Universidade Federal do Rio Grande do Sul

Instituto de informática

Ciência da Computação

# Pesquisa sobre dados do TSE sobre eleições 2018

Leonardo Holtz de Oliveira e Otávio Moraes Brito Capelão

INF01124 - Classificação e Pesquisa de dados - Turma B

Porto Alegre

24/11/2018

## Introdução

Como parte do trabalho final da cadeira de Classificação e Pesquisa de dados, devíamos tratar uma quantidade de dados em um arquivo textual de escolha livre, desde que envolvesse a utilização dos métodos e estruturas estudadas durante as aulas. Como tema, foi escolhida a seguinte problemática:

O problema visto nas eleições de 2018, assim como no arquivo de dados tratado é que muitas candidaturas foram anuladas porque os candidatos não tinham os requerimentos necessários para a eleição, entretanto, o nome destes candidatos ainda constam no arquivo do Tribunal Superior Eleitoral no seu repositório de dados eleitorais e além disso, estão totalmente desorganizados. Para contornar este problema, após a criação e a inserção de todos os candidatos nas estruturas de dados, deseja-se identificar os dados daqueles que de fato não concorreram no 1° turno da eleição e também os daqueles que participaram do 2° turno, já que seus nomes aparecem repetidos na tabela de dados. Com isso, o programa, além de possibilitar a busca por um candidato específico, possibilitará a criação de novos arquivos que são os dados de candidatos que de fato concorreram somente no 1° turno, os candidatos que concorreram no 2° turno e candidatos que tiveram sua candidatura anulada por não seguirem requerimentos necessários para concorrerem ou que cancelaram por decisão própria. Estes arquivos criados, serão organizados seguindo a ordem alfabética dos nomes dos candidatos.

## Método e Implementação

Para o principal deste trabalho foi usada a linguagem C++. As estruturas de dados utilizadas são as classes e funções provenientes da linguagem e a *Árvore Trie*, um tipo de árvore de prefixos comumente utilizada para a organização de palavras.

Para obter tais dados, utilizamos o arquivo em *csv* obtido no site do TSE sobre as eleições de 2018. Um script em Python foi criado e usado antes para a substituição de caracteres de difícil manipulação como acentos e cedilhas, que a linguagem C++ não consegue tratar naturalmente

para o processamento correto dos dados. O arquivo resultante desse script chama-se *candidatos_2018_BRASIL_updated.csv*.

Para a leitura dos dados foi necessário ler partes da cada linha até o caractere *';'* pois em arquivos csv, ele é o token necessário para dividir os dados em células, quando o arquivo for aberto em modo de planilhas. O arquivo original possui mais de 50 campos por candidato, sendo a maioria irrelevantes para a classificação de cada candidato e redundantes, como vários campos onde se obtinha, por exemplo, o código do estado de um candidato, a sigla do estado e logo após o nome por extenso do estado. Foi decidido então que os campos mais importantes para a nossa aplicação eram os de nome, partido, unidade de federação, cargo, turno, situação e número de cada candidato, pois esses são geralmente as informações mais relevantes para a população brasileira na hora de escolher e identificar um candidato. Cada campo possui também aspas em suas extremidades no arquivo de entrada, e como isso dificultaria a inserção dos nomes na árvore, foi necessário a criação de uma função que faria a retirada destas aspas.

Logo após isso, era crucial, a partir da definição deste trabalho, que os dados fossem inseridos em um arquivo binário. Houve uma grande dificuldade nessa parte da aplicação, pois até então, havíamos criado estruturas específicas para cada candidato. Na verdade, "candidato" é uma classe que possui os campos mencionados anteriormente que foram retirados do arquivo bruto de dados (nome, partido, etc). Entretanto todos esses campos eram da classe string. Para a linguagem C++, existe uma certa dificuldade em escrever strings em arquivos binários porque, ao contrário de arrays de chars, encontrados naturalmente na linguagem C, as strings, quando são declaradas, possuem tamanhos variáveis, ou seja, o dado que é inserido da string é o que definirá o seu tamanho final em bytes. Como cada dado do candidato conta com um tamanho variável, não era possível guardar diretamente uma classe no arquivo ou um nodo da trie diretamente. Foi essencial guardar estas strings separadamente, guardando primeiro o tamanho de uma e logo em seguida o dado da string, para cada um dos sete campos de cada candidato. A leitura do arquivo binário segue de maneira equivalente, foi necessário receber o tamanho da próxima string e

utilizar a função *resize* para reajustar o tamanho em bytes de uma variável da classe string, onde esta variável iria receber o dado da string que foi guardada.

