

Jose Diego - 1761, Vitor - 3049

Trabalho de Projeto e Análise de Algoritmos

Brasil

1 de Dezembro de 2018

Jose Diego - 1761, Vitor - 3049

Trabalho de Projeto e Análise de Algoritmos

Trabalho referente a implementação de algoritmos para resolução de problemas baseado em casamento de padrões estudados em sala, para a disciplina de Projeto e Analise de Algoritmos.

Universidade Federal de Viçosa

Campos Florestal

Ciência da Computação

Brasil

1 de Dezembro de 2018

Resumo

Em ciência da computação vários são os problemas enfrentados na implantação de uma solução baseado em um contexto da vida real, abstrair uma solução viável é sempre um desafio. Várias são as técnicas de casamento de padrão, algumas delas muito eficientes. Neste trabalho abordaremos duas dessas técnicas, implementando seus respectivos algoritmos, como também discutiremos seus resultados.

Palavras-chaves: algoritmos, casamento padrão, BMH, KMP.

Lista de ilustrações

Figura 1 – Implementação BMH	7
Figura 2 – Tela do BMH	8
Figura 3 – Implementação do KMP	9
Figura 4 – Tela do KMP	9
Figura 5 – Tabela de Resultados	10
Figura 6 – Gráfico dos Resultados	10
Figura 7 – Implementação do BMHS	11
Figura 8 – Implementação do Shift-And	12
Figura 9 – Grafico de comparação	13

Sumário

1	Introdução	5
2	Metodologia	6
3	Desenvolvimento	7
3.1	Tarefa A	7
3.1.1	BMH	7
3.1.2	Execução do BMH	8
3.1.3	KMP	8
3.1.4	Execução do KMP	9
3.2	Tarefa B	10
3.2.1	Resultados	10
3.3	Atividade Extra	11
3.3.1	BMHS	11
3.3.2	Shift-And	11
3.3.3	Comparações	12
	Conclusão	14
	Referências	15

1 Introdução

O texto aqui apresentado refere-se o relatório do trabalho prático 4 da disciplina de programação e análise de algoritmos, neste trabalho será realizado a implementação de duas tarefas. Na primeira, Tarefa A, teremos as seguintes técnicas analisadas BMH e KMP para casamento exato de padrão. Já na Tarefa B, realizaremos testes e discutiremos os resultados.

Este trabalho segue dividido na seção 2: Metodologia; seguido da seção 3: Desenvolvimento detalhando a realização do trabalho, e por fim a Conclusão e Referências.

2 Metodologia

A metodologia adotada consiste primeiramente, no estudo teórico da técnica de dos dois algoritmos a serem implementados **BMH** e **KMP**, para a solução do problema de casamento do padrão, em seguida a implementação destes na linguagem C conforme a descrição das tarefas contidas no relatório do trabalho prático.

No desenvolvimento do trabalho foi utilizado um máquina com sistema operacional **Ubuntu 16.04 64 bits, 6 GB de RAM e processador i5-3337U CPU 1.80 GHz**. Sobre a arquitetura utilizada, foi utilizado a IDE Code Blocks para execução dos programas desenvolvidos, ferramenta escolhida devido ao seu grande uso por parte da comunidade da área e recomendações ao longo do curso.

Além disso foi utilizado a ferramenta de edição online **overleaf** para a criação e edição compartilhada deste texto. E o google drive para gerenciamento de versões do programa desenvolvido, tudo seguindo boas práticas de estruturas de dados como em [Ziviani \(2010\)](#).

3 Desenvolvimento

3.1 Tarefa A

Na tarefa A, veremos a implementação e análise dos dois algoritmos estudados em sala, primeiro do **BMH**, logo em seguida do **KMP**.

3.1.1 BMH

O algoritmo Boyer-Moore-Horspool é um aprimoramento do algoritmo Boyer-Moore. O algoritmo Boyer-Moore foi publicado em 1977, mas em 1980 Horspool apresentou uma simplificação no algoritmo original, tão eficiente quanto o algoritmo original, que ficou conhecida como Boyer-Moore-Horspool (BMH).

