Data Structures: Linked Lists and Hash Tables

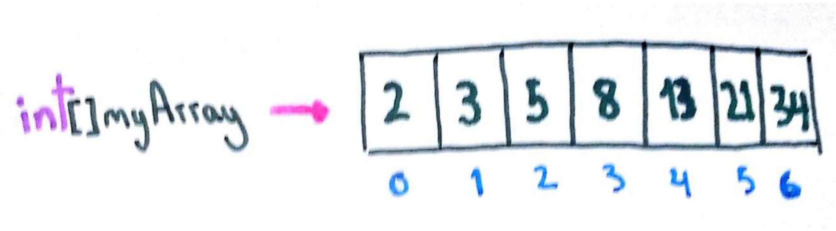
Data structures

Data structures are a way of organizing collections of data in the computer's memory

Data structures

Data structures are a way of organizing collections of data in the computer's memory

Examples of data structures we have seen so far



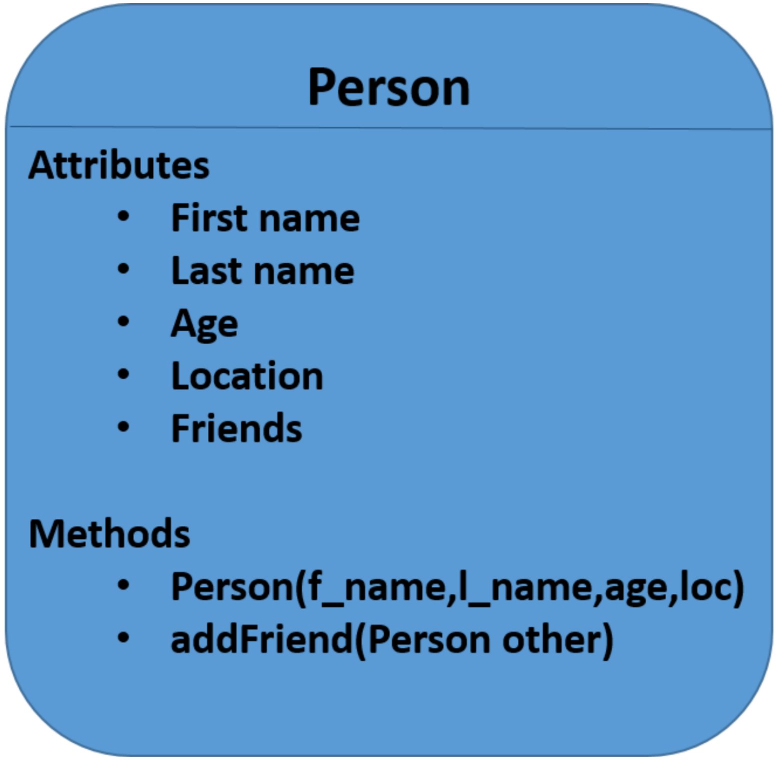
- Arrays

Data structures

Data structures are a way of organizing collections of data in the computer's memory

Examples of data structures we have seen so far

- Objects



Person

Attributes

- **First name**
- **Last name**
- **Age**
- **Location**
- **Friends**

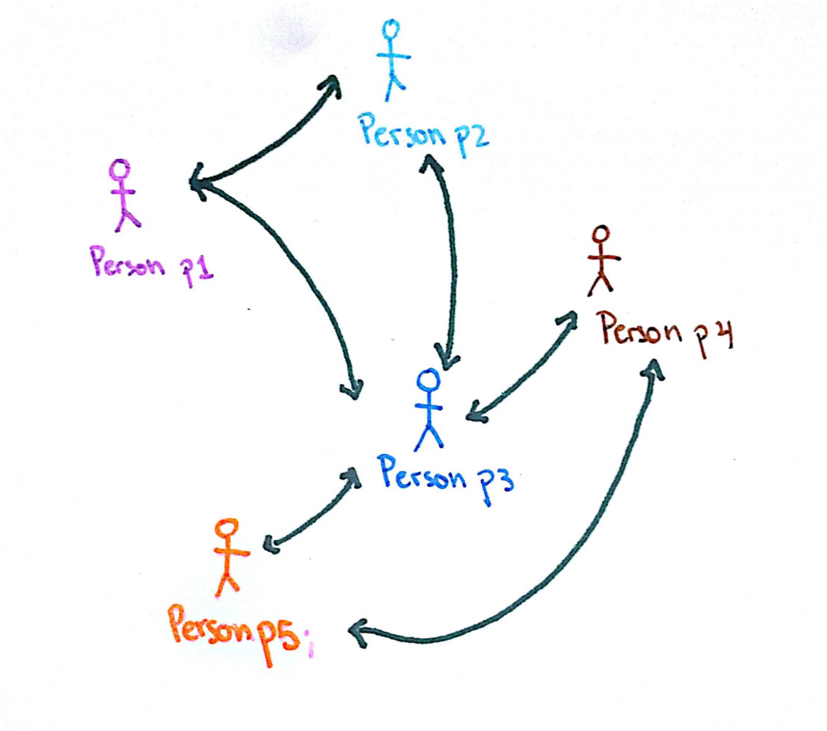
Methods

- **Person(f_name,l_name,age,loc)**
- **addFriend(Person other)**

Data structures

Data structures are a way of organizing collections of data in the computer's memory

Examples of data structures we have seen so far



- Networks of cities (Assignment 4), a social network (last class)

Data structures

Data structures are a way of organizing collections of data in the computer's memory

Examples of data structures we have seen so far

- Arrays
- Objects
- Networks of cities (Assignment 4), a social network (last class)
- **Linked Lists (today's class)**
- **Hashtables (today's class)**

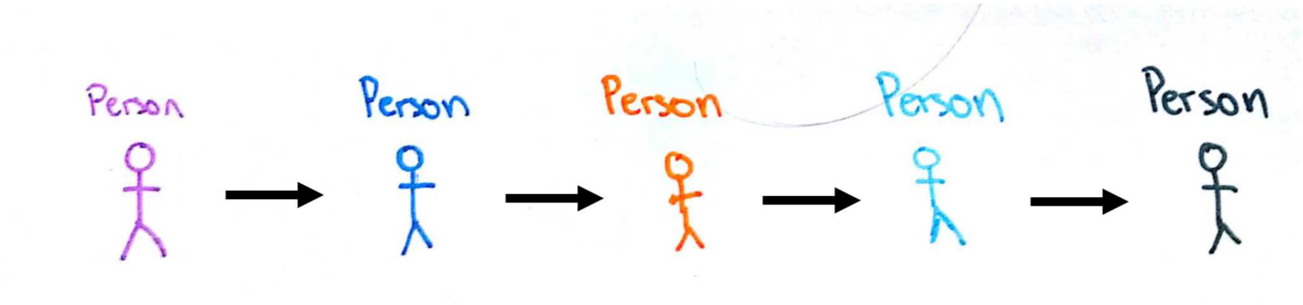
Linked Lists

Linked Lists

This is a visual representation of a linked list using the Person objects from the social network

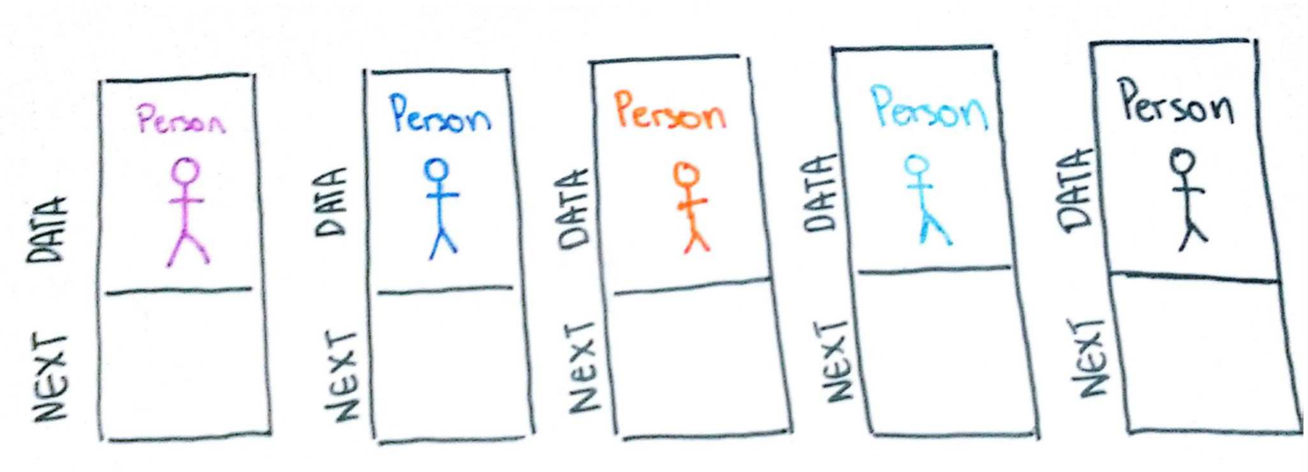
Linked Lists

This is a visual representation of a linked list using the Person objects from the social network



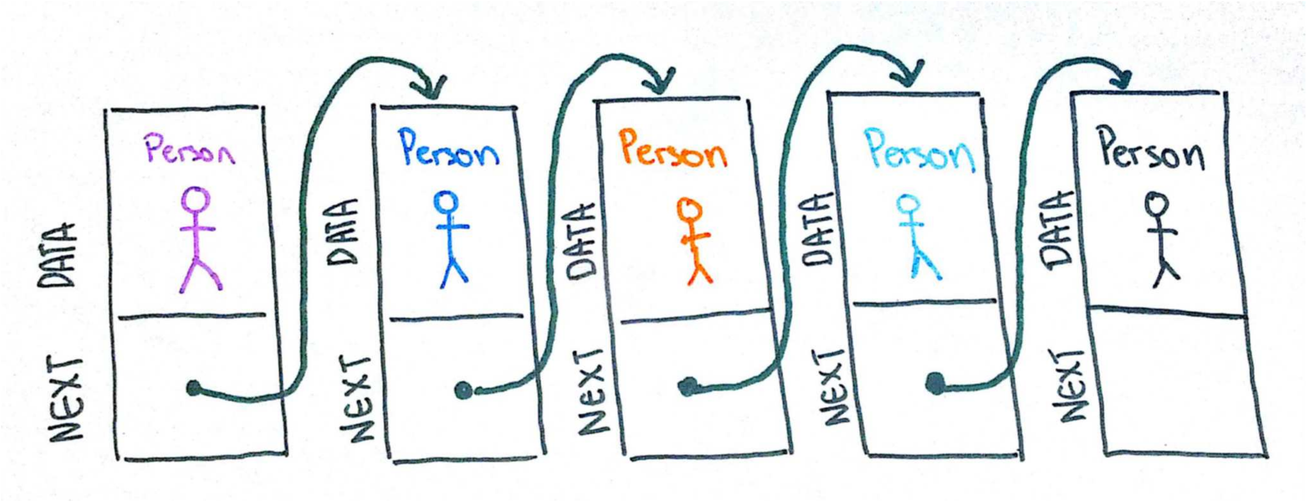
Linked Lists

This is a visual representation of a linked list using the Person objects from the social network



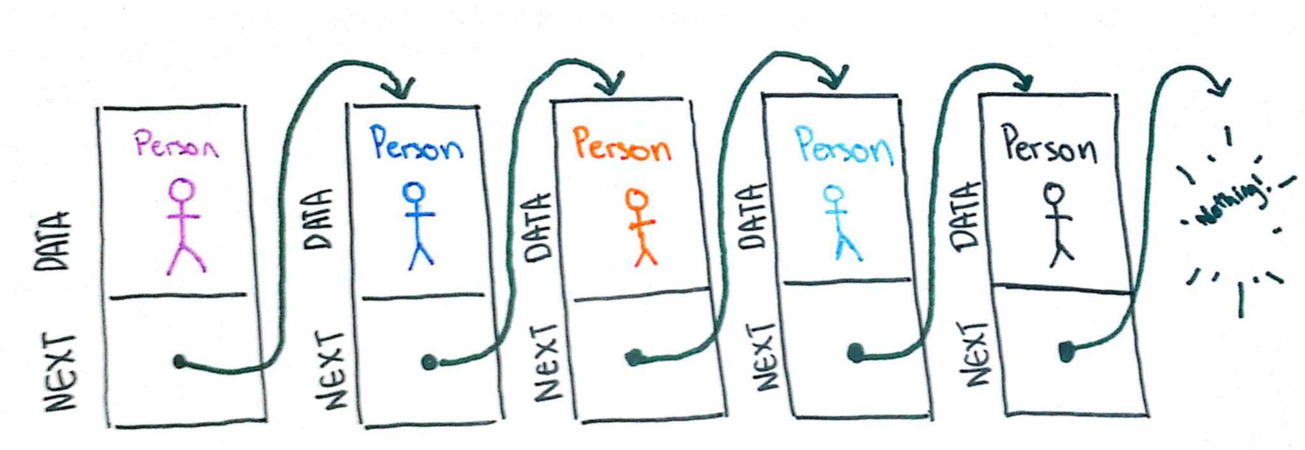
Linked Lists

This is a visual representation of a linked list using the Person objects from the social network



