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

Informações adicionais

Exemplo de uma imagem real em tons de cinza

Cor branca: valor 255 

Cor preta: valor 0 

Valores próximos a 255 são claros. Valores próximos a zero são escuros. Valores intermediários são cinza (cinza escuro se mais próximo do zero; cinza claro se mais próximo do 1).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	67	123	255	255	255	255	255
2	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	120	52	90	255	255	255	255	255
3	255	255	255	255	255	255	255	94	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	58	71	83	255	255	255	255	255
4	255	255	255	255	255	255	255	51	120	255	255	255	255	255	255	255	255	255	255	255	255	255	255	61	84	79	76	255	255	255	255	255
5	255	255	255	255	255	255	255	55	57	161	255	255	255	255	255	255	255	255	255	255	255	98	60	87	98	80	71	255	255	255	255	255
6	255	255	255	255	255	255	255	63	109	74	100	255	255	255	255	255	255	255	255	130	104	60	85	102	88	83	81	65	255	255	255	255
7	255	255	255	255	255	255	255	88	139	141	137	96	73	133	255	255	177	114	79	88	110	100	79	73	87	80	67	255	255	255	255	255
8	255	255	129	99	125	255	224	144	202	216	162	144	139	187	235	232	213	158	127	103	91	83	79	75	89	78	75	255	255	255	255	255
9	255	62	13	35	19	57	134	255	223	236	132	70	147	218	169	222	74	96	137	92	89	85	81	77	92	76	83	255	255	255	255	255
10	134	10	216	242	117	47	19	209	103	168	105	51	155	210	116	255	125	6	127	94	90	86	82	78	94	73	94	255	255	255	255	255
11	104	37	251	255	122	86	41	58	151	10	11	64	140	197	43	45	30	15	133	94	90	86	82	78	98	68	118	255	255	255	255	255
12	209	21	159	175	81	59	51	11	133	133	111	129	132	150	153	55	40	103	109	93	89	85	81	79	100	56	255	255	255	255	255	255
13	255	27	69	66	61	49	37	13	76	113	110	110	111	126	137	137	135	110	95	91	87	84	80	87	89	60	255	255	255	255	255	255
14	255	94	26	54	42	31	24	11	70	102	104	105	105	104	103	101	98	95	80	48	82	81	78	98	68	95	255	255	255	255	255	255
15	255	255	88	15	26	23	15	6	91	97	99	100	100	99	98	96	93	57	10	6	32	79	89	91	54	127	255	255	255	255	255	255
16	255	255	255	123	18	1	7	66	118	111	99	94	93	90	82	66	45	22	32	82	74	75	93	60	80	120	133	255	255	255	255	255
17	255	255	255	255	255	209	116	57	63	91	114	113	101	93	88	86	83	82	82	78	78	115	93	65	99	121	128	209	255	255	255	255
18	255	255	255	255	255	255	133	118	85	58	56	80	102	108	111	106	101	105	103	94	78	56	68	106	109	185	135	255	255	255	255	255
19	255	255	255	255	255	255	209	130	118	107	84	61	52	51	57	65	71	59	52	52	55	75	94	88	114	185	233	186	118	120	255	255
20	255	255	255	255	255	255	255	255	135	125	115	107	99	89	78	70	65	72	81	92	98	105	114	123	135	240	255	255	99	114	83	255
21	255	255	255	255	255	255	255	255	255	255	209	131	125	120	117	115	114	114	116	120	124	132	255	255	255	255	255	255	110	92	79	56	240
22	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	108	72	43	98
23	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	144	75	47

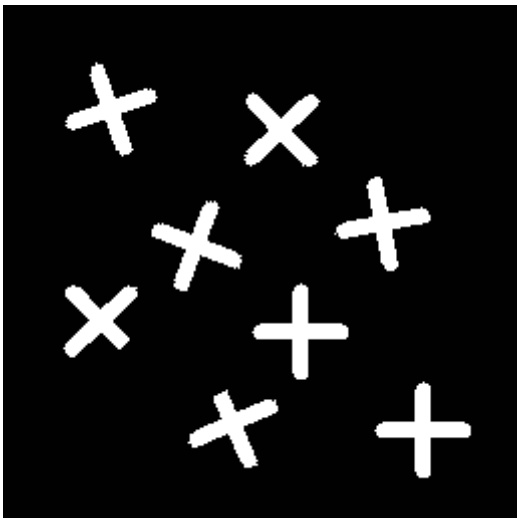
Segmentação da imagem. Segmentar a imagem usando o *threshold* é bem simples. Dado o valor de thr, os pixels (elementos da matriz) que são maiores que thr serão transformados em 1, e os menores em 0. Por exemplo, se consideramos a imagem anterior e um valor de limiarização thr = 200, a imagem resultante fica assim:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1
3	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1
4	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1
5	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1
6	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
7	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
8	1	1	0	0	0	1	1	0	1	1	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
9	1	0	0	0	0	0	0	1	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
10	0	0	1	1	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
11	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
12	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
13	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
14	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
15	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
16	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
17	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
18	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
19	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1
20	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1
21	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	1	1
22	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
23	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0

