UFOP
Universidade Federal
Ouro Preto

RELATÓRIO

OURO PRETO - MG
Agosto/2017

Equipe:
Gabriel Lima
Mauro Del Gaudio
Yago Santos

RELATÓRIO

OURO PRETO - MG
Agosto/2017

Resumo

Com a finalidade de aprofundar os conhecimentos obtidos na disciplina de Estrutura de Dados II foi realizado este trabalho, onde foram aplicados algoritmos de casamento de cadeias (Força bruta, Boyer-Moore com a heurística ocorrência, Boyer-Moore-Horspool, Shift-And Exato e Boyer-Moore-Horspool-Sunday). Será apresentada, também, uma análise experimental destes métodos.

O principal objetivo do trabalho é a comparação do desempenho dos métodos citados acima em diferentes situações, levando em consideração as seguintes características:

- Número de deslocamentos realizados, quando for o caso, na varredura do texto T;
- Número de comparações entre os caracteres do texto T e do padrão P, quando for o caso; valores do campo de ordenação dos registros;
- Tempo de execução (tempo do término de execução menos o tempo do início de execução).

Palavras-chave: Casamento de cadeias, Força Bruta, Boyer Moore, Shift-And Exato.

SUMÁRIO

1 – Introdução 5

2 – Implementação 5

2.1 Método 1 – Força Bruta 5

2.2 Método 2 – Boyer Moore com a heurística ocorrência 6

2.3 Método 3 – Boyer-Moore-Horspool 7

2.4 Método 4 – ShiftAndExato 7

2.5 Método 5 – Boyer-Moore-Horspool-Sunday 8

3 - Programa Principal 8

4 – Análises 8

4.1 Análise comparativa de desempenho entre os métodos 8

5 - Informações complementares 15

6. Referências bibliográficas 15

7 – Código 16

1. Introdução

A relevância desse trabalho no campo da computação, pauta-se pela busca de possibilidades de exploração de algoritmos de casamento de cadeias e suas potencialidades, com intuito de oferecer qualidade e eficiência aos usuários no que tange o problema do casamento exato, ou aproximado, de um determinado padrão, para encontrar suas ocorrências dentro de um texto.

Tais algoritmos possuem uma importante aplicação em diversas áreas do conhecimento, como no estudo de sequências de DNA, algoritmos de busca em texto, aplicações em editores de texto e mesmo em segurança de redes.

Para o estudo de tais métodos, foram implementados os seguintes algoritmos:

1. Força Bruta
2. Boyer-Moore (utilizando a heurística de ocorrência)
3. Boyer-Moore-Horspool
4. Shift-And Exato
5. Boyer-Moore-Horspool-Sunday

Com o intuito de comparar os métodos estudados, foram realizados testes variando o tamanho do arquivo e o tamanho da palavra buscada. Em cada um dos testes foram obtidas as seguintes medidas:

- O número de comparações entre os caracteres do texto e do padrão buscado.
- Número de de deslocamentos realizados.
- Tempo de execução do programa.

Para cada um dos algoritmos foram realizados testes com arquivos de texto de tamanho(n) variados, com os seguintes tamanhos:

- Pequeno: $3.000 \leq N \leq 5.000$
- Médio: $220.000 \leq N \leq 250.000$
- Grande: $1.700.000 \leq N \leq 2.000.000$

