Relatório: Trabalho Prático - Árvores Digitais

Instituto de Ciências Exatas e Tecnológicas,
Universidade Federal de Viçosa, Florestal, MG, Brasil

Algoritmos e Estruturas de Dados II

Caio Henrique Dias Rocha: 3892
Leonardo Augusto Alvarenga e Silva: 3895
Pedro Cardoso de Carvalho Mundim: 3877
William Lucas Araújo Santos: 3472

Florestal

2020

Resumo

Neste trabalho, buscamos a implementação e comparação de 2 (duas) estruturas de dados estudadas: (1) Árvore TRIE TST; e (2) Árvore PATRICIA. Ambas estruturas foram implementadas utilizando linguagem C, em que foi utilizado um texto como objeto de análise e as devidas comparações foram aplicadas em termos do custo computacional.

Palavras-chave: Árvores Digitais. TRIE. PATRICIA.

Introdução

O objetivo do trabalho é implementar os TADs Árvore TRIE TST e PATRICIA para armazenar palavras de um texto, para fins de comparação em termos do custo computacional das operações de inserção e pesquisa. Os TADs foram implementados em linguagem C. Para nos auxiliar, foram usados os slides das aulas ministradas pela professora Glaucia, além dos exemplos atribuídos no site da UFMG, disponibilizados pelo Professor Doutor Nívio Ziviani. É importante salientar também, o uso do repositório do aluno Gustavo Viegas como referência adicional.

Informações do Sistema

Informações do sistema

Data/hora atual:	segunda-feira, 26 de outubro de 2020, 19:19:51
Nome do computador:	ROCKFALLOUT
Sistema operacional:	Windows 10 Enterprise 64 bits (10.0, Compilação 18363)
Idioma:	português (Configuração regional: português)
Fabricante do sistema:	System manufacturer
Modelo do sistema:	System Product Name
BIOS:	BIOS Date: 08/10/12 18:34:52 Ver: 06.08
Processador:	Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz (8 CPUs)
Memória:	8192MB RAM
Arquivo de paginação:	11220MB usados, 2299MB disponíveis
Versão do DirectX:	DirectX 12

Desenvolvimento

Para execução e implementação dos algoritmos, foi utilizada como IDE o CodeBlocks, que dispõe do compilador GNU GCC. Para facilitar a organização e atualização do código necessário para o trabalho, utilizamos um repositório hospedado no GitHub.

Implementações:

A imagem abaixo expõe a implementação da estrutura “nó” da árvore PATRICIA, bem como outras definições necessárias para o funcionamento da árvore digital.

```
typedef enum {
    Internal,
    External
} PatriciaNodeType;
typedef char String[MAX_SIZE];
typedef short StringIndex;
typedef struct PatriciaNode *PointerPAT;
typedef struct PatriciaNode {
    union {
        struct {
            StringIndex index;
            char compare;
            PointerPAT left, right;
        } InternNode;

        String word;
    } Node;
    PatriciaNodeType type;
} PatriciaNode;
```

Já na figura a seguir, observamos as funções necessárias para o funcionamento da árvore PATRICIA. Maiores detalhes podem ser entendidos no próprio código através de comentários.

```c
// Testa se um nó é externo ou não
short isExternal(PointerPAT node);

// Cria um novo nó interno
PointerPAT CriarNoInterno(PointerPAT *left, PointerPAT *right, int index, char compare);

// Cria um novo nó externo
PointerPAT CriarNoExterno(String word, PointerPAT *p);

// Busca uma dada palavra
short PATRICIABuscarPalavra(String word, PointerPAT tree, int *compPesquisaPAT);

// Função interna de inserção, criando os nodes nescessários
PointerPAT InternaInserirNaPATRICIA(String word, PointerPAT *tree, int index, char differentChar);

// Insere uma palavra na PATRICIA
PointerPAT PATRICIAInserirPalavra(String word, PointerPAT *tree);

// Printa todas as palavras em ordem alfabetica
void PrintaAPatricia(PointerPAT tree);

// Imprime o numero de comparações de inserção
void NumeroDeComparacoesPAT(PointerPAT raiz);

// Insere as palavras do texto dado na PATRICIA
void PATRICIAInserirTexto (const char *texto, PointerPAT **raiz);
```

Ao executarmos o programa, veremos o menu principal contendo todas as opções de chamadas de funções para ambas as árvores.