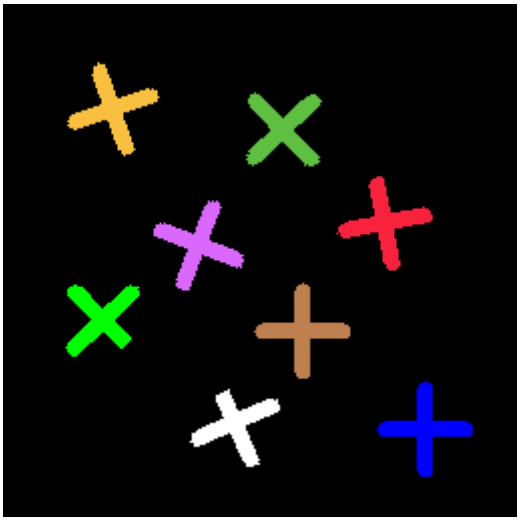
Algoritmo para detecção de componentes conexos

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 algortimo deverá percorrer a imagem e rotular cada objeto com um rótulo diferente

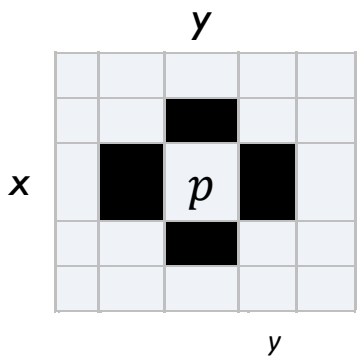
Antes de rotular

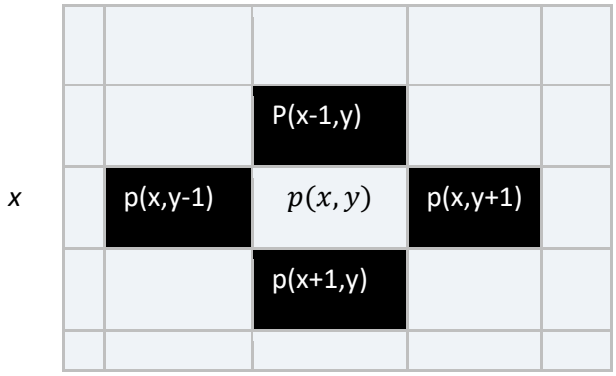
0	0	0	0	0
0	**1**	0	**1**	0
0	**1**	0	0	0
0	0	0	**1**	0
0	0	**1**	**1**	0
0	0	**1**	**1**	0
0	0	0	0	0

Após rotular

0	0	0	0	0
0	**1**	0	**2**	0
0	**1**	0	0	0
0	0	0	**3**	0
0	0	**3**	**3**	0
0	0	**3**	**3**	0
0	0	0	0	0

Vizinhos de um pixel p





Algoritmo

** ao implementar modifique o algoritmo de foma 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) {
       lista_proximos.push_back(p);
       while !vazia(lista_proximos) {
          // busca o próximo ponto da lista
          p_atual = pop(lista_proximos);
          // atribui o label a posição (i,j)
          im_rot(p_atual.x,p_atual.y) = label;

          // 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)
             // adiciona o ponto na lista para rotular posteriormente
             push(lista_proximos,p);
          fim
          // 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){
             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){
             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){
             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	0	1	1	1	1	1	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	1	0	0	0
1	1	1	0	1	1	1	1	1	0
0	1	0	0	1	0	0	0	0	0
0	1	1	0	1	1	1	0	0	0
0	0	1	0	1	0	0	0	1	0
0	0	1	0	0	0	1	1	1	0
0	0	1	1	1	1	1	0	0	0
0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	0	0	0

Resposta

0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	1	0	0	0
2	**2**	1	0	1	1	1	1	1	0
0	**2**	0	0	1	0	0	0	0	0
0	**2**	**2**	0	1	1	1	0	0	0
0	0	**2**	0	1	0	0	0	1	0
0	0	**2**	0	0	0	1	1	1	0
0	0	**2**	**2**	**2**	**2**	**2**	0	0	0
0	0	0	0	0	0	**2**	0	0	0
0	0	0	0	0	0	**2**	**2**	**2**	**2**
0	0	0	0	0	0	0	0	0	0