Documentação de Crescimento de Regiões

Aluno: Tainan Henrique de Albuquerque

1. Introdução

A manipulação de pilhas, listas e filas dinâmicas possui aplicações diversas quando falamos de programação. O uso de TAD (Tipo Abstrato de Dados) é relevante para encapsular detalhes da implementação das funções e structs.
O escopo desse trabalho é elaborar um programa que faça leitura de uma imagem PGM e a converta em uma imagem PPM. As funções que realizaram essa mudança devem estar contidas no TAD que será criado. Espera-se praticar com esse trabalho a construção e manipulação de listas encadeadas, além dos conceitos práticos de vetores, ponteiros, typedef, structs, entre outras.

2. Implementação

Estrutura de dados

Para implementação do trabalho foi criada uma matriz dinâmica de struct utilizando alocação de memória. Segue um exemplo para ilustrar a forma de cada struct.

cor Final
int pixel
int linha
int coluna
int percorrido
struct matriz *prox

Descrição dos termos da struct matriz:

- **cor Final** é uma struct que contêm o valor do RGB (Red, Green, Blue) da imagem de saída que será uma PPM
- **pixel** terá o valor que a matriz fornecida terá naquele local, esse valor se refere ao tom de cinza da imagem PGM
- **linha** e **coluna** serão os indicadores da posição dessa "célula" na matriz
- **Percorrido** terá a utilidade de dizer se essa posição foi ou não visitada
- ***prox** é um ponteiro de matriz (podemos fazer analogia ao ponteiro de célula apresentado em sala) ele consegue apontar em outra posição da matriz.

int R
int G
int B

Descrição dos termos da struct cor:

- **R** guarda o valor na escala de vermelho
- **G** guarda o valor na escala de verde
- B guarda o valor na escala de azul

matriz *primeiro
matriz *ultimo

Descrição dos termos da struct lista:

- ***primeiro** e ***ultimo** são ponteiros usados na lista para indicar qual é a primeira e última posição dentro da lista

int x
int y
cor RGB

Descrição dos termos da struct semente:

- **x** e **y** vão guarda a posição da semente dentro da matriz
- **cor RGB** é uma struct que guarda os valores dos níveis de vermelho, verde e azul da semente

Funções e Procedimentos

No TAD existem as seguintes funções:
void criaLista(Lista *lista): recebe uma copia do endereço da lista para criar ela naquele lugar. Essa função nada mais faz, do que criar uma cabeça para lista e apontar os ponteiros primeiro e ultimo para a cabeça, além de apontar o ponteiro prox da cabeça para NULL (nada).

void insere (Lista *lista, matriz *resolucao): recebe uma copia do endereço da lista e uma posição da matriz, ele coloca essa posição da matriz no final da lista e aponta o ponteiro prox da posição para NULL. Nessa função ela avalia se lista está vazia ou não, quando a lista esta vazia ela adiciona esse elemento fazendo com que o ponteiro prox da cabeça aponte para ele e que o ultimo também aponte, porém quando já tem elementos, a função apenas adiciona essa posição passada no final e faz com que o ponteiro de prox onde o ponteiro ultimo da lista está apontando passar a apontar nesse novo elemento e aponta o ponteiro ultimo para essa nova posição, também aponta o ponteiro prox do elemento para NULL.

void retira (Lista *lista): recebe uma copia do endereço da lista e retira o primeiro elemento da lista após a cabeça. Para realizar essa retirada ele aponta o ponteiro do prox da cabeça para o segundo da lista e aponta o ponteiro prox da posição retirada da lista para NULL. Nessa função também compara se existe apenas um elemento na lista que será passada, isso ocorre porque se a lista possuir apenas um elemento além da cabeça, quando se retira ele a lista fica vazia, então se tal comparação for verdadeira ele retira esse elemento e faz ponteiro ultimo apontar para a cabeça.

void compara (Lista *lista, matriz **resolucao, local semente, int T, int l, int c, int coluna, int linha): recebe uma copia do endereço da lista, uma copia do endereço de toda a matriz, uma semente, o valor do limiar T, a posição da semente em x, a posição da semente em y, a quantidade de linhas e colunas que a matriz possui. Nessa função será realizada a comparação entre os píxeis da semente e da sua vizinhança, quando a diferença em módulo for menor ou igual o limiar T, essa posição é inserida no final da lista de comparação pela função insere, além de colorir as posições em que a diferença seja menor ou igual ao limiar T e mudar o valor do pixel para o valor do pixel da semente. Nessa função existem algumas regras para a comparação, como não comparar com usa semente que já tenha sido colorido, não comprar a semente com uma posição inexistente na matriz.

Programa Principal

O programa principal primeiramente testa se foi passada uma imagem para depois criar as variáveis necessárias para execução e as outras, como a matriz e a semente que estão no TAD. Como o arquivo da imagem e o auxiliar recebem o mesmo nome é necessário apenas copiar o nome para vetores de caracteres auxiliares e concatenar a as extensões de cada um para abri-los, tal ação é executada pelo comando strcpy, strcat e fopen. Depois de abrir os arquivos, o primeiro passo há ser executado é ler o arquivo auxiliar que contêm as o número de sementes, o limiar, a posição de cada semente e as cores em RGB delas. Após a leitura do auxiliar é feita leitura do arquivo que contêm a imagem, nesse momento além de ler o valor do pixel é também inicializado o RGB de cada posição com o nível de cinza que esta na imagem PGM, marcado a posição de cada pixel da imagem e inicializado o valor de percorrido com zero, que significa que aquela posição não foi colorida. Em seguida é chamada a função para comparar a sementes com as posições vizinhas dela (cima, direita, embaixo e esquerda), no fim da comparação o programa principal chama a função retira para retirar a semente da lista de comparação e entra em um while que compara todos os elementos da lista e os retira após a comparação até que não sobre mais nenhum na lista. Finalizada toda a comparação da imagem é realizada a construção de uma nova imagem,

agora no formato de PPM, pelo comando printf. O programa principal também libera toda a memória alocada.

Organização do Código, Decisões de Implementação e Detalhes Técnicos

O código esta dividido em três arquivos principais: tp1.c com o programa principal, imagem.h e imagem.c com o TAD.

Para fazer o programa utilizei uma lista, no entanto pode se notar que essa lista encadeia tem um comportamento peculiar que já conhecemos, tal comportamento recebe o nome de fila, por sempre adicionar no final e retirar o início.

O programa utilizado para escrever o código foi o Notepad++ e o compilador foi o Prompt de Comando, também conhecido como cmd, no sistema operacional Windows 7. Para executar o código basta usar o cmd e entrar na pasta em que o arquivo está, a seguir digitar "gcc tp1.c –Wall imagem.c" e aguardar gerar o executável, com ele em mãos, é necessário digitar "a.exe nome_da_imagem_sem_a_exetensao", assim será gerado um novo arquivo no formato PPM, se seu visualizador não for capaz de ver a imagem, você poderá vê-las no seguinte link: http://paulcuth.me.uk/netpbm-viewer/.

3. Estudo de complexidade

Na analise de complexidade será feita em função das variáveis **N** e **M**, as quais representam o número de colunas e linhas da matriz.

void criaLista (Lista *lista): a função **criaLista** ela é chamada apenas uma vez no programa e realiza um número fixo de atribuições, realiza 4 atribuições. Portanto a função **O(1)**.

void insere(Lista *lista, matriz *resolucao): a função **insere** realiza um número fixo de atribuições e comparações, no melhor caso ela realiza 1 comparação e 2 atribuições, no pior caso 1 comparação e 3 atribuições. Portanto a função **O(1)**.

void retira (Lista *lista): a função **retira** realiza atribuições e comparações um numero fixo de vezes, no melhor caso são 0(zero) comparações e 0(zero) atribuições e no pior caso 2 comparações e 3 atribuições. Portanto possui a complexidade de **O(1)**.

void compara(Lista *lista, matriz **resolucao, local semente, int T, int l, int c, int coluna, int linha): a função **compara** realiza a comparações e atribuições um numero fixo de vezes, que no melhor caso é 1 atribuição e 4 comparações, no pior caso são 25 atribuições e 16 comparações, ademais no pior caso ela ainda chama 4 vezes uma função **O(1)**. Portando conclui-se que essa função é **O(1)**.

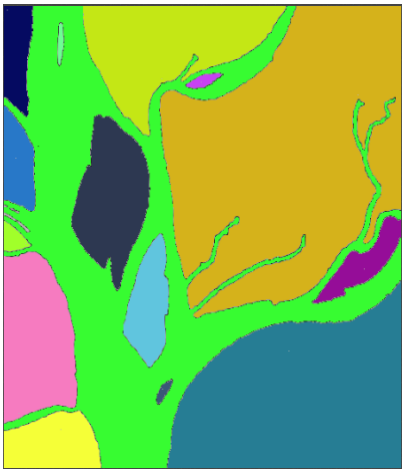
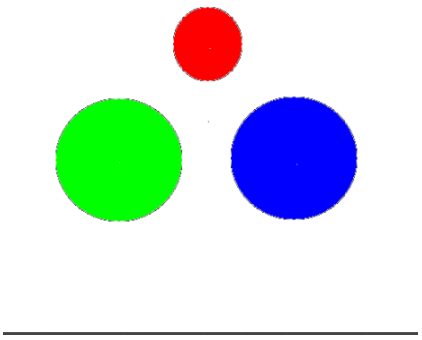
Programa Principal: o programa principal possui alguns loops dentro de outros e também loops sozinhos com números de vezes executadas diferentes, no entanto os todos os loops teriam no pior caso que rodar o número máximo de vezes que seria o tamanho da matriz, ou seja, **N*M** para matrizes não quadradas e N^2 para matrizes quadradas. Existe também um loop que possui um while com algumas chamadas de funções **O(1)**, porém o pior caso seria ter um número $\mathbf{N^2}$ de sementes e uma matriz quadrada **NxN**, no entanto apesar de na primeira vez que o loop rodar o while poder executar $\mathbf{N^2}$ vezes nas outras $\mathbf{N^2-1}$ vezes que o loop rodar ele nem entraria no while porem ainda chamaria a função compara e retira todas as $\mathbf{N^2-1}$ **vezes**, porque a lista seria sempre vazia, pois todas as posições da matriz já foram visitadas, ou seja teríamos para esse loop uma função de complexidade do tipo:

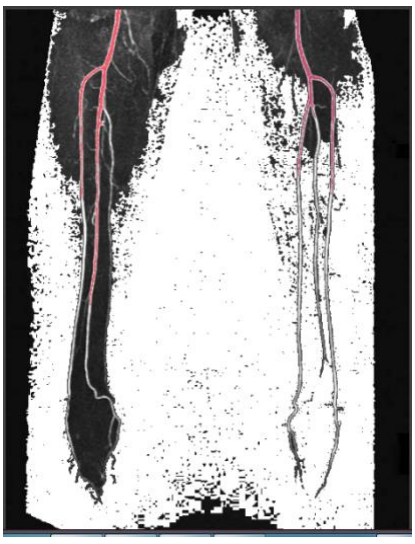
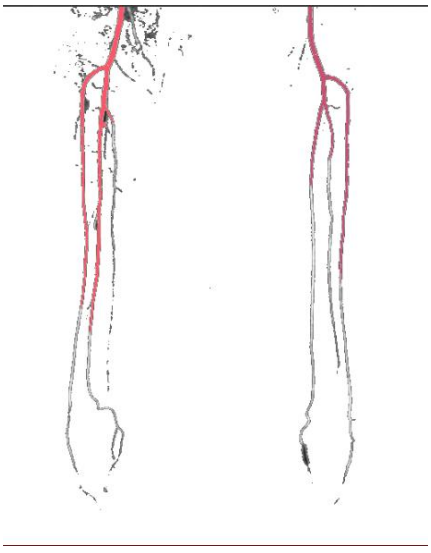
$f(N) = 1*N^2*2*2+(N^2-1)*(3+2)+(3+2) = 4N^2+5N^2-5+5 = 9N^2 = O(Max(N^2,1) = O(N^2)$

Portando a complexidade do algoritmo é dada pelo maior custo operacional dentro do programa, ou seja, o algoritmo tem custo de **$O(N^2)$** para matrizes NxN e custo **O(N*M)** para matrizes NxM.

4. Testes

Foram realizados vários testes para averiguar a execução do código. Os testes foram realizados em um Lenovo g400s com memória de 4GB. Segue a imagens geradas pelo programa.

Obs.: a quarta imagem é dada quando uso o limiar do arquivo auxiliar e a quinta imagem é dada ao aumentar o limiar para 50. No entanto, a quinta imagem é a que mais se aproxima da imagem que esta no enunciado do TP1.

5. Conclusão

A implementação do trabalho foi feita sem maiores problemas e os resultados então dentro das expectativas, apesar de uma saída possuir um desvio ele pode ser corrigido mudando o limiar da imagem para 50, como observado anteriormente. A maior dificuldade encontrada foi no momento de atribuir os endereços aos vetores e na retirada o último elemento da lista, porém foi corrigido posteriormente.

Referências

[1] Ziviani, Nívio, Projeto de Algoritmos com Implementações em Pascal e C, 3ª Edição revista e ampliada, Editora Cengage Learning, 2011.

[2] Backes, André, Livro - Linguagem C: Completa e Descomplicada, 1ª Edição, Editora Elsevier, 2012.

Anexos

Listagem dos programas

tp1.c
imagem.c
imagem.h