Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação
SCC0201 – Introdução à Ciência da Computação II

Trabalho 5 - Optical Character Recognition (OCR)

Professores: Dr. Rodrigo Fernandes de Mello (`mello@icmc.usp.br`)
Dr. Moacir Antonelli Ponti (`moacir@icmc.usp.br`)
Estagiários PAE: Felipe Simões Lage Gomes Duarte (`fgduarte@icmc.usp.br`)
Gabriel de Barros Paranhos da Costa (`gbpcosta@icmc.usp.br`)
Tiago Santana de Nazaré (`tiagosn@usp.br`)

Objetivo

Implementar um OCR utilizando morfologia matemática. Parte desse trabalho terá como base o seu código implementado nos dois últimos trabalhos, os quais não serão explicados nesse texto; você deverá consultar as especificações anteriores caso tenha alguma dúvida.

O que é OCR?

OCR (Optical Character Recognition) é o nome dado para a conversão de caracteres em uma imagem (e.g. digitalização de um livro ou documento) em caracteres de máquina (e.g. ASCII).

Morfologia Matemática

Morfologia matemática é uma área de processamento de imagens que trabalha com a geometria dos objetos. Nela aplica-se diversas transformações (ou operações) morfológicas que são feitas entre duas imagens. As imagens usadas nessas transformações são, na maior parte das vezes, binárias. No entanto, também é possível aplicar essas transformações em imagens em escala de cinza, por exemplo.

Para facilitar a visualização das imagens binárias, convencionamos que os *pixels* que têm valor 1 serão representados pela cor preta e os *pixels* que têm valor 0 são representados pela cor branca, como mostrado na Figura 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	0	0	0	0	0	0
0	0	0	**1**	0	0	0	0	0	0
0	0	**1**	**1**	**1**	0	**1**	**1**	**1**	0
0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0
0	0	**1**	**1**	**1**	0	**1**	**1**	**1**	0
0	0	0	**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	0	0

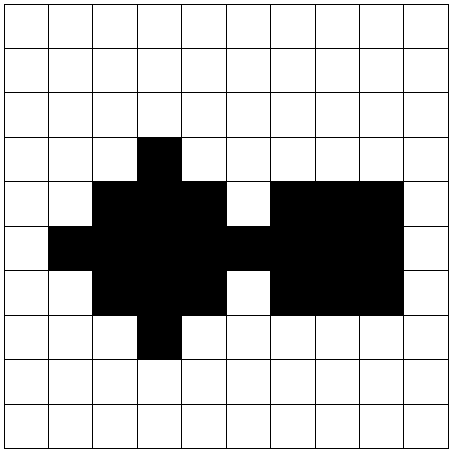
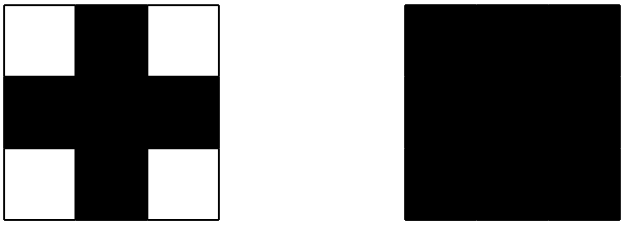


Figura 1: Exemplo da representação visual usada nas outras imagens deste texto. A primeira matriz mostra os valores binários dos *pixels* e a segunda a representação visual (como uma imagem binária) da primeira matriz. Nesses exemplos os *pixels* com valor 0 são representados em branco e os *pixels* com valor 1 são representados em preto.

Como mencionado anteriormente, uma operação morfológica é feita usando duas imagens. A primeira delas é uma imagem "qualquer" e a segunda é conhecida como elemento estruturante. O elemento estruturante é uma imagem (matriz), menor que a imagem a ser transformada, e que tem uma geometria previamente definida. Dois exemplos são: o elemento 3×3 do tipo disco e o elemento 3×3 do tipo quadrado, mostrados na Figura 2.



(a) Disco 3×3 (b) Quadrado 3×3

Figura 2: Exemplos de elementos estruturantes.

Dilatação

A operação de dilatação ($A \oplus B$) — que é exemplificada nas Figuras 3 e 4 — gera uma versão dilatada da imagem. Tal operação pode ser definida, para uma imagem A e um elemento estruturante B, da seguinte forma:

- Começa-se com a imagem resultante da dilatação S tendo todos os *pixels* iguais a 0;
- Desloca-se o elemento estruturante B colocando o seu centro em cada um dos pontos da imagem A;
  - Em cada ponto (i, j) no qual o centro de B coincidir com um *pixel* de valor 1 em A faz-se a união do elemento B (com centro em (i, j)) com a imagem resultante S. Isso significa que, considerando o centro do elemento B posicionado em (i, j), todos os *pixels* da imagem de saída S sobre os quais há um *pixel* de B com valor 1 passam a valer 1.

Uma outra maneira de definir dilatação é como o conjunto de todos os pontos z, para os quais o elemento estruturante B e a imagem A se sobreponham em pelo menos um pixel de valor 1. Sendo assim, a dilatação pode ser definida como:

$$A \oplus B = \left\{ z | \hat{B}_z \cap A \neq \emptyset \right\},$$

onde z representa as coordenadas dos *pixels* da imagem dilatada e $\hat{B}_z$ é o elemento estruturante refletido[1] e centrado no *pixel* z.

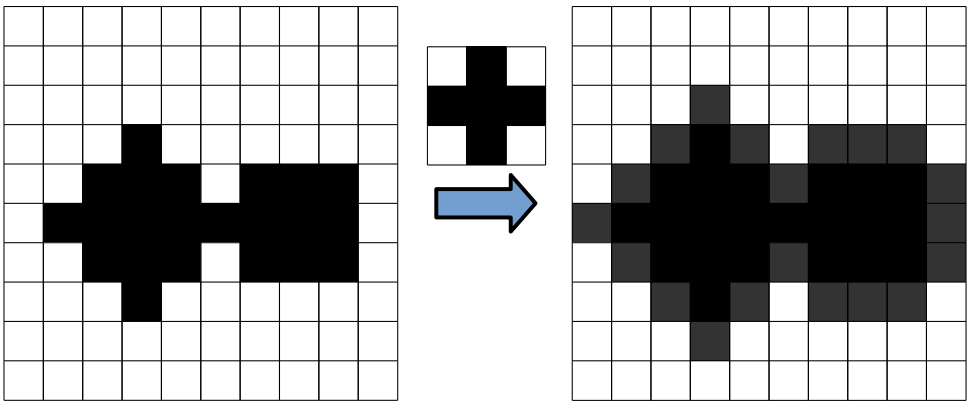


Figura 3: Exemplo de dilatação usando um elemento estruturante do tipo disco 3×3. Os *pixels* em cinza escuro, assim como os *pixels* em preto, têm valor 1. A cor cinza foi utilizada para destacar os *pixels* da imagem original que apresentam valores diferentes na imagem resultante.

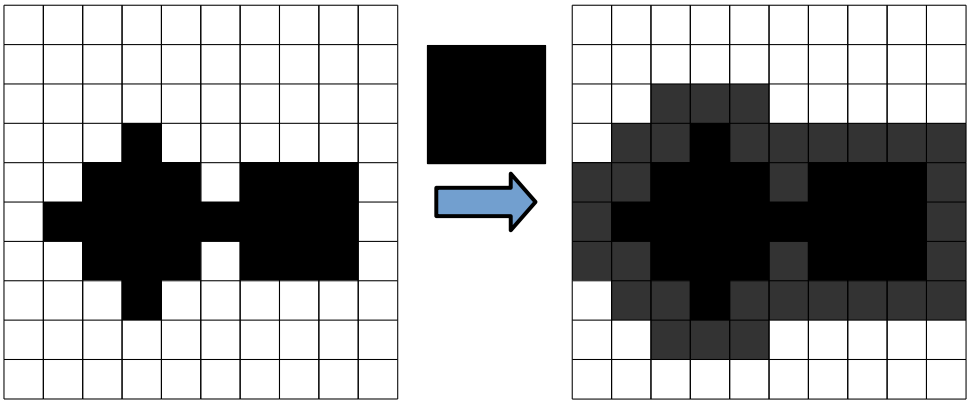


Figura 4: Exemplo de dilatação usando um elemento estruturante do tipo quadrado 3×3. Os *pixels* em cinza escuro, assim como os *pixels* em preto, têm valor 1. A cor cinza foi utilizada para destacar os *pixels* da imagem original que apresentam valores diferentes na imagem resultante.

Erosão

A operação de erosão ($A \ominus B$) — que é exemplificada nas Figuras 5 e 6 — tem como saída uma versão erodida da imagem original. Essa operação pode ser definida para uma imagem A e um elemento estruturante B da seguinte forma:

[1] Como em todos os casos abordados nesse trabalho os elementos estruturantes são simétricos e as operações são feitas deslocando-se o centro do elemento, a reflexão sempre será o próprio elemento. Portanto, não há necessidade de tratá-la.

- Começa-se com a imagem resultante da erosão S tendo todos os *pixels* iguais a 0;
- Desloca-se o centro do elemento estruturante B por toda a imagem A;
  - Somente nos pontos (i, j) nos quais **todos** os *pixels* de valor 1 do elemento estruturante B coincidem com *pixels* de valor 1 na imagem A a imagem de saída S terá *pixels* com valor 1.

Uma outra maneira de definir erosão é como o conjunto de todos os pontos z, para os quais o elemento estruturante B está contido em A (todos os *pixels* 1 de B sobrepõem *pixels* 1 em A). Sendo assim, a erosão pode ser definida como:

$$A \ominus B = \{z | B_z \subseteq A\},$$

onde z representa as coordenadas dos *pixels* da imagem erodida e B_z é o elemento estruturante centrado no *pixel* z.

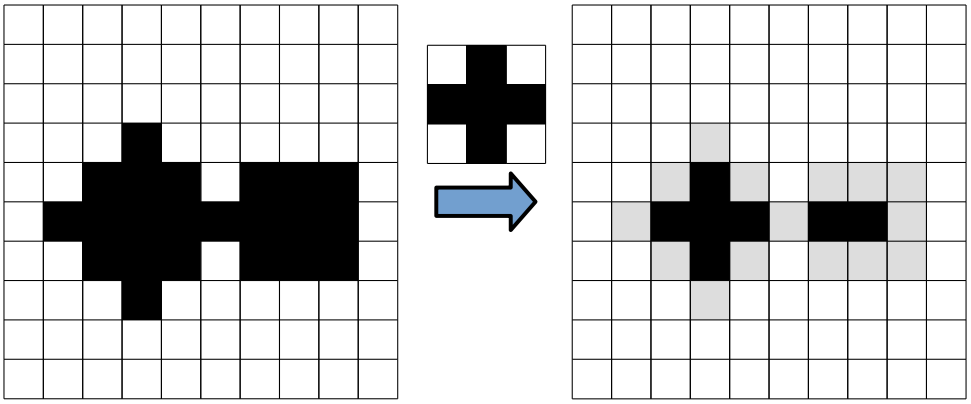


Figura 5: Exemplo de erosão usando um elemento estruturante do tipo diamante 3×3. Os *pixels* em cinza claro, assim como os *pixels* em branco, têm valor 0. A cor cinza foi utilizada para destacar os *pixels* "removidos" da imagem após a erosão.

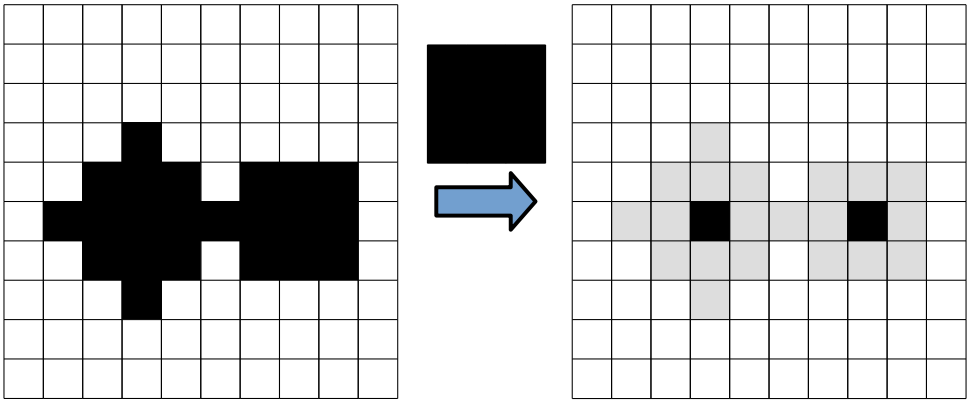


Figura 6: Exemplo de erosão usando um elemento estruturante do tipo disco 3×3. Os *pixels* em cinza claro, assim como os *pixels* em branco, têm valor 0. A cor cinza foi utilizada para destacar os *pixels* "removidos" da imagem após a erosão.

Abertura

A operação de abertura é uma combinação das operações de erosão e dilatação, na qual, dados uma imagem A e um elemento estruturante B, primeiro faz-se uma erosão e, em seguida, uma dilatação em A. Ambas as operações usam B como elemento estruturante. Sendo assim, a abertura é definida da seguinte forma:

$$A \circ B = (A \ominus B) \oplus B$$

Fechamento

A operação de fechamento também é uma combinação das operações de erosão e dilatação. No entanto, na abertura, dados uma imagem A e um elemento estruturante B, inicialmente faz-se uma dilatação e depois uma erosão de A. Assim como na abertura, ambas as operações usam B como elemento estruturante. Consequentemente, o fechamento é definido da seguinte forma:

$$A \bullet B = (A \oplus B) \ominus B$$

Requisitos

Nessa seção são apresentados alguns detalhes/requisitos da implementação do sistema. As operações que o sistema deve realizar, assim como os padrões de entrada e saída, serão apresentados na seção Operações.

1 – Armazenamento das Imagens Binárias na Base de Dados

As imagens binárias serão armazenadas na base de dados desenvolvida no Trabalho 3. Para isso, sua base de dados deverá suportar um novo tipo de dados (`byte[n]`). O tipo `byte[n]` consiste em um vetor de n posições, no qual cada posição tem 1 `byte` e cada um dos 8 `bits` desse `byte` representa um *pixel* da imagem binária. As imagens devem ser armazenadas usando o menor número de `bytes` possível. No entanto, é importante notar que, como a memória é alocada por bytes (e como 1 byte = 8 bits), uma imagem que tenha 9 *pixels* — e portanto precisa de 9 bits para ser armazenada — usará 2 bytes (16 bits) em disco.

Outro fator importante do armazenamento das imagens no disco é que o arquivo `.schema` sempre seguirá o seguinte padrão:

```
table <NOME_TABELA>
nrows int
ncols int
bits byte[n]
class char[m]
```

onde `nrows` é o números de linhas da imagem, `ncols` é o número de colunas da imagem, `bits` é o vetor de `bytes` com os *pixels* da imagem, `class` é a classe da imagem e `n` e `m` são inteiros e maiores do que zero. Sendo assim, no caso onde tem-se imagens 3×3, o arquivo poderia ser, por exemplo:

```
table ocr1
nrows int
ncols int
bits byte[2]
class char[2]
```

É importante lembrar que você deve incluir os atributos `id` e `dist`, assim como descrito no Trabalho 4.

2 – Processamento das Imagens Binárias em Memória

As imagens devem ser armazenadas na memória utilizando arranjos do tipo `unsigned char` e, consequentemente, não teremos na memória a mesma configuração por bits utilizada no disco.

As operações de erosão, dilatação, abertura e fechamento devem ser implementadas considerando que as imagens estão na memória. Para cada uma dessas operações serão passadas duas imagens binárias — uma imagem "qualquer" e uma imagem com o elemento estruturante — e seu programa deverá ser capaz de realizar a operação morfológica. Os resultados dessas operações morfológicas serão armazenados em memória (não devem ser incluídos na base de dados).

É importante notar que ao realizar as operações morfológicas há um problema de tratamento de bordas. Esse problema ocorre quando, por exemplo, o centro do elemento estruturante está posicionado sobre o *pixel* $(0, 0)$. Nesse caso, alguns *pixels* do elemento estruturante ficam fora da imagem sobre a qual a operação morfológica está sendo realizada, como mostra a Figura 7.

