Prova P1

Algoritmos e Estrutura de Dados II
2021/2 - 17/03/2022

Nome:
Matrícula:

Questão 1 (1,5 pontos)
Considere o algoritmo a seguir que calcula o menor número de multiplicações escalares para uma Multiplicação de Cadeias de Matrizes:

```
LowCost(p, i, j)
 1  if i == j then
 2    return 0
 3  else
 4    low = MAX_VALUE
 5    for k = i to j-1 do
 6      costa = LowCost(p, i, k)
 7      costb = LowCost(p, k+1, j)
 8      cost = costa + costb + (p[i-1]*p[k]*p[j])
 9      if cost < low then
10        low = cost
11    return low
```
(p = dimensões das matrizes, i = matriz inicial cálculo, j = matriz final cálculo)

Dentre as observações abaixo sobre o algoritmo, selecione aquela que caracteriza um comportamento de problemas superpostos, que permite a otimização do mesmo:
(   ) O algoritmo combina duas características: é recursivo e possui uma repetição.
(   ) Todo algoritmo que possui duas chamadas recursivas (como nas linhas 6 e 7) tem problemas superpostos.
(   ) Sempre que qualquer algoritmo recursivo realiza cálculo com um vetor, há repetição de subproblemas.
(X) As chamadas recursivas das linhas 6 e 7 retornam sempre o mesmo resultado para um dado valor de *p*, *i* e *j*, além disto, chamadas recursivas com o mesmo valor para *p*, *i* e *j* acontecem com frequência, ocasionando repetição do mesmo processo de cálculo.
(   ) NDA

CRITÉRIO DE CORREÇÃO: -0,7 ponto caso haja a seleção de 1 opção adicional além da correta e 0 caso haja a seleção de uma opção incorreta ou a seleção de mais do que 2 opções.

Questão 2 (1,5 pontos)
Suponha que você tenha usado duas sequências "A" e "B" como dados de entrada para um algoritmo de programação dinâmica que resolve o problema da maior subsequência comum entre duas sequências. Como resultado, você obteve a sequência "C". Neste contexto, é correto afirmar que
(   ) Certamente não há outra subsequência comum a "A" e "B" com o mesmo comprimento que "C".
(   ) Certamente não há subsequência comum a "A" e "B" com comprimento menor que "C".
(   ) Pode haver uma subsequência comum a "A" e "B" com comprimento maior que "C", mas somente "C" é uma subsequência de comprimento ótimo.

(X) Dependendo de "A" e "B", pode haver outras sequências com o mesmo comprimento que "C".
(   ) NDA

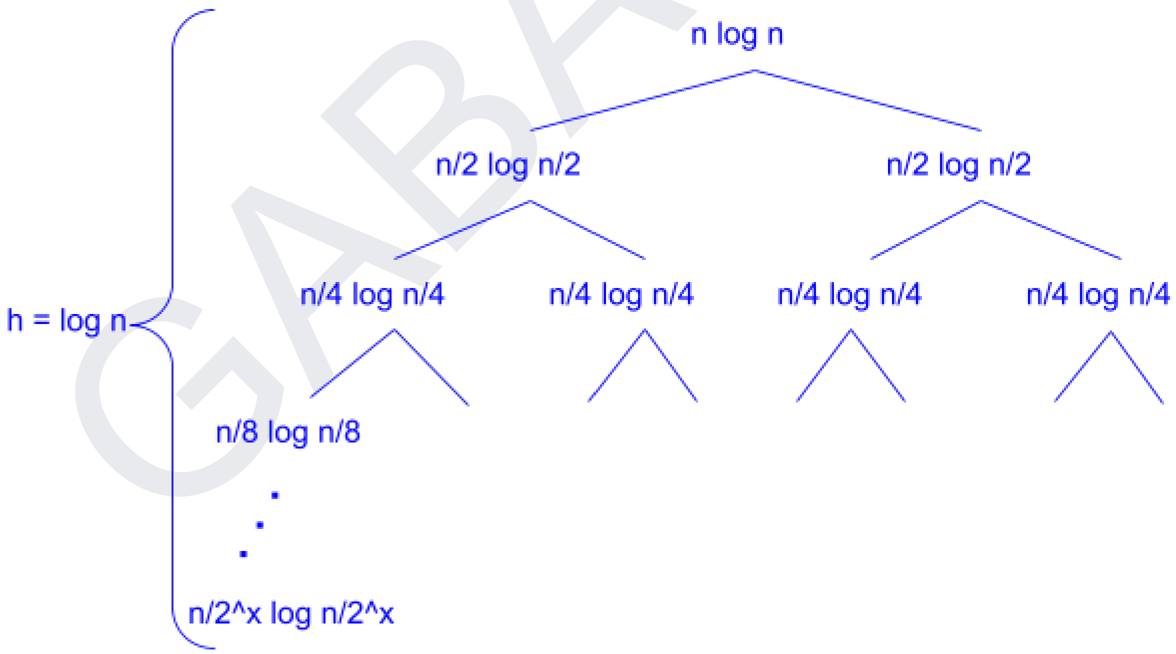
CRITÉRIO DE CORREÇÃO: -0,7 ponto caso haja a seleção de 1 opção adicional além da correta e 0 caso haja a seleção de uma opção incorreta ou a seleção de mais do que 2 opções.

