Instituto de Computação
Departamento de Ciência da Computação
Universidade Federal da Bahia (UFBA) - Salvador, BA - Brasil

MATA54 - Estrutura de Dados II
Prof. Flávio Morais de Assis Silva

Alunos grupo 1: João Lucas Lima de Melo (joaollm@ufba.br); Luca Assis Argolo (luca.argolo@ufba.br)

Relatório de Implementação Hashing Perfeito em Dois Níveis

Sumário.

Introdução.

O quarto trabalho prático da disciplina Estrutura de Dados e Algoritmos II propõe, como visto em sala, a implementação de um algoritmo de reconhecimento de padrão de texto sobre a aplicação desenvolvida no terceiro trabalho. Nos baseamos nos conteúdos abordados em aula e no livro *Algoritmos: Teoria e Prática* para o desenvolvimento do projeto.

Não houve a necessidade de implementação de novas estruturas, uma vez que já constava no terceiro trabalho todas as funcionalidades da *árvore k-d* necessárias para esta etapa. Bastou, portanto, desenvolver apenas as funções solicitadas pela especificação e o algoritmo de reconhecimento de padrão de texto.

Referente à base do trabalho atual, a *árvore k-d* implementada na terceira etapa, foram implementados três componentes: Um arquivo contendo o índice da árvore, um arquivo auxiliar que recebe os registros usados para a construção do índice, e os arquivos representando as diferentes páginas.

O arquivo do índice recebe dois diferentes tipos de dados, NoAno (contendo um valor inteiro *ano* e booleanos *esq* e *dir*), e NoNome (contendo um valor de string *autor* e booleanos *esq* e *dir*). A cada dado recebido para a construção do índice, uma estrutura correspondente à chave da profundidade, nome do autor ou ano de publicação, será criada e inserida no arquivo até o fim da leitura dos registros.

Enquanto os registros são recebidos para a construção do índice, eles são armazenados em um arquivo *reg.dat*. Após a construção do índice, tendo acesso aos registros usados para a estrutura da árvore, o programa itera pelos livros em *reg.dat* alocando-os nas suas devidas páginas.

As páginas são criadas em função da iteração dos registros pelo índice que elas guardam. Um livro que, comparado com a raiz, foi alocado à direita, comparado e alocado à esquerda, comparado novamente e, por fim, alocado à esquerda será armazenado no arquivo *dee.dat*.

Os registros e células sempre foram manipulados por meio do uso de funções gerenciadoras de arquivos, não havendo alocação em memória principal. Os arquivos foram escritos de tal forma que possam ser consultados uma vez escritos e o programa ter sido finalizado.

Uma vez desenvolvida a base do programa, utilizando uma *árvore k-d* e organizando os registros em páginas, desenvolvemos a seção da aplicação responsável pelo reconhecimento de padrão de texto. O algoritmo desenvolvido foi o KMP, dividido entre a função de criação da *tabela pi* e a de *reconhecimento*.

A aplicação, quando solicitada a operação de busca por uma palavra no texto, itera sobre o índice (criado na terceira etapa do trabalho), acessa todas as páginas existentes, consulta o nome do arquivo .txt que consta nos registros para que então possa acessá-lo. A busca por uma palavra em um arquivo específico ou por todos depende da operação solicitada pelo usuário.

A operação de cálculo da *tabela pi*, pré-processamento de padrão, itera exclusivamente sobre a palavra. Dessa forma, não há necessidade de consulta e acessos ao índice, páginas ou arquivos.

Estruturas da Árvore.

Decidimos criar três principais estruturas que serão iteradas, inseridas e manipuladas pela aplicação: *Livro, NoAno, NoAutor*.

NoAno e NoAutor: Estruturas usadas na criação do índice. Um comporta um valor inteiro *ano*, e o outro um astring *autor*. Ambos possuem dois booleanos *esq* e *dir*, referentes ao nó à esquerda e direita do dado.

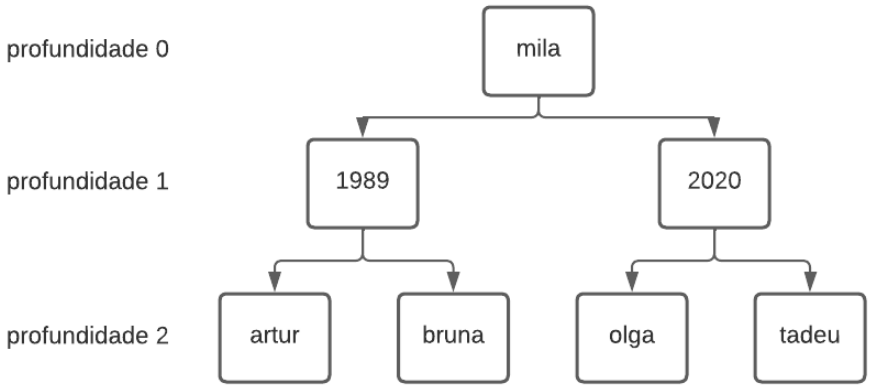

Figura 1.