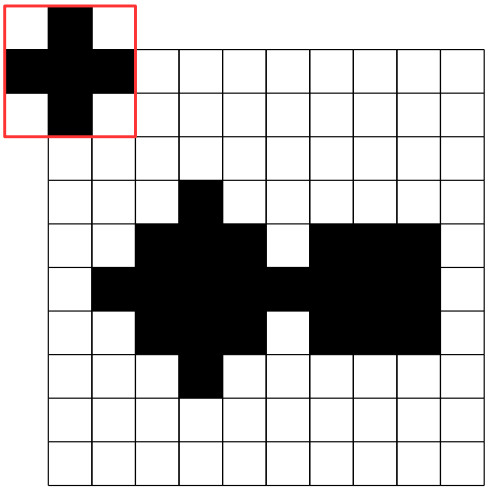


Figura 7: Exemplo do problema de processamento da borda. Quando o centro do elemento estruturante é colocado sobre o *pixel* $(0, 0)$ da imagem, alguns *pixels* do elemento estruturante não sobrepõem nenhum *pixel* da imagem (ficam fora da imagem).

Entre as várias maneiras de tratar esse problema a mais simples (e a que será adotada nesse trabalho) é simplesmente não processar os *pixels* das bordas. Dessa forma, nesse exemplo, o processamento só aconteceria nos *pixels* destacados na Figura 8. É bastante importante observar que o tamanho da borda que não será processada depende do tamanho do elemento estruturante e que seu programa terá que considerar isso.

3 – Classificação das Imagens (OCR)

Para detectar a qual caractere corresponde uma nova imagem, seu programa deverá executar o algoritmo KNN sobre a base de dados armazenada, assim como foi feito no Trabalho 4. A base de dados irá conter nos seus registros caracteres alfanuméricos que servirão como base para o reconhecimento.

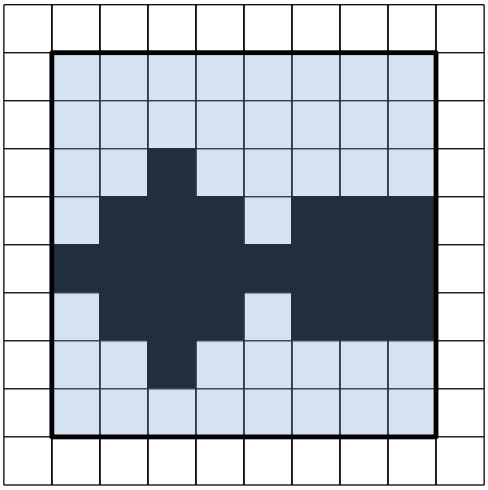


Figura 8: Somente os *pixels* dentro da região azul serão processados. É importante notar que a borda tem tamanho 1 porque o elemento estruturante usado no processamento é de tamanho 3×3. Logo, a borda depende do tamanho do elemento estruturante.

Tarefa (entrada e saída)

Nessa seção são apresentadas as operações que seu sistema deve realizar e como serão as entradas e saídas de cada operação. Assim como nos trabalhos 3 e 4, seu programa deverá tratar dados em disco. Para fazer isso, antes de qualquer operação seu programa irá ler o nome de um arquivo `.schema` em seguida receberá dados da mesma forma que no Trabalho 4. Também como no Trabalho 4 deverá ser criado o campo `dist` que não conta no arquivo `.schema`.

Importante: casos em que **não** será feito o uso da base de dados — a entrada será a palavra `none` no lugar de um arquivo `.schema`, e portanto não será necessário carregar nenhum dado e a próxima entrada na `stdin` será referente a uma operação.

Seu programa executará operações até receber comando `exit`.

Nas próximas seções são descritas as operações que seu sistema deve implementar. Os títulos das seções indicam se a operação fará uso de disco ou se só fará uso de memória.

1 – Distância Entre Duas Imagens (Memória)

Essa operação é chamada de `mem_dist` e será executada em **memória**. Nela serão passados as dimensões das imagens e os bits de cada imagem, como mostrado abaixo. Seu programa deve calcular a distância (número de bits diferentes) entre as duas imagens e imprimir na saída padrão a seguinte mensagem: `dist = <DISTANCIA>\n`. Essa distância é conhecida como Distância de Hamming para números binários (`https://pt.wikipedia.org/wiki/Dist%C3%A2ncia_de_Hamming`).

```
mem_dist
2 5
1010101010
1111111111
```

2 – Operações Morfológicas (Memória)

Essa operação é chamada de `mem_op` e também será feita completamente em **memória**. Nela serão passados as dimensões da imagem a ser processada, os bits da imagem, as dimensões do

elemento estruturante, os bits do elemento estruturante e a operação morfológica a ser executada (`erode`, `dilate`, `open` ou `close`). Abaixo tem-se um exemplo da entrada dessa operação:

```
mem_op
5 5
00000011100111001110000000
3 3
010111010
dilate
```

