Curso de Tecnologia em Sistemas de Computação
Disciplina: Estrutura de Dados e Algoritmos
Gabarito - AD1 - 2º Semestre de 2015

1. (1,5) Desenvolva um algoritmo *não recursivo* que, dado um vetor V contendo n elementos (onde cada elemento é um número inteiro), elimina todas as ocorrências de elementos repetidos em V. Exemplo: se $n = 9$ e $V = [1, 2, 5, 2, 1, 9, 1, 8, 9]$, o algoritmo deve modificar n e V de modo que tenhamos $n = 5$ e $V = [1, 2, 5, 9, 8]$ após a execução do algoritmo. Determine a complexidade do seu algoritmo.

   Resposta: O Algoritmo 1 mostra uma implementação não recursiva para eliminar repetições em um vetor V com n elementos dado como entrada. O algoritmo percorre o vetor da posição inicial até n através da variável i. Este percurso é marcado pelas linhas 4 a 11 e representado pelo laço **enquanto**. Para cada iteração i, é feita a verificação da ocorrência do elemento $V[i]$ nas n' primeiras posições do vetor. Esta verificação é feita nas linhas 6 e 7 através da variável j. Em cada iteração na linha 6, é feita a comparação dos valores $V[j]$ e $V[i]$. Caso esses elementos sejam iguais, então significa que $V[i]$ é um elemento repetido de V dentre seus n' primeiros. Dessa forma $V[i]$ deve ser desconsiderado, onde isto é feito na linha 11 com a passagem para a próxima iteração. Caso contrário, significa que todos os primeiros n' elementos de V são diferentes de $V[i]$. Assim, devemos adicioná-lo ao vetor resultante. Isto é feito nas linhas 8 a 10.

   Como o algoritmo percorre o vetor das posições 1 até n' a cada iteração i, podemos ver que o Algoritmo 1 possui complexidade $O(n^2)$.

Algoritmo 1: Algoritmo não *recursivo* que elimina elementos repetidos de um um vetor.

Entrada: Vetor V composto por n inteiros.
Saída: Vetor V modificado contendo seus n' primeiros elementos distintos.

```
1  se n > 1 então
2      n' ← 1;
3      i ← 2;
4      enquanto i ≤ n faça
5          j ← 1;
6          enquanto V[j] ≠ V[i]  e  j ≤ n' faça
7              j ← j + 1;
8          se j > n' então
9              n' ← n' + 1;
10             V[n'] ← V[i];
11         i ← i + 1;
```

Resultado: V;

2. (1,5) Repita o exercício anterior, mas desta vez desenvolvendo um algoritmo *recursivo* para o mesmo problema.

   Resposta: O Algoritmo 2 mostra uma versão recursiva do Algoritmo 1, onde o laço do tipo **enquanto** na

linha 4 é substituído por uma chamada recursiva considerando o $(i+1)$-ésimo elemento de V.

Algoritmo 2: Alg(V, n', i, n)

Entrada: Vetor V composto por n inteiros. O algoritmo inicia com $n' = 1$ e $i = 2$.
Saída: Vetor V modificado contendo seus n' primeiros elementos distintos.

```
1 se i ≤ n então
2     j ← 1;
3     enquanto V[j] ≠ V[i]  e  j ≤ n' faça
4         j ← j + 1;
5     se j > n' então
6         n' ← n' + 1;
7         V[n'] ← V[i];
8     Alg(V, n', i + 1, n);
```

Resultado: V;

3. (1,0) Escrever as seguintes funções em notação O:
$n^2 + n^2 \log n$; $3n - 5$; $\log^2(n/2)$; $2^n + n!$; $(n+1)^n$; 100; $\sqrt{n} + \log n$.

Resposta: $n^2 + n^2 \log n = O(n^2 \log n)$; $3n - 5 = O(n)$; $\log^2(n/2) = O(\log^2(n))$; $2^n + n! = O(n!)$; $(n+1)^n = O(n^n)$; $100 = O(1)$; $\sqrt{n} + \log n = O(\sqrt{n})$.

4. (1,5) Sejam L_1 e L_2 duas listas ordenadas, simplesmente encadeadas com nó-cabeça. Cada lista L_i não contém elementos repetidos, mas pode ocorrer de um mesmo elemento estar em ambas as listas. Apresentar um algoritmo que construa uma lista ordenada contendo os elementos de ambas as listas, eliminando repetições. Por exemplo, se a lista L_1 contém os elementos $23, 34, 56, 78, 89$ e a lista L_2 contém os elementos $12, 23, 29, 45, 56, 67, 77, 89$, então a lista resultante é $12, 23, 29, 34, 45, 56, 67, 77, 78, 89$.