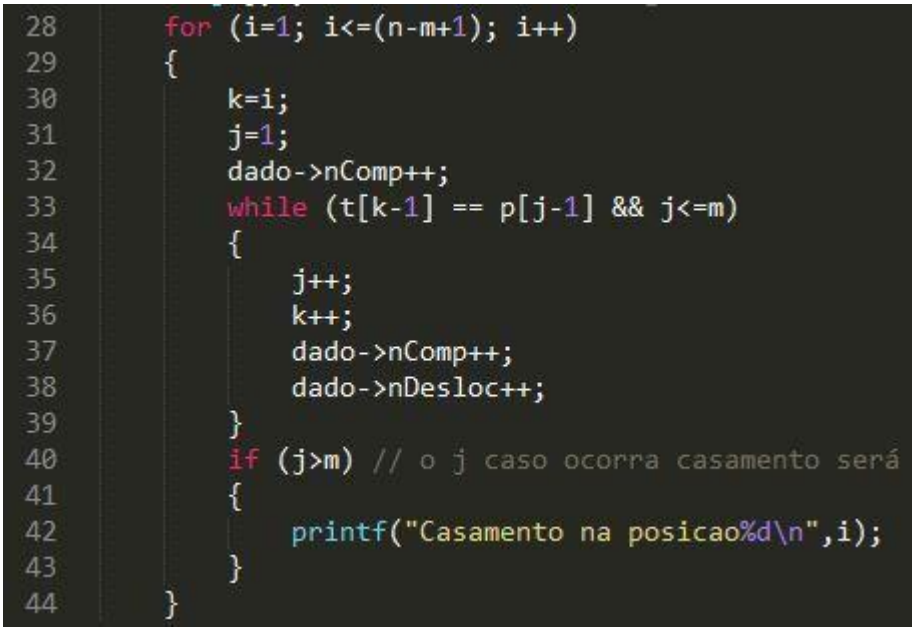
Para cada um dos arquivos foram realizadas buscas com palavras com os seguintes tamanhos:

- $3 \leq M \leq 5$
- $10 \leq M \leq 15$
- $25 \leq M \leq 30$

2. Implementação

2.1 Método 1 – Força bruta

É o mais simples dos algoritmos apresentados nesse trabalho, o algoritmo tenta casar qualquer subcadeia no texto com o padrão.



```
28      for (i=1; i<=(n-m+1); i++)
29      {
30          k=i;
31          j=1;
32          dado->nComp++;
33          while (t[k-1] == p[j-1] && j<=m)
34          {
35              j++;
36              k++;
37              dado->nComp++;
38              dado->nDesloc++;
39          }
40          if (j>m) // o j caso ocorra casamento será
41          {
42              printf("Casamento na posicao%d\n",i);
43          }
44      }
```

O custo do algoritmo de Força Bruta é de $C_n = m \times n$ para o pior caso e de $C_n - = c/c\text{-}1\ (1\text{-}\ 1/c^m)x\ (n\text{-}m+1) + O(1)$ para o caso esperado.

2.2 Método 2 – Boyer Moore com a heurística ocorrência

O algoritmo Boyer-Moore para casamento exato de cadeias consiste em pesquisar um padrão P numa janela que percorre um texto T. Para cada posição da janela, o algoritmo verifica o casamento de um sufixo da janela com um sufixo do padrão. Essa verificação é realizada no sentido da direita para a esquerda. Se nesse processo não ocorreu uma desigualdade (colisão), conclui-se que uma ocorrência do padrão P foi encontrada no texto T. Caso tenha ocorrido uma desigualdade, o padrão terá que ser deslocado para a direita. O algoritmo, neste caso, dispõe de duas heurísticas para calcular o deslocamento do padrão para que possa ser realizada uma nova tentativa de casamento - heurísticas ocorrência e casamento. Nesse trabalho foi utilizado a heurística de ocorrência.

- Heurística ocorrência: realiza o alinhamento da posição no texto T que causou a colisão com o primeiro caractere no padrão P que case com ele. Esse alinhamento é realizado através do deslocamento sucessivo do padrão P para a direita, até que a condição P[posição_corrente] == T[caractere_que_colidiu].

- Heurística casamento: o alinhamento é realizado através do deslocamento do padrão P para a direita. A diferença é que se verifica se o padrão casa com o pedaço do texto casado no momento em que ocorre a colisão.

  Devido a escolha de heurística, surgiram simplificações desse algoritmo com o passar do tempo, como as propostas de Horspool, por exemplo, que será também objeto desse trabalho.

2.3 Método 3 - Boyer-Moore_Horspool

O algoritmo Boyer-Moore-Horspool foi proposto em 1980 e apresentou uma simplificação do algoritmo Boyer-Moore, tornando-o mais rápido na pesquisa de padrões em um texto.

Horspool propos criar uma tabela de deslocamento para cada caractere do texto, correspondente ao ultimo caractere do padrão.

A tabela de deslocamento dos caracteres é gerada pela formula a seguir:

- d[x] = min{ j tal que (j = m) | (1 ≤ j < m P[m-j] = x)}

Independente do padrão a tabela de deslocamento é gerada durante a inicialização do algoritmo, para caracteres fora do padrão o valor de deslocamento é igual a m, para os caracteres dentro do padrão o deslocomento é calculado de acordo com sua posição no padrão.

Após gerar a tabela de deslocamento o algoritmo inicia a verificação do texto, o padrão é analisado com uma janela que se desloca ao longo do mesmo, a janela que é analisada com o padrão se desloca de acordo com a tabela de deslocamento. A comparação entre a janela e o padrão é feita de trás para frente, logo após a janela é comparada com padrão e é deslocada quando há uma colisão (caracteres distintos entre padrão e janela), de acordo com o ultimo caractere do texto referente a esse caractere no padrão.

A fase de pré-processamento do padrão ocorre nos dois primeiros loops do código acima. No pré-processamento é calculado a tabela de deslocamentos ***d***. O deslocamento pode ser computado com base apenas no padrão e no alfabeto, e a complexidade de tempo e de espaço para essa fase é O(c), onde c é o tamanho do alfabeto.

Para a fase de pesquisa, o pior caso do algoritmo é O(mn), caso os deslocamentos ocorram apenas de 1 em 1. No entanto o melhor caso é O(n/m), onde os deslocamentos são maiores, do tamanho do padrão. O caso esperado também é O(n/m), se *c* é pequeno e *m* não é muito grande.

2.4 – Shift-And Exato

No Shift-And Exato ocorre um pré processamento dos dados. Cada máscara do alfabeto utilizado é inicializada com zero, o vetor do padrão é percorrido e para cada letra encontrada é atribuído 1 na posição atual da máscara correspondente.

```
Long Masc[MAXCHAR], R = 0;
fseek(pArq, 0, 2); // Deslocando o ponteiro para o final do arquivo;
int tamArq = ftell(pArq); // Tamanho do arquivo
int i=0;
rewind(pArq);   //Voltando o ponteiro para o inicio do arquivo;

char *t = (char *) malloc(tamArq);   // vetor texto

while (!feof(pArq+)) {
    t[i] = fgetc(pArq);
    i++;
}
fclose(pArq);
// Fim de manipulacao de arquivo
```

Para cada novo caractere lido do texto, o valor da máscara R' é atualizado pela expressão: R' = ((R >> 1) | 10m−1) & M[T[i]].

A cadeia vazia também é marcada como um sufixo, permitindo um casamento na posição corrente do texto.

Do conjunto obtido, são mantidas apenas as posições em que houve um casamento com o padrão.

O custo do algoritmo Shift-And é O(n), desde que as operações possam ser realizadas em O(1) e o padrão caiba em umas poucas palavras do computador.

2.5 Boyer-Moore-Horspool-Sunday

Em 1990, surge mais uma variante do algoritmo de Boyer-Moore, uma melhoria proposta por Sunday, consiste em endereçar a tabela com o caractere no texto correspondente ao próximo caractere após o ultimo caractere do padrão. Para pré processar o padrão, o valor inicial do alfabeto utilizado na tabela de deslocamento é registrado com m+1. Dessa forma, os m caracteres iniciais do padrão, são usados para obter os outros valores da tabela.

3. Programa Principal

O programa principal executa testes em arquivos de textos de tamanhos variados: pequeno ($5.000 \leq n \leq 8.000$), médio ($200.000 \leq n \leq 250.000$) e grande ($1.500.000 \leq n \leq 2.000.000$).

Os parâmetros para definir o método, o texto e o padrão a ser procurado no texto são passados na execução do programa via terminal com a seguinte estrutura

casamento <método> <texto> <padrão> -P

-P é um parâmetro opcional para ser utilizado quando se deseja que os valores referentes aos quesitos de análise sejam exibidos na tela.