Questão 3 (1,5 pontos)
Programação Dinâmica é utilizada para solucionar:
(   ) Todos algoritmos do tipo dividir e conquistar.
(X) Problemas de otimização, utilizando informações já calculadas para descobrir a solução ótima.
(X) Problemas baseados na resolução recursiva e repetida de subproblemas ótimos.
(X) Problemas baseados em subproblemas superpostos mas não independentes.
(   ) NDA

CRITÉRIO DE CORREÇÃO: 1 ponto para a seleção de 1 opção correta, 1,25 pontos para a seleção de duas opções corretas e 1,5 pontos para a seleção de três opções corretas.

Questão 4 (1,5 pontos)
Qual a solução da recorrência T(n) = 2T(n/2) + n log n? Mostrar o passo a passo para obtenção da solução.

Resposta:
Ao usar árvore de recursão para resolver a recorrência temos:



Temos que o nível x da árvore de recursão contém 2^x chamadas chamadas recursivas, e o tempo em cada uma delas é $(n/2^x)$ log $(n/2^x)$. Portanto, o trabalho total realizado no nível x é n log $(n/2^x)$.
Como o tamanho do problema é reduzido pela metade em cada chamada recursiva, a profundidade da árvore de recursão é log n.

Limite superior em $T(n)$:

Temos log $(n/2^x) \leq$ log n. Em palavras, o tempo total utilizado em cada nível da árvore de recursão é no máximo n log n.
Dado que existem log n níveis na árvore de recursão, o tempo total na árvore de recursão é no máximo (n log n) $\times$ log n = n log^2 n.

Note que o Teorema Mestre não se aplica a esse caso, pois ele não satisfaz os casos 1, 2 ou 3 do Teorema. O caso 3, por exemplo, afirma que f(n) deve ser polinomialmente maior, mas aqui é assintoticamente maior que $n^{\log_b a}$ a apenas um fator log n.

CRITÉRIO DE CORREÇÃO: Será aceito o uso de qualquer método para solução de recorrência desde que apresentados os detalhes da solução.

Questão 5 (2 pontos)
Existem diversas variantes do ***Problema da Mochila***. Uma dessas variantes pode ser definida conforme a seguir: a entrada é uma lista de inteiros positivos x_1, ..., x_n e outro inteiro positivo *K*. Estamos interessados em selecionar alguns dos números x_1, ..., x_n de modo que sua soma seja a maior possível, sem exceder *K*. Por exemplo, se a lista for 3,10,4,6,8 e K = 19, podemos selecionar 3, 10 e 6, obtendo uma soma igual a 19. Podemos pensar nisso como os pesos dos itens que são colocados em uma mochila, onde a mochila só pode conter um peso total de K.

Uma maneira conveniente de indicar quais números escolher consiste em encontrar um conjunto de índices *I*, contendo os índices dos valores na lista a serem considerados. Por exemplo, se forem considerados o primeiro, o terceiro e o quarto números, o conjunto de índices é {1,3,4}. Seja *I* um conjunto de índices, defina S(*I*) como a soma de todos os x_i para *i* em *I*. Por exemplo, se *I* é {1,3,4}, então S(*I*) é $x_1 + x_3 + x_4$. O problema da mochila é encontrar um conjunto de índices I que seja um subconjunto de {1, ..., *n*} tal que S(*I*) seja o maior possível, sujeito à restrição de que S(*I*) não seja maior que *K* .
Para *w* = 0, ..., *K* e *j* = 0, ..., *n*, $W_{j,w}$ será 1 se houver um conjunto de índices *I* que seja um subconjunto de {1, ..., *j* } tal que S(*I*) = *w*, e $W_{j,w}$ será 0, caso contrário. No caso em que *j* = 0, é necessário que *I* esteja vazio. Dito de outra forma, $W_{j,w}$ informa se é possível obter uma soma de exatamente *w* se você só puder escolher entre os primeiros *j* números na lista x_1, ..., x_n.

