ESTRUTURA DE DADOS I

Nomes:

Eduardo Figueiredo Freire Andrade - 11232820

Rafaela Cristina Bull - 11233651

PROJETO II - VERIFICADOR DE PALAVRAS

Modelagem da Solução

Estrutura e Tad:

Inicialmente foi escolhida uma estrutura para a implementação do dicionário, como a operação mais importante e recorrente no programa será a busca, foi escolhida a árvore AVL cujo dado armazenado é do tipo vetor de char (também definido como ITEM). Esse tipo de árvore mantém seu balanceamento independente da ordem em que os elementos são inseridos, esse balanceamento faz com que as buscas ocorram com maior eficiência, mantendo-se mesmo no pior caso com a complexidade de processamento O(log(n)).

A implementação dessa estrutura é feita de maneira dinâmica, utilizando-se de alocação dinâmica e ligações com ponteiros que evitam o desperdício de espaço, utilizando apenas a memória necessária para os elementos inseridos. Além disso, para a manipulação dessa estrutura foi criado um Tad cuja a definição se encontra no arquivo *AVL.h* e seu código fonte no arquivo *AVL.c*, sendo essas funções:

AVL* avl_criar(): Função responsável pela criação da árvore, alocando e retornando o ponteiro para árvore nova.

Boolean avl_apagar(AVL *arvore): Função responsável por liberar toda memória utilizada na árvore, dos elementos a árvore em si.

Boolean avl_inserir(AVL *T, ITEM item): Função responsável por inserir um novo elemento (do tipo ITEM) na árvore já criada, irá realizar as rotações de balanceamento quando necessário.

Boolean avl_remover(AVL *T, char *chave): Função responsável por remover uma elemento da árvore, também realiza rotações quando necessário.

ITEM* avl_buscar(AVL *T, char *chave): Função responsável por buscar uma string dentro da árvore, caso a encontre irá retornar um ponteiro para aquele item, caso contrário retornará um ponteiro para null.

int avl_altura(AVL*T): Função que retorna a altura da árvore inteira.

void avl_imprimir(AVL *T): Função responsável por imprimir a árvore de maneira ordenada.

Foi criado também um segundo tad denominado AVLREC, no qual também foi implementado uma árvore do tipo AVL dessa vez com um item (ITEMREC) composto de duas informações, *palavra* do tipo vetor de char e *recorrencia* do tipo inteiro, a classificação dos itens é feita de acordo com a informação *palavra*.

Para esse tad, além das funções básicas: AVLREC* avlrec_criar(); Boolean avlrec_apagar(AVLREC *arvore); Boolean avlrec_inserir(AVLREC *T, char item[MAXREC]); Boolean avlrec_remover(AVLREC *T, char *chave); ITEMREC* avlrec_buscar(AVLREC *T, char *chave); int avlrec_altura(AVLREC*T) e avlrec_imprimir(AVLREC *T), que possuem o mesmo funcionamento que o anterior para o tipo de dado definido. Foi criada a função ITEMREC* avlrec_toarray(AVLREC *arvore, int *tam), responsável por transformar a árvore em um vetor ordenado quando necessário.

No arquivo principal do trabalho *main.c* foi implementada a lógica do algoritmo, que será explicada na próxima seção, e algumas funções auxiliares.

Lógica do Algoritmo:

Primeiramente, é definida a struct DICIONARIO que será usada para armazenar a avl que contêm as palavras do dicionário e o seu id. Também é declarada a função comparadora que será usada em um qsort na operação 4 para ordenar o array de palavras recorrentes que foram encontradas no dicionário de acordo com a sua recorrência e em ordem alfabética.

Então, são declaradas todas as variáveis que serão usadas no programa e é iniciado o while loop que continuamente recebe o input da operação desejada para então, por meio de um switch case, redirecioná-lo para o bloco de código correspondente à operação desejada. O default do switch case encerra o loop e

desaloca todas as variáveis, encerrando o programa.

Operação 1:

Primeiro, a flag “funcionou” e definida como FALSE por padrão, então, inicia-se um for loop que itera pelo array de ponteiros para dicionários checando se alguma posição do array está “livre”, ou seja, algum dos ponteiros está apontando para NULL.

Caso seja encontrado alguma posição livre, é alocado um dicionário nesta posição e seu id é definido como o índice do array + 1, (o “+1” é necessário pois os índices do array iniciam no 0 e os ids dos dicionários iniciam no 1), Então, inicia-se um while loop onde são escaneadas e inseridas as palavras na avl do dicionário, uma por uma, até que seja digitado um “#”. Depois, a flag “funcionou” é definida como TRUE.

Caso nenhuma posição do dicionário esteja livre, a flag continua como FALSE e nenhum dicionário é alocado.

No final, temos um if que checa a flag “funcionou” para ver se um dicionário foi alocado, caso a flag seja TRUE, é impresso “DICIONARIO <índice do dicionário> CRIADO” caso a flag seja false, são escaneadas as palavras que deveriam ser inseridas no dicionário e descartadas, para então ser impresso IMPOSSIVEL CRIAR. Finalmente, é dado um break e reinicia-se o while desde o começo.

Operação 2:

Primeiro é escaneado o id do dicionário e deste subtraído 1 para corresponder com o seu índice no array, então, um condicional checa se esse dicionário já foi alocado ou se ele se encontra desalocado (ponteiro apontando para NULL).

Caso o dicionário exista, são escaneadas 2 variáveis, uma depois da outra, em um while loop. A primeira, “char_flag” é uma string que tem função dupla, ela indica se foi digitado o “#” que encerra o while loop e, caso não seja digitado o “#”, ela é convertida para inteiro por meio da função “atoi()” da biblioteca string.h, que representará a operação a ser feita no dicionário(0 para deleção e 1 para inserção). A segunda variável representa a palavra com a qual será feita a operação e só é escaneada se a primeira variável não for “#”.

Logo após, entra-se em um condicional, que checa a primeira variável para

determinar qual operação deverá ser feita no dicionário.

Caso a operação seja deletar, a flag “funcionou” recebe o output da função do TAD "avl_remover", com a segunda variável que corresponde a palavra a ser removida e a avl do dicionário como argumentos. A função retorna TRUE se a palavra estiver presente na avl e FALSE se a palavra não estiver presente na avl. Então, a partir da flag “funcionou” determina-se se a operação foi bem sucedida e imprime-se “<palavra> EXCLUIDA DE <índice do dicionário>" caso a flag seja TRUE e "<palavra> INEXISTENTE EM <índice do dicionário>" caso a flag seja FALSE.

Caso a operação seja inserir, a flag funcionou recebe o output da função do TAD "avl_inserir", com a segunda variável que corresponde a palavra a ser inserida e a avl do dicionário como argumentos. A função retorna TRUE se a palavra não estiver presente na avl e FALSE se a palavra estiver presente na avl. Então, a partir da flag “funcionou” determina-se se a operação foi bem sucedida e imprime-se “<palavra> INSERIDA EM<índice do dicionário>" caso a flag seja TRUE e "<palavra> JA EXISTE EM <índice do dicionário>" caso a flag seja FALSE.

Após as operações, é dado um break saindo da operação e retornando para o início do while.

Caso o dicionário não exista, são escaneadas as palavras da mesma forma, encerrando após ser inserido o # mas não é feito nada com elas e elas são descartadas. Então, é impresso “DICIONARIO <índice do dicionário> INEXISTENTE” depois, e dado um break e e encerrada a operação, voltando para o começo do while.

Operação 3:

Primeiramente, é escaneado o id do dicionário e então é subtraído 1 dele para corresponder ao índice do dicionário no array. Então, um condicional checa se o dicionário já foi alocado ou se ele ainda não existe.

Se o dicionário existir, e apagada a sua avl pela função do TAD “avl_apagar”, e dado free no dicionário, e seu ponteiro e apontando para NULL. Ademais, é impresso “DICIONARIO <índice do dicionário> APAGADO” e declarado um break, saindo da operação e retornando para o começo do while.

Se o dicionário não existir, apenas é impresso “DICIONARIO <índice do dicionário> INEXISTENTE “ e declarado um break, encerrando a operação.

Operação 4:

Primeiro, é escaneado o índice do dicionário desejado e o número de palavras recorrentes a serem impressas. Então, um condicional checa se o dicionário já foi alocado.

Caso o dicionário exista, são criadas 2 avls, uma para as palavras que foram achadas no dicionário e suas recorrências (AVLrec) e outra para as palavras que não foram achadas no dicionário (AVLrejects). Então, são escaneadas as palavras, uma por uma até ser digitado o “#” que quebra o while, e conferidas na avl do dicionário por meio da função do TAD “avl_buscar” que retorna NULL se a palavra não esteja presente no dicionário e retorna um ponteiro para o ITEM se ele for achado na avl.

Caso a palavra seja encontrada na avl do dicionário, ela é buscada na avl das palavras recorrentes por meio da função do TAD “avlrec_buscar” que funciona da mesma forma que a “avl_buscar”. Se for encontrada, e somado 1 a sua recorrência, se não for encontrada, ela é inserida na avl das palavras recorrentes.

Caso a palavra não seja encontrada na avl do dicionário, ela apenas é inserida na avl das palavras rejeitadas.

Acabado a inserção, a avl das palavras rejeitadas e impressa por meio da função do tad “avl_imprimir”, já em ordem alfabética, devido ao funcionamento da AVL. Então, a avl das palavras recorrentes e transposta para um array usando a função do TAD “avlrec_to_array”. Então, esse array é ordenado por um qsort usando o comparador “comparador_recorrentes” previamente citado e impresso em ordem até o número de palavras indicado no input caso este seja menor ou igual ao seu número de elementos , se não, ele é impresso por inteiro. Caso o número de palavras indicado seja menor ou igual a zero, apenas e impresso “IMPOSSIVEL INFORMAR <número de palavras> PALAVRAS MAIS FREQUENTES”.

Caso o dicionário não exista, apenas são escaneadas as palavras e é impresso “DICIONARIO INEXISTENTE”. Então, as avls são apagadas e o array recebe free.