O programa principal também é responsável por tratar possíveis erros de entrada ou avisar erros ocorridos durante o programa.

4. Análises

4.1 Análise comparativa de desempenho entre os métodos

T : Pequeno.txt 7.379 caracteres

P : escr 4 caracteres

Forca Bruta:

Comparações: 8621

Deslocamento: 815

Tempo de execução 1.00 milisegundos

Boyer-Moore com heuristica de ocorrencia:

Comparações: 9997

Deslocamento: 13571

Tempo de execução 1.00 milisegundos

Boyer-Moore-Horspool:

Comparações: 2125

Deslocamento: 80

Tempo de execução 0.00 milisegundos

Boyer-Moore-Horspool-Sunday:

Comparações: 1819

Deslocamento: 143

Tempo de execução 5.00 milisegundos

Shift-And Exato:

Comparações: 7577

Deslocamento: 7577

Tempo de execução 0.00 milisegundos

T : Medio.txt 231.982 caracteres

P : escr 4 caracteres

Forca Bruta:

Comparações: 272561

Deslocamento: 25439

Tempo de execução 24.00 milisegundos

Boyer-Moore com heuristica de ocorrencia:

Comparações: 317415

Deslocamento: 429938

Tempo de execução 31.00 milisegundos

Boyer-Moore-Horspool:

Comparações: 69317

Deslocamento: 2988

Tempo de execução 31.00 milisegundos

Boyer-Moore-Horspool-Sunday:

Comparações: 59599

Deslocamento: 5131

Tempo de execução 15.00 milisegundos

Shift-And Exato:

Comparações: 247125

Deslocamento: 247125

Tempo de execução 14.00 milisegundos

T : Grande.txt 1.652.628 caracteres

P : escr 4 caracteres

Força Bruta:

Comparações: 1897521

Deslocamento: 172896

Tempo de execução 162.00 milisegundos

Boyer-Moore com heuristica de ocorrencia:

Comparações: 2218627

Deslocamento: 3003329

Tempo de execução 152.00 milisegundos

Boyer-Moore-Horspool:

Comparações: 486539

Deslocamento: 24060

Tempo de execução 385.00 milisegundos

Boyer-Moore-Horspool-Sunday:

Comparações: 420095

Deslocamento: 399091

Tempo de execução 506.00 milisegundos

Shift-And Exato:

Comparações: 1724628

Deslocamento: 1724628

Tempo de execução 157.00 milisegundos

T : Pequeno.txt 7.379 caracteres

P : escrita nos au 14 caracteres

Força Bruta:

Comparações: 8382

Deslocamento: 815

Tempo de execuão 5.00 milisegundos

Boyer-Moore com heuristica de ocorrencia:

Comparações: 9017

Deslocamento: 14020

Tempo de execução 1.00 milisegundos

Boyer-Moore-Horspool:

Comparações: 1383

Deslocamento: 138

Tempo de execução 0.00 milisegundos

Boyer-Moore-Horspool-Sunday:

Comparações: 935

Deslocamento: 66

Tempo de execução 0.00 milisegundos

Shift-And Exato:

Comparações: 7577

Deslocamento: 7577

Tempo de execução: 15.00 milisegundos

T : Medio.txt 231.982 caracteres

P : escrita nos au 14 caracteres

Força Bruta:

Comparações: 272568

Deslocamento: 25449

Tempo de execução: 15.00 milisegundos

Boyer-Moore com heuristica de ocorrencia:

Comparações: 293131

Deslocamento: 457282

Tempo de execução 15.00 milisegundos

Boyer-Moore-Horspool:

Comparações: 44534

Deslocamento: 4447

Tempo de execução 31.00 milisegundos

Boyer-Moore-Horspool-Sunday:

Comparações: 28883

Deslocamento: 1977

Tempo de execução: 26.00 milisegundos

Shift-And Exato:

Comparações: 247125

Deslocamento: 247125

Tempo de execução 22.00 milisegundos

T : Grande.txt 1.652.628 caracteres

P : escrita nos au 14 caracteres

Forca Bruta:

Comparações: 1897752

Deslocamento: 173130

Tempo de execução 163.00 milisegundos

Boyer-Moore com heuristica de ocorrencia:

Comparações: 2054026

Deslocamento: 3193187

Tempo de execução 169.00 milisegundos

Boyer-Moore-Horspool:

Comparações: 316974

Deslocamento: 35190

Tempo de execução 190.00 milisegundos

Boyer-Moore-Horspool-Sunday:

Comparações: 203958

Deslocamento: 13584

Tempo de execução 150.00 milisegundos

Shift-And Exato:

Comparações: 1724628

Deslocamento: 1724628

Tempo de execução 169.00 milisegundos

T : Pequeno.txt 7.379 caracteres

P : olharnadireçãocontrariadias 27caracteres

Força Bruta:

Comparações: 8173

Deslocamento: 622

Tempo de execução 12.00 milisegundos

Boyer-Moore com heuristica de ocorrencia:

Comparações: 8039

Deslocamento: 14726

Tempo de execução 0.00 milisegundos

Boyer-Moore-Horspool:

Comparações: 466

Deslocamento: 39

Tempo de execução 2.00 milisegundos

Boyer-Moore-Horspool-Sunday:

Comparações: 473

Deslocamento: 67

Tempo de execução 15.00 milisegundos

Shift-And Exato:

Comparações: 7577

Deslocamento: 7577

Tempo de execução 1.00 milisegundos

T : Medio.txt 231.982 caracteres

P : olharnadireçãocontrariadias 27caracteres

Força Bruta:

Comparações: 266594

Deslocamento: 19495

Tempo de execução 15.00 milisegundos

Boyer-Moore com heuristica de ocorrencia:

Comparações: 262066

Deslocamento: 481649

Tempo de execução 19.00 milisegundos

Boyer-Moore-Horspool:

Comparações: 14566

Deslocamento: 1122

Tempo de execução 34.00 milisegundos

Boyer-Moore-Horspool-Sunday:

Comparações: 16071

Deslocamento: 2026

Tempo de execução 31.00 milisegundos

Shift-And Exato:

Comparações: 247125

Deslocamento: 247125

Tempo de execução 15.00 milisegundos

T : Grande.txt 1.652.628 caracteres

P : olharnadireçãocontrariadias 27caracteres

Força Bruta:

Comparações: 1858793

Deslocamento: 134191

Tempo de execução 164.00 milisegundos

Boyer-Moore com heuristica de ocorrencia:

Comparações: 1831414

Deslocamento: 3357168

Tempo de execução 162.00 milisegundos

Boyer-Moore-Horspool:

Comparações: 102843

Deslocamento: 6854

Tempo de execução 152.00 milisegundos

Boyer-Moore-Horspool-Sunday:

Comparações: 114082

Deslocamento: 13293

Tempo de execução 168.00 milisegundos

Shift-And Exato:

Comparações: 1724628

Deslocamento: 1724628

Tempo de execução 169.00 milisegundos

Na execução dos testes, constatamos que o algoritmo que obteve melhor resultado foi o algoritmo de Boyer-Moore-Horspool-Sunday, obtendo os menores numeros de comparações, deslocamento e tempo que os demais.

O algoritmo **Força Bruta**, se mostrou melhor que o método de Boyer-Moore com heuristica de ocorrencia, em relação a comparação, deslocamento e tempo de execução em casos de ocorrência com menos de 25 caracteres.

A razão para isso é a baixíssima probabilidade de encontrar um casamento de padrão com palavras muito grandes. Caso o texto encontre-se em linguagem natural, dificilmente encontraremos alguma palavra com mais de 4 caracteres, e mesmo um texto gerado aleatoriamente, como no caso estudado, a medida que o tamanho da palavra buscada cresce, a probabilidade de encontrar mais caracteres com casamento decresce. Dessa forma, o algoritmo de busca será interrompido, em média, na mesma posição. Portanto, o número de comparações será independente do tamanho gerado.

Uma outra consequência da baixa probabilidade de encontrar-se casamento de padrões é que o número de comparações será sempre um pouco maior que o número de caracteres do texto, pois normalmente a busca por casamento de padrões será interrompida já nos primeiros caracteres.

O algoritmo que obteve os piores resultados foi o de **Boyer-Moore** com heuristica de ocorrencia, o qual teve o maior número de deslocamento para todos os casos, chegando a duplicar o número de acordo com seus caracteres.

Pode-se observar que o número de comparações não muda praticamente nada conforme o tamanho das palavras chaves aumenta. Isso se deve por que a fase de pre-processamento varia de acordo com o tamanho da palavra buscada. Já a busca será realizada em um loop variando de 1 até o tamanho do texto, em que o paralelismo de bit torna tais operações com custo O(1).

Desta forma, o algoritmo fará o número de comparações de acordo com o tamanho da entrada do texto em que se fará a busca da palavra chave, tornando o algoritmo O(n).

5. Informações complementares

Todos os testes foram realizados em uma máquina com a seguinte configuração:

- Processador: Intel® Core™ i5 4210U (1.7 GHz até 2.7 GHz, 3 MB L3 Cache);
- Placa gráfica: NVIDIA® GeForce® 710 M Graphics 2 GB gDDR3 de memória dedicada (Optimus™);
- Memória RAM: 8 GB DDR3L (1600 MHz);
- Armazenamento: 1TB S-ATA III Hard Drive (5400RPM).

6. Referências bibliográficas:

- Processamento de Cadeias de Caracteres* (8 de Novembro de 2010), (*Transparências elaboradas por Fabiano Cupertino Botelho, Charles Ornelas Almeida, Israel Guerra e Nivio Ziviani). Disponível em http://www2.dcc.ufmg.br/livros/algoritmos/cap8/slides/c/completo1/cap8.pdf;
- Casamento de Cadeias, Estrutura de Dados II, (Prof. Guilherme Tavares de Assis). Disponível em: http://www.decom.ufop.br/guilherme/BCC203/geral/ed2_casamento-cadeias.pdf.

7. Código

main.c

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "TADCasamentoCadeia.h"

void visu()
{
	printf("\n");
	for(int i=0; i<75; i++)
	{
		printf("_");
	}
	printf("\n");
}

int main(int argc, char *argv[])
{
    if (atoi(argv[1]) < 1 || atoi(argv[1])>5)
    {
        printf("Metodo invalido. Programa encerrado");
        return 0;
    }
    if (argc < 4)
    {
        printf("Falta parametros. Programa encerrado");
        return 0;
    }
```

```
if (strcmpi(argv[2], "-P") == 0 || strcmpi(argv[3], "-P") == 0)
{
    printf("Parametro invalido. Programa encerrado");
    return 0;
}

clock_t inicio, fim;
inicio = clock();
TComparacao dado;
dado.nComp = dado.nDesloc = 0;

switch (atoi(argv[1]))
    {
            case 1:
                    visu();
                    forcaBruta(argv[2],argv[3],strlen(argv[3]),&dado);
                    fim = clock();
                    break;
            case 2:
                    visu();
                    BM_Ocorrencia(argv[2], argv[3], strlen(argv[3]), &dado);
                    fim = clock();
                    break;
            case 3:
                    visu();
                    BMH(argv[2],argv[3],strlen(argv[3]),&dado);
                    fim = clock();
                    break;
            case 4:
                    visu();
```

```
                ShiftAndExato(argv[2], argv[3], strlen(argv[3]), &dado);
                fim = clock();
                break;
        case 5:
                visu();
                BMHS(argv[2],argv[3],strlen(argv[3]),&dado);
                fim = clock();
                break;
        default:
                printf("\nValor invalido\n");
                break;
    }
    if (strcmpi(argv[4], "-P") == 0)
        {
        printf("\nTempo de execucao %.2f milisegundos.\nForam realizadas %d comparacoes e %d
 deslocamento.\n", difftime(fim, inicio), dado.nComp, dado.nDesloc);
        visu();
    }
        return 0;
 }
```

ForcaBruta.c

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

void forcaBruta(char nmArq[],char p[],int m, TComparacao *dado)
{

   FILE *pArq = fopen(nmArq, "r");
   if (!pArq)
   {                                // Verificando a abertura do arquivo
      perror("Erro ao abrir o arquivo");
      exit(0);
   }
   fseek(pArq, 0, 2);                    // Deslocando o ponteiro para o final do arquivo;
```

```
    int n = ftell(pArq);                    // Tamanho do arquivo;
    rewind(pArq);                           // Deslocando o ponteiro para o inicio do arquivo;
    char *t = (char *) malloc(n);
    int i=0;
    printf("\n\nForca Bruta:\n");
    while (!feof(pArq))
    {
        t[i] = fgetc(pArq);                 // Copia todo o texto para t
        i++;
    }
    fclose(pArq);
    long j,k;
    for (i=1; i<=(n-m+1); i++)
    {
        k=i;
        j=1;
        dado->nComp++;
        while (t[k-1] == p[j-1] && j<=m)
        {
            j++;
            k++;
            dado->nComp++;
            dado->nDesloc++;
        }
        if (j>m)                            // o j caso ocorra casamento será maior pois a comparação é feita até
j<=m;
        {
            printf("Casamento na posicao: %d\n",i);
        }
    }

}
```

BM_Ocorrencia.c

```
#include <stdio.h>
#include <stdlib.h>
#include "TADCasamentoCadeia.h"

void BM_Ocorrencia(char nmArq[], char p[], int tamPadrao, TComparacao *dado)
{

    FILE *pArq = fopen(nmArq, "r");
    if (!pArq) {
        perror("Erro ao abrir o arquivo");
        exit(0);
    }
   fseek(pArq, 0, 2); // Ir para o fim do arquivo
```

```
    int tamArq = ftell(pArq); // Tamanho do arquivo
    rewind(pArq); // Volta o cursor do arquivo para o inicio

    char *texto = (char *) malloc(tamArq);
    int i=0, k, j;

    while (!feof(pArq)) {
        texto[i] = fgetc(pArq);
        i++;
    }
    fclose(pArq);

    i=tamPadrao;
    printf("\nBoyer-Moore com heuristica de ocorrencia:\n");

    while (i<=tamArq) {
        k=i;
        j=tamPadrao;                    // Verificando igualdade entre os caracteres

        dado->nComp++;
        while (texto[k-1] == p[j-1] && j>0) {  // busca por semelhantes
            j--;
            k--;
            dado->nDesloc++;
            dado->nComp++;
        }
        if (j == 0)
        {
            printf("Casamento exato na posicao: %d\n", k+1);
            i++;
            continue;
        }

        int salto=1;
        j--;

                                // Calcula qual e o deslocamento para casamento exato no caracter distinto
        dado->nComp++;
        while (texto[k-1] != p[j-1] && j>0) {
            j--;                // Leitura da direita para a esquerda no padrao
            salto++;
            dado->nDesloc++;
            dado->nComp++;
        }
        i+=salto;                   // Soma o salto (deslocamento)
    }
    dado->nDesloc+=tamArq;
    free(texto);
}
```

BMH.c

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

#define MAXCHAR 256

void BMH(char nmArq[],char p[],int m,TComparacao *dado)
{
   FILE *pArq = fopen(nmArq, "r");
   if (!pArq)
   {
      perror("Erro ao abrir o arquivo");
      exit(0);
   }

   fseek(pArq, 0, 2); // Ir para o fim do arquivo.
   int n = ftell(pArq); // Tamanho do arquivo.
   rewind(pArq); // Volta o cursor do arquivo para o inicio.

