Универзитет „Св. Кирил и Методиј“ - Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Напредно програмирање

Аудиториски вежби 8

Верзија 1.0, 23 Ноември, 2016

Содржина

1. Колекции (List)

1.1. Count occurrence

Да се имплементира следниот метод кој го враќа бројот на појавување на стрингот `str` во колекцијата од колекција од стрингови:

```
public static int count(Collection<Collection<String>> c, String str)
```

- Да претпоставиме дека `Collection c` содржи `N` колекции и дека секоја од овие колекции содржи `N` објекти. Кое е времето на извршување на вашиот метод?
- Да претпоставиме дека е потребно 2 милисекунди да се изврши за N = 100. Колку ќе биде времето на извршување кога N = 300?

Решение (`CountCollection.java`)

```
public static int count(Collection<Collection<String>> c, String str) {
    int count = 0;
    for (Collection<String> sub : c) {
        for (String s : sub) {
            if (s.equals(str)) {
                ++count;
            }
        }
    }
    return count;
}

static long countFunc(Collection<Collection<String>> collection, String str) {
    return collection.stream()
            .mapToLong(coll ->
                    coll.stream()
                            .filter(s -> s.equals(str))
                            //.peek(s -> System.out.println("s: " + s))
                            .count()

            )//.peek(i -> System.out.println("i: " + i))
            .sum();
}
```

1.2. Reverse list

Да се напише метод за печатење на колекција во обратен редослед со помош на `Collections API` но без употреба на `ListIterator`.

Решение (`CountCollection.java`)

```
    public static <T> void printReverse(Collection<? extends T> collection) {
        int size = collection.size();
        Object[] array = collection.toArray();
        for (int i = size - 1; i >= 0; --i) {
            System.out.println(array[i]);
        }
    }


    static <T> void reversePrint(Collection<? extends T> collection) {
        Object[] array = collection.toArray(new Object[collection.size()]);
        for (int i = array.length - 1; i >= 0; --i) {
            System.out.println(array[i]);
        }
    }
```

1.3. Equals performance

Методот `equals` кој е прикажан, враќа `true` ако две листи имаат иста големина и ако ги содржат истите елементи во ист редослед. Да претпоставиме дека `N` е големината на овие листи.

```
public boolean equals(List<Integer> left, List<Integer> right) {
    if(left.size() != right.size()) {
        return false;
    } else {
        for(int i = 0; i < left.size(); ++i) {
          if(!left.get(i).equals(right.get(i))) {
              return false;
          }
        }
    }
    return true;
}
```

- Кое е времето на извршување ако двете листи се `ArrayLists`?
- Кое е времето на извршување ако двете листи се `LinkedLists`?
- Да претпоставиме дека за 4 секунди се извршува методот за две еднакво големи поврзани листи `LinkedList` со 10,000 елементи. Колку време ќе биде потребно за извршување со две еднакво големи поврзани листи со 50,000 елементи?
- Објаснете со една реченица како да го направиме алгоритмот поефикасен за сите видови листи?

1.4. The Sieve of Eratosthenes

Ситото на Ерастотен (The Sieve of Eratosthenes) е древен алгоритам за

генерирање прости броеви. Да ја земеме следната листа со броеви: 2 3 4 5 6 7 8 9 10

Алгоритмот започнува со првиот прост број во листата, тоа е 2 и потоа ги изминува останатите елементи од листата, со тоа што ги отстранува сите броеви чии што множител е 2 (во овој случај, 4, 6, 8 и 10), со што остануваат 2 3 5 7. Го повторуваме овој процес со вториот прост број во листата, тоа е 3 и итерираме низ остатокот од листата и што ги отстрануваме броевите чии што множител е 3 (во овој случај 9), а остануваат 2 3 5 7. Потоа ја повторуваме постапката со секој следен прост број, но не се отстрануваат елемнти, затоа што нема броеви чии што множители се 5 и 7. Сите броеви кои остануваат во листата се прости.

Да се имплементира алгоритмот со користење на `ArrayList` од цели броеви која што е иницијализирана со вредности од 2 до 100. Имплементацијата може да итерира низ листата со индекс од `0` од `size() - 1` за да го земе тековниот прост број, но треба да користи `Iterator` за да го скенира остатокот од листата и ги избрише сите елементи чии што множител е тековниот прост број. Отпечатете ги сите прости броеви во листата.

