Duas abordagens gerais baseadas em mineração de dados para recomendação de nomes

Érick Oliveira Rodrigues

Instituto de Ciência da Computação – Universidade Federal Fluminense (UFF)

Resumo. *Este artigo descreve duas abordagens que objetivam construir uma lista de nomes recomendados baseando-se principalmente nas características de um ou vários nomes de entrada. As abordagens aqui formalizadas também foram efetivamente utilizadas para construir um arquivo texto (para posterior submissão) contendo recomendações de nomes baseadas no histórico de usuários e nomes fornecidos pelo domínio (nameling.net) no que tange ao 15º Discovery Challenge – Recommeding Given Names promovido pela PKDD.*

Introdução

Como entrada, duas bases de dados oficiais foram disponibilizadas no âmbito do desafio de recomendação. Uma das bases contém exatamente 60 923 usuários distintos distribuídos em 515 849 tuplas que indicam as atividades que cada usuário realizou em um determinado tempo específico, assim como o nome acessado associado a essa atividade. A segunda base de dados contém todos os nomes que poderiam impreterivelmente ser recomendados a um usuário arbitrário dado o seu histórico. Essa base de dados contém 36 434 nomes, cada um em uma tupla diferente.

1. Primeira Abordagem

A primeira abordagem se concentrou em avaliar os pares ou implicações frequentes que ocorriam dentre todos os usuários. Dado nomes arbitrários n_1 e n_2; $n_1 \Leftrightarrow n_2$ (n_1 implica n_2 / n_2 implica n_1) tem suporte 1 se n_1 e n_2 aparecem no histórico de apenas um usuário. O suporte está evidentemente relacionado à quantidade de vezes que os nomes ocorrem conjuntamente dentre usuários distintos.

1.1 Tratamento da Base de Dados

Nota-se que a base de dados de atividades por usuário contém muitos nomes que não estão na base de nomes, de forma semelhante, também existem muitos nomes que podem ser recomendados que não estão na base de atividades. Um análise desse detalhe descrito foi realizada e a interseção de nomes comuns às duas bases está ilustrada abaixo.

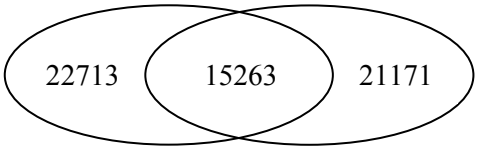

Figura 1. O conjunto da esquerda representa a quantidade de nomes distintos na base de atividades (37 976). Já no conjunto da direita, temos a quantidade de nomes distintos contidos na base de nomes (36 434).

Nota-se pela Figura 1 que apenas 15 263 nomes estão em ambas as bases de dados. Logo, realizamos um tratamento para apagar, da base de atividades, todas as tuplas que continham nomes que não estavam na base de nomes. Ou seja, após o tratamento, a base de atividades conterá 15 263 nomes distintos distrubuídos em 315 331 tuplas. Fica evidente que uma quantidade enorme de nomes está sendo desconsiderada nesta abordagem.

1.2 Síntese do Arquivo de Implicações Frequentes

Após a exclusão de diversas tuplas, processo evidenciado no tópico 1.1, a lógica das implicações frequentes de nomes foi colocada em efetiva prática. Nesse momento, construímos um novo arquivo de texto contendo tuplas com o formato descrito no item 1. Para cada ocorrência das implicações, acrescentamos o suporte do par de nomes em uma unidade.

```
Emil<->Oskar,115
Oskar<->Emil,115
Anna<->Emma,95
Emma<->Anna,95
Johanna<->Katharina,89
Katharina<->Johanna,89
Johanna<->Elisabeth,85
Mia<->Emma,85
Emma<->Mia,85
...
```

Figura 2. Fragmento do arquivo de pares de nomes frequentes. O número após a vírgula indica o suporte de ocorrência, e vírgula em si pode ser ignorada, assim como o símbolo de implicação (<->). Nota-se que a cada ocorrência temos a mesma implicação duplicada com os nomes invertidos de posição, isso se deve pelo fato da busca no arquivo ficar facilitada.

O processo de construção do arquivo ilustrado pela Figura 2 é simples. Criamos uma *hashtable* global que armazena as implicações como *key* e o suporte como *value*. A cada histórico de um usuário qualquer, fazemos o arranjo dos nomes que o mesmo acessou. Cada arranjo se tornará uma implicação frequente. Se essa implicação gerada não está

presente na *hashtable* global, então adicionamos a mesma com suporte 1. Caso contrário, incrementamos o *value* (suporte) da regra já existente.

1.3 Síntese do Arquivo de Recomendações

Dado que já temos o arquivo com os pares frequentes de nomes, o objetivo agora é gerar um arquivo final de recomendação de nomes para um dado histórico de um usuário arbitrário. Para construir este arquivo utilizamos as implicações frequentes geradas para ordenar, por transitividade, os nomes que devem ser recomendados a cada usuário. Supondo que um usuário contenha n tuplas de atividades, geramos, para k ($1 \leq k \leq n$) nomes diferente entre si dentre essas tuplas, k listas. Dessas k listas pegamos a lista referente ao último nome que ocorre no histórico do usuário e a tratamos como a lista principal. Em relação ao resto das k -1 listas, avaliamos quais nomes ocorrem mais que uma vez em pelo menos duas listas. Para cada um desses nomes, reordenamos a posição dos mesmos na lista principal, respeitando o suporte dos nomes como critério principal de ordenação, e utilizando a ordem em que os nomes foram gerados como critério de desempate. De forma a gerar para cada tupla uma lista de recomendações, partimos da seguinte lógica:

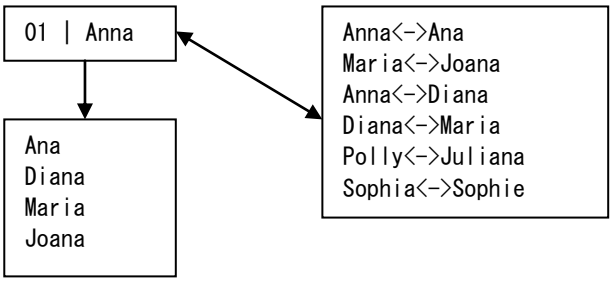

Figura 3. Representação da construção da lista de recomendações dado um nome que o usuário acessou. No canto esquerdo superior temos um dos nomes presentes no histórico do usuário. Ao lado direito temos o arquivo de pares frequentes servindo como base para a construção da lista de recomendações para o usuário 01.

Por transitividade, montamos a lista de recomendações. Anna se liga a Ana e depois a Diana. Ana não se liga a mais ninguém. Diana, o segundo nome que já está na lista de recomendação, se liga a Maria e mais ninguém. Maria se liga a Joana. Dessa forma, temos a lista de recomendação para o usuário 01 completa.

2. Segunda Abordagem

A segunda abordagem se baseou em tentar considerar os nomes que não foram avaliados na abordagem anterior por não pertencerem às duas bases de dados de entrada. Aqui, portanto, nenhum tratamento da base de dados foi efetivamente realizado. Diferentemente da abordagem anterior, precisamos avaliar todos os nomes passíveis a

serem escolhidos para serem recomendados ao usuário. Dessa forma, percorremos toda a base de nomes uma vez e avaliamos quais nomes seriam ideais a serem retornados.