   char *texto = (char *) malloc(n);
   int x=0;
   while (!feof(pArq)) {
      texto[x] = fgetc(pArq);
      x++;
   }
   fclose(pArq);
   printf("\n\nBoyer Moore Horspool:\n");

   long i,j,k;
   long d[MAXCHAR + 1];

   for (j=0; j<=MAXCHAR; j++)                    //Para todos os valores fora do padrão o valor de
deslocamento é igual a "m".

      d[j] = m;
   for (j=1; j<m; j++)                           //Calcula o valor de deslocamento para os caracteres dentro do
padrão "p".
      d[(unsigned char)p[j-1]] = m-j;
   i=m;
   while (i<=n)
   {
      k=i;
      j=m;
      dado->nComp++;
      while (texto[k-1] == p[j-1] && j>0) {
         k--;
         j--;
         dado->nComp++;
         dado->nDesloc++;
      }
      if (j == 0)
         printf("Casamento na posicao: %3ld\n",k+1);
```

```
        i += d[(unsigned char)texto[i-1]];
    }
}
```

ShiftAndExato.c

```
#include <stdio.h>
#include <stdlib.h>
#include "TADCasamentoCadeia.h"

#define MAXCHAR 256

void ShiftAndExato(char nmArq[], char p[], int tamPadrao, TComparacao *dado)
{
    printf("\n\nShift And Exato:\n");
    FILE *pArq = fopen(nmArq, "r");
    if (!pArq)
    { //Checando a abertura do arquivo;
        perror("Erro ao abrir o arquivo");
        exit(0);
    }

    long Masc[MAXCHAR], R = 0;
    fseek(pArq, 0, 2);                //O ponteiro e deslocado para o final do arquivo;
    int tamArq = ftell(pArq);
    int i=0;
    rewind(pArq);                     //O ponteiro e deslocado para o inicio do arquivo;

    char *t = (char *) malloc(tamArq);   // vetor texto

    while (!feof(pArq)) {
        t[i] = fgetc(pArq);
        i++;
    }
    fclose(pArq);


    for (i=0; i<MAXCHAR; i++)
        Masc[i] = 0;
    for (i=1; i<=tamPadrao; i++)
        Masc[p[i-1] + 127] |= 1 << (tamPadrao - i);
```

```
    for (i=0; i<tamArq; i++) {
        R = ((((unsigned long) R)>>1) | (1<<(tamPadrao-1))) & Masc[t[i] + 127];
        dado->nComp++;
        dado->nDesloc++;
        if ((R & 1) != 0)
        {
            printf("Casamento na posicao: %d\n", i-tamPadrao+2);
        }
    }
}
```

BMHS.c

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

#define MAXCHAR 256

void BMHS(char nmArq[],char p[],int m,TComparacao *dado)
{
   FILE *pArq = fopen(nmArq, "r");
   if (!pArq)
   {
      perror("Erro ao abrir o arquivo");
      exit(0);
   }

   fseek(pArq, 0, 2); // Ir para o fim do arquivo.
   int n = ftell(pArq); // Tamanho do arquivo.
   rewind(pArq); // Volta o cursor do arquivo para o inicio.

   char *texto = (char *) malloc(n);
   int x=0;
   while (!feof(pArq)) {
      texto[x] = fgetc(pArq);
      x++;
   }
   fclose(pArq);
   printf("\n\nBoyer Moore Horspool Sunday:\n");

   long i,j,k;
   long d[MAXCHAR + 1];

   for (j = 0; j <= MAXCHAR; j++)                    //Para todos os valores fora do padrão o valor de
deslocamento é igual a "m".

      d[j] = m + 1;
```

```
    for (j = 1; j <= m; j++)                    //Calcula o valor de deslocamento para os caracteres dentro
do padrão "p".
        d[(unsigned char)p[j - 1]] = m - j + 1;
    i = m;
    while (i <= n)
    {
        k = i;
        j = m;
        dado->nComp++;
        while (texto[k-1] == p[j-1] && j>0) {
            k--;
            j--;
            dado->nComp++;
            dado->nDesloc++;
        }
        if (j == 0)
            printf("Casamento na posicao: %3ld\n",k + 1);
        i += d[(unsigned char)texto[i]];
    }
}
```