MAC0122 - Princípios de Desenvolvimento de Algoritmos
EP02

Paulo Miranda & Marco A. Gerosa

[1]Instituto de Matemática e Estatística (IME)
Universidade de São Paulo (USP)

1. Estrutura dos arquivos de imagens no formato PPM

A reprodução de cores em dispositivos eletrônicos, como monitores de computador, "datashows", scanners e câmeras digitais, segue o padrão do sistema de cores RGB. No sistema RGB, cada cor é definida pela quantidade de vermelho (Red em inglês), verde (Green em inglês) e azul (Blue em inglês) que a compõem. Cada cor no sistema RGB é identificada por uma tripla ordenada (r, g, b) de números inteiros, com $0 \leq r \leq 255$, $0 \leq g \leq 255$ e $0 \leq b \leq 255$. Uma imagem colorida (ex: fotografia digital) e bidimensional pode ser representada por três matrizes $R(x, y)$, $G(x, y)$ e $B(x, y)$, com M linhas e N colunas, correspondendo aos canais das componentes vermelho, verde e azul, respectivamente.

Existem diversos formatos de arquivo para armazenar uma imagem digital colorida em um computador (ex: JPG, PNG, PCX, BMP). O formato PPM (*Portable Pixel Map*) é o formato mais simples existente. Esse formato não usa algoritmos de compressão dos dados. O formato PPM do tipo P3 é simplesmente um arquivo texto que pode ser visualizado em um editor de textos. Cada imagem PPM é composta por:

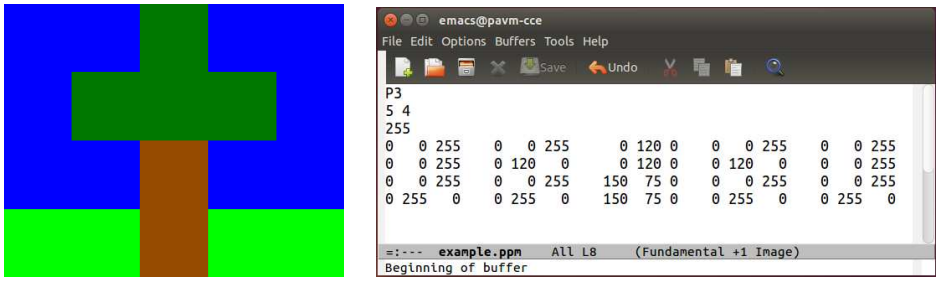
1. Uma primeira linha identificando o tipo de arquivo. Sempre será **P3** no nosso caso.
2. Opcionalmente, uma linha com comentários iniciada com o caracter **'#'**. Por questões de simplicidade, considere que as imagens que iremos trabalhar não possuem esta linha.
3. Uma linha contendo as dimensões da imagem: o número de colunas (largura) e o número de linhas (altura).
4. A próxima linha armazena o valor máximo da escala de intensidade para as componentes das triplas ordenadas. Em geral, será usado o valor 255.
5. Na sequência temos as triplas ordenadas das cores dos pixels que são armazenadas percorrendo as matrizes na ordem da esquerda para a direita e de cima para baixo.

Portanto, um arquivo PPM possui a seguinte estrutura interna:

```
P3
Largura Altura
255
Quantidade de vermelho do pixel (0,0)
Quantidade de verde do pixel (0,0)
Quantidade de azul do pixel (0,0)
Quantidade de vermelho do pixel (1,0)
Quantidade de verde do pixel (1,0)
Quantidade de azul do pixel (1,0)
Quantidade de vermelho do pixel (2,0)
Quantidade de verde do pixel (2,0)
Quantidade de azul do pixel (2,0)
```

```
...
Quantidade de vermelho do pixel (x,y)
Quantidade de verde do pixel (x,y)
Quantidade de azul do pixel (x,y)
...
Quantidade de vermelho do pixel (Largura -1, Altura -1)
Quantidade de verde do pixel (Largura -1, Altura -1)
Quantidade de azul do pixel (Largura -1, Altura -1)
```

Na Figura 1 temos um exemplo de uma pequena imagem colorida e do seu arquivo texto correspondente no formato PPM do tipo P3. Para mais informações sobre o formato PPM consulte a página: `http://netpbm.sourceforge.net/doc/ppm.html`.



