Dicionário de anagramas

Caio Eduardo Theodoro
RA: 2044560

Universidade Tecnológica Federal do Paraná - UTFPR
Campus Campo Mourão
Campo Mourão, Paraná, Brasil

Resumo

Neste relatorio, será documentado e detalhado os processos da implementação do dicionário de anagramas para palavras da lingua portuguesa em C.

1 Funções de cabeçalho :

A Figura 1 mostra todas as funções e a struct usadas na implementação. Nessa seção será descrita o objetivo de casa uma para posteriormente a implementarmos.

```c
typedef struct Dicionario
{
    char *palavra;
    char *palavraOrdenada;

} Dicionario;

int sizeFile();
int converte_letras(char c);
char *ordenaLinha(char palavra[]);
Dicionario **criaDicionario(int size);
void insereDicionario(Dicionario **dicionario);
void ImprimeFaixa(char* chave, Dicionario** dicionario, int meio);
int BuscaBinaria(char* chave, Dicionario** dicionario, int e, int d);
void printDicionario(Dicionario **dicionario, const int size);
char *OrdenaPalavra(Dicionario *dicionario);
Dicionario *inserePalavra(const char palavra[], const unsigned int pos);
void quicksort(Dicionario** dicionario, unsigned int len);
int separa (Dicionario** dicionario, int p, int r);
char *OrdenaChave(const char* chave);
```

Figura 1 – func.h

2 Implementação :

2.0.1 int sizeFile();

int sizeFile(); é usada na função main e retorna o tamanho do arquivo de texto **br.txt**. Essa função é usada posteriormente para criarmos dinamicamente a struct **Dicionario**.

```c
int sizeFile()
{
    FILE *arquivo = fopen("br.txt", "r");
    int lines = 0;
    char palavra[255];
    char *linha;

    while (!feof(arquivo))
    {
        linha = fgets(palavra, 255, arquivo);
        if (linha)
            lines++;
    }
    fclose(arquivo);

    return lines;
}
```

Figura 2 –

A função abre o arquivo com **FILE *arquivo = fopen("br.txt", "r");**, depois no laço **while (!feof(arquivo))** ele lê o arquivo e a cada linha, lines++ é incrementada. depois o arquivo é fechado e lines é retornado.

2.0.2 Dicionario **criaDicionario(int size);

Dicionario **criaDicionario(int size); é usada na função main e cria o ponteiro de ponteiro do vetor dinamico de structs. Nessa vetor será armazenado todas palavras do anagrama, bem como suas versões ordenadas.

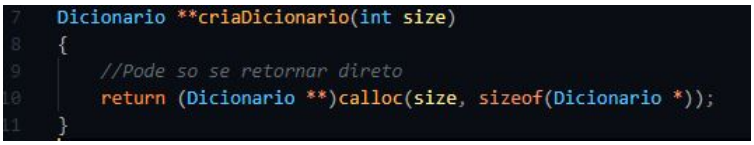

Figura 3 –

A variável int size é o valor retornado da função **int sizeFile();**. A struct é alocada com **calloc** tendo como parâmetro esse mesmo size.

2.0.3 void insereDicionario(Dicionario **dicionario);

insereDicionario(Dicionario **dicionario); é usada na função main e insere o arquivo **br.txt** em todas as **i** posições do dicionario, criando um novo laço para cada linha.

Figura 4 –

O arquivo é aberto com **fopen**, e roda enquanto o arquivo não seja nulo. a função**fgets** pega linha por linha do código e se for diferente de nulo, é chamada a função **InserePalavra**, passando a palavra e a posição a ser inserida. A Figura 5 mostra a função **InserePalavra**.

```c
Dicionario *inserePalavra(const char palavra[], const unsigned int pos)
{

    int c=0;
    char ch;
    int size = strlen(palavra);
    char *temp = calloc(size, sizeof(char));
    strcpy(temp, palavra);

   while (palavra[c] != '\0') {
      ch = temp[c];
      if (ch >= 'A' && ch <= 'Z')
         temp[c] = temp[c] + 32;
      c++;
   }
    temp[size-1] = '\0';

    Dicionario *d = (Dicionario *)malloc(sizeof(Dicionario));
    d->palavra = temp;
    return d;
}
```

Figura 5 –

Na função InserePalavra, primeiro descobrimos o tamanho da linha com **strlen(palavra)**, depois alocamos **temp** com calloc, e então copiamos o conteúdo de palavra para temp usando **strcpy.**

Feito isso, percorremos toda a linha com uma função while, e para cada valor entre A e Z, adicionamos + 32. Essa função é tabelada em ASCII, onde a letra minuscula de uma palavra é sua maiúscula + 32, do mesmo jeito sua maiuscula é a minuscula - 32.

