Universidade Federal de Santa Catarina Centro Tecnológico - CTC
Departamento de Informática e Estatística - INE

Relatório

Estrutura de Dados

Prof.
Aldo Von Wangenheim e
Alexandre Goncalves Silva

Alunos
Micael Angelo Sabadin Presotto (22104063) e
Eduardo Gwoszdz De Lazari (22100612)

Classes:

Trie:

A classe Trie possui apenas um parâmetro: raíz. Esse parâmetro representa o nó raiz da árvore, que a partir dele serão geradas todas as outras palavras. Suas funções (além de construtor e destrutor) são inserir e procurar.

A função de inserir é responsável por guardar a palavra na árvore. Para isso, é necessário percorrer os nós da árvore, criar nós caso ainda não existam e armazenar o comprimento do seu significado e posição armazenadas no dicionário. Além disso, ele também atualiza a quantidade de prefixos que cada trecho de palavra é.

A função de procurar é responsável por procurar uma palavra na árvore. Para isso, é necessário percorrer os nós da árvore relacionados com as letras da palavra. Caso ela exista, ele retorna o último nó da palavra, contendo seus dados (posição e comprimento). Caso contrário, retorna nullptr.

NoTrie:

A classe NoTrie possui 4 parâmetros = filhos, posição, comprimento e prefixos. O parâmetro filhos representa os 26 filhos (26 letras do alfabeto) que o nó pode se conectar. Posição representa a posição da palavra no dicionário. Comprimento representa o comprimento do seu significado. Prefixos representa quantas palavras são derivadas a partir desse nó.

A classe NoTrie possui apenas uma função (fora construtores): deletar. Essa função serve para deletar todos seus filhos de maneira recursiva, ajudando no destrutor da Árvore (classe Trie). O construtor padrão cria 26 filhos nullptr para esse nó e define todos os parâmetros como 0. Além dele, existem 2 outros construtores, que utilizam dele para alterar parâmetros, como é o exemplo do 2º, onde é necessário informar a posição e comprimento para criar o nó.

Problemas:

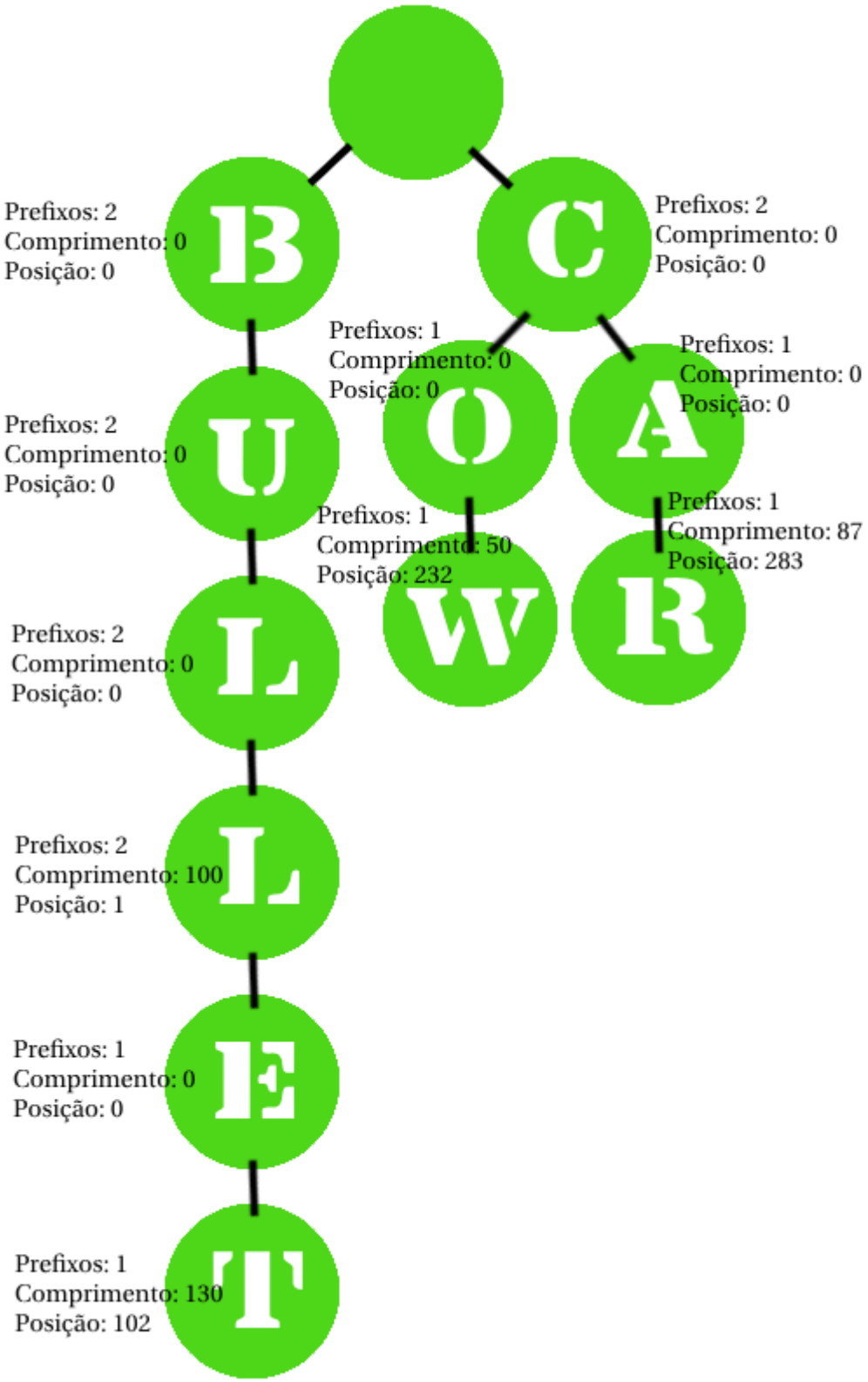
Primeiro problema:

Para a identificação de prefixo, foi utilizada a lógica de alterar o parâmetro prefixo, que todos nós possuem, e incrementá-lo quando uma nova inserção passa por ele.

Para o insert utilizamos a seguinte lógica: Primeiramente colocamos todas as letras para minúsculas, pois assim, não há problemas de palavras como “ave” e “aVE” serem guardadas diferentes. Com isso, percorremos a string passada letra por letra, e as subtraímos por ‘a’, para assim, pegarmos seu valor em ASCII.

Após isso, acessamos o índice do vetor filhos que corresponde à subtração. Caso ele ainda seja nullptr, é atribuído à ele um NoTrie. Se for o fim da palavra, é utilizado o construtor completo (pois assim, é definido os valores de comprimento e posição), e se for somente uma letra da palavra, é utilizado o construtor primário. A cada nó que se passa na inserção, é incrementado a sua variável prefixos, pois ele será um prefixo da nova palavra.

Para exemplificar melhor a ideia do prefixo, iremos exemplificar com uma imagem representando a inserção de duas palavras : bull e bullet.



Exemplo de como os nós se comportam na Trie.

Como podemos ver na imagem, a variável prefixos de ‘b’,‘u’,’l’,’l’, será 2, ou seja, indica que bull é prefixo de duas palavras: ele mesmo e bullet. Assim, conseguimos retornar a quantidade de prefixos que as palavras representam.

No próximo trecho, é requisitado ao usuário múltiplas vezes palavras a serem encontradas no dicionário. Caso for somente um prefixo de uma palavra, e não uma palavra em si, é apenas retornada a quantidade de palavras armazenadas que podem ser formadas por ele. Caso for uma palavra, são retornados sua posição no dicionário, o comprimento de seu significado, e a quantidade de prefixos que a palavra pode ser. Por exemplo, a palavra “ela” além de ser uma palavra, pode ser um prefixo de “elástico” e “elasticidade”, sendo assim, seu argumento “prefixos” será igual a 3. Caso o usuário digite “0”, o programa será finalizado.

Segundo problema:

O segundo problema se trata da explicação de como fizemos a lógica para encontrar a posição da palavra e o tamanho da linha.

Primeiramente iniciamos um inteiro posição com 0, pois logicamente, a primeira palavra estará na posição 0 do arquivo. Após isso,é utilizada uma lógica de a cada linha verificar a posição de abertura e fechamento dos colchetes ([ ]), pois dentro deles se encontra a palavra. Além disso, ele obtém o tamanho da linha (que será responsável pelo comprimento) e a posição. Para obter a posição, é utilizada a posição anterior + o tamanho da linha anterior + 1, pois todas as palavras estarão no começo de uma linha. Após isso, a árvore é responsável por utilizar seu método inserir para adicionar a palavra à árvore.

```
ifstream my_file(filename);
if (my_file.is_open()) {
    string linha;
    size_t tamanho_anterior = 0;
    while (getline(my_file, linha)) {
        string palavra = linha.substr(linha.find('[',0) + 1, linha.find(']',0) - 1);
        t→inserir(palavra, tamanho_anterior, linha.length());
        tamanho_anterior += linha.length() + 1;
    }
    my_file.close();
}
```

Trecho de código responsável para obter as palavras, seus comprimentos e posições e assim inserir na Trie.

Conclusão:

Em geral, o grupo não encontrou muitas dificuldades na realização do projeto. A questão mais debatida foi a de como guardar cada letra associada a um nó por um índice inteiro, que como explicado previamente, utilizou-se a subtração por ‘a’. Além do mais, a questão de transformar a string para minúscula foi implementada, mesmo os inputs não possuindo palavras com caracteres maiúsculos, para adaptar o código a todos os casos possíveis.

Referências

Geeks for Geeks

Estrutura de Dados - Aula 20 - Árvores N árias; Tries