MAC0122 Princípios de Desenvolvimento de Algoritmos
Bacharelado em Estatística, Matemática e Matemática Aplicada
Prova Substitutiva – 1 de dezembro de 2016

Nome: _____

Assinatura: _____

Nº USP: _____ Prof: _____

Instruções:

1. Não destaque as folhas deste caderno. A prova pode ser feita a lápis.
2. A prova consta de 4 questões. Verifique antes de começar a prova se o seu caderno está completo.
3. As questões podem ser resolvidas em qualquer página. Ao escrever uma solução (ou parte dela) em página diferente do enunciado, escreva QUESTÃO X em letras ENORMES junto da solução.
4. As soluções devem ser em Python. Você pode usar apenas recursos de Python vistos em aula. Você pode definir funções auxiliares e usá-las à vontade. Cuidado com a legibilidade e, principalmente, com a TABULAÇÃO.
5. As soluções não precisam verificar consistência de dados.
6. Não é permitido o uso de folhas avulsas para rascunho, a consulta a livros, apontamentos, colegas ou equipamentos eletrônicos. Desligue o seu celular e qualquer equipamento que possa perturbar o andamento da prova.

DURAÇÃO DA PROVA: 2 horas

Questão	Valor	Nota
1	2,5	
2	2,5	
3	2,5	
4	2,5	
Total	10,0	

Questão 1 (vale 2,5 pontos)

Nessa questão você deverá escrever uma função `troque_cor()` para trocar a cor dos pixels de uma região de uma dada imagem.

A imagem é representada por um objeto `img` da classe `ndarray` ou simplesmente `array`. Cada pixel de `img` tem uma cor que é um valor inteiro entre `0` e `255`. Um pixel é **vizinho** de outro se estiver imediatamente acima, abaixo, à esquerda ou à direita do outro. Observe que nas bordas e cantos da imagem alguns vizinhos podem não existir.

A região que terá a sua cor trocada é dada através da posição `[px,py]` de um de seus pixels. Esse pixel é chamado de **semente**. Todos os pixels na região formada pela semente e pelos pixels:

- vizinhos da semente que tem a mesma cor que a semente;
- vizinhos dos vizinhos da semente ... que tem a mesma cor que a semente;
- vizinhos, dos vizinhos, dos vizinhos da semente ...
- ...

terão a sua cor trocada para uma dada `nova_cor`.

A fim de armazenar os pixels que terão a sua cor trocada para `nova_cor` a sua função `troque_cor()` **deverá utilizar** uma pilha. Inicialmente a pilha contém apenas a semente. Em cada iteração a função deve desempilhar e trocar a cor de um pixel e empilhar todos os vizinhos desse pixel que têm a cor original da semente.

Qual o consumo de tempo da sua função `troque_cor()`?

```
def troque_cor(img, px, py, nova_cor):
    ''' (int array, int, int, int) -> None

    Recebe um array img de inteiros representando uma imagem,
    uma posição [px, py] de uma semente e um inteiro nova_cor.
    A função troca para nova_cor a cor dos pixels da região "conexa" que
    contém a semente e é formada pelos pixels que têm a cor original da semente.

    A sua função deve usar obrigatoriamente uma pilha para amazenar as posições
    dos pixels que terão a sua cor trocada.

    Pré-condição: a função supõe que cor original da semente é diferente de
        nova_cor.
    '''
```

Questão 2 (vale 2,5 pontos)

Uma variação do algoritmo **Mergesort iterativo** consiste de duas fases:

(1) percorrer os elementos dados e determinar sequências (maximais) de elementos que já estejam em ordem crescente;

(2) efetuar a intercalação das sequências obtidas na fase (1), duas a duas, até que se obtenha apenas uma sequência.

Por exemplo, para a sequência:

```
           26    5    77    1    51    15    48    60    11    59
```

temos que

```
fase (1): [26]  [5    77]  [1    51]  [15    48    60]  [11    59]

fase (2): [5    26    77]  [1    15    48    51    60]  [11    59]

          [1    5    15    26    48    51    60    77]  [11    59]

          [1    5    11    15    26    48    51    59    60    77]
```

Suponha que uma sequência ordenada por esta variante do Mergesort tem `n` elementos.

Item 2(a)
Qual é o consumo de tempo dessa variante do Mergesort no **melhor caso**?
Apresente um exemplo em que o melhor caso ocorre.

Item 2(b)
Qual é o consumo de tempo dessa variante do Mergesort no **pior caso**?
Apresente um exemplo em que o pior caso ocorre.

Item 2(c)
Qual é o consumo de tempo do Mergesort tradicional no **pior caso** e no **melhor caso**?

Questão 3 (vale 2,5 pontos)

Item 3(a)

O que é um max-heap? A lista abaixo é um max-heap? Justifique.

1										11
15	16	32	27	25	30	13	11	39	7	6

Transforme esse vetor em um max-heap da mesma maneira que é feito na fase de pré-processamento do Heapsort. Exiba o conteúdo do vetor após cada alteração feita pela função `peneira()`.

Item 3(b)
Escreva uma função `ff()` que **recebe** um vetor `v` e um índice `n` tais que `v[1:n]` é um max-heap e **transforma** `v[1:n+1]` em max-heap. O consumo de tempo de sua função **deve** ser O(lg `n`).

Questão 4 (vale 2,5 pontos)

Considere a seguinte versão simplificada de um jogo de bingo que chamaremos de `BIN`.

Em um jogo de `BIN` cada jogador possui uma cartela. Alguns exemplos de cartelas de BIN são mostrados abaixo:

```
Cartela 1:       Cartela 2:       Cartela 3:
 B 3 7 1          B 3 6 2          B 5 0 1
 I 3 8 0          I 4 5 8          I 5 0 1
 N 9 7 5          N 2 6 1          N 7 0 5
```

Cada linha da cartela, identificadas pelas letras `'B'`, `'I'`, e `'N'`, possui 3 números entre 0 e 9, sem repetição. Note, entretanto, que um número **pode se repetir** em linhas distintas da cartela.

Em um jogo de BIN, um mestre de cerimônias anuncia uma combinação após a outra sem repetições, como, por exemplo, `'B2'`, `'I7'`, `'N4'`, ... Portanto, cada **combinação** é composta por uma letra que identifica a linha (`'B'`, `'I'`, ou `'N'`) e um número (entre 0 e 9).

Cada jogador deve **marcar** na sua cartela as combinações anunciadas que estejam presentes na cartela. Uma **cartela é vencedora** se todas as suas 9 combinações estiverem marcadas.

Nesta questão você deverá escrever um programa (função `main()`) que simula um jogo de BIN. Para resolver essa questão você deve utilizar a seguinte classe `Cartela`, **sem implementá-la**.

```
class Cartela:
    def __init__(self):
        '''(Cartela) -> None

        Cria uma cartela de BIN.
        '''

    def __str__(self):
        '''(Cartela) -> str

        Retorna um string que mostra o conteudo da cartela como:
            B 3 7 1
            I 3 8 0
            N 9 7 5
        '''

    def marca(self, lance):
        '''(Cartela, str) -> bool

        Recebe uma cartela self, um string lance representando e uma
        combinação e marca essa combinação caso esteja presente na cartela.
        O método retornar True se todas as combinações da cartela estiverem
        marcadas, e False em caso contrário.
        '''
```

Utilizando a classe `Cartela` escreva um programa em Python (função `main()`) que simule um jogo de `BIN` com apenas 2 jogadores.

Cada jogador deve possuir uma cartela. O programa deve ler as combinações do teclado (na forma de strings como `'B4'`, `'I7'`, `'N3'` etc) até que a cartela de um dos (ou ambos) jogadores tenha todas as suas combinações marcadas. Quando uma cartela tiver todas as suas combinações marcadas, o programa deve imprimir "BIN!!".

Além disso, quando o jogo terminar, o programa deve imprimir a sequência completa de combinações, e imprimir a(s) cartela(s) vencedora(s) e a cartela perdedora (se houver).

Por exemplo, considere as seguintes cartelas

```
Cartela 1:      Cartela 2:      Cartela 3:
 B 0 1 4         B 8 0 5         B 8 1 3
 I 4 6 1         I 2 8 5         I 1 6 4
 N 8 0 6         N 2 3 5         N 3 8 2
```

e a seguinte sequência de lances:
`'B4', 'B0', 'I6', 'I1', 'N6', 'N0', 'N2', 'B8', 'B3', 'N3', 'N8', 'B1', 'I4'`.

A coluna da esquerda mostra como o seu programa deve se comportar quando os jogadores possuem as cartelas 1 e 2, respectivamente, e a coluna da direita mostra como o seu programa deve se comportar quando os jogadores possuem as cartela 1 e 3, para a mesma entrada.

```
BEM VINDO AO BIN!!
Digite um lance: B4
Digite um lance: B0
Digite um lance: I6
Digite um lance: I1
Digite um lance: N6
Digite um lance: N0
Digite um lance: N2
Digite um lance: B8
Digite um lance: B3
Digite um lance: N3
Digite um lance: N8
Digite um lance: B1
Digite um lance: I4

BIN!!

cartela vencedora:
B 1 4 0
I 4 1 6
N 6 8 0

cartela perdedora:
B 8 5 0
I 5 2 8
N 3 5 2

lances: B4 B0 I6 I1 N6 N0 N2 B8 B3 N3 N8 B1 I4
```

```
BEM VINDO AO BIN!!
Digite um lance: B4
Digite um lance: B0
Digite um lance: I6
Digite um lance: I1
Digite um lance: N6
Digite um lance: N0
Digite um lance: N2
Digite um lance: B8
Digite um lance: B3
Digite um lance: N3
Digite um lance: N8
Digite um lance: B1
Digite um lance: I4

BIN!!

BIN!!

cartela vencedora:
B 0 1 4
I 1 6 4
N 6 8 0

cartela vencedora:
B 8 3 1
I 4 6 1
N 8 3 2

lances: B4 B0 I6 I1 N6 N0 N2 B8 B3 N3 N8 B1 I4
```