Estrutura de Dados - 1o. período de 2010

Gabarito da Primeira Avaliação à Distância

1. (1,0) O tamanho da entrada de um problema computacional está em função das variáveis n e m. Coloque as seguintes funções em ordem crescente de complexidade, sabendo que $m = \Theta(n)$: $O(m^2 + n)$, $O(m \log n)$, $O(mn^2)$, $O(\log m)$, $O(m^2 + \log n)$.

   Resposta: $O(\log m)$, $O(m \log n)$, $O(m^2 + \log n)$, $O(m^2 + n)$, $O(mn^2)$.

2. (1,5) As afirmações abaixo estão corretas. Justifique-as.

   a. Se um limite inferior para um problema P é n^2, então nenhum algoritmo ótimo para P pode ter complexidade de pior caso $O(n \log n)$.

   Resposta: Se n^2 é um limite inferior de P, então, pela definição de limite inferior, qualquer algoritmo para P executa, no pior caso, $\Omega(n^2)$ passos (inclusive um algoritmo ótimo). Como $n \log n$ é assintoticamente menor que n^2, então nenhum algoritmo para P (ótimo ou não) pode ter complexidade de pior caso $O(n \log n)$.

   b. Se a entrada de um problema P tem tamanho $O(m + n)$, então qualquer algoritmo para P tem complexidade $\Omega(m + n)$, usando qualquer definição (melhor caso, pior caso ou caso médio).

   Resposta: Considerando um problema P, será sempre necessário, em qualquer caso, ler a entrada do problema. Logo, a complexidade de qualquer algoritmo para P é limitada inferiormente por $m + n$, em qualquer caso.

   c. Se dois algoritmos A_1 e A_2 para um problema P têm complexidades de pior caso $O(n^2)$ e $O(n^3)$, respectivamente, então A_2 não é ótimo.

   Resposta: Um algoritmo ótimo tem a menor complexidade de pior caso possível. Como A_2 tem complexidade de pior caso maior que A_1, então A_2 não pode ser ótimo.

3. (1,5) Faça um algoritmo que receba como entrada uma lista L desordenada com 52 elementos, onde cada elemento $L[i]$ corresponde a uma carta de baralho e contém dois campos: $L[i].valor$ (que pode valer $2, 3, \cdots, 10, J, Q, K, A$), e $L[i].naipe$ (que pode valer *ouros*, *copas*, *paus* ou *espadas*). O algoritmo deve ordenar as cartas de acordo com as seguintes regras: os valores crescem na ordem $2, 3, \cdots, 10, J, Q, K, A$; os naipes crescem na ordem *ouros*, *copas*, *paus*, *espadas*; o campo *naipe* tem preponderância sobre o campo *valor*, isto é, um 10 de copas vale menos que um 3 de paus.

   Resposta: Sejam os valores das cartas armazenados como $2 \cdots 10$,$J = 11$,$Q = 12$,$K = 13$, $A = 14$.

```
para i := 1 até n − 1 faça
    menor := i
    naipe := L[i].naipe
    valor := L[i].valor
    para j := i + 1 até n faça
        se naipe = "ouros" então
            se (L[j].naipe = "ouros") e (L[j].valor < valor) então
                menor := j
        senão
            se naipe = "copas" então
                se (L[j].naipe = "ouros") então
                    menor := j
                senão
                    se (L[j].naipe = "copas") e (L[j].valor < valor) então
                        menor := j
            senão
                se naipe = "paus" então
                    se (L[j].naipe = "ouros") ou (L[j].naipe = "copas") então
                        menor := j
                    senão
                        se (L[j].naipe = "paus") e (L[j].valor < valor) então
                            menor := j
                senão
                    se (L[j].naipe = "ouros") ou (L[j].naipe = "copas") ou
                                    (L[j].naipe = "paus") então
                        menor := j
                    senão
                        se (L[j].valor < valor) então
                            menor := j
    se menor ≠ i então
        L[i].valor := L[j].valor
        L[i].naipe := L[j].naipe
        L[j].valor := valor
        L[j].naipe := naipe
```

4. (1,0) Determinar a expressão de complexidade média de uma busca NÃO ORDENADA de 10 chaves, em que a probabilidade de busca da chave i é 10% maior que a da chave $i+1$, $i = 1, \cdots, 9$. Supor, ainda, que a probabilidade de a chave procurada se encontrar na lista é igual a 100%.

