Trabalho Prático 01

João Moreno Rodrigues Falcão
2015058510

1. Introdução

Esse trabalho tem como objetivo exercitar os conhecimentos em modularização de códigos, recursividade, alocação dinâmica e estruturas de dados. A tarefa do programa é percorrer um arquivo PGM em busca de um pixel de valor zero. Buscamos recursivamente e empilhamos a sua posição para manter a memória do caminho percorrido. Dessa forma utilizamos os conhecimentos sobre pilhas, para gerenciar o caminho percorrido; sobre recursividade, para implementar a lógica de busca do pixel de valor zero; e sobre abertura e fechamento de arquivos, para ler a imagem.

2. Implementação

O código foi dividido em três módulos, Pilha, PGM e Recursiva. No módulo Pilha está implementada a estrutura de dados Pilha, com todas as funções necessárias para a sua manipulação. No módulo PGM está a estrutura de dados para imagens PGM, com as funções de leitura e escrita implementadas. No módulo Recursiva encontramos as funções responsáveis por gerenciar as chamadas de outros módulos e realizar toda a lógica recursiva do programa.

3. Análise de Complexidade

- **Módulo Pilha**

  As funções desse módulo tem complexidade sempre linear ou constante, pois se tratam de funções de alocação de memória e de busca sequencial em pilhas.

- **Módulo PGM**

  As funções desse módulo tem complexidade constante ou linear com relação ao número de pixels da imagem, pois se tratam de funções de alocação e desalocação de memória para armazenar imagens do tipo PGM.

- **Módulo Recursiva**

  A função ImprimeCaminho tem complexidade linear com dependência no tamanho da pilha que recebe. Já a função BuscaCaminho tem complexidade no melhor caso constante, quando o pixel de valor 0 é o pixel inicial, e no pior caso é linear com relação ao número de pixels, pois no pior caso a função irá percorrer topos os pixels da imagem.

4. Testes

Os testes realizados foram um sucesso. O código foi compilado utilizando o comando:

```
$ gcc main.c recursiva.c pilha.c PGM.c -Wall -Werror -Wextra
```

E não gerou nenhum aviso ou erro. Quando estudado pelo programa Valgrind, ele também não gerou nenhum erro ou aviso, foi utilizado o comando:

```
$ valgrind --leak-check=full -v ./a.out input/teste1.pgm
```

A execução do programa gerou esse resultado:

```
$ ./a.out input/teste1.pgm


(0,0), (0,1), (1,1), (1,2), (2,2).
```

Que era o esperado. Quando executamos para o arquivo teste2 essa é a resposta:

```
$ ./a.out input/teste2.pgm


(0,0), (1,0), (2,0), (2,1), (1,1), (1,2), (0,2), (0,3), (1,3),
(1,4), (0,4), (0,5), (1,5), (2,5), (2,4), (3,4), (4,4), (4,3),
(3,3), (3,2), (4,2), (5,2), (5,1), (6,1), (7,1), (8,1), (8,0),
(9,0), (9,1), (9,2), (9,3), (9,4), (9,5), (8,5), (7,5), (7,6).
```

Que também era a resposta esperada.

5. Conclusão

O trabalho foi um sucesso. Exercitei os conteúdos programados e o resultado foi satisfatório.

Anexos

- main.c
- recursiva.c
- recursiva.h
- PGM.c
- PGM.h
- Pilha.c
- Pilha.h