Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
Instituto Federal Catarinense
Câmpus Videira

ARTHUR MARCOS SCHNEIDER DE OLIVEIRA

RELATÓRIO ALGORITMOS

Videira (SC)
2020

ARTHUR MARCOS SCHNEIDER DE OLIVEIRA

RELATÓRIO ALGORITMOS

Relatório de Conclusão da Disciplina de Algoritmos apresentado ao Curso de Ciência da Computação do Instituto Federal de Educação Ciência e Tecnologia Catarinense – Câmpus Videira para obtenção da média final na referida disciplina.
Orientador(a): Manassés Ribeiro.

Videira (SC)

2020

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Este relatório possui o foco nos conhecimentos adquiridos e utilizados durante o período de ensino na disciplina de Algoritmos do Instituto Federal Catarinense. Com o objetivo de demonstrar como funcionam os procedimentos obrigatórios, métodos e organização utilizados para a resolução de um problema em específico na programação em linguagem C. Além de valorizar e expressar, as experiências ganhas e o aprimoramento de diferentes conhecimentos e desempenho na área, através de pesquisas.

O projeto final desse semestre foi iniciado no dia 18 de julho, sendo finalizado no dia 26 de julho, estando focado na resolução de um dos problemas que eram possíveis escolher, sendo especificamente o: “Problema 1: Processamento de array (matriz)”.

1.1 METODOLOGIA

Este relatório estará apresentando diferentes conceitos atrelados a linguagem de programação em C utilizando o ambiente de desenvolvimento integrado (IDE): Visual Studio Code, para toda a programação. Será analisado os motivos e resultados da conclusão que foi determinada a partir da programação feita para o problema. Explicando a lógica e ferramentas para a construção do algoritmo.

2 TAREFAS REALIZADAS

Assim como o problema já demonstra pelas imagens e textos disponibilizados, foi decidido começar por uma listagem de aspectos obrigatórios para o algoritmo: Não houve uso de variável global; estruturas de repetição; modularização (mín.: 1 módulo de procedimento e 2 de ação); passagem de parâmetro por valor e referência; registros; alocação dinâmica de memória e matrizes dinâmicas (ponteiros).

Antes de compreender o código em si é preciso ter a concepção de dois termos para a resolução do problema, que é convolução e kernel. Respectivamente o processo de convolução discreta, como é conhecido, envolve o processamento de imagens. A imagem original será interpretada pelo computador através de uma matriz com diferentes números, e essa matriz entrará em contato com outra matriz com valores já determinados chamada de “núcleo” ou “kernel”. Esse contato entre elas é representado através de uma função (que será abordada posteriormente), gerando então uma terceira matriz feita pelo contato entre a matriz original e a de kernel.

Figura 1 – Exemplo atrelado a imagem original e como o computador a interpreta.

Fonte: ELO7 (2020).

3 CONSTRUÇÃO DO ALGORITMO E RESULTADOS

No início do código, contém as bibliotecas usadas para o algoritmo: “<stdio.h>” e “<stdlib.h>”. Logo em seguida é feita a declaração do registro, que conterá as informações das dimensões (linha e coluna) da matriz que serão fornecidas pelo usuário. Posteriormente é feita a prototipação, sendo toda a modularização feita no algoritmo e nesses módulos há a passagem de parâmetro por valor e referência.

Dentro da função “int main”, teremos a variável “matriz1” para o registro “sDimensaoMatriz”, tendo assim as variáveis “matriz1.linha” e “matriz1.coluna” dentro de suas funções: “int EscolhaLinhaMax” e “int EscolhaColunaMax”, retornando então seus valores inteiros que foram inseridos pelo usuário. Vale ressaltar que foram criadas as variáveis “linhamax” e “colunamax” dentro da função “int main”, sendo essas o valor retornado das funções citadas anteriormente.

Em seguida é criada duas variáveis (“matriz” e “matrizaux”), com ambas contendo um ponteiro (linha) dentro de outro ponteiro (coluna) e sendo atreladas a seus respectivos módulos: “int** CriarMatriz” e “int** CriarMatrizAuxiliar”, dentro desses módulos é feita a alocação dinâmica de memória baseada nas informações coletadas do usuário, formando assim um registro e alocação para a linha e outro para a coluna, em ambas as matrizes. Para alocar os dados das matrizes com valores fornecidos pelo usuário foi utilizada a estrutura de repetição “for”.

Posteriormente, encontra-se os módulos “void EscolhaValoresDentroMatriz” e “void EscreverValoresDentroMatriz” (com ambos utilizando a estrutura de repetição “for”, para as linhas e colunas), onde o primeiro indicará os valores inteiros que o usuário colocará dentro da matriz, baseada também nas dimensões que ele havia escolhido anteriormente, já o outro módulo mostrará a matriz completa seguindo a ordem dos valores e dimensões que o usuário colocou anteriormente.

O próximo módulo na função “int main” é “void ProcessoConvolucaoEscolhaKernel”. Dentro dele será encontrado outras funções, como no caso a: “int EscolhaKernel”, onde dependendo do número que for escolhido levará para um filtro diferente (feito através de um “switch-case”), modificando a construção da matriz baseado no número escolhido, variando entre 7 filtros

(“KernelCaso1”, “KernelCaso2” e outros), além de posteriormente com a alteração dos valores, ainda será escrito a nova matriz formada através da original e do filtro escolhido. Por último há uma outra função “void FinalizarPrograma” (sendo nesse caso o dígito “0”). Por conta de cada um dos casos apresentarem uma diferente matriz determinada, a função de como esses valores são calculados se encontram justamente em cada um dos casos (variando os números).

Tudo o que foi citado anteriormente explica o motivo da existência de duas matrizes (matriz e matriz auxiliar). Isso se deve pelo fato de que a função que gerará a nova matriz, alteraria os valores de dentro da matriz original, sendo assim a matriz original passaria pelo processo de convolução considerando valores que já estariam alterados dentro da mesma, ou seja, não se baseando nos valores originais dados pelo usuário, a mesma mostraria valores incorretos, sendo assim, os resultados são passados para a segunda matriz, não alterando os valores na matriz principal, trazendo resultados corretos.

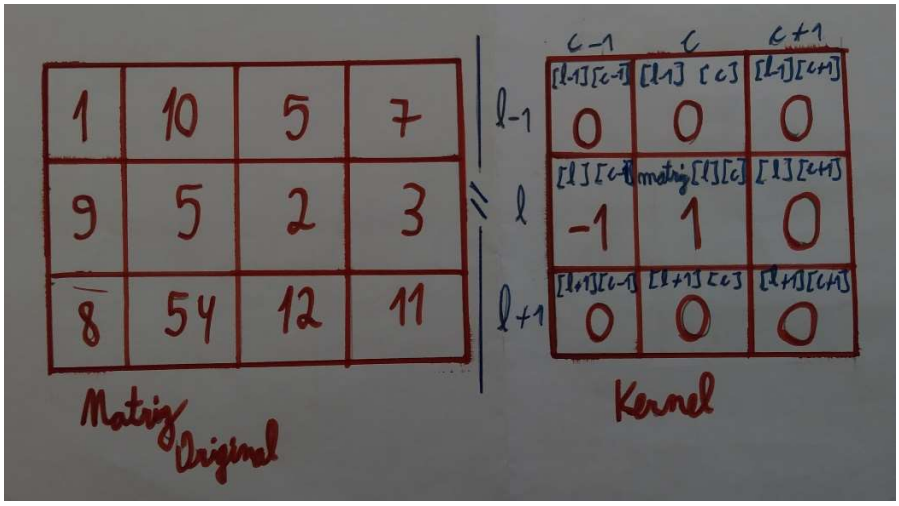
Figura 2 – Parte do código exemplificando os dois parágrafos anteriores.

```
void KernelCaso1(int linhamax, int colunamax, int **matriz, int
**matrizaux){
    for(int l=0;l<linhamax;l++){
        for(int c=0;c<colunamax;c++){
            if(l==0 || c==0 || l==linhamax-1 || c==colunamax-1){
                printf("\t");
            }
            else{
                matrizaux[l][c]=((matriz[l-1][c-1]*0)+(matriz[l-1][c]*0)
                +(matriz[l-1][c+1]*0)+(matriz[l][c-1]*-1)+(matriz[l][c]
                *1)+(matriz[l][c+1]*0)+(matriz[l+1][c-1]*0)+(matriz[l+1]
                [c]*0)+(matriz[l+1][c+1]*0));
                printf("%i\t", matrizaux[l][c]);
            }
        }
        printf("\n");
    }
}
```

Fonte: O Autor (2020).

A função do caso 1 se encontra dentro do “else” e essa seria sua forma através do código, adaptando isso para um desenho da matriz ficaria dessa forma:

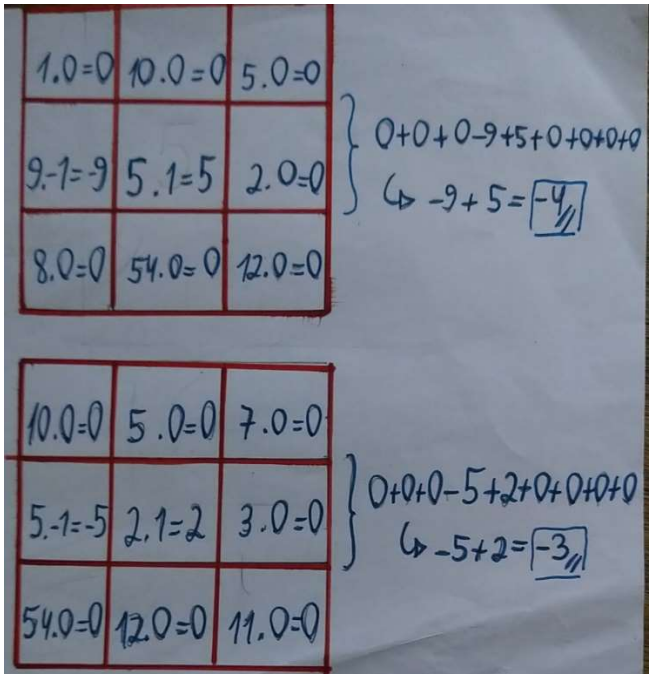
Figura 3 – Exemplo da função mostrada na figura anterior.



Fonte: O Autor (2020).

Considerando os dados da figura acima e colocando-os na função mostrada na figura 2 seria resolvido dessa forma:

Figura 4 – Desenvolvimento da função.



Fonte: O Autor (2020).

Vale ressaltar que caso não houvesse a “matrizaux”, o resultado para o número original “2” não seria igual, pois sofreria a influência do outro número original “5” que agora nessa etapa já teria se transformado em “-4”, o daria o resultado “-4+2=2” sendo então, incorreto.

Seguindo com o algoritmo, o resultado no final é visto da seguinte forma:

Figura 5 – Resultado da função.

Fonte: O Autor (2020).

No resultado, todos os valores na borda da matriz são ignorados e são apenas mostrados os que não se encontram consequentemente na primeira ou última linha e na primeira e última coluna.

Para finalizar então é retornado à função “int main” onde será realizada a desalocação da memória com a função “void DesalocaMatrizeMatrizaux”, desalocando então as informações que haviam sido armazenadas em “int** matriz” e “int **matrizaux”.

REFERÊNCIAS

CASTRO, Wellington. **DIP03 – Matriz de convolução e detecção de bordas.** Disponível em: <https://capivararex.wordpress.com/2016/04/25/dip03-matriz-de-convolucao-e-deteccao-de-bordas/>. Acesso em: 18 jul. 2020.

DOCS.GIMP. **Matriz de convolução.** Disponível em: <https://docs.gimp.org/2.8/pt_BR/plug-in-convmatrix.html>. Acesso em: 18 jul. 2020.

ELO7. **Princípios de Processamento de Imagens: Uma introdução à Convolução.** Disponível em: <https://elo7.dev/convolucao/>. Acesso em: 18 jul. 2020.

UFPR. **Exame Final de Algoritmos e Estruturas de Dados I.** Disponível em: <http://www.inf.ufpr.br/cursos/ci055/Provas_antigas/final-20181.pdf>. Acesso em: 18 jul. 2020.