   Resposta:

   Seja p a probabilidade de busca da chave 10. Temos então que $\left(\frac{11}{10}\right)^9 p + \left(\frac{11}{10}\right)^8 p + \left(\frac{11}{10}\right)^7 p + \cdots + \left(\frac{11}{10}\right)^2 p + \left(\frac{11}{10}\right) p + p = 1$. Logo,

$$\begin{aligned} p \sum_{i=1}^{10} \left(\frac{11}{10}\right)^{i-1} &= 1 \\ p \left(\frac{\left(\frac{11}{10}\right)^{10} - 1}{\frac{11}{10} - 1} \right) &= 1 \\ p &\approx 0,063 \end{aligned}$$

   A expressão da complexidade média neste caso é dada por:

$$\begin{aligned} C.M. &= \sum_{i=1}^{10} (p_i \,.\, t_i) \\ &= p \left[\left(\frac{11}{10}\right)^9 .1 + \left(\frac{11}{10}\right)^8 .2 + \left(\frac{11}{10}\right)^7 .3 + \cdots + \left(\frac{11}{10}\right) .9 + 10 \right] \\ &\approx 4,7 \end{aligned}$$

5. (1,5) Sejam L_1 e L_2 duas listas ordenadas, simplesmente encadeadas com nó-cabeça. Apresentar um algoritmo que construa uma lista ordenada contendo os elementos que pertencem apenas a uma das listas, mas não a ambas. (Isto é, elementos da interseção das duas listas devem ser descartados.) Suponha que os elementos em cada lista são todos distintos. Se L_1 contém n elementos e L_2 contém m elementos, o seu algoritmo deve ter complexidade $O(n+m)$.

   Resposta:

   $pont1 := ptlista1 \uparrow .prox$ % ponteiro para a lista L_1
   $pont2 := ptlista2 \uparrow .prox$ % ponteiro para a lista L_2
   $ptaux := ptnovo$ % a lista resultante iniciará em $ptnovo$

   enquanto $pont1 \neq \lambda$ e $pont2 \neq \lambda$ faça
   se $pont1 \uparrow .info < pont2 \uparrow .info$ então
   $ocupar(pt)$
   $pt \uparrow .info := pont1 \uparrow .info$
   $pt \uparrow .prox := \lambda$
   $ptaux \uparrow .prox := pt$
   $ptaux := pt$ % ptaux aponta para o último nó
   $pont1 := pont1 \uparrow .prox$
   senão
   se $pont1 \uparrow .info > pont2 \uparrow .info$ então
   $ocupar(pt)$
   $pt \uparrow .info := pont2 \uparrow .info$
   $pt \uparrow .prox := \lambda$
   $ptaux \uparrow .prox := pt$
   $ptaux := pt$
   $pont2 := pont2 \uparrow .prox$
   senão
   $pont1 := pont1 \uparrow .prox$
   $pont2 := pont2 \uparrow .prox$

   enquanto $pont1 \neq \lambda$ faça
   $ocupar(pt)$
   $pt \uparrow .info := pont1 \uparrow .info$
   $pt \uparrow .prox := \lambda$
   $ptaux \uparrow .prox := pt$
   $ptaux := pt$
   $pont1 := pont1 \uparrow .prox$

   enquanto $pont2 \neq \lambda$ faça
   $ocupar(pt)$
   $pt \uparrow .info := pont2 \uparrow .info$
   $pt \uparrow .prox := \lambda$
   $ptaux \uparrow .prox := pt$
   $ptaux := pt$
   $pont2 := pont2 \uparrow .prox$

6. (1,0) Escreva um algoritmo RECURSIVO baseado no método de divisão e conquista para encontrar o maior elemento de uma lista com n elementos, baseado no seguinte princípio: divide-se a lista ao meio e encontra-se recursivamente o maior elemento de cada uma das metades; a seguir, combina-se as duas soluções parciais na solução final. É possível implementar este algoritmo de modo a realizar no máximo $n-1$ comparações entre elementos, tal como no método sequencial?