Resposta: O Algoritmo 3 descreve uma solução para o problema em questão. Os ponteiros auxiliares Pont_1, Pont_2 e Pont_3 servem para percorrermos as listas L_1, L_2 e L_3, respectivamente. Usamos uma função chamada Novo_Nó_Cabeça() para indicar a criação de uma lista vazia cujo nó-cabeça é retornado. A função Novo_Nó() serve para indicar a criação de um nó vazio.

O algoritmo executa o percurso das duas listas buscando o menor elemento a cada passo. Como ambas L_1 e L_2 estão ordenadas, o menor elemento não percorrido em cada uma é sempre o próximo a ser visitado. No momento em que alguma das listas se torna vazia, condição verificada pelo comando **enquanto** na linha 5, o algoritmo efetua a cópia do restante da outra lista, o que segue nas linhas 21 a 26. A linha 27 finaliza a nova lista ao fazer Pont_3↑.prox apontar para λ.

5. (1,5) Considere cadeias de caracteres onde o primeiro caractere (o mais à esquerda) é um dígito $d \in \{0, 1, 2, \ldots, 9\}$, e os caracteres seguintes são "I" ou "R", onde "I" significa "inserção" e "R" significa "remoção". Uma cadeia deste tipo é *admissível* se representa uma sequência possível de operações sobre uma pilha que contém inicialmente d elementos, e é *inadmissível* caso contrário. Exemplos: a cadeia 3IIRRRRRR é inadmissível, pois existindo inicialmente 3 elementos na pilha e realizando a seguir duas inserções, é impossível realizar seis remoções na sequência; já a cadeia 2RRIIIRRIR é admissível, e após a realização de todas as operações, a pilha conterá 1 elemento. Desenvolva um algoritmo que leia uma cadeia de caracteres do tipo descrito acima, e decida se a mesma é admissível ou inadmissível; no caso de ser admissível, o algoritmo deve imprimir o número de elementos existentes na pilha após a realização de todas as operações.

Resposta: O Algoritmo 4 representa uma solução para o problema de decisão sobre uma cadeia ser ou não admissível.

Algoritmo 3: Lista_Ord_SEncad_Sem_Repet(PT_1, PT_2)

Entrada: Nós cabeça PT_1 e PT_2 das listas ordenadas e simplesmente encadeadas L_1 e L_2, respectivamente.

Saída: Nó-cabeça PT_3 da nova lista ordenada e simplesmente encadeada L_3 formada pela união dos elementos de L_1 e L_2.

```
1  Pont_1 ← PT_1 ↑.prox;
2  Pont_2 ← PT_2 ↑.prox;
3  PT_3 ← Novo_Nó_Cabeça();
4  Pont_3 ← PT_3;
5  enquanto Pont_1 ≠ λ e Pont_2 ≠ λ faça
6  |  Novo ← Novo_Nó();
7  |  Pont_3↑.prox ← Novo;
8  |  Pont_3 ← Pont_3↑.prox;
9  |  se Pont_1↑.info < Pont_2↑.info então
10 |  |  Pont_3↑.info ← Pont_1↑.info;
11 |  |  Pont_1 ← Pont_1↑.prox;
12 |  senão
13 |  |  Pont_3↑.info ← Pont_2↑.info;
14 |  |  se Pont_1↑.info = Pont_2↑.info então
15 |  |  |  Pont_1 ← Pont_1↑.prox;
16 |  |  Pont_2 ← Pont_2↑.prox;
17 se Pont_1 = λ então
18 |  PT ← Pont_2;
19 senão
20 |  PT ← Pont_1;
21 enquanto PT ≠ λ faça
22 |  Novo ← Novo_Nó();
23 |  Pont_3↑.prox ← Novo;
24 |  Pont_3 ← Pont_3↑.prox;
25 |  Pont_3↑.info ← PT↑.info;
26 |  PT ← PT↑.prox;
27 Pont_3↑.prox ← λ;
```

Resultado: PT_3;

Algoritmo 4: Pilha_Admiss(V, n)

Entrada: Vetor V de n elementos, cuja primeira posição contém um dígito $d \in \{0, 1, 2, \ldots, 9\}$ e as demais posições contém ou um caractere "R" ou um caractere "I".

Saída: *Sim* se o vetor representa uma cadeia admissível e *Não*, caso contrário.

```
1  d ← V[1];
2  se 0 ≤ d ≤ 9 e n ≥ 2 então
3      i ← 1;
4      repita
5          i ← i + 1;
6          se V[i] = "I" então
7              d ← d + 1;
8          senão
9              d ← d − 1;
10     até d < 0 ou i = n;
11 se d < 0 então
       Saída: Não;
12 senão
13     Imprime(d);
       Saída: Sim;
```

6. (1,0) Considere um vetor ordenado L contendo todos os múltiplos de três entre 1 e 100. Determine o número exato de comparações que a busca binária realiza ao buscar no vetor L o elemento x, nos seguintes casos: (a) $x = 29$; (b) $x = 99$; (c) $x = 50$. Responda também: qual é o pior caso da busca binária (em número de comparações) neste exemplo?

Resposta: Podemos ver que o vetor L possui 33 posições, já que para cada posição i temos o elemento $3 \times i$ ocupando a mesma. As Tabelas abaixo mostram as comparações feitas pela busca binária considerando cada valor dado de x. A Tabela 1 mostra m total de 5 comparações, enquanto as Tabelas 2 e 3 mostram um total de 6 comparações. Como o pior caso do algoritmo ocorre quando a busca segue até que reste apenas um único elemento a ser comparado, o número máximo de passos é dado por $\lfloor \log n \rfloor + 1$. Como $n = 33$, temos um total de 6 comparações no pior caso.

Elementos possíveis	Elemento comparado	Elementos descartados
1 até 33	$x = 3 \times 17$ (não)	17-33
1 até 16	$x = 3 \times 8$ (não)	1-8
9 até 16	$x = 3 \times 12$ (não)	12-16
9 até 11	$x = 3 \times 10$ (não)	10-11
9	$x = 3 \times 9$ (não)	9

Tabela 1: Execução de busca binária considerando $x = 29$.

7. (1,0) Dê exemplo de um vetor V com 6 elementos que leva o algoritmo de ordenação pelo método da bolha a realizar o maior número possível de *trocas entre elementos* quando a entrada é V. Desenhe todas as trocas de elementos efetuadas pelo algoritmo, passo a passo.

Resposta: Podemos verificar que o pior caso do algoritmo de ordenação pelo método da bolha ocorre quando o vetor está ordenado em ordem decrescente. A Tabela 4 mostra a execução deste algoritmo em relação ao vetor

Elementos possíveis	Elemento comparado	Elementos descartados
1 até 33	$x = 3 \times 17$ (não)	1-17
18 até 33	$x = 3 \times 25$ (não)	18-25
26 até 33	$x = 3 \times 28$ (não)	26-28
29 até 33	$x = 3 \times 31$ (não)	29-31
32 até 33	$x = 3 \times 32$ (não)	32
33	$x = 3 \times 33$ (sim)	

Tabela 2: Execução de busca binária considerando $x = 99$.

Elementos possíveis	Elemento comparado	Elementos descartados
1 até 33	$x = 3 \times 17$ (não)	17-33
1 até 16	$x = 3 \times 8$ (não)	1-8
9 até 16	$x = 3 \times 12$ (não)	9-12
13 até 16	$x = 3 \times 14$ (não)	13-14
15 até 16	$x = 3 \times 15$ (não)	15
16	$x = 3 \times 16$ (não)	16

Tabela 3: Execução de busca binária considerando $x = 50$.

$V = [6, 5, 4, 3, 2, 1]$, onde podemos ver um total de 15 trocas.

8. (1,0) Repita o exercício anterior, mas agora considerando o método de ordenação por seleção. Responda: o número de trocas (no pior caso) realizadas pelo método de ordenação por seleção é maior ou menor que o número de trocas (no pior caso) realizadas pelo *Bubblesort*?

   Resposta: Podemos verificar que o número de trocas do algoritmo de ordenação por seleção é menor que o do *Bubblesort*. A Tabela 5 mostra a execução deste algoritmo em relação ao vetor $V = [6, 5, 4, 3, 2, 1]$, onde podemos ver um total de apenas 6 trocas.

6	5	4	3	2	1	Troca
6*	5	4	3	2	1	
6	5*	4	3	2	1	x
5*	6	4	3	2	1	
5	6	4*	3	2	1	x
5	4*	6	3	2	1	x
4*	5	6	3	2	1	
4	5	6	3*	2	1	x
4	5	3*	6	2	1	x
4	3*	5	6	2	1	x
3*	4	5	6	2	1	
3	4	5	6	2*	1	x
3	4	5	2*	6	1	x
3	4	2*	5	6	1	x
3	2*	4	5	6	1	x
2*	3	4	5	6	1	
2	3	4	5	6	1*	x
2	3	4	5	1*	6	x
2	3	4	1*	5	6	x
2	3	1*	4	5	6	x
2	1*	3	4	5	6	x
1*	2	3	4	5	6	
1	2	3	4	5	6	

Tabela 4: Execução do algoritmo de ordenação pelo método da bolha.

6	5	4	3	2	1	Troca
6	5	4	3	2	1*	x
1*	5	4	3	2	6	
1*	5	4	3	2*	6	x
1*	2*	4	3	5	6	
1*	2*	4	3*	5	6	x
1*	2*	3*	4	5	6	x
1*	2*	3*	4*	5	6	x
1*	2*	3*	4*	5*	6	x
1*	2*	3*	4*	5*	6*	

Tabela 5: Execução do algoritmo de ordenação por seleção.