Universidade do Minho

Projecto Java de Laboratório de Informática
Relatório

Discentes:
Axel Ferreira - a53064
João Rua - a41841

Docentes:
F. Mário Martins
João Miguel Fernandes
João Luís Sobral

June 18, 2013

Grupo

(a) nome : Axel Ferreira
número : 53064
mail : axelferreira@me.com

(b) nome : João Rua
número : 41841
mail : joaorua@gmail.com

Contents

Abstract

Foi desenvolvido um programa no âmbito da U.C. de Laboratório de Informática III , capaz de utilizar o conteúdo processado pelo pelo anterior programa desenvolvido em C, também nesta UC, capaz de ler um ficheiro contendo um conjunto de autores e respetivos co-autores, e responder a alguns queries interativos sobre estes dados. Estes dados de autorias, e co-autorias são retirados do website DBLP.

1 Introdução

No âmbito da unidade curricular Laboratórios de Informática III foi proposta a realização de um projeto que dá continuidade ao projeto anteriormente desenvolvido em C nesta mesma UC. Este novo projeto conta com essencialmente duas partes. A primeira diz respeito á leitura de dados de memória secundária e população de estruturas de dados em memória central, gravação destas estruturas de dados em memória persistente em modo binário, bem como a criação de alguns queries de forma a permitir uma consulta interativa aos dados. Desenvolveram-se ainda alguns métodos que permitem consultas sobre as estruturas de dados. A Segunda parte prevê o teste de performance do código e respetivas estruturas de dados criados na 1ª parte, relativamente a estruturas de dados alternativas. De forma a facilitar esta segunda parte, o grupo teve o cuidado de criar uma interface cada vez que foi utilizada uma estrutura de dados do Java.Collections, de forma a permitir alterar as estruturas utilizadas alterando apenas, e se necessário, esta interface.

1.1 Estrutura do Relatório

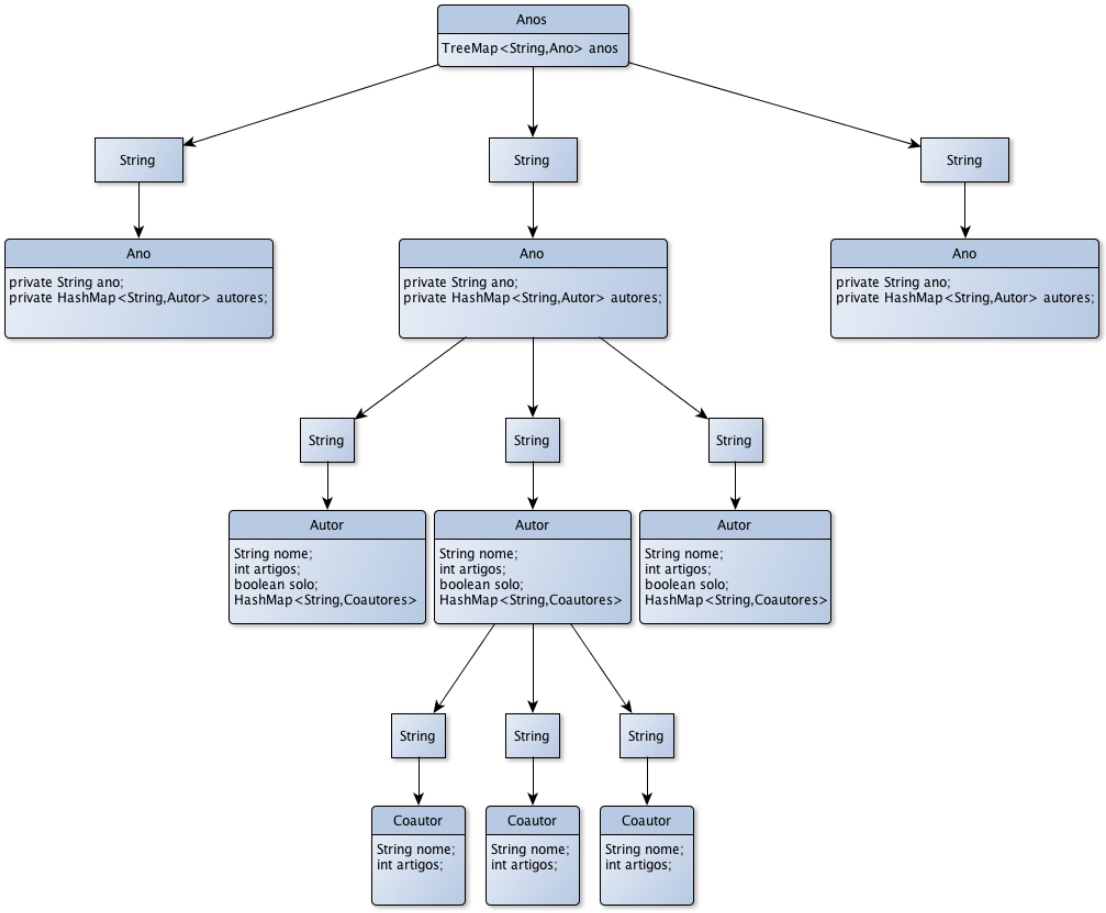
Este relatório inicia-se com uma capa, incluindo o título do projeto, a data, a identificação dos autores e da equipa docente que acompanhou o projeto. Segue-se o Índice, o Abstract que resume o projeto, a Introdução ao mesmo (onde se

explicita o objetivo a atingir) e a Estrutura do Relatório. No desenvolvimento são explicadas as classes criadas e a razão das estruturas de dados escolhidas, bem como as consultas estatísticas e interativas. Por fim apresentam-se a Conclusão.

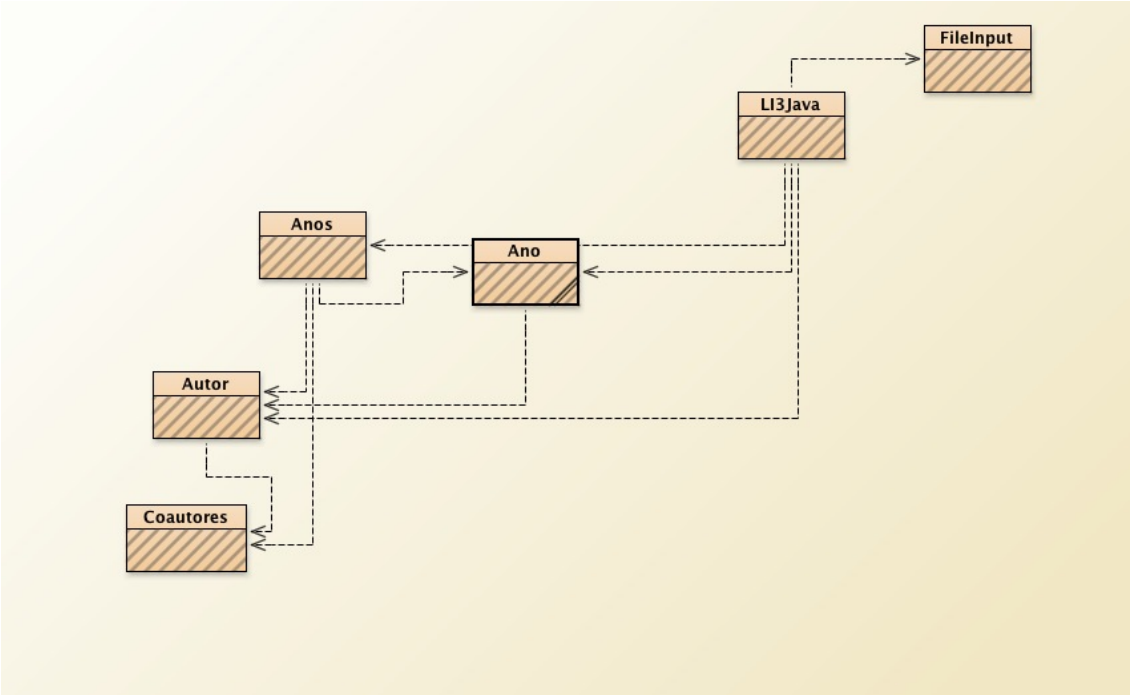
2 Classes e Estruturas de Dados

2.1 Classes Criadas

Na criação deste programa foram desenvolvidas algumas classes que são explicadas abaixo. Neste diagrama encontra-se explicada a estrutura de dados utilizada no programa.



O diagrama abaixo mostra a hierarquia de classes.



- Anos - Esta classe armazena toda a estrutura de dados. Contém um TreeMap em que são inseridos todos os anos. Cada ano é inserido, utilizando como chave a String contendo a numeração do ano. A escolha do TreeMap deveu-se ao facto de manter a ordem do ano. Esta ordem facilita a impressão e travessia ordenada necessária para algumas queries.

```
private TreeMap<String,Ano> anos;
```

- Ano - Classe criada para guardar o conteúdo de cada Ano, contém uma String com o nome do ano e um HashMap com os Autores. Cada autor é inserido utilizando como chave a String com o nome do mesmo. A utilização do HashMap deve-se ao facto de não haver vantagem associada á ordem de armazenamento.

```
private String ano;
private HashMap<String,Autor> autores;
```

- Autor - Classe criada para guardar o nome do Autor, numero de publicações, e rede de co-autores. Esta última é armazenada num HashMap de co-autores. Cada co-autor é inserido utilizando como chave a String com o nome do mesmo. A utilização do HashMap deve-se ao facto de não haver vantagem associada á ordem de armazenamento.

```
private String nome;
private int artigos;
private HashMap<String,Coautores> coautores;
```

- Coautor - Classe criada para guardar o conteúdo de cada co-autor, resumindo-se ao nome e numero de artigos publicados em comum com o respetivo autor.

```
private String nome;
private int artigos;
```

- FileInput - Esta classe faz todo o parsing e leitura dos dados de ficheiros com que posteriormente as estruturas de dados são povoadas. Contém apenas como variável de classe o nome do ficheiro que deve ler sempre. Contém ainda um método de classe que devolve o nome do ficheiro. Bem como dois métodos que devolvem o conteúdo dos ficheiros.

  ```
  public static final String ficheiro ="publicx.txt";
  ```

3 Consultas Estatísticas

3.1 Dados do último ficheiro lido

Quando o ficheiro é lido, é apresentado no ecrã o nome do ficheiro, seguido do número total de artigos, e número total de nomes lidos, número total de nomes distintos bem como o intervalo fechado de anos em que os artigos foram lidos. É ainda apresentada alguma informação respeitante aos dados atuais da estrutura de dados, nomeadamente nº total de autores, nº total de artigos, de um só autor, e nº de autores que apenas publicaram a solo. Finalmente é ainda apresentada toda a sequência ordenada de anos seguido do respetivo número de publicações.

3.2 Consultas Interativas

Nas queries interativas foi implementado um sistema de menus por prompt em que são mostradas as opções disponíveis ao utilizador do software que posteriormente seleciona a opção pretendida. Aqui é pedido um intervalo de anos para responder as seguintes queries.

- Top # de nº de publicações
  Para cada ano do intervalo, os autores são inseridos como Key num HashMap e o nº de publicações como Value. Se o autor já existir é apenas somado o nº de publicações deste ano as existentes, caso contrário o autor é inserido. No fim é impressa uma lista dos # elementos com mais publicações decrescentemente organizada, e em caso de empate é impresso por ordem alfabética.

- Top # de co-autores
  A classe Ano tem um método topCoArtigos() que devolve um HashMap com todas as combinações possíveis não repetidas autor-coautor (Key) e respetivo número de publicações(Value). Posteriormente, para cada ano do intervalo dado é percorrido o topCoAutores() e são inseridos numa estrutura final todos os autores-coautores, e em caso de já existirem, é apenas somado o nº de publicações. Finalmente é criado um TreeMap com um comparator que ordena por nº de Artigos e em caso de empate alfabeticamente toda a estrutura.

- Listagem de co-autores comuns a uma lista de autores

Dada uma lista de autores, é utilizado o método listaCoautoresDeAnoPorIntervalo() que recebe o ano inicial, final e o nome do autor, devolve a lista de co-autores deste autor, isto é repetido para todos os autores. A primeira lista é inserida numa lista final (TreeSet), e para cada uma das listas seguintes são removidos os nomes da lista final que não constam nas listas seguintes. No fim, a lista final é impresso alfabeticamente.

- Listagem de autores que publicam em todos os anos do intervalo
  Inicialmente é usado o método listaAutores() que devolve a lista de todos os autores que publicaram nesse ano. Estes autores são inseridos numa estrutura final TreeSet. Para cada um dos anos seguintes é executado o método listaAutores(), e são removidos os autores da estrutura final, caso não se encontrem na nova lista.

3.3 Consultas Globais

Foram criadas ainda duas queries que efetuam travessias transversais á estrutura de dados.

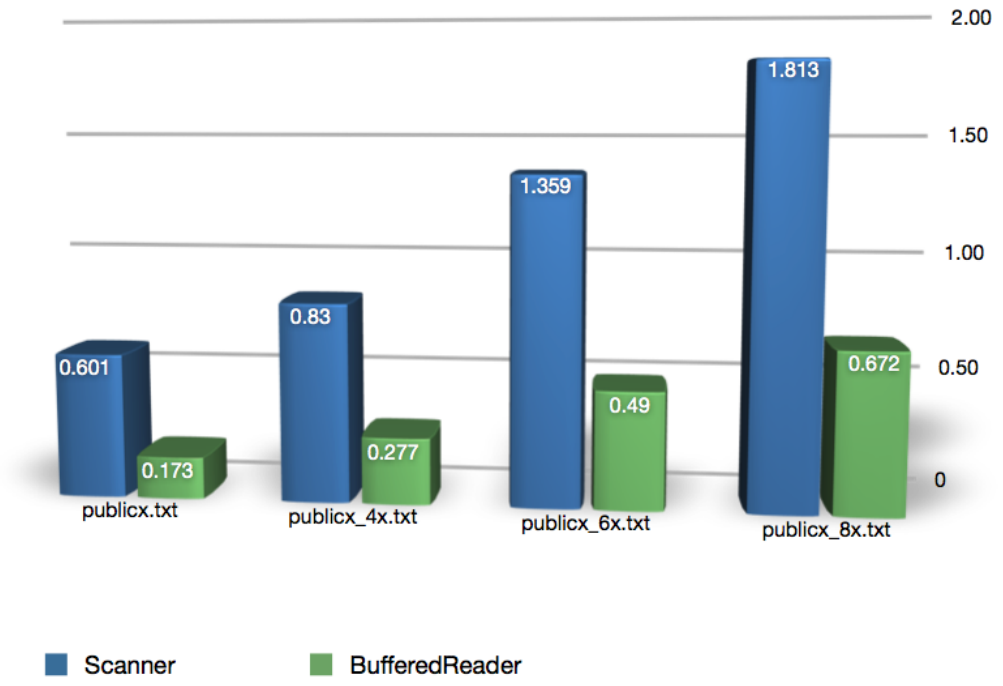
- # de linhas duplicadas
  Esta querie foi calculada com recurso a uma nova leitura do ficheiro publicx.txt, em que as linhas foram inseridas numa estrutura Set. Posteriormente, a diferença entre este Set e a estrutura que tem o nº de linhas inicialmente lidas responde á querie.

- tabela de autores/coautores com menos de # co-autores
  Para responder a esta querie é criado um TreeMap em que a Key é o nome de cada autor, e o Value corresponde a um TreeSet com o nome dos respetivos coautores. Posteriormente é pedido o número máximo de co-autores a imprimir. O TreeMap de autores e TreeSet de co-autores foram escolhidos para permitir a impressão ordenada alfabeticamente.

4 Medidas de Performance

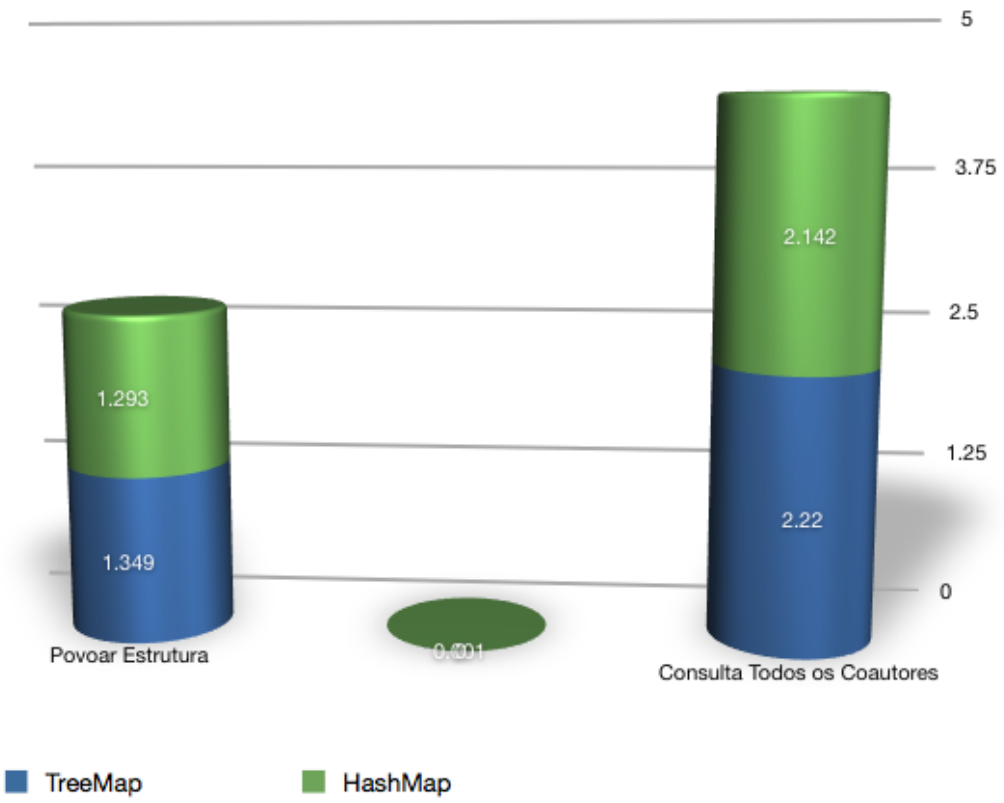
4.1 Tempos de Leitura

Foram realizados testes de tempos de leitura, tendo como base o ficheiro publicx.txt múltiplos do conteúdo deste ficheiro (4x, 6x, 8x), com as classes Scanner e BufferedReader tendo sido obtidos os tempos do gráfico abaixo, com o tempo medido em segundos. Estes tempos são referentes apenas á leitura, e não incluem o parsing. Foi utilizada a classe System.nanoTime para estas medições. Estes resultados espelham a diferença do buffer de 1KB utilizado pela classe Scanner vs o buffer de 8KB utilizado pela classe BufferedReader.



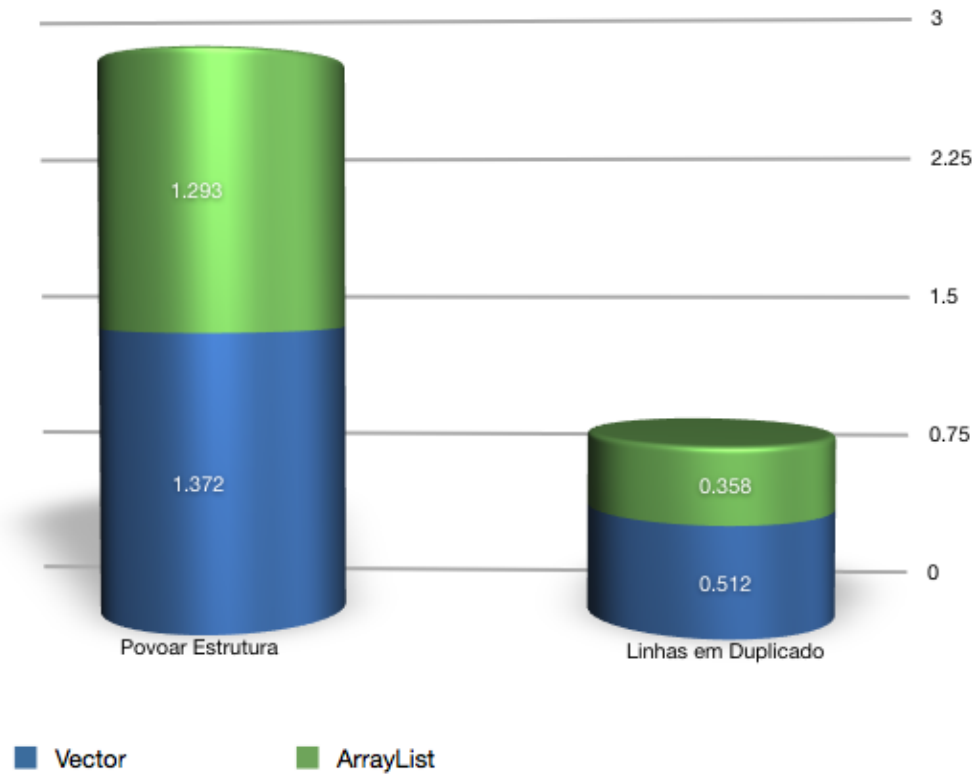
4.2 Performance das Estruturas

Foram realizados alguns testes de performance ás estruturas de dados escolhidas, e posteriormente foram substituídas as estruturas de dados e realizados novamente os testes. Os resultados são apresentados abaixo na forma de gráficos. Os teste foram realizados sem printes para o ecrã. Este gráfico compara a diferença de tempo da povoação, e execução de duas das travessias mais pesadas, comparando a utilização de TreeMap vs HashMap.



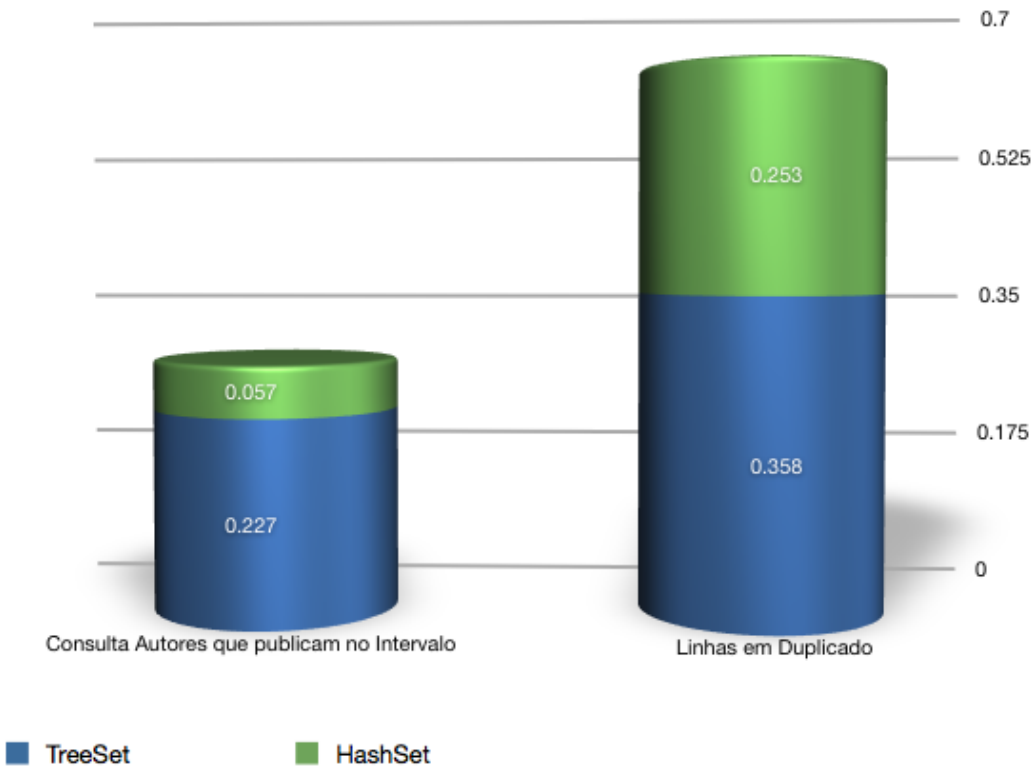
	TreeMap	HashMap
Povoar Estrutura	1.348965	1.293414
Consulta Total de Artigos por Ano	5.41E-04	1.29E-04
Consulta Todos os Coautores	2.219872	2.142017

Este gráfico salienta a diferença de tempo de execução entre a utilização do Vector e do ArrayList na povuação das estruturas, e posteriormente na querie de duplicação de linhas.



	Vector	ArrayList
Povoar Estrutura	1.371829	1.293414
Linhas em Duplicado	0.511943	0.358043

Finalmente, o gráfico seguinte compara a utilização de TreeSet vs HashSet, na execução de duas queries, a que calcula as linhas duplicadas, e a que faz a listagem dos autores que publicam em todos os anos do intervalo.



	TreeSet	HashSet
Consulta Autores que publicam no Intervalo	0.226867	0.057075
Linhas em Duplicado	0.358043	0.252949

5 Conclusão

A principal dificuldade que foi ultrapassada, foi a escolha da estrutura de dados a utilizar uma vez que certas estruturas seriam melhores para determinadas queries e piores para outras. Optou-se pela estrutura implementada uma vez que globalmente é melhor na maioria dos casos.