2.1 Enriquecimento dos Nomes

Percebe-se que, intrinsecamente aos nomes, somos capazes de extrair outras características do mesmo e não somente o nome como um todo. Logo, de forma a não errar na classificação e no enriquecimento desses nomes utilizando heurísticas duvidosas, apenas atributos que poderiam ser extraídos com uma heurística muito forte, ou seja, com uma heurística de ampla aceitação, foram considerados. Dessa forma, transformamos a tupla "nome" em uma nova tupla com os seguintes atributos derivados do nome: tamanho do nome, quantidade de vogais, quantidade de consoantes, caractere inicial e caractere final.

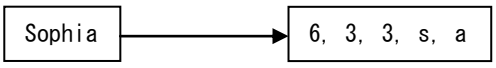

Figura 4. Enriquecimento de um nome arbitrário conforme os critérios estabelecidos no texto.

2.2 Similaridade de Nomes

Um algoritmo para analisar a similaridade entre *strings* foi utilizando para avaliar a semelhança entre os distintos nomes. O algoritmo em questão foi o StrikeAMatch [1]. A aplicação da análise de semelhança se deu da seguinte forma:

i. Para cada usuário **u**:
  i) Para cada nome **k** acessado pelo usuário **u**:
    (1) Para cada nome **n** ($1 \leq \mathbf{n} \leq 36\,434$) da base de nomes
      (a) Avaliar o nível de similaridade (StrikeAMatch) do nome da posição **n** com o nome **k** e guardar em uma variável relacionada ao nome da posição **n** a sua respectiva pontuação.

No fim da execução deste algoritmo, teremos, para cada usuário, toda a lista de nomes disponíveis não ordenada. Cada nome terá uma pontuação específica.

2.3 Implicações Frequentes por Atributo

Por fim, utilizando a mesma abordagem de pares frequentes apresentada anteriormente, embora aplicada de uma forma não tão semelhante, extraímos regras de associação dos valores de um mesmo atributo. Uma regra muito frequente encontrada, por exemplo, foi a implicação "a ⇔ a" para o caractere de término dos nomes. Um nome que termina em

“a”, geralmente é um nome feminino, e, quem escolhe nomes femininos, de certa forma tende a escolher outros nomes femininos terminados em “a”.

```
Atributo 1: 7<->6 <sup: 221915>
Atributo 1: 4<->5 <sup: 212662>
Atributo 2: 2<->2 <sup: 1029968>
Atributo 2: 2<->3 <sup: 833350>
Atributo 3: 5<->4 <sup: 291648>
Atributo 3: 2<->3 <sup: 219270>
Atributo 4: a<->a <sup: 68875>
Atributo 4: m<->m <sup: 62433>
Atributo 5: a<->a <sup: 238092>
Atributo 5: n<->a <sup: 144919>
...
```

Figura 5. Um fragmento do arquivo que contém as regras frequentes por atributo. O “<sup : n>” indica o suporte com que a regra ocorreu. Lembrando que a ordem dos atributos é a mesma descrita no item 3.1.

Com essas regras geradas, as utilizamos para “reordenar” a posição dos nomes na lista final de recomendações por usuário. Ou seja, uma quantidade de pontos a mais também foi atribuído aos nomes que apresentaram características dentro de algumas regras selecionadas de acordo com o histórico do usuário. Para um certo usuário conter uma regra, o mesmo precisa ter acessado algum(ns) nome(s) que satisfaz(em) a mesma em qualquer um dos lados da seta de “implicação”. Após selecionar as regras específicas de cada usuário, aplicamos a mesma aos nomes que estão sendo avaliados durante a contagem de pontos descrita no item 2.2, aumentando a pontuação dos nomes que vierem a conter as características da regra. Após o término de toda a pontuação reordenamos a lista de forma decrescente em função da quantidade de pontos de cada nome e, finalmente, teremos uma lista com os nomes mais fortemente recomendáveis no início da lista.

References

1. White, Simon. “How to Strike a Match”, http://www.catalysoft.com/articles/StrikeAMatch.html.

Apêndice

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Scanner;


public class Main {
    public static final int tamanholista = 1000;

    public static void main(String[] args) {

        Processos.etapa6_ordenar();

    }
    public static class Processos{


        public static void etapa1(){ //tratamento da base inicial, arrancando todos os nomes que não estejam presentes
na base de nomes
            ArrayList<String> nomes = new ArrayList<String>();
            ArrayList<String> actnomes = new ArrayList<String>();
            ArrayList<String> nomesduasbases = new ArrayList<String>();

            String origem = "C:\\Users\\Érick\\Documents\\Plastino_Desafio\\nameling.trainData", destino =
"C:\\Users\\Érick\\Documents\\Plastino_Desafio\\dest.txt";
            String nomeorigem = "C:\\Users\\Érick\\Documents\\Plastino_Desafio\\namelist.txt";

            String line = null;


            try {

                BufferedReader in = new BufferedReader(new InputStreamReader(new FileInputStream(origem), "UTF8"));
                BufferedReader innomes = new BufferedReader(new InputStreamReader(new FileInputStream(nomeorigem),
"UTF16"));

                while ((line = innomes.readLine()) != null){
                    nomes.add(line);
                }
                innomes.close();


                BufferedWriter out = new BufferedWriter(new OutputStreamWriter
                          (new FileOutputStream(destino),"UTF8"));

                String linha[];
                while((line = in.readLine()) != null) {
                    System.out.println(line);

                    line = line.replaceAll("\\t\\t", "");
                    line = line.replaceAll(",", "");
                    line = line.replaceAll("'", "´");
                    line = line.replaceAll("\"", "");
                    linha = line.split("\\t"); //tab

                    if (linha.length > 3){
                        if (!linha[1].equals("LINK_CATEGORY_SEARCH") && isNumeric(linha[3]) && !linha[2].contains("%")
                                && !linha[2].contains("/")){
                            if (!actnomes.contains(linha[2])) actnomes.add(linha[2]); //contagem de nomes repetidos
                            if (nomes.contains(linha[2]) && !nomesduasbases.contains(linha[2]))
nomesduasbases.add(linha[2]); //nomes presentes nas duas bases

                            if (nomes.contains(linha[2])){ //printar apenas uma lista que contenha nomes que estão na
base de nomes
                                for (int i = 0; i < linha.length; i++){
                                    if (i != linha.length - 1) out.write(linha[i] + ",");
                                    else out.write(linha[i]);
                                }

                                out.newLine();
                            }
                        }
                    }

                }

                out.close();

                in.close();


                System.out.println(nomes.size() + " nomes totais (base de nomes).");
                System.out.println(actnomes.size() + " nomes distintos na base de atividades. \n" +
                        nomesduasbases.size() + " nomes presentes nas duas bases.");

            }
```

```
            catch(FileNotFoundException ex) {

            }
            catch(IOException ex) {

            }

        }


        public static void etapa2(){//gerar e ordenar os conjuntos frequentes de nomes acessados por usuário
            Hashtable<Integer, Hashtable<String, Integer>> usuarios = new Hashtable<Integer, Hashtable<String,
Integer>>();

            String origem = "C:\\Users\\Érick\\Documents\\Plastino_Desafio\\dest.txt", destino =
"C:\\Users\\Érick\\Documents\\Plastino_Desafio\\dest2.txt";


            String line = null;

            try {

               BufferedReader in = new BufferedReader(new InputStreamReader(new FileInputStream(origem), "UTF8"));
               BufferedWriter out = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(destino),"UTF8"));

              String linha[];
              while((line = in.readLine()) != null) {
                  System.out.println(line);
                  linha = line.split(","); //virgula
                  //linha[0] = linha[0].substring(1); //bug fix
                  int usu = Integer.parseInt(linha[0]);
                  //pesos das atividades
                  int peso = 1;
                  if (linha[1].equals("ADD_FAVORITE") || linha[1].equals("NAME_DETAILS")) peso = 2;
                  //fim peso
                  if (!usuarios.containsKey(usu)) {
                      Hashtable<String, Integer> aux = new Hashtable<String, Integer>();
                      aux.put(linha[2], peso);
                      usuarios.put(usu, aux);
                  }else{
                      if (usuarios.get(usu).containsKey(linha[2])){ //se o usuário já contem o nome listado
                          if (usuarios.get(usu).get(linha[2]) < peso){ // se o peso de antes é menor que o peso novo
                              usuarios.get(usu).remove(linha[2]);
                              usuarios.get(usu).put(linha[2], peso);
                          }
                      }else{
                          usuarios.get(usu).put(linha[2], peso);
                      }
                  }
              }
              System.out.println(usuarios.size());
              in.close();


              Iterator it = usuarios.entrySet().iterator();
              int contagemgeral = 0;
              while (it.hasNext()) {
                  contagemgeral ++; System.out.println("Porcentagem da análise do array de usuários: " +
contagemgeral*100/usuarios.size());
                  Map.Entry entry = (Map.Entry) it.next();
                  Integer key = (Integer) entry.getKey();
                  Iterator it2 = usuarios.get(key).entrySet().iterator();

                  short cont = 1;
                  while (it2.hasNext()){
                      Map.Entry entry2 = (Map.Entry) it2.next();
                      String key2 = (String) entry2.getKey();
                      Integer peso2 = usuarios.get(key).get(key2);

                      for (int i=cont; i < usuarios.get(key).size(); i++) {
                          String key21 = (String) usuarios.get(key).keySet().toArray()[i];
                          Integer peso21 = (Integer) usuarios.get(key).values().toArray()[i];

                          if (peso2 != peso21) peso2 = 1; //se os dois não tiverem peso alto, então a relação dos dois
juntos é fraco
                          adicionarFreq(key2, key21, peso2);
                      }
                      cont ++;
                  }

              }

              //ordenar implicações frequentes
              ArrayList<Vector2> freqordenados = new ArrayList<Vector2>();
              Object[] keyset = frequentes.keySet().toArray();
              Object[] values = frequentes.values().toArray();
              for (int i=0; i<frequentes.size(); i++){

                  int j = 0; boolean sair = false;
                  if (freqordenados.size() != 0){
                      while (j < freqordenados.size() && !sair){
                          if (freqordenados.get(j).peso > (Integer)values[i]) j++;
                          else  sair = true;
                      }
                  }

                  freqordenados.add(j, new Main().new Vector2((String)keyset[i], (Integer)values[i]));
                  System.out.println("Porcentagem da ordenação: " + freqordenados.size()*100/frequentes.size() + " --
Saída: " + freqordenados.get(j).relacao + "," + freqordenados.get(j).peso);
              }
```

```
            //escrever frequentes
            for (int i=0; i<freqordenados.size(); i++){
                String key = freqordenados.get(i).relacao;
                String saida = key + "," + freqordenados.get(i).peso;
                out.write(saida);
                out.newLine();

                System.out.println("Porcentagem de escrita: " + i*100/freqordenados.size() + " -- Saída: " + saida);
            }
            out.close();



        }catch(IOException ex) {}

    }

    public static Scanner in = new Scanner(System.in);
    public static ArrayList<String> dest3 = new ArrayList<String>();
    public static void etapa3(){//gerar arquivo final a partir dos conjuntos frequentes
        String origem, destino;
        int qntdavaliar[] = new int[2];


        //entrada de dados
        String entrada;
        System.out.println("Qual o diretório do arquivo dest.txt e dest2.txt?");
        entrada = in.nextLine();
        origem = (entrada.length() == 0) ? "C:\\Users\\Érick\\Documents\\Plastino_Desafio\\dest.txt" : entrada +
"\\dest.txt";
        if (entrada.length() != 0) {
            dirdest2 = entrada + "\\dest2.txt";
            destino = entrada + "\\dest3.txt";
        }else destino = "C:\\Users\\Érick\\Documents\\Plastino_Desafio\\dest3.txt";
        System.out.println("Intervalo de usuários a ser avaliado, exemplo: '0-100', ou seja [0,100) neste caso.
Enter em branco avalia tudo.");
        entrada = in.nextLine();
        if (entrada.length() == 0){
            qntdavaliar[0] = -1; qntdavaliar[1] = -1;
        }else{
            qntdavaliar[0] = Integer.parseInt(entrada.split("-")[0]);
            qntdavaliar[1] = Integer.parseInt(entrada.split("-")[1]);
        }
        //fim da entrada de dados


        //origem = "C:\\Users\\Érick\\Documents\\Plastino_Desafio\\dest~lol.txt";


        //id, nome
        Hashtable<Integer, ArrayList<String>> nomesporuser = new Hashtable<Integer, ArrayList<String>>();


        String line = null;

        try {

            BufferedReader in = new BufferedReader(new InputStreamReader(new FileInputStream(origem), "UTF8"));
            BufferedWriter out = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(destino),"UTF8"));

            String linha[];
            while((line = in.readLine()) != null) {
                linha = line.split(",");
                //linha[0] = linha[0].substring(1); //bug fix

                if (!nomesporuser.containsKey(Integer.parseInt(linha[0]))){
                    ArrayList<String> nomes = new ArrayList<String>();
                    nomes.add(linha[2]);
                    nomesporuser.put(Integer.parseInt(linha[0]), nomes);
                }else{
                    nomesporuser.get(Integer.parseInt(linha[0])).add(linha[2]);
                }
            }

            in.close();



            //avaliar um intervalo de usuários
            int inicio = 0, fim = 0;
            if (qntdavaliar[0] != -1){ //se é para avaliar apenas um intervalo de usuários
                inicio = qntdavaliar[0];
                fim = qntdavaliar[1];
            }else fim = nomesporuser.size();


            //percorrer os usuários e avaliar as listas geradas para cada nome que o mesmo acessou
            ArrayList<Vector2> listafinal = new ArrayList<Vector2>();
            Iterator it = nomesporuser.entrySet().iterator(); int contagem = 0;
            while (contagem < inicio){
                contagem++;
                it.next();
            }
            int userid; contagem = inicio;
            while (it.hasNext() && (contagem < fim)){
                contagem ++; System.out.println("Porcentagem de escrita por usuários totais: " +
(contagem*100/nomesporuser.size()));
                Map.Entry entry = (Map.Entry) it.next();
```

```
                userid = (Integer) entry.getKey();
                ArrayList<ArrayList<Vector2>> listas = new ArrayList<ArrayList<Vector2>>();
                for (int j=0; j<nomesporuser.get(userid).size(); j++){//nomes para cada usuário
                    listas.add(gerarLista(nomesporuser.get(userid).get(j), nomesporuser.get(userid)));
                }

                listafinal = combinarListas(listas);


                //escrever arquivo
                out.write(Integer.toString(userid));
                for (int i=0; i<listafinal.size(); i++){
                    out.write("\t");
                    out.write(listafinal.get(i).relacao);
                }
                System.out.println(" ... Escreveu usuário " + (contagem - 1) + "...");
                out.newLine();

                listas.clear();
            }

            System.out.println("Finalizado.");

            out.close();


        }catch(IOException e){}

    }


    //Combinar resultados de listas diferentes, cada uma baseada em um nome
    public static ArrayList<Vector2> combinarListas(ArrayList<ArrayList<Vector2>> listas){
        Hashtable<String, Integer> todososnomes = new Hashtable<String, Integer>();
        Integer[] ocorrencianomesnaslistas = new Integer[listas.size()];
        int listaprioritaria = -1;

        for (int i=0; i<ocorrencianomesnaslistas.length; i++) ocorrencianomesnaslistas[i] = 0;

        if (listas.size() > 1){
            for (int i=0; i<listas.size(); i++){//todas as listas recebidas

                for (int j=0; j<listas.get(i).size(); j++){//todos os nomes dentro de cada lista
                    if (todososnomes.contains(listas.get(i).get(j).relacao)){
                        int qntd = todososnomes.get(listas.get(i).get(j).relacao) + 1;
                        todososnomes.remove(listas.get(i).get(j).relacao);
                        todososnomes.put(listas.get(i).get(j).relacao, qntd);

                        ocorrencianomesnaslistas[i] += 2;
                    }else{
                        todososnomes.put(listas.get(i).get(j).relacao, 1);
                        ocorrencianomesnaslistas[i] += 1;

                    }
                }

            }
        }


        //escolher lista prioritaria
        int maiorp = 0;
        for (int i=0; i<ocorrencianomesnaslistas.length; i++){
            if (ocorrencianomesnaslistas[i] > maiorp) listaprioritaria = i;
        }

        //ordenar nomes por ordem de maior ocorrência
        ArrayList<String> nomesordenados = new ArrayList<String>();
        while (todososnomes.size() > 0){
            int maior = 0, ocorrencia = 0;
            String chave = "", key = "";
            Iterator it = todososnomes.entrySet().iterator();
            while (it.hasNext()){
                  Map.Entry entry = (Map.Entry) it.next();
                  key = (String) entry.getKey();
                  ocorrencia = todososnomes.get(key);

                  if (ocorrencia > maior) {
                      maior = ocorrencia;
                      chave = key;
                  }
            }
            if (chave.length() == 0) chave = key;
            if (maior > 1) nomesordenados.add(0, chave); //nomes que aparecem em duas listas, ou seja, em mais de
uma lista apenas
            todososnomes.remove(chave);
        }

        //vetor tem que ser do tamanho da lista prioritaria, ou da lista final
        Vector2[] listaaux;
        if (listas.size() == 1) listaaux = new Vector2[listas.get(0).size()];
        else if (listas.size() == 2) listaaux = new Vector2[listas.get(1).size()];
        else {
            if (listaprioritaria == -1){
                int indice = 0, maior = 0;
                for (int i=0; i<listas.size(); i++)
                    if (listas.size() > maior) {maior = listas.size(); indice = i;}
                listaprioritaria = indice;
            }
            listaaux = new Vector2[listas.get(listaprioritaria).size()];
        }
```

```
            //reordenar nomes que aparecem mais de uma vez, escolher a lista que contem mais nomes que aparecem mais
vezes nas outras listas
            ArrayList<Vector2> listafinal = new ArrayList<Vector2>();
            if (listas.size() == 1) listafinal = listas.get(0);
            else{

                if (listas.size() == 2) listafinal = listas.get(1); //melhorar depois
                else listafinal = listas.get(listaprioritaria);


                for (int i=0; i<listafinal.size(); i++){//analisar os elementos da lista com cada um dos outros
elementos da mesma, valorizar nomes de ocorrência maior
                    for (int j=0; j<listafinal.size(); j++){
                        if (listafinal.get(i).peso == listafinal.get(i).peso){
                            if (contemString(nomesordenados, listafinal.get(j).relacao) &&
!contemString(nomesordenados, listafinal.get(i).relacao)){
                                listaaux[i] = listafinal.get(j);
                                listaaux[j] = listafinal.get(i);
                                //listaaux.add(i, listafinal.get(j));
                                //listaaux.add(j, listafinal.get(i));
                            }
                        }
                    }
                }
                //repassar valores
                for (int i=0; i<listafinal.size(); i++){
                    if (listaaux[i] == null){
                        listaaux[i] = listafinal.get(i);
                    }
                }
                //for (int i=0; i<listaaux.length; i++) System.out.println("Tamanho: " + listaaux.length + "Nome : "
+listaaux[i].relacao);

            }

            if (listaaux.length != 0)
                if (listaaux[0] != null){
                    listafinal.clear();
                    for (int i=0; i<listaaux.length; i++){
                        listafinal.add(listaaux[i]);
                    }
                }

            return listafinal;
        }

        //Retornar lista com 1000 nomes
        //nome1, <lista de nomes <nome2, peso ou suporte>>
        public static String dirdest2 = "C:\\Users\\Erick\\Documents\\Plastino_Desafio\\dest2.txt";
        public static Hashtable<String, ArrayList<Vector2>> conjuntosfreq = new Hashtable<String,
ArrayList<Vector2>>();
        public static ArrayList<String> nomessuporte = new ArrayList<String>();
        public static ArrayList<Vector2> gerarLista(String nomeentrada, ArrayList<String> nomesexcluidos){
            if (conjuntosfreq.size() == 0){ //primeiro acesso, inicializar hashtable
                String conjuntos = dirdest2;
                try{
                    BufferedReader in = new BufferedReader(new InputStreamReader(new FileInputStream(conjuntos),
"UTF8"));
                    String line;
                    String[] split1 = new String[2], split2 = new String[2];
                    while((line = in.readLine()) != null) {
                        split1 = line.split(",");
                        split2 = split1[0].split("<->");
                        if (conjuntosfreq.containsKey(split2[0])){
                            conjuntosfreq.get(split2[0]).add(new Main().new Vector2(split2[1],
Integer.parseInt(split1[1])));
                        }else{
                            ArrayList<Vector2> nomesrelacionados = new ArrayList<Vector2>();
                            nomesrelacionados.add(new Main().new Vector2(split2[1], Integer.parseInt(split1[1])));
                            conjuntosfreq.put(split2[0], nomesrelacionados);
                        }
                        nomessuporte.add(split2[0]);
                        //System.out.println("Carregando conjuntos frequentes...");
                    }
                    in.close();

                }catch (IOException e){}
            }


            //parte do processamento da lista de 1000
            ArrayList<Vector2> saida = new ArrayList<Vector2>();
            String nomebase = nomeentrada;
            Integer contador = 0; boolean sair = false;
            String nomerelacionado;

            if (conjuntosfreq.containsKey(nomeentrada)){
                while((saida.size() < tamanholista) && !sair){
                    ArrayList<Vector2> aux = new ArrayList<Vector2>();

                        for (int i=0; i<conjuntosfreq.get(nomebase).size(); i++){
                            nomerelacionado = conjuntosfreq.get(nomebase).get(i).relacao;
                                if (!saida.contains(conjuntosfreq.get(nomebase).get(i))
                                        && (!nomerelacionado.equals(nomeentrada))
                                        && (!contemString(nomesexcluidos,nomerelacionado)))
                                            aux.add(conjuntosfreq.get(nomebase).get(i));
                        }
```

```
                    //for (int i=0; i<aux.size(); i++) System.out.print("--"+aux.get(i).relacao);
                        //melhorar \/
                    for (int i=0; i<aux.size(); i++){
                        if (saida.size() < tamanholista) {
                            boolean contemnome = false;
                            if (!saida.contains(aux.get(i))){
                                for (int j=0; j<saida.size() && !contemnome; j++){
                                    if (saida.get(j).relacao.equals(aux.get(i).relacao)) contemnome = true;
                                }
                                if (!contemnome)
                                    saida.add(aux.get(i));


                            }
                        }
                    }


                    if (contador >= saida.size()) sair = true;
                    else nomebase = saida.get(contador).relacao;
                    contador ++;

                }
            }

            //podar se estiver maior que o limite (pode acontecer)
            while (saida.size() > tamanholista) saida.remove(saida.size()-1);



            //se tiver menor que o limite da lista, adicionar mais nomes com maior suporte
            String key = "";
                for (int i=0; i<nomessuporte.size() && (saida.size() < tamanholista); i++){
                    key = nomessuporte.get(i);
                    boolean contem = false;
                    for (int j=0; j<saida.size() && !contem; j++){
                        if (saida.get(j).relacao.equals(key)) contem = true;
                    }
                    if (!contem) saida.add(new Main().new Vector2(key, conjuntosfreq.get(key).get(0).peso));
                }


            //for (int h=0; h<saida.size(); h++) System.out.println(nomeentrada + "hauiehduea" +
saida.get(h).relacao);
            return saida;
        }


        //Conjuntos frequentes manipulação
        static Hashtable<String, Integer> frequentes = new Hashtable<String, Integer>();
        public static void adicionarFreq(String nome1, String nome2, Integer peso){
            String aux = nome1 + "<->" + nome2,
                    aux2 = nome2 + "<->" + nome1;

            //System.out.println(nome1 + "<->" + nome2 + " | peso: " + peso);

            if (frequentes.containsKey(aux)){
                int pesovar = frequentes.get(aux) + peso;
                frequentes.remove(aux);
                frequentes.put(aux, pesovar);
                frequentes.remove(aux2);
                frequentes.put(aux2, pesovar);
            }else{
                frequentes.put(nome1 + "<->" + nome2, peso);
                frequentes.put(nome2 + "<->" + nome1, peso);
            }
        }



        public static void etapa4(){ //processar base de atividades enriquecendo-a com mais atributos
            String origem = "C:\\Users\\Érick\\Documents\\Plastino_Desafio\\nameling.trainData",
                    destino = "C:\\Users\\Érick\\Documents\\Plastino_Desafio\\dest4.txt";
            try {
                BufferedReader in = new BufferedReader(new InputStreamReader(new FileInputStream(origem), "UTF8"));
                BufferedWriter out = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(destino),"UTF8"));
                String line, saida;
                while((line = in.readLine()) != null) {
                    String[] separado = line.split("\\t");
                    saida = enriquecerNome(separado[2]);
                    out.write(separado[0] + "," + saida);
                    System.out.println(saida);
                    out.newLine();
                }

                out.close();
                in.close();

            } catch (IOException e) {}
        }


        public static String enriquecerNome(String nome){
            String saida;

            //tamanho
            saida = Integer.toString(nome.length());
            saida += ",";

            /*
            //masculino, feminino
```

```
            boolean masc = false, fem = false;

            if (!masc) masc = nome.endsWith("ling");
            if (!masc) masc = nome.endsWith("eoir");
            if (!masc) masc = nome.endsWith("inho");
            if (!masc) masc = nome.endsWith("óir");
            if (!masc) masc = nome.endsWith("eoir");
            if (!masc) masc = nome.endsWith("one");
            if (!masc) masc = nome.endsWith("shi");
            if (!masc) masc = nome.endsWith("ner");
            if (!masc) masc = nome.endsWith("ro");
            if (!masc) masc = nome.endsWith("ya");
            if (!masc) masc = nome.endsWith("in");
            if (!masc) masc = nome.endsWith("ed");
            if (!masc) masc = nome.endsWith("th");
            if (!masc) masc = nome.endsWith("lo");
            if (!masc) masc = nome.endsWith("t");
            if (!masc) masc = nome.endsWith("o");

            if (!masc){
                if (!fem) fem = nome.endsWith("schaft");
                if (!fem) fem = nome.endsWith("aison");
                if (!fem) fem = nome.endsWith("inha");
                if (!fem) fem = nome.endsWith("lann");
                if (!fem) fem = nome.endsWith("lann");
                if (!fem) fem = nome.endsWith("tion");
                if (!fem) fem = nome.endsWith("sion");
                if (!fem) fem = nome.endsWith("eit");
                if (!fem) fem = nome.endsWith("eid");
                if (!fem) fem = nome.endsWith("ung");
                if (!fem) fem = nome.endsWith("tät");
                if (!fem) fem = nome.endsWith("eog");
                if (!fem) fem = nome.endsWith("die");
                if (!fem) fem = nome.endsWith("eine");
                if (!fem) fem = nome.endsWith("ess");
                if (!fem) fem = nome.endsWith("ión");
                if (!fem) fem = nome.endsWith("óg");
                if (!fem) fem = nome.endsWith("ie");
                if (!fem) fem = nome.endsWith("ta");
                if (!fem) fem = nome.endsWith("la");
                if (!fem) fem = nome.endsWith("mi");
                if (!fem) fem = nome.endsWith("yo");
                if (!fem) fem = nome.endsWith("ko");
                if (!fem) fem = nome.endsWith("il");
                if (!fem) fem = nome.endsWith("ag");
                if (!fem) fem = nome.endsWith("li");
                if (!fem) fem = nome.endsWith("ah");
                if (!fem) fem = nome.endsWith("dd");
                if (!fem) fem = nome.endsWith("ne");
                if (!fem) fem = nome.endsWith("a");
                if (!fem) fem = nome.endsWith("y");
                if (!fem) fem = nome.endsWith("e");

            }
            if (masc) saida += "masculino";
            else{
                if (fem) saida += "feminino";
                else saida += "neutro";
            }*/
            saida += "neutro,";

            //quantidade de vogais
            int qtd = 0;
            if (nome.contains("a") || nome.contains("A")) qtd ++;
            if (nome.contains("e") || nome.contains("E")) qtd ++;
            if (nome.contains("i") || nome.contains("I")) qtd ++;
            if (nome.contains("o") || nome.contains("O")) qtd ++;
            if (nome.contains("u") || nome.contains("U")) qtd ++;
            saida += qtd + ",";


            //quantidade de consoantes
            saida += Integer.toString(nome.length() - qtd) + ",";


            //caractere inicial
            if (Character.toLowerCase(nome.charAt(0)) == ',') saida += "?";
            else saida += Character.toLowerCase(nome.charAt(0));
            saida += ",";


            //caractere final
            if (nome.charAt(nome.length() - 1) == ',') saida += "?";
            else saida += nome.charAt(nome.length() - 1);
            saida += ",";

            //radicais
            int radical = 0;/*
            if (nome.contains(" ")) radical = 11;
            if (nome.contains("ł")) radical = 9;
            if (nome.contains("ø")) radical = 12;
            if (nome.contains("ž") || nome.contains("ë")) radical = 12;
            if (nome.contains("ë")) radical = 13;
            if (nome.contains("tsu") || nome.contains("shi") || nome.contains("dzu")) radical = 1;
            if (nome.contains("nh") || nome.contains("ss") || nome.contains("lh") || nome.contains("ç")) radical = 2;
            if (nome.contains("ä") || nome.contains("ì")) radical = 3;
            if (nome.contains("á") || nome.contains("é") || nome.contains("í") || nome.contains("ó") ||
nome.contains("ú")) radical = 5;
            if (nome.contains("ê") || nome.contains("ô") || nome.contains("û") || nome.contains("â")) radical = 6;
            if (nome.contains("mm") || nome.contains("kl") || nome.contains("nn")) radical = 7;
            if (nome.contains("vd") || nome.contains("vd") || nome.contains("zh")) radical = 8;
            if (nome.contains("ɛ") || nome.contains("ɑ") || nome.contains("ɑ") || nome.contains("ʁ") ||
nome.contains("φ") || nome.contains("й")) radical = 14;
            if (nome.contains("ş") || nome.contains("î")) radical = 15;
```

```
            if (nome.contains("ü") || nome.contains("ö")) radical = 16;
            if (nome.contains("ph") || nome.contains("th") || nome.contains("mn")) radical = 17;
            if (nome.contains("-")) radical = 10;*/

            saida += radical;


            return saida;

        }



        public static void etapa5(){//gerar conjuntos frequentes com o enriquecimento dos nomes
            String origem = "C:\\Users\\Érick\\Documents\\Plastino_Desafio\\dest4.txt",
                   destino = "C:\\Users\\Érick\\Documents\\Plastino_Desafio\\dest5.txt";
            try{
                BufferedReader in = new BufferedReader(new InputStreamReader(new FileInputStream(origem), "UTF8"));
                BufferedWriter out = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(destino),"UTF8"));

                //ler usuários e suas respectivas tuplas de nomes enriquecidas
                Hashtable<Integer, ArrayList<String>> usuarios = new Hashtable<Integer, ArrayList<String>>();
                String line;
                while((line = in.readLine()) != null) {
                    String[] separado = line.split(",");
                    if (!usuarios.containsKey(Integer.parseInt(separado[0]))){
                        ArrayList<String> aux = new ArrayList<String>();
                        aux.add(line.substring(separado[0].length() + 1));
                        usuarios.put(Integer.parseInt(separado[0]), aux);
                    }else{
                        if (!usuarios.get(Integer.parseInt(separado[0])).contains(line.substring(separado[0].length()
+ 1))) //caso já tenha adicionado o nome
                            usuarios.get(Integer.parseInt(separado[0])).add(line.substring(separado[0].length() +
1));
                    }

                }

                in.close();

                 //separar os conjuntos frequentes
                Hashtable<String, Integer> regras = new Hashtable<String, Integer>(); //implicação, suporte

                ArrayList<String> tuplas;
                int key = 0, contagem = 0, tamanho = 7; //tamanho = quntd de atributos (0 a 6)
                Iterator it = usuarios.entrySet().iterator();
                while (it.hasNext()){
                      Map.Entry entry = (Map.Entry) it.next();
                      key = (Integer) entry.getKey();
                      tuplas = usuarios.get(key);

                      String[] splittedi = tuplas.get(0).split(","), splittedj;
                      String aux;

                      //separar os conjuntos frequentes
                      if (tuplas.size() > 1){
                          for (int i=0; i<tuplas.size(); i++){
                              for (int j=i; j<tuplas.size(); j++){
                                  splittedi = tuplas.get(i).split(",");
                                  splittedj = tuplas.get(j).split(",");
                                  for (int k=0; k<tamanho; k++){
                                      aux = "Atributo " + k + ": ";
                                      if (!regras.containsKey(aux + splittedi[k] + "<->" + splittedj[k])){
                                          regras.put(aux + splittedi[k] + "<->" + splittedj[k], 1);
                                          if (!splittedi[k].equals(splittedj[k])) //pra não contar duas vezes com
elementos iguais
                                              regras.put(aux + splittedj[k] + "<->" + splittedi[k], 1);
                                      }else{

                                          regras.put(aux + splittedi[k] + "<->" + splittedj[k], regras.get(aux +
splittedi[k] + "<->" + splittedj[k]) + 1);
                                          if (!splittedi[k].equals(splittedj[k]))  //pra não contar 2 vezes
                                              regras.put(aux + splittedj[k] + "<->" + splittedi[k], regras.get(aux +
splittedj[k] + "<->" + splittedi[k]) + 1);
                                      }
                                  }
                              }
                          }
                      }

                      contagem ++;
                      System.out.println("Porcentagem de separação dos conjuntos frequentes: " +
contagem*100/usuarios.size() + "%");
                }


                //ordenar os conjuntos, complexidade muito ruim (entretanto foi mais fácil fazer dessa forma)
                ArrayList<String> saida = new ArrayList<String>();
                Integer suporte  = 0, maior = -1, tamanhototal = regras.size();
                String indice = "", key2 = "";
                while (regras.size() > 0){
                    maior = -1;
                    it = regras.entrySet().iterator();
                    while (it.hasNext()){
                          Map.Entry entry = (Map.Entry) it.next();
                          key2 = (String) entry.getKey();
                          suporte = regras.get(key2);

                          if (suporte > maior){
                              maior = suporte;
                              indice = key2;
```

```
                        }
                    }
                    saida.add(indice + " <sup: " + regras.get(indice) + ">");
                    regras.remove(indice);
                    System.out.println("Porcentagem de ordenação: " + (100 - regras.size()*100/tamanhototal) + "%");
                }


                //escrever no arquivo
                for (int i=0; i<saida.size(); i++){
                    out.write(saida.get(i));
                    System.out.println(saida.get(i));
                    out.newLine();
                }


                out.close();
            }catch (IOException e){}
        }


        public static void etapa5_complemento(){
            String freqorigem = "C:\\Users\\Érick\\Documents\\Plastino_Desafio\\dest5.txt",
                    destino = "C:\\Users\\Érick\\Documents\\Plastino_Desafio\\dest5_c.txt";
            try{
                BufferedReader freq = new BufferedReader(new InputStreamReader(new FileInputStream(freqorigem),
"UTF8"));
                BufferedWriter out = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(destino),"UTF8"));

                Hashtable<String, ArrayList<Integer>> sup = new Hashtable<String, ArrayList<Integer>>();
                Hashtable<String, ArrayList<String>> regra = new Hashtable<String, ArrayList<String>>();
                String line; String[] separado;
                ArrayList<Integer> aux; ArrayList<String> aux2;
                while((line = freq.readLine()) != null) {
                    separado = line.split("<->");
                    if (!sup.containsKey(separado[0])){
                        aux = new ArrayList<Integer>();
                        aux.add(Integer.parseInt(separado[1].split(" ")[2].substring(0, separado[1].split("
")[2].length() - 1)));
                        aux2 = new ArrayList<String>();
                        aux2.add(line);
                        sup.put(separado[0], aux);
                        regra.put(separado[0], aux2);
                    }else{
                        sup.get(separado[0]).add(Integer.parseInt(separado[1].split(" ")[2].substring(0,
separado[1].split(" ")[2].length() - 1)));
                        regra.get(separado[0]).add(line);
                    }
                }
                freq.close();

                ArrayList<String> saida = new ArrayList<String>(), saidaord = new ArrayList<String>();
                Iterator it = sup.entrySet().iterator();
                Map.Entry entry;
                String chave;
                while (it.hasNext()){
                      entry = (Map.Entry) it.next();
                      chave = (String) entry.getKey();
                      boolean sair = false;
                      for (int i=0; i<sup.get(chave).size() - 1 && !sair; i++){
                          if ((sup.get(chave).get(i) < sup.get(chave).get(i + 1) * 1.25f) && sup.get(chave).get(i) >
1000){
                              saida.add(regra.get(chave).get(i));
                          }else{
                              sair = true;
                          }
                      }
                }

                int maior = -1, indice = 0, suporte = 0;
                while (saida.size() > 0){
                    maior = -1; indice = 0;
                    for (int i=0; i<saida.size(); i++){
                        suporte = Integer.parseInt(saida.get(i).split(" ")[4].substring(0, saida.get(i).split("
")[4].length() - 1));
                        if (suporte > maior ){
                            maior = suporte;
                            indice = i;
                        }
                    }
                    saidaord.add(saida.get(indice));
                    saida.remove(indice);
                }

                for (int i=0; i<saidaord.size(); i++){
                    out.write(saidaord.get(i));
                    out.newLine();
                }
                out.close();

            }catch(IOException e){}
        }


        //construindo as regras de classificação do usuário
        static ArrayList<String> regrasuser, regrasuserord, tuplas;//atributo, peso, regra
        static ArrayList<Byte> ativportupla;
        static Hashtable<String, Integer> multvalor;
        static Hashtable<String, Long> nomepeso;
        static ArrayList<String> listafinal;
```

```
        static ArrayList<String[]> espacoseparado, espacoseparado2;
        //ArrayList<Long> listafinalpeso;
        static int numusuario = 0;
        static int contagem = 0; static long maior = -1; static String nomevez, chave, auxstr, prefixo, chavefinal;
        static String[] separado2;
        static Iterator<Entry<Integer, ArrayList<String>>> it;
        static Iterator<Entry<String, Long>> it2;
        static Map.Entry<Integer, ArrayList<String>> entry;
        static Map.Entry<String, Long> entry2;
        public static void etapa6(){ //aplicar conjuntos frequentes de regras que se adaptam ao usuário em todos os
nomes passíveis à seleção
            System.out.println("1 ou 2?");
            Scanner teclado = new Scanner(System.in);
            int id = Integer.parseInt(teclado.nextLine());

            String origem = "C:\\Users\\Érick\\Documents\\Plastino_Desafio\\nameling.trainData",
                    destino = "C:\\Users\\Érick\\Documents\\Plastino_Desafio\\dest6_" + id + ".txt",
                    freqorigem = "C:\\Users\\Érick\\Documents\\Plastino_Desafio\\dest5_c.txt",
                    nomes = "C:\\Users\\Érick\\Documents\\Plastino_Desafio\\namelist.txt";
            try{
                BufferedReader in = new BufferedReader(new InputStreamReader(new FileInputStream(origem), "UTF8"));
                BufferedReader freq = new BufferedReader(new InputStreamReader(new FileInputStream(freqorigem),
"UTF8"));
                BufferedWriter out = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(destino),"UTF8"));
                BufferedReader namelist = new BufferedReader(new InputStreamReader(new FileInputStream(nomes),
"UTF16"));


                //carregando usuários
                Hashtable<Integer, ArrayList<String>> usuarios = new Hashtable<Integer, ArrayList<String>>();
                Hashtable<Integer, ArrayList<Byte>> ativporuser = new Hashtable<Integer, ArrayList<Byte>>();
                String line, nomeadd; byte pesoatividade;
                while((line = in.readLine()) != null) {
                    pesoatividade = 1;
                    String[] separado = line.split("\t");
                    if (separado[1].equals("ADD_FAVORITE")) pesoatividade = 3;
                    //separado[0] = separado[0].substring(1); //bug fix
                    nomeadd = line.substring(separado[0].length() + 1);
                    if (!usuarios.containsKey(Integer.parseInt(separado[0]))){
                        ArrayList<String> aux = new ArrayList<String>();
                        aux.add(nomeadd);
                        usuarios.put(Integer.parseInt(separado[0]), aux);
                        ArrayList<Byte> aux2 = new ArrayList<Byte>();
                        aux2.add(pesoatividade);
                        ativporuser.put(Integer.parseInt(separado[0]), aux2);
                    }else{
                        if (!usuarios.get(Integer.parseInt(separado[0])).contains(nomeadd)){
                            usuarios.get(Integer.parseInt(separado[0])).add(nomeadd);
                            ativporuser.get(Integer.parseInt(separado[0])).add(pesoatividade);
                        }
                    }

                }
                in.close();



                //carregando conjuntos frequentes, ou regras frequentes
                Hashtable<Integer, ArrayList<String>> regrasfreq = new Hashtable<Integer, ArrayList<String>>();
//numero do atributo, regras associadas
                Hashtable<String, Integer> supregrasfreq = new Hashtable<String, Integer>();
                ArrayList<String> regrasfreqtuplas = new ArrayList<String>();
                byte indinicial = (byte) "Atributo ".length();
                Integer indice, aux;
                String[] separado;
                String[] nomeenrq;
                while((line = freq.readLine()) != null) {
                    line = line.substring(indinicial);
                    separado = line.split(" ");

                    indice = Integer.parseInt(line.substring(0, 1));

                    if (!regrasfreq.containsKey(indice)) regrasfreq.put(indice, new ArrayList<String>());

                    if (!regrasfreq.get(indice).contains(separado[1])){
                        regrasfreq.get(indice).add(separado[1]);
                        aux = separado.length - 1;
                        supregrasfreq.put(separado[1], Integer.parseInt(separado[aux].substring(0,
separado[aux].length() - 1)));
                    }

                    regrasfreqtuplas.add(line);
                }
                freq.close();


                //carregar lista de nomes possíveis
                ArrayList<String> nomespossiveis = new ArrayList<String>();
                while((line = namelist.readLine()) != null) {
                    nomespossiveis.add(line);
                }
                namelist.close();

                //separar valor da direita das implicações para não ficar realizando a conta toda hora
                ArrayList<String> impdireita = new ArrayList<String>(),
                semsuporte = new ArrayList<String>();
                for (int i=0; i<regrasfreqtuplas.size(); i++){
                    impdireita.add(regrasfreqtuplas.get(i).split(" ")[1].split("<->")[1]);
                    semsuporte.add("Atributo " + regrasfreqtuplas.get(i).split(" <")[0]);
                }
                //separar o suporte das regras também /\
```

```
                //usuarios já computados
                ArrayList<Integer> usercomp = new ArrayList<Integer>();
                in = new BufferedReader(new InputStreamReader(new FileInputStream(".\\teste\\dest6_" + id + ".txt"),
"UTF8"));
                while ((line = in.readLine()) != null){
                    line = line.substring(0, 40);
                    usercomp.add(Integer.parseInt(line.split("\\t")[0]));
                }
                in.close();


                it = usuarios.entrySet().iterator();
                while (it.hasNext()){//pra todos os usuarios
                      contagem ++;
                      entry = (Map.Entry<Integer, ArrayList<String>>) it.next();
                      numusuario = (Integer) entry.getKey();
                      if ((id % 2 == contagem % 2) && !usercomp.contains(numusuario)){
                          tuplas = usuarios.get(numusuario);
                          ativportupla = ativporuser.get(numusuario);
                          multvalor = new Hashtable<String, Integer>();
                          regrasuser = new ArrayList<String>();



                          //construir regras por user
                          for (int i=0; i<tuplas.size(); i++){
                              separado = tuplas.get(i).split("\t");
                              nomevez = enriquecerNome(separado[1]);
                              separado = nomevez.split(",");
                              for (int k=0; k<separado.length; k++){//rodar os atributos
                                  int atratual = 0; auxstr = "";
                                  prefixo = "Atributo ";
                                  for (int f=0; f<regrasfreqtuplas.size(); f++){
                                      atratual = Integer.parseInt(regrasfreqtuplas.get(f).substring(0, 1));
                                      if (atratual == k){
                                          if(impdireita.get(f).equals(separado[k])){
                                              auxstr = semsuporte.get(f);
                                              if (!regrasuser.contains(auxstr)) regrasuser.add(auxstr);
                                              if (!multvalor.contains(auxstr)){
                                                  multvalor.put(auxstr, 1 * ativportupla.get(i));
                                              }else{
                                                  multvalor.put(auxstr, (multvalor.get(auxstr) + 1)*
ativportupla.get(i));
                                              }
                                          }
                                      }
                                  }

