Curso de Tecnologia em Sistemas de Computação
Disciplina: Estrutura de Dados e Algoritmos
AP1 - Segundo Semestre de 2015

Nome -
Assinatura -

Observações:

1. Prova sem consulta e sem uso de máquina de calcular.
2. Use caneta para preencher o seu nome e assinar nas folhas de questões e nas folhas de respostas.
3. Você pode usar lápis para responder as questões.
4. Ao final da prova devolva as folhas de questões e as de respostas.
5. Todas as respostas devem ser transcritas nas folhas de respostas. As respostas nas folhas de questões não serão corrigidas.

---

1. Leia atentamente o texto a seguir, e depois resolva as questões abaixo.

   Sabemos que um problema computacional pode admitir vários algoritmos para resolvê-lo, que podem ter complexidades de pior caso diferentes. Dentre esses algoritmos, aqueles que possuem as menores complexidades de pior caso podem ou não ser ótimos, pois o que caracteriza um algoritmo ótimo não é o fato de ele ser "melhor" do que os outros, mas é algo relacionado à complexidade intrínseca do problema que ele resolve, isto é, ao limite inferior do problema.

   (a) (1,0) Dê a definição formal de limite inferior de um problema computacional.

   Resposta: Seja $A = \{A_1, A_2, \dots\}$ o conjunto de todos os algoritmos que resolvem um certo problema P. Seja também f_{A_i} a função que representa a complexidade do algoritmo A_i, para todo $i > 0$. Um limite inferior para P é uma função g tal que $f_{A_i} = \Omega(g)$, para todo $A_i \in A$. Ou seja, qualquer algoritmo que resolve P executa pelo menos $\Omega(g)$ passos.

   (b) (1,0) Dê a definição formal de algoritmo ótimo.

   Resposta: Seja g um limite inferior para um problema P. Um algoritmo ótimo A que resolve P é tal que sua complexidade é dada por $f = O(g)$. Dessa forma, o algoritmo A possui complexidade de pior caso $\Omega(g)$ e $O(g)$. Em outras palavras, um algoritmo é ótimo se sua complexidade de pior caso é dada pelo limite inferior para o problema.

2. Considere um volumoso cadastro de CPF's, no qual as seguintes operações são executadas:

   1. Busca de um CPF no cadastro.
   2. Inserção de um novo CPF no cadastro.
   3. Remoção de um CPF do cadastro.

   Suponha ainda que, para cada CPF, existe um contador que é incrementado de uma unidade a cada vez que este é buscado no cadastro. Isto forma uma estatística dos acessos.

   Temos a seguir várias estruturas de dados que poderiam ser utilizadas para implementar o cadastro de CPF's:
   a. Lista linear não ordenada.

Oper.\Estrut.	a	b	c	d	e	f
1		✓	✓			
2	✓		✓	✓		✓
3					✓	✓

Tabela 1: Representação das operações executadas por estruturas.

b. Lista linear ordenada por CPF.
c. Lista linear ordenada (decrescentemente) por contador.
d. Lista simplesmente encadeada não ordenada.
e. Lista simplesmente encadeada ordenada pelo CPF.
f. Lista simplesmente encadeada ordenada (decrescentemente) pelo contador.

Resolva as questões a seguir, justificando:

A Tabela 1 ilustra as operações com suas respectivas estruturas.

(a) (1,0) Quais as melhores estruturas em relação à operação 1?

Resposta: Como a busca em listas não ordenadas possui complexidade $O(n)$, enquanto a busca feita em listas lineares ordenadas pode ser feita em tempo $O(\log n)$, temos que as estruturas b e c são as mais indicadas. A estrutura c é mais adequada que as estruturas de listas encadeadas, já que a ordem dos elementos mais frequentemente acessados representa uma tendência das próximas buscas.

(b) (1,0) Quais as melhores estruturas em relação à operação 2?

Resposta: Como efetuar inclusões em estruturas que não necessitam de ordenação pode ser feito em $O(1)$, como no caso das estruturas a (incluindo o novo elemento na última posição livre do vetor) e d (incluindo o novo elemento no final da lista). Além disso, as estruturas c e f também possuem complexidade $O(1)$, pois podemos inserir o novo elemento no final da lista com contador igual a 1. Porém, caso não se saiba que o elemento a ser inserido já consta na estrutura, a inserção em todas elas possui complexidade $O(n)$, pois é necessária a busca pelo elemento antes da inserção. Além disso, como para manter a ordenação em um vetor após uma inclusão é necessário mover todos os elementos maiores que o novo elemento uma posição a frente, essa operação também necessita de $O(n)$ passos no pior caso.

(c) (1,0) Quais as melhores estruturas em relação à operação 3?

Resposta: Analogamente ao item (a), a busca pelo elemento desejado é feita com $O(n)$ passos, exceto no caso da lista ordenada. Porém, a remoção em qualquer caso é feita em $O(n)$, exceto para listas encadeadas, cuja complexidade de remoção é igual a $O(1)$. Assim, todas as estruturas não encadeadas podem exigir o percurso de toda a lista no pior caso, além de ao menos $O(\log n)$ passos adicionais para a busca do elemento a ser removido, o que leva a escolha das listas encadeadas.