Após isso inserimos ” na posição size-1 para determinar o fim da string. Então, é criado dinamicamente um laço de Dicionario*, que é atribuido a variavel temp contendo a palavra para d->palavra. Como a função inserePalavra retorna um dicionario, é atribuido dicionario na posição i para cada vez que inserePalavra é chamado, sendo assim, a estruturado fica parecida com a Figura 6 em decorrer das iterações.

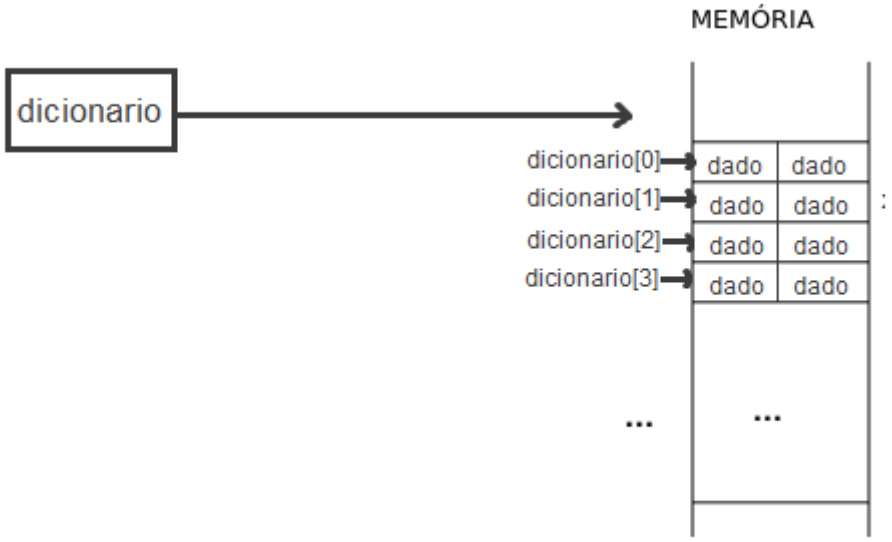

Figura 6 –

Ainda na função **InsereDicionario**, temos a linha **dicionario[i]->palavraOrdenada = OrdenaPalavra(dicionario[i]);**. Essa instruçao chama a função OrdenaPalavra, que retorna uma string. A Figura 7 mostra seu interior.

```
86  char *OrdenaPalavra(Dicionario *dicionario){
87
88      unsigned long size = strlen(dicionario->palavra);
89      char *temp = calloc(size, sizeof(char));
90
91      strcpy(temp, dicionario->palavra);
92      char aux;
93      int n = strlen(temp);
94
95      for (int i = 0; i < n; i++)
96      {
97          for (int j = (i+1) ; j < n; j++)
98          {
99              if ((int)temp[i] > (int)temp[j])
100             {
101                 aux = temp[i];
102                 temp[i] = temp[j];
103                 temp[j] = aux;
104             }
105         }
106     }
107     return temp;
108
109 }
```

Figura 7 –

Basicamente nessa função criamos uma string dinamicamente com o tamanho de dicionario->palavra (**que é a palavra do Dicionario[i] enviado a função**), depois copiamos o conteudo de dicionario->palavra para temp, e então percorremos todos os caracteres da palavra a ordenando pelos valores ASCII de seus caracteres. Por fim a função retorna a string que é atribuida a **dicionario[i]->palavraOrdenada**.

2.0.4 void quicksort(Dicionario** dicionario, unsigned int len);

void quicksort(Dicionario dicionario, unsigned int len);** é usada na função main para ordenar as palavras ordenadas dos vetores de anagrama. No caso, foi usado o algoritmo de quicksort usando o metodo de varredura(swap). A Figura 8 mostra a função de troca usada dentro do quicksort e a Figura 9 mostra o algoritmo de quicksort.

```
202 void troca(char  **arg1, char  **arg2)
203 {
204     char *tmp = *arg1;
205     *arg1 = *arg2;
206     *arg2 = tmp;
207 }
```