Pela extrema simplicidade de implementação e comprovada eficiência, o BMH deve ser escolhido em aplicações de uso geral que necessitam realizar casamento exato de cadeias. É baseado na pesquisa de P em uma janela que desliza ao longo de T, pesquisando por um sufixo da janela que corresponde a um sufixo de P, por comparações da direita para a esquerda. Inicialmente, o padrão procurado deve ser alocado em uma tabela. Cada letra representará uma posição dessa tabela.

Figura 1: Implementação BMH

```

9 void BMH(char T[], char P[], int linha)
10 {
11     int n = strlen(T);
12     int m = strlen(P);
13     long i, j, k, d[MAX + 1];
14
15     // Preprocessamento para obter a tabela de deslocamento
16     for(j=0; j<=MAX; j++)
17     {
18         d[j] = m;
19     }
20     for(j=1; j<m; j++)
21     {
22         d[P[j-1]] = m - j;
23         i = m;
24     }
25
26     while(i<=n)
27     {
28         k = i;
29         j = m;
30         // Procura por um sufixo do texto que casa com um sufixo do padrão
31         while(T[k-1] == P[j-1] && j > 0)
32         {
33             k--;
34             j--;
35         }
36
37         if(j==0)
38         {
39             printf("\tCasamento na posicao: %3ld, na linha %d\n", k, linha);
40         }
41         // Deslocamento da janela de acordo com o valor da tabela de deslocamento relativo ao caractere que esta
42         // na i-esima -1 posicao do texto, ou seja, a posicao do ultimo caractere do padrao p.
43         i = i + d[T[i-1]];
44     }
45 }
46

```

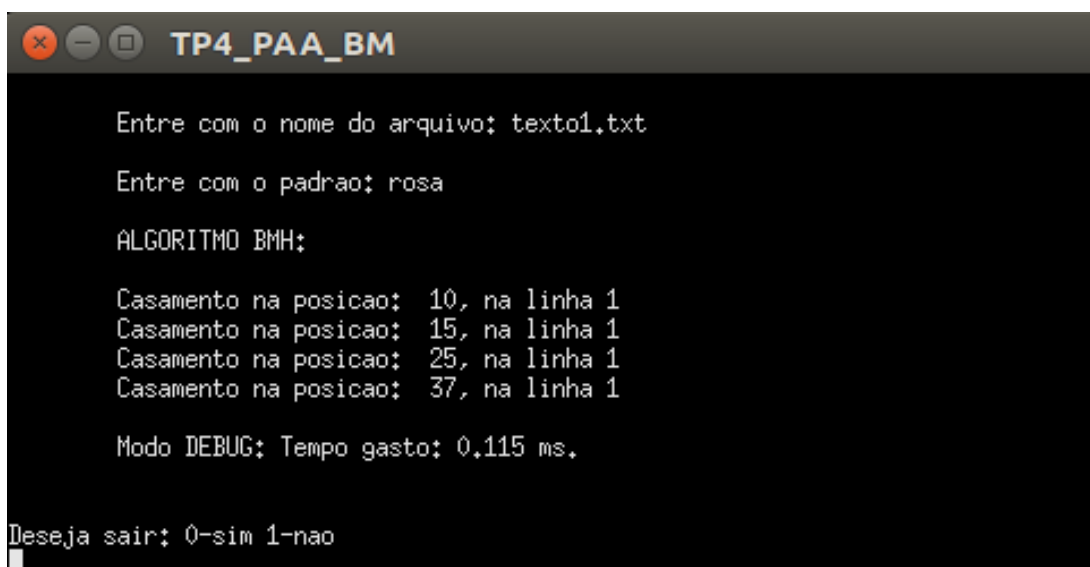
O algoritmo possui uma implementação simples como podemos ver na Figura 1, onde há inicialmente o pré processamento para obter a tabela de deslocamento, logo

depois procura por um sufixo do texto que casa com um sufixo do padrão e finalmente faz o deslocamento da janela de acordo com o valor da tabela de deslocamento relativo ao caractere que está na i -ésima-1 posição do texto, ou seja, a posição do último caractere do padrão p .

3.1.2 Execução do BMH

A implementação do algoritmo foi realizada em um TAD, para termos modularidade. A execução se dá facilmente através do código fonte encontrado no arquivo .zip contido no trabalho. No arquivo BMH.c temos a implementação de uma função apresentada na Figura 1. Para obter o texto, o usuário deve fornecer o nome do arquivo de texto. Após a leitura, o texto é armazenado em um vetor. O padrão também será fornecido pelo usuário. A tela do algoritmo pode ser vista na Figura 2.

Figura 2: Tela do BMH



```
TP4_PAA_BM

Entre com o nome do arquivo: texto1.txt

Entre com o padrao: rosa

ALGORITMO BMH:

Casamento na posicao: 10, na linha 1
Casamento na posicao: 15, na linha 1
Casamento na posicao: 25, na linha 1
Casamento na posicao: 37, na linha 1

Modo DEBUG: Tempo gasto: 0.115 ms.

Deseja sair: 0-sim 1-nao
```

A Figura 2 está em Modo Análise que mostra o tempo de execução gasto. Essa opção pode ser ativada ou desativada, bastando apenas alterar a variável de Debug, 0 para desativo ou 1 para ativo, no arquivo main.c.

3.1.3 KMP

O algoritmo Knuth-Morris-Pratt foi proposto em 1977 e é o primeiro algoritmo cujo pior caso tem complexidade de tempo linear no tamanho do texto, ou seja, $O(n)$. É um dos algoritmos mais famosos para resolver o problema do casamento de cadeias. É baseado na leitura de caractere por caractere do texto.

Na Figura 3 temos a função do algoritmo responsável pela construção da tabela do maior prefixo.

Figura 3: Implementação do KMP

```

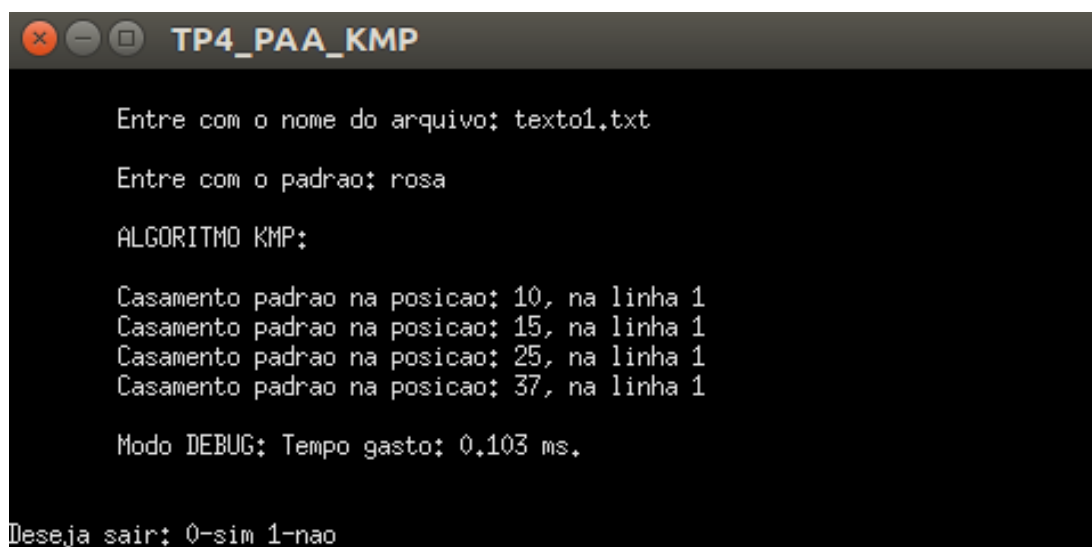
47 void KMPMatcher(char *pattern, int M, int *pi)
48 {
49     int q = 0; // q número de caracteres do prefixo.
50     int i;
51
52     pi[0] = 0;
53     i = 1;
54
55     // Calcula pi[i] para i = 1 até o tamanho do padrão (M-1).
56     while(i < M)
57     {
58         if(pattern[i] == pattern[q])
59         {
60             q++;
61             pi[i] = q;
62             i++;
63         }
64         else
65         {
66             if(q != 0)
67             {
68                 // O vetor pi que armazena o maior prefixo não reduzido encontrado.
69                 q = pi[q-1];
70             }
71             else
72             {
73                 pi[i] = 0;
74                 i++;
75             }
76         }
77     }
78 }
79

```

3.1.4 Execução do KMP

Agora logo na Figura 4 podemos ver a imagem da tela de execução do algoritmo em modo Debug para análise. A execução se dá facilmente através do código fonte encontrado no arquivo .zip contido no trabalho.

Figura 4: Tela do KMP



```

TP4_PAA_KMP

Entre com o nome do arquivo: texto1.txt

Entre com o padrao: rosa

ALGORITMO KMP:

Casamento padrao na posicao: 10, na linha 1
Casamento padrao na posicao: 15, na linha 1
Casamento padrao na posicao: 25, na linha 1
Casamento padrao na posicao: 37, na linha 1

Modo DEBUG: Tempo gasto: 0.103 ms.

Deseja sair: 0-sim 1-nao

```

Como podemos ver o programa fornece a posição e a linha aonde ocorreu o casamento de padrão. Em seguida na tarefa B, teremos a discussão dos resultados comparando os dois algoritmos implementados.

3.2 Tarefa B

Agora na tarefa B, apresentaremos os resultados encontrados pelas atividades na tarefa anterior.

3.2.1 Resultados

Para obter os dados para análises e testes de ambos os algoritmos, foram utilizados quatro textos e quatro padrões. Os textos e os padrões são os seguintes como mostra a Figura 5.

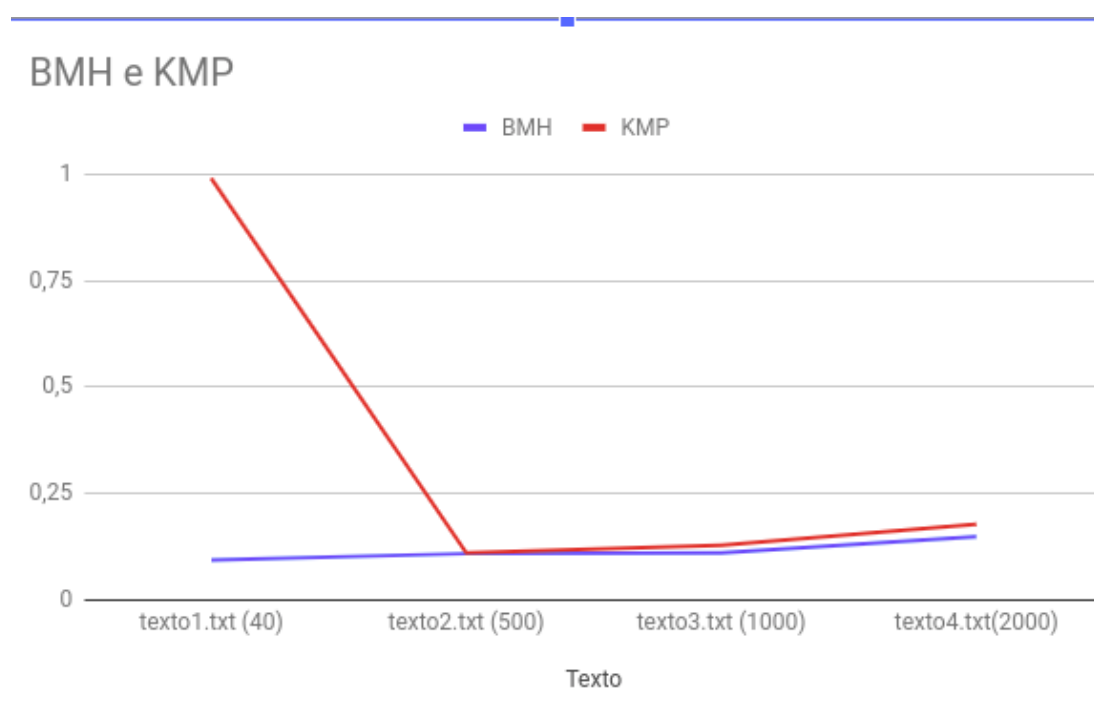
Figura 5: Tabela de Resultados

Padrão	Texto	BMH	KMP
rosa	texto1.txt (40)	0,094	0,99
Lorem	texto2.txt (500)	0,109	0,111
pretium	texto3.txt (1000)	0,111	0,129
eget	texto4.txt (2000)	0,149	0,178

Na tabela podemos ver os tempos em ms gastos por cada algoritmo, como também o tamanho dos textos em quantidade de caracteres de cada arquivo de entrada.

Apartir dessa tabela foi contruido um gráfico para melhor análise dos dados, no podemos o ver na Figura 6.

Figura 6: Gráfico dos Resultados



Ambos algoritmos são bastante conhecidos quando se trata de casamento exato de padrões. Como esperado, o resultado do BMH é melhor que o KMP, pois a complexidade do algoritmo BMH é melhor que a do KMP. O algoritmo KMP possui complexidade linear, ou seja, $O(n)$. No entanto, o algoritmo BMH possui complexidade sublinear, $O(n/m)$. Logo por isso ele apresenta um melhor resultado.

3.3 Atividade Extra

3.3.1 BMHS

Na implementação do BMH foi observado que facilmente poderíamos a partir dele chegar a implementação do algoritmo BMHS, que endereça a tabela com o caractere no texto correspondente ao próximo caractere após o último caractere do padrão, em vez de deslocar o padrão usando o último caractere como no algoritmo BMH. Que através de execuções feitas obtivemos resultados melhores, já que os deslocamentos são mais longos podendo ser iguais a $m + 1$, levando a saltos relativamente maiores para padrões curtos.

Figura 7: Implementação do BMHS

```

47 void BMHS(char T[], char P[], int linha)
48 {
49     int n = strlen(T);
50     int m = strlen(P);
51     long i, j, k, d[MAX + 1];
52
53     // Preprocessamento para obter a tabela de deslocamento
54     for(j=0; j<=MAX; j++)
55     {
56         d[j] = m + 1;
57     }
58     for(j=1; j<m; j++)
59     {
60         d[P[j-1]] = m - j + 1;
61         i = m;
62     }
63
64     while(i<=n)
65     {
66         k = i;
67         j = m;
68         // Procura por um sufixo do texto que casa com um sufixo do padrão
69         while(T[k-1] == P[j-1] && j > 0)
70         {
71             k--;
72             j--;
73         }
74
75         if(j==0)
76         {
77             printf("\tCasamento na posicao: %3ld, na linha %d\n", k, linha);
78         }
79         // Deslocamento da janela de acordo com o valor da tabela de deslocamento relativo ao caractere que esta
80         // na i-esima -1 posicao do texto, ou seja, a posicao do ultimo caractere do padrao p.
81         i = i + d[T[i]];
82     }
83 }

```

Na Figura 7, podemos ver as modificações feitas no algoritmo BMH para obtermos o algoritmo BMHS.

3.3.2 Shift-And

Optamos também por implementar o algoritmo do Shift-And que através do paralelismo de bits faz uma das buscas por padrões mais rápidas, uma vez que o número de operações diminui expressivamente. Essa transforma toda a cadeia de caracteres em bits

armazenado os em uma máscara onde será feita diversas operações de ou, and e shift na busca do casamento como o padrão.