                              }
                          }

                          regrasuserord = regrasuser;

                          //podar a quantidade de regras
                          for (int i=35; i<regrasuserord.size(); i++){ //valor inicial de i indica quantas regras
serão retiradas, valor empírico
                              regrasuserord.remove(i);
                          }


                          //aplicar regras geradas aos nomes possíveis
                          espacoseparado = new ArrayList<String[]>(); espacoseparado2 = new ArrayList<String[]>();
                          for (int i=0; i<regrasuserord.size(); i++){
                              separado = regrasuserord.get(i).split(" ");
                              espacoseparado.add(separado);
                              espacoseparado2.add(separado[2].split("<->"));
                          }
                          nomepeso = new Hashtable<String, Long>();
                          long pontos, porcentagem = 0;
                          int numatrib; nomeenrq = null;
                          for (int j=0; j<nomespossiveis.size(); j++){
                              chavefinal = nomespossiveis.get(j);
                              pontos = 0; porcentagem = 0;
                              nomeenrq = enriquecerNome(nomespossiveis.get(j)).split(",");

                              porcentagem /= tuplas.size();
                              pontos += porcentagem;
                              for (int i=0; i<regrasuserord.size(); i++){
                                  separado = espacoseparado.get(i);
                                  numatrib = Integer.parseInt(separado[1].substring(0, 1));
                                  separado2 = espacoseparado2.get(i);
                                  if (nomeenrq[numatrib].equals(separado2[1])){
                                      pontos +=
Math.ceil(multvalor.get(regrasuserord.get(i))*supregrasfreq.get(separado[2])/87000f);
                                  }
                              }
                              nomepeso.put(nomespossiveis.get(j), pontos);
                          }

                          //ordenar nomepeso na lista final de 1000
                          listafinal = new ArrayList<String>();
                          chave = ""; chavefinal = ""; entry2 = null;
                          while (listafinal.size() < tamanholista){
                              maior = -1;
                              it2 = nomepeso.entrySet().iterator();
                              while (it2.hasNext()){
                                  entry2 = (Map.Entry<String, Long>) it2.next();
                                  chave = (String) entry2.getKey();
                                  if (nomepeso.get(chave) > maior){
```

```
                                    maior = nomepeso.get(chave);
                                    chavefinal = chave;
                                }
                            }
                            listafinal.add(chavefinal);
                            nomepeso.remove(chavefinal);
                        }


                        //escrever lista final
                        out.write(Integer.toString(numusuario) + "\t");
                        for (int i=0; i<listafinal.size(); i++){
                            out.write(listafinal.get(i) + "\t");
                        }
                        out.newLine();

                        if (contagem%20 == 0) System.gc();
                        System.out.println("Porcentagem de escrita por usuário (" + contagem + "): " +
contagem*100/usuarios.size() + "%");
                    }
                }//ainda dentro de um usuário

                out.close();

            }catch (IOException e){}
        }


        public static void etapa6_ordenar(){
            try{
                BufferedReader in;
                BufferedWriter out = new BufferedWriter(new OutputStreamWriter(new
FileOutputStream(".\\final.txt"),"UTF8"));
                BufferedWriter out2 = new BufferedWriter(new OutputStreamWriter(new
FileOutputStream(".\\final_ord.txt"),"UTF8"));
                Hashtable<Integer, String> usuarios = new Hashtable<Integer, String>();
                String line; String primeirap;
                for (int i=1; i<5; i++){
                    in = new BufferedReader(new InputStreamReader(new FileInputStream(".\\" + i + ".txt"), "UTF8"));
                    while ((line = in.readLine()) != null){
                        usuarios.put(Integer.parseInt(line.substring(0, 30).split("\\t")[0]), line);
                        out.write(line);
                        out.newLine();
                    }
                    in.close();
                }
                out.close();

                ArrayList<String> ordenado = new ArrayList<String>();
                Map.Entry<Integer, String> entry;
                int menor = 999999999;
                int tamini = usuarios.size();
                while (usuarios.size() > 0){
                    menor = 999999999;
                    Iterator<Entry<Integer, String>> it = usuarios.entrySet().iterator();
                    while (it.hasNext()){
                          entry = (Entry<Integer, String>) it.next();
                          numusuario = (Integer) entry.getKey();
                          if (numusuario < menor){
                              menor = numusuario;
                          }
                    }
                    ordenado.add(usuarios.get(menor));
                    usuarios.remove(menor);
                    System.out.println("Porcentagem de ordenação: " + (100 - usuarios.size()*100/tamini));
                }


                for (int i=0; i<ordenado.size(); i++){
                    out2.write(ordenado.get(i));
                    out2.newLine();
                }

                out2.close();


            }catch (IOException e){

            }
        }



        //Função aux2
        public static int contemSV(ArrayList<Vector2> array, String str){
            for (int i=0; i<array.size(); i++){
                if (array.get(i).relacao.equals(str)) return i;
            }
            return -1;
        }

        //Função auxiliar
        public static boolean isNumeric(String s) {
            return s.matches("\\d+");
        }

        //Outra função aux
        public static boolean contemString(ArrayList<String> a, String nome){
            for (int i=0; i< a.size(); i++){
```

```
                if (a.get(i).equals(nome)) return true;
            }
            return false;
        }
    }


    //classe auxiliar para armazenar uma string e um inteiro
    public class Vector2 {
        public String relacao;
        public int peso;

        Vector2(String relacao, int peso){
            this.relacao = relacao;
            this.peso = peso;
        }
    }


    /** @return an array of adjacent letter pairs contained in the input string */
       private static String[] letterPairs(String str) {
           int numPairs = str.length()-1;
           String[] pairs = new String[numPairs];
           for (int i=0; i<numPairs; i++) {
               pairs[i] = str.substring(i,i+2);
           }
           return pairs;
       }
    /** @return an ArrayList of 2-character Strings. */
       private static ArrayList<String> wordLetterPairs(String str) {
           ArrayList<String> allPairs = new ArrayList<String>();
           // Tokenize the string and put the tokens/words into an array
           String[] words = str.split("\\s");
           // For each word
           for (int w=0; w < words.length; w++) {
               // Find the pairs of characters
               String[] pairsInWord = letterPairs(words[w]);
               for (int p=0; p < pairsInWord.length; p++) {
                   allPairs.add(pairsInWord[p]);
               }
           }
           return allPairs;
       }
    /** @return lexical similarity value in the range [0,1] */
       public static double compareStrings(String str1, String str2) {
           ArrayList<String> pairs1 = wordLetterPairs(str1.toUpperCase());
           ArrayList<String> pairs2 = wordLetterPairs(str2.toUpperCase());
           int intersection = 0;
           int union = pairs1.size() + pairs2.size();
           for (int i=0; i<pairs1.size(); i++) {
               String pair1=pairs1.get(i);
               for(int j=0; j<pairs2.size(); j++) {
                   String pair2=pairs2.get(j);
                   if (pair1.equals(pair2)) {
                       intersection++;
                       pairs2.remove(j);
                       break;
                   }
               }
           }
           return (2.0*intersection)/union;
       }
}
```