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

# Напредно програмирање

Аудиториски вежби 9

Верзија 1.0, 1 Декември, 2016

# Содржина

# 1. Колекции (Set, Map)

## 1.1. Birthdays paradox

Парадоксот на роденден е дека постои изненадувачки голема веројатност дека двајца луѓе во иста просторија имаат ист роденден. Под ист роденден подразбираме дека се родени на ист ден во годината (ги игнорираме престпните). Да се напише програма која апроксимативно ја пресметува веројатноста дека 2-ца или повеќе луѓе во иста просторија имаат ист роденден, за 2 до 50 луѓе во просторија. Програмата треба да го пресмета одговорот со симулација. Во повеќе обиди (пр. 5000), се доделува случаен роденден (пр. број од 1 - 365) на сите во просторијата. Роденденте се чуваат во `HashSet`. Како што се генерираат случајни родендени, со помош на методот `contains` на `HashSet` се проверува дали тој роденден е веќе назначен во собата. Ако е тоа случај, се зголемува еден бројач кој брои колку пати барем 2-ца во собата имаат ист роденден и се преминува на следниот обид. Излезот од вашата програма треба да биде во следниот облик. Броевите нема да бидат сосема точни затоа што се употребуваат случајни броеви.

```
For 2 people, the probability of two birthdays is about 0.002
For 3 people, the probability of two birthdays is about 0.0082
For 4 people, the probability of two birthdays is about 0.0163
...
For 49 people, the probability of two birthdays is about 0.9654
For 50 people, the probability of two birthdays is about 0.969
```

*Решение (`Birthdays.java`)*

```
package mk.ukim.finki.np.av9;

import java.util.*;

public class Birthdays {
    static final int NUM_TRIALS = 5000;

    private final int maxPeople;

    public Birthdays(int maxPeople) {
        this.maxPeople = maxPeople;
    }

    public Map<Integer, Double> simulate(int numTrials) {
        Random rand = new Random();
        Map<Integer, Double> probabilities = new TreeMap<>();
        for (int numPeople = 2; numPeople <= maxPeople; ++numPeople) {
            double probability = simulation(numPeople, numTrials, rand);
            probabilities.put(numPeople, probability);
        }
        return probabilities;
    }

    private double simulation(int numPeople, int numTrials, Random random) {
        int positiveEvents = 0;
        for (int i = 0; i < numTrials; ++i) {
            if (singleExperiment(numPeople, random)) {
                ++positiveEvents;
            }
        }
        return positiveEvents * 1.0 / NUM_TRIALS;
    }

    private boolean singleExperiment(int numPeople, Random random) {
        int sameBirthday = 1;
        Set<Integer> room = new HashSet<>();
        for (int i = 0; i < numPeople; ++i) {
            int birthday = random.nextInt(365) + 1;
            if (!room.add(birthday)) {
                ++sameBirthday;
            }
        }
        return sameBirthday >= 2;
    }

    public static void main(String[] args) {
        Birthdays birthdays = new Birthdays(50);
        Map<Integer, Double> probabilities = birthdays.simulate(NUM_TRIALS);
        probabilities.forEach((key, value) ->
                System.out.printf("For %d people, the probability of two birthdays is
about %.3f\n", key, value));
    }
}
```

## 1.2. Names

Текстуалните датотеки `boynames.txt` и `girlnames.txt`, содржат листа од 1,000 од најпопуларните машки и женски имиња во САД за 2005 година. Секој ред од датотеките се состои од име и број на регистрирани раѓања таа година со тоа име, со тоа што имињата се подредени по популарност. Пример, датотеката `girlnames.txt` започнува со

```
Emily 25494
Emma 22532
```

Ова означува дека Emily е најпопуларно име со 25,494 регистрирани имиња.

Напишете програма која ќе најде колку имиња се наоѓаат и на двете листи, машката и женската.

*Решение (`Names.java`)*

```
package mk.ukim.finki.np.av9;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public class Names {

    public List<Name> readNames(String fileName) throws IOException {
        return Files.lines(Paths.get(fileName))
                .map(Name::ofString)
                .collect(Collectors.toList());
    }

    public List<DuplicateName> findDuplicates(List<Name> girls, List<Name> boys) {
        Map<String, DuplicateName> duplicates = new HashMap<>();
        girls.forEach(name ->
                duplicates.computeIfAbsent(name.getName(), key -> new DuplicateName(key,
name.getCount()))
        );
        boys.forEach(name ->
                duplicates.computeIfPresent(name.getName(), (key, duplicateName) -> {
                    duplicateName.setCountBoys(name.getCount());
                    return duplicateName;
                })
        );
        return duplicates.values().stream()
                .filter(name -> name.getCountBoys() > 0)
                .collect(Collectors.toList());
    }


    public static void main(String[] args) {
        Names names = new Names();
        try {
            List<Name> girls = names.readNames("examples/data/girlnames.txt");
            List<Name> boys = names.readNames("examples/data/boynames.txt");
            List<DuplicateName> duplicates = names.findDuplicates(girls, boys);
            duplicates.forEach(System.out::println);
        } catch (IOException e) {
            System.err.println(e.getMessage());
        }
    }
}

class Name {
    private final String name;
    private final int count;

    Name(String name, int count) {
        this.name = name;
        this.count = count;
    }

    public static Name ofString(String line) {
```

```
        String[] parts = line.split("\\s+");
        String name = parts[0];
        int count = Integer.parseInt(parts[1]);
        return new Name(name, count);
    }

    @Override
    public String toString() {
        return String.format("%s %d", name, count);
    }

    public String getName() {
        return name;
    }

    public int getCount() {
        return count;
    }
}

class DuplicateName extends Name {
    private int countBoys;

    public DuplicateName(String name, int count) {
        super(name, count);
    }

    public int getCountBoys() {
        return countBoys;
    }

    public void setCountBoys(int countBoys) {
        this.countBoys = countBoys;
    }

    @Override
    public String toString() {
        return String.format("%s %d", super.toString(), countBoys);
    }
}
```

## 1.3. Audition

Да се имплементира класа за аудиција `Audition` со следните методи:

- `void addParticipant(String city, String code, String name, int age)` - додава нов кандидат со код `code`, име и возраст за аудиција во даден град `city`. Во ист град не се дозволува додавање на кандидат со ист код како некој претходно додаден кандидат (додавањето се игнорира, а комплексноста на овој метод треба да биде `O(1)`)
- `void listByCity(String city)` - ги печати сите кандидати од даден град подредени според името, а ако е исто според возраста (комплексноста на овој метод не треба да надминува `O(n log2(n))`, каде `n` е бројот на кандидати во дадениот град).

*Решение (`AuditionTest.java`)*

```
package mk.ukim.finki.np.av9;
```

```
import java.util.*;

public class AuditionTest {
    public static void main(String[] args) {
        Audition audition = new Audition();
        List<String> cities = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);
        while (scanner.hasNextLine()) {
            String line = scanner.nextLine();
            String[] parts = line.split(";");
            if (parts.length > 1) {
                audition.addParticipant(parts[0], parts[1], parts[2],
                        Integer.parseInt(parts[3]));
            } else {
                cities.add(line);
            }
        }
        for (String city : cities) {
            System.out.printf("+++++ %s +++++\n", city);
            audition.listByCity(city);
        }
        scanner.close();
    }
}

class Audition {
    HashMap<String, HashSet<Participant>> participants;

    public Audition() {
        participants = new HashMap<>();
    }

    public void addParticipant(String city, String code, String name, int age) {
        Set<Participant> cityParticipants = participants.computeIfAbsent(city, key -> new
HashSet<>());
        cityParticipants.add(new Participant(code, name, age));
    }

    public void listByCity(String city) {
        participants.get(city).stream()
                .sorted(Comparator
                        .comparing(Participant::getName)
                        .thenComparing(Participant::getAge))
                .forEach(System.out::println);
    }
}

class Participant {
    private final String code;
    private final String name;
    private final int age;

    public Participant(String code, String name, int age) {
        this.code = code;
        this.name = name;
        this.age = age;
    }

    @Override
    public boolean equals(Object obj) {
        Participant p = (Participant) obj;
        return code.equals(p.code);
    }

    @Override
    public int hashCode() {
        return code.hashCode();
    }

    @Override
    public String toString() {
        return String.format("%s %s %d", code, name, age);
    }

    public String getName() {
```

```
        return name;
    }

    public int getAge() {
        return age;
    }
}
```

