UNIVERSIDADE FEDERAL DO AMAZONAS – UFAM

INSTITUTO DE COMPUTAÇÃO

FT05 – ENGENHARIA DA COMPUTAÇÃO

Algoritmos e estruturas de dados II

IEC013

Manaus-AM

2020

21951799 – Vinícius Patrício Medeiros da Silva

Árvores patrícia

Relatório de algoritmo apresentado junto ao curso de Engenharia da Computação como requisito à obtenção de nota parcial para o período especial 2020.

Professor: Edson Nascimento

Manaus-AM

2020

1-INTRODUÇÃO

O trabalho consiste em implementar um programa que leia um texto qualquer, no caso um arquivo no formato .txt, e imprima na ordem alfabética, todas as palavras com 3 ou mais caracteres e a(s) linha(s) na(s) qual(is) elas aparecem no texto.

A leitura não deverá ser feita de qualquer jeito, é necessário que seja desprezado os espaços em branco como também sinais de pontuação. Além disso, a leitura deve converter todas as letras maiúsculas para minúsculas.

Para propor um programa com essa finalidade, é necessário a implementação a estrutura árvore patrícia. A cada palavra lida, o algoritmo deve pesquisar à árvore para verificar se a palavra já está presente. Caso já esteja presente, é necessário adicionar o numero da linha à lista de linhas dessa palavra. Se não estiver na árvore patrícia, é necessário criar um novo nó na árvore e iniciar a lista de linhas junto a palavra lida.

A árvore patrícia é uma representação mais compacta de uma trie onde os nós que teria um único filho são agrupados nos seus antecessores, tornando-a mais indicada a esse tipo de problema, sua estrutura é bastante dinâmica.

Essa estrutura de dados foi criada por Donald Morrison, nada mais é quem um algoritmo com a finalidade de buscas em árvores com conteúdos sendo palavras e nós binários, armazenando conteúdo apenas nos nós folhas.

2-IMPLEMENTAÇÃO

2.1 DEFINIÇÕES E VARIÁVEIS GLOBAIS

- **Definições**

```
#define true 1 // verdadeiro
#define false 0 // falso
```

Essas definições foram dadas apenas para ficar mais claro algumas condições que foram usadas no código, ao invés de comparar com 0 e 1, fica bem mais claro por exemplo igualarmos uma sentença a true que igualar 1, ou seja, foram utilizadas para ficar mais fácil o entendimento do código.

- **Variáveis globais**

```
int linha = 1; // linha vai ser incrementada assim que estivermos no final da linha lida
```

A variável linha irá indicar que linha está sendo efeituada a leitura, a cada quebra de linha do arquivo a variável linha será incrementada.

2.2 ESTRUTURAS

- **Lista**

Lista e carga:

```
typedef struct carga TELE;
struct carga{
    int carga;
    struct carga *prox;
};

typedef struct lista TLista;
struct lista{
    int tamanho;
    TELE *inicio;
    TELE *fim;
};
```

Foi definido a estrutura lista com a finalidade de armazenar o número da linha onde a palavra aparece, como isso pode variar de texto para texto, foi necessário trabalhar exatamente com lista por tratar-se de uma estrutura dinâmica. No caso do trabalho, não foi necessário trabalhar com listas neutras, uma vez que precisamos apenas armazenar um inteiro. O struct carga vai ser um elemento da lista, ele vai armazenar um inteiro carga (no caso vamos colocar a linha) e o endereço do próximo elemento da lista.

Criar Lista:

```
TLista *criarLista(){
    TLista *lista = malloc(sizeof(TLista));
    lista->tamanho = 0;
    lista->inicio = NULL;
    lista->fim = NULL;

    return lista;
}
```

A função criarLista irá alocar na memória para estrutura do tipo struct lista.

Criar elemento da lista:

```
TELE *criarElemento(int elemento){

    TELE *carga_nova = malloc(sizeof(TELE));
    carga_nova->carga = elemento;
    carga_nova->prox = NULL;

    return carga_nova;
}
```

criarElemento irá criar o elemento da lista dado um número inteiro, no caso do trabalho, essa entrada vai ser a linha onde a palavra se encontra.

Inserir na lista um novo elemento:

```
void inserirLista(TLista *lista, TELE *carga){
    if(lista->tamanho == 0){
        lista->inicio = carga;
        lista->fim =  carga;

    }else{
        lista->fim->prox = carga;
        lista->fim = carga;
    }
    lista->tamanho ++;
}
```

Dado um elemento do tipo TELE (struct carga) e uma lista, essa função irá inserir esse elemento sempre no fim da lista.

Verifica se a lista é vazia:

```
int listaVazia(TLista *lista){
    return lista->tamanho == 0 ? 1 : 0;
}
```

Essa função irá receber uma lista e retornará 1 caso ela não tenha elementos e retornará 0 caso ela tenha.

Imprimir lista:

```
void imprimirLista(TLista *lista){

    TELE *caminhador = lista->inicio;

    while(caminhador != NULL){
        printf("%d ",caminhador->carga);
        caminhador = caminhador->prox;
    }
}
```

Essa função irá imprimir a carga dos elementos da lista. Para o trabalho, ela será usada para imprimir as linhas onde a palavra se encontra no texto.

Obs: A estrutura de lista foi implementada na lista.c e foi criado a lista.h, esta vai ser usada nó código principal (patricia.c).

- **Árvore patrícia**

Estrutura da Árvore patrícia:

```
typedef struct no_patricia PNO;
struct no_patricia{

   int pos_caracter;  // posição do caractere a ser comparado
   TLista *linhas;    // uma lista que guarda em que linhas encontra-se as palavras
   char palavra[20];  // palavra que o nó/raiz irá armazenar
   PNO *filho_esq;
   PNO *filho_dir;

};
```

A variável pos_caracter irá armazenar a posição onde deverá ser comparo as palavras na hora da inserção, no caso dessa implementação optou-se de utilizar um inteiro que diz a posição da comparação ao invés de usar um char que armazena o caractere. Linhas irá armazenar a linha onde a palavra ocorreu no texto, o vetor palavra irá armazenar a palavra daquele nó específico, e por se tratar de uma árvore binária, a estrutura de cada nó terá dois filhos.

Criar raiz:

```
PNO *criarRaiz(int pos_char, char palavra[20], int tamanho){

   PNO *nova_raiz = malloc(sizeof(PNO));
   nova_raiz->pos_caracter = pos_char;
   nova_raiz->linhas = criarLista();
   strncpy(nova_raiz->palavra, palavra,tamanho);


   nova_raiz->filho_esq = NULL;
   nova_raiz->filho_dir = NULL;

   return nova_raiz;
}
```

A função irá alocar na memória para um ponteiro de uma estrutura do tipo PNO, ela recebe a posição de onde deve ocorrer a comparação, recebe a palavra que deve ser inserida, e o tamanho da palavra. Por ser um novo raiz/nó seus filhos recebem NULL.

2.3 FUNÇÕES AUXIARES

Tamanho da menor string:

```
int tamanhoMenorString(char palavra1[20], char palavra2[20]){
    int menor = strlen(palavra1);

    if(menor > strlen(palavra2)){
        menor = strlen(palavra2);
    }
    return menor;
}
```

A função retorna o tamanho da menor string dada duas strings.

Último caractere em comum:

```
int ultimoCharComum(char palavra1[20], char palavra2[20]){

    int menor = tamanhoMenorString(palavra1,palavra2); // saber até onde i (contador) ser ic
rementado
    int i = 0; // contador de caracteres iguais
    while(palavra1[i] == palavra2[i] && (i < menor)){
        i++;
    }
    if( i == menor){ // uma das palavras é sub-palavra da outra
        return -1;
    }
    i-= 1;
    if(i < 0){ // não tem nenhum char em comum
        i = 0;
    }
    return i;
}
```

Retorna a posição do último caractere em comum entre duas strings, caso uma seja uma sub-palavra (sub-string) da outra, deve retornar -1, caso não tenha nada em comum, retornará 0, em nenhum desses casos, irá retornar à posição do último caractere em comum.

Primeiro caractere distinto:

```
int primeiroCharDistinto(char palavra1[20], char palavra2[20]){

   int menor = tamanhoMenorString(palavra1,palavra2); // menor foi criado para saber qual vai ser o limite do icremento do contador;
   int contador = 0; // contador de caracteres iguais

   // o contador irá ser icrementado enquanto os caracteres das strings forem iguais e ele ser menor que o tamanho da menor string
   while(palavra1[contador] == palavra2[contador] && contador < menor){
      contador ++;
   }
   return contador;
}
```

A função irá retorna o índice do primeiro caractere diferente entre dias strings.

Menor string a partir de um caractere:

```
char *menorStringChar(char palavra1[20], char palavra2[20]){
   int aux = primeiroCharDistinto(palavra1,palavra2);
   return (palavra1[aux] < palavra2[aux])? palavra1 : palavra2;

}
```

A função irá informar a menor string entre duas strings dada a posição onde ocorre a diferença de caractere.

Maior string a partir de um caractere:

```
char *maiorStringChar(char palavra1[20], char palavra2[20]){
   int aux = primeiroCharDistinto(palavra1,palavra2);
   return (palavra1[aux]>palavra2[aux])? palavra1 : palavra2;
}
```

A função irá informar a maior string entre duas strings dada a posição onde ocorre a diferença de caractere.

Filho menor:

```
PNO *filhoMenor(PNO *filho_esq, PNO *filho_dir){
    char* aux = menorStringChar(filho_esq->palavra,filho_dir->palavra);
    if(strcmp(filho_esq->palavra,aux)==0){
        return filho_esq;
    }else{
        return filho_dir;
    }
}
```

A função irá retorna o nó com a menor string.

Filho maior:

```
PNO *filhoMaior(PNO *filho_esq, PNO *filho_dir){
    char* aux = maiorStringChar(filho_dir->palavra,filho_esq->palavra);
    return strcmp(filho_esq->palavra,aux) == 0 ? filho_esq : filho_dir;
}
```

Será dado o nó com a maior string.

Folha:

```
int folha(PNO *raiz){
    if( (raiz->filho_dir==NULL) &&( raiz-
>filho_esq==NULL)){ // caso  o nó não tenha filhos, então ele é uma folha
        return true;
    }else{
        return false;
    }
}
```

Folha irá retornar 1 caso a raiz passado para função for uma folha e 0 caso não seja.

Imprimir Arvore:

```
void imprimirArvore(PNO *raiz){
   setlocale(LC_ALL, "Portuguese_Brazil");
   if(raiz != NULL){
      imprimirArvore(raiz->filho_esq);
      if(strlen(raiz->palavra)>=3 && listaVazia(raiz->linhas) == 0){
         printf("%s ",raiz->palavra);
         imprimirLista(raiz->linhas);
         printf("\n");
      }
     imprimirArvore(raiz->filho_dir);
   }

}
```

ImprimirArvore imprimi a árvore passado na chamada da função.

Número de pontuações:

```
int numeroPontuacao(char palavra[20]){
   int tamanho_palavra = strlen(palavra);
   int contador = 0;
   for(int i = 0; i<tamanho_palavra; i++){
      if (palavra[i] == '.' || palavra[i] == ',' || palavra[i] == '!' || palavra[i] == '?' || palavra[i] == '"'
){
         contador++;
      }
   }
   return contador;
}
```

A função irá informar quantas pontuações a string tem.

Caractere pontuação:

```
int charPontuacao(char caracter){
    if(caracter == '.' || caracter == ',' || caracter == '!' || caracter
=='?' || caracter == '"'){
        return true;
    }
    return false;
}
```

A função verifica se a caractere passado é uma pontuação.

Tirar pontuação:

```
char *tirarPontuacao(char palavra[20]){
    int tamanho_palavra = strlen(palavra);
    int i = 0;
    int num = 1;
    int numero_pontuacao = numeroPontuacao(palavra);
    if(charPontuacao(palavra[0]) == true){  // verifica se o inicio tem a
spas
        for(int j = 0;  j < tamanho_palavra-1;j++){
            palavra[j] = palavra[j+1];
        }
        palavra[tamanho_palavra-1] = '\0';
        i++;
        while (i < numero_pontuacao) // irá fazer as remoções das pontuaç
ões que são armazenadas no final da string caso tenha
        {
            if(charPontuacao(palavra[strlen(palavra)-i])==true){
                palavra[strlen(palavra)-i] = '\0';
            }
            i++;
        }

    }else{ // caso não tenha pontuação no inicio, irá remover caso tenha
no final
        while(i < numero_pontuacao){
            if(charPontuacao(palavra[strlen(palavra)-i-1]) == true){
                palavra[strlen(palavra)-i-1] = '\0';
            }
            i++;
        }

    }
    return palavra;
}
```

tirarPontuação retira as pontuações de uma string, a primeira condição verifica se possui uma pontuação no inicio da string, por exemplo uma string dada por “carro”, ele vai verificar que essa string tem duas pontuações, vai verificar que possui uma no início, ela vai ser retirada, caso tenha mais, ele vai sair retirando as pontuações do final, caso só tenha no final, será retirada as pontuações do final.

2.4 FUNÇÕES PRINCIPAIS

Lado inserção:

```
int ladoInsercao(PNO *raiz, char palavra_nova[20]){

    int aux = primeiroCharDistinto(raiz->palavra,palavra_nova);
    int menor = tamanhoMenorString(raiz->palavra,palavra_nova);

    if(aux>menor-1){  // caso de uma string ser sub-string da outra
        int menorPalavra1 = primeiroCharDistinto(palavra_nova,raiz-
>filho_esq->palavra);
        int menorPalavra2 = primeiroCharDistinto(palavra_nova,raiz-
>filho_dir->palavra);
        if(menorPalavra2>menorPalavra1){ // a palavra guardada no filho d
ireto tem mais caracteres em comum com a nova palavra a ser inserida
            return 2;
        }else if (menorPalavra2 == menorPalavra1){ // caso tenha a mesma
quantidade de caracteres em comum com as palavras do os dois filhos do nó
            // irá verificar onde deve ser inserido a partir da posição o
nde ocorre a diferença de de caracter
            if(raiz->filho_dir-
>palavra[menorPalavra2] > palavra_nova[menorPalavra2]){
                return 1;
            }else{
                return 2;
            }
        }else{ // a palavra guardada no filho esq tem mais caracteres em
comum com a nova palavra a ser inserida
            return 1;
        }
    } // caso não seja uma sub-string
    if(palavra_nova[aux] > raiz->palavra[aux]){
        return 2;
    }else{
        return 1;
    }
}
```

A função irá retornar onde deve ser inserido a nova palavra dada uma raiz, para isso ela retornará 1 caso for necessário inserir no filho esquerdo do nó e retornará 2 caso for inserir no filho direito do nó.

Busca palavra:

```
int buscaPalavra(PNO *raiz, char palavra[20], int flag){

    if(strcmp(palavra,raiz-
>palavra) == 0){ // casso a palavra seja igual a palavra armazenada no nó
/raiz
        flag = true;
    }else{
        if( folha(raiz) == false){ // caso não seja folha, ele irá ficar
percorrendo pelo filhos que estão a esquerda
            flag = buscaPalavra(raiz->filho_esq,palavra,flag);
            if(flag == 0){ // não foi nada encontrado nos filhos da esque
rda, agora irá percorrer pelos filhos a direita
                flag = buscaPalavra(raiz->filho_dir,palavra,flag);
            }
        }else{ // caso não seja encontrado nada
            flag = 0;
        }
    }
    return flag;
}
```

BuscaPalavra irá busca na raiz árvore se a palavra passada no parâmetro da função está na árvore, caso esteja irá retornar 1, caso contrário irá retornar 0.

Atualizar linha:

```
PNO *atualizarLinha(PNO *raiz, char palavra[20]){
    if(strcmp(palavra,raiz-
>palavra) == 0){ // se raiz/nó tenha tenha a string, irá ser atualizado a
 lista de linhas
        TELE *carga_nova = criarElemento(linha);
        inserirLista(raiz->linhas,carga_nova);
        return raiz;
    }else{
        if(folha(raiz) == false){ // irá percorre os filhos caso a raiz/n
ó não forem folhas;
           raiz->filho_esq = atualizarLinha(raiz->filho_esq,palavra);
           raiz->filho_dir = atualizarLinha(raiz->filho_dir,palavra);
        }else{ // caso a raiz/nó não for folha, não há o que se atualizar
, uma vez que eles não armazenam palavras
            return raiz;
        }
    }
    return raiz;
}
```

A função irá verificar se a string está na árvore, caso esteja vai ser atualizado a lista de linhas, essa atualização será dada inserindo a linha atual que está sendo verificada na lista. Por isso, tornamos a linha como uma variável global.

Inserir Árvore:

```
PNO *inserirArvore(PNO *raiz_pai, PNO *raiz, char palavra_nova[20]){
    if(palavra_nova[0]=='*'){ // caso seja um asterisco, é para pular a l
inha
        linha++;
        return raiz;
    }
    // caso a raiz seja nula;
    if(raiz == NULL){

        raiz = criarRaiz(0,palavra_nova,strlen(palavra_nova));
        TELE *carga = criarElemento(linha);
        inserirLista(raiz->linhas, carga);
    // raiz != NULL
    }else{
        // primeiro será verificado se a palavra já não está na árvore
        int flag = buscaPalavra(raiz,palavra_nova,flag);
        if(flag == 1){ // se estiver na árvore, será atualizado lista de
linhas com essa palavra associada
            raiz = atualizarLinha(raiz, palavra_nova);
            return raiz;
        // caso ela não esteja
        }else{
                if(folha(raiz) == true){ // caso se o nó/raiz for uma fol
ha

                // sera obtido a posicao do ultimo caracter em comum com
a palavra nova e com a palavra da raiz
                int pos_caracter;
                pos_caracter = ultimoCharComum(raiz-
>palavra,palavra_nova);

                char caracterString[20]={""};
                strncpy(caracterString, palavra_nova, pos_caracter + 1);


                if(raiz_pai != NULL){
                    // Se sub-string criada for igual a sub-
string do pai, vai ser obtida uma nova sub-string
```

```
                    if(strcmp(caracterString,raiz_pai->palavra) == 0){




                     pos_caracter = primeiroCharDistinto(raiz-
>palavra,palavra_nova);
                        strncpy(caracterString,maiorStringChar(raiz-
>palavra,palavra_nova), pos_caracter + 1);
                    }

                }
                PNO *raiz_antiga = raiz;
                int nova_pos_char = 0;

                if(raiz_pai != NULL){
                    nova_pos_char = primeiroCharDistinto(raiz_pai-
>palavra,palavra_nova);
                }
                // criando uma nova raiz com uma sub-string em comum
                PNO *raiz_nova = criarRaiz(nova_pos_char,caracterString,1
);

                // criando o novo nó/filho com a nova palavra que vai ser
 inserida
                PNO *novo_filho = criarRaiz(primeiroCharDistinto(raiz_nov
a->palavra,palavra_nova),palavra_nova,strlen(palavra_nova));
                TELE *carga_nova = criarElemento(linha);
                inserirLista(novo_filho->linhas,carga_nova);
                // atualizando pos_caracter da raiz antiga
                raiz_antiga-
>pos_caracter = primeiroCharDistinto(raiz_nova->palavra, raiz_antiga-
>palavra);
                // definindo os filhos
                raiz_nova-
>filho_esq = filhoMenor(raiz_antiga,novo_filho);
                raiz_nova-
>filho_dir = filhoMaior(novo_filho,raiz_antiga);

                return raiz_nova;
            }else{ // caso o nó/raiz não seja uma folha, será verificado
onde deve ocorrer a inserção
                    if(ladoInsercao(raiz,palavra_nova) == 1){
                    raiz->filho_esq = inserirArvore(raiz,raiz-
>filho_esq, palavra_nova);
                }else if(ladoInsercao(raiz, palavra_nova) == 2){
                    raiz->filho_dir = inserirArvore(raiz, raiz-
>filho_dir, palavra_nova);
```

```
                }

            }

        }

    }
    return raiz;
}
```

Primeira a função irá verificar se a nova palavra que deve ser inserida é igual a string *, caso ela seja igual, quer dizer que houve a quebra de linha, nessa situação será incrementado a variável global linha. Segundo caso irá verificar se a raiz passada na chamada da função é igual a NULL, se for igual, então estamos no caso inicial, e a palavra será inserida na raiz. Se não for nenhum desses dois casos, primeiramente será verificado se a palavra está na árvore, se estiver será chamada à função atualizaLinha, caso não esteja, será analisado se á raiz é folha, caso seja, é necessário que tenha a inserção ali, será verificados as condições e será inserido o elemento. Caso ela não seja uma folha, então irá percorrer pelos filhos, no caso será usado a função ladoInserção para saber em que filho deve ser feito a inserção.

FUNCIONAMENTO E TESTES

Na main() do código, serão inicializadas uma variável raiz que recebe NULL, uma string auxiliar chamada palavra, que irá receber as strings do arquivo, e também o arquivo. Será verificado se o ponteiro do arquivo é igual a NULL, caso seja, houve uma má referência do arquivo, caso contrário, a palavra irá receber as strings do arquivo, será tirada as pontuações e a string terá todos os seus caracteres em minúsculo. Depois dessas alterações, a palavra será inserida na árvore caso ela seja maior ou igual a 3 e se ela for a string *.

```
if(arquivo == NULL){
        printf("Arquivo nao encontrado\n");
    }else{
        while(!feof(arquivo)){

            fscanf(arquivo,"%s",palavra); // a string palavra recebendo a
 string do texto
            strlwr(palavra); // deixa em letra minúscula
            strcpy(palavra, tirarPontuacao(palavra)); // tirando as pontu
ações da palavra
            // fará a inserção caso a palavra seja maior ou igual a 3  ou
 ela seja apenas * simbolizando a leitura da outra linha
            if(strlen(palavra)>=3 || strcmp(palavra,"*") == 0){
                raiz=inserirArvore(NULL,raiz,palavra);

            }
        }
    }
```

O programa utiliza como entrada um arquivo de texto .txt, ao ser executado, ele irá diretamente imprimir as letras em ordem alfabética do texto, então caso seja necessário mudar o arquivo para efetuar outro teste, é necessário que altere a referência que está no fopen na última linha do código a seguir:

```
setlocale(LC_ALL, "portuguese_Brazil");

    FILE *arquivo;
    arquivo = fopen("alterar_aqui.txt","r");
```

Os testes foram feitos a partir dos documentos de textos disponibilizados no colabweb.

1º Teste:

brasil.txt

```
índio 38
antes 8 32
armaram 3 27
assim 20 44 57 65
brasil 1 17 21 41 45 54 58 62 66 71
carros 12
cazuza 1
cartão 15
chefe 14
cigarro 10
convencer 4 28
cara 18 42 55 63
confia 24 48 61 69 70
convidaram 1 25
cores 37
crédito 15
compositores: 1
desimportante 50
dizer 40
droga 6 30
elegeram 13
essa 6 30
estacionando 12
esta 2 26
fantástico 34
ficar 20 44 57 65
fim 36
fiquei 11
festa 2 26
garota 34
gente 20 44 57 65
grande 49
homens 3 27
instante 51
israel 1
meu 15 36
mim 24 48 61 69 70
mostra 18 42 55 63
malhada 7 31
nada 14
```

Coluna I – teste 1

```
nascer 8 32
navalha 16
negócio 22 46 59 67
nenhum 51
nem 10
nome 23 47 60 68
não 1 9 13 25 33 35 53 53
ofereceram 9
paga 19 43 56 64
pagar 5 29
pátria 49
pobre 2 26
porta 11
prá 40
programada 39
pra 2 4 20 26 28 44 57 65
que 3 7 27 31 36
quem 19 43 56 64
qual 22 46 59 67
quero 19 43 56 64
romero 1
sócio 23 47 60 68
será 36
sim 40 40
sortearam 33
subornaram 35
sem 5 29
taba 38
teu 22 23 46 47 59 60 67 68
trair 52 53
tua 18 42 55 63
toda 6 30
uma 16
vem 7 31
ver 5 19 29 37 43 56 64
vou 52 53
```

Coluna II – teste 1

Teste 2:

faroeste.txt

```
ódio 4 20 91 125
ônibus 34
ainda 6
altar 10
ali 12 144 146
aos 18 19
aprendeu 8
aquilo 12
acabou 60
aceitou 33
achar 23
ai! 58
ajudar 52 168
alimentar 50
amigos 61
ano 38
aprendiz 39
agora 69
alta 82 163
amar 79
aqui 32 148
arrependeu 74 159
asa 61
acabar 108
acontece 105
antes 91
apareceu 106
armar 102
ascendente 90
até 8 44 49 146
amanhã 127
acabo 130
amor 132
apertar 118
aquela 132
assistir 138
arma 111 152
armas 129
atirar 110
acertou 140
aplaudir 142
atirava 139
acreditou 164
atiro 154
```

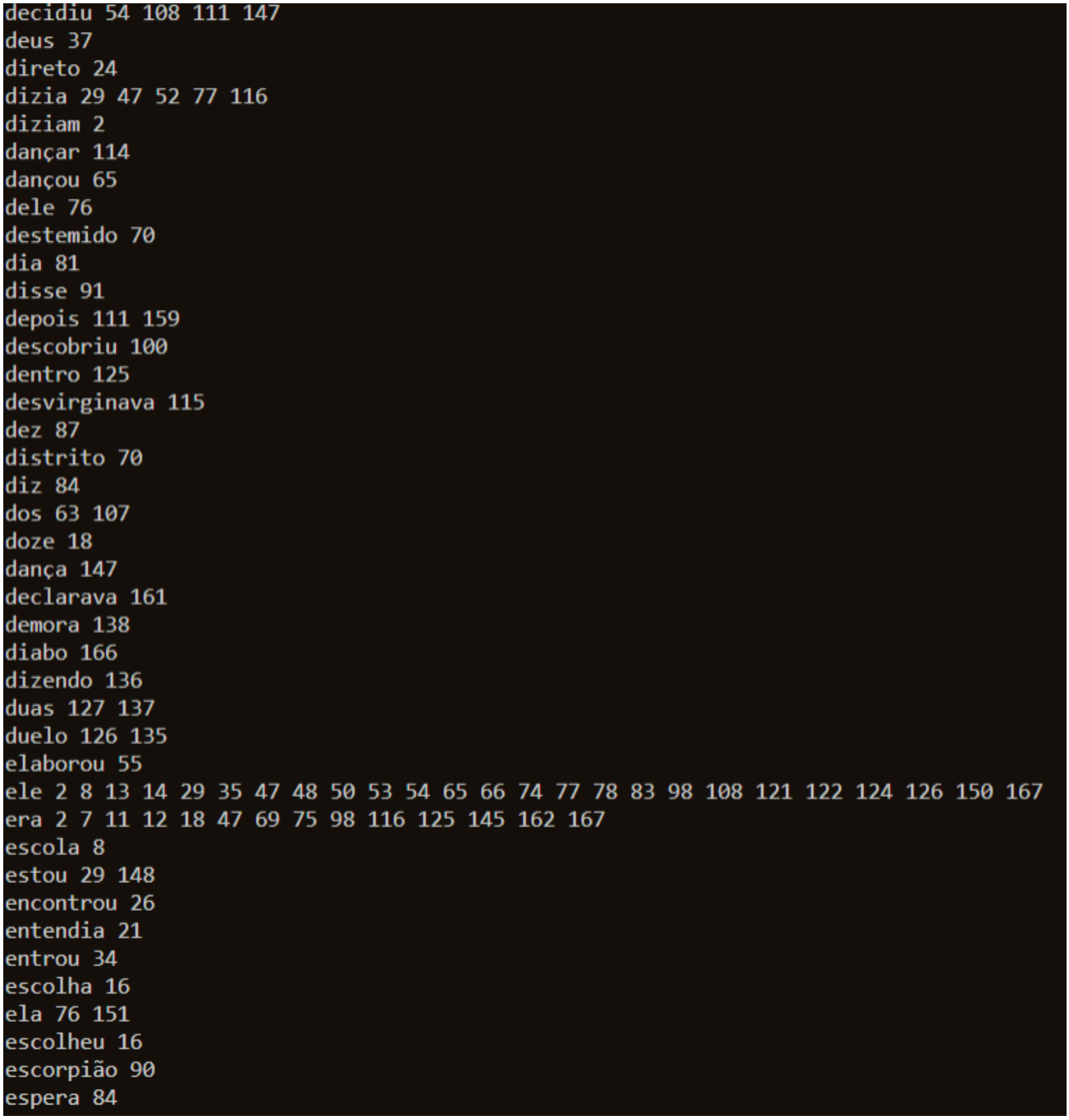
Coluna I – teste 2

```
atrás 88
aumentou 20
bandido 5 69 158
boiadeiro 26 27
bagulho 58
bastardo 44
bestificado 35
bisavô 44
bolívia 45 103
boto 85
bom 58
bomba 85
banca 85
boyzinho 63
brasília 29 166
bebedeira 99
brincar 18
bandeirinhas 142
bosal 132
brigar 112
brinque 90
burguesia 163
câmeras 143
cão 96
caboclo 1
com 6 8 26 35 60 80 82 88 90 91 101 108 123 130 157 160 166
compositores: 1
caixinha 10
causa 22
coisas 14 46
como 21 54 96
cidade 17 35 37 41 57 63 163
classe 22 82
colocavam 10
cansado 23
comia 17
cafezinho 25
central 34
chegando 25 121
comprou 24
carpinteiro 39 78
cem 40
```

Coluna II – teste 2

```
começar 48 56
começo 38
conhecia 43
conversa 53
cor 22
cortar 39
criança 5 86 145
capitão 72
começou 64 118 140
colégio 86
conheceu 73
casa 89 117 121
casar 77 120
coração 76 95
certo 97
consequências 96
começasse 112
contrabando 103
corpo 67
cristo 1 49 59 69 76 97 104 107 110 117 125 133 140 157 161
casou 123
ceilândia 127
chamou 126
cegou 149
chorou 121
circo 148
cinco 158
coisa 153
conseguiu 165
costas 139 154
crente 116
crucificado 56
deixou 3
deu 4 135 152 158
diferente 11
dinheiro 9 15 42 50 82 102
diante 20
discriminação 22
dali 60
dava 50
```

Coluna III – teste 2



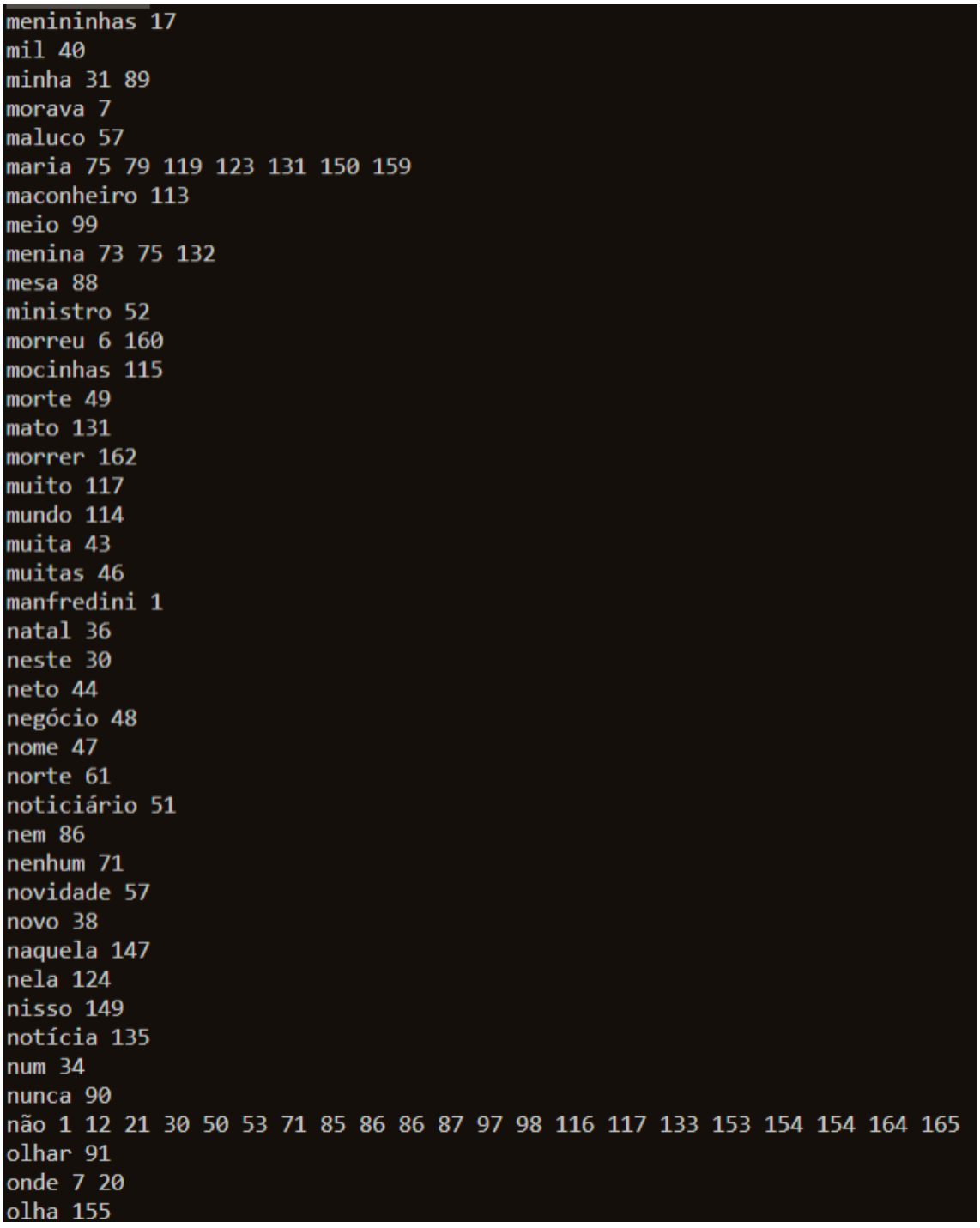
Coluna IV – teste 2

```
entrar 95 147
essas 95
estrelas 87
embebedou 99
embora 119
então 121 126 137 150
estava 97
eles 164
escolher 129
essa 168
estupro 67
falar 26 167
faroeste 1
fico 32
ficou 23 35 59 107
fez 61 114 124
filha 31 155
federal 70
faço 86
faz 83 168
festa 62
fica 88
filho 80 124
foi 19 24 25 26 28 56 66 73 98 122 138
frequentava 61
falou 101
fazer 133
filmava 144
frente 128
funcionava 21
futuro 98
fazenda 3
ganhava 40
garganta 141
gastar 42
general 72 87
gente 43 120 144 168
história 164
homem 139 153
hora 136
horas 51 127 137
igreja 9
```

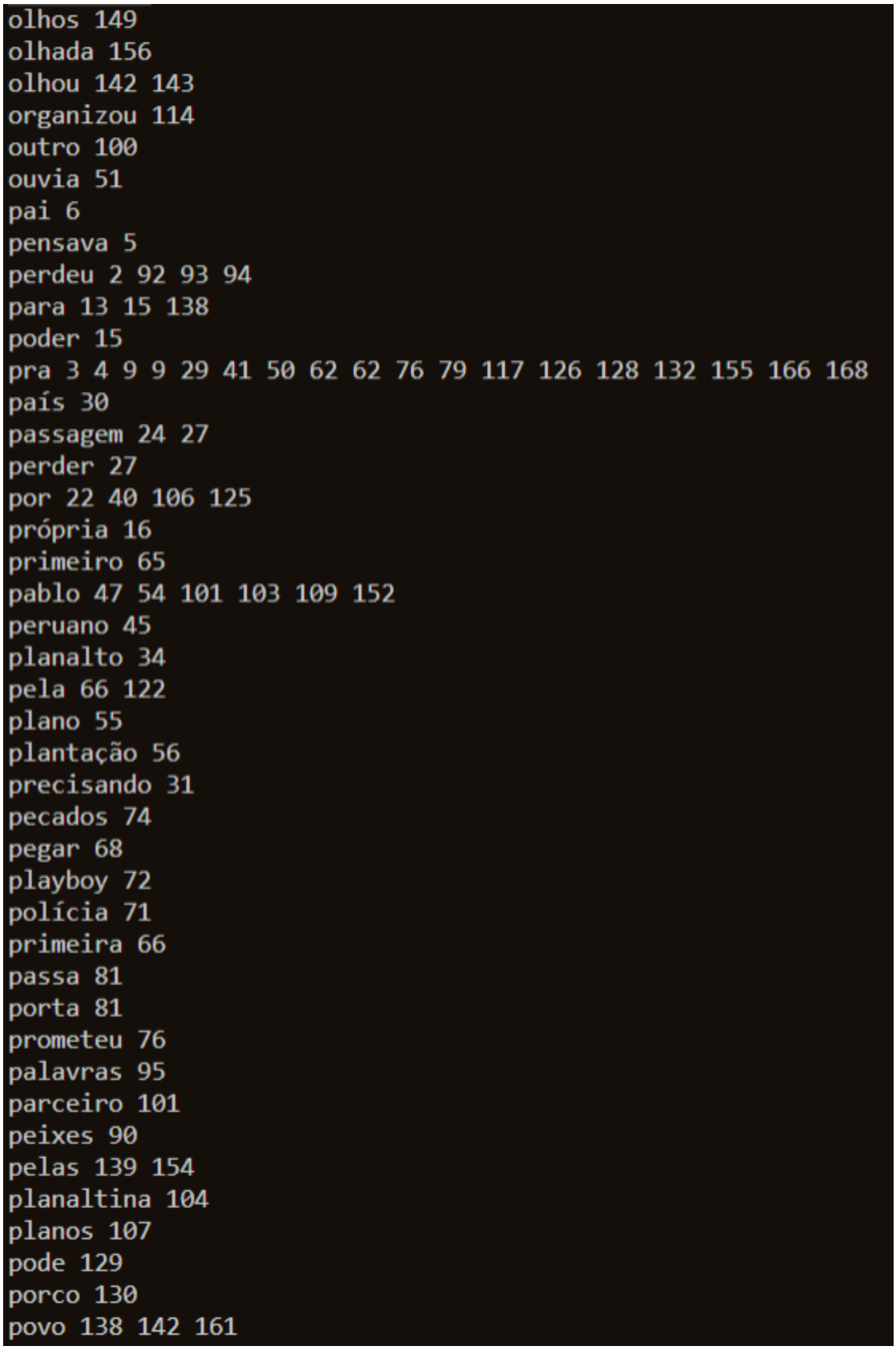
Coluna V – teste 2

```
indo 29
influência 63
indecorosa 83
inferno 66 122
incerto 98
inocentes 115
interessante 43
irmão 92 93 94
isso 86
jeremias 105 112 113 123 126 153
jesus 4
joão 1 28 33 59 84 108 142 160 161 165
jornal 85
junto 160
juntou 15
jurei 132
junior 1
lúcia 75 79 119 123 131 150 159
lhe 4 28 152
libertar 62
linda 37 75
lembrou 145
local 136
logo 57 57
lote 128
lugar 12 30 32 100
luzes 36
mão 82 88
médico 18
mês 40
mais 6 53 55
marasmo 3
medo 1 71
mesmo 11 11 130
mandado 19
mar 13
mas 28 37 50 53 63 91 105 109 116
melhor 30 89
meu 32 37 92 93 94 132 156
madeira 39
```

Coluna VI – teste 2



Coluna VII – teste 2



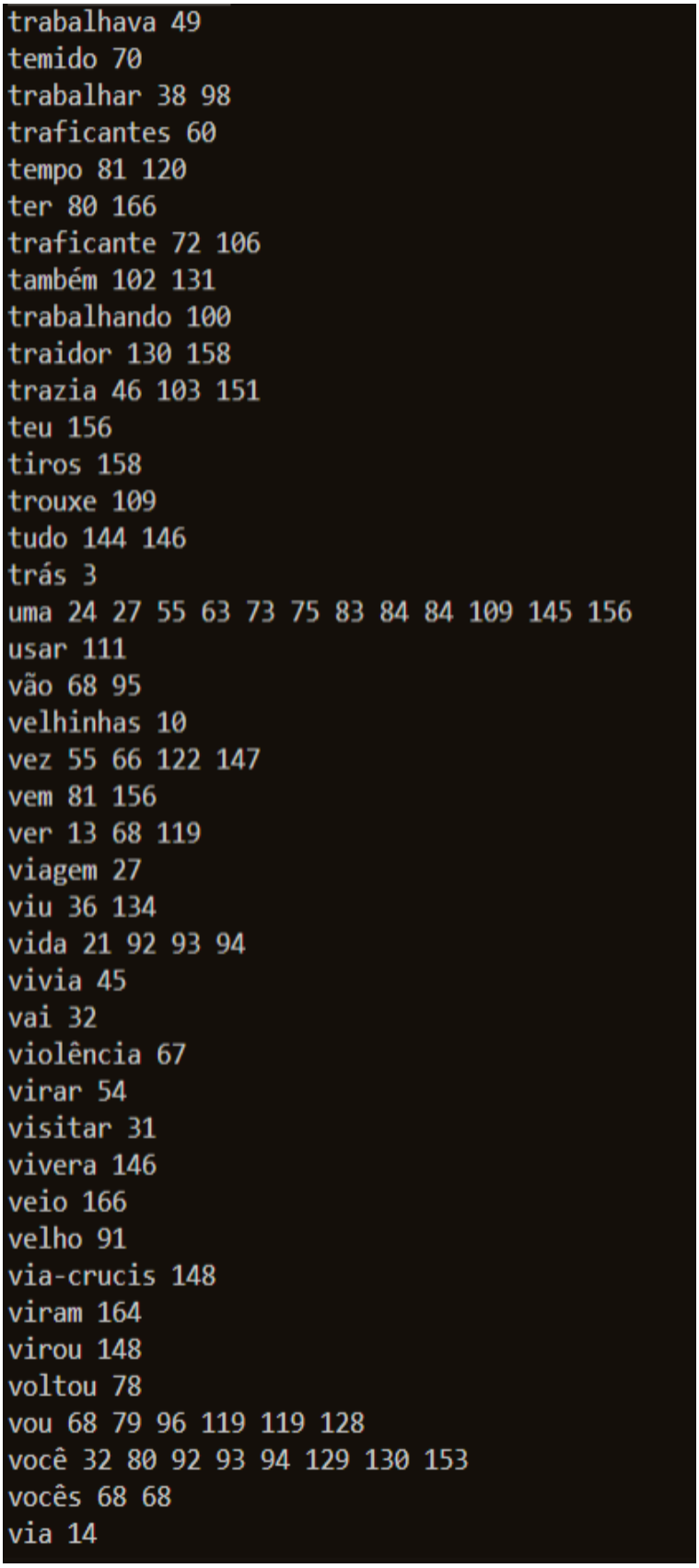
Coluna VIII – teste 2

```
proposta 33 83
pras 142 143
primo 152
perdão 156
porque 162
protejo 87
presidente 167
protetor 160
puta 155
pro 19 66 122 142 143 167
professor 8 18
quando 2 5 6 73 134 145 166
quem 26 132
queria 13 53 77 101 102 165 167
quero 80
quinze 19
que 2 4 10 11 12 14 37 45 48 52 52 54 77 84 88 97 100 101 105 108 112 116 128 130 133 135 139 144 146 152 153 161 164 165 168
reformatório 19
resposta 23 84 84
rapaz 42
rico 59
rodoviária 36
repente 63
rock 62
renome 106
revendia 104
roubar 9 64
razão 136
repórter 134
reconheceu 150
rezar 116
rockonha 114
roubo 65
renato 1
sábado 137
sangue 4 141 156
santo 1 49 55 59 69 76 97 104 107 110 117 125 133 140 157 161 162
sentir 4 156
ser 5 56 78
sair 13 89 91
sentia 11 12
sertania 7
```

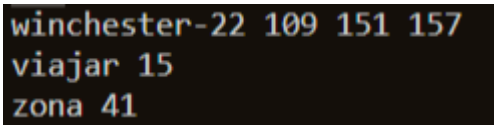
Coluna IX – teste 2

```
seu 4 12 20 42 44 47 52 55 67 98 100 130 152 160
soldado 6
souberam 57
saindo 36
salvador 24
salvar 28
sempre 52 79
sete 51
sem 56 138
sexta-feira 41
senhor 82 89
seus 74 149
sob 63
sabendo 107
saudade 118
sabia 110 116 133 162
segunda 122
sem-vergonha 113 155
sofrer 96 169
sentindo 141
solidão 16
sol 149
sorrir 140
sorveteiro 143
sou 153
sua 22 22 33 92 93 94
suas 129
tal 1 105
tanto 18 20
tinha 1 27 71 100 102
tiro 6
todo 3 42 114 138
todos 2 60 74
televisão 14 134
tentar 23
tem 58
terror 7 20
toda 168
todas 17
taguatinga 40
tomar 25
trabalhador 42
```

Coluna X – teste 2



Coluna XI – teste 2



Coluna XII – teste 2

3-CONCLUSÃO

O objetivo do trabalho foi concluído, foi impresso alfabeticamente as palavras junto com as linhas onde elas aparecem no texto, entretanto vimos que caso o texto tenha palavras com acentos gráficos, as palavras acentuadas são consideradas como menor alfabeticamente, fazendo com que eles sejam mostrados primeiramente.

Tratando-se da construção do código, uma das maiores dificuldades do trabalho foi, primeiramente, a implementação das árvores patrícias, existe muito pouco material disponibilizado na internet que nos dá informação clara de como acontece a inserção na árvore, ficando assim difícil de efetuar a implementação. A outra dificuldade, bem menor comparada com a citada anteriormente, foi como seria para pegar as strings do arquivo, e depois veio o problema das strings terem pontuações, tendo a necessidade de criar uma função que removesse as pontuações. Apesar desses problemas que surgiram durante a implementação do código, foram encontradas soluções para cada um, de forma que fosse possível realizar o trabalho.