Jgi cb Jgqrq

Data Structures
Week #2

Practice

- Go over Geeks or Geeks Singly Linked Lists

  https://www.geeksforgeeks.org/data-structures/linked-list/singly-linked-list/

- Practice implementing a Singly Linked List on Leet Code

  https://leetcode.com/problems/design-linked-list/

What is a Linked List?

What is a Linked List?

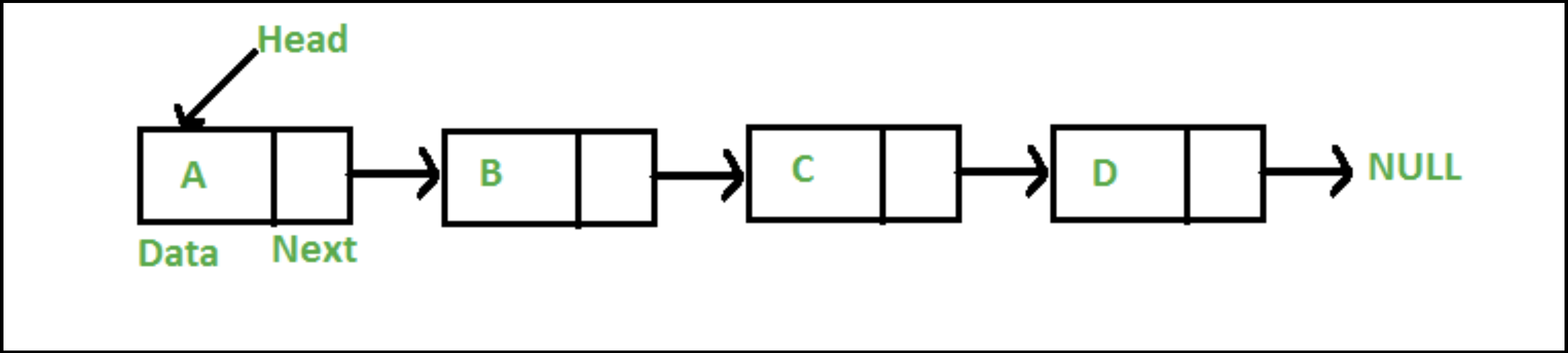
Individual elements (nodes) that each hold two pieces of information:

What is a Linked List?

Individual elements (nodes) that each hold two pieces of information:

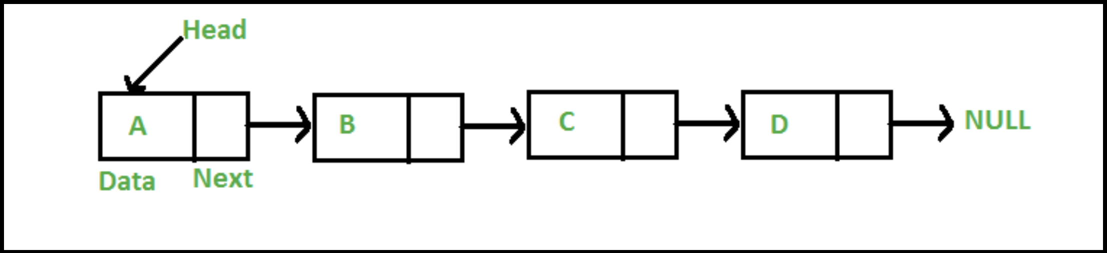
1) data (integer, string, anything!)

2) reference to the next node


Head
A
B
C
D
NULL
Data
Next

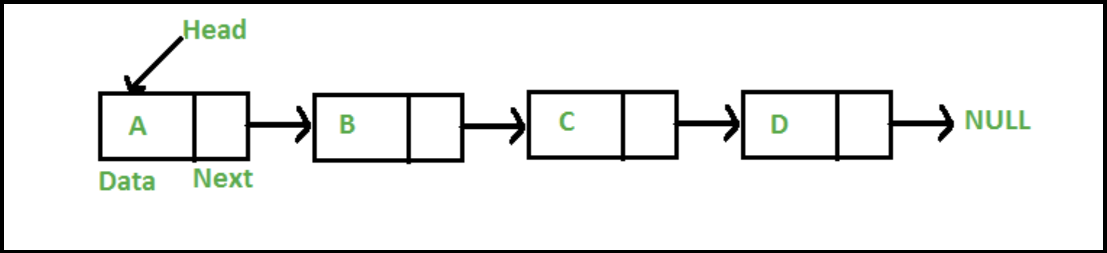
```java
class LinkedListNode {
    String data;
    LinkedListNode next;

    LinkedListNode(String data) {
        this.data = data;
    }
}
```

```java
class LinkedListNode {
    String data;
    LinkedListNode next;

    LinkedListNode(String data) {
        this.data = data;
    }
}
```



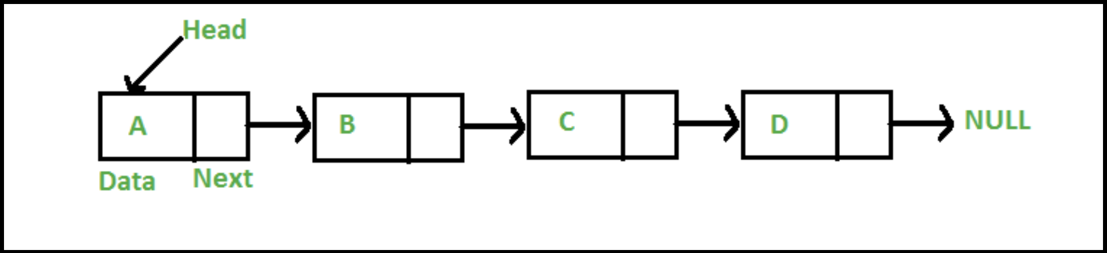
```java
LinkedListNode head = new LinkedListNode("a");
```

```
class LinkedListNode {
    String data;
    LinkedListNode next;

    LinkedListNode(String data) {
        this.data = data;
    }
}
```



```
LinkedListNode head = new LinkedListNode("a");
LinkedListNode n1 = new LinkedListNode("b");
LinkedListNode n2 = new LinkedListNode("c");
LinkedListNode n3 = new LinkedListNode("d");
```

```
class LinkedListNode {
    String data;
    LinkedListNode next;

    LinkedListNode(String data) {
        this.data = data;
    }
}
```



```
LinkedListNode head = new LinkedListNode("a");
LinkedListNode n1 = new LinkedListNode("b");
LinkedListNode n2 = new LinkedListNode("c");
LinkedListNode n3 = new LinkedListNode("d");

head.next  = n1;
n1.next = n2;
n2.next = n3;
```

Why are they useful?

Why are they useful?

Web browser’s history (previous / next page)

Why are they useful?

Web browser's history (previous / next page)

Memory management (blocks of memories)

Why are they useful?

Web browser’s history (previous / next page)

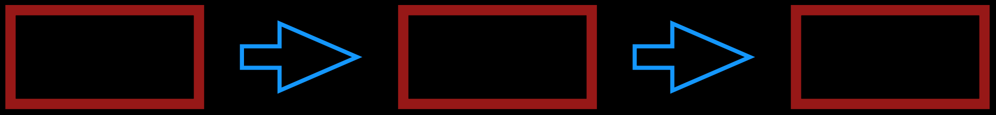
Memory management (blocks of memories)

Hash tables (resolving collisions)

Types of Linked Lists

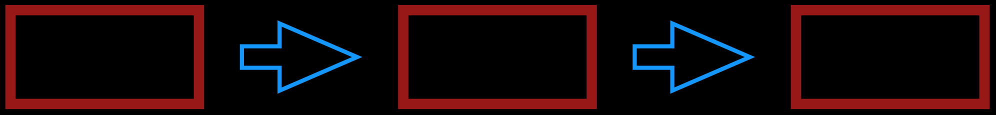
Types of Linked Lists

Singly linked

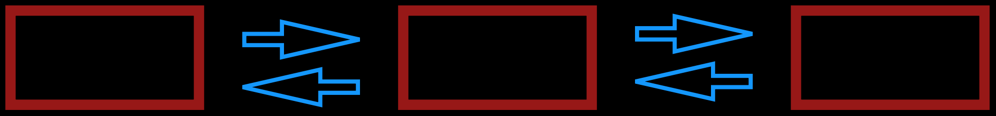


Types of Linked Lists

Singly linked

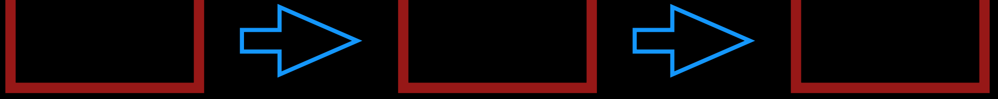


Doubly linked

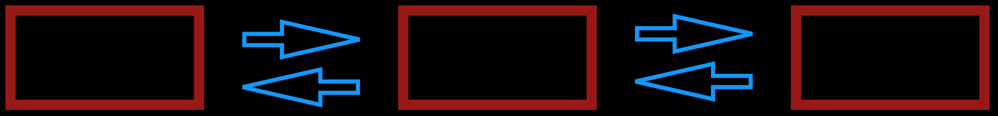


Types of Linked Lists

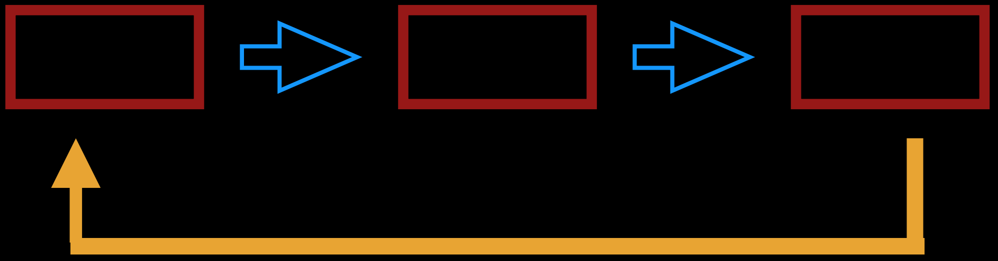
Singly linked



Doubly linked

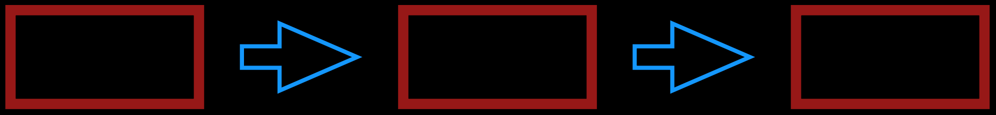


Circularly linked

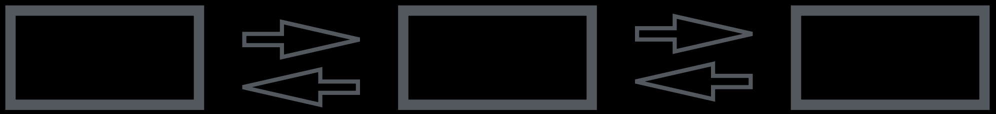


Types of Linked Lists

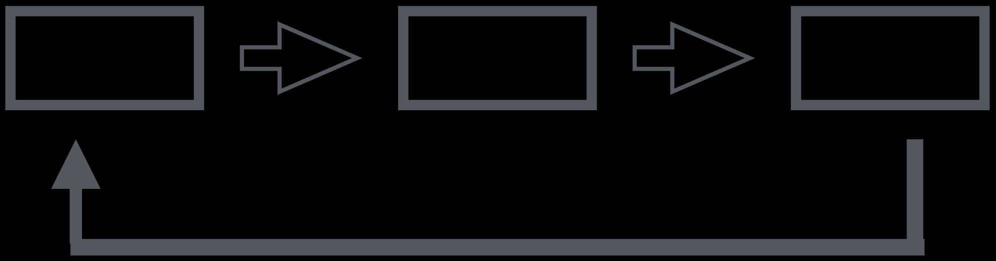
Singly linked



Doubly linked



Circularly linked



Pros and Cons

Always think about space / time complexity when choosing a data structure!