A saída para essa operação deve seguir o seguinte formato, no qual há um espaço entre cada *pixel* e um `\n` ao final de cada linha da imagem:

```
im:\n
<matriz de pixels da imagem>\n
el:\n
<matriz de pixeis do elemento estruturante>\n
out:\n
<matriz de pixeis do resultado do operação morfológica>\n
```

Para o exemplo de entrada dado nessa seção a saída seria:

```
im:
0 0 0 0 0
0 1 1 1 0
0 1 1 1 0
0 1 1 1 0
0 0 0 0 0
el:
0 1 0
1 1 1
0 1 0
out:
0 1 1 1 0
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
0 1 1 1 0
```

3 – Exibir o Schema da Base de Dados (Disco)

Essa operação é chamada pelo comando `dump_schema` e deve seguir o mesmo padrão dos dois trabalhos anteriores. É importante notar que o tipo `byte` será impresso de uma forma parecida com o tipo `char`. Sendo assim, um exemplo de saída dessa operação seria:

```
table tabela1(24 bytes)
id int(4 bytes)
nrows int(4 bytes)
ncols int(4 bytes)
bits byte[5](5 bytes)
class char[2](2 bytes)
dist double(8 bytes)
```

4 – Exibir os Registros da Base de Dados (Disco)

Essa operação é chamada de `dump_data` e é igual a operação `dump_data` do Trabalho 4, exceto pelo fato de que nesse trabalho você deve imprimir, também, dados do tipo `byte`. A impressão desse tipo de dados será feita imprimindo todos os bits da matriz em uma única linha de modo que primeiro seja impressa a primeira linha da imagem, depois a segunda linha da

imagem e assim por diante. Sendo assim, suponha que temos os bits da seguinte imagem em um campo chamado `bits` do tipo `byte[9]`:

```
0 1 0
1 1 1
0 1 0
```

A impressão ficaria da seguinte forma:

```
bits = 010111010
```

5 – Exibir os Vizinhos Mais Próximos (Disco)

Para essa operação — que será representada pelo comando `ocr_dump_nn` — serão fornecidos os seguintes parâmetros:

1. As dimensões da imagem;
2. Os bits da imagem;
3. As dimensões de um elemento estruturante;
4. Os bits do elemento estruturante;
5. O nome de uma operação morfológica (`erode`, `dilate`, `open` ou `close`) e
6. O número n de vizinhos.

Um exemplo de entrada seria:

```
mem_op
5 5
00000011100111001111000000
3 3
010111010
dilate
3
```

Com esses parâmetros, primeiramente, seu programa deve carregar a imagem e o elemento estruturante na memória. Depois disso seu sistema deve usar a imagem resultante para encontrar e exibir os n vizinhos mais próximos da mesma maneira como foi feito no Trabalho 4.

No entanto, diferentemente do Trabalho 4, a distância entre as imagens deve ser calculada considerando somente o campo do tipo `byte` e a medida de distância usada deve ser o número de `bits` diferentes entre as duas imagens.

Com relação à saída (impressão) dos resultados, primeiramente, deve-se imprimir a imagem de entrada, o elemento estruturante e o resultado da operação morfológica. Isso deverá ser feito seguindo o mesmo padrão da operação `mem_op`. Em seguida, deve-se imprimir os vizinhos mais próximos (do mais próximo para o mais distante) usando o mesmo padrão de impressão da operação `dump_data`.

6 – Classificar Imagem (Disco)

Essa operação será executada usando o comando `ocr_knn` e terá o mesmo tipo de entrada da operação anterior. A saída dessa operação deverá ser classe gerada pelo maior número de votos no KNN seguida de um `\n`. Deve-se usar os mesmos critérios de desempate do trabalho anterior.

IMPORTANTE

- Utilize alocação dinâmica (memória heap) para armazenar os dados;
- Não esqueça de liberar toda memória alocada antes de encerrar a execução do seu programa;
- Crie quantas funções achar necessário, a modularização do sistema será levado em consideração no processo de correção.
- **PRAZO FINAL: 19/10 - 23:59:59**