Contextualização do problema

Nonato Sales é aluno do mestrado em Engenharia elétrica da UFPI, seu orientador é o prof. Antonio Oseas. Cada **orientando** (**aluno**) possui um **orientador** (**discente**) e em alguns casos, tem-se o papel do **co-orientador** (**discente**). Um orientador pode orientar até 10 alunos, sendo, 40% de graduação e 60% a nível de pós-graduação (mestrado e doutorado).

TAD 1 – Cadastro

Desenvolva uma solução para efetuar os cadastros de **discente** e **docente** usando TAD e respeitando os limites impostos de cadastro. Deve-se ter atenção nas modularizações a serem seguidas**, espera-se que os conceitos e funcionalidades aprendidos em sala de aula possam ser aplicados na criação de TADs** para: **orientando** e **orientador** com as respectivas funções: **cadastrar, remover, altarar, buscar e mostrar),** além destas, outras funções deve ser implementadas, como: **listar alunos de um determinado orientador; listar alunos que não possuem orientador; mudar orientador de um determinado aluno**. Além disso, cada aluno cadastrado e alocado a um orientador, deverá ter uma senha de acesso, ou seja, pode haver alunos cadastrados sem orientador. A senha do aluno será usada para acessar as funcionalidades que serão definidas e implementadas nas TADs 2 e 3. Professores não precisam de senhas de acesso, ou seja, acessam qualquer struct sem restrições.

Structs a serem seguidas:

```
typedef struct pessoa Pessoa;
typedef struct docente Docente;
typedef struct discente Discente;
struct pessoa{
   char *nome;
   int idade, ID;//identificador
   int matricula;//deve ser gerada automaticamente
};
struct docente{
   Pessoa info_docente;
   int qtd_orientacoes_graduacao, qtd_orientacoes_pos_graduacao;
};
struct discente{
   Pessoa info_discente;
   int nivel;// 1- graduacao ou 2 - posgraduacao
   char *nome_curso;
   int senha, ID_orientador, ID_coorientador;
};
```

É importante usar os menus e submenus para facilitar a navegação dos usuários, pensem nisso também.

Não bastando apenas os cadastros, todos os discentes, assim como Nonato Sales, esperam-se que sejam alunos aplicados. Nonato possui seu trabalho de mestrado voltado ao processamento de imagens em duas dimensões. Seu orientador lhe passou algumas tarefas, sendo estas, implementadas conforme TADs, abaixo:

TAD 2 – Imagem

- Esta TAD representa a estrutura básica de uma imagem 2D, e esta, possui comportamentos e composição própria. A imagem é representada por uma matriz de Altura e Largura definidas por algum meio. Elas são compostas por *pixels*, cada pixel possui coordenadas x e y (2D) e valor de intensidade que podem variar entre 0 e 255, vejam as sctructs abaixo.

```
typedef struct pixel Pixel;
typedef struct imagem Imagem;
struct pixel {
   int x, y, valor_pixel;
};
struct imagem {
   int altura, largura;
   Pixel *pixels_imagem;//variar de 0 ate 255
};
```

Algumas funções que são importantes nesta TAD, a saber: **criar uma imagem (em tempo de execução), carregar imagem (de um arquivo txt), salvar imagem (em arquivo txt), criar cópia de imagem (em tempo de execução), salvar cópia de imagem (salvar em txt)** e outras que porventura, venham a necessitar.

O arquivo TXT com a imagem salva deve ter a seguinte estrutura: na primeira linha, estão as dimensões (altura (linhas) x largura (coluna)) da imagem, separadas por "espaço", exemplo: 10 18. O restante das linhas deve ser o conteúdo da imagem, ou seja, cada linha/coluna da matriz imagem, será uma linha/coluna no arquivo TXT. Os valores devem ser sempre separados por espaço para faciliatar a leitura em outros códigos.

TAD 3 – Funções para manipulação da imagem

- Máximo valor
- Mínimo valor
- Cálculo de distâncias
  - Euclidian
  - Manhattan
- Local Binary Pattern (LBP)

  - Esta é uma técnica que permite ao usuário gerar uma nova representação da imagem. Neste método cada pixel de uma imagem é substituído por um valor binário. Este valor é determinado pela comparação de uma matriz quadrada contendo os pixels vizinhos, onde cada vizinho é comparado com o valor central, conforme a seguinte condição: $b_{ij}\begin{cases}0, v_{ij} < v_c \\ 1, v_{ij} \geq v_c\end{cases}$

    onde *vij* é o valor de um *pixel* na posição(i,j) e *vc* é o valor central. Os valores obtidos para cada vizinho são concatenados e o número binário gerado é convertido na base decimal para substituir o valor central vc. A Figura 1 exemplifica este pro- cesso, para uma matriz 3 × 3 (considerar este valor para implementação) de pixels vizinhos. Contudo, o tamanho e o formato da vizinhança podem variar.

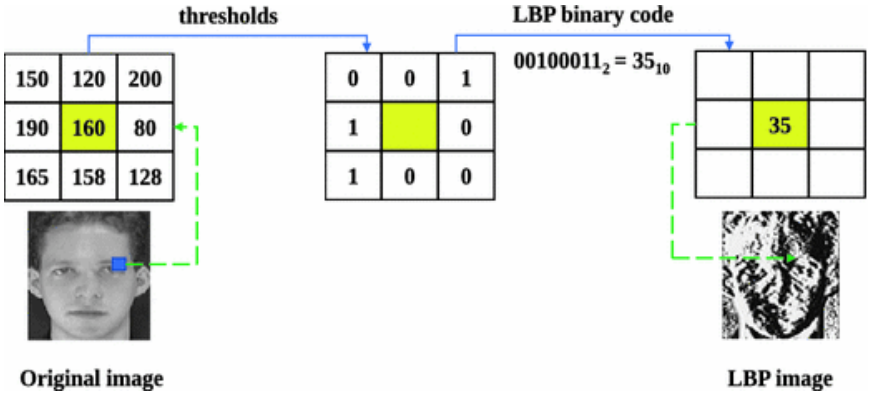
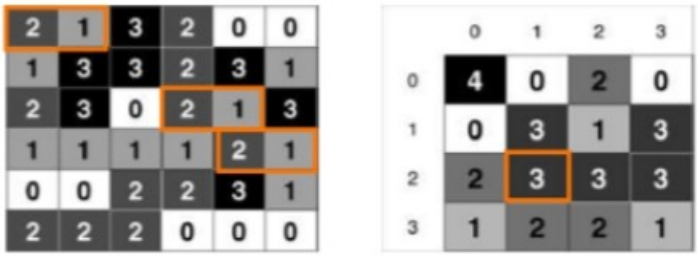


Figura 1 - exemplo do funcionanto do LBP para um pixel

- Matriz de Coocorrência
  - A matriz de coocorrência, do inglês Gray-Level Co-Occurrence Matrix (GLCM), foi criada, inicialmente, para extração de atributos visando descrever texturas em imagens. A matriz de coocorrência C é uma matriz quadrada de dimensões Ng × Ng que mapeia o número de transições de intensidades entre pixels de um imagem I, segundo um critério de vizinhança entre eles. Sejam i,j = {0,1, · · · ,Ng − 1} as intensidades possíveis, os elementos da matriz C são definidos por:

$$\mathfrak{C}_{ij} = \#\{(i,j) : q \in V_p(d,\theta), i = \mathbf{I}(p), j = \mathbf{I}(p+q)\} \quad \forall p \in \mathcal{D}_I$$

em que Vp é uma função de vizinhança do pixel p, dado um deslocamento d ∈ Z+ e um ângulo θ em uma das quatro direções possíveis, ou seja, θ ∈ {0◦, 45◦, 90◦, 135◦}. I(p + q) é uma translação de I(p) por q. O operador #{·} denota a cardinalidade, ou seja, o número de elementos do conjunto.



(b) **I**, $N = 4$ (d) GLCM

(b) **I** (d) $\mathfrak{C}$ para $d = 1, \theta = 0°$

Figura 2.3: (a) Uma imagem contendo $N_g = 4$ níveis de cinza. (b) Exemplo de GLCM para a vizinhança de um pixel à direita. O elemento em destaque registra o número de transições do par de intensidades (2,1).

é importante destacar, que a direção será escolhida pelo usuário no momento de usar a função.

- Filtro da média
  - O filtro de media e implementado da seguinte maneira , temos uma janela que percorrerá toda a imagem, o elemento central dessa janela recebera a media de todos os elementos da janela. Quanto maior for a janela, mais influencia dos vizinhos este pixel sofrerá e maior será o efeito de bluring, visto que levaremos em consideração um numero maior de pixels. Por exemplo vejamos a janela 3x3 abaixo:

$$\begin{bmatrix} 244 & 247 & 245 \\ 252 & 12 & 238 \\ 244 & 245 & 250 \end{bmatrix}$$

A media dos valores será: (12 + 238 + 244 + 244 + 245 + 245 + 247 + 250 + 252)/9 = 219. Assim o valor desse *pixel* que era 12 será de 219

- Filtros de mediana
  - Os filtros de mediana reduzem o bluring e preserva a edging coloca o valor da mediana no elemento do meio o tamanho da janela importa. A desvantagem principal do filtro de mediana em uma vizinhança retangular é o dano causado nas linhas finas e curvas agudas. O pixel é substituído pelo valor médio de seus vizinhos , caso o tamanho da sua janela seja par , então o valor da mediana será a media dos dois valores centrais. Esse filtro é um dos melhores filtros de suavização que preserva o contorno. Nos pegamos a mediana ordenando em ordem crescente ou decrescente, os valores dos pixels e pegamos o valor médio . Por exemplo: Dada a seguinte janela 3x3

$$\begin{bmatrix} 244 & 247 & 245 \\ 252 & 12 & 238 \\ 244 & 245 & 250 \end{bmatrix}$$

Ordenando os valores teremos: 12 238 244 244 245 245 247 250 252. O quinto valor será a mediana, ou seja, 245.

O problema do Nonato está longe de ser resolvido, entretanto, essas funções servirão de base para que ele construa parte de sua solução final.

Algumas observações:

- É importante que as funções implementadas na TAD estejam simples/abstratas ao ponto de outro usuário usá-las, sem a necessidade de detalhes de implementação.
- As structs definidas acima devem ser consideradas igualmente estão definidas, facilitando o reuso por outros códigos.