UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

INSTITUTO DE INFORMÁTICA

Bacharelado em Engenharia da Computação

INF01124 – Classificação e Pesquisa de Dados – Turma B

Prof. Renan Maffei

Alexandre de Almeida Lima - Matr.: 273164

TRABALHO FINAL DA DISCIPLINA DE CLASSIFICAÇÃO E PESQUISA DE DADOS

ANÁLISE DOS ACIDENTES DE TRÂNSITO OCORRIDOS NO MUNICÍPIO DE PORTO ALEGRE

Porto Alegre

26 de novembro de 2018

PROBLEMA

O problema escolhido foi analisar os índices de acidentes de trânsito na cidade de Porto Alegre, verificando-se onde, quando, como ocorrem esses acidentes. Essa verificação foi feita através de um programa feito com base nos dados fornecidos pela EPTC desde o ano 2000.

Os requisitos para o funcionamento do programa é a base de dados fornecida pela EPTC (http://datapoa.com.br/dataset/acidentes-de-transito) e a manipulação destes dados.

Com o programa, é possível analisar diversos fatores dos acidentes, assim como comprovar a violência no trânsito da capital gaúcha (em 2016 foram 12.515 acidentes registrados no total).

IMPLEMENTAÇÃO

O programa foi desenvolvido em linguagem C# no Visual Studio 2015 e com a utilização de Windows Forms, criando uma interface mais gráfica e interativa do que as linhas de comando as quais estamos habituados no curso.

Foram escritos 5 códigos, mas todos pertencentes ao mesmo *namespace "Trabalho_Final"* e à mesma classe TF_CPD, herdada de Form.

- **TF_CPD.cs:** O código TF_CPD contém um só método, o de inicialização. Nesse método é feito a leitura do arquivo binário para dos dados processados anteriormente. Os dados, novos ou lidos do arquivo gravado, são dispostos em índices e em um arquivo invertido para consulta aos dados, ou seja, a organização dos arquivos é indexado, com um arquivo principal binário de extensão .axl e um conjunto de índices gerados pelo programa. Inicialmente tentei implementar árvores B ou B+, mas encontrei grande dificuldade em trabalhar numa linguagem de alto nível sem poder utilizar ponteiros, tendo que criar classes para manipulação dos dados.

- **Índices.cs:** Nesta parte do código é desenvolvida uma estrutura de dado que utilizei durante todo o programa: a *struct* End_Dado, composta por um endereço e um dado, ambos do tipo *string*. Foi criado também todos os índices necessários para acesso aos dados. Cada índice é formado por uma lista de End_Dado (*List*<End_Dado>). São elas que guardam as informações durante a execução do programa e as gravam no Arquivo Principal.axl (binário).

Utilizar essa estrutura era necessário pois eram muitos dados e todos eles ligados a outros dados dos acidentes: endereço, data, tipo de veículo, número de feridos e mortos, etc. A estrutura facilitava neste sentido de união dos dados, mas criar outras estruturas e

métodos para esse tipo de variável se tornou difícil. Utilizei também as estruturas nativas do C#: *List*, *Queue*, *Stack* e o tipo de variável *string*. Utilizaria também a estrutura *Dictionary*, mas pensei que seria fácil demais trabalhar com ela, e como foi especificado que devemos trabalhar majoritariamente com as nossas estruturas, criei a End_Dado, semelhante a *Dictionary*.

- **Botões.cs:** Esta era a parte responsável pela interface dos botões do formulário e suas ações. É neste trecho do programa que ocorre a inclusão de novos dados externos. Os dados coletados precisam ter a extensão .csv (comma-separated values), comum em tabelas e facilmente visualizado no bloco de notas (se comporta como arquivo .txt) e no Excel (como um arquivo .xls). O programa faz a validação do arquivo para se certificar que é de extensão .csv, mas não faz a validação da organização interna dos arquivos (necessário para o programa funcionar). Durante a leitura do arquivo, o programa vai também gravando o Arquivo Principal na memória secundária na medida que os dados são lidos. Também atualiza os índices de acesso e o arquivo invertido para busca de informações.

- **Arquivo Invertido.cs:** Aqui estão implementados os métodos e estruturas para a busca de dados do programa.

Para a criação do arquivo invertido, primeiramente foi escrita uma *struct* Lista_Invertida, composta por uma *string* chave e uma *List<string>* que serve como a lista de *postings*. A criação da instância do arquivo invertido foi

```
Lista_Invertida[] arquivo_invertido = new Lista_Invertida[99991];
```

Onde 99991 foi o número primo mais próximo de 100.000 por causa do tratamento de colisão da tabela hash.

O tratamento de colisões escolhido foi o de Endereçamento Aberto: Busca Linear, passo 7, pela simplicidade.

O acesso às listas do arquivo invertido é feito através de uma função hash, porém não implementei a função hash, e sim utilizei o método .GetHashCode.

No mesmo código é feita a busca de informações, onde o usuário digita o termo que deseja procurar, o programa utiliza esse termo como chave para a função hash e aponta para o arquivo invertido, que retorna as posições do termo pesquisado.

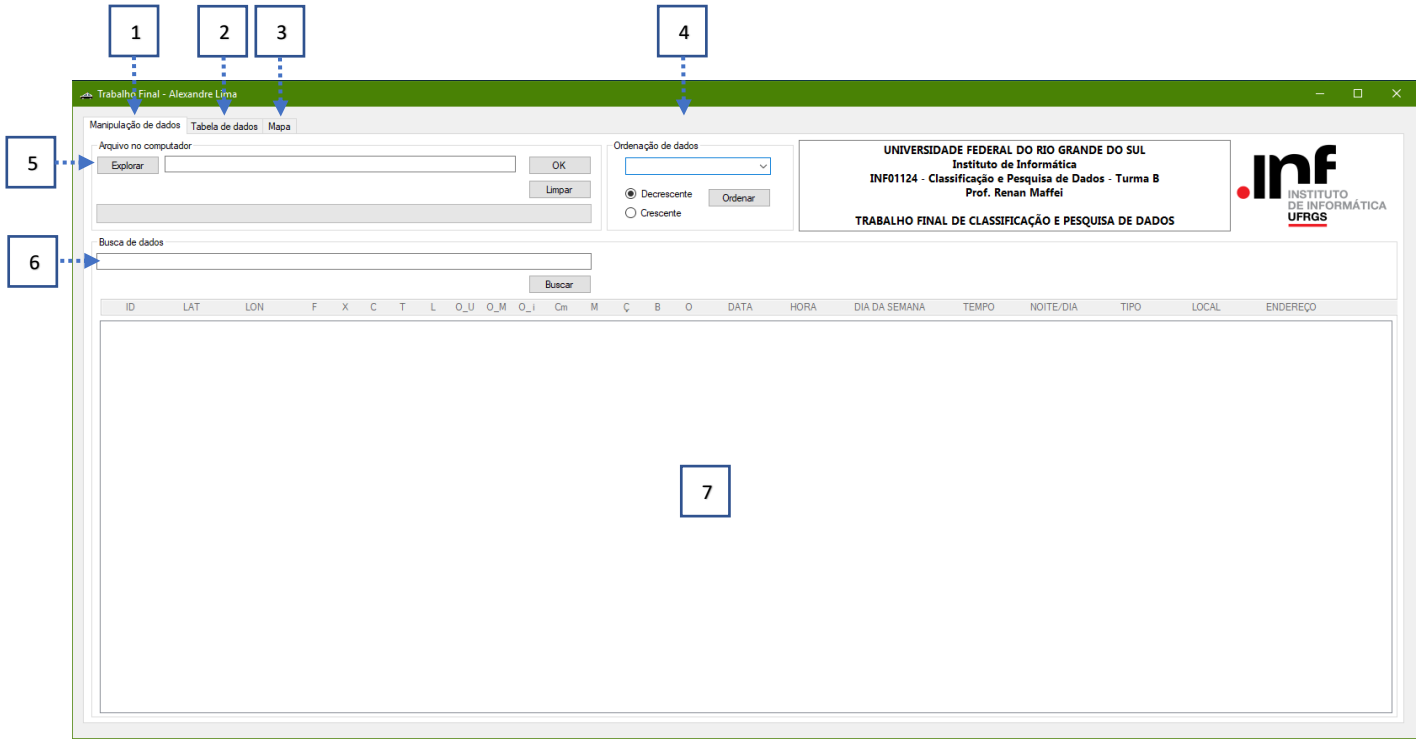
- **Ordenação.cs:** O último código do programa é responsável pela ordenação dos dados dependendo do critério escolhido pelo usuário, que podem ser escolhidos estão numa ComboBox na aba principal.