Tomando como exemplo o índice representado na figura 1. A imagem ilustra uma instancialização de um índice comportando 7 diferentes dados, utilizando um inteiro e uma string como chaves para a construção da estrutura. Cada nó seria representado por um *NoAno*, em profundidades ímpares, ou *NoAutor*, em profundidades pares.

Livro: Estrutura utilizada para receber dados da aplicação, que serão organizados e armazenados em páginas. Recebe um valor inteiro *ano_publicacao* e strings de 20 caracteres *autor, titulo* e *nome_arquivo*. Os valores dos livros serão usados para a construção do índice, mas serão apenas armazenados em páginas.

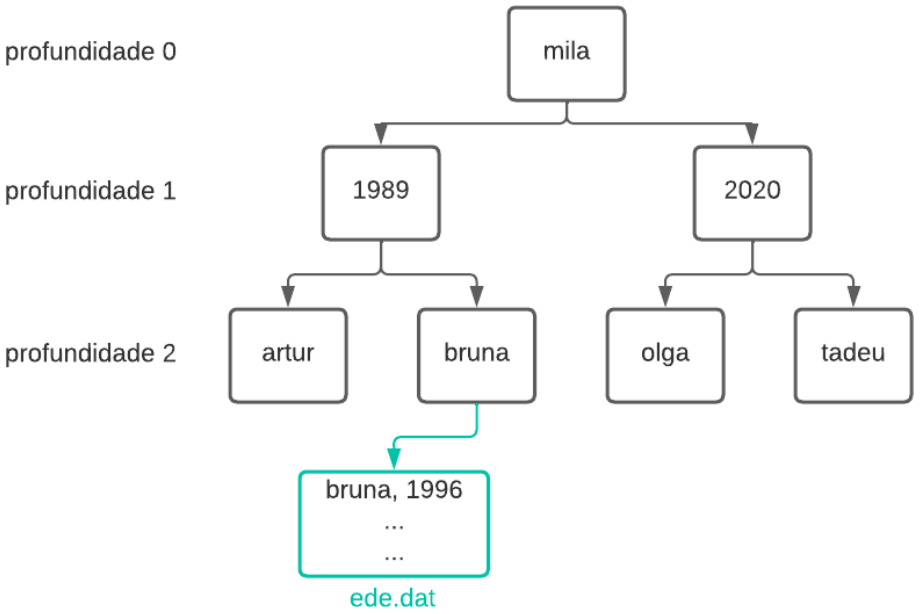

Figura 2.

Tomando como exemplo a figura 2 e o livro que contém os valores de *nome_autor* e *ano_publicacao* atribuídos como Bruna e 1996. O livro terá seus valores comparados com cada nó da árvore, deslocando à esquerda de *mila*, à direita de *1989* e esqueda de *bruna,* sendo alocado no arquivo *ed.dat*.

Reconhecimento de Padrões.
Não houve a necessidade de criação de estruturas para reconhecimento de padrões de texto, já que o índice da árvore, os registros, páginas e arquivos de texto existem e estão devidamente organizados no momento em que as operações de busca por palavras são solicitadas pelo usuário.

Desenvolvemos, portanto, apenas o algoritmo de reconhecimento de padrão de texto. Como solicitado, implementamos o algoritmo KMP, dividido entre a função de criação da *tabela pi* e a de *reconhecimento*.

A construção da *tabela pi*, dada pela função *constroiTabelaPi*, itera exclusivamente sobre uma palavra, não havendo necessidade de iteração sobre o índice, páginas ou registros. Para cada caractere de uma palavra, o algoritmo atribui um valor, referente ao resultado do processamento de "extensão" do maior prefixo possível comparado com o sufixo.

A busca de uma palavra em um texto ocorre por meio da função de reconhecimento *buscaKMP*. A sequência de caracteres a ser buscada e o nome do arquivo são recebidos pela função, que abrirá o arquivo solicitado, invocará a função de criação da *tabela pi* e, após esse pré-processamento da palavra, iterará sobre todos os caracteres do arquivo buscando pelo reconhecimento do padrão de caracteres no texto, informando a linha e posição no arquivo onde a palavra ocorre.

Estruturação do Código e Compilação.
Para melhor organização do código, separamos nossa implementação em múltiplos arquivos:

main.c: Contém a lógica de entrada básica e chama as outras funções a depender da operação escolhida.

registros.h: Define as estruturas utilizadas na aplicação. As estruturas utilizadas na criação do índice *NoAno* e *NoAutor*, contendo booleanos *esq* e *dir* (sinalizando a existência ou não de um nó à esquerda ou direita) e armazenando os valores chaves inteiro *ano* e string *autor*, respectivamente. É definido ainda a estrutura principal pela qual o programa iterará, *Livro*, contendo um inteiro *ano_publicacao*, e strings *autor, titulo* e *nome_arquivo*.

constroiArovre.c: Define a função responsável por criar o índice da aplicação, uma árvore k-d de duas dimensões que itera sobre *nome de autor* nos índices pares e *ano de publicação* em