Para cada candidato então, conforme as strings vão sendo lidas do arquivo binário, são criadas e alocadas em memória estruturas do tipo TrieNode que possuem um objeto da classe "candidato" e dados adicionais que caracterizam aquele nodo. Estes dados eram booleanos informando se o nodo era uma raíz, se o nodo possui um dado completo de um candidato, a letra que caracterizava aquele nodo e ponteiros para outros nodos, que são seus filhos. É importante enfatizar de que os filhos de um nodo é um array de ponteiros para o tipo estrutura TrieNode. Os índices desses arrays representam os caracteres usados nos nomes dos candidatos, sendo no total 27 índices, sendo o primeiro deles dedicado ao caractere espaço e os demais dedicados para os caracteres que compõem a ordem das letras do alfabeto em um dicionário, ou seja, de A até Z. A montagem da Trie e a adição das letras e dados dos candidatos é feita de maneira muito simples, primeiramente é primordial utilizarmos o nome do candidato que estamos lendo do arquivo binário para a inserção dos dados dele na árvore. Um ponteiro auxiliar parte da raiz e cada uma das letras do candidato é transformada em um índice onde testamos se o ponteiro auxiliar possui um filho neste mesmo índice. Caso o ponteiro auxiliar possua o filho, o ponteiro é atualizado para apontar agora para o seu filho, caso o contrário, um nodo com a letra que está sendo tratada é criado e o array de ponteiros do índice usado agora apontará para o nodo criado e o auxiliar será atualizado. Ao chegarmos na última letra do nome, todos os dados do candidato que obtemos a partir do arquivo binário são colocados no nodo daquela letra.

Como dito anteriormente na introdução do problema, alguns candidatos aparecem repetidos no arquivo original por justamente terem suas candidaturas sempre refeitas, por mudança de cargo ao qual estavam concorrendo ou por mudança de turno da eleição. Portanto, para não haver candidatos repetidos na Trie, quando fosse detectada a já existência de um candidato no nodo que representa a letra final do seu nome, as informações que iriam permanecer no nodo seriam as que tivessem a situação final do candidato. Ou seja, se o campo

de situação do candidato fosse eleito, suplente ou não eleito, as informações que possuem este campo é que irão prevalecer no final.

O uso da estrutura da Trie é justificada a partir das funções que o usuário seleciona para trabalhar sobre ela. A primeira delas é a função de pesquisa de um candidato, onde o usuário deve digitar o nome completo do candidato e o programa fará uma busca desse nome na árvore Trie e mostrará na tela o restante das informações do candidato que o candidato escolheu. Isso é muito prático em uma árvore Trie pois para a busca do nome, o programa procura na árvore pelos índices referentes a cada letra do nome digitado, verificando se há um nodo sendo apontado pelo ponteiro dos filhos. No momento em que o programa não acha um dos nodos, ele sabe que aquele nome não está dentro da Trie. Isso poderia certamente ser substituído por uma tabela hash, onde o nome digitado passaria por uma função hash e o candidato poderia ser facilmente achado, caso ele estivesse ali, a partir de um índice da tabela. Entretanto, para a próxima função, uma hash não serviria muito bem. Esta função é a varredura completa dos candidatos na estrutura para ordená-los em ordem alfabética por nome. Ao se fazer uma varredura em uma Trie, da esquerda para a direita, pela forma como convencionamos os arrays de filhos nas ordens dos caracteres, quando achamos um candidato após o outro, eles sempre estarão em ordem alfabética, sem precisarmos utilizar nenhum algoritmo de ordenação de strings. Assim, a função de varredura seleciona os candidatos a partir de certos atributos, explicados mais pra frente no guia de uso, e os coloca em um novo arquivo csv totalmente ordenados pelo nome.