Figura 8 –

Primeiramente é verificado se a variavel**len(tamanho do vetor)** é menor ou igual a 1, oque verifica se o vetor já foi percorrido e ordenado.

```
196  void quicksort(Dicionario** dicionario, unsigned int len) //quick sort swap(varredura)
197  {
198      unsigned int i, pivo=0;
199
200      if (len <= 1)
201          return;
202
203      troca(dicionario+((unsigned int)rand() % len), dicionario+len-1);  // troca para um valor qualque
204
205      for (i=0;i<len-1;++i) // reseta a chave pra zero e escaneia
206      {
207          if (strcmp(dicionario[i]->palavraOrdenada, dicionario[len-1]->palavraOrdenada) < 0) // strcmp
208              troca(dicionario+i, dicionario+pivo++);
209      }
210
211      // move a chave para seu lugar
212      troca(dicionario+pivo, dicionario+len-1);
213
214      // invoca a subsequencia recursiva
215      quicksort(dicionario, pivo++);
216      quicksort(dicionario+pivo, len - pivo);
217  }
```

Figura 9 –

Depois temos a função **troca(dicionario+((unsigned int)rand() percent len), dicionario+len-1);** que basicamente chama a função de troca para um valor qualquer menor que o valor de tamanho da função. a struct referenciada é a contento todas as [i] posições, como mostrado na Figura 6.

Depois no ciclo for é resetado a chave e ele escaneia comparando se **strcmp(dicionario[i]->palavraOrdenada, dicionario[len-1]->palavraOrdenada) < 0**, que é 0 caso **dicionario[i]->palavraOrdenada** tenha um valor ASCII menor que **dicionario[len-1]->palavraOrdenada**. Então se troca o pivô com a posição dicionario + i.
Em sequencia, se troca o pivô para posição de origem e então, é invocada a subsequência recursiva do quicksort, fazendo pivô++ e diminuindo pivô do tamanho. Com o quicksort assim, concluimos o requisito maximo de processamento de(n lg(n)).

Usando a função auxiliar para imprimir o vetor, podemos ver o funcionamento do quicksort, como mostra a Figura 10.

```
PS C:\Users\caio-\OneDrive\Área de Trabalho\ana> ./main

palavra 0 : aldebara palavra ordenada 0: aaabdelr

palavra 1 : absalao palavra ordenada 1: aaablos

palavra 2 : abraao palavra ordenada 2: aaabor

palavra 3 : aconcagua palavra ordenada 3: aaaccgnou

palavra 4 : acaia palavra ordenada 4: aaaci

palavra 5 : alsacia palavra ordenada 5: aaacils

palavra 6 : alasca palavra ordenada 6: aaacls

palavra 7 : alexandria palavra ordenada 7: aaadeilnrx

palavra 8 : alexandra palavra ordenada 8: aaadelnrx

palavra 9 : adriana palavra ordenada 9: aaadinr

palavra 10 : afeganistao palavra ordenada 10: aaaefginost
```

Figura 10 – teste após usar função quicksort

2.0.5 char* OrdenaChave(palavra);

char* OrdenaChave(palavra) é usada na função main para ordenar a palavra a ser buscada e seus anagramas. Sua função segue a mesma ideia da função **OrdenaPalabra**, onde todos caracteres da palavra são percorridos e ordenados pelo ASCII. A Figura 11 mostra a função.

```
120  char *OrdenaChave(const char* chave){
121
122      int size = strlen(chave);
123      char *temp = calloc(size, sizeof(char));
124      strcpy(temp, chave);
125      char aux;
126      int n = strlen(temp);
127
128      for (int i = 0; i < n; i++)
129      {
130          for (int j = (i+1) ; j < n; j++)
131          {
132              if ((int)temp[i] > (int)temp[j])
133              {
134                  aux = temp[i];
135                  temp[i] = temp[j];
136                  temp[j] = aux;
137              }
138          }
139      }
140
141      return temp;
142
143  }
```

Figura 11 –

2.0.6 char* deixaMinusculo(palavra);

char* deixaMinusculo(palavra) é usada na função main para deixa a palavra a ser buscada minuscula. Ele segue a mesma ideia da função **InserePalavra** da Figura 5.

```
220  char* deixaMinusculo(const char* palavra){
221      int c=0;
222      char ch;
223      int size = strlen(palavra);
224      char *temp = calloc(size, sizeof(char));
225      strcpy(temp, palavra);
226
227      while (palavra[c] != '\0') {
228          ch = temp[c];
229          if (ch >= 'A' && ch <= 'Z')
230              temp[c] = temp[c] + 32;
231          c++;
232      }
233
234      return temp;
235  }
```

Figura 12 – função deixaMinusculo

2.0.7 BuscaBinaria(char* chave, Dicionario** dicionario, int e, int d);

BuscaBinaria(char* chave, Dicionario dicionario, int e, int d)** é usada na função main para buscar a palavra a ser buscada e a retornar a primeira posição de seus anagramas.

