Data Structures and Algorithms

Stacks and Queues

Contents

1. Stack

For this example, we would refer to our List Node algorithm in the previous chapter to facilitate our thought process.

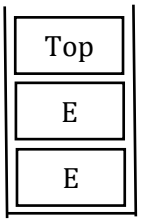
1.1 Last In, First Out

Any element that enters in last will be retrieved out first.
For Example, if Insert Order is 3, 4, 5, 5 will be retrieved first, followed by 4 then 3.

1.2 Stack Implementation

```
class Stack<T> {
    public ListNode<T> top;
    public int size;

    public Stack() {
        top = null;
        size = 0;
    }
}
```



1.3 Push it into the Stack!

With reference to list node in the previous chapter:

```
class ListNode<T> {
    public T element;
    public ListNode<T> prev;
    public ListNode<T> next;

    public ListNode(T element) {
        this.element = element;
    }
}
```

To push an element into the stack, you
1) make it the top element if the stack is empty
2) else, push it on top of the previous element

```
public void push(T element) {
    ListNode<T> temp = new ListNode<T>(element, null, null);
    size ++;
    if (isEmpty()) {
        top = temp;
    } else {
        top.next = temp;
        temp.prev = top;
        top = temp;
    }
}
```

Exercise 1:

Exercise 1.1: peek(): peeks at the topmost element of the stack.

```
public T peek() {
    // consider case if stack is empty!
}
```

Exercise 1.2: pop(): removes the topmost element of the stack, if any.

```
public T pop() {
    // consider case if stack is empty!
}
```

2. Stack Java API

https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html

Note: methods are almost identical to the methods discussed earlier. Just be careful of the method names during your tests / assignments!

3. Stack Applications

3.1 Evaluating Arithmetic Expressions

When given an arithmetic expression (which is normally infix), what you would want to do is to evaluate it. But what's the quickest way, well since we know:

Prefix	operator	operand	operand
Infix	operand	operator	operand
Postfix	operand	operand	operator

*Note: Priority of operands goes as follows: () > / and * > + and –*

So how do you do it?
1) Convert your Infix to Postfix using a stack
Covered in Exercise 3

2) Evaluate the Postfix expression using another stack
Covered in Exercise 3

3.2 Checking Matching Brackets

https://open.kattis.com/problems/brackets

Simply put,
1) If open bracket, push onto stack
2) If closed bracket, check if top of stack matches, if no match return false. Else, pop.

If the stack is empty after the expression, return true. Else, return false.

```
public boolean checkBrackets(String expression) {
    char[] charArray = expression.toCharArray();
    Stack<Character> stack = new Stack<Character>();
    for (int i = 0; i < charArray.length; i ++) {
        char c = charArray[i];
        if (c.isClosedBracket()) { // try it!
            if (stack.isEmpty()) {
                return false;
            } else {
                if (c.isMatchingBracket(stack.peek())) { // try it!
                    stack.pop();
                } else {
                    return false;
                }
            }
        } else if (c.isOpenBracket()) { // try it!
            stack.push(c);
        } else {
            // you have most likely met with an empty space
        }
    }

    return !stack.empty();
}
```

Exercise 3:

Exercise 3.1 Based on the Arithmetic Expression problem come up with the following helper function -> checkPriority(): checks if priority of operator1 > operator2 or not.

```
public boolean checkPriority(String op1, String op2) {
    // do for + - * and /
}
```

Exercise 3.2: infixToPostfix(): converts an infix String to a postfix String.

```
public String infixToPostfix(String infix) {
    String result = "";
    String[] arr = infix.replaceAll(" ", "").split("");
    Stack<String> s = new Stack<String>();
    for (int i = 0; i < infix.length(); i ++) {
        String c = arr[i];
        // consider if c is an open bracket
        // consider if c is a closed bracket
        // consider if c is an operand
        // consider if c is an operator
    }

    // what happens if the stack is still not empty?

    return result;
}
```

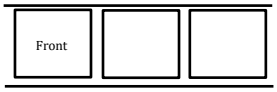
Exercise 3.3: evaluatePostfix(): evaluates the postfix expression.

```
public int evaluatePostfix(String postfix) {
    String[] arr = infix.replaceAll(" ", "").split("");
    Stack<Integer> s = new Stack<Integer>(); // read carefully!
    for (int i = 0; i < postfix.length(); i ++) {
        char c = postfix.charAt(i);
        // consider if c is an operand
        // consider if c is an operator
    }

    // return value if any
}
```

4. Queue

4.1 First In, First Out



Like an ordinary queue, whoever comes in first, get served first.
So the first element that is enqueued, is dequeued first too.