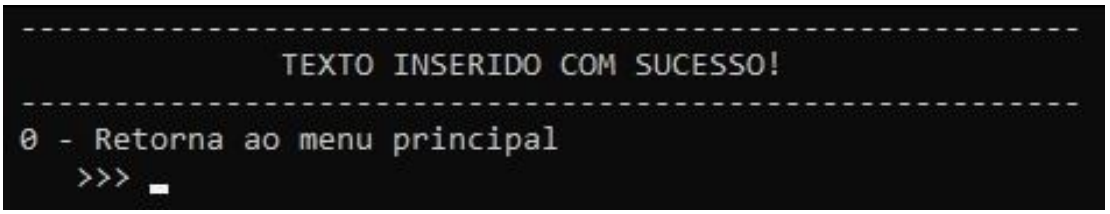
```
 ------------------------------------------------------------
|                        UFV - CAF                           |
|              Trabalho Pratico I  - AEDS II                 |
|____________________________________________________________|
|              Caio Rocha              -  3892               |
|              Leonardo Alvarenga      -  3895               |
|              Pedro Carvalho          -  3877               |
|              William Araujo          -  3472               |
 ------------------------------------------------------------

 ------------------------------------------------------------
|                      MENU PRINCIPAL                        |
|                                                            |
|   1 - Escolher arvore:                                     |
|   2 - Inserir palavra no texto:                            |
|   3 - Contar palavras:                                     |
|   4 - Exibir Texto:                                        |
|   5 - Inserir texto a partir de um arquivo:                |
|   0 - Sair                                                 |
 ------------------------------------------------------------
Entre com uma opcao: _
```

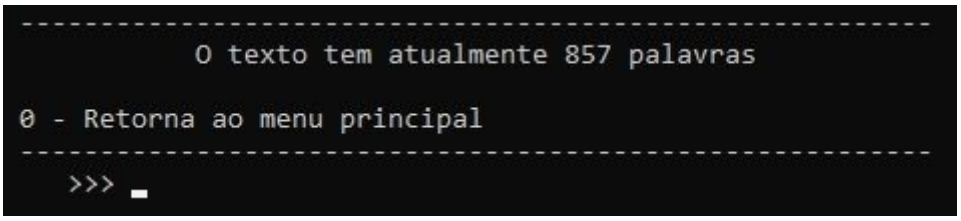
Após o início de sua execução, podemos inserir o nome do arquivo do qual desejamos copiar o texto a ser inserido nas árvores digitais. Ao selecionar a opção número cinco, uma mensagem pede pelo nome do arquivo procurado:

```
 --------------------------------
|    Digite o nome do arquivo    |
 --------------------------------
  >>> texto.txt_
```

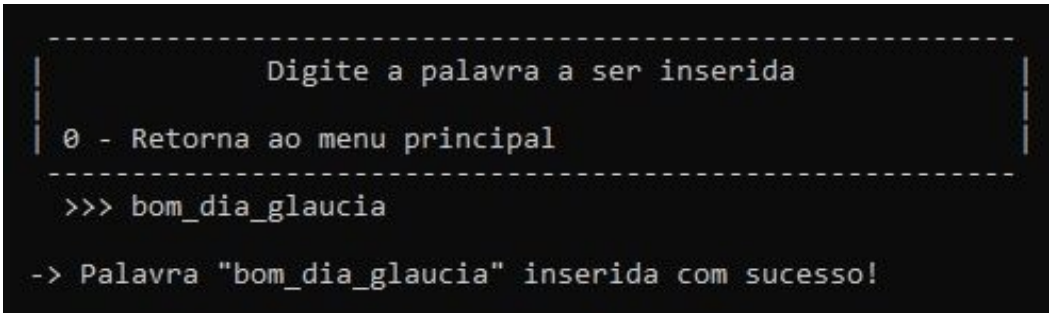
Tendo entrado com o nome de um arquivo válido, outra mensagem toma a tela, desta vez expondo o sucesso da inserção do texto, significando que agora podemos utilizá-lo nas árvores digitais.



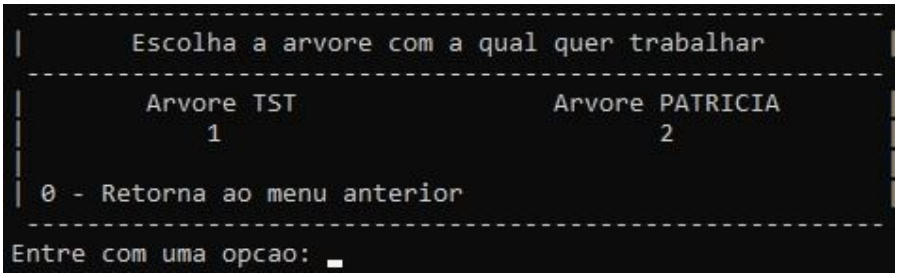
Através do menu, podemos acessar diversas funcionalidades como contar o número de palavras no texto selecionando a opção três:



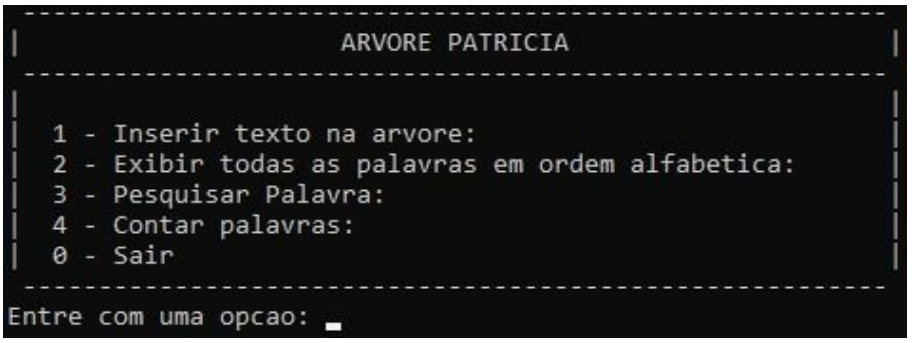
Ou inserir uma nova palavra no texto manualmente, escolhendo a opção dois, como visto na figura a seguir:



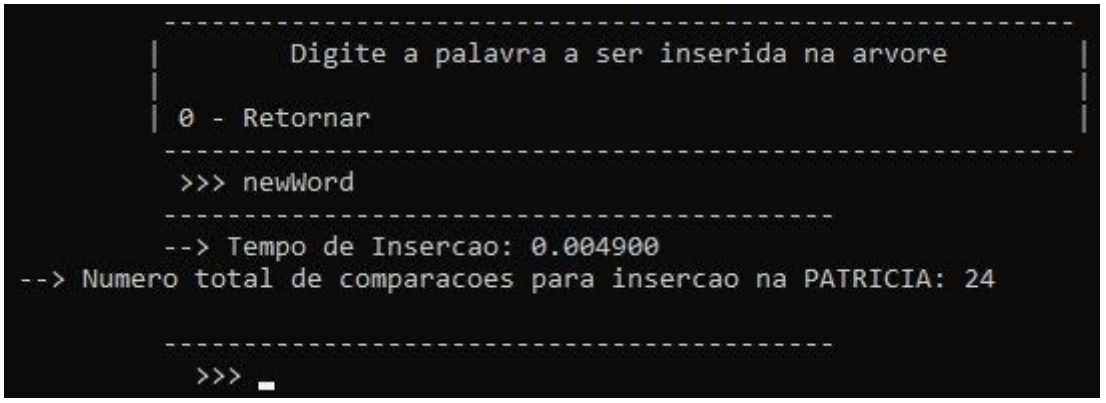
Uma vez que a inserção do arquivo no programa foi concluída, devemos escolher com qual árvore iremos trabalhar através da opção número um, como na figura abaixo:



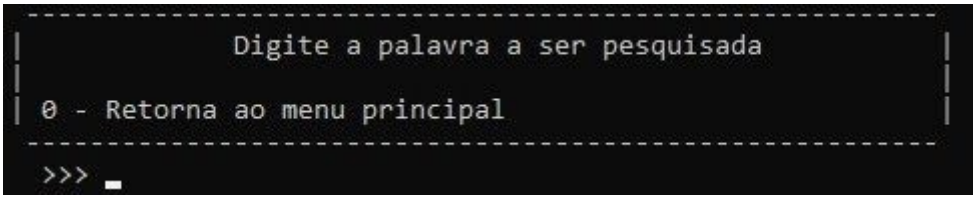
Ao escolhermos a Árvore PATRICIA, outro menu é mostrado:



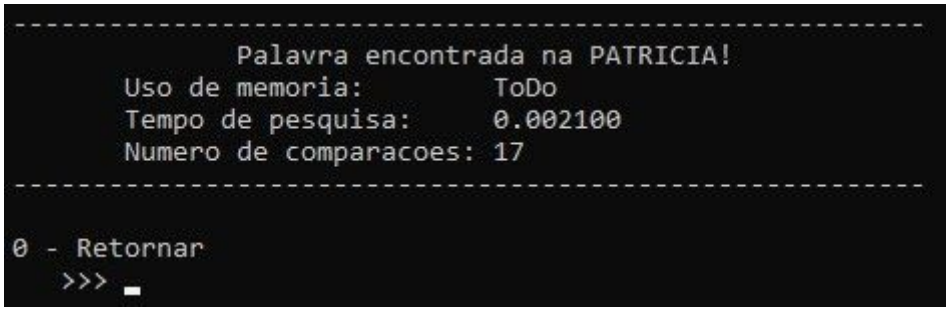
Devemos em seguida inserir a palavra na árvore, para obtermos melhores dados para comparação.



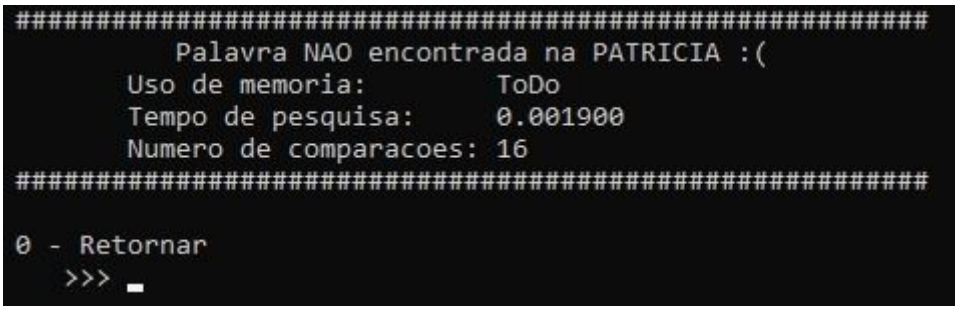
Podemos também buscar por uma palavra na estrutura, por exemplo, através da opção três.



Não importando se a palavra existe dentro da árvore:



Ou não:



A imagem abaixo expõe a implementação da estrutura “nó” da árvore TRIE TST, bem como outras definições necessárias para o funcionamento da árvore digital.

```
typedef struct NO* PointerTST;

typedef struct NO{
    char letter;
    bool endOfString: true;
    PointerTST left, eq, right;
}No;
```

Já na figura a seguir, observamos as funções necessárias para o funcionamento da árvore TRIE TST. Maiores detalhes podem ser entendidos no próprio código através de comentários.

```c
// Funcao para inicializacao da TST
PointerTST* initializeTST(){

// Funcao para alocar e retornar um novo noh dinamicamente
PointerTST newNode(char info){

// Funcao recursiva para insercao de uma nova palavra na arvore trie TST
void insert(PointerTST *root, char *word){

// Funcao secundaria, executa o percurso transversal na arvore
void traversetst(PointerTST root, char *buffer, int depth){

// Funcao primaria, executa o percurso transversal na arvore chamando a funcao anterior
void TraverseTST(PointerTST root){

// Funcao de pesquisa
int searchTST(PointerTST root, char *word, int *comparacoes)

// Mostrar o contador e zera-o em seguida
void NumeroDeComparacoesTST(PointerTST raiz);

// Insere o texto na TST
void TSTInserirTexto(const char *texto, PointerTST **raiz);
```

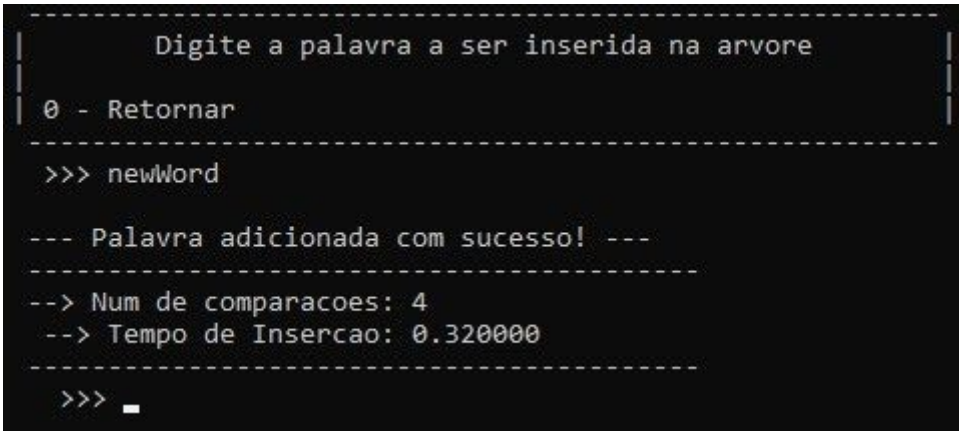
Voltemos para o menu do programa, desta vez iremos para as operações na TRIE TST. Para isso selecionaremos a opção um para acessarmos a árvore desejada, como na figura abaixo.

```
 ------------------------------------------------------------
|       Escolha a arvore com a qual quer trabalhar           |
 ------------------------------------------------------------
|        Arvore TST                Arvore PATRICIA           |
|            1                            2                  |
|                                                            |
| 0 - Retorna ao menu anterior                               |
 ------------------------------------------------------------
Entre com uma opcao: _
```

Selecionamos então a opção um, que nos leva a um menu secundário, dedicado à árvore TRIE.

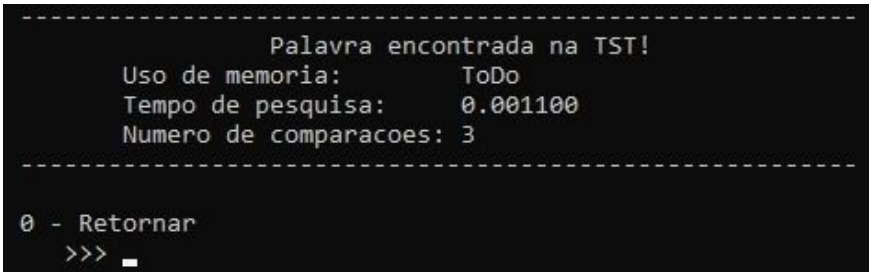
```
 ------------------------------------------------------------
|                       ARVORE TST                           |
 ------------------------------------------------------------
|                                                            |
|  1 - Inserir texto na arvore:                              |
|  2 - Exibir todas as palavras em ordem alfabetica:         |
|  3 - Pesquisar Palavra:                                    |
|  4 - Contar palavras:                                      |
|  0 - Sair                                                  |
 ------------------------------------------------------------
Entre com uma opcao: _
```

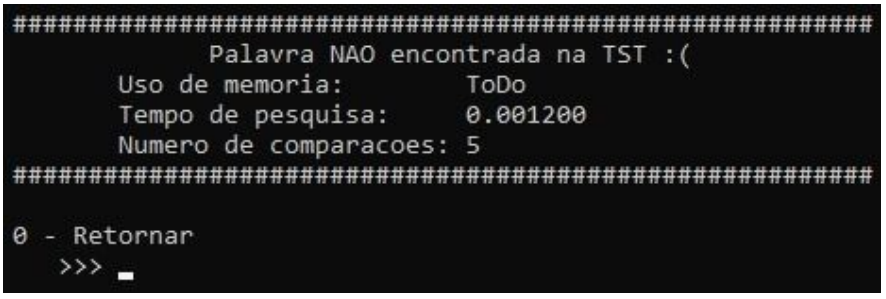
Assim como na árvore anterior, podemos inserir uma palavra nova na árvore para obtermos melhores dados para comparação.



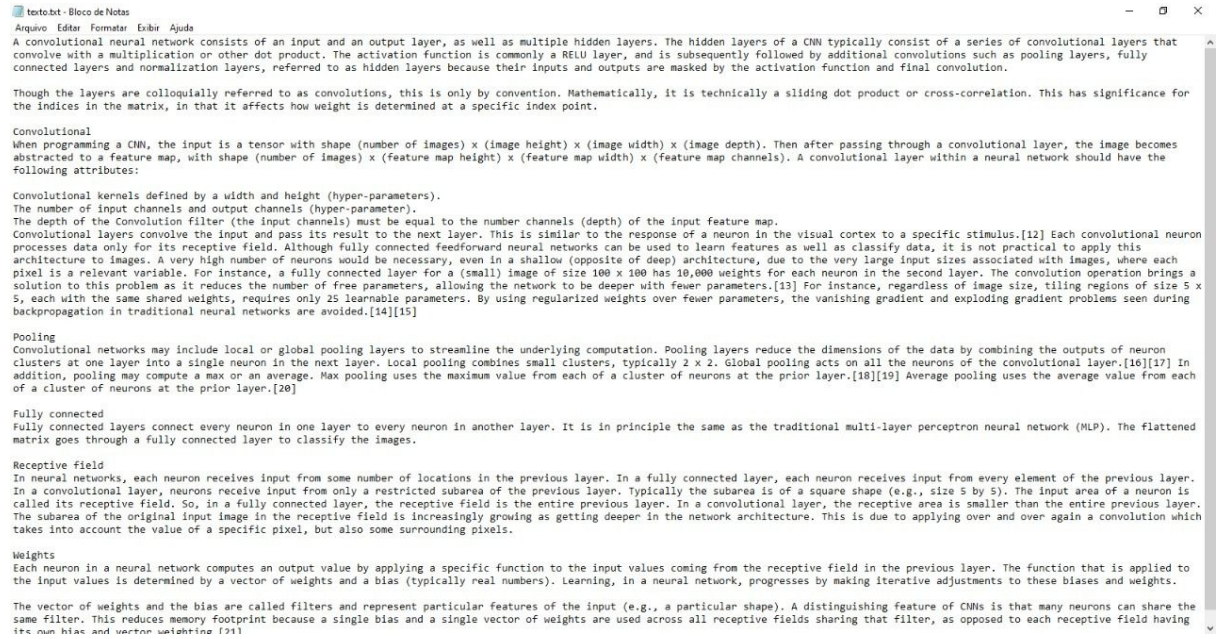
Aqui, todas as operações que vimos anteriormente sendo realizadas pela árvore PATRICIA podem ser executadas também pela TRIE TST.

Como, por exemplo, a pesquisa, que independente da existência da palavra, nos mostrará algum resultado:





Adiante, temos o texto que foi utilizado nas operações apresentadas. É importante salientar que, devido ao fato de não usar acentuação, o texto dado está em inglês.


texto.txt - Bloco de Notas

Arquivo Editar Formatar Exibir Ajuda

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or other dot product. The activation function is commonly a RELU layer, and is subsequently followed by additional convolutions such as pooling layers, fully connected layers and normalization layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution.

Though the layers are colloquially referred to as convolutions, this is only by convention. Mathematically, it is technically a sliding dot product or cross-correlation. This has significance for the indices in the matrix, in that it affects how weight is determined at a specific index point.

Convolutional
When programming a CNN, the input is a tensor with shape (number of images) x (image height) x (image width) x (image depth). Then after passing through a convolutional layer, the image becomes abstracted to a feature map, with shape (number of images) x (feature map height) x (feature map width) x (feature map channels). A convolutional layer within a neural network should have the following attributes:

Convolutional kernels defined by a width and height (hyper-parameters).
The number of input channels and output channels (hyper-parameter).
The depth of the Convolution filter (the input channels) must be equal to the number channels (depth) of the input feature map.
Convolutional layers convolve the input and pass its result to the next layer. This is similar to the response of a neuron in the visual cortex to a specific stimulus.[12] Each convolutional neuron processes data only for its receptive field. Although fully connected feedforward neural networks can be used to learn features as well as classify data, it is not practical to apply this architecture to images. A very high number of neurons would be necessary, even in a shallow (opposite of deep) architecture, due to the very large input sizes associated with images, where each pixel is a relevant variable. For instance, a fully connected layer for a (small) image of size 100 x 100 has 10,000 weights for each neuron in the second layer. The convolution operation brings a solution to this problem as it reduces the number of free parameters, allowing the network to be deeper with fewer parameters.[13] For instance, regardless of image size, tiling regions of size 5 x 5, each with the same shared weights, requires only 25 learnable parameters. By using regularized weights over fewer parameters, the vanishing gradient and exploding gradient problems seen during backpropagation in traditional neural networks are avoided.[14][15]

Pooling
Convolutional networks may include local or global pooling layers to streamline the underlying computation. Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, typically 2 x 2. Global pooling acts on all the neurons of the convolutional layer.[16][17] In addition, pooling may compute a max or an average. Max pooling uses the maximum value from each of a cluster of neurons at the prior layer.[18][19] Average pooling uses the average value from each of a cluster of neurons at the prior layer.[20]

Fully connected
Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network (MLP). The flattened matrix goes through a fully connected layer to classify the images.

Receptive field
In neural networks, each neuron receives input from some number of locations in the previous layer. In a fully connected layer, each neuron receives input from every element of the previous layer. In a convolutional layer, neurons receive input from only a restricted subarea of the previous layer. Typically the subarea is of a square shape (e.g., size 5 by 5). The input area of a neuron is called its receptive field. So, in a fully connected layer, the receptive field is the entire previous layer. In a convolutional layer, the receptive area is smaller than the entire previous layer. The subarea of the original input image in the receptive field is increasingly growing as getting deeper in the network architecture. This is due to applying over and over again a convolution which takes into account the value of a specific pixel, but also some surrounding pixels.

Weights
Each neuron in a neural network computes an output value by applying a specific function to the input values coming from the receptive field in the previous layer. The function that is applied to the input values is determined by a vector of weights and a bias (typically real numbers). Learning, in a neural network, progresses by making iterative adjustments to these biases and weights.

The vector of weights and the bias are called filters and represent particular features of the input (e.g., a particular shape). A distinguishing feature of CNNs is that many neurons can share the same filter. This reduces memory footprint because a single bias and a single vector of weights are used across all receptive fields sharing that filter, as opposed to each receptive field having its own bias and vector weighting.[21]

Considerações Finais

Após implementado o projeto, nos surpreendemos com a diferença visível no número de comparações necessárias para execução de certas operações em cada uma das árvores digitais. Dentre elas, a pesquisa chamou a maior atenção, dado que obtemos uma diferença de mais 14 comparações em uma pesquisa bem sucedida entre as duas árvores (3 para TRIE TST e 17 para a PATRICIA). Seguem abaixo os gráficos mostrando os respectivos tempos e número de comparações gastos para pesquisa e inserção em cada uma das árvores, o que nos deu uma melhor compreensão acerca das diferenças entre as árvores TRIE TST e PATRICIA:

No gráfico, podemos observar que o tempo para busca na árvore PATRICIA foi maior que para a árvore TRIE TST.

Como já mencionado, o número de comparações da árvore PATRICIA também foi maior que para a árvore TRIE TST.

Em relação ao tempo de inserção, podemos concluir que para a PATRICIA o tempo foi menor.

Já em relação às comparações é visível que para a TRIE TST o número foi menor. Essas conclusões também podem ser pensadas com relação ao conteúdo dado em sala de aula.

Por fim, concluímos com o trabalho que, as árvores digitais TRIE e PATRICIA, podem ser usadas para várias implementações que necessitam de um bom desempenho em pesquisa e uso de memória, pois, se comparadas com outros tipos abstratos de dados vistos anteriormente, possuem desempenho superior na maioria dos casos quando equiparadas a listas ou vetores, por exemplo.