A função roda enquanto a posição esquerda for menor ou igual a posição direita do conjunto e retorna caso essa condição não seja satisfeita OU a posição da palavra tenha sido encontrada.
A linha **if (strcmp(dicionario[meio]->palavraOrdenada, chave)== 0)** é satisfeita quando a

```c
int BuscaBinaria(char* chave, Dicionario** dicionario, int e, int d){
    int flag = 1;
    int meio;
    if( e <= d){
     meio  = (e + d) / 2;
        if (strcmp(dicionario[meio]->palavraOrdenada, chave)== 0)
        {
            while (flag == 1)
            {
                if (strcmp(dicionario[meio]->palavraOrdenada, chave) ==0)
                {
                    meio--; // retorna de posição enquanto for chave.
                }
                else
                    flag = 0;
            }
            return meio;
        }
        else{
            if(strcmp(dicionario[meio]->palavraOrdenada, chave) > 0)
                return BuscaBinaria(chave, dicionario, meio + 1, d);
            else
                return BuscaBinaria(chave, dicionario, e, meio - 1);
            }
    }
    return -1;
}
```

Figura 13 –

posição **meio** do dicionario de palavras ordenadas for igual a palavra a ser buscada. Caso satisfeita, um ciclo de while é rodado, voltando uma posição para cada anagrama encontrado na posição anterior, e por fim, retornando a posição para a main.

O Else da função basicamente compara se a palavra ordenada na posição do meio é menor ou maior que a chave. Caso seja maior, ele corta o vetor de busca pela metade na **direita**, caso contrario, na**esquerda**, fazendo então a função recursiva para o caso.

Com o algoritmo de Busca Binária é então satisfeita a condição de busca em (lg n).

Feita a Busca Binária, usamos a função auxiliar **ImprimeFaixa** para imprimir todos os Anagramas da palavra a ser buscada, e com isso, finalizamos a implementação do algoritmo.

2.1 Testes

Para efeito de prova, vamos realizar alguns testes no programa. A função main para os testes está na Figura abaixo:

```c
#include <stdio.h>
#include <stdlib.h>
#include "func.h"

int main()
{
  int pos;
  char* palavra = "";
  int length = sizeFile();
  Dicionario **dicionario = criaDicionario(length);
  insereDicionario(dicionario);

  quicksort(dicionario, length);

  printDicionario(dicionario, length);

  palavra = OrdenaChave(palavra);
  palavra = deixaMinusculo(palavra);

  pos = BuscaBinaria(palavra, dicionario, 0, length);
  printf("\n\nPosição inicial dos Anagramas: %d\n", pos);


  ImprimeFaixa(palavra,dicionario,pos);


  return 0;
}
```

Figura 14 –

2.1.1 palavra = Abner

```
Posição inicial dos Anagramas: 41108
Anagrama [0] abner
Anagrama [1] berna
Anagrama [2] berna
```

Figura 15 –

2.1.2 palavra = Carlos

```
Posição inicial dos Anagramas: -1
```

Figura 16 –

2.1.3 palavra = Amor

```
Posição inicial dos Anagramas: 48111
Anagrama [0] armo
Anagrama [1] amor
Anagrama [2] roma
```

Figura 17 –

2.2 Nota:

Meu compilador estava com erro na hora da execução e foi necessário eu compilá-lo na web. Como no compilador que executei o código existia limite de alocação tive que tirar grande parte das palavras do arquivo texto, oque resultou nos testes não mostrarem todos os anagramas.

2.3 Conclusão

A implementação do dicionário de anagramas nos ajuda a entender métodos de manipulamento de palavras, ao invés de tipicamente numeros para algoritmos de ordenação e de busca como quicksort e bubblesort.

Referências

Quick sort:
https://www.geeksforgeeks.org/quick-sort/ .

Bubble sort:
https://www.geeksforgeeks.org/bubble-sort/