Cracking the Coding Interview – STUDY GUIDE

by **reah miyara**

May 24, 2016

Programming Paradigms

Object Oriented
In object oriented languages **data and methods of manipulating the data are kept as a single unit called an object**. The only way a user can access the data is via object's methods. Therefore the inner workings of an object may be changed without affecting any code that uses the object.

> **Polymorphism**
> Providing or supplying a single interface to be used with entities of different types.

Declarative
In declarative languages the computer is told **what the problem is, not how to solve the problem** –the program is structured as a collection of properties to find in the expected result, not as a procedure to follow. It's a style of expressing the logic of a computation without describing its control flow. This is in **contrast with imperative programming**, which implements algorithms in explicit steps. *Given a database or a set of rules, the computer tries to find a solution matching all the desired properties., e.g. SQL*

Imperative
Imperative programming focuses on **how a program operates**, consisting of commands for the computer to perform. This **contrasts declarative programming**.

Functional
Functional programming is a **subset of declarative programming**. Programs are written using functions, *blocks of code intended to behave like mathematical functions.* Functional languages discourage changes in the value of variables through assignment.

Arrays & Strings

1. **Determine if string has unique characters; without additional data structures.**

   - split up characters from string `chars[] = str.toCharArray();`
   - get single character from string `char c = str.charAt(i);`
   - cast char to int, `int value = str.charAt(i);` , (this is because the primitive char datatype is a 16 bit unsigned integer)
   - shift 1 by the int casted from the char && with a checker 0, if result isn't 0 then not unique, if it is checker |= result

2. **Reverse a String**

   - Strings are immutable so use StringBuffer, *(mutable, stored on the heap, each method is thread safe, synchronized)*, or StringBuilder, *(same methods as StringBuffer but not synchronized, hence not thread safe)*, and finalize the string using `StringBuilder.toString()`

- method 1: copy backwards into StringBuffer linearly
  `for (int i = s.length()-1; i >= 0; i--) { sb.append(s.charAt(i)); } return sb.toString();`
- method 2 (in place): turn string into char array
  `char[] reverse = s.toCharArray();` and concurrently swap last and first chars until reach the middle
  `for(int i = 0, j = s.length()-1; i < s.length()/2; i++, j--) {`
  `char temp = reverse[i]; reverse[i] = reverse[j]; reverse[j] = temp;`
  `} return new String(reverse);`
- method 3: `return new StringBuilder(s).reverse().toString();`

3. **Given 2 strings write a method to determine if one is a permutation of the other**

- check to see if lengths are the same `if(s1.length() != s2.length()) return false;`
- method 1, sort O(nlogn): convert each string to an array of chars
  `char[] chars = s.toCharArray();` sort each in place `Arrays.sort(chars);` compare using
  `Objects.equals(chars1, chars2);` which checks for `null` unlike `s1.equals(s2);`
- initialize array of ints, if unicode, or hashmap otherwise to keep count of chars in s1.
  `int[] charCount = new int[128];` or `HashMap<Character, Integer> charCount = HashMap<C,I>();`
- iterate over first string incrementing count for char
  `for(int i = 0; i < s1.length(); i++) {`
  `charCount[s1.charAt(i)]++;}` or
  `if(map.get(s1.charAt(i) != null) { map.put(s1.charAt(i), map.get(s1.charAt(i)) + 1); }`
  `else { map.put(s1.charAt(i), 1); }`
- iterate over second string decrementing count for char, returning false if count falls below 0
  `for(int i = 0; i < s2.length(); i++) { if(--charCount[s2.charAt(i)] < 0) return false; }` or
  `if(charCount.get(s2.charAt(i)) == null || (charCount.get(s2.charAt(i)) - 1 < 0)) { return false; }`

  `} else { charCount.put(s2.charAt(i), charCount.get(s2.charAt(i)) - 1); }`

4. **Write a method to replace all spaces in a string with '%20'**

- method 1, StringBuffer: create a StringBuffer object, convert String to char array, append each char that does not equal ' ' to the StringBuffer otherwise append '%20'
- method 2, using only primitive types: count the spaces in the String, calculate new size of char array needed to contain new string, copy each char if not ' ', otherwise '%20'by using a second iterator j
  `int newSize = s.length() + spaceCount * 2; char[] modified = new char[newSize];`
- method 3, given char[] that can fit new string and true length of original string: calculate new size of string again by first counting spaces `int index = s.length + spaceCount * 2 - 1;`
  now in order to not overwrite original characters when replacing ' ', insert each char backwards with the index calculated and true length given, finally returning new string

```
for(int i = length; i >=0; i--) {
        if(s[i] == ' ') {
                s[index--] = '0';
                s[index--] = '2';
                s[index--] = '%';
        } else {
                s[index--] = s[i];
        }
}
return new String(s);
```

General Java Knowledge

Unicode

Java inherently supports unicode, encoding standard which contains 128 different characters, for char primitive types.

code input sample

```
public class isUnique {
        public static boolean isUnique(String s) {
                // hello world
                HashMap <Character, Boolean>  charMap = new HashMap<Character,
                   Boolean>();
        return true;
        }
}
```

Properties of supremum and infimum

Let h be a given positive number and let S be a set of real numbers.
(a) If S has a supremum, then for some x in S we have $x > \sup S - h$.
(b) If S has an infimum, then for some x in S we have $x < \inf S + h$.

Well-ordering principle

Every nonempty set of positive integers contains a smallest member.

Triangle inequality

For arbitrary real numbers x and y, $|x + y| \leq |x| + |y|$. More generally, for arbitrary real numbers a_1, a_2, $\ldots$, a_n, we have $|\sum_{k=1}^{n} a_k| \leq \sum_{k=1}^{n} |a_k|$.

The Cauchy-Schwarz inequality

If $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ are arbitrary real numbers, we have $\left(\sum_{k=1}^{n} a_k b_k\right)^2 \leq \left(\sum_{k=1}^{n} a_k^2\right)\left(\sum_{k=1}^{n} b_k^2\right)$. The equality sign holds if and only if there is a real number x such that $a_k x + b_k = 0$ for each $k = 1, 2, \ldots, n$.

Complex Field

Field properties

$(a, b) = (c, d)$ means $a = c$ and $b = d$
$(a, b) + (c, d) = (a + c, b + d)$
$(a, b)(c, d) = (ac - bd, ad + bc)$ $x + y = y + x$
$x + (y + z) = (z + y) + z$
$x(y + z) = xy + xz$
$e^{z+2n\pi i} = e^z$

Polar coordinates

$x = r\cos\theta \qquad y = r\sin\theta$
r is the modulus or absolute value of (x, y), equal to $\sqrt{x^2 + y^2}$.
θ is the angle between (x, y) and the x-axis, and is called the argument of (x, y), or the principal argument if $-\pi < \theta \leq \pi$.
Polar form of z: Every complex number $z \neq 0$ can be expressed as $z = re^{i\theta}$.

Complex exponential

If $z = (x, y)$, then $e^z = e^x(\cos y + i\sin y)$
$e^a e^b = e^{a+b}$

Derivatives and integrals

If $f = u + iv$, then $f'(x) = u'(x) + iv'(x)$
$\int_a^b f(x)\,dx = \int_a^b u(x)\,dx + i\int_a^b v(x)\,dx$
$(e^{tx})' = te^{tx}$
$\int e^{tx}\,dx = \frac{e^{tx}}{t}$