Universidade Federal de Minas Gerais

Engenharia de Controle e Automação

Trabalho Prático 0

Trabalho referente à disciplina de Algoritmos e Estruturas de Dados II do primeiro semestre do ano de 2016.

Rodrigo Lima de Araujo

2015116383

Belo Horizonte

2016

1. Introdução

O objetivo desse trabalho é desenvolver um código na linguagem C, utilizando os conteúdos aprendidos em sala nas disciplinas de Algoritmos e Estruturas de Dados I e II, que seja capaz de, dadas duas imagens no formato PGM, detectar qual ponto em uma imagem é exatamente igual a outra.

2. Implementação

Inicialmente foi necessária a implementação de um TAD (Tipo Abstrato de Dados) para a manipulação de imagens no formato PGM. O TAD implementado possui as estruturas de dados PGM e Ponto, utilizadas para representar a imagem em PGM e um Ponto na imagem respectivamente.

```
5   typedef struct {//estrutura de dados para salvar a imagem PGM
6       unsigned char maximo;//Valor máximo que um pixel pode assumir
7       unsigned char **dados;//Matriz para guardar os valores de cada pixel
8       int c;//comprimento da imagem em pixels (número de colunas da matriz)
9       int l;//altura da imagem em pixels (número de linhas da matriz)
10  }PGM;
11  typedef struct{//estrutura de dados para indicar um ponto na imagem(matriz)
12      int x, y;//coordenadas x e y
13  }Ponto;
```

Figura 1 - Estruturas de Dados

Foram necessárias ainda a implementação de cinco funções para o correto funcionamento do TAD, depois foram implementadas mais duas para a conclusão da Atividade Extra, a *LePGM*, utilizada para a leitura de um arquivo do tipo PGM, a *JanelaDeslizante*, utilizada para varrer o conteúdo da imagem principal, procurando o ponto de *Match*, a *CorrelacaoCruzada*, utilizada para comparar o conteúdo da imagem principal com a secundária por meio do algoritmo de Correlação Cruzada, a *GeraSaida* utilizada para gerar um arquivo da extensão txt com o ponto inicial de *Match* e a *LiberaMemoria*, utilizada para liberar a memória alocada dinamicamente para as imagens. Para execução da Atividade Extra foram implementadas as funções *JanelaDeslizanteMod* e *CorrelacaoCruzadaMod*, a *JanelaDeslizanteMod* tem funcionamento parecido com a *JanelaDeslizante*, a diferença é que ao invés de chamar a *CorrelacaoCruzada*, chama a *CorrelacaoCruzadaMod*, além disso ao invés de comparar a Correlação Cruzada dos pontos para encontrar a maior, ela compara a Correlação Cruzada do ponto atual com 0 (zero), se essa comparação for verdadeira, o ponto de *Match* foi encontrado.

```
15  PGM* LePGM(char* entrada);
16  Ponto JanelaDeslizante(PGM *cena, PGM *obj);
17  int CorrelacaoCruzada(PGM *cena, PGM *obj, Ponto *p);
18  void GeraSaida(Ponto p, char *nomearquivo);
19  void LiberaMemoria(PGM *img);
20  Ponto JanelaDeslizanteMod(PGM *cena, PGM *obj);
21  int CorrelacaoCruzadaMod(PGM *cena, PGM *obj, Ponto *p);
```

Figura 2 - Assinatura das Funções

A função *PGM* LePGM (char* entrada)* tem como parâmetro de entrada uma *string* contendo o nome do arquivo PGM a ser lido. O parâmetro de saída é um ponteiro para a estrutura PGM contendo os dados lidos do arquivo (imagem) armazenados em

suas variáveis. Ela aloca memória dinamicamente para a estrutura PGM e para a matriz contendo os dados.

A função *Ponto JanelaDeslizante (PGM *cena, PGM *obj)* tem como parâmetros de entrada um ponteiro para estrutura PGM cena (principal) e um ponteiro para estrutura PGM objeto (secundária). E seu parâmetro de saída é um Ponto em que foi encontrado o valor de maior Correlação Cruzada (*Match*), para isso ela subdivide a imagem cena em pequenas imagens do tamanho da imagem objeto e então aplica o algoritmo de Correlação Cruzada para comparação dos valores, depois desliza por todas as linhas e colunas repetindo o processo e salvando o maior valor de Correlação Cruzada até que toda a imagem tenha sido testada.

A função *int CorrelacaoCruzada (PGM *cena, PGM *obj, Ponto *p)* tem como parâmetros de entrada um ponteiro para o *PGM* contendo a imagem principal (cena), um ponteiro para a imagem secundária (objeto) e um ponteiro para o ponto inicial da subdivisão atual da cena, ela retorna uma variável do tipo inteiro (*int*) contendo o valor da Correlação Cruzada atual. Essa função é utilizada exclusivamente no TAD, chamada pela função *JanelaDeslizante* para encontrar o ponto de *Match*. A *CorrelacaoCruzada* multiplica os pontos de índices iguais da imagem objeto e da subdivisão da imagem cena, depois soma os resultados de cada multiplicação, retornando esse valor como a Correlação Cruzada do ponto atual.

A função *void GeraSaida (Ponto p, char *nomearquivo)* tem como parâmetros de entrada um *Ponto* p e uma *string* contendo o nome do arquivo txt a ser gerado. Não possui parâmetros de saída. Ela simplesmente cria um arquivo txt com o nome informado pelo usuário (se um arquivo com mesmo nome informado já existe, esse arquivo é sobrescrito) e escreve as coordenadas do Ponto Inicial de *Match*.

A função *void LiberaMemoria (PGM *img)* tem como parâmetro de entrada um ponteiro para uma estrutura de dados do tipo *PGM* e não possui parâmetros de saída. Essa função libera a memória alocada dinamicamente para o tipo *PGM* para que não haja o *Leak* de memória.

3. Resultados

Foi necessária uma etapa grande de testes e ajustes para que o programa funcionasse corretamente. Inicialmente muitos erros de sintaxe e lógica foram detectados e corrigidos, depois, por um erro de interpretação na proposta do trabalho, as coordenadas x e y estavam sendo utilizadas de forma trocada (x para linhas e y para colunas), após uma leitura mais cuidadosa das especificações fornecidas pelo professor, esse pequeno problema foi corrigido. Após essa correção, o programa encontrava o ponto correto segundo o gabarito disponibilizado pelo professor, no entanto havia um *Leak* de memória que, após toda a execução do programa, inclusive encontrando o ponto de *Match*, fazia com que o prompt parasse de funcionar e tivesse que ser fechado. Após algumas noites mal dormidas e muita ajuda dos monitores, o erro foi descoberto, as estruturas *PGM* não estavam sendo completamente liberadas, e a linha de código "*PGM *img = (PGM*) malloc (sizeof(PGM));*" incluída na função *LePGM* juntamente com a outra linha "*free(img);*" incluída na função *LiberaMemoria* resolveram o problema.

4. Atividade Extra

Após leitura e testes utilizando o primeiro algoritmo de Correlação Cruzada proposto, notou-se que as vezes os valores mais altos de Correlação Cruzada poderiam ser possuídos por diferentes pontos, fazendo com que esse algoritmo não fosse tão eficiente. Para melhorar a eficiência do programa, várias alterações foram imaginadas. Nesse processo, ficou evidente que seria bem difícil implementar um algoritmo que fosse ótimo, ou seja, resolvesse o problema para toda e qualquer combinação de pixels da imagem. No entanto, uma alternativa mais eficiente foi encontrada. Essa solução consiste em uma mudança de operação na Correlação Cruzada, ao invés de uma multiplicação entre os itens do objeto e os do recorte de cena, foi feita uma subtração. Nesse caso a Correlação Cruzada tem que, necessariamente, resultar em 0 (zero). Ainda existem problemas com essa implementação, como no caso de que os itens do objeto e do recorte de cena tenham os mesmos valores, com índices diferentes. Contudo a possibilidade desse novo algoritmo falhar é muito menor do que a do anterior.

5. Conclusão

Levando em consideração as dificuldades encontradas, como por exemplo, muita manipulação com ponteiros, pequenos erros com alocação dinâmica de memória, primeiro contato com argumentos passados por linha de comando, interpretação da proposta, o Trabalho Prático 0 foi muito proveitoso. Alguns conceitos novos foram vistos (TAD) e outros, que precisavam de revisão, foram revisados (alocação dinâmica, manipulação com ponteiros). O resultado final, de acordo com os exemplos de *output* para cada *input* fornecidos pelo professor, está correto, o que é motivo de orgulho, uma vez que esse trabalho demandou bastante tempo e esforço.

5. Referências

CodingUnit, "*C Tutorial – The functions malloc and free".* Disponível em <https://www.codingunit.com/c-tutorial-the-functions-malloc-and-free>. Acesso em 10/04/2016;

ZIVIANI, Nivio "*PROJETO DE ALGORITMOS: COM IMPLEMENTAÇOES EM PASCAL E C*" 3ª Edição, 2010.