Programming competition guidebook

Juan J. Alvarez
juan.alvarez7@upr.edu
Computer Science Department
University of Puerto Rico
At Bayamón

Dr. Juan Solá Sloan
juan.sola@upr.edu
Computer Science Department
University of Puerto Rico
At Bayamón

Contents

1 Mathematics

1.1 Check whether a number is prime

In this section we talk about how to programatically check the primality of a given number.

1.1.1 Pseudocode

Algorithm 1 Prime check algorithm

```
procedure isPrime($n$)
    if $n < 2$ then return $false$
    if $n == 2$ then return $true$
    if $n\%2 == 0$ then return $false$
    for $x = 3$ ; $x^2 <= n$ ; $x+ = 2$ do
        if $n\%x == 0$ then return $false$
    return $true$
```

1.1.2 Implementation

```
public static boolean isPrime(int n) {
        if (n < 2)
                return false;
        if (n == 2)
                return true;
        if (n % 2 == 0)
                return false;
        for (int x = 3; x * x <= n; x += 2)
                if (n % x == 0)
                        return false;
        return true;
}
```

1.2 List the divisors of a number

In this section we talk about how to programatically list the divisors of a given number.

1.2.1 Pseudocode

Algorithm 2 Divisor list algorithm

```
procedure listDivisors($n$)
    $list$ = new empty list of numbers
    add 1 to $list$
    $mpd = \sqrt{n}$
    for $x = 2$ ; $x <= mpd$ ; $x + +$ do
        if $n\%x == 0$ then
            add $x$ to $list$
            if $n \div x \neq x$ then
                add $n/d$ to $list$
    add $n$ to $list$
return $list$
```

1.2.2 Implementation

```
private static ArrayList<Integer> getDivisors(int input) {
        ArrayList<Integer> list = new ArrayList<Integer>();
        list.add(1);
        int maxD = (int) Math.sqrt(input);
        for (int i = 2; i <= maxD; i++) {
                if (input % i == 0) {
                        list.add(i);
                        int d = input / i;
                        if (d != i)
                                list.add(d);
                }
        }
        list.add(input);
        return list;
}
```

1.3 Calculating Factorials

1.3.1 Pseudocode

Algorithm 3 Factorial Algorithm

```
procedure FACTORIAL($n$)
    $total = 1$
    for $x = n$ ; $x > 1$ ; $x++$ do
        $total* = x$
    return $total$
```

1.3.2 Implementation

This method has a huge flaw due to the fact that it uses 64-bit signed integers (Java long) to calculate the result, it is not capable of calculating the factorial of any number larger than 16.

```
public static long fac(int n) {
        long total = 1;
        for (int x = n; x > 1; x--)
                total *= n;
        return total;
}
```

1.3.3 BigInteger Implementation

This implementation was done using the BigInteger class to solve the limitational problems of using 64-bit integers.

```
public static BigInteger fac(BigInteger n) {
        BigInteger one = BigInteger.ONE;
        BigInteger total = one;
        BigInteger x;
        for (x = n; x.compareTo(one) == 1; x = x.subtract(one))
                total = total.multiply(n);
        return total;
}
```

1.4 Sum of Natural Numbers

The sum of the sequence of natural numbers $\{1, 2, 3, ..., n\}$ can be written as

$$\sum_{x=1}^{n} x = \frac{n(n+1)}{2}$$

.

1.5 Divisibility Rules

2	The last digit is 0, 2, 4, 6 or 8. Or the modulus operation with 2 yields 0.
3	The sum of the digits is divisible by 3.
4	The number formed by the last two digits is divisible by 4.
5	The last digit is either 0 or 5.
6	It is divisible by 2 AND it is divisible by 3.
7	If the last digit multiplied by two and subtracted from the rest of the number is divisible by 7.
8	The last three digits are divisible by 8.
9	The sum of the digits is divisible by 9.
10	The last digit is 0.
11	The difference between the sum of the odd placed digits and the sum of the even placed digits is divisible by 11.
12	The number is divisible by both 3 and 4.
13	Subtract 9 times the last digit from the rest of the number, the result is divisible by 13.
14	It is divisible by 2 and 7.
15	It is divisible by 3 and 5.
16	The last 4 digits are divisible by 16.

