UNIVERSIDADE FEDERAL DE SERGIPE

Departamento De Computação - DCOMP
COMP0432 - Processamento de Imagens (2019.2 - T02) - BeatrizTrinchão

Relatório trabalho Processamento de imagem

Vitor Hugo Ribeiro Tiburtino De Melo
Kaio Henrique da Costa Ferreira

São Cristóvão-SE, janeiro de 2019

1 - Introdução

Com o intuito de promover a aplicação dos assuntos vistos em sala, foi pedido como resultado final do projeto abrir e salvar um arquivo do tipo .pbm, a retirada dos ruídos das imagens. Além da contagem de linhas, colunas e detecção de palavras. Como primeiro passo para realização deste trabalho, foi feita uma análise sobre os conhecimentos gerais dos alunos do grupo nas linguagens de programação, foi então decidido utilizar JAVA, a tarefa para abertura do arquivo ficou com o aluno Kaio além da criação do método de dilatação e a detecção de palavras, e parte de salvar e tirar o ruído com o aluno Vitor Hugo, além da contagem de linhas e colunas. Após alguns contratempos enfim foi realizada esta tarefa com êxito.

2 - Atividades Realizadas

Após a definição da linguagem utilizada foi feito uma série de testes para verificar os métodos de abrir, editar e salvar em um arquivo em java. Em seguida como é impossível calcular o número de linhas e colunas com os ruídos, foi feito uma análise nos tipos de filtros, para saber qual seria mais eficaz para a retirado dos ruídos. Com a retirada dos ruídos foi realizada a contagem de número de linhas e colunas, além da detecção de palavras.

2.1 - Abrir e salvar arquivos .pbm

Na classe chamada manipulador de arquivos foram feitos dois métodos um medo que lê o arquivo e já gera uma matriz a partir dele:

```java
public static int[][] geraMatriz(String path) throws IOException {
    int linha = 0;
    int coluna = 0;
    int i = 0;
    int[][] matriz = null;

    String[] dados;

    BufferedReader arquivo = new BufferedReader(new FileReader(path));
    String info = "";
    while (true) {
        //tratamento de excessão
        if (info == null)
            break;
        //comparação para o poder ser lida a primeira linha
        else if (info.equals(""))
            info = arquivo.readLine();
        else {
            //Este if é feito para adquirir o tamanho das linhas e colunas
            if (i == 2) {
                //é feito um split para separ o valor das linhas e colunas sendo parametro o espaco
                dados = info.split(" ");

                //onde é recebido o valor da coluna, em string e é transformado para inteiro;
                coluna = Integer.parseInt(dados[0]);// 1653
                //onde é recebido o valor da linha, em string e é transformado para inteiro;
                linha = Integer.parseInt(dados[1]);// 2338

                //é definida o tamanho da matriz com base nos valores recebidos anteriormente
                matriz = new int[linha][coluna];

                linha = coluna = 0;

            }
```

```java
            //repetição para poder lê o valor de cada pixel
            if (i > 2) {
                //é percorrido a linha recebido do arquivo
                for (int j = 0; j < info.length(); j++) {
                    //é feita uma substituição dos \n presente no arquivo por vazio,
                    //caso não seja feito este procedimento o tamanho da imagem tomará formas
                    //diferentes, pois acabara tendo menos valor do que devia.
                    info = info.replaceAll("\n", "").replace("\r", "");
                    //é recebido o valor do pixel em string e transformado em inteiro, sendo salvo na posição da matriz
                    matriz[linha][coluna] = Integer.parseInt(String.valueOf(info.charAt(j)));

                    //acrescimo da coluna, para percorrer o arquivo
                    coluna++;

                    //feito uma verificação, caso o valor da coluna seja igual ao da largura da matriz,
                    //ele 0 o valor da coluna e aumenta +1 o valor da linha.
                    if (coluna == (matriz[0].length)) {
                        coluna = 0;
                        linha++;
                    }
                }
            }
        }

        //lê uma nova linha do arquivo.
        info = arquivo.readLine();
        i++;

    }
    //efetua o fechamento do arquivo.
    arquivo.close();
    //retorna a matriz.
    return matriz;
}
```

E o método que escreve no arquivo uma matriz:

```java
public static void escritor(int matriz[][]) throws IOException {

    BufferedWriter buffWrite = new BufferedWriter(new FileWriter("novaImagem.pbm"));
    buffWrite.append("P1 \n");
    buffWrite.append("# CREATOR: GIMP PNM Filter Version 1.1\n");
    buffWrite.append(matriz[0].length + " " + matriz.length + "\n");
    for(int i = 0; i < matriz.length; i++) {
        for(int j = 0; j < matriz[0].length; j++) {
            buffWrite.append(Integer.toString(matriz[i][j]));
        }
        buffWrite.append("\n");
    }

    buffWrite.close();

}
```

2.2 Retirar o sal e pimenta

Em uma busca sobre qual seria o melhor filtro para poder realizar a retirada do sal e pimenta, foi descoberto que o filtro da mediana é ideal para a realização deste problema, nele é selecionando um determinado pixel, que será o pixel do cálculo no momento. Sendo que o valor da mediana é o valor do pixel selecionado de um conjunto qualquer (Em nosso caso é um conjunto de 9 números, no qual foi escolhido o algoritmo insertion Sort para realização da ordenação crescente, por ter uma maior eficiência neste caso segundo [Sousa et. al. 2017]), organizados em ordem de grandeza. No trabalho de [Jain et.al 1995] é afirmado que este filtro possui grande eficiência para a realização desta tarefa, logo foi utilizado este filtro, porém ele suaviza um pouco as letras, mas para solucionar este problema foi passado o tratamento de dilatação. A seguir segue o método de filtro da mediana e o de dilatação :

```java
public static int[][] filtroMediana(int matriz[][]) {

    //definida a mascara
    int mask[] = new int[9];
    //definida a matriz que será salvo os pixles depois da realização da mediana.
    int matrizMediana[][] = new int[matriz.length][matriz[0].length];

    //for para percorrer as linhas
    for (int linha = 0; linha < matriz.length; linha++) {
        //for para percorrer as colunas;
        for (int coluna = 0; coluna < matriz[0].length; coluna++) {
            //verificação para constar que a máscara 3x3 estará contida toda na img;
                    if ((matriz.length - 1) > linha && linha > 0 && (matriz[0].length - 1) > coluna && coluna > 0) {
                        //foi decidido setar de vez os valores no vetor, para evitar a utilização de dois for's.
                        mask[0] = matriz[linha - 1][coluna - 1];
                        mask[1] = matriz[linha - 1][coluna];
                        mask[2] = matriz[linha - 1][coluna + 1];
                        mask[3] = matriz[linha][coluna - 1];
                        mask[4] = matriz[linha][coluna];
                        mask[5] = matriz[linha][coluna + 1];
                        mask[6] = matriz[linha + 1][coluna - 1];
                        mask[7] = matriz[linha + 1][coluna];
                        mask[8] = matriz[linha + 1][coluna + 1];

                        //chama a função insertionSort para ordenar os valores;
                        insertionSort(mask);
                        //após a ordenação, é escolhido o pixel do meio para guarda na posição na matriz.
                        matrizMediana[linha][coluna] = mask[4];

                    } else
                        matrizMediana[linha][coluna] = 0;
                }
            }


    return matrizMediana;
}
```

```java
public static void insertionSort(int[] vet) {
    for (int i = 0; i < vet.length; i++) {

        int num = vet[i];

        for (int j = i - 1; j >= 0 && vet[j] > num; j--) {

            vet[j + 1] = vet[j];
            vet[j] = num;

            }
        }
    }
```

```java
//método de dilatcão em linha horizontal
public static int[][] dilatacao(int matriz[][]) {

    int novaMatriz[][] = new int[matriz.length][matriz[0].length];
    //for para percorrer as linhas
    for (int i = 0; i < matriz.length; i++) {
        //for para percorrer as colunas
        for (int j = 0; j < matriz[0].length; j++) {
            //if de verificação para caso o elemento estrutucante caiba completamente na matriz
            if ((i - 1) > 0 && (i + 1) < matriz.length) {
                if (matriz[i - 1][j] == 1 || matriz[i][j] == 1 || matriz[i + 1][j] == 1)
                    novaMatriz[i][j] = 1;
                else
                    novaMatriz[i][j] = 0;
                //if para caso o elemento estrutucante não caiba totalmente na matriz
            } else if ((i - 1) < 0) {
                if (matriz[i][j] == 1 || matriz[i + 1][j] == 1)
                    novaMatriz[i][j] = 1;
                else
                    novaMatriz[i][j] = 0;

                //if para caso o elemento estrutucante não caiba totalmente na matriz

            } else if ((i + 1) > matriz.length)
                if (matriz[i - 1][j] == 1 || matriz[i][j] == 1)
                    novaMatriz[i][j] = 1;
                else
                    novaMatriz[i][j] = 0;
        }
    }

    return novaMatriz;
}
```

2.4 - Contagem de linhas e colunas

O programa procura na matriz as linhas que tenham pelo menos um 1 e após essa linha com 1s tenha uma linha apenas com 0s. Com isso o programa determina que ali há uma linha no texto. O de colunas basicamente realiza a mesma função com a alteração de procurar pela coluna com um intervalo maior de 0s na coluna.

2.4.1 Contagem de linhas:

```
// Aqui é feito a procuca das linhas do arquivo que tem 0 e que tem pelo menos um 1 na linha inteira
for(int i = 0; i < matrizImagem.length; i++) {
    for(int j = 0; j < matrizImagem[0].length; j++) {
        if(matrizImagem[i][j] == 1) vetorLinhas[i] = 1;
    }
}
```

```
// contabilizando uma sequencia de 1 terminando com uma linha em zero no vetor, significa que tem uma linha de texto
for(int i = 1; i < vetorLinhas.length; i++) {
    if(vetorLinhas[i] == 1 && vetorLinhas[i - 1] == 0) linhas++;
}
```

2.4.2 Contagem de colunas:

```
// O mesmo é feito aqui, só que por coluna ele procuca qual coluna tem apenas 0 e qual coluna tem pelo menos um 1
for(int i = 0; i < matrizImagem[0].length; i++) {
    for(int j = 0; j < matrizImagem.length; j++) {
        if(matrizImagem[j][i] == 1) vetorColunas[i] = 1;
    }
}
```

```
// o mesmo feito para a contagem de colunas de texto, só que com um intervalo maior
for(int i = 7; i < vetorColunas.length; i++) {
    if(vetorColunas[i] == 1 && vetorColunas[i - 1] == 0 && vetorColunas[i - 2] == 0 && vetorColunas[i - 3] == 0 &&
            vetorColunas[i - 4] == 0 && vetorColunas[i - 5] == 0 && vetorColunas[i - 6] == 0 && vetorColunas[i - 7] == 0) {
        colunas++;
    }
}
```

.

2.5 Detecção de palavras

Para realizar a detecção de palavras, foi utilizado o vetor das linhas para fazer o recorte de linhas que tinha no texto. Com esse recorte feito a análise coluna a coluna, buscando as palavras para ser pontilhado a posição dos lados, acima e abaixo. Assim fazendo a detecção de todas as palavras do texto.

```java
// aqui ele destaca uma linha de texto com a ajuda da contagem de linhas,
//e depois e feito um feito um recorte das palavras circulando cada uma separadamente,
//utiliza uma contagem de coluna para verificar a separacão das palavras
public static int[][] palavras(int matriz[][], int vetor[]) {
    int aux = 0;
    int x = 0;
    for(int i = 0; i < (vetor.length - 1); i++) {
        if(vetor[i] == 0 && vetor[i+1] == 1) {
            aux = i;
        } if(i > 0) {
            if(vetor[i] == 0 && vetor[i-1] == 1 && aux != 0) {
                int vetorCol[] = new int[matriz[0].length];

                for(int j = 1; j < (vetorCol.length - 1); j++) {
                    for(int k = aux; k <= i; k++) {
                        if(matriz[k][j] == 1) vetorCol[j] = 1;
                    }
                }
```

```java
                for(int j = aux; j <= i; j++) {
                    for(int k = 5; k < (vetorCol.length - 5); k++) {
                        if(vetorCol[k] == 0 && vetorCol[k-1] == 0
                                && vetorCol[k-4] == 0 && vetorCol[k-5] == 0
                                && vetorCol[k-3] == 0 && vetorCol[k-2] == 0
                                && vetorCol[k+1] == 1) {
                            matriz[j][k] = 1;
                            x = k;
                        } if(k > 0) {
                            if(vetorCol[k] == 0 && vetorCol[k+1] == 0
                                    && vetorCol[k+4] == 0
                                    && vetorCol[k+3] == 0
                                    && vetorCol[k+5] == 0
                                    && vetorCol[k+2] == 0
                                    && vetorCol[k-1] == 1) {
                                matriz[j][k] = 1;

                                for(int l = x; l <= k; l++) {
                                    matriz[aux][l] = 1;
                                    matriz[i][l] = 1;
                                }
                            }
                        }
                    }
                }
            }
        }
    }

    return matriz;
}
```

2.6 - Dificuldades

Ao longo do trabalho foi encontrado dificuldades na leitura do arquivo onde o \n acabava alterando o número de caracteres. Houve também uma confusão na hora da leitura do tamanho das imagens, onde foi trocado os valores das linhas e colunas, proporcionando uma imagem cheia de erro falhas ao passar o filtro. A seguir imagem que exemplifica este erro:

Uma segunda dificuldade foi encontrada na hora de realizar a contagem de colunas, no qual alguns ruídos pimentas ainda restaram na imagem. Para solucionar foi passado uma segunda vez o filtro da mediana.

3 - Conclusão

Com o intuito de realizar o processamento de imagens foi realizado a abertura e o fechamento da imagem, aplicando um filtro de mediana para retirar os ruídos de sal e pimenta. Foi feita uma contagem sobre as linhas e colunas da matriz comparando os 1 para definir as linhas e colunas de imagem. Além da realização de palavras, sendo que em alguns casos foi circulado até pontos.

4 - Referências

SANCHES, Carlos H. et al. Técnicas de Suavização de Imagens e Eliminação de Ruídos. **Anais do EATI–Encontro Anual de Tecnologia da Informação. Frederico Westphalen–RS**, p. 21-30, 2015.

SOUZA, Jackson EG; RICARTE, João Victor G.; DE ALMEIDA LIMA, Náthalee Cavalcanti. Algoritmos de Ordenação: Um estudo comparativo. **Anais do Encontro de Computação do Oeste Potiguar ECOP/UFERSA (ISSN 2526-7574)**, n. 1, 2017.