Estrutura de Dados - 1º. período de 2017

## Primeira Avaliação a Distância

1. Para cada item abaixo, responda "certo" ou "errado", justificando (meio ponto cada):

   a. Se a complexidade de pior caso de um algoritmo $A$ para um problema $P$ for $O(f)$, então o número de passos que um algoritmo ótimo para $P$ efetua, em qualquer caso, é igual a $O(f)$.

   *Resposta:* Certo. Seja $l$ um limite inferior para $P$. Por definição, um limite inferior para um problema $P$ é uma função $l$, tal que a complexidade de pior caso de qualquer algoritmo que resolva $P$ é $\Omega(l)$. Além disso, um algoritmo $A$ ótimo para $P$ é aquele cuja complexidade de pior caso é $O(l)$. Dessa forma, qualquer que seja a entrada para $A$, o mesmo executará em um tempo $g$, tal que $g = O(l) = O(f)$. Logo $g = O(f)$.

   b. Se a complexidade de melhor caso de um algoritmo for $\Theta(f)$, então o número de passos que o algoritmo efetua, qualquer que seja a entrada, é $O(f)$.

   *Resposta:* Errado. Um exemplo pode ser dado pelo problema de ordenação de uma lista com $n$ elementos, cuja complexidade de pior caso é $O(n \log n)$, enquanto a complexidade de melhor caso é igual a $\Theta(n)$.

2. (1,0) Considere uma sequência de elementos $f_1, f_2, \ldots, f_n$, definida do seguinte modo: $f_1 = 0$, $f_2 = -1$, $f_3 = 1$, $f_j = f_{j-1} - f_{j-2} + f_{j-3}$ para $j > 3$. Elaborar algoritmos, um recursivo e outro não recursivo, para determinar o elemento $f_n$ da sequência. Determinar a complexidade dos algoritmos.

**Algoritmo 1:** Algoritmo recursivo com chamada $f_n$ e que utiliza um vetor auxiliar $V$.

**Entrada:** Inteiro positivo $n$.
**Saída:** Valor de $f_n$.

**se** $n > 3$ **então**
    $f_n \leftarrow f_{n-1} - V[n-2] + V[n-3]$;
**senão**
    **se** $n = 1$ **então**
        $f_n \leftarrow 0$;
    **senão**
        **se** $n = 2$ **então**
            $f_n \leftarrow -1$;
        **senão**
            $f_n \leftarrow 1$;
$V[n] \leftarrow f_n$;
**retorna** $f_n$;

*Resposta:* Não é difícil verificar que ambos os algoritmos executam em tempo linear, ou seja, ambos possuem complexidade $O(n)$. Note também que o Algoritmo 1 possui apenas uma chamada recursiva.

**Algoritmo 2:** Algoritmo não recursivo.

**Entrada:** Inteiro positivo $n$.
**Saída:** Valor de $f_n$.

$f_1 \leftarrow 0$;
$f_2 \leftarrow$ -1;
$f_3 \leftarrow 1$;
**para** $j \leftarrow 4, \ldots, n$ **faça**
    $f_j \leftarrow f_{j-1} - f_{j-2} + f_{j-3}$;
    $f_{j-3} \leftarrow f_{j-2}$;
    $f_{j-2} \leftarrow f_{j-1}$;
    $f_{j-1} \leftarrow f_j$;
**retorna** $f_n$;

3. (2,0) Escreva um algoritmo que inverte uma lista simplesmente encadeada com nó cabeça. Exemplo: se a lista contém os elementos $1, 3, 5, 9, 2$, nesta ordem, então a lista resultante contém os elementos $2, 9, 5, 3, 1$. Atenção: não é permito o uso de estruturas auxiliares para realizar a inversão, como um vetor auxiliar ou uma pilha. Responda: sem o uso de estruturas auxiliares, é possível realizar a inversão em tempo inferior a $O(n^2)$?

**Algoritmo 3:** Inversão de uma lista simplesmente encadeada.

**Entrada:** Nó cabeça PTlista da lista simplesmente encadeada $L$ com $n > 1$ elementos.

1 pt_inicio ← PTlista↑.prox;
2 pt_iprox ← pt_inicio↑.prox;
3 pt_fim ← pt_inicio;
4 pt_fprox ← pt_fim↑.prox;
5 **enquanto** *pt_fprox* $\neq \lambda$ **faça**
6     pt_fim ← pt_fprox;
7     pt_fprox ← pt_fim↑.prox;
8 **enquanto** *PTlista↑.prox* $\neq$ *pt_fim* **faça**
9     PTlista↑.prox ← pt_iprox;
10     pt_inicio↑.prox ← pt_fprox;
11     pt_fprox ← pt_inicio;
12     pt_fim↑.prox ← pt_fprox;
13     pt_inicio ← PTlista↑.prox;
14     pt_iprox ← pt_inicio↑.prox;

*Resposta:* O Algoritmo 3 inverte uma lista simplesmente encadeada com o uso de quatro ponteiros adicionais: dois no início e dois no fim da lista. As linhas 1–4 apenas iniciam os ponteiros do início e fim da lista. As linhas 5–7 movem os ponteiros do fim da lista para suas posições corretas, onde pt_fim marca o último nó da lista não nulo. As linhas 8–14 iterativamente invertem a lista, onde é feita a remoção do primeiro nó da lista para o final da mesma. Note que o ponteiro pt_fim não é alterado, o que significa que ele não marca mais o final da lista, e sim o final da lista anterior a esta etapa, onde o primeiro nó é inserido.

