Week 4 - Friday

COMP 2100

Last time

- What did we talk about last time?
- Doubly linked list implementation

Questions?

Project 1

Bitmap Manipulator

Linked List Stack

Stack Implementations

- Dynamic array
  - Advantages: pop and top are O(**1**)
  - Disadvantages: limited size, making push O(***n***) in the worst case (still O(1) amortized)
- Linked list
  - Advantages: push, pop, and top are O(**1**)
  - Disadvantages: slightly slower than the array version, considerably more memory overhead

Linked list implementation

```
public class ListStack<T> {
  private static class Node<T> {
    public T data;
    public Node<T> next;
  }

  private Node<T> top = null;
  private int size = 0;

  public void push(T value) {}
  public T pop() {}
  public T peek() {} // instead of top
  public int size() {}
}
```

Linked List Push

Linked List Pop

Linked List Peek

Linked List Size

Queue Implementations

- Circular array
  - Advantages: dequeue and front are O(**1**)
  - Disadvantages: limited size, making enqueue O(***n***) in the worst case (still O(1) amortized)
- Linked list
  - Advantages: enqueue, dequeue, and front are O(**1**)
  - Disadvantages: slightly slower than the array version, considerably more memory overhead

Linked list implementation

```
class ListQueue<T> {
  private class Node<T> {
    public T data;
    public Node<T> next;
  }
  private Node<T> head = null;
  private Node<T> tail = null;
  private int size = 0;

  public void enqueue(T value) {}
  public T dequeue() {}
  public T front() {}
  public int size() {}
}
```

Linked List Front

Linked List Size

Linked List Enqueue

Linked List Dequeue

Review

Week 1

- Programming model
- Java
  - OOP
  - Polymorphism
  - Interfaces
  - Exceptions
  - Generics
- Java Collections Framework

Week 2

- Big Oh Notation
  - Formal definition: $f(n)$ is $O(g(n))$ if and only if
    - $f(n) \leq c \cdot g(n)$ for all $n > N$
    - for **some** positive real numbers c and N
  - Worst-case, asymptotic, upper bound of running time
  - Ignore lower-order terms and constants
- Big Omega and Big Theta
- Abstract Data Types
- Array-backed list

Week 3

- Stacks
  - FILO data structure
  - Operations: push, pop, top, empty
  - Dynamic array implementation
- Queues
  - FIFO data structure
  - Operations: enqueue, dequeue, front, empty
  - Circular (dynamic) array implementation
- JCF implementations: **`Deque<T>`** interface
  - **`ArrayDeque<T>`**
  - **`LinkedList<T>`**

Week 4

- Linked lists
  - Performance issues
  - Single vs. double
  - Insert, delete, find times
- Linked list implementation of stacks
- Linked list implementation of queues

Sample Problems

Running time

- Let ***M*** and ***N*** be two integers, where ***M*** is no larger than ***N***
- Use Big Θ notation to give a tight upper bound, in terms of ***N***, on the time that it will take to
  - Add ***M*** and ***N*** (by hand, using the normal algorithm)
  - Multiply ***M*** and ***N*** (by hand, using the normal algorithm)
- Use Big Θ notation to give the same bounds but this time in terms of ***n***, where ***n*** is the number of digits in ***N***

What's the running time in Θ?

```
int end = n;
int count = 0;
for (int i = 1; i <= n; i++) {
  end /= 2;
  for (int j = 1; j <= end; j++) {
    count++;
  }
}
```

What's the running time in Θ?

```
int end = n;
int count = 0;
for (int i = 1; i <= n*n; i+=2) {
    count++;
}
```

What's the running time in Θ?

```
int count = 0;
for (int i = 1; i <= n; i++) {
  for (int j = 1; j <= n/j; j++) {
    count++;
  }
}
```

Lighten

- If we increase the R, G, and B values of every pixel by 25%, the image will get lighter
- Let **`Color`** be the following:

```
public class Color {
    public int red;
    public int green;
    public int blue;
}
```

- Let **`pixels`** be a **`Color[][]`** array with **`height`** rows and **`width`** columns
- Write the code to lighten the image by 25% (by multiplying by 1.25)
- Don't forget to round the results before storing them back into each color component

Reverse a linked list

Assume the following:

```
public class List {
  private static class Node {
    public int data;
    public Node next;
  }

  private Node head = null;
  …
}
```

Write a method in **`List`** that reverses the linked list.

Code to reverse a linked list

```
public void reverse() {
  if (head != null) {
     Node reversed = head;
     Node temp = head;
     Node rest = head.next;
     temp.next = null;
     while(rest != null) {
           temp = rest;
           rest = rest.next;
           temp.next = reversed;
           reversed = temp;
     }
     head = reversed;
  }
}
```

Palindrome

- Write a method that takes a **`CharBuffer`** object
- The **`CharBuffer`** object has two methods:
  - **`nextChar()`** which extracts a char from the input stream
  - **`hasNextChar()`** which returns true as long as there is another char to extract
- The method should return **`true`** if the input stream is a palindrome (the same backwards as forwards) and **`false`** otherwise
- Use no **`String`** objects or arrays (other than the ones embedded in the stack)
- Hint: Use at least 3 JCF **`Deque`** (stack) objects

Upcoming

Next time…

- Exam 1!

Reminders

- Keep reading Chapter 3
- Finish Project 1
  - **Due tonight by midnight!**
- **Exam 1 next Monday**