Por o programa manipular os arquivos por índices, isso torna o processo de ordenação mais fácil. O método de ordenação escolhido foi o RadixSort MSD, pois com ele é possível classificar *strings*, que é o tipo mais comum de variável no programa, ou seja, é uma escolha muito conveniente, mas também muito complexa de programar, já que o argumento de entrada é a estrutura *List*<End_Dado>, pois todos os dados estão ligados a um endereço.

O RadixSort classifica todos os critérios, exceto o ID, que não é uma *List*<End_Dado> e sim uma *List<string>*. Como seria necessário muitas modificações no método RadixSort ou mesmo a criação de outro, optei por utilizar o método .Sort da própria linguagem, lembrando que, por ser uma linguagem .NET, o método .Sort do C# é na verdade o IntroSort, apresentado na aula 12 – Métodos de Ordenação: Epílogo (slide 4).

Além do RadixSort, foi pensado também em utilizar o MergeSort ou QuickSort, pois já tenho os algoritmos prontos do Laboratório 2 e também por suas características de boa complexidade, mas o RadixSort era claramente a melhor escolha (e a mais complicada de implementar).

GUIA DE USO



1 – Aba de Manipulação de Dados

2 – Aba de Tabela de Dados

3 – Aba do Mapa

4 – ComboBox para selecionar critério que deseja ordenar, crescente ou decrescente

5 – Buscar arquivo para inserir no programa

6 – Busca de palavras no arquivo

7 – Área onde é listado os acidentes por ordem ou pelo resultado da busca

Exemplo de uso para inserção de um novo arquivo:

1. Na aba “Manipulação de Dados” (1), clique no botão “Explorar” (5)
2. Uma janela para selecionar um arquivo aparecerá. Procure e selecione o arquivo “acidentes_2016” e clique em “Abrir” e em “Ok”
3. O programa carregará o arquivo assim que a barra de status estiver cheia e a frase “UPLOAD COMPLETO” aparecer
4. Na “Tabela de dados” (2), aparecerá todos os registros lidos
5. Para buscar uma informação, digite a palavra que deseja e clique em “Buscar” (6)
6. Os resultados da busca serão exibidos na caixa 7
7. No item 4, é possível escolher um dos critérios da ComboBox para ordena-los. Eles aparecerão em ordem na caixa 7
8. Todos os dados inseridos são gravados em um arquivo externo .axl, e quando iniciar o programa novamente, os dados gravados estarão disponíveis.

CONSIDERAÇÕES FINAIS

- O objetivo de analisar os dados dos acidentes de carro em Porto alegre foi alcançado.
- O programa está limitado a um único tipo de arquivo de leitura e a uma única formatação possível
- Lentidão para carregar os dados (grande número de informações)
- Ocorrem erros eventuais
- No capítulo Implementação, há palavras coloridas: as azuis são palavras/ informações que aprendemos nas aulas e/ou facilitam o trabalho, já as vermelhas foram as que dificultaram. RadixSort está em laranja pois, apesar de ter facilitado o trabalho de ordenar *strings*, foi complicado escrever o método com *List*<End_Dado> como argumento.

REFERÊNCIAS

SZWARCFITER, Jayme L.; MARKENZON, Lilian. Estruturas de Dados e seus Algoritmos. 3ª. ed. Rio de Janeiro: LTC, 2015. 302 p.

HEJLSBERG, Anders et al. The C# Programming Language. 3ª. ed. [S.l.]: Addison-Wesley, 2009. 754 p.

SANTOS, Clesio Saraiva dos; AZEREDO, Paulo Alberto de. Tabelas: Organização e Pesquisa. Porto Alegre: Bookman: Instituto de Informática da UFRGS, 2001. 85 p. v. 10.

.NET API Browser. 2018. Disponível em: <https://docs.microsoft.com/en-us/dotnet/api/>. Acesso em: 26 nov. 2018.

EMPRESA PÚBLICA DE TRANSPORTE E CIRCULAÇÃO. (Porto Alegre). Acidentes de Trânsito - #datapoa. 2018. Disponível em: <http://datapoa.com.br/dataset/acidentes-de-transito>. Acesso em: 26 nov. 2018.

Slides utilizados nas aulas de Classificação e Pesquisa. 2018. Disponível em: https://moodle.ufrgs.br/course/view.php?id=58136. Acesso em: 26 nov. 2018.