Algoritmos e Estruturas de Dados

2ª Série

Problema: Pandemia Z (Primeira Parte)

N. 45827 Nome Daniel Azevedo

Licenciatura em Engenharia Informática e de Computadores

Semestre de Verão 2019/2020

18/05/2020

Índice

Conteúdo

Introdução

O programa simula o controlo da informação referente a pacientes testados durante uma pandemia. O centro em questão pretende obter resposta para as seguintes perguntas: quantas e quais as pessoas que podem ficar infetadas com este vírus; dada uma localidade, e se a mesma poderá ser considerada uma zona de risco. Cada amostra é anonimizada e registada com um identificador, localidade, os indivíduos que a pessoa contacta diariamente, a data em que foi produzida, e o estado clínico do paciente relativamente ao vírus.

O objetivo deste trabalho é armazenar e tratar as informações relativas a cada individuo, para que os seus valores possam estar em constante atualização, e possam ser extraídos de forma rápida.

Pandemia Z **(Primeira Parte)**

Esta aplicação têm 4 funcionalidade disponíveis para o usuário: **add sourcefile.EZ** com o qual pode adicionar um novo ficheiro; **town_hall** que permite obter uma contagem por concelho de quantos indivíduos estão infetados, suscetíveis ou resistentes; **top k** que lista no standard output os k concelhos com mais infetados do programa; e por fim **exterminate** que encerra a aplicação avisando o utilizador antes.
Duas das 4, como podemos reparar requerem informação relativa ao Distrito, o que sugere uma hierarquia na forma como a informação deve estar disposta. Então para o mesmo efeito, foi criado duas classes individuas: Classe **Person** que representa a informação do individuo testado no centro, e a Classe **District** que representa apenas os números relativos aos resultados dos testes por distrito. Havia a hipótese de haver indivíduos já testados, a serem testados novamente no centro, o que me obrigava a atualizar a informação anterior desse paciente. Daí ter adotado pela estrutura de armazenamento HashMap, pois quando a key ou value é igual a nova a inserir, o valor no seu interior que correspondente a essa chave ou valor em particular é substituída pelo seu novo valor.

O método **insertInformation** é responsável por extraír a informação contida nos ficheiro .EZ, vai "ler" cada linha do ficheiro, e reparti-la de forma a que esta possa ser enviada para o Construtor da Classe Person, e assim possa ser criado o objeto associado à mesma. Contudo, não nos podemos esquecer de atualizar respetivos valores do Distrito desta pessoa. Para esse efeito crio o objeto auxiliar do tipo District para identificar se é necessário adicionar uma nova chave ao mapa, ou apenas atualizar os atributos relativos ao objeto contido nesta.

O método **userInterface** apenas têm o preceito de mostrar as diversas funcionalidades explicadas anteriormente. Encontra-se em loop, até decisão contrária do usuário terminar o programa ao introduzir a opção exterminate. O método auxiliar ask é usado para obter a resposta fornecida pela consola. Apenas existem 2 opções no switch mais pertinentes, top e town_hall. Town_hall percorre as diferentes chaves do HashMap district, e imprime os valores associados. Top necessitou de mais trabalho e atenção, foi-me sugerido pelo Docente a utilização de um algoritmo que não ordena se os Distritos em relação ao seu número de infetados, mas sim organiza-se. Isto porque só precisamos do k com mais infetados, e não nos é imposto a condição de estarem ordenados. Daí ter adotado o algoritmo *MaxHeap*, o qual têm um desempenho muito bom (n log (n)) em qualquer um dos seus casos (Ω, Θ, ou O). Iremos verificar esta propriedade do algoritmo nos resultados experimentais a seguir.

O código do programa está disposto aqui:

- **Código Java**

```
public class ZPandemic {

   private static Map<String, Person> people = new HashMap<>();
   private static Map<String, District> district = new HashMap<>();
   private static boolean on = true;
   private static int subjectsTotalNumber;




   public static void main(String[] args) throws IOException {
      BufferedReader fileInput = new BufferedReader(new FileReader(args[0]));
      Scanner scanner = new Scanner(fileInput);
      insertInformation(scanner);
      userInterface();
   }

   public static void insertInformation(Scanner in) {

      subjectsTotalNumber += Integer.parseInt(in.nextLine().trim());
      while(in.hasNext()) {
         String subject = in.nextLine();
         String[] values = subject.split(" ");
         Person tested = new Person(Integer.parseInt(values[0]),values[1],values[2].charAt(0),
values[3]);
         District districtAux = district.get(values[1]);
         if(districtAux == null) {
            districtAux = new District(values[1]);
            district.put(values[1], districtAux);
         }
         switch (values[2].charAt(0)) {
            case 'I':
               districtAux.incrementInfected(districtAux.getInfected());
               break;
            case 'R':
               districtAux.incrementResistant(districtAux.getResistant());
               break;
            case 'S':
               districtAux.incrementSusceptible(districtAux.getSusceptible());
               break;
            default:
               System.out.println("Invalid clinical state on the file");
               break;
         }
         people.put(values[0], tested);
      }
   }

   public static void userInterface() throws IOException {
      while(on) {
         String screen = "\n##########Insert your choice########## \n" +
               "Add new file ?\n--->Type: add yourFileName\n\n" +
```

```
                "Show the information of all Districts ?\n--->Type: town_hall\n\n" +
                "Get k districts with more infected ?\n--->Type: top k\n\n" +
                "Shut down masterPiece ?\n--->Type: exterminate\n";
        Scanner in = new Scanner(System.in);
        String answer = ask(screen, in);
        answer.trim();
        String[] aux = answer.split(" ");
        switch (aux[0]) {
            case "add":
                BufferedReader getThem = new BufferedReader(new FileReader(aux[1]));
                Scanner populate = new Scanner(getThem);
                insertInformation(populate);
                break;
            case "town_hall":
                district.entrySet().forEach(entry->{
                    System.out.println("" + entry.getValue().toString());
                });
                break;
            case "top":
                getKBiggest(Integer.parseInt(aux[1]));
                break;
            case "exterminate":
                System.out.println("Switching off");
                try {
                    TimeUnit.SECONDS.sleep(2);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                on = false;
                break;
            default:
                System.out.println("Command Invalid! Please try again");
        }
        System.out.println("\nHow many subjects on the system = "+subjectsTotalNumber);
    }
}

private static String ask(String message, Scanner in) {
    System.out.println(message);
    String userInput = in.nextLine();
    return userInput;
}

//TODO: Penso que era a isto que a Professora se referia
private static void getKBiggest(int k) {
    if(k > 20 ||k  < 0 ||k > district.size()){
        System.out.println("Invalid k, must be between 0 and 20");
        System.exit(0);
    }
    District[] values = new District[district.size()];
    int counter = 0;
    for(District d: district.values()) {
        values[counter++] = d;
    }
    Comparator<District> cmp = new Comparator<>() {
        @Override
        public int compare(District, District t1) {
            return district.getInfected() - t1.getInfected();
        }
```

```
        @Override
        public boolean equals(Object o) {
            return false;
        }
    };
    buildMaxHeap(values, district.size(), cmp);
    for(int i = 0; i < k; ++i){
        System.out.println(values[i]);
    }
}

public static void buildMaxHeap(District[] h, int n, Comparator<District> cmp) {
    for (int i = n / 2 - 1; i >= 0; i--) {
        if(h[i]!=null) maxHeapify(h, i, n, cmp);
    }
}

public static void maxHeapify(District[] h, int i, int n, Comparator<District> cmp) {
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    int largest;
    if (l < n && cmp.compare(h[l], h[i]) > 0)
        largest = l;
    else largest = i;
    if (r < n && cmp.compare(h[r], h[largest]) > 0)
        largest = r;
    if (largest != i) {
        exchange(h, i, largest);
        maxHeapify(h, largest, n, cmp);
    }
}

private static void exchange(District[] h, int i, int j) {
    District aux = h[i];
    h[i] = h[j];
    h[j] = aux;
}
}
```

```
public class Person {

    private int id;
    private String district;
    private char clinicalState;
    private String date;
    private String contactedPersons;


    public Person(int id, String district, char clinicalState, String date) {
        this.id = id;
        this.district = district;
        this.clinicalState = clinicalState;
        this.date = date;
    }

    public String toString() {
        return "PersonId = "+id+" District = "+district;
    }
```

```
public class District {

   private String district;
   private int infected = 0;
   private int resistant = 0;
   private int susceptible = 0;

   public District(String district) {

      this.district = district;
   }

   public String toString() {
      return "\nDistrict = "+district+"\nSusceptible = "+susceptible+"\nResistant =
"+resistant+"\nInfected = "+infected;
   }

   public String getDistrict(){
      return district;
   }

   public int getInfected() {
      return this.infected;
   }

   public int getResistant() {
      return resistant;
   }

   public int getSusceptible() {
      return susceptible;
   }

   public void incrementInfected(int previous) {
      this.infected = previous + 1;
   }

   public void incrementResistant(int previous) {
      this.resistant = previous + 1;
   }

   public void incrementSusceptible(int previous) {
      this.susceptible = previous + 1;
   }
}
```

Resultados experimentais

De forma a poder testar o programa corretamente, tive que criar o programa a seguir. Este permite-me criar ficheiros de grandes dimensões e com valores aleatórios enquadrados no problema. Basta modificar o campo *int howManyPersons* (número de pessoas pretendidas) e *String fileName* (nome do ficheiro pretendido) para os objetivos pretendidos, e o ficheiro é gerado.

- **Código Java**

```
public class filesGenerator {

   final static int howManyPersons = 1000000;
   final static String[] districts = {"Aveiro", "Porto", "Braga", "Viana_do_Caselo", "Vila_Real",
"Bragança", "Castelo_Branco",
         "Guarda", "Viseu", "Évora", "Santarém", "Lisboa", "Setúbal", "Beja", "Faro", "Portalegre",
"Leiria", "Coimbra", "Madeira", "Açores"};
   final static Character[] clinicalState = {'S', 'I', 'R'};
   final static String fileName = "file4.EZ";

   public static void main(String[] args) throws FileNotFoundException {
      File = new File(fileName);
      PrintStream output = new PrintStream(file.getName());
      output.println(howManyPersons);
      for (int i = 0; i < howManyPersons; ++i) {
         int whichDistrict = (int) (Math.random() * 20);
         int whichClinicalState = (int) (Math.random() * 3);
         int howManyPersonsContacted = (int) (Math.random() * 7);
         StringBuilder contactedPersonsID = new StringBuilder();
         for (int j = 0; j < howManyPersonsContacted; ++j) {
            contactedPersonsID.append((int) (Math.random() * howManyPersonsContacted) + " ");
         }
         int id = (int) (Math.random() * howManyPersons);
         RandomDates rdn = new RandomDates();
         output.println("" + id + " " + districts[whichDistrict] + " " + clinicalState[whichClinicalState] +
" " + rdn.date + " " + contactedPersonsID);
      }
   }
}
```

```
public class RandomDates {

   public String date;

   public RandomDates(){
      this.date =  ""+createRandomDate(1900, 2000);
   }

   public int createRandomIntBetween(int start, int end) {
      return start + (int) Math.round(Math.random() * (end - start));
   }
```

```
    public LocalDate createRandomDate(int startYear, int endYear) {
        int day = createRandomIntBetween(1, 28);
        int month = createRandomIntBetween(1, 12);
        int year = createRandomIntBetween(startYear, endYear);
        return LocalDate.of(year, month, day);
    }
}
```

Criei 6 ficheiros com o seguinte número de indivíduos:

file1.EZ – 10.000 (335 KB);

file2.EZ – 100.000 (3 446 KB);

file3.EZ – 500.000 (17 662 KB);

file4.EZ – 1.000.000 (35 442 KB);

file5.EZ – 10.000.000 (364 143 KB);

theBigO.EZ – 100.000.000 (3 739 186 KB).

Para medir os tempos em cada um dos testes, foi utilizado os métodos estáticos:

```
long initTime = System.currentTimeMillis();
long elapsedTime = System.currentTimeMillis()-initTime;
System.out.println("\nTime taken to run:");
System.out.print(new SimpleDateFormat("mm:ss.SSS").format(new Date(elapsedTime)));
```

O ficheiro theBigO.EZ não serviu para os resultados, pois resultou nos seguintes erros:

Exception in thread "main" java.lang.OutOfMemoryError: Java heap space

at java.base/java.nio.HeapCharBuffer.toString(HeapCharBuffer.java:688)

at java.base/java.nio.CharBuffer.toString(CharBuffer.java:1626)

at java.base/java.util.regex.Matcher.toMatchResult(Matcher.java:274)

at java.base/java.util.Scanner.match(Scanner.java:1399)

at java.base/java.util.Scanner.nextLine(Scanner.java:1652)

at series.serie2.ZPandemic.insertInformation(ZPandemic.java:36)

at series.serie2.ZPandemic.userInterface(ZPandemic.java:81)

at series.serie2.ZPandemic.main(ZPandemic.java:29)

Então criei outro ficheiro file6.EZ com apenas 50.000.000 (1 864 119 KB) pessoas de forma a substituí-lo. Continuou a dar erro, então fiquei me apenas pelos 5 ficheiros.

Primeiro gostava de analisar os tempos de leitura do ficheiros com diferentes objetos. Neste caso vou comparar *Scanner* com *BufferedReader* em termos de custo de tempo na leitura dos ficheiros anteriores.

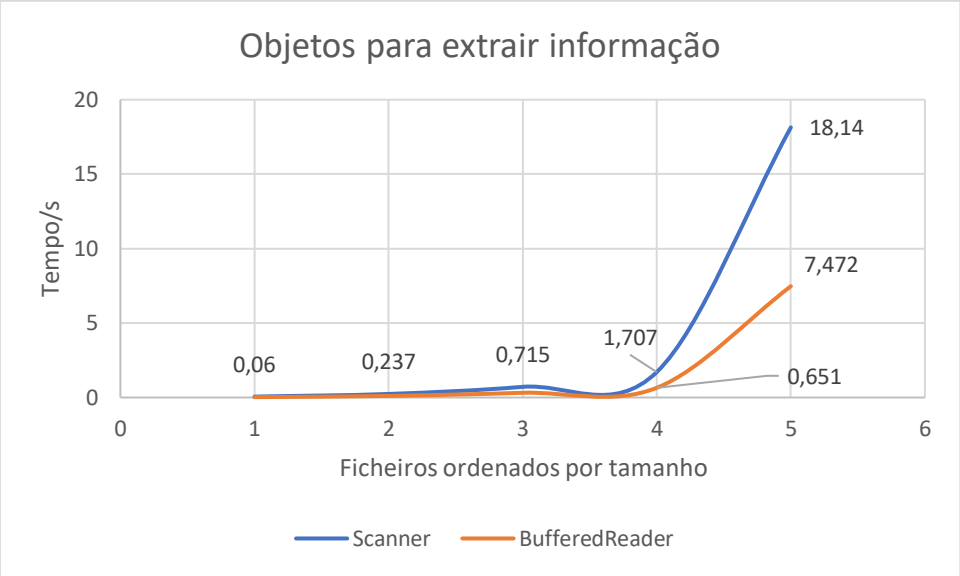

Figura 1 – Tempo de extração de informação dos ficheiros (máximo de 20 Distritos)

Como podemos ver, Scanner ao longo do tempo aproxima-se do dobro do custo de tempo em relação ao BufferedReader. Este fator levou me a substituir o Scanner anteriormente implementado no trabalho por BufferedReader.

Indivíduos	Tempo/s
10012	0,010
110012	0,011
610012	0
1610012	0,001
11610012	0,011

Figura 2 – Custo em segundos da funcionalidade town_hall em 20 Distritos

Esta funcionalidade têm um custo semelhante em qualquer uma das situações devido ao facto de haver no máximo 20 Distritos (Portugal). Logo independentemente do número de pessoas que o programa venha a obter, o número de objetos diferentes a listar irá sempre ser no máximo 20.

