# US307 – Análise de Complexidade

importCabazFromCSV(String filePath,MapGraph<User, Integer> graph) throws IOException {  
 try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {  
 String line;  
 String id;  
 reader.readLine();  
 while ((line = reader.readLine()) != null) {  
  
 String[] values = line.split(",");  
 id = values[0].replace("\"", "");  
 int dia = Integer.*parseInt*(values[1].replace("\"", ""));  
 int numProdutos = values.length - 2;  
 Map<Product, Float> produtos = new HashMap<>();  
 for (int i = 0; i < numProdutos; i++) {  
 if (Float.*parseFloat*(values[i + 2].replace("\"", "")) != 0) {  
 produtos.put(new Product("Prod" + (i+1)), Float.*parseFloat*(values[i + 2].replace("\"", "")));  
 }  
 }  
 String finalId1 = id;  
 User c = graph.vertex(user -> user.getUserID().equals(finalId1));  
 if (!produtos.isEmpty()) {  
 Cabaz cabaz = new Cabaz(c, dia, produtos);  
 if (c != null) {  
 switch (id.charAt(0)) {  
 case 'C':  
 if (cabazC.containsKey(dia)) {  
 cabazC.get(dia).put((Client) c, cabaz);  
 } else {  
 Map<Client, Cabaz> cabazes = new HashMap<>();  
 cabazes.put((Client) c, cabaz);  
 cabazC.put(dia, cabazes);  
 }  
 break;  
 case 'E':  
 if (cabazE.containsKey(dia)) {  
 cabazE.get(dia).put((Firm) c, cabaz);  
 } else {  
 Map<Firm, Cabaz> cabazes = new HashMap<>();  
 cabazes.put((Firm) c, cabaz);  
 cabazE.put(dia, cabazes);  
 }  
 break;  
 case 'P':  
 if (cabazP.containsKey(dia)) {  
 cabazP.get(dia).put((Producer) c, cabaz);  
 } else {  
 Map<Producer, Cabaz> cabazes = new HashMap<>();  
 cabazes.put((Producer) c, cabaz);  
 cabazP.put(dia, cabazes);  
 }  
 break;  
 default:  
 break;  
 }  
 }  
 }  
 }  
 }

|  |  |  |
| --- | --- | --- |
| Linhas | Código | Complexidade |
| 1-6 | public void importCabazFromCSV(String filePath,MapGraph<User, Integer> graph) throws IOException {  try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {  String line;  String id;  reader.readLine(); | 1 |
| 7 | while ((line = reader.readLine()) != null) | Sendo l, o número de linhas no documento csv escolhido; |
| 8-12 | String[] values = line.split(",");  id = values[0].replace("\"", "");  int dia = Integer.parseInt(values[1].replace("\"", ""));  int numProdutos = values.length - 2;  float[] produtos = new float[numProdutos]; | Uma vez que este código são apenas simples atribuições a complexidade não aumenta; |
| 13-15 | if (Float.parseFloat(values[i + 2].replace("\"", "")) != 0) {  produtos.put(new Product("Prod" + (i+1)), Float.parseFloat(values[i + 2].replace("\"", ""))); | A complexidade passar a ser l^2, de facto o ciclo for corre “numProdutos” vezes sendo este o mesmo número de linhas presentes no ficheiro “csv”; |
| 16 | String finalId1 = id; | Apenas uma simples atribuição; |
| 17 | User c = graph.vertex(user -> user.getUserID().equals(finalId1)); | O algoritmo “vertex” apresenta uma complexidade de V, onde V é o número de vértices no grafo. Como este se encontra dentro do “while” da linha 7, a complexidade passa a ser l(l+V); |
| 18-54 | Cabaz cabaz = new Cabaz(c, dia, produtos);  if (!produtos.isEmpty()) {  if (c != null) {  switch (id.charAt(0)) {  case 'C':  if (cabazC.containsKey(dia)) { cabazC.get(dia).put((Client) c, cabaz);  } else {  Map<Client, Cabaz> cabazes = new HashMap<>(); cabazes.put((Client) c, cabaz); cabazC.put(dia, cabazes);  }  break;  case 'E':  if (cabazE.containsKey(dia)) { cabazE.get(dia).put((Firm) c, cabaz);  } else {  Map<Firm, Cabaz> cabazes = new HashMap<>(); cabazes.put((Firm) c, cabaz); cabazE.put(dia, cabazes);  }  break;  case 'P':  if (cabazP.containsKey(dia)) { cabazP.get(dia).put((Producer) c, cabaz);  } else {  Map<Producer, Cabaz> cabazes = new HashMap<>(); cabazes.put((Producer) c, cabaz);  cabazP.put(dia, cabazes);  }  break;  default:  break;}}}}} | No restante código apenas são realizadas simples atribuições, logo não alterando o tempo de complexidade |