A partir dessa definição, temos a seguinte recorrência:
$W_{j,0}$ = 1 para *j* >= 0 (já que você pode obter uma soma total de 0 não levando nada).
$W_{0,w}$ = 0 para *w* > 0.
$W_{j,w} = W_{j-1,w}$ ou $W_{j-1,\, w-x_j}$. (Se você quiser obter uma soma total de *w* usando apenas os primeiros *j* números, você pode usar x_j ou não. Se você optar por não usar x_j , então você deve obter a soma *w* usando apenas os primeiros *j* - 1 números . Se você usar x_j , deverá obter uma soma de *w* - x_j com os primeiros *j* - 1 números. Adicionar x_j dá a soma *w*.)

Note que a variante definida anteriormente não pede o subconjunto *I*, somente S(*I*), a maior quantidade que pode ser colocada na mochila. Descreva um algoritmo que resolva esse problema em tempo O(*nK*). Assuma que as operações aritméticas e lógicas levam tempo constante.

Resposta:
A ideia é usar as equações apresentadas no enunciado, mas armazená-las em uma matriz para evitar cálculos repetidos. A resposta é o maior *w* tal que $W_{n,w}$ = 1.

```
Crie uma matriz bidimensional W.
para j = 0,...,n faça W[j,0] = 1;
para w = 1,...,K faça W[0,w] = 0;
para j = 1,...,n faça
    para w = 1,...,K faça
        W[j,w] = W[j-1,w] ou W[j-1,wx[j]];
para w = K, ..., 0 faça
   se W[n,w] == 1 então
      Retornar w.
```

CRITÉRIO DE CORREÇÃO: Além da proposta de algoritmos que resolvam o problema com a complexidade descrita, serão considerados algoritmos que resolvam o problema com complexidade superior à solicitada (1,5 ponto) ou soluções parciais (entre 0,5 e 1,5 pontos, dependendo de quão próximo a proposta esteja da solução do problema.

Questão 6 (2 pontos)
Mostrar, passo a passo, a ordenação realizada pelo Quicksort da string composta pelas 4 primeiras letras do seu nome + as 3 últimas letras do seu sobrenome.

Resposta:
Entrada utilizada
['L', 'U', 'C', 'I', 'N', 'T', 'O']

Escolha do último elemento como pivô: 'O'
Situação do array após uso do pivô 'O': ['L', 'C', 'I', 'N', 'O', 'T', 'U']
Todos os elementos maiores que 'O' estão à sua direita e todos os elementos menores que pivô 'O' estão à sua esquerda.

Escolha do novo pivô como sendo o último elemento à esquerda do pivô anterior: 'N'
Situação do array após uso do pivô 'N': ['L', 'C', 'I', 'N', 'O', 'T', 'U']
Todos os elementos maiores que 'N' estão à sua direita e todos os elementos menores que pivô 'N' estão à sua esquerda.

Escolha do novo pivô como sendo o último elemento à esquerda do pivô anterior: 'I'
Situação do array após uso do pivô 'I': ['C', 'I', 'L', 'N', 'O', 'T', 'U']
Todos os elementos maiores que 'I' estão à sua direita e todos os elementos menores que pivô 'I' estão à sua esquerda. Com isso, temos que todo o lado esquerdo do primeiro pivô selecionado ('O') está ordenado.

Escolha do novo pivô como sendo o último elemento do lado direito do primeiro pivô selecionado ('O'): U
Situação do array após uso do pivô 'U': ['C', 'I', 'L', 'N', 'O', 'T', 'U']

Array ordenado: ['C', 'I', 'L', 'N', 'O', 'T', 'U']

CRITÉRIO DE CORREÇÃO: 0,5 ponto para a definição correta da entrada utilizada e 1,5 pontos para a aplicação do algoritmo, podendo haver débito de pontos caso ocorram erros na aplicação do algoritmo.