UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

IMPLEMENTAÇÃO DO AUTOCOMPLETAR EM LINGUAGEM C COM ESTRUTURAS DE DADOS TIPO LISTA E ÁRVORE

NATANAEL DA SILVA DEBONA - 00219823
RAFAEL JÚNIOR RIBEIRO- 00265830

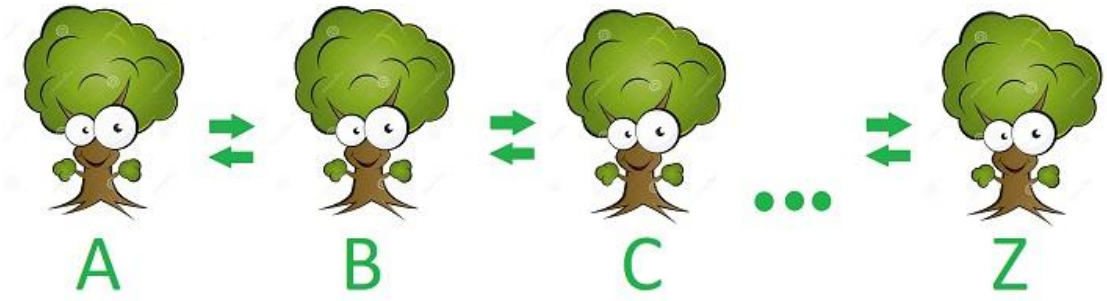
Porto Alegre, 27 de julho de 2017

Introdução

Este relatório tem como finalidade expor a escolha das estruturas utilizadas, bem como as funções implementadas, e apresentar os motivos que justificam tais escolhas.

A estrutura utilizada está dividida em 2 partes, criação da estrutura dicionário, a qual recebe as palavras que serão usadas como base de dados, e criação da estrutura previsões, a qual recebe prefixos de palavras e apresenta as previsões para estes prefixos ao usuário.

A organização dos dados para a criação do dicionário se deu através de uma lista duplamente encadeada de AVLs. A organização dos dados na parte das previsões se deu por uma ABP.



TAD do programa

```
struct lista
{
char letra_index;
pNodoA *letra_arvore;
lista *prox;
lista *ant;
};
structTNodoA
{
char palavra[10];
int peso;
int FB;          /*Campo não usado na ABP*/
pNodoA *esq;
pNodoA *dir;
};
```

/*Inicia a estrutura que irá receber as palavras do wikitionary. Uma floresta - lista - de árvores*/
lista* inicia_floresta(lista *inicio);
/*Na floresta haverá uma árvore para cada letra do alfabeto. Essa função inicia cada uma delas*/
lista* inicia_Arvore(lista *floresta, char letra);
/*Função com o objetivo de pegar o endereço da árvore 'z', a árvore final da floresta*/
lista* pega_fim(lista *inicio);
/*Calcula a altura. Útil para calcular o fator do nodo de uma árvore*/
int Altura (pNodoA *a);
/*A partir da altura retornada, calcula o fator do nodo de uma árvore*/
intCalcula_FB(pNodoA *a);
/*Funções relacionada a operação com arvores AVL*/
pNodoA* rotacao_direita(pNodoA *pt);
pNodoA* rotacao_esquerda(pNodoA *pt);
pNodoA* rotacao_dupla_direita (pNodoA* pt);
pNodoA* rotacao_dupla_esquerda (pNodoA* pt);
pNodoA* Caso1 (pNodoA* a, int *ok);
pNodoA* Caso2 (pNodoA *a, int *ok);
pNodoA* insere_na_arvore(pNodoA *a, char *palavra,int peso, int *ok);
/*Preparação para o início da inserção de dados na floresta. Chama a função insere_na_árvore
setando as variáveis necessárias para execução desta função*/
void insere_na_floresta(lista *inicioflorestа,lista*fimfloresta,char *palavra,int peso);
/*Printa a árvore de uma letra. útil para debugar o código*/
void printaArvore(pNodoA *arvore_letra);

/*Printa a floresta/dicionário. útil para debugar o código*/
voidprintaFloresta(lista *inicio);
/*Dado uma letra, retorna a árvore desta letra*/
lista* busca_arvore(lista *iniciofloresta, lista *fimfloresta,char *palavra);
/*Dado um prefixo do arquivo de consulta, retorna uma árvore com TODAS as previsões desse prefixo.*/
pNodoA* pega_previsoes(lista *iniciofloresta, lista *fimfloresta,char *palavra,char *palavraLimite);
/*Constrói a arvore de previsões procurando a partir da esquerda da árvore*/
pNodoA* busca_previsoesESQUERDA(pNodoA *arvore,char *palavra, char *palavraLimite, pNodoA *arvorePrevisoes,int *flag);
/*Constrói a árvore de previsões procurando a partir da direita da árvore*/
pNodoA* busca_previsoesDIREITA(pNodoA *arvore,char *palavra, char *palavraLimite,pNodoA *arvorePrevisoes,int *flag);
/*Insere efetivamente uma palavra do dicionário na árvore de previsões*/
pNodoA* insere_previsoes(pNodoA *arvore, int peso, char *palavra);
/*Escreve as previsões no arquivo de saída*/
void escreve_previsoes(FILE *arquivoSaida,pNodoA *arvorePrevisoes, int *cont,int limitePrevisoes);

Análise

1 - Criação do dicionário

Para efetuar a montagem na memória das palavras recebidas pelo arquivo de palavras .txt, o layout utilizado se deu na junção de duas estruturas de dados, uma lista duplamente encadeada e uma árvore AVL. Avaliamos também a possibilidade da implementação de apenas uma única árvore AVL englobando todas as palavras, mas notamos que deste modo teríamos uma busca mais ineficiente.

1.1 - Lista duplamente encadeada

Ao ser iniciado o programa, é efetuada a criação de uma lista duplamente encadeada, ordenada em ordem alfabética e com cada nodo representando uma árvore AVL, que por sua vez representa uma letra do alfabeto.

Como ideia inicial, tínhamos proposto utilizar uma ABP, porém já estávamos utilizando uma ABP na criação das previsões, então avaliamos que seria mais elegante utilizar uma LDE, pois abrangeria de maneira mais completa o conteúdo visto em aula. Além disto, a diferença no tempo de processamento geral do programa ao usar uma ABP no lugar da LDE não é significativa, visto que a lista é percorrida por ambos os lados, tanto no sentido a-> z quanto no sentido z->a.

1.2 - Árvore AVL

Árvores oferecem uma busca mais otimizada do que listas. e entre os tipo de árvores estudadas, a que oferece o melhor balanceamento possível é a AVL. A chave de ordenamento nesta árvore é a palavra.

2 - Formação das previsões

Aqui fizemos uso de uma ABP, pois é forma mais eficiente de inserir novas previsões, sem levar em consideração o balanceamento, pois este é desnecessário, tendo em vista que são visitados todos os nodos da árvore. A chave de ordenamento nessa árvore é o peso da palavra.

Conclusão

Ao realizarmos tal trabalho, conseguimos colocar em prática a teoria vista durante o semestre, passada pela professora Renata Galante, e solidificar na memória estes conhecimentos.

Vimos também que o recurso de autocompletar é bastante utilizado no dia-a-dia, em motores de busca na Web, serviços de mensagem de celular e editores de código e passamos a entender na prática o seu funcionamento.

Ao enfrentar os problemas que foram surgindo na codificação do programa, conseguimos aprimorar os nossos conhecimentos de depuração de código, além de entender, de forma simplificada, a confecção de um projeto corporativo.