Felipe Roza Bonetti (12011BCC032)
Pedro Henrique Marra Araújo (12011BCC008)

Implementação de um Índice Remissivo com implementação de Árvore AVL em C

Uberlândia, Minas Gerais
2022

Sumário

Objetivos Gerais

Esse trabalho teve como objetivo geral a criação de um tipo abstrato de dados de implementação de índices remissivos para a transformação de documentos-texto em seus respectivos índices.

Objetivos Específicos

Esse trabalho teve como objetivo específico a criação de um programa em linguagem C que estrutura o formato de índices remissivos supracitados utilizando árvores AVL, fornecendo funções básicas de criação e impressão destes para o usuário. Para tanto, foram criados os tipos abstratos árvore AVL de duplas palavra-lista e uma lista de inteiros. Além disso, foi criado um menu para a manipulação desses índices (criação por arquivo e inserção unitária de cada palavra).

Repositório

Todos os códigos foram disponibilizados em um repositório no GitHub, que pode ser acessado clicando aqui. Nele, encontra-se, também, um executável com um menu de aplicação do código.

TADs

A seguir, uma breve documentação de todos os TADs criados para o trabalho, incluindo o prório TAD do grafo em si.

Lista de Inteiros

Dados: define uma estrutura (`struct noLista`) sendo o nó da lista (com `int` e `struct noLista*`). Utilizou-se a implementação de uma lista com cabeçalho, pois é interessante termos o tamanho da lista em tempo constante ($\mathcal{O}(1)$) (nesse caso, esse tamanho será a quantidade de palavras inseridas, contabilizando mesmo aquelas que não entraram por já pertencerem à lista). Definiu-se, também, (`struct noLista*`) como (`Lista`).

Consistência dos Dados: nessa lista, é possível a inserção somente de qualquer valor inteiro maior que zero, pois é uma lista de inteiros referentes às páginas de cada palavra entrada na árvore AVL. Além disso, é uma lista ordenada crescente, sem elementos duplicados e o tamanho (guardado no cabeçalho) é referente à quantidade de elementos entrados (contabilizando as duplicatas de inteiros). Não há remoção de elementos, pois segue a desnecessidade de remoção do TAD da Árvore AVL.

- `cria_lista`

  Entrada: um endereço de lista.

  Processo: cria uma lista na condição de vazia.

  Saída: 1, se sucesso. 0, caso contrário.

- `lista_vazia`

  Entrada: uma lista válida.

  Processo: verifica se a lista está vazia.

  Saída: 1, se válida e vazia. 0, caso contrário.

- `tamanho_lista`

  Entrada: uma lista e um endereço de inteiro.

  Processo: retorna ao endereço o tamanho da lista.

  Saída: 1, se lista e endereço válidos. 0, caso contrário.

- `insere_elem`

Entrada: uma lista e um inteiro.

Processo: insere o inteiro na lista de forma crescente e sem duplicatas. De qualquer forma, acresce o tamanho da lista (guardada no cabeçalho).

Saída: 1, se lista válida e inteiro maior que zero. 0, caso contrário.

- `pertence`

  Entrada: uma lista e um inteiro.

  Processo: verifica se o inteiro pertence à lista.

  Saída: 1, se lista válida e inteiro pertencente à lista. 0, caso contrário.

- `imprimir_lista`

  Entrada: uma lista e um ponteiro para um arquivo.

  Processo: imprime a representação da lista no arquivo apontado.

  Saída: 1, caso lista e ponteiro válidos. 0, caso contrário.

- `apaga_lista`

  Entrada: um endereço de lista.

  Processo: apaga a lista.

  Saída: 1, se endereço válido. 0, caso contrário.

Árvore AVL de Entradas

Dados: define uma estrutura (`struct noArv`) consistindo de uma palavra (`char pal[]`), dois ponteiros para as subárvores da esquerda e direita (`struct noArv*`), uma (`Lista linhas`) lista de inteiros consistindo nas linhas da palavra do nó e a altura do nó em relação à raiz (`int altura`). Define-se, também, (`Arv`) como o ponteiro para esse nó de árvore (`struct noArv*`), que a partir daqui referiremos como a árvore em si. Por fim, define-se (`struct item`) como uma entrada do texto (a dupla palavra-linha) e (`Item`) como o ponteiro para essa estrutura, o qual a partir daqui referiremos como o item em si.