Pode-se notar que o algoritmo executa em tempo $O(n)$. Porém, isso se deve ao fato do uso dos ponteiros adicionais que marcam as partes descritas acima da lista. Isto evita que se tenha que percorrer a lista para acessar tais posições. Sem o uso destes ponteiros ou uma estrutura auxiliar não poderíamos acessar tais posições em tempo constante, mas sim em tempo $O(n)$. Isso resulta em um tempo $O(n^2)$.

4. (2,0) Faça uma tabela que liste as complexidades de melhor e pior caso dos algoritmos de busca, inserção e remoção, em listas sequenciais e simplesmente encadeadas, ordenadas e não ordenadas. Note que a tabela contém 12 entradas! Observação: as complexidades de inserção e remoção devem desconsiderar o tempo prévio para fazer a busca do elemento.

   *Resposta:*Na tabela abaixo, cada célula contém as complexidades de melhor e pior caso de cada operação, respectivamente.

| | Sequencial | | Simplesmente Encadeada | |
|---|---|---|---|---|
| | ordenada | não ordenada | ordenada | não ordenada |
| Busca | $O(1)$ e $O(\log n)$ | $O(1)$ e $O(n)$ | $O(1)$ e $O(n)$ | $O(1)$ e $O(n)$ |
| Inserção | $O(1)$ e $O(n)$ | $O(1)$ e $O(1)$ | $O(1)$ e $O(n)$ | $O(1)$ e $O(1)$ |
| Remoção | $O(1)$ e $O(n)$ | $O(1)$ e $O(n)$ | $O(1)$ e $O(n)$ | $O(1)$ e $O(1)$ |

5. (2,0) Compare as complexidades dos métodos de ordenação vistos nas aulas quando o vetor de entrada já está ordenado crescentemente, supondo elementos distintos entre si. Repita o exercício quando o vetor de entrada vem dado em ordem decrescente.

   *Resposta:* Utilizando o método de ordenação por seleção em um vetor ordenado tanto crescentemente como em ordem decrescente, são feitas $n$ trocas, onde as trocas são todas dos elementos com sua própria posição, uma vez que a cada passo o primeiro elemento da lista ainda não ordenada é o menor de todos. Porém, o número de comparações continua da ordem de $\Theta(n^2)$, dado que o algoritmo percorre todo o vetor a cada iteração para encontrar o menor elemento. Já o método da bolha não efetua trocas no caso do vetor ordenado crescentemente, enquanto efetua $\Theta(n^2)$ trocas no caso do vetor ordenado em ordem decrescente. Portanto a complexidade no primeiro caso é $\Theta(n)$, enquanto que no segundo é $\Theta(n^2)$.

6. (2,0) Considere uma lista sequencial $L$ com $n$ elementos distintos, e um vetor $V$ com $m$ elementos quaisquer. Deseja-se realizar a seguinte tarefa: para cada elemento $x$ de $V$, deve-se verificar se $x$ está em $L$. Se $x$ não estiver em $L$, deve-se inseri-lo em $L$; caso contrário, nada a fazer. Em qualquer caso, continua-se a tarefa para o próximo elemento de $V$. Elabore um algoritmo eficiente para resolver este problema. Procure elaborar um algoritmo com a melhor complexidade possível.

   *Resposta:*O Algoritmo 4 executa a operação descrita. Primeiramente é feita a ordenação tanto da lista $L$ quanto do vetor $V$. Além disso é criada uma lista encadeada vazia $E$, que irá armazenar a lista resultante $L'$ após a adição de novos elementos de $V$. Como $L$ e $V$ estão ordenados, podemos efetuar a busca dos elementos de $V$ em $L$ através de busca binária, o que leva $O(\log n)$ cada uma. Caso a busca retorne falso, o elemento é adicionado ao final da lista $E$. Como $V$ pode conter elementos repetidos, evitamos e repetição da busca de vários do mesmo elemento em $L$ com as linhas 20–29. A complexidade deste algoritmo depende dos valores $m$ e $n$, uma vez que a ordenação de $L$ e $V$ custa $O(n \log n)$ e $O(m \log m)$, respectivamente, enquanto o número de buscas binárias é $O(m \log n)$. Assim, se $m < n$, então a complexidade é dada pela ordenação de $L$, caso contrário é dada pela ordenação de $V$.

**Algoritmo 4:** Adição de elementos de um vetor $V$ à uma lista sequencial $L$.

**Entrada:** Nó cabeça PTlista da lista simplesmente encadeada $L$ com $n > 1$ elementos.

```
L ← Ordenação(L);
V ← Ordenação(V);
E ← CriarListaEncadeada();
i ← 1;
j ← 1;
enquanto i < m faça
    se j ≤ n então
        sair ← falso;
        enquanto sair = falso faça
            se L[j] < V[i] então
                InserirNoFinal(L[j], E);
                j ← j + 1;
                se j > n então
                    sair ← verdadeiro;
            senão
                sair ← verdadeiro;
    se BuscaBinaria(V[i], L) = 0 então
        InserirNoFinal(V[i], E);
    sair ← falso;
    enquanto sair = falso faça
        se i < m então
            se V[i] = V[i + 1] então
                i ← i + 1;
            senão
                sair ← verdadeiro;
        senão
            se BuscaBinaria(V[i], L) = 0 então
                InserirNoFinal(V[i], E);
            sair ← verdadeiro;
Copia(L,E);
```