14    }
15
16    public boolean contains(int value) {
17       return this.data[hashCode(value)] == value;
18    }
19
20    public void remove(int value) {
21       this.data[hashCode(value)] = null;
22    }
```

HashSet Attempt #1 Problem!

Consider the following insertions: 1, 21

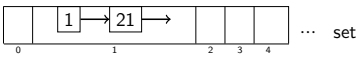
First:

Then:

Uh oh! We've overwritten the one!

How can we fix this?

Instead of storing an integer, let's store a list of integers



String HashCode!


```
1  public int hashCode() {
2     int h = hash;
3     if (h == 0 && value.length > 0) {
4        char val[] = value;
5
6        for (int i = 0; i < value.length; i++) {
7           h = 31 * h + val[i];
8        }
9        hash = h;
10    }
11    return h;
12 }
```