Todavia acrescentei o dobro de Distritos para simular valores noutros Países e para o âmbito de comparar resultados. Repliquei novamente os 5 ficheiros com o mesmo número de pessoas, mas aumentei o número de Distritos possíveis para 40. Estão aqui os novos resultados:

Indivíduos	Tempo/s
10012	0,011
110012	0,001
610012	0
1610012	0
11610012	0,001

Figura 3 - Custo em segundos da funcionalidade town_hall em 40 Distritos

Como podemos observar na figura 3, não criou alterações no tempo decorrido tempo, contudo aumentou a sua complexidade do espaço para o dobro (chaves no *HashMap)*. Os diferentes valores foram originados por variações do sistema em que aplicação foi executada (distintos processos a correr).

Mas à medida que o *HashMap* se dirige para valores muitos altos não tenho dúvidas que haverá alterações no tempo de execução da funcionalidade. A sua complexidade do tempo no pior caso é O(n), e como a sua complexidade do espaço também é O(n), sabemos que à medida que nos aproximamos de valores muitos elevados (chaves diferentes ou *hash code* iguais) na estrutura de dados, também nos aproximamos de O(n). Felizmente este pior caso não surge frequentemente.

O mesmo vai acontecer com a funcionalidade top k, pois a situação é idêntica. Top k têm custo O(k log n). Todavia no Heap, se aumentarmos gradualmente o seu número de elementos a sua complexidade no espaço continua a ser O(1). E independentemente dos casos do algoritmo (Ω, Θ, O) o seu custo temporal irá ser sempre k log n. Podemos averiguar esta situação nos resultados a seguir:

Indivíduos	Tempo/s
10012	0,0014
100012	0
500012	0,001
1000012	0,009
10000012	0,009

Figura 4 – TOP 20 em 20 distritos

Indivíduos	Tempo/s
10012	0,01
100012	0
500012	0
1000012	0
10000012	0,001

Figura 5 – TOP 5 em 20 distritos

Indivíduos	Tempo/s
10012	0
100012	0
500012	0,001
1000012	0
10000012	0

Figura 6 – TOP 1 em 20 distritos

Ainda tentei com um ficheiro de 100 Distritos diferentes, mas continuava a obter os mesmos resultados.

Indivíduos	Tempo/s
10012	0,002
110012	0,001
610012	0,001
1610012	0,001
11610012	0,002

Figura 7 – TOP 80 em 100 distritos

Indivíduos	Tempo/s
10012	0,001
110012	0
610012	0
1610012	0
11610012	0

Figura 8 – TOP 20 em 100 distritos

Indivíduos	Tempo/s
10012	0,001
110012	0
610012	0
1610012	0,001
11610012	0,001

Figura 9 – TOP 5 em 100 distritos

Indivíduos	Tempo/s
10012	0,001
110012	0
610012	0,001
1610012	0
11610012	0

Figura 10 – TOP 1 em 100 distritos

O processo mais dispendioso de todo o algoritmo é a leitura de ficheiros, e como desconheço a composição dos algoritmos responsáveis pela mesma, apenas posso presumir os seus custos. Através da leitura do gráfico 1 diria que o custo de Scanner se assemelha a O(n^2) e de BufferedReader a O(n), pelo menos é o que 10 milhões de leituras e a orientação das funções ao longo do tempo nos insinuam.

Conclusão

Achei muito adequado e motivador o tema do trabalho, pelo facto de estar relacionado com a pandemia presente no Mundo, o COVID-19. Inicialmente tive dificuldades em perceber como os HashMap funcionavam, ou como disponham a sua informação. Algumas análises feitas por mim podiam talvez ser mais aprofundadas, como é o caso de não ter conseguido comprovar o pior caso no HashMap, mas penso que seria necessário muitos mais dados para o efeito. Pude também observar que os algoritmos implementados não dispõem de memória muito elevada, pois 50 milhões de elementos já me impossibilitava de prosseguir o uso dos mesmos. E que nem todos os algoritmos da biblioteca com funções idênticas se comportam de maneira igual ao longo do seu tempo e espaço.