- `cria_arv_vazia`

  Entrada: uma árvore.

  Processo: cria uma árvore na condição de vazia.

  Saída: 1, se sucesso. 0, caso contrário.

- `cria_arv`

  Entrada: um endereço de árvore, duas árvores e um item.

  Processo: cria uma árvore com o item na raiz e como subárvores da raiz as duas árvores entradas.

  Saída: 1, se sucesso ao criar. 0, caso contrário.

- `arv_vazia`

  Entrada: uma árvore.

  Processo: verifica se a árvore está vazia.

  Saída: 1, se vazia. 0, caso contrário.

- `arv_altura`

  Entrada: uma árvore.

  Processo: acha a altura da árvore.

  Saída: a altura da árvore.

- `arv_tamanho`

Entrada: uma árvore.

Processo: contabiliza a quantidade de nós na árvore.

Saída: a quantidade de nós encontrados.

- `arv_palavra_pertence`

  Entrada: uma árvore, uma palavra e um endereço de árvore.

  Processo: verifica se a palavra encontra-se na árvore e endereça o nó da possível palavra encontrada no endereço entrado.

  Saída: 1, se encontrada. 0, caso contrário.

- `arv_item_pertence`

  Entrada: uma árvore, um item e um endereço de árvore.

  Processo: verifica se o item encontra-se na árvore e endereça o nó do possível item encontrado no endereço entrado.

  Saída: 1, se encontrado. 0, caso contrário.

- `insere_arv`

  Entrada: um endereço de árvore e um item.

  Processo: insere o item na árvore.

  Saída: 1, se inserido com sucesso. 0, caso contrário.

- `esvazia_arv`

  Entrada: um endereço de árvore.

  Processo: deixa a árvore na condição de vazia.

  Saída: nada.

- `apaga_arv`

  Entrada: um endere;o de árvore.

  Processo: apaga a árvore.

  Saída: nada.

- `imprimir_indice`

Entrada: uma lista, um ponteiro para um arquivo e um tempo da biblioteca `time.h`.

Processo: imprime a representação da árvore no formato de um índice remissivo (com quantidade de palavras, a quantidade de palavras distintas e tempo e eventual tempo que foi necessário para a construção da árvore).

Saída: 1, se sucesso ao imprimir. 0, caso contrário.

Comentário sobre os Algoritmos

As funcionalidades solicitadas foram implementadas pelas seguintes funções:

(a) *" [...] cada termo encontrado no documento teria uma lista de linhas em que foi encontrado"*: utilizamos a lista de inteiros com cabeçalho pela eficiência em buscar seu tamanho, como já comentado.

(b) *" [...] desconsideradas diferenças entre letras maiúsculas e minúsculas"*: preferimos deixar a rigor do usuário (ou seja, da `main`) de tratar a entrada, ou seja, colocar todas as letras em minúsculas, para que o TAD da Árvore AVL ficasse o mais genérico possível. Pois então, para tratar as palavras com letras maiúsculas, criamos a função `to_lower` que, dado uma palavra, coloca ela em minúsculo.

(c) *" [...] se a palavra aparece mais de uma vez numa determinada linha, esta linha aparece apenas uma vez no índice"*: para isso, mantemos a lista de inteiros sem duplicatas (pois, assim, ao imprimir, não teríamos uma mesma linha repetida).

(d) *"O índice deve ser retornado em um arquivo texto (.txt) com as três linhas finais (total de palavras, total de palavras distintas, tempo de construção), e o documento de entrada também deve ser do tipo texto simples (.txt)."*: a partir do nosso menu, caso escolha a opção `2`, será pedido o diretório de um arquivo para a leitura; para a impressão do índice, tanto é possível imprimir na tela (opção `4`), quanto em no arquivo `indice.txt` (opção `5`). Verifique-se que não há nenhum índice em utilização (caso tenha, destrua-o pela opção `7` antes de construir outro).

