



Universidade Federal de Viçosa - *Campus Florestal*
Bacharelado em Ciência da Computação
Disciplina: CCF330 - Projeto e Análise de Algoritmos
Prof. Dr. Daniel Mendes Barbosa

Trabalho Prático 03 de PAA

Casamento de padrões

Alunos:

Camila C Guimarães	2256
Samuel S C Nascimento	2662

Florestal, 2 de Dezembro de 2019

Conteúdo

1	Objetivo	3
2	Introdução	4
3	Contexto	5
3.1	Casamento de padrões	5
3.2	O sistema	5
4	Desenvolvimento do programa	6
4.1	Parte A e B	6
4.1.1	Arquivo 'partea.h'	6
4.1.2	Arquivo 'parte.c'	7
4.1.2.1	Função 'forcaBruta();'	7
4.1.2.2	Função 'BMHS();'	7
4.1.2.3	Função 'solucao();'	9
4.1.2.4	Outras funções	10
4.1.3	Arquivo 'main.c'	11
4.2	Parte C	11
4.2.1	Arquivo 'ParteC.h'	11
4.2.2	Arquivo 'ParteC.c'	12
4.2.2.1	Função 'IniciarVetorChar()'	12
4.2.2.2	Função 'IniciarMascara()'	13
4.2.2.3	Função 'ShiftAnd()'	14
4.2.2.4	Função 'ShiftAndAproximado()'	16
4.2.3	Arquivo 'main.c'	19
5	Testes	20
5.1	Parte A e B	20
5.1.1	Resultados	20
5.2	Parte C	22
5.2.1	Resultados	22
6	Conclusão	23
	Referências	24

1 Objetivo

O objetivo deste trabalho é mostrar na pratica a funcionalidade de algoritmos de casamento exato e aproximado de padrões. Para isso, o trabalho foi dividido em três partes (A, B e C).

Na tarefa A foi implementado 2 algoritmos diferentes (força bruta e BMHS) que resolvem o problema de casamento exato de padrões. Na tarefa B foram feitos os testes dos algoritmos da tarefa A.

A tarefa C tem o objetivo de implementar o casamento aproximado com Shift-And.

Além da implementação dos algoritmos para solucionar os problemas apresentados pelo professor Daniel Mendes, também foi feito a analise dos métodos utilizados para casamento exato e como eles se comportam para diferentes entrada.

2 Introdução

Este relatório é a documentação de um trabalho prático da disciplina de Projeto e Análise de Algoritmos. Nele consta a explicação do projeto implementado, os métodos implementados e os resultados obtidos.

Como dito anteriormente, a parte A do trabalho trata do casamento exato de padrões de duas formas diferentes. O algoritmo lê o arquivo com o texto, pede ao usuário para entrar com o padrão que deverá ser procurado e retorna o número de padrões encontrados e onde eles foram encontrados. A implementação do casamento exato foi feita de duas formas diferentes - força bruta e BMHS - e o usuário pode escolher qual forma deseja utilizar.

Na parte B foram feitas execuções com diferentes entradas para analisar o comportamento dos algoritmos de casamento exato. Ao final são apresentados gráficos com os dados analisados para mostrar o comportamento da execução.

A parte C do trabalho tem como objetivo a implementação do algoritmo Shift-And aproximado. Foi implementado primeiro, portanto, o algoritmo Shift-and para casamento exato, já que o Shift-And aproximado é uma modificação do exato. O algoritmo funciona da maneira que o usuário deve entrar com o Texto, o Padrão, o tamanho de K e as operações de inserção, substituição ou remoção. A ideia era implementar as duas formas e permitir que o usuário fizesse a escolha de qual gostaria de utilizar, contando então como o Shift-And exato um algoritmo extra, porém tivemos problemas ao implementar o Shift-And aproximado e ele não está funcionando como devia.

O trabalho foi implementado e testado utilizando a linguagem de programação C, as IDEs *CodeBlocks* e *Clion* e o sistema operacional *Windows 10* e *Linux*. As testes da parte A foram feitos no *Linux*.

3 Contexto

3.1 Casamento de padrões

Em Ciência da Computação, casamento de padrões é o ato de verificar a presença de um padrão em um conjunto de dados. O casamento de padrões é usado para testar se o padrão desejado encontrasse no texto em questão.

O casamento de padrões pode ser feita de forma exata (quando o padrão tem que ser encontrado no texto sem passar por nenhuma modificação) ou de forma aproximada (quando o padrão pode passar por um número K de operação de inserção, remoção ou substituição para ser encontrado no texto). O casamento exato é um subconjunto do casamento aproximado onde o padrão não precisou de nenhuma operação, ou seja, $K = 0$.

3.2 O sistema

Os problemas A e B apresentados na especificação do trabalho consistem em implementar funções de casamento exato de padrão e fazer o teste desses algoritmos. O programa criado possui um total de 3 arquivos, sendo eles um arquivo 'main.c', responsável por dar início a aplicação exibindo um menu personalizado, e os outros dois arquivos resumem-se a um TAD casamentoExato, sendo um arquivo de extensão '.h' e um '.c'. No arquivo 'casamentoExato.c' encontra-se a implementação das funções responsáveis pela resolução do problema proposto.

O problema C demonstrados na especificação do trabalho disposto no PVAnet, consiste na implementação do algoritmo Shift-And para casamento aproximado e após feita a impletação fazer testes para tipos de entradas diferentes. O algoritmo está contido em um total de 3 arquivos, onde se encontrar o 'main.c', responsável por apresentar ao usuário uma interface personalizada e os dois outros arquivos que se resumem a uma estrutura de dados 'ParteC.h' (nela esta contida o TAD para casamento aproximado do shift-and) e o arquivo 'ParteC.c' (responsável por implementar as funções contidas no arquivo ponto H).

4 Desenvolvimento do programa

4.1 Parte A e B

4.1.1 Arquivo 'partea.h'

Nesse arquivo foi feita a definição do 'modoDebug', a implementação das estruturas de dados utilizadas e o escopo das funções que foram implementadas.

O 'modoDebug' é uma variável definida no início do programa e utilizada para configurar se o programa irá executar em modo de análise ou não. Ela foi definido como 1 e será comparada com a opção escolhida pelo usuário. Caso as duas sejam iguais, o programa será executado com o modo análise ativado e irá retornar para o usuário o tamanho do texto e o tempo utilizado na busca do padrão. Além disso, também será apresentado o número de padrões encontrados e onde eles foram encontrados - essa última parte também é informado quando o modo análise não está ativo.

As estruturas criadas e usadas no programa podem ser vistas no algoritmo 1 apresentado abaixo.

Código 1: Estruturas utilizadas

```
1 typedef struct Texto{
2     char *caracteres;
3     long int tamanho;
4 }TipoTexto;
5
6 typedef struct Padrao{
7     char *palavraPadrao;
8     long int qtdOcorrencia;
9     long int *posicaoOcorrencias;
10    long int tamanhoPadrao;
11 }TipoPadrao;
12
13 typedef struct Analise{
14     double tempo;
15 }TipoAnalise;
```

A estrutura 'Texto' foi usada para guardar as informações sobre o texto. Nela foram salvos os dados lidos no arquivo ('*caracteres') e a quantidade de caracteres lidos ('tamanho').

Também foi necessária a criação das estruturas 'Padrao' (para salvar a palavra que seria buscada, a quantidade de vezes e as posições que o padrão apareceu e a quantidade de caracteres do padrão) e 'Analise' (que tem a variável usada para guardar o tempo utilizado na pesquisa).

4.1.2 Arquivo 'parte.c'

4.1.2.1 Função 'forcaBruta();'

A função 'forcaBruta();' é a responsável por resolver o problema utilizando força bruta. Ela compara carácter por carácter para fazer a comparação.

Logo abaixo está o código que executa exatamente o que foi supracitado.

Código 2: Função 'forcaBruta();'

```
1  int forcaBruta(TipoTexto texto, TipoPadrao *padrao){
2      long int i, j, k, o = 0;
3
4      padrao->qtdOcorrencia = 0;
5      padrao->tamanhoPadrao = strlen(padrao->palavraPadrao);
6
7      for (i = 0; i <= (texto.tamanho - padrao->tamanhoPadrao + 1)
8          ; i++){
9          j = 0; k = i;
10
11         while (toupper(texto.caracteres[k]) == toupper(padrao->
12             palavraPadrao[j])){
13             j++; k++;
14         }
15
16         if (j == (padrao->tamanhoPadrao)){
17             padrao->posicaoOcorrencias[o] = i;
18             padrao->qtdOcorrencia ++;
19             o++;
20         }
21     }
22
23     if (padrao->qtdOcorrencia == 0){
24         return 0;
25     } else{
26         return 1;
27     }
28 }
```

4.1.2.2 Função 'BMHS();'

Essa função utiliza o conceito de janela deslizando para fazer a comparação dos caracteres.

Primeiro é feito um pre-processamento do padrão para obter a tabela de deslocamento. Nessa fase, todas as entradas da tabela recebe o tamanho do padrão + 1. A seguir, um anel variando de 1 até o tamanho do padrão é utilizado para obter os outros valores da tabela utilizando os primeiros caracteres do padrão.

A fase de pesquisa é constituída por dois anéis aninhados. O primeiro varia 'i' até o tamanho do texto e o segundo faz a comparação entre o carácter do padrão e o do texto e, enquanto a comparação for verdadeira, 'k' e 'j' são decrementados. Quando o **while** retornar falso é feita a comparação entre 'j' e 0 e se for verdade significa que o padrão foi encontrado no texto, então o vetor 'posicaoOcorrencias[]' salva a posição que padrão foi encontrado. Ao final do primeiro anel é feito o calculo do deslocamento da janela, onde a variável 'i' recebe 'i' mais o valor salvo na tabela de deslocamento na posição 'texto.caractere[i]'

O código 3 mostra a implementação da função 'BMHS();'.

Código 3: Função 'BMHS();'

```

1  int BMHS(TipoTexto texto , TipoPadrao *padrao){
2      long int i , j , k , d[MAXTAM + 1] , o = 0;
3
4      padrao->tamanhoPadrao = strlen(padrao->palavraPadrao);
5      padrao->qtdOcorrencia = 0;
6
7      for (j = 0; j <= MAXTAM; j++)
8          d[j] = padrao->tamanhoPadrao + 1;
9
10     for (j = 1; j <= padrao->tamanhoPadrao; j++)
11         d[padrao->palavraPadrao[j - 1]] = padrao->tamanhoPadrao
12             - j + 1;
13
14     i = padrao->tamanhoPadrao;
15
16     while (i <= texto.tamanho) {
17         k = i;
18         j = padrao->tamanhoPadrao;
19
20         while (toupper(texto.caracteres[k - 1]) == toupper(
21             padrao->palavraPadrao[j - 1]) && j > 0){
22             k--; j--;
23         }
24         if (j == 0) {
25             padrao->posicaoOcorrencias[o] = i - 1;
26             padrao->qtdOcorrencia ++;
27             o++;
28         }
29         i = i + d[texto.caracteres[i]];
30     }
31
32     if (padrao->qtdOcorrencia == 0){
33         return 0;
34     }else{
35         return 1;
36     }

```

4.1.2.3 Função 'solucao();'

Essa função tem como principal objetivo dar início a aplicação, sendo assim responsável por alocar espaço para o vetor que salvará as posições que o padrão aparece no texto e chamar as funções que irão fazer o casamento de acordo com a implementação que o usuário deseja.