Решение (`EratosthenesSieve.java`*)*

```
package mk.ukim.finki.np.av8;

import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;
import java.util.stream.IntStream;

public class EratosthenesSieve {

    static boolean isPrime(int n) {
        return n >= 2 && IntStream.rangeClosed(2, n / 2)
                .noneMatch(x -> n % x == 0);
    }

    List<Integer> getPrimes(int n) {
        List<Integer> primes = new ArrayList<>();
        IntStream.rangeClosed(2, n)
                .forEach(primes::add);
        for (int i = 0; i < primes.size(); ++i) {
            int currentPrime = primes.get(i);
            ListIterator<Integer> it = primes.listIterator(i + 1);
            while (it.hasNext()) {
                if (it.next() % currentPrime == 0) {
                    it.remove();
                }
            }
        }
        return primes;
    }

    public static void main(String[] args) {
        /*for (int i = 0; i < 1000; ++i) {
            if (isPrime(i)) {
                System.out.println(i);
            }
        }*/
        EratosthenesSieve sieve = new EratosthenesSieve();
        List<Integer> primes = sieve.getPrimes(1000);
        primes.forEach(System.out::println);

    }
}
```

1.5. Suitors

In an ancient land, the beautiful princess Eve had many suitors. She decided on the following procedure to determine which suitor she would marry. First, all of the suitors would be lined up one after the other and assigned numbers. The first suitor would be number 1, the second number 2, and so on up to the last suitor, number n. Starting at the first suitor, she would then count three suitors down the line (because of the three letters in her name) and the third suitor would be eliminated from winning her hand and removed from the line. Eve would then continue, counting three more suitors, and eliminating every third suitor. When she reached the end of the line, she would reverse direction and work her way back to the beginning. Similarly, on reaching the first person in line, she would reverse direction and make her way to the end of the line. For example, if there were five suitors, then the

elimination process would proceed as follows:

```
12345 Initial list of suitors; start counting from 1.
1245 Suitor 3 eliminated; continue counting from 4 and bounce from end
back to
Suitor 4 eliminated; continue counting back from 2 and bounce from
front back to 2.
15
Suitor 2 eliminated; continue counting forward from 5.
1
Suitor 5 eliminated; 1 is the lucky winner.
```

Write a program that uses an `ArrayList` or `Vector` to determine which position you should stand in to marry the princess if there are n suitors. Your program should use the `ListIterator` interface to traverse the list of suitors and remove a suitor. Be careful that your iterator references the proper object upon reversing direction at the beginning or end of the list. The suitor at the beginning or end of the list should only be counted once when the princess reverses the count.

```java
package mk.ukim.finki.np.av8;

import java.util.List;
import java.util.ListIterator;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

public class LuckySuitor {
    private final List<Integer> positions;

    public LuckySuitor(int n) {
        positions = IntStream.rangeClosed(1, n)
                .mapToObj(Integer::new)
                .collect(Collectors.toList());
    }


    public int getWinner() {
        ListIterator<Integer> listIterator = positions.listIterator();
        boolean toRight = true;
        while (positions.size() != 1) {
            int last = -1;
            for (int i = 0; i < 3; ++i) {
                if (listIterator.hasNext() && toRight) {
                    last = listIterator.next();
                    if (!listIterator.hasNext()) {
                        toRight = false;
                        listIterator.previous();
                    }
                    //System.out.println("->: " + last);
                } else {
                    if (listIterator.hasPrevious()) {
                        last = listIterator.previous();
                        if (!listIterator.hasPrevious()) {
                            toRight = true;
                            listIterator.next();
                        }
                        //System.out.println("<-: " + last);
                    }
                }
            }
            //System.out.println("Remove: " + last);
            listIterator.remove();
            //System.out.println("DIR: " + (toRight ? "->" : "<-"));
        }
        return positions.get(0);
    }

    public static void main(String[] args) {
        LuckySuitor luckySuitor = new LuckySuitor(5);
        System.out.println("Winner: " + luckySuitor.getWinner());
    }
}
```

2. Изворен код од примери и задачи

https://github.com/finki-mk/NP/

Source Code ZIP