Pros and Cons

Always think about space / time complexity when choosing a data structure!

Good for:
- insertion / deletion from one end

Pros and Cons

Always think about space / time complexity when choosing a data structure!

Good for:
- insertion / deletion from one end

Bad for:
- searching

Pros and Cons

Always think about space / time complexity when choosing a data structure!

Good for:
- insertion / deletion from one end

Bad for:
- searching

https://www.geeksforgeeks.org/linked-list-set-1-introduction/

Linked List Tips and Tricks

Linked List Tips and Tricks

Take multiple passes through the linked list

- get length
- save other information about contents

Linked List Tips and Tricks

Take multiple passes through the linked list
- get length
- save other information about contents

Two pointers
- ‘race car’ strategy with one regular pointer, and one fast pointer

Linked List Tips and Tricks

Take multiple passes through the linked list
- get length
- save other information about contents

Two pointers
- ‘race car’ strategy with one regular pointer, and one fast pointer

Dummy node
- helpful for preventing errors when returning ‘head’

Understand

Match

Pseudocode / Plan

Implement

Review

Evaluate

Understand

Match

Pseudocode / Plan

Implement

Review

Evaluate

Understand

If you don’t understand the problem completely, you can’t solve it correctly

Understand

If you don’t understand the problem completely, you can’t solve it correctly

You are often given vague questions to test your ability to gather requirements

Understand

Asking clarifying questions

Understand

Asking clarifying questions

- is `x` input possible?

Understand

Asking clarifying questions

- is `x` input possible?
- will the input always be sorted?

Understand

Asking clarifying questions

- is `x` input possible?
- will the input always be sorted?
- are there any space and time constraints?

Understand

Asking clarifying questions

- is `x` input possible?
- will the input always be sorted?
- are there any space and time constraints?
- given `x` input, is the expected output `y`?

Understand

Match

Pseudocode / Plan

Implement

Review

Evaluate

Match

Match

Think about what data structures might be helpful for the problem (hash map, queue, binary search, …)

Match

Think about what data structures might be helpful for the problem (hash map, queue, binary search, …)

See if there are any specific techniques that you can apply (ie : using a dummy node, or taking multiple passes)

Plan / Pseudocode

Start to figure out how you would solve the problem

Plan / Pseudocode

Start to figure out how you would solve the problem

Can you create any ‘magic’ helper methods that would simplify the solution? (ie : getLength(), reverse())

Plan / Pseudocode

Start to figure out how you would solve the problem

Can you create any ‘magic’ helper methods that would simplify the solution? (ie : getLength(), reverse())

Talk through different approaches you can take, and their tradeoffs

Plan / Pseudocode

Start to figure out how you would solve the problem

Can you create any ‘magic’ helper methods that would simplify the solution? (ie : getLength(), reverse())

Talk through different approaches you can take, and their tradeoffs

Be able to verbally describe your approach and explain how an example input would produce the desired output

Understand

Match

Pseudocode / Plan

Implement

Review

Evaluate

Implement

Implement

Try to spent ~15 minutes on a problem

Implement

Try to spent ~15 minutes on a problem

Goal for each problem is to have it pass a simple test case

Understand

Match

Pseudocode / Plan

Implement

Review

Evaluate

Review

Review

Run through your solution with test cases

- catch edge cases
- fix potential bugs

Evaluate

Evaluate

Analyze the time / space complexity of your final solution

Evaluate

Analyze the time / space complexity of your final solution

If applicable, discuss any ‘shortcuts’ or tradeoffs that you made

Understand

Match

Pseudocode / Plan

Implement

Review

Evaluate

Let’s discuss the problems together!

Questions?