## Guia de uso

Para executar o programa é necessário que certos arquivos estejam dentro do diretório do executável. O primeiro deles é o arquivo *candidatos_2018_BRASIL_updated.csv*. Como dito anteriormente, este é um arquivo que foi modificado a partir de um script em Python que retirava acentos e caracteres especiais que a língua portuguesa apresenta, mas que o C++ não suporta sem bibliotecas e funções especiais. Ele é necessário pois será dele que obteremos os dados essenciais

para a aplicação. O arquivo original, *candidatos_2018_BRASIL.csv* está disponível no site do Tribunal Superior Eleitoral, no seu repositório de dados eleitorais. O segundo arquivo é *dados_candidatos.bin*, e caso a segunda opção do menu do programa seja escolhida (que será explicada mais para frente), ele será o arquivo binário necessário para que se faça a leitura de dados diretamente.

Ao executar o programa, uma tela de boas vindas é apresentada e o usuário deve selecionar uma de três opções que o programa mostra, digitando o número da opção no teclado. A primeira opção lê os dados dos candidatos do arquivo *candidatos_2018_BRASIL_updated.csv*. Caso o arquivo não esteja dentro da mesma pasta do executável, o programa informará isso e voltará ao menu. Os dados lidos a partir do arquivo são escritos no arquivo *dados_candidatos.bin*, que é criado caso não exista, ou substituído caso ele já exista. Os dados de cada candidato são guardados separadamente, justamente por um problema de um benefício que o C++ traz que são as classes de string. Como a maioria dos dados são em formato string, eles variam de tamanho a todo momento, e portanto, o tamanho das strings deve ser adicionado ao arquivo para que a recuperação delas na leitura seja sucedida. Após a inserção de todos os dados no arquivo binário, o programa reabre este mesmo arquivo para inserir candidato a candidato em uma árvore Trie. Este passo também acontece imediatamente quando se escolhe a opção 2 do programa, quando há a existência do arquivo binário no diretório do programa, mesmo antes de executarmos ele. A terceira opção do programa apresenta uma mensagem final e a aplicação é fechada.

Após a inserção dos dados na Trie, um segundo menu aparece para o usuário, da mesma forma como o primeiro, e o usuário deve digitar a opção que deseja. A primeira opção é a pesquisa por um candidato específico na Trie a partir de um nome que o usuário oferece. Este nome é procurado ao verificarmos cada letra dele nos nodos da árvore. Quando o nodo final é encontrado, as informações obtidas do candidato aparecerão na tela para o usuário ler, e ele poderá selecionar outra opção do menu. A segunda opção permite a criação das tabelas mencionadas anteriormente, o usuário deverá selecionar qual tabela ele deseja e o programa fará

uma varredura da esquerda para a direita, procurando pelos nodos cujos candidatos possuem os dados que satisfaçam a tabela selecionada, esses dados são colocados em ordem alfabética pelo nome do candidato em arquivos que estão no mesmo diretório do programa. Estas tabelas estão no mesmo formato que o arquivo bruto, entretanto, elas são melhores organizadas e possuem somente os dados mais importantes e de interesse do eleitor na hora de votar em um candidato. A última opção do menu, novamente, fechará a aplicação. É importante reforçar que o arquivo binário de dados e as tabelas permanecem no diretório mesmo após a aplicação fechar.

## Considerações Finais

O uso da árvore Trie facilita muito a resolução do problema. Como os dados de entrada estão completamente desorganizados, a busca por informações somente neles se torna completamente inviável. Ao organizarmos todos os dados em uma Trie, não só facilitamos a busca por uma informação específica em uma grande quantidade de dados por meio de um dicionário, como também podemos filtrar e organizar os dados em ordem alfabética mais facilmente, sem precisarmos utilizar um algoritmo de ordenamento como o Radix Sort em uma grande lista unidimensional.

# Referências

https://pt.wikipedia.org/wiki/Trie- Definição da estrutura Árvore Trie

https://www.geeksforgeeks.org/trie-insert-and-search/- Explicação de o implementação de uma Trie em C++.

http://www.cplusplus.com. Documentação da linguagem C++.

https://www.cs.usfca.edu/~galles/visualization/Trie.html- Visualização do funcionamento de uma Trie.

Aula 20 - Árvores TRIE e PATRICIA - INF01124 - Classificação e Pesquisa de Dados, MOODLE UFRGS - Slides das aulas de Árvores da cadeira CPD do Professor Renan Maffei.

Aula 22 - Organização de Arquivos - INF01124 - Classificação e Pesquisa de Dados, MOODLE UFRGS - Slides das aulas de arquivos da cadeira de CPD do professor Renan Maffei.