(e) *" [...] três linhas finais (total de palavras, total de palavras distintas, tempo de construção)"*: o total de palavras distintas é a quantidade de nós encontrados na árvore. Assim, na própria impressão, enquanto ela imprime em-ordem (para termos a ordem alfabética), ela própria vai acrescendo a quantidade de nós encontrados. Analogamente, a quantidade total de palavras é a soma de todos os tamanhos de todas as listas encontradas nos nós, que a própria função de impressão já acresce. No final, como foram guardados esses valores, conseguimos facilmente imprimi-los. Já o tempo de construção, decidimos por utilizar a função `clock` da biblioteca `time.h`, por ser universal para todas as plataformas.

Infelizmente, um ponto ruim dela é a impossibilidade de contabilizar algo mais preciso do que só segundos (como milissegundos).

Abaixo, segue as principais funções utilizadas para a implementação desse índice remissivo.

`insere_elem`:

A função necessita das funções auxiliares de rotação (semelhantes às mostradas em sala de aula) e de uma função auxiliar `insere_arv_` para que sua assinatura pudesse permanecer retornando a validade da inserção do elemento na árvore. Aqui, colocaremos só essas duas funções principais de inserção.

```
int insere_arv(Arv* arv, Item item){
    if(arv == NULL || item == NULL) return 0;

    *arv = insere_arv_(*arv, item);
    if(*arv == NULL) return 0;
    return 1;
}

Arv insere_arv_(Arv arv, Item item) {
    if (arv == NULL){
        Arv novo = (Arv)malloc(sizeof(struct noArv));
        if(novo == NULL) return novo;

        strcpy(novo->pal, item->pal);
        if(!cria_lista(&novo->linhas) ||
        !insere_elem(novo->linhas, item->linha)){
            free(novo);
            return NULL;
        }
        novo->esq = NULL;
        novo->dir = NULL;
        novo->altura = 1;
        return novo;
    }
```

```
    if(strcoll(item->pal, arv->pal) < 0)
        arv->esq = insere_arv_(arv->esq, item);
    else if(strcoll(item->pal, arv->pal) > 0)
        arv->dir = insere_arv_(arv->dir, item);
    else{
        insere_elem(arv->linhas, item->linha);
        return arv;
    }

    arv->altura = 1 + max(arv_altura(arv->esq),
    arv_altura(arv->dir));

    int balanceamento = fator_balanceamento(arv);
    if (balanceamento > 1 &&
    strcoll(item->pal, arv->esq->pal) < 0)
        return rotacao_direita(arv);

    if (balanceamento < -1 &&
    strcoll(item->pal, arv->dir->pal) > 0)
        return rotacao_esquerda(arv);

    if (balanceamento > 1 &&
    strcoll(item->pal, arv->esq->pal) > 0) {
        arv->esq = rotacao_esquerda(arv->esq);
        return rotacao_direita(arv);
    }

    if (balanceamento < -1 &&
    strcoll(item->pal, arv->dir->pal) < 0) {
        arv->dir = rotacao_direita(arv->dir);
        return rotacao_esquerda(arv);
    }

    return arv;
}
```

imprimir_indice:

Analogamente, temos uma função auxiliar para conseguir imprimir os metadados do índice. A impressão do índice em si (ou seja, das palavras e suas páginas) acontece na função recursiva `imprimir_indice_`, que é, em suma, uma impressão em-ordem.

```
int imprimir_indice(Arv arv, FILE* fp, clock_t tempo){
    if(fp == NULL) return 0;

    fprintf(fp, "Índice:\n\n");
    int p = 0, pd = 0;
    imprimir_indice_(arv, fp, &p, &pd);
    fprintf(fp, "\nNúmero total de palavras: %d.\n", p);
    fprintf(fp, "Numero de palavras distintas: %d.\n", pd);
    if(tempo >= 0)
        fprintf(fp, "Tempo de construção do índice usando
        árvore AVL: %lds.", tempo);
}

void imprimir_indice_(Arv arv, FILE* fp, int* p, int* pd){
    if(!(arv_vazia(arv))){
        int t, *li, i;
        setlocale(LC_ALL, "Portuguese_Brasil");

        (*pd)++;
        tamanho_lista(arv->linhas, &t);
        (*p) += t;

        imprimir_indice_(arv->esq, fp, p, pd);
        fprintf(fp, "%s ", arv->pal);
        imprimir_lista(arv->linhas, fp);
        imprimir_indice_(arv->dir, fp, p, pd);
    }
}
```

