Linked Lists & Hash Tables

John Rodriguez

Linked Lists

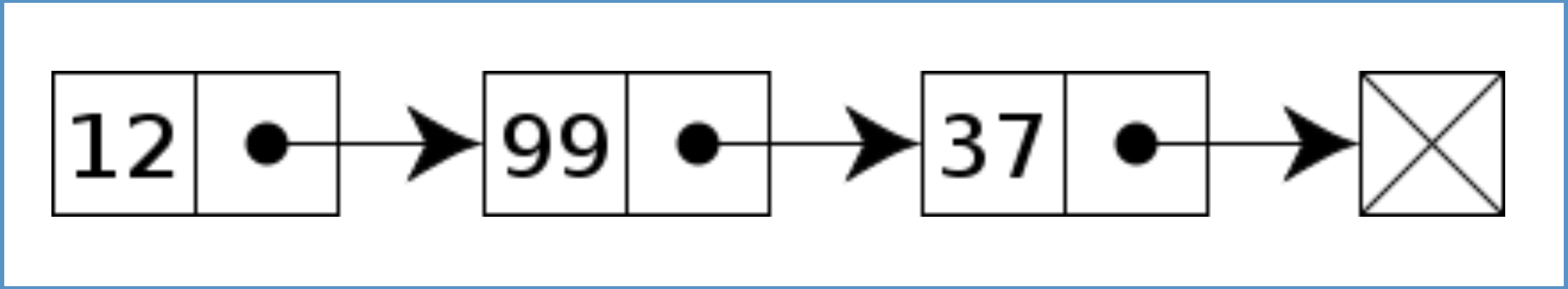
Linked Lists

- a data structure consisting of a group of nodes which together represent a sequence
- Simplest form: each node is composed of data and a reference (aka “link”) to the next node in the sequence
- More complex variants add additional links

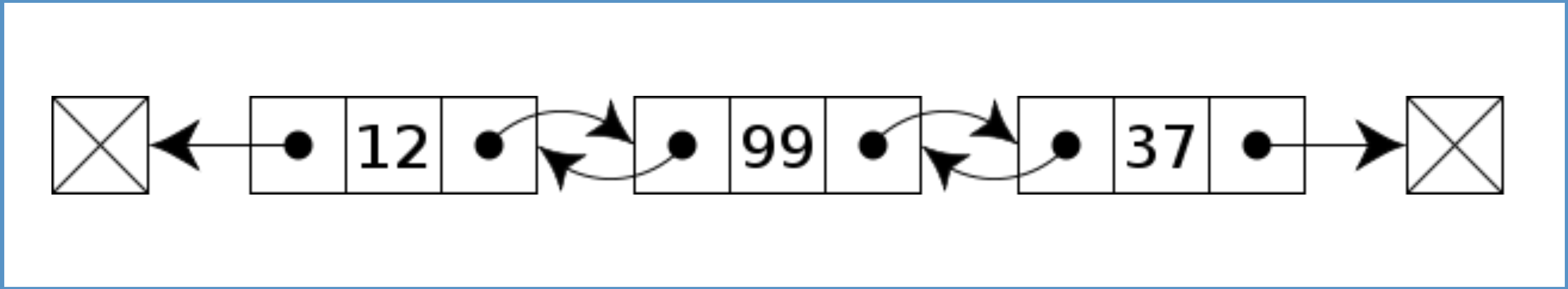
Types of Linked Lists

- Singly-Linked
- Doubly-Linked
- Circular Linked

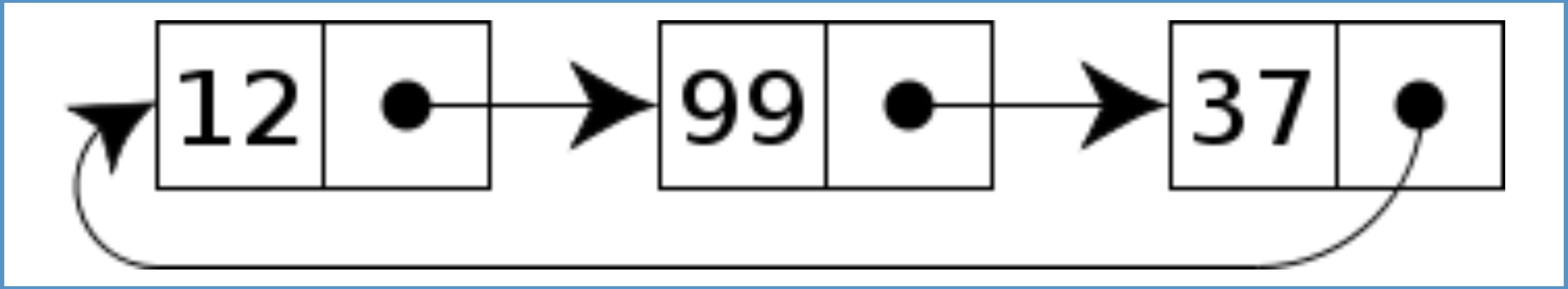
Singly-Linked



Doubly-Linked



Circularly-Linked



Operations

- 1. Traversing a linked list.
- 2. Appending a new node (to the end) of the list
- 3. Prepending a new node (to the beginning) of the list
- 4. Inserting a new node to a specific position on the list
- 5. Deleting a node from the list
- 6. Updating the data of a linked list node