É nessa função que é feita a escolha de qual função de soma será chamada de acordo com a escolha do usuário. Caso o usuário passe como opção o número "1" a função chamada é a 'forcaBruta();', caso a opção digitada for "2" a função 'BMHS();' é chamada.

Nessa função também é feita a comparação entre o modo de operação escolhido pelo usuário e o 'modoDebug' e a impressão dos retornos da função (tamanho do texto, quantidade de ocorrências, posição das ocorrências e tempo de execução).

No código 4 pode ser vista parte dessa função.

Código 4: Função 'solucao();'

```
1 void solucao(TipoTexto texto, TipoPadrao *padrao, TipoAnalise *  
  analise, int algoritmo, int modoAnalise){  
2   int verificacao, i;  
3   clock_t tempo;  
4  
5   padrao->posicaoOcorrencias = malloc(texto.tamanho * sizeof(  
     char));  
6  
7   if (algoritmo == 1){  
8       tempoInicial(&tempo);  
9       verificacao = forcaBruta(texto, padrao);  
10      analise->tempo = tempoFinalizado(tempo);  
11  
12      if (verificacao == 0){  
13          if (modoAnalise == modoDebug){  
14              printf("\n * Modo analise ativo *\n\n");  
15              printf("\n Implementacao escolhida: Forca bruta  
                ");  
16              printf("\n Nao foi encontrada nenhuma  
                ocorrencia do padrao");  
17              printf("\n Tamanho do texto: %d", texto.tamanho  
                );  
18              printf("\n Tempo utilizado na busca: %.5lf ms\n  
                \n", analise->tempo);  
19          } else{  
20              printf("\n * Modo analise desativado *\n\n");  
21              printf("\n Implementacao escolhida: Forca Bruta"  
                );  
22              printf("\n Nao foi encontrada nenhuma  
                ocorrencia do padrao \n\n");  
23          }  
24      } else{  
25          if (modoAnalise == modoDebug){
```

```

26         printf("\n * Modo analise ativo *\n\n");
27         printf("\n Implementacao escolhida: Forca Bruta"
28             );
29         printf("\n Numero de padroes encontrados: %d",
30             padrao->qtdOcorrencia);
31         printf("\n Ocorrencias dos padroes nas posicoes:
32             ");
33         for (i = 0; i < padrao->qtdOcorrencia; i++){
34             printf("%d", padrao->posicaoOcorrencias[i]);
35             if (i < padrao->qtdOcorrencia - 1)
36                 printf (" , ");
37         }
38         printf("\n Tamanho do texto: %d", texto.tamanho)
39             ;
40         printf("\n Tempo utilizado na busca: %.5lf ms\n\
41             n", analise->tempo);
42     }else{
43         printf("\n * Modo analise desativado *\n\n");
44         printf("\n Implementacao escolhida: Forca bruta"
45             );
46         printf("\n Numero de padroes encontrados: %d",
47             padrao->qtdOcorrencia);
48         printf("\n Ocorrencias dos padroes nas posicoes:
49             ");
50         for (i = 0; i < padrao->qtdOcorrencia; i++){
51             printf("%d", padrao->posicaoOcorrencias[i]);
52             if (i < padrao->qtdOcorrencia - 1)
53                 printf (" , ");
54         }
55         printf ("\n\n");
56     }
57 }
58
59 if (algoritmo == 2){
60     //continua para o outro algoritmo
61 }

```

4.1.2.4 Outras funções

Foram criadas as funções 'lerArquivo();', 'calculaTamanho();', 'imprimeTexto();', 'tempoInicial();' e 'tempoFinalizado();'.

A função 'lerArquivo();', lê o arquivo com o texto e salva os caracteres lidos em um vetor que foi alocado anteriormente pela função 'calculaTamanho();'. A função 'imprimeTexto();' também lê o arquivo, mas ao invés de salvar o que foi lido ela apenas imprime o texto na tela.

Por último, as funções 'tempoInicial();' e 'tempoFinalizado();' são utilizadas para saber qual o tempo utilizado para fazer a busca pelos casamentos.

4.1.3 Arquivo 'main.c'

Nesse arquivo foram feitos os menus utilizados pelos usuários.

No primeiro menu o usuário pode escolher entre "carregar um arquivo de dados", "imprimir arquivos" e "sair do programa".

Caso o usuário escolha a primeira opção, o programa pedirá para ele entrar com o nome do arquivo e depois o usuário será levado a um outro menu onde escolherá qual método ele deseja utilizar para fazer o casamento. Ao escolher o método, o usuário irá para o ultimo menu onde escolherá se deseja ou não ativar o 'modoDebug'.

A imagem 1 mostra o que foi descrito acima com exceção à escolha do 'modoDebug'.

```
*****
*                                     *
*                               Escolha uma das opcoes abaixo                *
*          1 - Carregar um arquivo de dados                                *
*          2 - Imprimir arquivo                                             *
*          3 - Sair do programa                                             *
*                                     *
*****
Entre com a opcao valida: 1
Digite o nome do arquivo de texto (sem a extensao): texto1
Arquivo lido com sucesso!

*****
*                                     *
*                               Escolha qual dos algoritmos deseja executar  *
*          1 - Forca bruta                                                  *
*          2 - BMHS                                                         *
*          3 - Voltar                                                        *
*          4 - Sair do programa                                             *
*                                     *
*****
Entre com a opcao valida:
```

Figura 1: Menu 1 e 2

A opção "2" permite ao usuário imprimir o texto que tem no arquivo que ele quiser. Por fim, a opção "3" termina o programa.

4.2 Parte C

4.2.1 Arquivo 'ParteC.h'

Nesse arquivo está contido as estruturas de dados utilizadas e o cabeçalho das funções que foram implementadas no ponto C. A seguir é apresentado a estrutura implementada no arquivo 'ParteC.h'.

Código 5: Estruturas para casamento aproximado

```
1 typedef struct {
2     char Caractere;
3     int *VetorBitsdoCaractere;
```

```

4  }Carac;
5
6  typedef struct {
7      Carac *VetorMascara;
8  } MascaraBits;
9
10 typedef struct {
11     int *CaracteresEspecificos;
12     int QtdCarac;
13 } VetorChar;
14
15 typedef struct {
16     int **R;
17     int **RLinha;
18     int **RAuxiliar;
19     int **RAuxiliar2;
20     int **RAuxiliar3;
21 } VetoresR;
22
23 void IniciarVetorChar(VetorChar *V, char *Palavra);
24 void IniciarMascara(MascaraBits *M, VetorChar *V, int
    QtdCharRepetidos, char *Palavra);
25 int Retorna(VetorChar *V);
26 void ShiftAnd(MascaraBits *M, VetoresR *Vet, VetorChar *V, char
    *Palavra, char *Texto);

```

A estrutura "Carac" e "MascaraBits" foram utilizadas para criar um vetor onde será armazenado todos os caracteres do Padrão definido pelo usuário no arquivo 'main' e um vetor de bits referente a cada um desses caracteres. Essa estrutura foi criada para se armazenar a Mascara de Bits que é o primeiro passo do algoritmo Shift-And. Já a estrutura "VetorChar", armazena os caracteres divergentes na palavra Padrão e a quantidade de caracteres que divergem na palavra. E por fim a estrutura "VetoresR", que é responsável por armazenar as matrizes das tabelas criadas pelo algoritmo Shift-And.

4.2.2 Arquivo 'ParteC.c'

Contém todas as funções contidas no cabeçalho do arquivo '.h' implementadas.

4.2.2.1 Função 'IniciarVetorChar()'

A função 'IniciarVetorChar();' é responsável por iniciar a estrutura de dados "VetorChar", armazenando em seu interior os caracteres divergentes da palavra Padrão. Logo a seguir está o código que executa exatamente o que foi supracitado.

Código 6: IniciarVetorChar();

```

1 void IniciarVetorChar(VetorChar *V, char *Palavra) {
2     int TamPalavra, i, j, cont = 0, k = 0;
3     char aux[56];
4     TamPalavra = strlen(Palavra);
5     strcpy(aux, Palavra);
6     for(i = 0; i < TamPalavra; i++) {
7         for(j = i+1; j < TamPalavra; j++) {
8             if(Palavra[i] == aux[j]) {
9                 aux[j] = 48;
10            }
11        }
12    }
13    for(i = 0; i < TamPalavra; i++) {
14        if(aux[i] != 48) {
15            cont++;
16        }
17    }
18    V->CaracteresEspecificos = malloc(cont * sizeof(int));
19    V->QtdCarac = cont;
20    for(i = 0; i < TamPalavra; i++) {
21        if(aux[i] != 48) {
22            V->CaracteresEspecificos[k] = aux[i];
23            k++;
24        }
25    }
26 }

```

4.2.2.2 Função 'IniciarMascara()'

Essa função faz uso das estruturas de dados "Carac" e "MascaraBits" uma tabela de pre-processamento, contendo uma mascara de bits para cada caractere da palavra Padrão. Logo a seguir está o código que executa exatamente o que foi supracitado.

Código 7: IniciarMascara();

```

1 void IniciarMascara(MascaraBits *M, VetorChar *V, int
   QtdCharRepetidos, char *Palavra) {
2     int i, j;
3     int TamPalavra;
4
5     TamPalavra = strlen(Palavra);
6     M->VetorMascara = malloc(QtdCharRepetidos * sizeof(int));
7     for(i = 0; i < QtdCharRepetidos; i++) {
8         M->VetorMascara[i].Caractere = V->CaracteresEspecificos[
           i];
9         M->VetorMascara[i].VetorBitsdoCaractere = malloc(
           TamPalavra * sizeof(int));
10    }

```

```

11
12     for(i = 0; i < QtdCharRepetidos; i++) {
13         for(j = 0; j < TamPalavra; j++) {
14             M->VetorMascara[i]. VetorBitsdoCaractere[j] = 0;
15         }
16     }
17
18     for(i = 0; i < QtdCharRepetidos; i++) {
19         for(j = 0; j < TamPalavra; j++) {
20             if(M->VetorMascara[i]. Caractere == Palavra[j]) {
21                 M->VetorMascara[i]. VetorBitsdoCaractere[j] = 1;
22             }
23         }
24     }
25
26     for(i = 0; i < QtdCharRepetidos; i++) {
27         printf("0 Caractere e: %c\n", M->VetorMascara[i].
                Caractere);
28         printf("Seu vetor de bits e: ");
29         for(j = 0; j < TamPalavra; j++) {
30             printf("%d", M->VetorMascara[i]. VetorBitsdoCaractere
                [j]);
31         }
32         printf("\n");
33     }
34 }

```

4.2.2.3 Função 'ShiftAnd()'

Essa é a função principal do algoritmo, ela é responsável por receber o texto e a palavra padrão, para assim realizar os cálculos a cerca dos bits de cada vetor R e retornar a possibilidade de ter ocorrido casamento ou não. Obs: essa função remete ao casamento exato. Logo a seguir está o código que executa exatamente o que foi supracitado.

Código 8: ShiftAnd();

```

1 void ShiftAnd(MascaraBits *M, VetoresR *Vet, VetorChar *V, char
  *Palavra, char *Texto) {
2     int k = 0;
3     int i, j, x;
4     int TamPalavra, TamTexto;
5     int aux;
6
7     TamPalavra = strlen(Palavra);
8     TamTexto = strlen(Texto);
9
10    printf("Tamanho Texto: %d\n", TamTexto);
11
12    Vet->R = (int**) malloc(TamTexto * sizeof(int*));
13    Vet->RLinha = (int**) malloc(TamTexto * sizeof(int*));
14    for(i = 0; i < TamTexto; i++) {

```

```

15     Vet->R[i] = (int*)malloc(TamPalavra * sizeof(int));
16     Vet->RLinha[i] = (int*)malloc(TamPalavra * sizeof(int));
17 }
18
19 for(i = 0; i < TamTexto; i++) {
20     for(j = 0; j < TamPalavra; j++) {
21         Vet->R[i][j] = 0;
22         Vet->RLinha[i][j] = 0;
23     }
24 }
25 Vet->R[0][0] = 1;
26
27 for(i = 0; i < TamTexto; i++) {
28     printf("Vetor RLinha e: ");
29     for(j = 0; j < TamPalavra; j++) {
30         for(x = 0; x < V->QtdCarac; x++) {
31             if(Texto[i] == M->VetorMascara[x].Caractere) {
32                 k = x;
33             }
34         }
35         if(Texto[i] == M->VetorMascara[k].Caractere) {
36             if(Vet->R[i][j] == 0 && M->VetorMascara[k].
37                 VetorBitsdoCaractere[j] == 0) {
38                 Vet->RLinha[i][j] = 0;
39             }
40             if(Vet->R[i][j] == 0 && M->VetorMascara[k].
41                 VetorBitsdoCaractere[j] == 1) {
42                 Vet->RLinha[i][j] = 0;
43             }
44             if(Vet->R[i][j] == 1 && M->VetorMascara[k].
45                 VetorBitsdoCaractere[j] == 0) {
46                 Vet->RLinha[i][j] = 0;
47             }
48             if(Vet->R[i][j] == 1 && M->VetorMascara[k].
49                 VetorBitsdoCaractere[j] == 1) {
50                 Vet->RLinha[i][j] = 1;
51             }
52             printf("%d", Vet->RLinha[i][j]);
53         } else {
54             Vet->RLinha[i][j] = 0;
55             printf("%d", Vet->RLinha[i][j]);
56         }
57     }
58     printf("\n");
59     if(i+1 < TamTexto) {
60         for(j = 0; j < TamPalavra; j++) {
61             if(Vet->RLinha[i][j] == 1) {
62                 aux = Vet->RLinha[i][j];
63                 Vet->R[i+1][j] = Vet->RLinha[i][j+1];
64                 Vet->R[i+1][j+1] = aux;
65             }
66         }
67         Vet->R[i+1][0] = 1;
68         printf("Vetor R e: ");

```

```

65         for(j = 0; j < TamPalavra; j++) {
66             printf("%d", Vet->R[i+1][j]);
67         }
68         printf("\n");
69     }
70     if(Vet->RLinha[i][TamPalavra-1] == 1) {
71         printf("Casou!");
72     }
73 }
74 }

```

4.2.2.4 Função 'ShiftAndAproximado()'

Assim como a função 'ShiftAnd();' supracitada, essa tem o objetivo de realizar os mesmo calculos, porém permitindo que o usuário entre com a quantidade de operações a serem realizadas (Substituição, Remoção e Inserção) e a variável K que representa o número limite de operações a serem realizadas. Logo a seguir está o código que executa exatamente o que foi supracitado.

Código 9: ShiftAndAproximado();

```

1 void ShiftAndAproximado(MascaraBits *M, VetoresR *Vet, VetorChar
   *V, char *Palavra, char *Texto, int PadraoK) {
2     int k = 0;
3     int i, j, x, s, z;
4     int TamPalavra, TamTexto;
5     int aux, Diminuir = 0, Diminuir2 = 0;
6
7     TamPalavra = strlen(Palavra);
8     TamTexto = strlen(Texto);
9
10    printf("Tamanho Texto: %d\n", TamTexto);
11
12    Vet->R = (int**) malloc(TamTexto * sizeof(int*));
13    Vet->RLinha = (int**) malloc(TamTexto * sizeof(int*));
14    Vet->RAuxiliar = (int**) malloc(TamTexto * sizeof(int*));
15    Vet->RAuxiliar2 = (int**) malloc(TamTexto * sizeof(int*));
16    Vet->RAuxiliar3 = (int**) malloc(TamTexto * sizeof(int*));
17    for(i = 0; i < TamTexto; i++) {
18        Vet->R[i] = (int*) malloc(TamPalavra * sizeof(int));
19        Vet->RLinha[i] = (int*) malloc(TamPalavra * sizeof(int));
20        Vet->RAuxiliar[i] = (int*) malloc(TamPalavra * sizeof(int));
21        Vet->RAuxiliar2[i] = (int*) malloc(TamPalavra * sizeof(int));
22        Vet->RAuxiliar3[i] = (int*) malloc(TamPalavra * sizeof(int));
23    }
24
25    for(i = 0; i < TamTexto; i++) {

```



```

26         for(j = 0; j < TamPalavra; j++) {
27             Vet->R[i][j] = 0;
28             Vet->RLinha[i][j] = 0;
29             Vet->RAuxiliar[i][j] = 0;
30             Vet->RAuxiliar2[i][j] = 0;
31             Vet->RAuxiliar3[i][j] = 0;
32         }
33     }
34     Vet->R[0][0] = 1;
35
36     for(z = 0; z < PadraoK; z++) {
37         if(z > 0) {
38             for(j = 0; j < TamPalavra; j++) { // Faz Raux1 = R0
39                 >>1
40                 if(Vet->R[z-1][j] == 1) {
41                     aux = Vet->R[z-1][j];
42                     Vet->RAuxiliar[z-1][j] = Vet->R[z-1][j+1];
43                     Vet->RAuxiliar[z-1][j+1] = aux;
44                 }
45             }
46             for(i = 0; i < TamTexto; i++) {
47                 for(j = 0; j < TamPalavra; j++) { // Faz Raux2 =
48                     R0>>1
49                     if(Vet->R[i][j] == 1) {
50                         aux = Vet->R[i][j];
51                         Vet->RAuxiliar2[i][j] = Vet->R[i][j+1];
52                         Vet->RAuxiliar2[i][j+1] = aux;
53                     }
54                 }
55                 for(j = 0; j < TamPalavra; j++) { // Faz Raux3 =
56                     RLinha0>>1
57                     if(Vet->RLinha[i][j] == 1) {
58                         aux = Vet->RLinha[i][j];
59                         Vet->RAuxiliar2[i][j] = Vet->RLinha[i][j
60                             +1];
61                         Vet->RAuxiliar2[i][j+1] = aux;
62                     }
63                 }
64             }
65             for(j = 0; j < TamPalavra; j++) {
66                 for(x = 0; x < V->QtdCarac; x++) {
67                     if(Texto[i] == M->VetorMascara[x].
68                         Caractere) {
69                         k = x;
70                     }
71                 }
72                 if(Texto[i] == M->VetorMascara[k]. Caractere)
73                     { // ((R1 >> 1) & M[T[i]])
74                     if(Vet->RAuxiliar[i][j] == 0 && M->
75                         VetorMascara[k]. VetorBitsdoCaractere [
76                             j] == 0) {
77                         Vet->RLinha[i][j] = 0;
78                     }
79                     if(Vet->RAuxiliar[i][j] == 0 && M->
80                         VetorMascara[k]. VetorBitsdoCaractere [

```

```

71         j] == 1) {
72             Vet->RLinha[i][j] = 0;
73         }
74         if (Vet->RAuxiliar[i][j] == 1 && M->
75             VetorMascara[k].VetorBitsdoCaractere[
76                 j] == 0) {
77             Vet->RLinha[i][j] = 0;
78         }
79         if (Vet->RAuxiliar[i][j] == 1 && M->
80             VetorMascara[k].VetorBitsdoCaractere[
81                 j] == 1) {
82             Vet->RLinha[i][j] = 1;
83         }
84         // ((R1 >> 1) & M[T[i]]) | R0
85         if (Vet->RLinha[i][j] == 0 && Vet->R[i][j]
86             == 0) {
87             Vet->RLinha[i][j] = 0;
88         }
89         if (Vet->RLinha[i][j] == 1 && Vet->R[i][j]
90             == 0) {
91             Vet->RLinha[i][j] = 1;
92         }
93         if (Vet->RLinha[i][j] == 1 && Vet->R[i][j]
94             == 1) {
95             Vet->RLinha[i][j] = 1;
96         }
97         if (Vet->RLinha[i][j] == 1 && Vet->R[i][j]
98             == 1) {
99             Vet->RLinha[i][j] = 1;
100         }
101         // ((R1 >> 1) & M[T[i]]) | R0 | (R0 >>
102             1)
103         if (Vet->RLinha[i][j] == 0 && Vet->
104             RAuxiliar2[i][j] == 0) {
105             Vet->RLinha[i][j] = 0;
106         }
107         if (Vet->RLinha[i][j] == 1 && Vet->
108             RAuxiliar2[i][j] == 0) {
109             Vet->RLinha[i][j] = 1;
110         }
111         if (Vet->RLinha[i][j] == 0 && Vet->
112             RAuxiliar2[i][j] == 1) {
113             Vet->RLinha[i][j] = 1;
114         }
115         if (Vet->RLinha[i][j] == 1 && Vet->
116             RAuxiliar2[i][j] == 1) {
117             Vet->RLinha[i][j] = 1;
118         }
119         // ((R1 >> 1) & M[T[i]]) | R0 | (R0 >>
120             1) | (R'0 >> 1)
121         if (Vet->RLinha[i][j] == 0 && Vet->
122             RAuxiliar3[i][j] == 0) {
123             Vet->RLinha[i][j] = 0;
124         }
125     }

```

```

109         if (Vet->RLinha[i][j] == 1 && Vet->
110             RAuxiliar3[i][j] == 0) {
111             Vet->RLinha[i][j] = 1;
112         }
113         if (Vet->RLinha[i][j] == 0 && Vet->
114             RAuxiliar3[i][j] == 1) {
115             Vet->RLinha[i][j] = 1;
116         }
117         if (Vet->RLinha[i][j] == 1 && Vet->
118             RAuxiliar3[i][j] == 1) {
119             Vet->RLinha[i][j] = 1;
120         }
121         // ((R1 >> 1) & M[T[i]]) | R0 | (R0 >>
122             1) | (R'0 >> 1) | 10 m 1
123         if (Vet->RLinha[i][0] == 0) {
124             Vet->RLinha[i][0] = 1;
125         }
126     } else {
127         Vet->RLinha[i][j] = 0;
128     }
129 }

```

4.2.3 Arquivo 'main.c'

Nesse arquivo foi implementado a interface com o usuário, onde o usuário pode escolher qual Shif-And ele quer utilizar, sendo eles Shift-And Exato e Aproximado. Logo após feito a escolha é necessário que o usuário informe o texto e o padrão para o caso do Shift-And Exato e o texto, o padrão, o K e quais operações serão realizadas. A imagem 2 mostra o que foi descrito acima.

```
C:\Users\Samuel\CLionProjects\ParteC\cmake-build-debug\ParteC.exe

*****
*
*                               *
*          Escolha uma das opcoes abaixo          *
*          1 - Algoritmo Shift-And Exato           *
*          2 - Algoritmo Shift-And Aproximado      *
*          3 - Sair                                *
*
*****

Entre com a opcao valida:1
  Entre com a palavra Padrao:
teste
Entre com o Texto:
teste testando|
```

Figura 2: Menu Shift-And

5 Testes

5.1 Parte A e B

Os testes foram realizados em cima de 9 textos diferentes.

Para calcular o tempo utilizado na execução do programa, foi utilizado as funções de tempo. A função que marcava o início da contagem do tempo foi inserida antes de chamar a função responsável pelo casamento e a função que retorna o tempo final foi inserida depois da função.

5.1.1 Resultados

Os resultados de cada teste foram inseridos em tabelas e depois foi gerado um gráfico referente ao tempo utilizado em cada uma das implementações. O gráfico permite uma melhor comparação entre os métodos.

A tabela e o gráfico gerado podem ser vistos a seguir. Os dados apresentados estão em mili segundos.

Na figura 3 e no gráfico 4 pode ser visto os resultados obtidos nas duas implementações. O gráfico foi gerado com os dados dessa tabela.

Tempo Total (ms)			
Padrão	Tamanho	Força Bruta	BMHS
rosa	41	0,003	0,005
texto	62	0,003	0,004
casa	84	0,003	0,005
algoritmo	1037	0,016	0,009
programacao	1391	0,017	0,007
implementa	1622	0,019	0,01
trabalho	1756	0,032	0,009
tsunami	1975	0,044	0,009
piramide	18816	0,186	0,063

Figura 3: Tabela de comparação

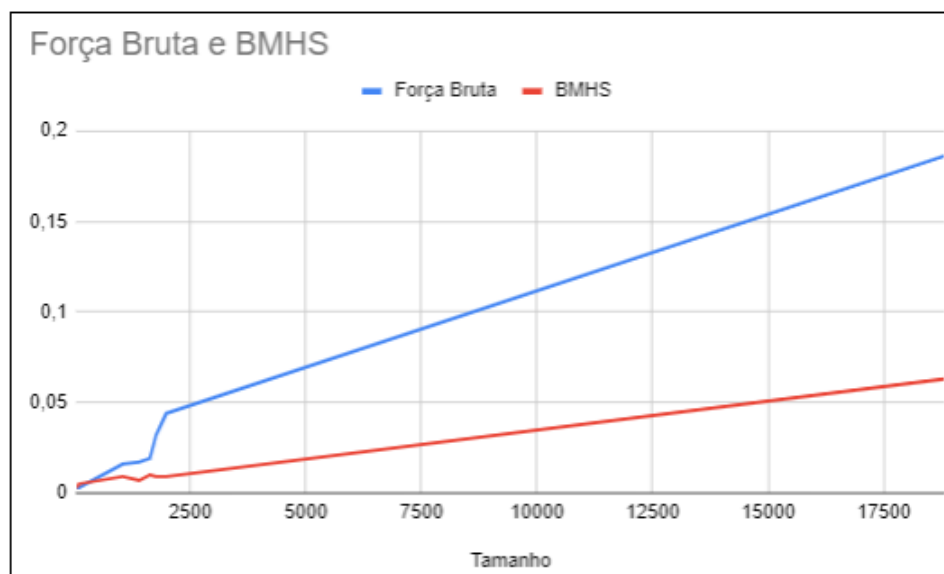


Figura 4: Tamanho da entrada X Tempo total

Observando a tabela e o gráfico verificamos que para entradas muito pequenas o algoritmo de força bruta é melhor. Mas para entradas maiores do que 1037 o algoritmo BMHS se tornou melhor e cada vez que a entrada aumentava mais, esse ultimo algoritmo foi se mostrando cada vez mais eficiente que o de força bruta.

Durante os testes, percebeu-se que o tempo utilizado variava dependendo

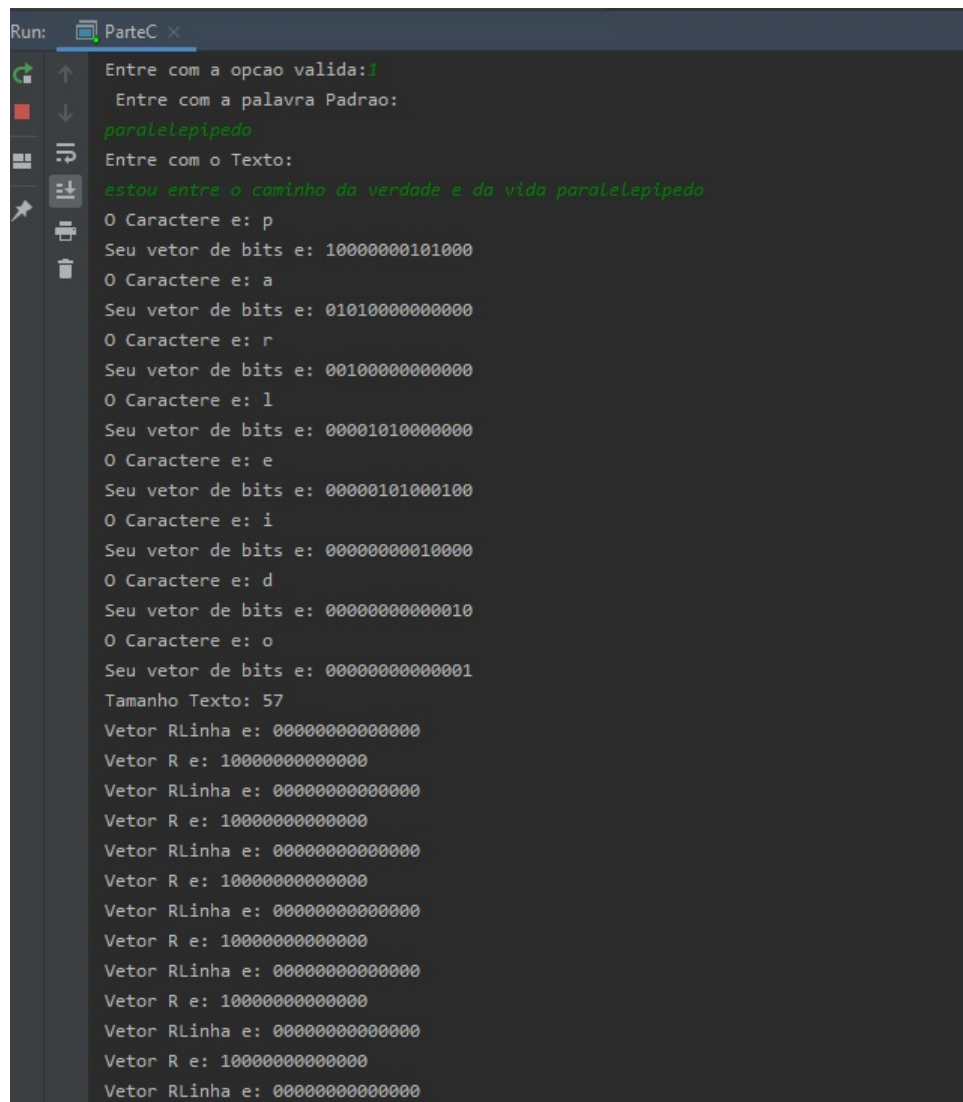
da quantidade de coisas que o computador estava fazendo ao mesmo tempo.

5.2 Parte C

5.2.1 Resultados

Os testes foram realizados em cima do algoritmo de Shift-And Exato, já que houve problemas de memória ao utilizar o Aproximado, já que a implementação foi feita de uma forma muito custosa.

A imagem 5 mostra o algoritmo em seu funcionamento.



```
Run: ParteC x
Entre com a opcao valida:1
Entre com a palavra Padrao:
paralelepipedo
Entre com o Texto:
estou entre o caminho da verdade e da vida paralelepipedo
O Caractere e: p
Seu vetor de bits e: 10000000101000
O Caractere e: a
Seu vetor de bits e: 01010000000000
O Caractere e: r
Seu vetor de bits e: 00100000000000
O Caractere e: l
Seu vetor de bits e: 00001010000000
O Caractere e: e
Seu vetor de bits e: 00000101000100
O Caractere e: i
Seu vetor de bits e: 00000000010000
O Caractere e: d
Seu vetor de bits e: 00000000000010
O Caractere e: o
Seu vetor de bits e: 00000000000001
Tamanho Texto: 57
Vetor RLinha e: 00000000000000
Vetor R e: 10000000000000
Vetor RLinha e: 00000000000000
Vetor R e: 10000000000000
Vetor RLinha e: 00000000000000
Vetor R e: 10000000000000
Vetor RLinha e: 00000000000000
Vetor R e: 10000000000000
Vetor RLinha e: 00000000000000
Vetor R e: 10000000000000
Vetor RLinha e: 00000000000000
Vetor R e: 10000000000000
Vetor RLinha e: 00000000000000
```

Figura 5: Funcionamento Shift-And

6 Conclusão

Com o trabalho também podemos aumentar nossos conhecimentos acerca dos problemas apresentados, obtendo um maior conhecimento sobre as técnicas de casamento aproximado e exato.

Referências

- [1] Wikipédia. *Casamento de padrões*. https://pt.wikipedia.org/wiki/Casamento_de_padr