Figura 8: Implementação do Shift-And

```
void ShiftAndAproximado(TipoTexto T, long n, TipoPadrao P, long m, int debug) {
    LARGE_INTEGER tf, ti, freq;
    float TempoTotal;

    QueryPerformanceCounter(&ti);

    long Masc[MaxCHAR], i, R = 0;
    long words=0;
    for (i = 0; i < MaxCHAR; i++)
        Masc[i] = 0;
    for (i = 1; i <= m; i++)
        Masc[P[i-1] + 127] |= 1 << (m - i);
    for (i = 0; i < n; i++)
    { R = (((unsigned long)R) >> 1) |
      (1 << (m - 1)) & Masc[T[i] + 127];
      if ((R & 1) != 0) {
          words++;
          printf(" Casamento na posição %3ld\n", i - m + 2);
      }
    }

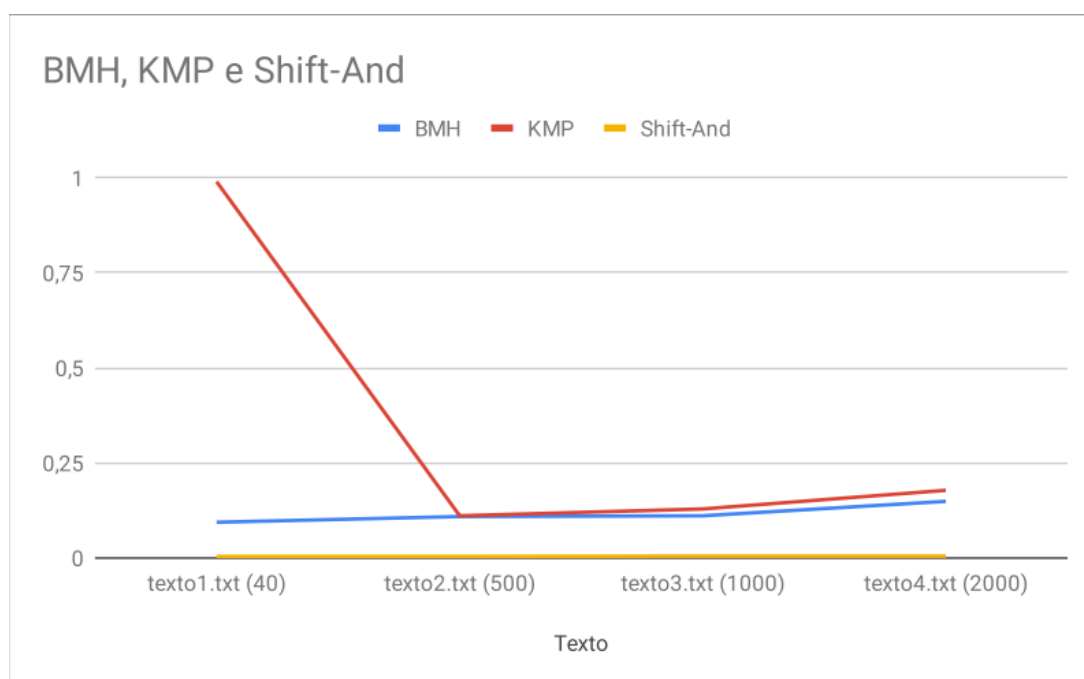
    printf("Houveram %ld ocorrências da palavra \"%s\"\n", words, P);
    QueryPerformanceCounter(&tf);
    QueryPerformanceFrequency(&freq);
    TempoTotal = (float) (tf.QuadPart - ti.QuadPart) / freq.QuadPart;
    if(debug==1)
        printf("O tempo total gasto para achar as ocorrências foi de: %f segundos \n\n", TempoTotal);
    system("Pause");
    system("CLS");
}
```

3.3.3 Comparações

Para melhor vermos os resultados, fizemos uma nova comparação gráfica entre os diversos algoritmos implementados, onde no gráfico podemos ver o resultado de cada algoritmo e observamos que o paralelismo de bits proporciona um grande desempenho.

Na Figura 8, podemos ver a implementação do algoritmo de Shift-And em C.

Figura 9: Grafico de comparação



Conclusão

Com este trabalho podemos entender a importância dos algoritmos de casamento exato de padrões, isso foi possível através das implementações realizadas, que juntamente com o estudo teórico, nos trouxe a visão de como é importante o trabalho realizado por esses algoritmos. E esta importância se dá pelo fato que em ciência da computação a vários contextos do mundo real que podem ser abstraídos, para utilização dos algoritmos estudados neste trabalho.

Referências

ZIVIANI, N. *Projeto de Algoritmos*. 3 edição. ed. [S.l.]: Cengage Learning, 2010. Citado na página [6](#).