CMPT 435L ALGORITHMS
Assignment 1

October 4th 2024

Jonathan "Grant" Bever

Link to github repo

PART 1: Using Stacks and Queues to identify palindromes

In order to use stacks and queues to identify palindromes we must first understand what they are and how their built-in operations are perfect for this task. While both data structures are quite similar in the fact that they consist of nodes (elements of the data structure that have a piece of data and a pointer to the next node), their operations are what separate them.

Stacks are Last In First Out, meaning the last node added is the first to go, think a stack of plates, you put a plate on top and you pull from the top. this is in contrast to Queues which are First In First out, for this think of a line of people, people join the line from the back and exit from the front. These data structures operations are as follows:

Stacks:	**Queues:**
Push() Adds a node to top of stack	enQueue() Adds node to top of queue
Pop() Removes node from top of stack	deQueue() Removes node from end of queue

There is also isEmpty() which checks if a node is empty

Refer to the page below for my implementation for this:

The first letter of a palindrome is the same as the last so if we put the same word into a stack and queue and the letters we got when we pushed and popped until isEmpty() are the same, then that word is a palindrome.

Listing 1: Stack Class

```
export class Stack {
  top: node | null;

  constructor() {
    this.top = null;
  }

  // LAST IN
  push(data: string): void {
    const newNode = new node(data);
    newNode.next = this.top;
    this.top = newNode;
  }

  // FIRST OUT
  pop(): string | null {
    if (this.top === null) return null;
    const poppedNode = this.top;
    this.top = this.top.next;
    return poppedNode.data;
  }

  isEmpty(): boolean {
    return this.top === null;
  }
}
```

Listing 2: Queue Class

```
export class Queue {
  front: node | null;
  rear: node | null;

  constructor() {
    this.front = null;
    this.rear = null;
  }

  // FIRST IN
  enqueue(data: string): void {
    const newNode = new node(data);
    if (this.rear === null) {
      this.front = this.rear = newNode;
    } else {
      this.rear.next = newNode;
      this.rear = newNode;
    }
  }

  // FIRST OUT
  dequeue(): string | null {
    if (this.front === null) return
        null;
    const dequeuedNode = this.front;
    this.front = this.front.next;
    if (this.front === null) this.rear
        = null;
    return dequeuedNode.data;
  }

  isEmpty(): boolean {
    return this.front === null;
  }
}
```

With these classes in place we can go ahead and create a simple function that takes in magicitems.txt, turns it into an array that's readable for stacks and queues (no spaces or caps) and feed it word by word into the algorithm

1 Palindrome Detection Code

```
1  for (let i=0;i<dataItems.length;i++){
2      const word = dataItems[i];
3      const {stack,queue}=
           processItemsWithStackQueue(word);
4
5  //Compare the stack and queue to check for
       a palindrome
6      if (compareStackQueue(stack, queue)) {
7        console.log(`${word} is a palindrome
             `);
8        palendromes.push(word);
9      }
10 };
```

```
1  export function processItemsWithStackQueue
       (word: string):{stack:Stack,queue:
       Queue}{
2    const stack = new Stack();
3    const queue = new Queue();
4  //Splits into letters -> pushes and
       enqueues it
5    for (const letter of word) {
6      stack.push(letter);
7      queue.enqueue(letter);
8    }
9    // back to index
10   return { stack, queue };
11 }
```

Line 3 on the left chunk of code sends the word to the function on the right where the word is pushed and enqueued character by character then sent back only to get sent to the function below where all the cool stuff happens.

```
1  function compareStackQueue(stack: Stack, queue: Queue): boolean {
2    while (!stack.isEmpty() && !queue.isEmpty()) {
3      if (stack.pop() !== queue.dequeue()) {
4        return false;
5      }
6    }
7    return true;
8  }
```

This function is super simple but can be a bit hard to understand because it deals with double negatives, lets dissect lines 2 and 3 because that's where everything happens.

Line 2 sets the while loop under the conditions of the loop while the stack isn't empty and the queue isn't empty aka while there are still elements in the data structure. Line 3 pops and dequeues an element, if that element is not the same then it moves on to the next word, if the element is the same then it keeps looping until all 15 palindromes are found.

PART 2: Shuffling and Sorting

Now that all the palindromes have been found let's move on to sorting.
In this assignment, we will be implementing the selection sort, insertion sort, merge sort, and quick sort but before that...

Knuth Shuffle

In order to test our sorts we first need to shuffle the array. So we will be making a version of the Knuth Shuffle. The Knuth shuffle is a simple yet thorough shuffle that works by randomly selecting items from the array and swaps them into a "shuffled zone" that starts at the end of the array.

```
1 export function shuffle(items: any) {
2     for (let i = items.length - 1; i > 1; i--) {
3         const j = Math.floor(Math.random() * (i + 1));
4         [items[i], items[j]] = [items[j], items[i]];
5     }
6     return items;
7 }
```

We see this working at line 2 where it starts the for loop at length - 1 because that's labeled as the shuffled zone where, through random selection on line 3, and swapping on line 4, shuffled items are brought into that zone and the index decrements until the entire array is shuffled.

Now that our array is shuffled we can move onto our first sort, selection sort.

Selection Sort