Relatório da implementação 3 – *upgrade* da tabela *hash*

Nome: Moisés Silva de Azevedo

RGA: 2022.0743.004-6

Disciplina: Estrutura de Dados e Programação I

Professor: Ronaldo Fiorilo

Introdução

O seguinte relatório da terceira implementação tem como objetivo esclarecer aspectos técnicos e sobre as principais decisões de projeto na implementação de um dicionário de palavras e sinônimos para a língua portuguesa, usando a linguagem de programação C e conceitos de estruturas de dados.

Como os dados estão estruturados

Os dados estão definidos em dois tipos: palavras e sinônimos. Um sinônimo é um nó de uma lista encadeada. Uma palavra é um nó de uma árvore AVL. Cada nó de árvore AVL possui um campo que aponta para uma lista de sinônimos.

As palavras estão guardadas em uma tabela de dispersão, onde as colisões são tratadas através do encadeamento externo em uma árvore AVL.

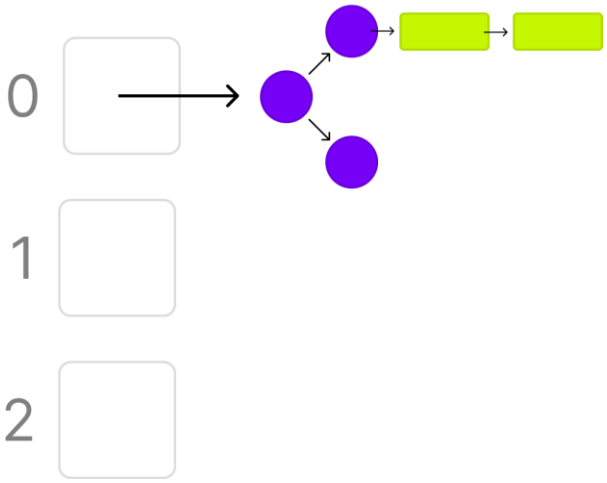

Figura 1 - Representação gráfica da estrutura dos dados

Tamanho da tabela

Esse foi um dos problemas da primeira implementação, onde escolhi um número muito pequeno para os requisitos reais do problema. Nessa implementação busquei entender como o programa se comportaria ao lidar com 400,000 palavras. Segue o raciocínio que me fez chegar à escolha do tamanho.

De acordo com a agência brasil, o vocabulário da língua portuguesa tem aproximadamente 370 mil palavras distintas. Levando em consideração esse fato, foi escolhido o tamanho de 4000 posições, pois é um número um pouco maior que 3700, 1% do total de palavras da Língua Portuguesa.

Além disso, uma tabela desse tamanho, em condições ideais (função de dispersão uniforme), tem a capacidade de acessar a informação em tempo O (1) para 4,000 palavras que estão na raiz do índice pesquisado (endereço base).

Se a tabela armazena 400,000 palavras, cada posição do vetor aponta para uma árvore AVL com **aproximadamente** 100 nós cada. Isso implica que os 99%

restantes das palavras tem tempo de acesso de até $O(\log_n)$, onde n não é 400 mil, mas sim 100.

100 * 4,000 = 400,000

Função de dispersão

A função de dispersão utilizada é a mesma da primeira implementação, com a alteração na constante k de multiplicação, nessa versão deixei o padrão 33 que normalmente é usado. A constante influência no peso que o caractere atual tem na palavra.

Exemplo: A palavra “laranja” tem três ‘a’, no entanto cada um está em uma posição diferente. Para obter o valor relativo desse caractere, multiplica-se o código ASCII por k^i, onde i é a posição do caractere na palavra.

Persistência dos dados

Os dados estão armazenados em um arquivo de texto chamado “data.txt”. Os dados estão organizados da seguinte forma: a primeira linha do arquivo guarda um número inteiro que representa a quantidade total de palavras da estrutura, esse número é usado para informar ao algoritmo de leitura quantas iterações ele deve fazer.

A próxima linha guarda uma palavra (*string*), a linha de baixo guarda a quantidade total de sinônimos associados a essa palavra e logo abaixo encontra-se a lista de sinônimos, um em cada linha.

Os dados das palavras não estão salvos em ordem alfabética, mas sim de acordo com o endereço base na tabela. Exemplo o endereço base 0 pode conter uma raiz cuja palavra começa com ‘R’, e o endereço base 1 pode ter uma raiz onde a palavra começa com ‘J’.

No entanto as listas de sinônimos estão em ordem alfabética.

Correção dos problemas da primeira implementação

- remove *str*: Corrigido a falha se segmentação ao tentar remover uma palavra que não existe na estrutura quando o programa está carregado.
- remove *str1 str2*: Corrigido, agora realiza a remoção bidirecional. Corrigido a falha de segmentação de um sinônimo que não está na lista de sinônimos.

Correção dos problemas da segunda implementação

- A inserção agora é realizada em um passo só, a função de inserção de um nó na árvore recebe como parâmetro a *str1*, *str2* que será inserido na lista de sinônimos do nó criado para *str1*. Se a busca encontrar a palavra *str1* na estrutura, apenas insere *str2* na lista, e não cria um nó para *str1*.
- Utilizado o percurso em pré-ordem para salvar os dados da árvore AVL.

Dificuldades do trabalho

Considero que foi a implementação mais tranquila, pois foi possível aproveitar bastante do trabalho feito nas implementações anteriores. O trabalho foi juntar as peças, corrigir e melhorar os pontos fracos das versões passadas.