

## Trabalho final ED1

Grupo: 03 alunos, mas com avaliação individual.

Uma imagem em tons de cinza pode ser vista como uma matriz de números inteiros de 1 byte

Faça um programa cujo executável chamará imm (imm.exe no windows; imm no linux)

O programa vai operar em linha de comando e terá as seguintes funcionalidades. Use argc e argv para lidar com os parâmetros

- imm -open file.txt
  - Abre uma imagem (formato texto) e mostra os valores dos pixels na tela
- imm -convert file.txt file.imm
  - Converte uma imagem no formato file.txt para o formato file.imm
- imm -open file.imm
  - Abra uma imagem (formato binária) e mostra os valores dos pixels na tela
- imm -segment thr file.imm segfile.imm
  - Faz o *thresholding* (limiarização da imagem) com um valor *thr* da imagem file.imm e escreve o resultado em segfile.imm
- imm -cc segfile.im outfile
  - Detecta os componentes conexos de uma imagem
- imm -lab imlab.txt imlabout.txt
  - Mostra o caminho a ser percorrido em um labirinto
- imm -outro-comando-que-não-existe
  - Mostra uma mensagem de erro de comando não encontrado

Etapas sugerida para construção do programa

- Etapa 1: um programa que é capaz de entender todos os comandos mas não vai executar nada ainda. Somente chamará funções vazias que mostrarão os argumentos passados pela linha de comando
- Etapa 2: comandos open (texto); convert; open (binário)
- Etapa 3: segment e componentes conexos
- Etapa 4: imagem labirinto

Condições

- Criar um TAD para lidar com as funções e operações da imagem baseando-se no TAD de matriz (pode-se alterar o TAD matriz ou usá-lo)
- Use os TADs criados ao longo do curso quando necessário
- Não utilize operações de entrada/saída em arquivos nas funções com operações específicas, como por exemplo a *segment* e *lab*. Modularize o seu código de forma que

exista somente uma função que faz leitura de arquivo e uma que faz escrita. Essas funções devem ser chamadas por outras quando necessário

- imm é o nome do programa que deve ser chamado em linha de comando
- .imm é o formato de arquivo binário que o programa imm saber ler
- No arquivo binário será necessário armazenar o tamanho da matriz
- No arquivo texto use o \n para descobrir o número de linhas da matriz

#### Como visualizar sua imagem

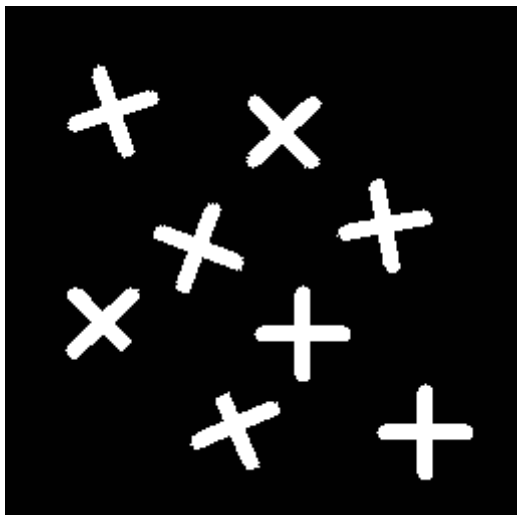
- É possível visualizar sua imagem facilmente utilizando um formato de imagem chamado pgm. Deixei no moodle um exemplo dessa imagem. Note que ela está no formato texto. Nesse formato, deve-se iniciar o arquivo com "P2", em seguida deve-se informar o tamanho da matriz (32 23) ; depois o valor máximo da matrix (255); e, por fim, a matriz com os valores dos pixels. Abra o arquivo no notepad para conferir.
- Não são muitos programas que leem esse formato. No windows o XNView consegue ler



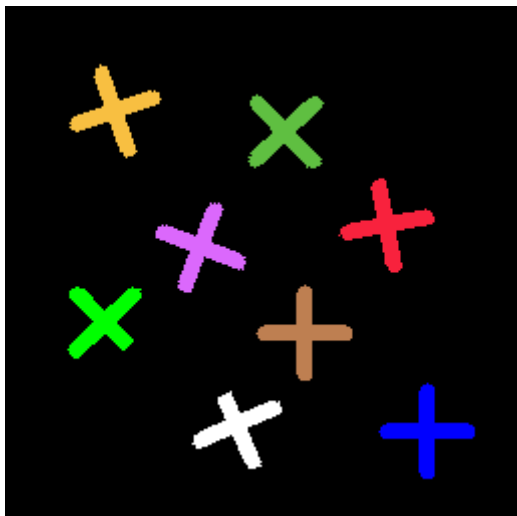
## Algoritmo para detecção de componentes conexos

A partir de uma imagem segmentada (i.e., que possui zeros e uns somente) podemos calcular seus componentes conexos, que são grupos de pixels

Por exemplo, na imagem abaixo temos 8 componentes conexos (8 objetos). Na rotulação de componentes conexos devemos dar um rótulo para cada pixel indicando qual o componente que ele pertence



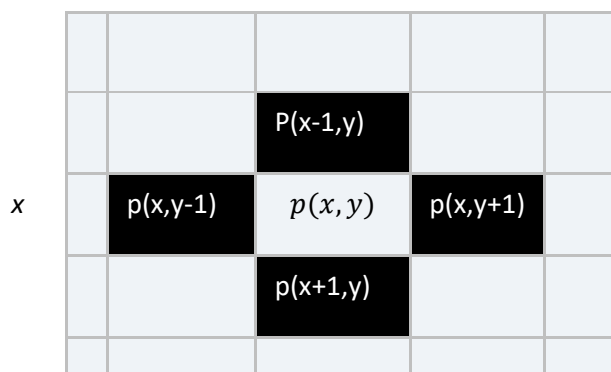
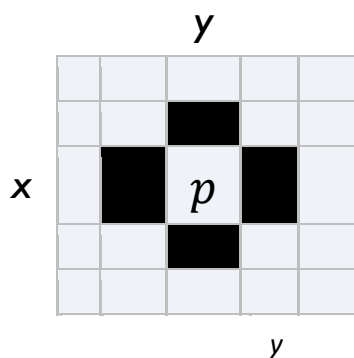
A imagem abaixo mostra o resultado após a aplicação da rotulação de componente conexo, em que cada objeto recebeu uma cor diferente (não será necessário mostrar a imagem colorida neste trabalho – a imagem abaixo é uma mera ilustração do processo)



Dessa forma, o algoritmo deverá percorrer a imagem e rotular cada objeto com um rótulo diferente

Antes de rotular					Após rotular				
0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	1	0	2	0
0	1	0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0	3	0
0	0	1	1	0	0	0	3	3	0
0	0	1	1	0	0	0	3	3	0
0	0	0	0	0	0	0	0	0	0

Vizinhos de um pixel  $p$



Algoritmo

\*\* ao implementar modifique o algoritmo de forma a substituir os 'if' que testam a vizinhança por um único 'if' dentro de um laço

```
// considerando que a borda da imagem são zeros
// im - imagem original
// im_rot - imagem rotulada - inicialmente zerada
```

```

label = 1;
lista_proximos = cria_lista();
Ponto p, p_atual;
for i = 1 ate nlinhas-1 {
    for j = 1 ate ncolunas-1 {
        // percorre toda a imagem em busca de um pixel foreground (valor 1)
        p.x = i;
        p.y = j;
        if (im(p.x,p.y)==1) and (im_rot(p.x,p.y)==0) {
            // atribui o label a posição (i,j)
            im_rot(p.x,p.y) = label;
            // inclui na lista de busca dos vizinhos
            lista_proximos.push_back(p);
        while !vazia(lista_proximos) {
            // busca o próximo ponto da lista
            p_atual = pop(lista_proximos);

            // buscando por pixels na vizinhança do ponto atual que são iguais a 1
            // ponto acima
            p.x = p_atual.x - 1;
            p.y = p_atual.y;
            // verifica if o ponto acima não é um e não foi rotulado
            if (im(p.x, p.y)==1) and (im_rot(p.x,p.y)==0){
                // atribui o label a posição atual
                im_rot(p.x,p.y) = label;
                // adiciona o ponto na lista para verificar vizinhos posteriormente
                push(lista_proximos,p);
            }
            // ponto abaixo
            p.x = p_atual.x + 1;
            p.y = p_atual.y;
            if (im(p.x, p.y)==1) and (im_rot(p.x,p.y)==0){
                // atribui o label a posição atual
                im_rot(p.x,p.y) = label;
                // busca o próximo ponto da lista
                push(lista_proximos,p);
            }
            // ponto à esquerda
            p.x = p_atual.x;
            p.y = p_atual.y - 1;
            if (im(p.x, p.y)==1) and (im_rot(p.x,p.y)==0){
                im_rot(p.x,p.y) = label;
                push(lista_proximos,p);
            }
            // ponto à direita
            p.x = p_atual.x;
            p.y = p_atual.y + 1;
            if (im(p.x, p.y)==1) and (im_rot(p.x,p.y)==0){
                im_rot(p.x,p.y) = label;
                push(lista_proximos,p);
            }
        } // enquanto
        label = label + 1;
    }
}

```

```
    } // if  
  }  
}
```

## Labirinto

O programa deverá receber uma imagem de zeros e uns (imagem binária) que representa um labirinto. O programa deverá descobrir sozinho qual é o caminho que deverá ser percorrido para descobrir a saída do labirinto

Assumir que todas as bordas são zeros, exceto dois pontos que representam a entrada e a saída do labirinto. A resposta deverá ser uma imagem igual à original, mas indicando com o valor 2 o caminho percorrido. Mostrar também as coordenadas (i,j) de cada ponto que pertence à esse caminho.

## Labirinto

0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	0	0
0	0	1	0	0	0	1	0	0
0	0	1	0	0	0	1	0	0
1	1	1	0	1	1	1	1	1
0	1	0	0	1	0	0	0	0
0	1	1	0	1	1	1	0	0
0	0	1	0	1	0	0	0	1
0	0	1	0	0	0	1	1	1
0	0	1	1	1	1	1	0	0
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0

Resposta

[illegible]