Node definition

```
public class Node {
  Node next;
  String data;
}
```

Node definition

```java
public class Node<E> {
  Node<E> next;
  E data;
}
```

Node definition

```java
public class Node {
  Node next;
  String data;
}
```

Node definition

```
public class Node {
  Node next;
  String data;

  public Node getNext() {
    return next;
  }

  public String getData() {
    return data;
  }
}
```

Node definition

```
public class Node {
  Node next;
  String data;
}
```

Advantages

- Linked lists are a dynamic data structure, allocating the needed memory while the program is running
- Insertion and deletion node operations are easily implemented in a linked list
- Linear data structures such as stacks and queues are easily executed with a linked list
- Expand in real time without memory overhead

Disadvantages

- They have a tendency to use more memory due to pointers requiring extra storage space
- Nodes in a linked list must be read in order from the beginning as linked lists are inherently sequential access
- Nodes are not stored contiguously, greatly increasing the time required to access individual elements within the list
- Difficulties arise in linked lists when it comes to reverse traversing. For instance, singly linked lists are cumbersome to navigate backwards and while doubly linked lists are somewhat easier to read, memory is wasted in allocating space for a back pointer.

Applications

- Stacks
- Queues
- Object Pooling (Android's *Message* pool)
  - http://grepcode.com/file/repository.grepcode.com/java/ext/com.google.android/android/5.1.1_r1/android/os/Message.java?av=f#106
- Linux process "task queue"
  - http://lxr.free-electrons.com/source/kernel/sched.c?v=2.6.36#L3831

Hash Tables

Hash Tables

- a hash table is a data structure used to implement a collection of (key, value) pairs
- each key appears only once in the collection
- other common names
  - map
  - dictionary
  - associative array

Operations

- insert
- replace
- delete
- find

Array

```
public void someMethod() {

  String[] stringArray = new String[5];

  stringArray[0] = "foo";  // 0 -> "foo"
  stringArray[2] = "bar";  // 2 -> "bar"

  String s1 = stringArray[0];  // "foo"
  String s2 = stringArray[1];  // null

  //...
}
```

Map

```java
public void someMethod() {

  Map<Integer, String> stringMap = new HashMap<>();

  stringMap.put(0, "foo");  // 0 -> "foo"
  stringMap.put(2, "bar");  // 2 -> "bar"

  String s1 = stringMap.get(0);   // "foo"
  String s2 = stringMap.get(1);   // null

  //...
}
```

Map

```
public void someMethod() {

  Map<String, String> stringMap = new HashMap<>();

  stringMap.put("key1", "foo");  // "key1" -> "foo"
  stringMap.put("key2", "bar");  // "key2" -> "bar"

  String s1 = stringMap.get("key1");   // "foo"
  String s2 = stringMap.get("key3");   // null

  //...
}
```

??

```
// "key1" -> "foo"
// "key2" -> "bar"
```

Q: How does an Object (in this case, a String) convert to a number?

A: using a hash function!

Hash Function

- A hash table uses a ***hash function*** to compute an index (***hash***) into an array of ***buckets*** or ***slots***, from which the desired value can be found
- Ideally, the hash function will assign each key to a unique bucket (***perfect hash function***)
- ….but it is possible that two keys will generate an identical hash causing both keys to point to the same bucket!
- So… a hash table should assume that hash ***collisions*** — different keys that are assigned by the hash function to the same bucket — will occur and must be accommodated in some way

Object.hashcode()

- Java’s built-in hash function

- if you want to use a custom Object in a HashMap, you **MUST** override this **and** equals()