2 Strings

2.1 Check if a string is a palindrome

A palindrome is a string that is written the same way as if it were written in reversed order. The perfect example of a palindrome is the word 'racecar', if you write racecar in reverse order you get 'racecar'. The following Java method is an efficient palindrome checker.

```
static boolean isPalindrome(String s) {
        for (int x = 0; x < s.length() / 2; x++)
                if (s.charAt(x) != s.charAt(s.length() - x - 1))
                        return false;
        return true;
}
```

3 Sets

3.1 Counting repeated elements in a set

The following method counts the repeated elements in the given list and stores the count for each unique element in it's own space in a HashMap. The returned value is a HashMap¡E, Integer¿ (E being the type of element being counted) containing the unique elements as the keys and their respective counts as the values.

```
static <E> HashMap<E, Integer> countRepeatedElements(ArrayList<E> list) {
        HashMap<E, Integer> map = new HashMap<E, Integer>();
        Integer count;
        for (E element : list) {
                count = map.get(element);
                if (count == null)
                        count = 0;
                count++;
                map.put(element, count);
        }
        return map;
}
```

This is an implementation using the previous code to count the repetitions in a list of Integers.

```
public static void main(String[] arguments) {
        Integer[] arr = { 1, 1, 1, 2, 2, 3, 3, 3, 3, 4, 5, 5, 5,
                 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6 };
        ArrayList<Integer> list = new ArrayList<Integer>();
        for (Integer i : arr)
                list.add(i);
        HashMap<Integer, Integer> map = countRepeatedElements(list);
        Set<Integer> set = map.keySet();
        for (Integer key : set)
                if (map.get(key) > 1)
                        System.out.println(String.format(
                                "%d was repeated %d times", key, map.get(key)));
}
```

3.2 Unique elements in a set

For this section we utilize a common method defined as follows.

```
static <E> HashSet<E> uniqueElements(ArrayList<E> list) {
        HashSet<E> set = new HashSet<E>();
        for (E e : list)
                set.add(e);
        return set;
}
```

3.2.1 Counting the unique elements in a set

With the previously defined method we can count the number of unique elements in a list like this:

```
uniqueElements(list).size()
```

3.2.2 Iterating over the unique elements in a set

With the previously defined method we can iterate over the unique elements in a list like this:

```
Set<E> set = uniqueElements(list);
        for(E e : set)
                doSomething(e);
```

3.3 Powersets

A powerset is a set containing all of the possible subsets of an original set. For example, the powerset of 1, 2, 3 is , 1, 2. 3, 1, 2, 1, 3, 2, 3, 1, 2, 3. Powersets in programming are useful to identify which combination of elements yields certain expected results, for example given the set 1, 2, 3, 4, 5 with powersets we can identify that the following set contains the subsets that when added equal 4 1, 3, 4.

3.3.1 Binary Powersets

A binary powerset uses 64 bit integers(long) to handle the combinations. Due to technical constraints using this type of powerset implementation the original set must not contain any more than 64 elements due to the fact that whether or not a single element exists in any particular subset is represented by a single bit in the combination integer. The following code is an example of binary powersets put to use with a set of characters, it prints every possible combination of characters (17 combinations for 4 characters).

```
static int getBit(long num, int position) {
        return (int) (num >> position & 1);
}

public static void main(String[] arguments) {
        char[] arr = "abcd".toCharArray();
        long x;
        int y;
        for (x = 0; x < (2 << arr.length - 1); x++) {
                for (y = 0; y < arr.length; y++)
                        if (getBit(x, y) == 1)
                                System.out.print(arr[y]);
                System.out.println();
        }
}
```