MC458 - Projeto e Análise de Algoritmos I
Prova Individual - 29/10/2018

1. Comece pelas questões que você tem mais certeza de saber fazer.
2. Não perca tempo com detalhes menores. Foque no que é relevante para resolver a questão e passe logo para a próxima.
3. Se sobrar tempo, você pode acrescentar mais detalhes.

Questão 1 (valor 2,5) Considere o problema de ordenar n inteiros, sendo todos eles pertencentes ao intervalo $0..k$. Sabemos que o CountingSort resolve este problema em tempo $O(n+k)$ de forma estável (isto é, elementos com chave igual mantêm a ordem que tinham na entrada), mas não é local (isto é, ele usa espaço adicional não-constante, que são as matrizes auxiliares B e C.)

Proponha um algoritmo que ordene n inteiros em 0..1 em tempo $O(n)$ que seja local. Seu algoritmo não precisa ser estável, mas precisa ser capaz de carregar junto com as chaves as informações auxiliares dos registros.

Questão 2 (Valor 2,5) Uma versão do método de ordenação QuickSort utiliza uma rotina Particiona(A, p, r, x), com $p \leq r$, que rearranja os elementos de um trecho $A[p..r]$ do vetor A em relação a um pivot x. Além de rearranjar os elementos, a rotina Particiona retorna um índice q entre p e r tal que os elementos em $A[p..q]$ são menores ou iguais a x e os elementos $A[q+1..r]$ são maiores que x.

Em algumas aplicações é importante que haja aproximadamente a mesma quantidade de elementos iguais a x antes e depois de q. Para atender a esta necessidade, escreva um procedimento Particiona(A, p, r, x) que rode em tempo $O(r-p+1)$ e retorne um índice q entre p e r tal que:

- os elementos em $A[p..q]$ são menores ou iguais a x;
- os elementos em $A[q+1..r]$ são maiores ou iguais a x;
- a diferença entre o número de elementos iguais a x em $A[p..q]$ e em $A[q+1..r]$ é no máximo 1.

Seu algoritmo não precisa ser local, e não deve fazer nada se $p > r$.

Questão 3 (Valor 2,5) Considere max-heaps quaternários, isto é, com até 4 filhos por nó, ao invés de 2. A propriedade de max-heap continua valendo, com cada pai tendo chave maior ou igual às chaves dos filhos, mas agora os filhos de $A[1]$ serão $A[2]$, $A[3]$, $A[4]$ e $A[5]$; os filhos de $A[2]$ serão $A[6]$, $A[7]$, $A[8]$ e $A[9]$; e assim por diante.

Sua tarefa nesta questão será programar a operação HEAP-INCREASE-KEY(A, i, x), que incrementa de $x \geq 0$ a chave na posição i do heap A e rearranja o heap em tempo $O(\lg n)$ para acomodar o novo valor.

Para ajudá-lo nesta tarefa, pode usar as seguintes rotinas para o cálculo dos pais e filhos de um índice no heap:

- LEFT$(i) = 4i - 2$
- MIDLEFT$(i) = 4i - 1$
- MIDRIGHT$(i) = 4i$
- MIDRIGHT$(i) = 4i + 1$
- PARENT$(i) = \lfloor (i + 2)/4 \rfloor$

Se $x < 0$, seu algoritmo não precisa fazer nada, ou pode levantar uma exceção, se você preferir.

Questão 4 (Valor 2,5) Escreva um algoritmo que rode em tempo $O(n)$ para resolver o seguinte problema: dado um vetor $A[1..n]$ com n números, determinar as três maiores diferenças da forma $A[i] - A[j]$, onde i e j são índices distintos entre 1 e n.

Você pode usar qualquer algoritmo visto em classe sem precisar produzir o código deste algoritmo, mas especifique-o bem e defina claramente seus parâmetros, se houver.

Boa sorte!