## 1.4. Именик

Да се имплементира класа за именик `PhoneBook` со следните методи:

- `void addContact(String name, String number)` - додава нов контакт во именикот. Ако се обидеме да додадеме контакт со веќе постоечки број, треба да се фрли исклучок од класа `DuplicateNumberException` со порака `Duplicate number: [number]`. Комплексноста на овој метод не треба да надминува $O(log N)$ за $N$ контакти.
- `void contactsByNumber(String number)` - ги печати сите контакти кои во бројот го содржат бројот пренесен како аргумент во методот (минимална должина на бројот [number] е 3). Комплексноста на овој метод не треба да надминува $O(log N)$ за $N$ контакти.
- `void contactsByName(String name)` - ги печати сите контакти кои даденото име. Комплексноста на овој метод треба да биде $O(1)$.

Во двата методи контактите се печатат сортирани лексикографски според името, а оние со исто име потоа според бројот.

*Решение (`PhoneBookTest.java`)*

```
package mk.ukim.finki.np.av9;

import java.util.*;

public class PhoneBookTest {

    public static void main(String[] args) {
        PhoneBook phoneBook = new PhoneBook();
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        scanner.nextLine();
        for (int i = 0; i < n; ++i) {
            String line = scanner.nextLine();
            String[] parts = line.split(":");
            try {
                phoneBook.addContact(parts[0], parts[1]);
            } catch (DuplicateNumberException e) {
                System.out.println(e.getMessage());
            }
        }
        while (scanner.hasNextLine()) {
            String line = scanner.nextLine();
            String[] parts = line.split(":");
```

```
            if (parts[0].equals("N")) {
                phoneBook.contactsByNumber(parts[1]);
            } else {
                phoneBook.contactsByName(parts[1]);
            }
        }
    }

}

class PhoneBook {
    Set<String> allNumbers;
    Map<String, Set<Contact>> byNumberParts;
    Map<String, Set<Contact>> byName;
    static Comparator<Contact> comparator = Comparator.comparing(Contact::getName)
.thenComparing(Contact::getNumber);

    public PhoneBook() {
        byNumberParts = new TreeMap<>();
        byName = new HashMap<>();
        allNumbers = new HashSet<>();
    }

    public void addContact(String name, String number) throws DuplicateNumberException {
        if (allNumbers.contains(number))
            throw new DuplicateNumberException(number);

        Contact contact = new Contact(name, number);

        Set<Contact> contactsByName = byName.computeIfAbsent(name, key -> new TreeSet<>
(comparator));
        contactsByName.add(contact);

        List<String> keys = getKeys(number, 3);
        for (String key : keys) {
            Set<Contact> contactsByNumber = byNumberParts.computeIfAbsent(key, k -> new
TreeSet<>(comparator));
            contactsByNumber.add(contact);
        }

    }

    private List<String> getKeys(String key, int minLen) {
        List<String> result = new ArrayList<>();
        for (int i = 0; i <= key.length() - minLen; ++i) {
            for (int len = minLen; len <= (key.length() - i); ++len) {
                String k = key.substring(i, i + len);
                result.add(k);
            }
        }
        return result;
    }

    // O(log N)
    public void contactsByNumber(String number) {
        if (byNumberParts.containsKey(number)) {
            byNumberParts.get(number).forEach(System.out::println);
        }
    }

    // O(1)
    public void contactsByName(String name) {
        if (byName.containsKey(name)) {
            byName.get(name).forEach(System.out::println);
        }
    }

    public static void main(String[] args) {
        new PhoneBook().getKeys("075444123", 3);
    }
}

class Contact {
    String name;
    String number;
```

```
    public String getName() {
        return name;
    }

    public String getNumber() {
        return number;
    }

    public Contact(String name, String number) {
        this.name = name;
        this.number = number;
    }

    @Override
    public String toString() {
        return String.format("%s %s", name, number);
    }

}

class DuplicateNumberException extends Exception {

    public DuplicateNumberException(String number) {
        super(String.format("Duplicate number: %s", number));
    }

}
```

# 2. Изворен код од примери и задачи

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

Source Code ZIP