Outras funções interessantes do TAD da Árvore AVL são as seguintes.

esvazia_arv:

```
void esvazia_arv(Arv* arv){
    if(arv == NULL || *arv == NULL) return;

    esvazia_arv(&(*arv)->esq);
    esvazia_arv(&(*arv)->dir);
    free(*arv);
    *arv = NULL;
}
```

rotacao_esquerda:

```
Arv rotacao_esquerda(Arv arv) {
    Arv x = arv->dir;
    Arv y = x->esq;

    x->esq = arv;
    arv->dir = y;

    arv->altura = max(arv_altura(arv->esq),
    arv_altura(arv->dir)) + 1;
    x->altura = max(arv_altura(x->esq),
    arv_altura(x->dir)) + 1;

    return x;
}
```

rotacao_direita:

```
Arv rotacao_direita(Arv arv) {
    Arv x = arv->esq;
    Arv y = x->dir;
```

```
    x->dir = arv;
    arv->esq = y;

    arv->altura = max(arv_altura(arv->esq),
    arv_altura(arv->dir)) + 1;
    x->altura = max(arv_altura(x->esq),
    arv_altura(x->dir)) + 1;

    return x;
}
```

Testando as Funções

Segue abaixo três poemas que utilizaremos como textos para o teste da criação dos índices remissivos. Para utilizá-los, execute `main.exe` do repositório do GitHub e use a opção `2`, digite o nome de determinado arquivo (verifique se não há nenhum outro índice em utilização) e descarregue o índice usando opção `5`.

Pensa em ti mesma, acharás
Melhor poesia,
Viveza, graça, alegria,
Doçura e paz.

Se já dei flores um dia,
Quando rapaz,
As que ora dou têm assaz
Melancolia.

Uma só das horas tuas
Valem um mês
Das almas já ressequidas.

Os sóis e as luas
Creio bem que Deus os fez
Para outras vidas.

Figura 1: *"Machado de Assis: A uma senhora que me pediu versos"*, disponibilizado em `1.txt`.

Mas há a vida
que é para ser
intensamente vivida, há o amor.

Que tem que ser vivido
até a última gota.
Sem nenhum medo.
Não mata.

Figura 2: *"Clarice Lispector: Mas há a vida"*, disponibilizado em `2.txt`.

Se sou amado
Quanto mais amado
Mais correspondo ao amor.
Se sou esquecido
Devo esquecer também
Pois amor é feito espelho
Tem que ter reflexo.

Figura 3: *"Paulo Neruda: Reflexo"*, disponibilizado em `3.txt`.

No meio do caminho tinha uma pedra
tinha uma pedra no meio do caminho
tinha uma pedra
no meio do caminho tinha uma pedra.

Nunca me esquecerei desse acontecimento
na vida de minhas retinas tão fatigadas.
Nunca me esquecerei que no meio do caminho
tinha uma pedra
tinha uma pedra no meio do caminho
no meio do caminho tinha uma pedra.

Figura 4: *"Carlos Drummond de Andrade: No Meio do Caminho"*, disponibilizado em `4.txt`.

Entrando o texto 1 (`1.txt`), é possível visualizar o arquivo resultado (`indice1.txt`) clicando aqui.

Entrando o texto 2 (`2.txt`), é possível visualizar o arquivo resultado (`indice2.txt`) clicando aqui.

Entrando o texto 3 (`3.txt`), é possível visualizar o arquivo resultado (`indice3.txt`) clicando aqui.

Entrando o texto 4 (`4.txt`), é possível visualizar o arquivo resultado (`indice4.txt`) clicando aqui.