(a) (b)

Figura 1: (a) Exemplo de uma imagem 5 x 4, e (b) seu arquivo correspondente no formato PPM sendo visualizado em um editor de textos (*emacs*). As seguintes triplas ordenadas (r, g, b) de cores foram usadas: azul (0,0,255), verde escuro (0,120,0), verde (0,255,0), e marrom (150,75,0).

2. Histograma de cores

Um histograma, também conhecido como distribuição de frequências, é um agrupamento de dados em classes, de tal forma que contabilizamos o número de ocorrências em cada classe.

Considere uma imagem colorida, com cores (r, g, b) compostas por números inteiros no intervalo de 0 a 255 (isto é, $0 \leq r \leq 255$, $0 \leq g \leq 255$ e $0 \leq b \leq 255$). Temos um total de $256 \times 256 \times 256 = 16.777.216$ cores possíveis. A fim de obter uma representação mais compacta da distribuição de cores presentes em uma imagem, consideramos para cada cor (r, g, b), com valores de 0 a 255, uma nova cor correspondente (r', g', b') com valores quantizados em 4 níveis, isto é, números de 0 a 3 obtidos por $(r', g', b') = (r/64, g/64, b/64)$. Com isso, teremos agora na nova quantização um total de $4 \times 4 \times 4 = 64$ cores. Podemos obter então um histograma compacto onde as classes correspondem a essas 64 cores. Esse histograma pode ser armazenado em um vetor $H[k]$, $k = 0, ..., 63$. Para uma dada cor (r', g', b'), o seu índice k correspondente é dado por $k = r' + 4 \times g' + 16 \times b'$. Dividindo o número de observações de cada cor pelo número total de pixels da imagem (isto é, $M \times N$), obtemos o histograma normalizado das frequência relativas.

3. Recuperação de Imagens por Conteúdo

Com o advento da internet e o aumento da quantidade de imagens digitais disponíveis, métodos de busca de imagens em grandes bases de dados se tornaram necessários. A recuperação de imagens

por conteúdo (*Content-based image retrieval* - CBIR) consiste na busca de imagens similares a uma imagem fornecida. A ideia é encontrar a relação de imagens similares, ranqueadas por sua similaridade em relação à imagem de busca.

Dada a imensa quantidade de imagens, para que esse processo seja viável precisamos de meios rápidos que permitam comparar imagens a um baixo custo computacional. Para isso, durante as comparações, não usamos diretamente as imagens, mas sim apenas algumas características compactas extraídas delas, os chamados descritores de imagem.

Na sua forma mais simples podemos usar o histograma normalizado de cor da imagem para fins de uso como descritor. Nesse caso, a dissimilaridade entre duas imagens pode ser medida por meio da distância euclidiana entre os seus histogramas, vistos como pontos em um espaço com 64 dimensões:

$$d(H_1, H_2) = \sqrt{\sum_{k=0}^{63} (H_1[k] - H_2[k])^2} \tag{1}$$

4. Atividade

Faça um programa que lê várias imagens coloridas. Para cada imagem o programa deve calcular o seu histograma normalizado de cor com 64 posições. O programa deve guardar os histogramas em uma lista ligada, juntamente com os nomes dos seus arquivos de imagem correspondentes. As matrizes das imagens não devem ser guardadas na lista, e devem ser liberadas após o cálculo dos seus histogramas.

Na sequência, o usuário fornece o nome do arquivo da imagem de busca. O programa deve então comparar o histograma dessa imagem com os outros histogramas já calculados, imprimindo a relação dos nomes dos arquivos das imagens mais similares, ordenados pela sua distância. Para ordenar os arquivos de acordo com a distância em relação à imagem de busca, deve ser construída uma segunda lista ligada de "Ranking", e a ordenação é feita por meio de uma função que insere elementos na lista sempre em uma posição de modo a conservar ela ordenada.

No início do programa, para ler a base de arquivos de imagens, considere que o usuário digita o nome de um arquivo texto, que contém dentro dele o nome de todos arquivos das imagens (um nome de imagem por linha). Para facilitar, coloque as imagens na mesma pasta do seu código fonte, de modo a não ter que especificar o caminho até o arquivo da imagem.

Para vocês testarem o programa, no site do PACA estão disponíveis 60 fotos da Web de animais no formato PPM (20 imagens de leões, 20 imagens de tigres e 20 imagens de zebras), e 40 imagens de desenhos animados. A ideia é que se o usuário colocar como imagem de busca uma imagem de zebra, então o programa deveria ser capaz de pegar nas primeiras posições as imagens de outras zebras. Note, porém, que como o descritor usado é muito simples, ele vai errar em vários casos, tendo um melhor desempenho na base de imagens de desenhos animados.

Na implementação, vocês devem considerar as seguintes estruturas:

```
struct ImagemRGB{
  int **R;
  int **G;
  int **B;
  int m;
  int n;
};
```

```
struct Descritor{
  char filename[512];
  float *hist;
  struct Descritor *prox;
};

typedef struct Descritor * ListaDescritores;

struct Ranking{
  char filename[512];
  float dist;
  struct Ranking *prox;
};

typedef struct Ranking * ListaRanking;
```

Vocês devem implementar pelo menos as seguintes funções:

```
int **AlocaMatriz(int m, int n);
```

Faz a alocação dinâmica de uma matriz com m linhas e n colunas. Devolve o endereço do vetor de apontadores para as linhas da matriz.

```
void LiberaMatriz(int **M, int m);
```

Libera toda a memória alocada da matriz M com m linhas.

```
struct ImagemRGB *AlocaImagemRGB(int m, int n);
```

Faz a alocação dinâmica de uma estrutura para armazenar uma imagem colorida com m linhas e n colunas. A função deve alocar as três matrizes dos canais vermelho, verde e azul.

```
void LiberaImagemRGB(struct ImagemRGB *I);
```

Libera toda a memória alocada de uma imagem colorida.

```
struct ImagemRGB *AbreImagemRGB(char filename[]);
```

Faz a leitura de uma imagem no disco. A função deve fazer a alocação dinâmica da imagem, e preencher os seus campos com os dados presentes no arquivo PPM.

```
float *HistogramaCores(struct ImagemRGB *I);
```

Função que calcula o histograma normalizado de cores de uma imagem. O histograma deve ser alocado de modo dinâmico como um vetor do tipo **float** com 64 elementos.

```
float DistanciaHistogramas(float *hist1, float *hist2);
```

Função que calcula a distância euclidiana entre dois histogramas.

```
void LiberaDescritores(ListaDescritores L);
```

Função que libera toda a memória de uma lista ligada de descritores.

```
void LiberaRanking(ListaRanking L);
```

Função que libera toda a memória da lista ligada do "ranking".

```
ListaRanking InsereNoRanking(ListaRanking R,
                             float d,
                             char filename[]);
```

Função que insere na lista do ranking um novo arquivo, que possui valor de distância "d" em relação à imagem de busca. A função deve alocar uma nova estrutura do tipo **struct Ranking**, que deve ser inserida na lista ligada em uma posição adequada, de modo a conservar a lista ordenada em ordem crescente de distâncias. O apontador "R" aponta para a primeira estrutura da lista fornecida. A função deve devolver ao final, o endereço do início da lista após a operação da inserção.