4.2 Queue Implementation

Important question: why do we put a back pointer?

```
protected ListNode<T> front;
protected ListNode<T> back;
protected int size;

public Queue() {
    this.front = null;
    this.back = null;
    this.size = 0;
}
```

4.3 Queue Up!

```
public void enqueue(T element) {
    size ++;
    if (isEmpty()) {
        front = new ListNode<T>(element, null, null);
        back = front;
    } else if (size == 1) {
        back = new ListNode<T>(element, front, null);
        front.next = back;
    } else {
        ListNode<T> temp = new ListNode<T>(element);
        back.next = temp;
        temp.prev = back;
        back = temp;
    }
}
```

Exercise 4:

Exercise 4.1: peek(): peeks at the front element of the queue, if any

```
public T peek() {
    // consider if the queue is empty
}
```

Exercise 4.2: dequeue(): removes the frontmost element, if any

```
public T dequeue() {
    // consider if the queue has 0, 1 and 2 elements
}
```

5. Queue Java API

https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html

Note: methods are almost identical to the methods discussed earlier. Just be careful of the method names during your tests / assignments!
Note: Queue is an interface! So how do you instantiate a Queue?

Queue<Integer> q = new LinkedList<>();

Bonus: Is there a possibility when you can enqueue and dequeue from the front and back?
YES! Take a look at the documentation for these two classes to find out more!
https://docs.oracle.com/javase/7/docs/api/java/util/Deque.html
https://docs.oracle.com/javase/7/docs/api/java/util/ArrayDeque.html

Important Methods:
offerFirst, offerLast, removeFirst, removeLast.

6. Queue Applications

6.1 Palindrome Checking

Given a String, how do you check if it is a palindrome?
For example, racecar, radar...
Hint: Use a stack and a queue
Covered in Exercise 6

Exercise 6:

Exercise 6.1: isPalindrome(): checks if the String is a Palindrome or not.

```
public boolean isPalindrome(String o) {
    char[] charArray = o.toCharArray();
    Stack<Character> s = new Stack<Character>();
    Queue<Character> q = new LinkedList<Character>();
    // do it yourself!
}
```

7. Bonus Questions

Note: I have put this up here for your own extra learning. These are not compulsory but good to try! I will not put the answers in this booklet. Hints can be found online but do attempt :D

Exercise 7.1: Implement a Queue using two stacks

Exercise 7.2: Implement a Stack using two queues.

Suggestion Solutions

Exercise 1:

```
public T peek() {
    if (isEmpty()) {
        return null;
    } else {
        return top.element;
    }
}

public T pop() {
    if (isEmpty()) {
        return null;
    } else {
        size --;
        if (top.prev != null) {
            top.prev.next = null;
        }
        ListNode<T> temp = top;
        top = top.prev;
        return temp.element;
    }
}
```

Exercise 3:

```
public boolean checkPriority(String op1, String op2) {
    if (op1.equals("*") || op1.equals("/")) {
        if (op2.equals("+") || op2.equals("-")) {
            return true;
        } else {
            return false;
        }
    } else {
        return false;
    }
}

public String infixToPostfix(String infix) {
    String result = "";
    String[] arr = infix.replaceAll(" ", "").split(""); // String array
    Stack<String> s = new Stack<String>();
    for (int i = 0; i < arr.length; i ++) {
        String c = arr[i];
        if (c.equals("(")) {
            s.push(c);
        } else if (c.equals(")")) { // pops all operators until open bracket
            while (!s.peek().equals("(")) {
                result += s.pop();
            }
            s.pop();
        } else if (isOperator(c)) { // do it yourself!
            while (!s.isEmpty() && isOperator(s.peek()) && checkPriority(s.peek(), c)) {
                result += s.pop();
            }
            s.push(c);
        } else {
            result += c;
        }
    }

    while (!s.isEmpty()) {
        result += s.pop();
    }

    return result;
}


public int evaluatePostfix(String postfix) {
    String[] arr = postfix.replaceAll(" ", "").split("");
    Stack<Integer> s = new Stack<Integer>();
    for (int i = 0; i < postfix.length(); i ++) {
        char c = postfix.charAt(i);
        if (isOperator(c)) { // do it yourself!
            int ans = evaluate(s.pop(), s.pop(), c); // do it yourself!
            s.push(ans);
        } else {
            s.push(Integer.parseInt(c));
        }
    }

    if (s.isEmpty()) {
        return 0;
    } else {
        return s.pop();
    }
}
```

Exercise 4:

```
public T peek() {
    if (isEmpty()) {
        return null;
    } else {
        return front.element;
    }
}
```

```
public T dequeue() {
    ListNode<T> temp;
    if (size == 0) {
        return null;
    } else if (size == 1) {
        temp = head;
        head = null;
        tail = null;
        size = 0;
        return temp.element;
    } else if (size == 2) {
        temp = head;
        back.prev = null;
        head = back;
        size = 1;
        return temp.element;
    } else {
        temp = head;
        head.next.prev = null;
        head = head.next;
        size --;
        return temp.element;
    }
}
```

Exercise 6:

```
public boolean isPalindrome(String o) {
    char[] charArray = o.toCharArray();
    Stack<Character> s = new Stack<Character>();
    Queue<Character> q = new LinkedList<Character>();

    for (int i = 0; i < charArray.length; i ++) {
        char c = charArray[i];
        s.push(c);
        q.offer(c);
    }

    while (!s.isEmpty() && !q.isEmpty()) {
        if (s.pop() != q.poll()) {
            return false;
        }
    }

    return true;
}
```