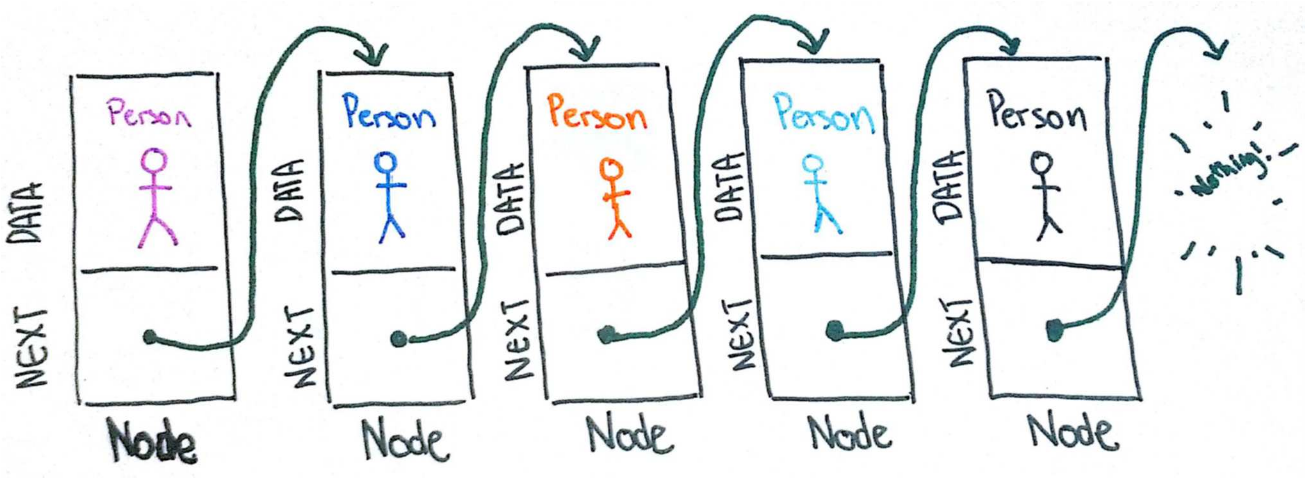
Linked Lists

This is a visual representation of a linked list using the Person objects from the social network



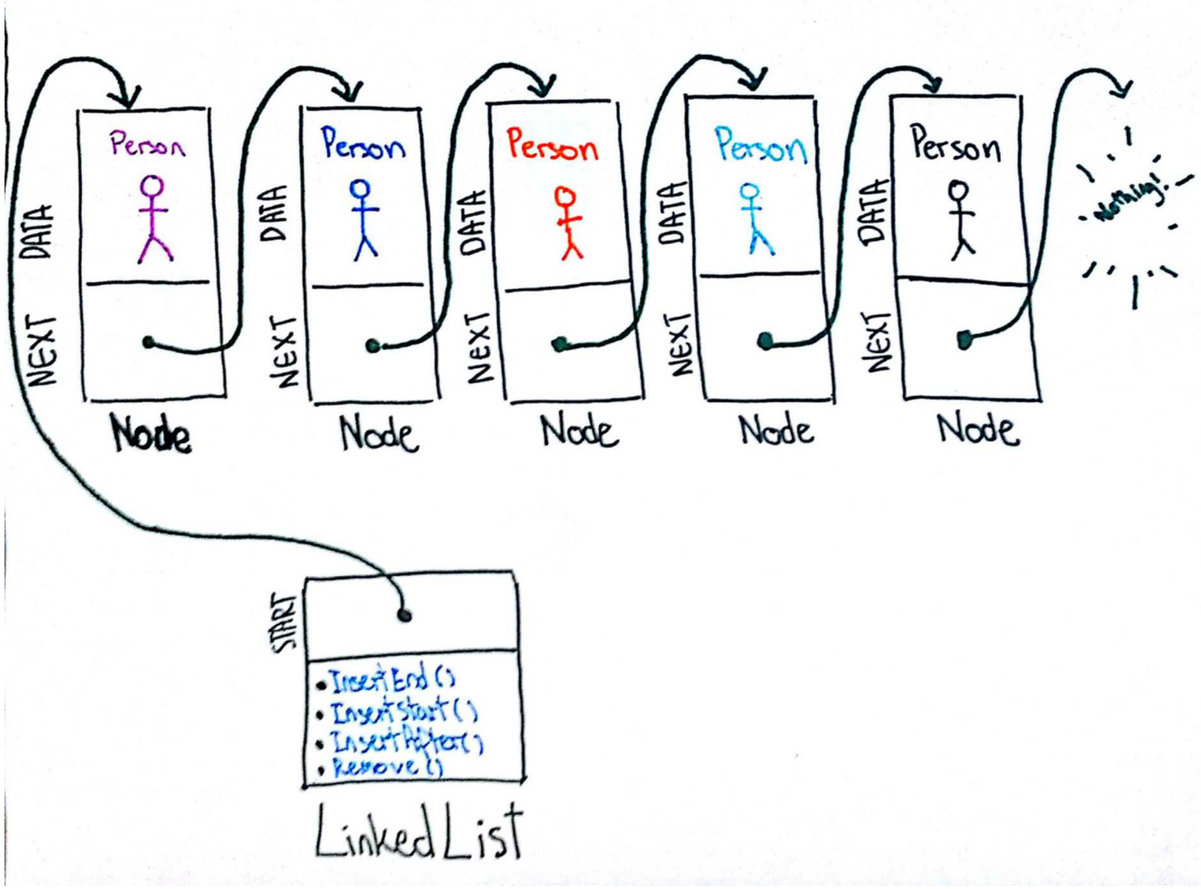
Linked Lists

This is a visual representation of a linked list using the Person objects from the social network



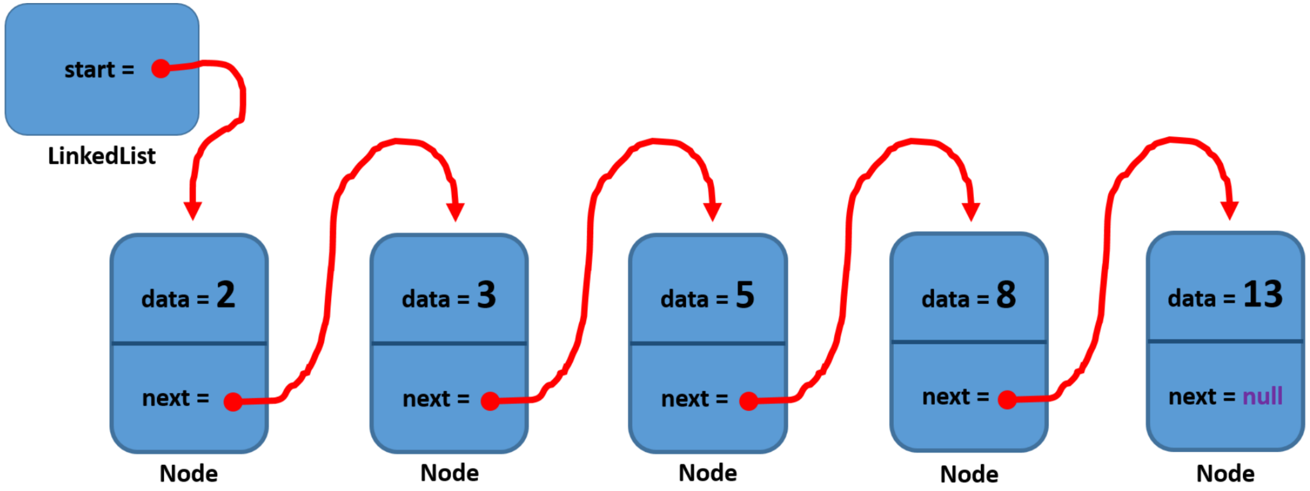
Linked Lists

This is a visual representation of a linked list using the Person objects from the social network



Linked Lists

This is a visual representation of a linked list using `int` values



Going through the components of a LinkedList

We need to define a class for each element in the list: the `Node` class

The `Node` class should have:

- A pointer to some data. We will call it its `value`. For simplicity, we will hold `int` numbers in our LinkedList
- A pointer to the next element in the list. Unsurprisingly, we will call it `next`

```java
    public class Node{
        private int value;
        private Node next;

        // add a constructor here

        // add getter and setter methods here
    }

```

Going through the components of a LinkedList

We need to define a class for doing operations on the list: the `LinkedList` class

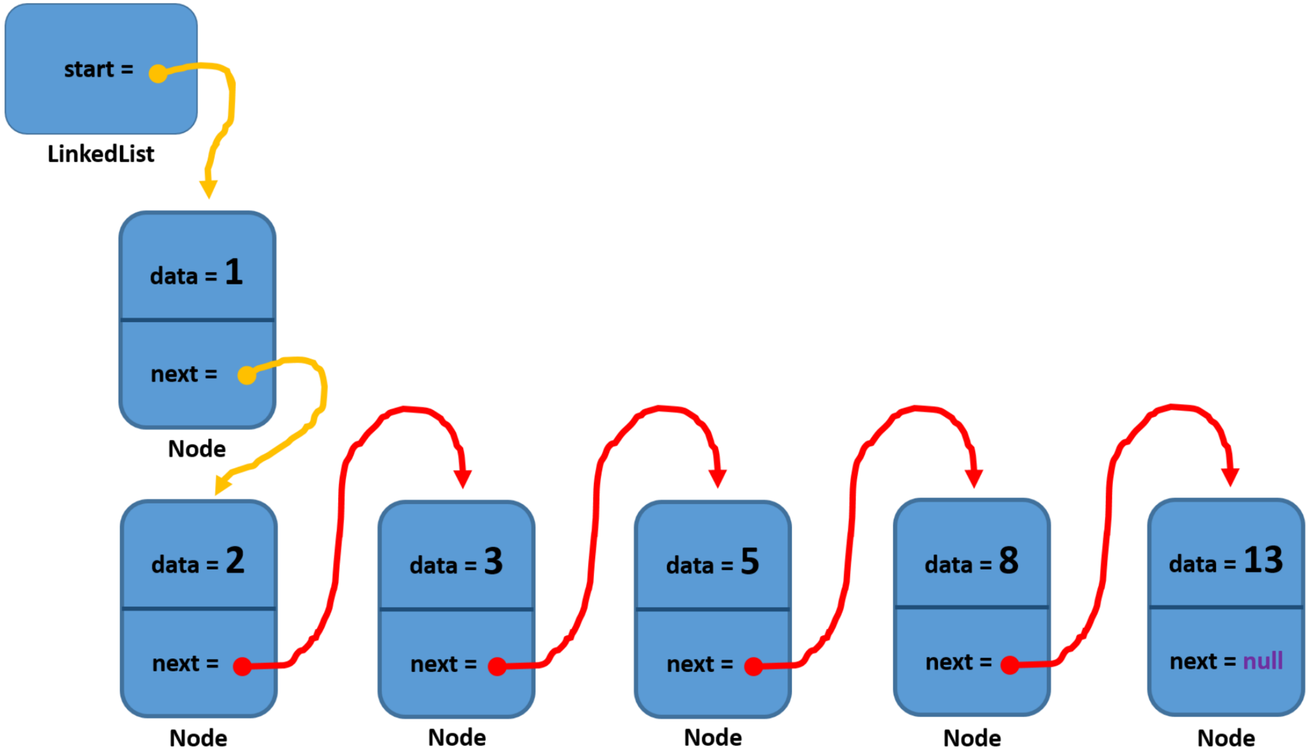
The `LinkedLsit` class should have:

- A pointer to the **first** element in the list. We will call it start.
- All the methods with the operations we want to make on lists

```
    public class LinkedList{
        private Node start;

        // add a constructor here

        // add Insertion, search and deletion methods here
    }

```

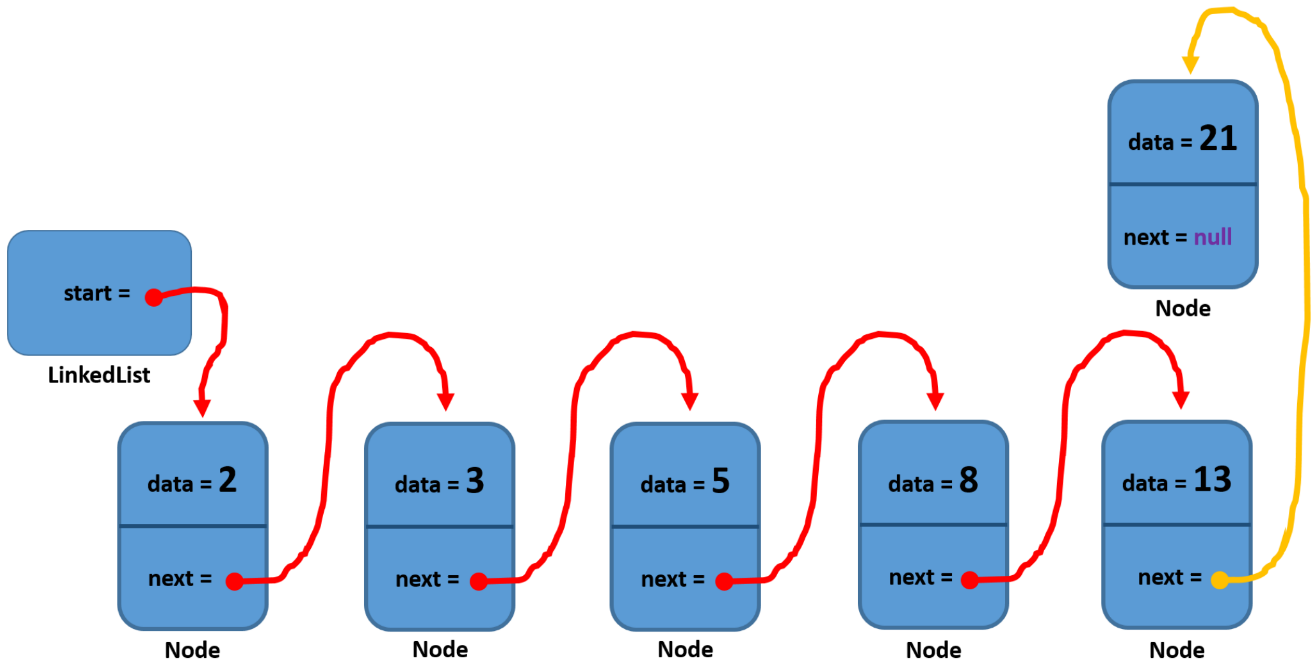
Inserting elements in a linked list

Inserting an element at the beginning



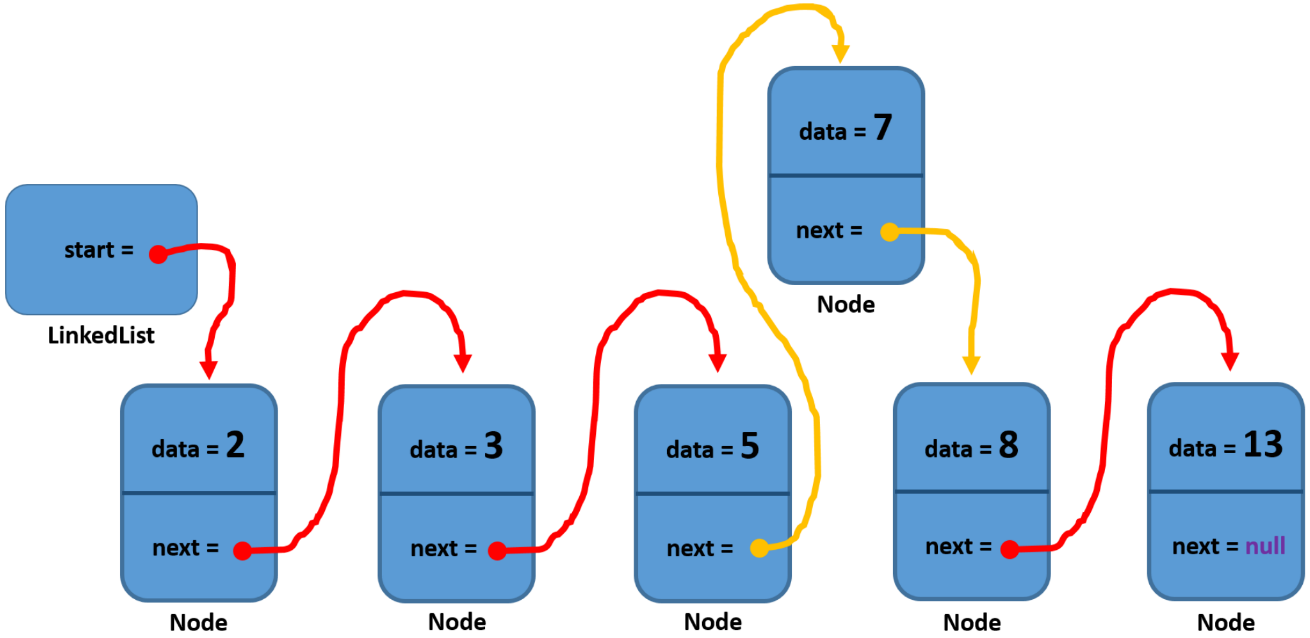
Inserting elements in a linked list

Inserting an element at the end



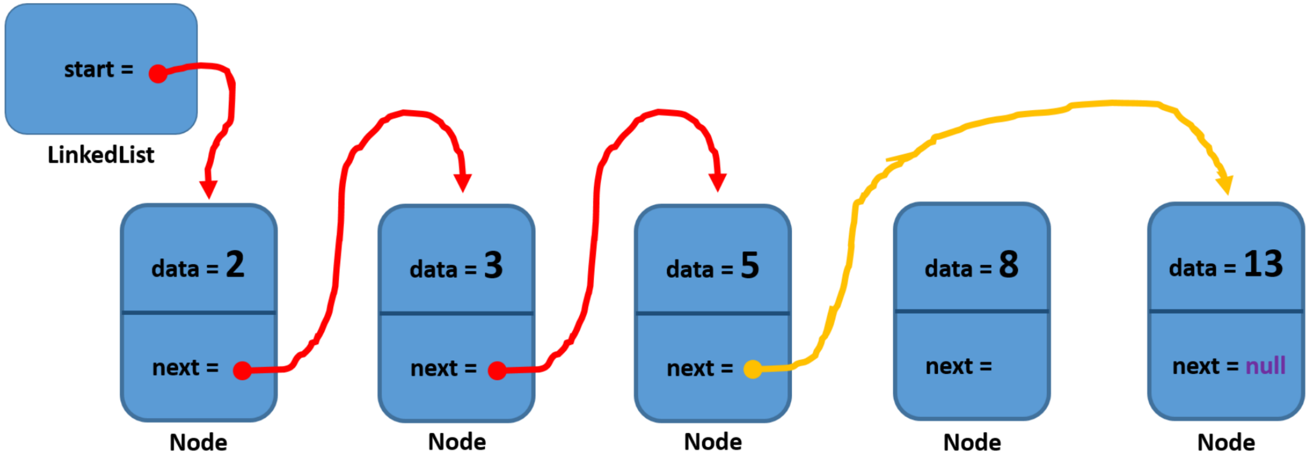
Inserting elements in a linked list

Inserting an element somewhere in the middle



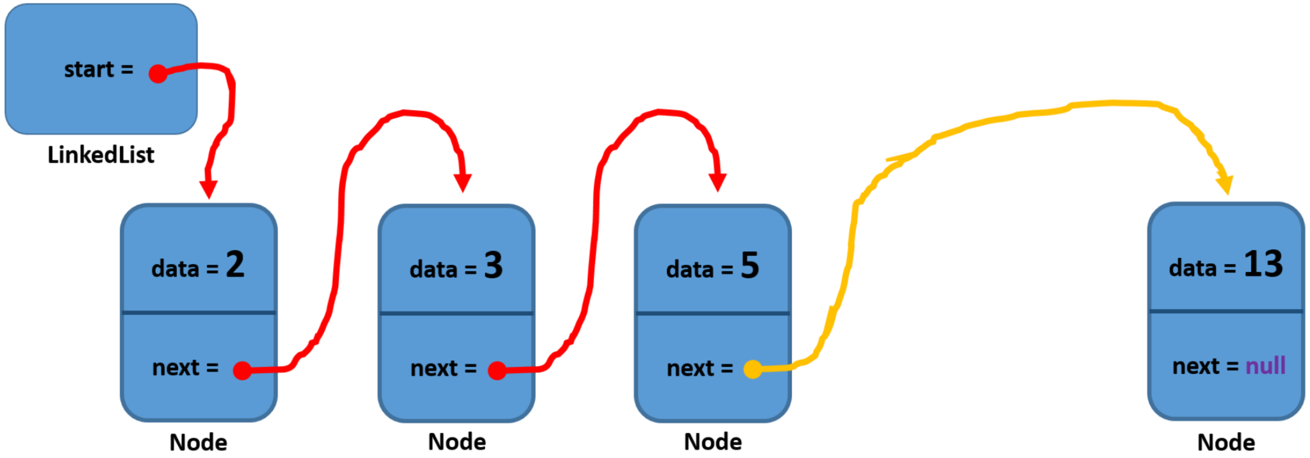
Inserting elements in a linked list

Removing an element from the list



Inserting elements in a linked list

Removing an element from the list



Some remarks about LinkedList