índices ímpares. O índice é construído utilizando estruturas do tipo *NoAno* e *NoAutor*, armazenado em um arquivo *arvore.dat.*
constroiArvore.h: Contém as declarações das funções implementadas em *constroiArovre.c.*

consultaFaixaAnos.c: Define a função responsável por realizar a consulta de um conjunto de registros por uma determinada faixa de valores. É dado o piso e teto (inclusivo) do intervalo buscado e é consultado, a partir do índice, todos os livros guardados em páginas cujo valor de *ano de publicação* esteja contido no intervalo.
consultaFaixaAnos.h: Contém as declarações das funções implementadas em *consultaFaixaAnos.c.*

consultaFaixaAutores.c: Define a função responsável por realizar a consulta de um conjunto de registros por uma determinada faixa de valores. É dado o piso e teto (inclusivo) do intervalo buscado e é consultado, a partir do índice, todos os livros guardados em páginas cujo valor de *nome de autor* esteja contido no intervalo.
consultaFaixaAutores.h: Contém as declarações das funções implementadas em *consultaFaixaAutores.c.*

consultaFaixaAutoresAnos.c: Define a função responsável por realizar a consulta de um conjunto de registros por uma determinada faixa de valores. É dado o piso e teto (inclusivo) de intervalos para *nome de autor* e *ano de publicação*. É buscado e consultado, a partir do índice, todos os livros guardados em páginas cujo valores estejam contidos no intervalo.
consultaFaixaAutoresAnos.h: Contém as declarações das funções implementadas em *consultaFaixaAutoresAnos.c.*

consultaSimples.c: Define a função responsável por realizar a consulta de um livro armazenado em página em função do valor do nome do autor. A busca ocorre iterando-se pelos índices da árvore k-d, consultando a existência do livro na sua devida página (em função da sua posição na estrutura).
consultaSimples.h: Contém as declarações das funções implementadas em *consultaSimples.c.*

imprimeIndiceArvore.c: Define a função responsável por iterar sobre o índice da árvore, in-order, imprimindo todos os nós encontrados, sinalizando seus elementos à esquerda, direita e a possível existência de página.
imprimeIndiceArvore.h: Contém as declarações das funções implementadas em *imprimeIndiceArvore.c.*

imprimePagina.c: Define a função responsável por iterar sobre o índice da árvore, in-order, consultando diferentes páginas existentes e imprimindo os valores dos registros armazenados.
imprimePagina.h: Contém as declarações das funções implementadas em *imprimePagina.c.*

inicializaPaginas.c: Define a função responsável por, após a construção do índice, inserir os livros utilizados em sua construção nas suas devidas páginas.

inicializaPaginas.h: Contém as declarações das funções implementadas em *inicializaPaginas.c.*

insereRegistros.c: Define a função responsável por receber os valores de um livro, instancializar um livro, atribuir a ele os valores recebidos e inseri-lo em uma página.
insereRegistros.h: Contém as declarações das funções implementadas em *insereRegistros.c.*

tabelaPiPalavra.c: Define a função responsável por receber a palavra cuja *tabela pi* deve ser criada. Invoca a função *constroiTabelaPi* para o processamento da tabela.
tabelaPiPalavra.h: Contém as declarações das funções implementadas em *tabelaPiPalavra.c.*

consultaPalavraArvore.c: Define as funções responsáveis pela consulta de uma palavra em todos os arquivos existentes. A consulta ocorre iterando sobre o índice da árvore, acessando todas as páginas e abrindo os arquivos referidos pelos registros armazenados. Uma vez aberto o arquivo de texto, a palavra é buscada através do algoritmo KMP.
consultaPalavraArvore.h: Contém as declarações das funções implementadas em *consultaPalavraArvore.c.*

consultaPalavraAutorTitulo.c: Define as funções responsáveis pela consulta de uma palavra em um arquivo cujo nome do autor e título da obra corresponda com a entrada do usuário. A consulta ocorre iterando sobre o índice da árvore e buscando nas páginas pelo registro correspondente com a entrada do usuário. Uma vez encontrado, a aplicação tenta abrir o arquivo referido pelo registro. Uma vez aberto o arquivo de texto, a palavra é buscada através do algoritmo KMP.
consultaPalavraAutorTitulo.h: Contém as declarações das funções implementadas em *consultaPalavraAutorTitulo.c.*

utils.c: Define a função responsável por realizar o cálculo de potência, reconhecimento de padrão de texto pelo algoritmo KMP e construção da *tabela pi*.
utils.h: Contém as declarações das funções implementadas em *utils.c.*

Para compilar nosso código, utilizamos o *gcc* na versão *gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0* e executamos o seguinte comando:

```
gcc main.c utils.c constroiArvore.c inicializaPaginas.c insereRegistro.c
consultaSimples.c consultaFaixaAutores.c consultaFaixaAnos.c
consultaFaixaAutoresAnos.c imprimeIndiceArvore.c imprimePagina.c
consultaPalavraArvore.c consultaPalavraAutorTitulo.c tabelaPiPalavra.c -lm
```