3. (1,0) Considere uma fila F contendo as posições de 1 a 5. A variável f marca a posição de início da fila ("frente"), e a variável r marca a posição de fim da fila ("retaguarda"). No início, a fila F encontra-se vazia, e as variáveis f e r valem zero.

   Usamos a notação R para denotar a operação de remoção de um elemento da fila F, e a notação $I(X)$ para denotar a operação de inserção de um elemento X na fila F.

   Considere a seguinte sequência de operações em F:

$$I(A), I(B), R, I(C), R, R, I(D), I(E), I(F), R, R, I(G)$$

   Desenhe como fica a fila F após a sequência de operações acima, e forneça os valores finais das variáveis f e r. Use um traço (–) para denotar as posições vazias. Como um exemplo de configuração, poderíamos ter: $F = (\ - \ - \ C \ D \ - \ )$, com $f = 3$ e $r = 4$.

   Resposta: A Tabela 2 representa a sequencia de operações e estado da fila, além dos valores de f e r.

4. É dado um vetor V com $n > 1$ elementos distintos.

   (a) (2,0) Escreva um algoritmo que monta um outro vetor M, contendo n elementos, com a seguinte propriedade: para cada índice i entre 1 e n, o elemento $M[i]$ armazena quantos elementos de V são menores do que $V[i]$. Como exemplo, considere o vetor $V = (\ 2\ 19\ 26\ 3\ 4\ 1\ 5\ )$.

Oper.	Pos. 1	Pos. 2	Pos. 3	Pos. 4	Pos. 5	f	r
	(-	-	-	-	-)	0	0
I(A)	(A	-	-	-	-)	1	1
I(B)	(A	B	-	-	-)	1	2
R	(-	B	-	-	-)	2	2
I(C)	(-	B	C	-	-)	2	3
R	(-	-	C	-	-)	3	3
R	(-	-	-	-	-)	0	0
I(D)	(D	-	-	-	-)	1	1
I(E)	(D	E	-	-	-)	1	2
I(F)	(D	E	F	-	-)	1	3
R	(-	E	F	-	-)	2	3
R	(-	-	F	-	-)	3	3
I(G)	(-	-	F	G	-)	3	4

Tabela 2: Representação do estado da fila.

Temos $n = 7$. Para este vetor V, temos que $M = (\ 1\ 5\ 6\ 2\ 3\ 0\ 4\ )$.

Algoritmo 1: Algoritmo que retorna um novo vetor M, onde $M[i]$ representa o número de valores de V menores que $V[i]$.

Entrada: Vetor V composto por n valores distintos.
Saída: Vetor M.

para $i \leftarrow 1, \ldots, n-1$ **faça**
    para $j \leftarrow i+1, \ldots, n$ **faça**
        se $V[j] < V[i]$ **então**
            $M[j] \leftarrow M[j] + 1$;
        senão
            $M[i] \leftarrow M[i] + 1$;
retorna M;

(b) (2,0) Suponha que o vetor M já esteja calculado. Explique em palavras como ordenar os elementos de V utilizando as informações contidas no vetor M.

Resposta: A ideia para ordenar V corretamente é mover o elemento pertencente à posição 1 para a posição $M[1]+1$, já que pela definição de M existem exatamente $M[1]$ elementos menores que $V[1]$. Porém, para que não haja uma sobrescrita do elemento $V[M[1]+1]$, devemos salvar o mesmo em uma variável auxiliar, digamos Aux, antes de efetuar a escrita. Com isso Aux será o próximo elemento a ser colocado em sua posição correta, que é dada tomando-se a posição $M[i]+1$, onde i é a posição original de Aux em V. Dessa forma, é preciso salvar o valor contido na posição $M[M[i]+1]$ e efetuar a troca. O algoritmo segue dessa forma até que todo elemento $V[j]$ esteja na posição $M[j]+1$, tomando V e M iniciais. O Algoritmo 2 representa essa ideia.

Algoritmo 2: Algoritmo que retorna um novo vetor M, onde $M[i]$ representa o número de valores de V menores que $V[i]$.

Entrada: Vetor V composto por n valores distintos e vetor M, onde $M[i]$ representa o número de valores de V menores que $V[i]$.
Saída: Vetor V ordenado.

count $\leftarrow$ 1;
$Aux_{ant} \leftarrow V[1]$;
$Aux_{prox} \leftarrow V[M[1]+1]$;
$i \leftarrow 1$;
enquanto count $< n$ **faça**
    $i \leftarrow M[i]+1$;
    $V[i] \leftarrow Aux_{ant}$;
    $Aux_{ant} \leftarrow Aux_{prox}$;
    $Aux_{prox} \leftarrow V[M[i]+1]$;
    count $\leftarrow$ count+1;
retorna V;