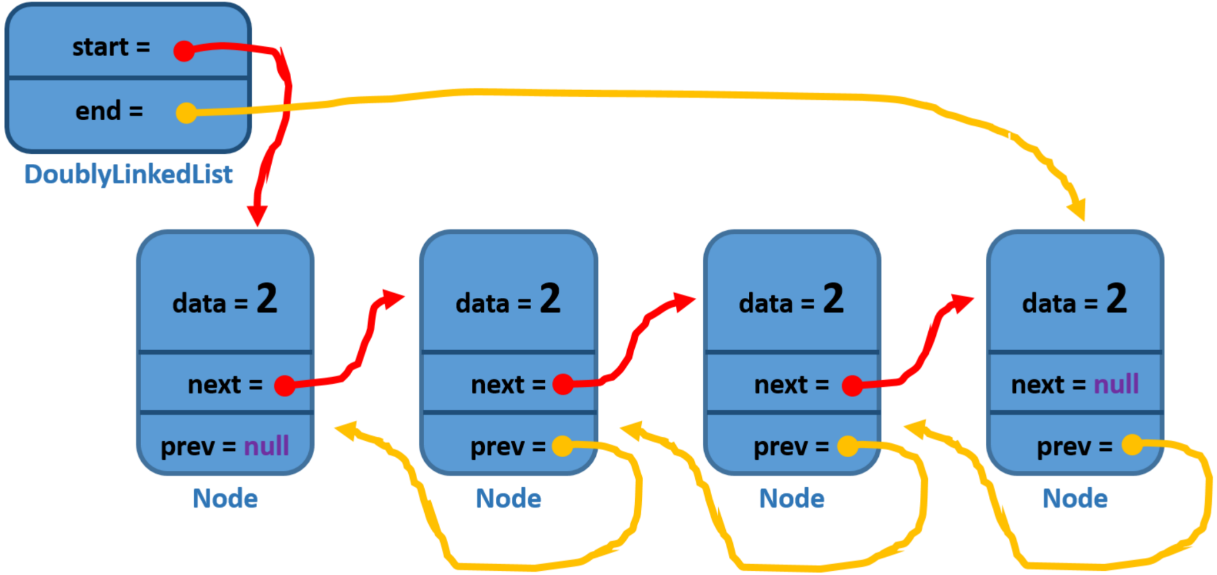
- A `LinkedList` allows us to add or remove elements without having to copy the whole data structure
- In a LinkedList with `n` elements, all operations (search, insertion, deletion) take at most `n` comparisons

In big O notation

Operations in a LinkedList have a running time of ==> O(n)

To try for yourself

- Based on the `LinkedList` and `Node` classes, implement a `DoublyLinkedList`
    - In a doubly linked list, all the nodes have a reference to its **predecessor** : the previous element in the list.

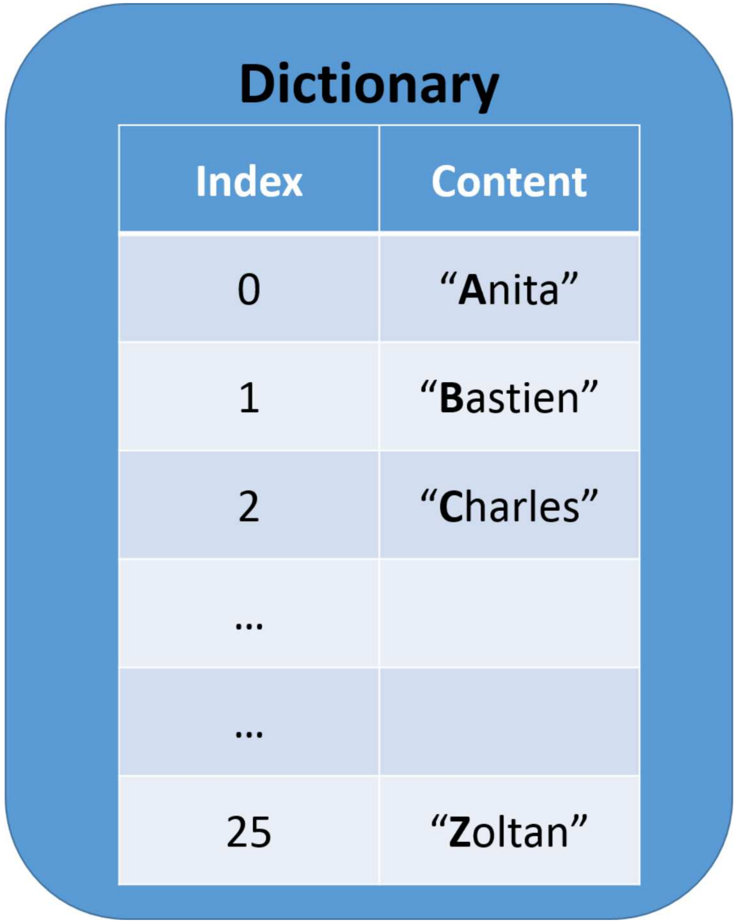


- Implement all the insert, search and delete operations

Hash tables

Hash tables

Hash tables can be viewed as dictionaries



Dictionary

Index	Content
0	“**A**nita”
1	“**B**astien”
2	“**C**harles”
…	
…	
25	“**Z**oltan”

We use a hashing function to determine where an element should go

Implementing a Hashtable

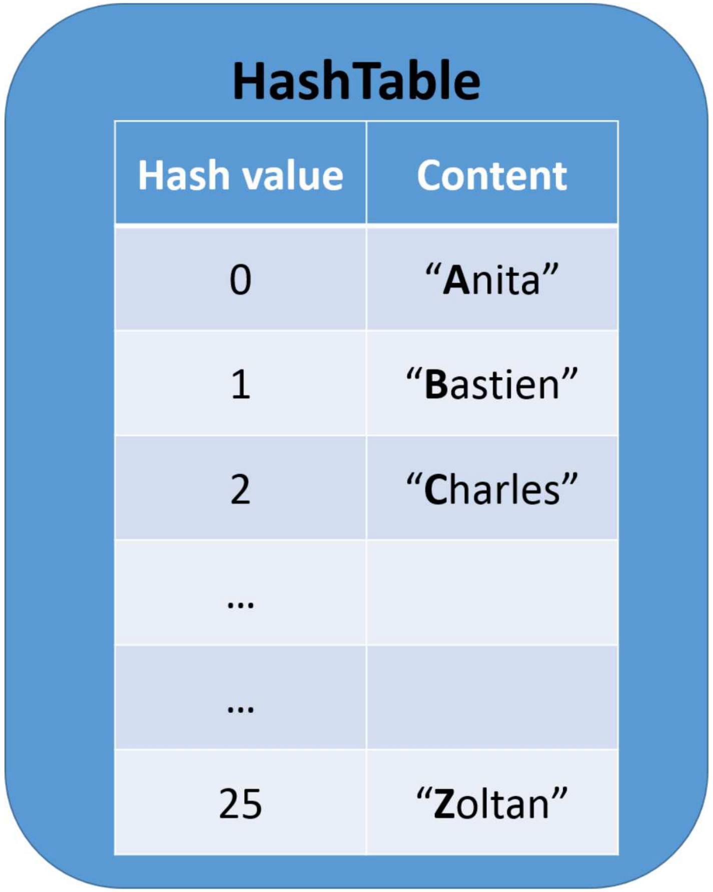
We use an array to store the entries in our dictionary

```
public class Hashtable{
    /*
     * the entries could be of any type
     * here, we use the String entries, for example.
     */
    private String[] entries;

    // add a constructor here

    // add Insertion, search and deletion methods here

    /*
     * The hashing function an index  in the entries
     * array for the given element
     */
    public int hashFunction( String name ){
        // calculate an int ( the hash values)
        // for the String name
    }
}

```

We need to implement the hashing function to determine where an element should go

Hash tables

The hash function maps data to hash values (positions in the array)



HashTable

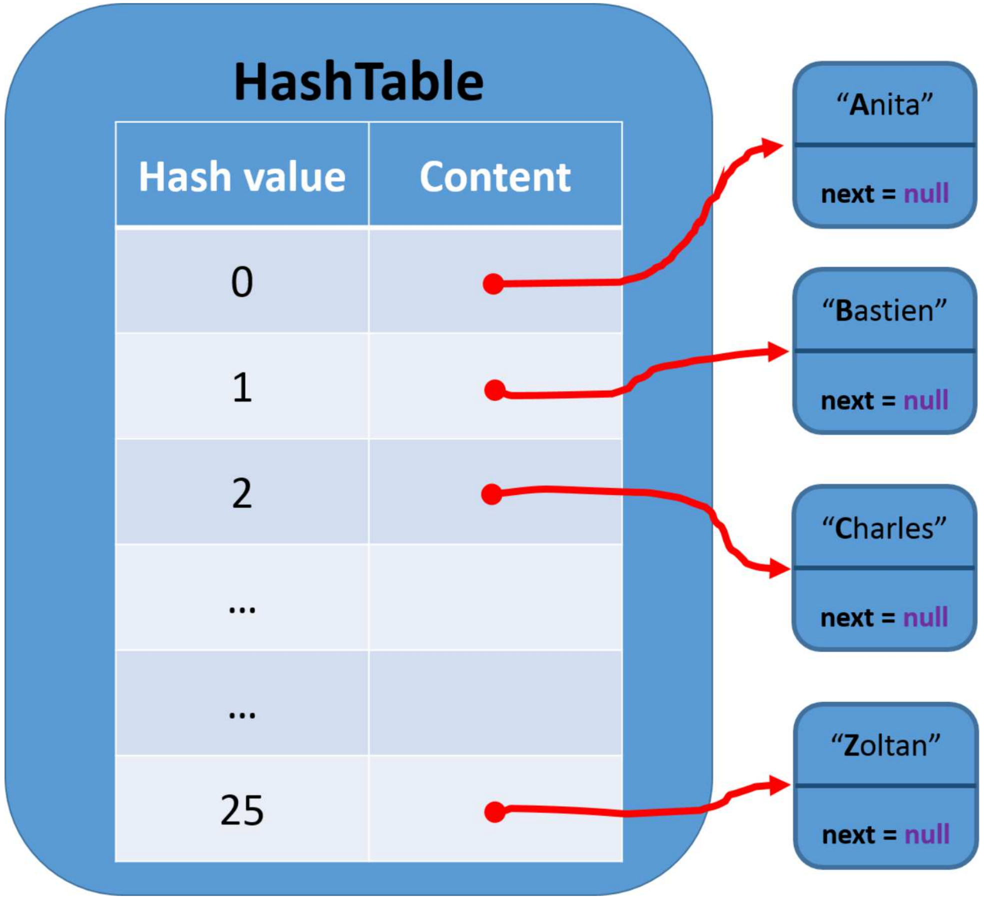
Hash value	Content
0	“**A**nita”
1	“**B**astien”
2	“**C**harles”
…	
…	
25	“**Z**oltan”

In this case, the hashing function maps the first letter of a name to a position in the array

When two or more elements have the same hash value, a collision occurs.

Hash tables with multiple elements per entry

A first step to deal with collisions, is to make each entry point to another data structure; e.g. a LinkedList



HashTable

Hash value	Content
0	
1	
2	
…	
…	
25	

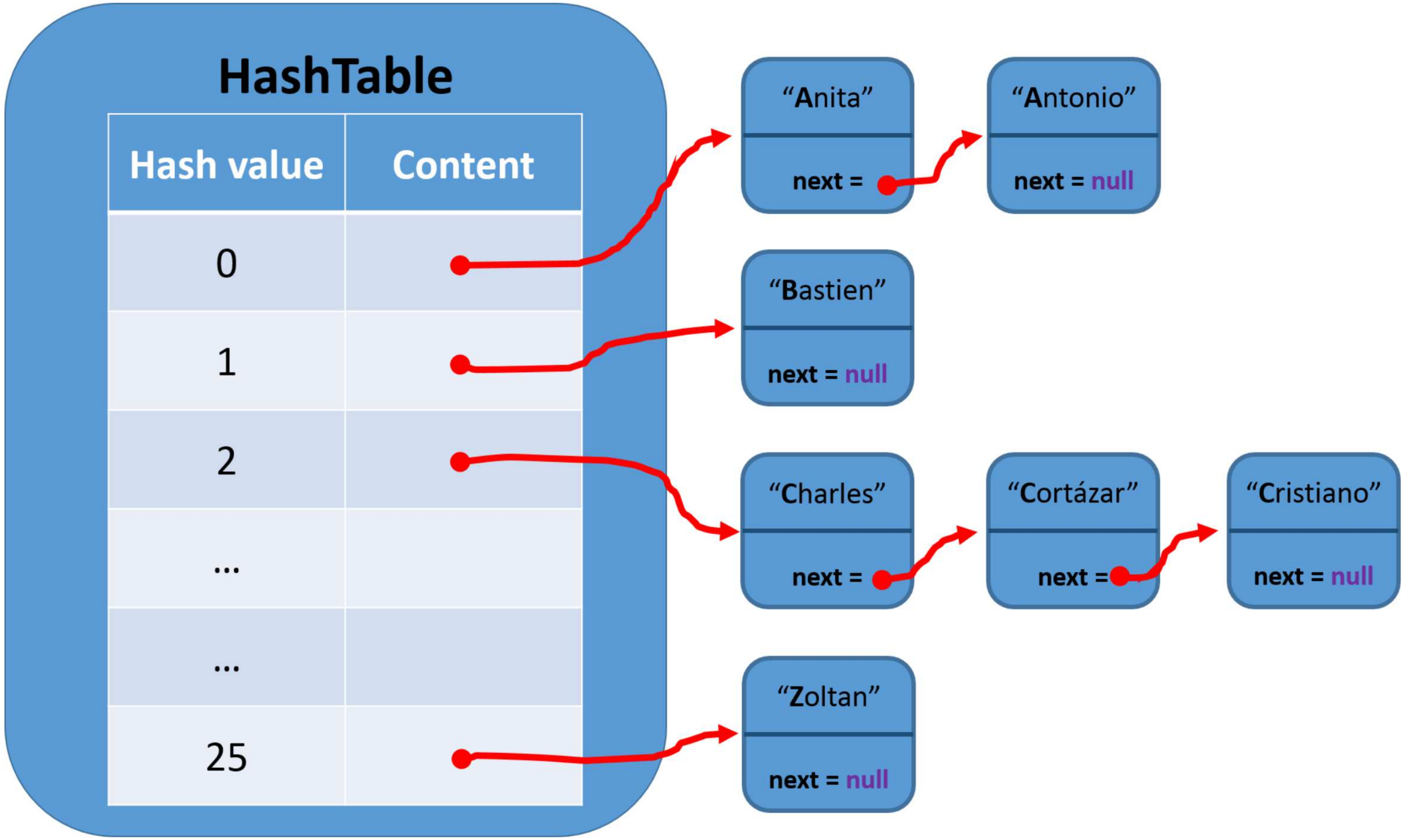
Implementing a Hashtable with LinkedLists for each position in the array

We use an array to store the entries in our dictionary

```java
public class HashtableL{
    /*
     * the entries could be of any type.
     * here, we use a LinkedList to keep
     * multiple entries in each poistion
     * of the array.
     */
    private LinkedList[] entries;

    // add a constructor here

    // add Insertion, search and deletion methods here

    /*
     * The hashing function an index  in the entries
     * array for the given element
     */
    public int hashFunction( String name ){
        // calculate an int ( the hash values)
        // for the String name
    }
}

```

Hash tables with multiple elements per entry

A first step to deal with collisions, is to make each entry point to another data structure; e.g. a LinkedList



Some remarks about Hash tables

- A `Hashtable` allows us to add or remove elements quickly by making use of a **hash function**
- To deal with multiple elements in a single location in the dictionary, we combine the hashing function with a `LinkedList`

In big O notation

Operations in a Hash Table have a running time of ==> $O(k)$, where `k` is the number of elements in each position of the entries array.

Operations in a Hash Table have a running time of ==> $O(1)$

Resources

- Classes and Objects: http://docs.oracle.com/javase/tutorial/java/javaOO/
- The Shoelace Algorithm: http://en.wikipedia.org/wiki/Shoelace_formula
- Suggested reading: How to think like a Computer Scientist, Chapter 11

← →

1 / 41

Go to slide: Go

Drawing Tools

□□□
□