Busca Binária

Murielly Oliveira Nascimento

Universidade Federal de Uberlândia
Faculdade de Computação
Prática de Algoritmos e Estrutura de Dados 2.

Uberlândia
2022

Murielly Oliveira Nascimento

Busca Binária

Prática para aprendizado de busca binária feita durante a matéria de Algoritmos e Estruturas de Dados 2 em 2022/1.

Área de concentração: Sistemas de Informação

Uberlândia
2022

Resumo

Descrição: Suponha um arquivo contendo entradas desordenadas para um dicionário da língua portuguesa seguindo o formato: termo - classificação - significado

O exercício prático consiste na implementação de um programa em linguagem C capaz de:

1. Ler o arquivo texto, e preencher um vetor contendo estruturas de dados baseada em um TAD Dicionário.

2. Ordenar o vetor por termo utilizando um método de ordenação de bom desempenho.

3. Permitir pesquisa por termo baseada em busca binária, implementada de forma recursiva. Para cada termo encontrado, deve-se exibir na tela a respectiva entrada completa: termo, classificação e significado.

Palavras-chave: Busca Binária.

Capítulo 1

Solução

Foi usada a linguagem C para a solução do problema. O código e é divido em arquivos para as TADs e para a função main. A estrutura DICIONARIO é armazenada da seguinte forma no arquivo dicionario.h.

```
#ifndef __APRENDIZADO_H_INCLUDED__
#define __APRENDIZADO_H_INCLUDED__

#define MAX 100

typedef struct
{
    char termo[MAX];
    char classificao[MAX];
    char significado[MAX];
} DICIONARIO;

#endif
```

Como definido no enunciado, os dados são recebidos do arquivo dados.txt e o TAD File é responsável por tratá-los e armazená-los num vetor da estrutura DICIONARIO.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "TAD_File.h"

DICIONARIO *open_txt(char *nome, int *tamanho)
{
    // O arquivo txt ja deve existir e portanto faremos apenas uma
   leitura
    FILE *dados = fopen(nome, "r");

    // Descobrimos quantas linhas e colunas o arquivo tem
    char caracter = '\0';
```

```
14      int i = 0, j = 0;
15
16      while (caracter != EOF)
17      {
18          caracter = fgetc(dados);
19          if (caracter == '\n')
20              i++;
21      }
22
23      // Voltamos o ponteiro para o inicio
24      rewind(dados);
25
26      // Linhas eh a quantidade de '\n' no arquivo
27      int linhas = i;
28      *tamanho = (linhas + 1) / 3;
29
30      // Criamos o vetor com os dados de aprendizagem
31      DICIONARIO *vetor = malloc((*tamanho) * sizeof(DICIONARIO));
32
33      if (vetor == NULL)
34          return NULL;
35
36      char aux[MAX] = {'\0'};
37
38      // Lemos os dados do arquivo e armazenamos no vetor
39      for (int i = 0; i < (*tamanho); i++)
40      {
41          fgets(aux, MAX, dados);
42          aux[strcspn(aux, "\n")] = 0; // Remove "\n" da string
43          strcpy(vetor[i].termo, aux);
44
45          fgets(aux, MAX, dados);
46          aux[strcspn(aux, "\n")] = 0;
47          strcpy(vetor[i].classificao, aux);
48
49          fgets(aux, MAX, dados);
50          aux[strcspn(aux, "\n")] = 0;
51          strcpy(vetor[i].significado, aux);
52      }
53
54      // Fechamos o arquivo e liberamos a estrutura que lida com ele
55      fclose(dados);
56
57      // A funcao retorna o vetor preenchido com dados
58      return vetor;
59  }
```

A biblioteca de funções desse TAD foi especificada da seguinte forma.

```c
#include "dicionario.h"

DICIONARIO *open_txt(char *nome, int *tamanho);
```

Usamos a função shellSort para ordenar a estrutura por termos, devido ao fato dela ser grande.

```c
void shellsort(DICIONARIO dados[], int n)
{
    int i = 0, j = 0, h = 1;
    DICIONARIO aux;

    do
    {
        h = h * 3 + 1;
    } while (h < n);

    do
    {
        h /= 3;
        for (i = h; i < n; i++)
        {
            aux = dados[i];
            j = i;
            while (strcmp(dados[j - h].termo, aux.termo) > 0)
            {
                dados[j] = dados[j - h];
                j -= h;
                if (j < h)
                    break;
            }
            dados[j] = aux;
        }
    } while (h != 1);
}
```

Por fim, implementamos a busca binária recursiva para encontrar um termo. Se ele não existir retornamos -1, se sim retornamos a sua posição no vetor dados.

```c
int busca_binaria(DICIONARIO dados[], char elemento[], int esq, int dir)
{
    int meio = 0;
    meio = (esq + dir) / 2;

    if (esq > dir)
        return -1;

    if (strcmp(dados[meio].termo, elemento) == 0)
        return meio;
```

```
11
12      else if (strcmp(dados[meio].termo, elemento) > 0)
13      {
14          dir = meio - 1;
15          busca_binaria(dados, elemento, esq, dir);
16      }
17      else if (strcmp(dados[meio].termo, elemento) < 0)
18      {
19          esq = meio + 1;
20          busca_binaria(dados, elemento, esq, dir);
21      }
22  }
```

A função main executa as demais funções e imprime o elemento da busca se ele for encontrado no vetor. Ao final liberamos a estrutura DICIONARIO.

```
1  int main(void)
2  {
3      // Dados s o passados para a estrutura DICIONARIO
4      char nome[] = "dados.txt"; // nome do arquivo
5      int tamanho = 0;           // tamanho do arquivo
6
7      DICIONARIO *dados = open_txt(nome, &tamanho);
8      DICIONARIO *iterator = dados;
9
10     shellsort(dados, tamanho);
11
12     for (int i = 0; i < tamanho; i++)
13     {
14         printf("%s\n", iterator->termo);
15         iterator++;
16     }
17
18     printf("Digite um termo: ");
19     char elemento[100] = {'\0'};
20     scanf("%s", elemento);
21
22     int pos = busca_binaria(dados, elemento, 0, tamanho - 1);
23     if (pos == -1)
24         printf("Termo n o encontrado\n");
25     else
26     {
27         printf("\nPalavra encontrada!\n");
28         printf("Termo: %s\n", dados[pos].termo);
29         printf("Classifica  o: %s\n", dados[pos].termo);
30         printf("Significado:%s\n", dados[pos].significado);
31     }
32     free(dados);
```

```
33      return 0;
34  }
```