Collision Resolution

- Open Addressing
- Separate Chaining

Collision Resolution

- **Open Addressing**
- Separate Chaining

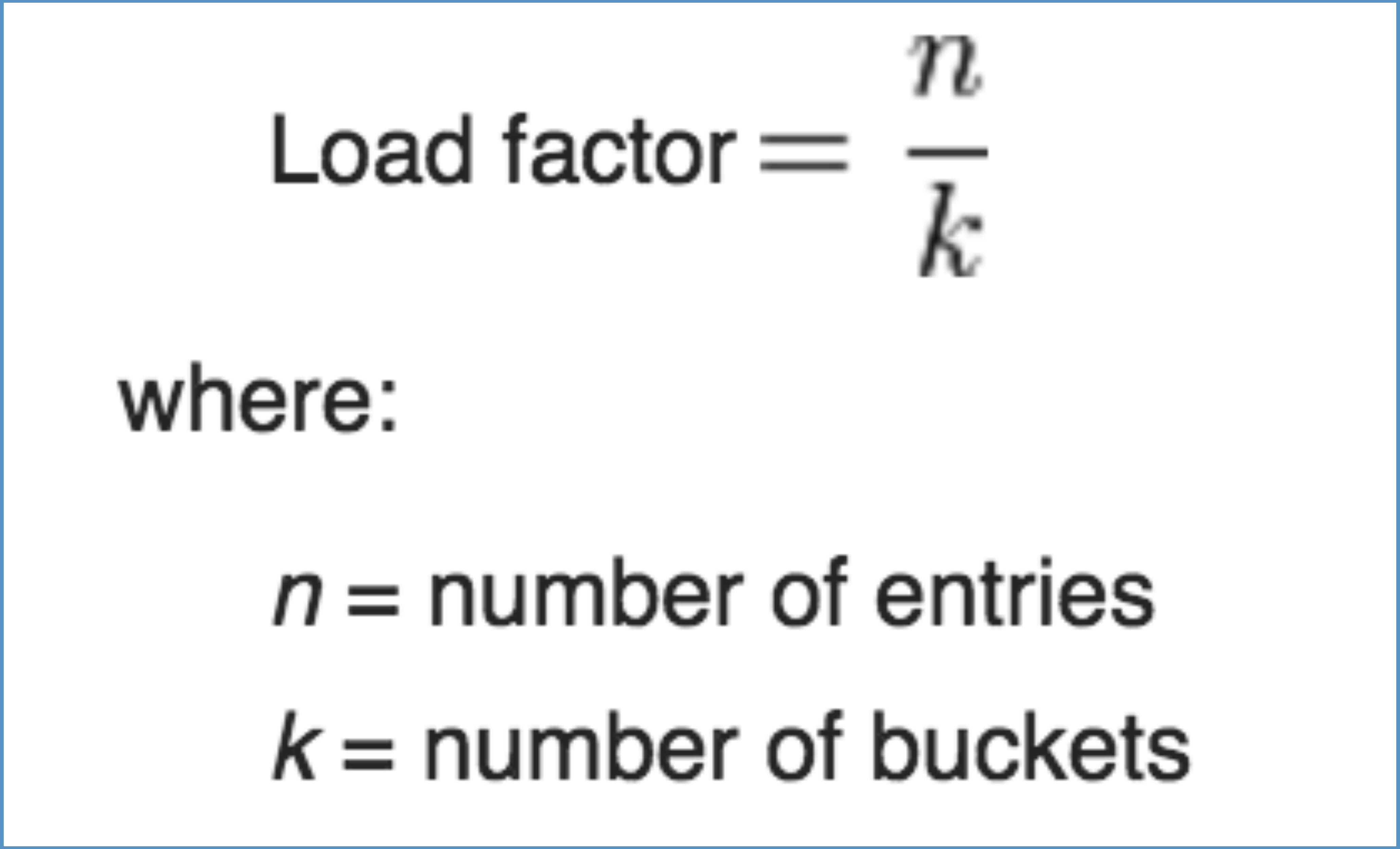
Open Addressing

- Linear Probing
- Double Hashing

Collision Resolution

- Open Addressing
- **Separate Chaining**

Load Factor



$$\text{Load factor} = \frac{n}{k}$$

where:

n = number of entries

k = number of buckets

Rehashing

- When the load factor of the hash table is too high, the hash table is not as effective
    - O(1) lookup is no longer guaranteed…

- In this case, we'll need a bigger table, iterate over the previous table, and insert the (key, value) pairs into the new table, using the new hash function
    - this takes time…and should occur as ***little*** as possible
    - O(N) cost is ***amortized*** so that it can be ignored

Analysis

- In a good hash table, the average cost for each lookup is independent of the number of elements stored in the table
- Many hash table designs also allow arbitrary insertions and deletions of key-value pairs, at (amortized) constant average cost per operation

Applications

- O(1) lookup!   Use it for everything!
- more efficient than search trees or any other table lookup structure
- database indexing
- caches (browser, DNS, ARP caches)
- sets (HashSet)
    - Gmail, DropBox attachments

Exercise

Given a list of integers, i.e., {1, 1, 1, 1, 2, 2, 3, 3, 5}, count how many times a given integer occurs, e.g.,

{1, 4}, {2, 2}, {3, 2}, {5, 1}

Homework

- Linked Lists
  - create a circularly-linked list
    - copy SinglyNode and SinglyList from GitHub
  - implement 1) addToHead, 2) addToTail, 3) delete, and 4) insertAfter

- Hash Tables
  - Given two strings, check if they're anagrams or not. Two strings are anagrams if they are written using the same exact letters, ignoring space, punctuation and capitalization. Each letter should have the same count in both strings. For example, 'eleven plus two' and 'twelve plus one' are meaningful anagrams of each other.

Exit Ticket

- See Slack channel