Soluções

Questão 1 (valor 2,5)

```
procedure ZeroOneSort(A)
    n ← length[A]
    i ← 1
    j ← n
    while i < j do
        if A[i] = 0 then
            i ← i + 1
        else if A[j] = 1 then
            j ← j − 1
        else                                   ▷ here A[i] = 1 and A[j] = 0
            A[i] ↔ A[j]
            i ← i + 1
            j ← j − 1
```

Questão 2 (valor 2,5)

Uma maneira é usar vetores auxiliares:

```
function Particiona(A, p, r, x)
    B ← [ ]
    C ← [ ]
    left ← true
    for i ← p to r do
        if A[i] < x then
            B.append(A[i])
        else if A[i] > x then
            C.append(A[i])
        else                                   ▷ here A[i] = x
            if left then
                B.append(A[i])
            else
                C.append(A[i])
            left ← not left
    A ← B + C                                  ▷ B concatenated to C
    return length[B]
```

Outra maneira é aproveitar o PARTICIONA tradicional e mudar um pouco para acomodar a necessidade de balancear os x's:

```
function PARTICIONA(A, p, r, x)
    i ← p − 1
    j ← p − 1
    for k ← p to r do
        if A[k] ≤ x then
            j ← j + 1
            A[j] ↔ A[k]
        if A[j] < x then
            i ← i + 1
            A[i] ↔ A[j]
    return ⌊(i + j)/2⌋
```

Questão 3 (valor 2,5)

```
procedure HEAP-INCREASE-KEY(A, i, x)
    if x > 0 then
        A[i] ← A[i] + x
        while i > 0 and A[PARENT(i)] < A[i] do
            A[PARENT(i)] ↔ A[i]
            i ← PARENT[i]
```

Questão 4 (valor 2,5)

O método consiste em encontrar os três maiores e os três menores, e então calcular 5 diferenças entre os grandes e os pequenos que certamente contêm as 3 maiores diferenças. Depois, é só ordenas as 5 candidatas e pegar as 3 maiores.

Vamos começar com subrotrinas para encontrar o maior e o menor.

```
function MAX(A, k)
    i ← 1
    for j ← 2 to k do
        if A[j] > A[i] then
            i ← j
    return i
```

function MIN(A, k)
    $i \leftarrow 1$
    for $j \leftarrow 2$ **to** k **do**
        if $A[j] < A[i]$ **then**
            $i \leftarrow j$
    return i

A seguir, rotinas para determinar os três maiores e os três menores.

function THREELARGEST(A)
    $i \leftarrow \text{MAX}(A, n)$
    $max \leftarrow A[i]$
    $A[i] \longleftrightarrow A[n]$
    $i \leftarrow \text{MAX}(A, n-1)$
    $max2 \leftarrow A[i]$
    $A[i] \longleftrightarrow A[n-1]$
    $i \leftarrow \text{MAX}(A, n-2)$
    $max3 \leftarrow A[i]$
    return $max, max2, max3$

function THREESMALLEST(A)
    $i \leftarrow \text{MIN}(A, n)$
    $min \leftarrow A[i]$
    $A[i] \longleftrightarrow A[n]$
    $i \leftarrow \text{MIN}(A, n-1)$
    $min2 \leftarrow A[i]$
    $A[i] \longleftrightarrow A[n-1]$
    $i \leftarrow \text{MIN}(A, n-2)$
    $min3 \leftarrow A[i]$
    return $min, min2, min3$

Por fim, o programa principal, que retorna as três maiores diferenças.

function FINDLARGESTDIFFERENCES(A)
    $n \leftarrow \text{length}[A]$
    if $n < 3$ **then**
        return "not enough elements"
    else
        max, $max2$, $max3 \leftarrow$ THREELARGEST(A)
        min, $min2$, $min3 \leftarrow$ THREESMALLEST(A)
        $B[1] \leftarrow max - min$

$B[2] \leftarrow max - min2$
$B[3] \leftarrow max - min3$
$B[4] \leftarrow max2 - min$
$B[5] \leftarrow max3 - min$
SORT(B) in descending order
return $B[1]$, $B[2]$, $B[3]$