   Resposta: Seja V o vetor com n elementos, indexado de 1 a n. O algoritmo abaixo realiza no máximo $n-1$ comparações entre elementos, pois a cada comparação, o elemento menor é descartado, não sendo mais comparado com nenhum outro.

```
função maior(i, j)
    se i = j então
        retornar V[i];
    senão
        se j = i + 1 então
            se V[i] > V[j] então
                retornar V[i];
            senão
                retornar V[j];
        senão
            meio = (i + j)/2;
            m1 = maior(i, meio);
            m2 = maior(meio + 1, j);
            retornar maior(m1, m2);
```

   Chamada externa: $maior(1, n)$

7. (1,5) 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.
   b. Lista linear ordenada por CPF.
   c. Lista linear ordenada (decrescentemente) por contador.
   d. Lista simplesmente encadeada não ordenada.

e. Lista simplesmente encadeada ordenada por CPF.
f. Lista simplesmente encadeada ordenada (decrescentemente) pelo contador.

Resolva as questões a seguir, justificando:
(a) Quais as melhores estruturas em relação à operação 1?

Resposta: São as estruturas (b), (c) e (f). A estrutura (b) possui complexidade de busca $O(\log n)$. Já as estruturas (c) e (f), embora possuam complexidade de pior caso de uma busca linear $O(n)$, possuem caso médio bem melhor, já que os CPFs mais acessados estão nas primeiras posições.

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

Resposta: São as estruturas (c) e (f). Considerando apenas a inserção, para as estruturas (a), (c) e (f) temos $O(1)$ passos (o novo CPF pode ser inserido no final da estrutura). No entanto, a inserção requer uma busca prévia, o que torna a operação $O(n)$ no pior caso, mas com caso médio bem melhor nas estruturas (c) e (f). A estrutura (b), embora tenha um tempo de busca melhor que as anteriores, tem tempo de inserção $O(n)$, devido ao deslocamento de elementos, quando a inserção se dá no meio da lista.

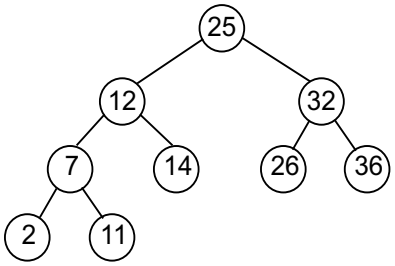
(c) Quais as melhores estruturas em relação à operação 3?

Resposta: É a estrutura (f). Considerando apenas a remoção, temos $O(1)$ passos, tornando-se $O(n)$ quando considerada a busca prévia, sendo bem melhor no caso médio. As estruturas (b) e (c), embora tenham tempos de busca tão bons quanto ou melhores que a estrutura (f), tem tempo de remoção $O(n)$, devido ao deslocamento de elementos, quando a remoção se dá no meio da lista. Já as estruturas (d) e (e), embora também tenham a remoção $O(1)$ e busca $O(n)$, têm um caso médio para busca maior que a estrutura (f).

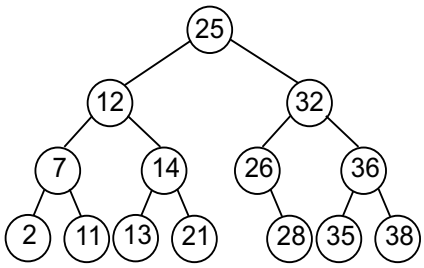
8. (1,0) Desenhe uma árvore binária T que satisfaça os requisitos pedidos, em cada caso.

   a. T é uma árvore completa, estritamente binária, com altura 4 e número mínimo de nós.

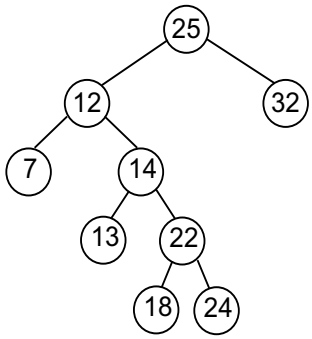
   Resposta:



b. T é uma árvore binária completa, não estritamente binária, com 4 níveis e número máximo de nós.
Resposta:



c. T é uma árvore estritamente binária com 9 nós e altura máxima.
Resposta:



d. T é uma árvore com 10 nós e altura mínima.
Resposta: