Alunos: Isaque Beirith e Luis Fernando Mendonça Junior

Projeto 2 - Relatório

Introdução

O algoritmo utilizado implementa operações em uma árvore de prefixos chamada Trie. O programa lê um arquivo de dicionário contendo palavras, constrói a árvore Trie com base nessas palavras e, em seguida, permite pesquisar prefixos e obter informações relacionadas a eles. Possui a classe principal denominada de TrieNode, que possui 5 métodos, além de 2 funções para realizar a implementação.

Classe

TrieNode

Essa é a classe principal do programa, ela representa um nó de uma árvore Trie, onde sua raiz é um caractere vazio (‘ ’), possuindo um vetor de ponteiros para 26 filhos, um para cada letra do alfabeto, e cada um desses nós possuem membros privados que contém: o caractere associado ao nó; novamente um vetor de ponteiros, onde cada posição aponta para um objeto TrieNode, que está associado a uma letra do alfabeto; o número de palavras que têm o prefixo representado por esse nó; a posição da palavra completa associada a esse nó no dicionário; e o comprimento da palavra completa associada a esse nó no dicionário.

O construtor da classe é o responsável pela criação do objeto TrieNode, ele recebe como argumento a letra que será associada a esse nó, e atribui a letra do argumento a letra membro do objeto TrienNode. Em seguida um loop é utilizado para gerar o vetor de tamanho 26, onde cada índice é reservado para um ponteiro para uma letra do alfabeto, mas inicialmente cada índice do ponteiro é nulo, garantido que inicialmente não haja filhos associados ao nó. Finalizando definindo os membros prefixos, posição e comprimento como zero.

Métodos

1. Insert

Esse é o método que tem a função de inserir uma palavra na árvore Trie. Ele recebe como argumento uma palavra, sua posição no dicionário e seu comprimento. Primeiramente ela faz uma verificação para se o tamanho da palavra é zero, caso positivo, a palavra vai estar vazia, não possuindo nada para ser inserido na árvore, então retorna imediatamente. Caso a palavra possua um tamanho maior que zero, a primeira letra da palavra é atribuída a um caractere 'c', e para determinar a posição correta da letra no array de filhos que correspondem a primeira letra da palavra, um valor inteiro é atribuído pela subtração de 'c' pelo valor de 'a' em ASCII, obtendo um número entre 0 e 25 que corresponde ao índice correto no vetor.

Após saber a posição ele verifica se o índice no array de filhos é nulo, caso positivo, quer dizer que não a letra não foi inserida como filho deste nó, então um novo nó é criado, onde 'c' é passado como argumento para o construtor e com o nó criado, se o tamanho da palavra é igual a 1, são atualizadas as informações de posição e comprimento do novo filho.

Recursivamente o método insert é chamado novamente no filho do nó atual, passando o mesmo comprimento e posição como argumentos, mas a palavra fornecida não possui mais o primeiro caractere, assim repetindo esse processo até a palavra original não possuir mais letras.

Por fim, o contador de prefixos do nó atual é incrementado em 1, para indicar que uma palavra com o prefixo que o nó representa foi inserida.

2. contarPrefixos

Esse é o método responsável por contar o número de palavras que possuem a palavra fornecida no argumento do método como prefixo na árvore Trie. A primeira verificação é feita para determinar se a palavra está vazia, caso positivo, quer dizer que não existem mais letras adiante para percorrer, assim retornando o valor do nó atual de prefixos.

Caso negativo, a primeira letra da palavra é atribuída a uma variável 'c', e para determinar a posição correta desta letra no array de filhos, um valor inteiro é atribuído pela subtração de 'c' pelo valor de 'a' em ASCII, obtendo um número entre 0 e 25 que corresponde ao índice correto no vetor. Se essa posição no vetor for nula, significa que não há palavras na árvore Trie que tenham esse prefixo,portanto, a função retorna 0. Caso a posição no vetor exista, recursivamente, o método é chamado novamente no filho do nó atual, passando como argumento a palavra original sem sua primeira letra, repetindo o processo até a palavra ficar vazia.

3. retornaPos

Esse é o método responsável por retornar a posição onde a palavra fornecida como argumento do método está localizada no dicionário. A primeira verificação é feita para determinar se a palavra está vazia, caso positivo, quer dizer que não existem mais letras adiante para percorrer, assim retornando a posição associada ao nó atual.

Caso negativo, a primeira letra da palavra é atribuída a uma variável 'c', e para determinar a posição correta desta letra no array de filhos, um valor inteiro é atribuído pela subtração de 'c' pelo valor de 'a' em ASCII, obtendo um número entre 0 e 25 que corresponde ao índice correto no vetor. Se essa posição no vetor for nula, significa que a palavra indicada não está presente na árvore Trie, portanto, a função retorna 0. Caso a posição no vetor exista, recursivamente, o método é chamado novamente no filho do nó atual, passando como argumento a palavra original sem sua primeira letra, repetindo o processo até a palavra ficar vazia.

4. retornaComp

Esse é o método responsável por retornar o comprimento no dicionário do significado da palavra fornecida como argumento do método. A primeira verificação é feita para determinar se a palavra está vazia, caso positivo, quer dizer que não existem mais letras adiante para percorrer, assim retornando o valor de comprimento do nó atual.

Caso negativo, a primeira letra da palavra é atribuída a uma variável 'c', e para determinar a posição correta desta letra no array de filhos, um valor inteiro é atribuído pela subtração de 'c' pelo valor de 'a' em ASCII, obtendo um número entre 0 e 25 que corresponde ao índice correto no vetor. Se essa posição no vetor for nula, significa que a palavra indicada não está presente na árvore Trie, portanto, a função retorna 0. Caso a posição no vetor exista, recursivamente, o método é chamado novamente no filho do nó atual, passando como argumento a palavra original sem sua primeira letra, repetindo o processo até a palavra ficar vazia.

5. delete

Esse é o método responsável por liberar a memória ocupada pela árvore Trie. Ele utiliza de um loop para percorrer o vetor de filhos do nó atual. Dentro do loop ele verifica se o filho correspondente ao nó atual é nulo, caso positivo, caso positivo, recursivamente a função é chamada no nó atual até chegar

em um nó que não possua filhos. Após sair do loop, com todos os nós filhos excluídos, o nó atual é deletado para liberar espaço na memória.

Funções

1. extractDado

Essa é a função responsável por extrair todas as palavras com seus significados do de um dicionário e construir uma árvore Trie com base nelas. Ela recebe o nome do arquivo na forma de uma string e também um ponteiro para a raiz da árvore como argumentos.

Primeiramente é declarada uma string 'word' que será utilizada para armazenar temporariamente cada palavra extraída do dicionário, após isso é realizada a leitura e o arquivo é aberto. É criada uma string 'line' para armazenar cada linha do arquivo, assim como uma variável inteira 'Pos' para a posição de cada palavra, iniciada com valor 0. É iniciado um loop em que é lido cada linha do arquivo aberto, em cada linha é utilizada a função find() para identificar a abertura e fechamento dos colchetes ('[',']') a partir da posição atual, assim pela posição da abertura e do fechamento do colchete é possível determinar a palavra dessa linha do dicionário. A palavra é extraída e utilizando a função insert ela é adicionada na árvore Trie, também são enviados como argumentos a posição 'Pos' atual e o comprimento da linha. A variável 'Pos' é atualizada com a soma de seu valor atual com o tamanho da linha, mais 1.

Após o término do loop o arquivo é fechado.

2. main

Essa é a função principal do projeto, ela executa a lógica central do programa, recebendo as informações, criando a árvore Trie utilizando os dados obtidos e realizando consulta nesses dados.

Ela inicia declarando um ponteiro para a raiz da árvore Trie, que é criada enviando um caractere vazio para o construtor. Após criadas duas strings, uma denominada 'filename', para armazenar o nome do arquivo fornecido, e uma denominada 'word', para armazenar as palavras inseridas para a consulta. O programa então aguarda a entrada com tais informações.

Então a função extractDado, utilizando o nome do arquivo como argumento e o ponteiro para a raiz da árvore, a função extrai os dados e constrói a árvore.

É iniciado um loop, dentro dele é recebida uma palavra inserida pelo usuário, é declarada uma variável booleana 'dict' iniciada como false. É verificado se o tamanho da palavra inserida é igual a zero, caso positivo o algoritmo sai do loop, caso negativo, as funções contarPrefixo, retornaPos e retornaComp são chamadas na raiz na árvore, sendo enviado para todas elas a palavra inserida, o retorno dessas funções são armazenados em suas respectivas variáveis.

É verificado se o número de prefixos é igual a zero, caso positivo o programa informa que a palavra inserida não é prefixo de nenhuma outra, caso negativo, ele informa de quantos prefixos a palavra inserida possui; então ele verifica se o comprimento da palavra é diferente de zero, caso positivo ele informa ao usuário a posição que ela se encontra no dicionário, caso negativo a palavra não está presente no dicionário. Após as verificações o loop começa novamente, até acabarem as palavras a serem inseridas, finalizando o programa.

Dificuldades

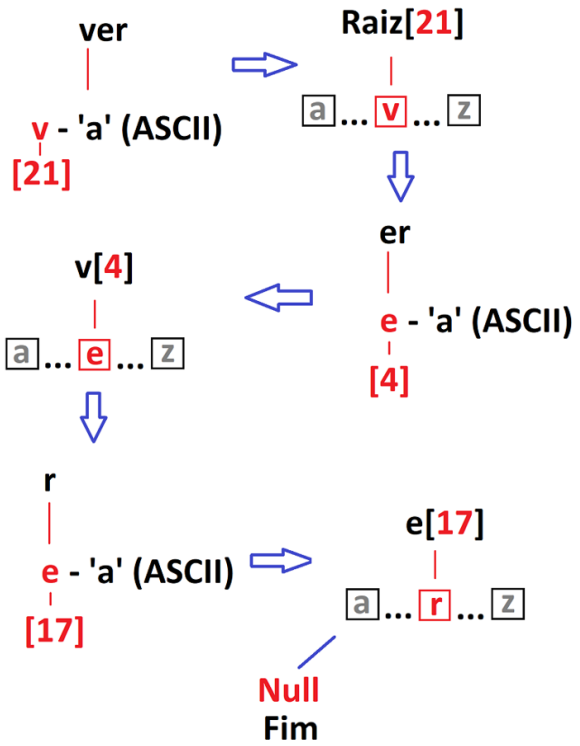
Nesse projeto nossa dificuldade inicial foi compreender o problema, tivemos que buscar em alguns artigos e visualizar algumas aulas para entender o comportamento da árvore Trie, tivemos também que ler o enunciado algumas vezes para compreender realmente o comportamento da árvore e como ela se comportava em relação às palavras, letras e os vetores filhos, mas após a compreensão da árvore Trie e de seu funcionamento, sua implementação foi relativamente simples.

Nossa segunda dificuldade apareceu na hora de apresentar o resultado das operações na tela, comparando o nosso resultado obtido com o resultado esperado nós notamos que a nossa implementação estava entendendo que todas as instâncias de palavras estavam presentes no dicionário, sejam elas palavras existentes ou não, e por consequência disso, elas estavam recebendo a posição de seus sufixos e esses resultados incorretos estavam saindo no output do programa.

Inicialmente para resolver isso, pensamos em colocar todas as palavras recebidas em um vetor global e apresentar somente as informações daquelas que estavam nesse vetor, mas acabamos nos confundindo e sem perceber adicionamos todas as instâncias da palavra nesse vetor, ou seja, o resultado era o mesmo. Para resolvermos esse erro em definitivo, descartamos o vetor de palavras e na hora de apresentar o resultado, nós verificamos se o comprimento da palavra é diferente de zero, assim assegurando que a palavra existe e que os dados estão corretos, uma solução mais simples do que esperávamos.

Explicação do algoritmo com imagens

Representação do método insert utilizando uma imagem:



Nessa demonstração, temos uma árvore Trie vazia e utilizamos o método insert para inserir nela a palavra ‘ver’. O método começa pegando a primeira letra da palavra e verificando sua posição no vetor de filhos, cria-se então um novo novo nó utilizando a letra ‘v’, após isso, o filho criado em ‘v’ chama recursivamente a função insert, mas sem a primeira letra, ele pega a nova primeira letra e verifica sua posição no vetor, assim é criado um novo nó para o filho e ele vai repetir o processo até a

palavra enviada pelo filho ser uma palavra vazia, assim o método insert irá finalizar e a árvore resultante será essa: