Lista de exercícios 3 (Programação Dinâmica)
Cana - 2025.2

**Questão 1.** Seja $P : \mathbb{N} \to \mathbb{N}$ uma função definida da seguinte forma: $P(0) = P(1) = P(2) = P(3) = P(4) = 0$ e, para $n \geq 5$,

$$P(n) = P\Big(\lfloor \frac{n}{2} \rfloor\Big) \ + \ P\Big(\lfloor \frac{n}{2} \rfloor + 1\Big) \ + \ P\Big(\lfloor \frac{n}{2} \rfloor + 2\Big) \ + \ n.$$

Escreva um algoritmo recursivo puro que recebe um número $n$ como entrada e retorna o valor exato de $P(n)$. Calcule a complexidade do seu algoritmo. Escreva agora um algoritmo de programação dinâmica para o mesmo problema e calcule a complexidade. Escreva também um algoritmo de memoização e calcule a complexidade. Qual dos três algoritmos é o mais rápido?

**Questão 2.** No problema da subsequência crescente mais longa, a entrada é uma sequência de números $a_1, \ldots, a_n$ . Uma subsequência é qualquer subconjunto desses números tomados em ordem, da forma, $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ onde $1 \leq i_1 < i_2 < \ldots, i_k \leq n$, e uma subsequência crescente é aquela na qual os números vão ficando estritamente maiores. Elabore um algoritmo de programação dinâmica para encontrar a subsequência crescente de maior comprimento.

**Questão 3.** É dado um tabuleiro quadriculado com 4 linhas e $n$ colunas e um número inteiro escrito em cada quadrado do tabuleiro. Também é dado um conjunto de $2n$ pedras e queremos colocar algumas delas ou todas elas no tabuleiro (cada pedra pode ser colocada em exatamente um quadrado) para maximizar a soma dos inteiros nos quadrados que são cobertos pelas pedras. Há uma restrição: para que disposição das pedras seja legal, nenhum par delas pode estar em quadrados adjacentes horizontal ou verticalmente (adjacência diagonal é permitida).

(a) Determine o número de padrões legais que podem ocorrer em alguma coluna (isoladamente, ignorando as pedras nas colunas adjacentes) e descreva estes padrões.

Chame dois padrões de compatíveis se eles podem ser colocados em colunas adjacentes em uma disposição legal. Vamos considerar subproblemas consistindo nas primeiras $k$ colunas $1 \leq k \leq n$. A cada subproblema pode ser atribuído um tipo, que é o padrão ocorrendo na última coluna.

(b) Usando as noções de compatibilidade e tipo, forneça um algoritmo de programação dinâmica de tempo $O(n)$ para computar uma disposição ótima.

**Questão 4.** Dadas duas strings $x = x_1x_2 \cdots x_n$ e $y = y_1y_2 \cdots y_m$, desejamos encontrar o comprimento da maior **substring (elementos contíguos)** comum delas, isto é, o maior $k$ para o qual existem índices $i$ e $j$ $x_ix_{i+1} \cdots x_{i+k-1} = y_jy_{j+1} \cdots y_{j+k-1}$. Mostre um algoritmo de programação dinâmica (em pseudocódigo) como fazer isso em tempo $O(mn)$.

**Questão 5.** Escreva um algoritmo iterativo (*"bottom up"*) de programação dinâmica em $O(nT)$ que recebe um inteiro positivo $T$ e uma lista com $n$ inteiros positivos $(a_1, a_2, \ldots, a_n)$ e decide se existe algum subconjunto desses inteiros cuja soma é igual a $T$. (Dica: Observe subconjuntos $(a_1, a_2, \ldots, a_k)$ e verifique se a soma é $s$ onde $1 \leq k \leq n$ e $1 \leq s \leq T$).

**Questão 6.** Considere uma região atravessada por um rio, e suponha que as cidade $A_1, A_2, \ldots, A_n$ estão em uma margem do rio, enquanto as cidades $B_1, B_2, \ldots, B_n$ se encontram na outra margem do rio. Os moradores desta região desejam construir pontes conectando as cidades e satisfazendo as condições de que duas pontes não podem se cruzar e deve existir pelo menos uma ponte conectando cada cidade ao outro

lado do rio. Para cada par de cidades $A_i, B_j$, o custo de construção de uma ponte entre as duas cidades é denotado por $c_{ij}$. Deseja-se que o custo total de construção das pontes seja o menor possível. Faça um algoritmo de programação dinâmica para encontrar uma solução ótima para o problema. Determine a complexidade do seu algoritmo.

**Questão 7.** Você recebe $n + 1$ números reais positivos $X = (x_0, x_1, \ldots, x_n)$ e uma sequência de $n$ operadores em $\{+, \times, \wedge\}$, onde $+$ significa soma, $\times$ significa multiplicação e $\wedge$ significa exponenciação. Essas sequências de números e operadores representam uma expressão matemática. Por exemplo, se $X = (0.3, 1, 4, 0.7, 0.2)$ e a sequência de operadores é $(+, \times, +, \times)$, então temos a expressão: $0.3{+}1{\times}4{+}0.7{\times}0.2$. Desejamos colocar parêntesis na expressão de modo que o resultado final seja o mínimo possivel. Também desejamos colocar parêntesis na expressão de modo que o resultado final seja o máximo possivel. Por exemplo, $0.3{+}(1{\times}4){+}(0.7{\times}0.2) = 4.34$, $(0.3{+}1){\times}(4{+}0.7){\times}0.2 = 1.222$, mas $(0.3{+}(1{\times}4){+}0.7){\times}0.2 = 1$. Elabore um algoritmo de programação dinâmica que obtém o modo de colocar parêntesis para obter o valor máximo e o modo de colocar parêntesis para obter o valor mínimo. A complexidade deve ser no máximo $O(n^3)$.

**Questão 8.** No problema do alinhamento de sequências, são recebidas duas sequências $X$ e $Y$ como entrada (não necessariamente do mesmo tamanho), e desejamos "parear"as duas sequências de forma que seus elementos fiquem "o mais próximo possível". Um elemento de $X$ pode ser pareado com uma cópia em $Y$, com outro elemento de $Y$ ou com um símbolo de espaço. Por exemplo, se $X = ATGTTATA$ e $Y = ATCGTCC$, podemos fazer a seguinte correspondência entre elas:

$$AT\&GTTATA$$

$$ATCGT\&C\&C$$

Para medir o quanto um certo alinhamento é bom, é recebida uma matriz $M$ que atribui um valor para cada par de elementos (inclusive espaço), representando uma medida de quanto dois elementos são próximos. Escreva um algoritmo para calcular um alinhamento que maximiza a soma dos valores pareados de acordo com $M$.

**Questão 9.** Você deve cortar um tronco de madeira em vários pedaços. A empresa mais em conta para fazer isso é a União Fácil Corte (UFC), que cobra de acordo com o comprimento do tronco a ser cortado. A máquina de corte deles permite que apenas um corte seja feito por vez. Se queremos fazer vários cortes, é fácil ver que ordens diferentes destes cortes levam a preços diferentes. Por exemplo, considere uma tora com 10 metros de comprimento, que tem que ser cortada a 2, 4 e 7 metros de uma de suas extremidades. Há várias possibilidades. Podemos primeiramente fazer o corte dos 2 metros, depois dos 4 e depois dos 7. Tal ordem custa $10{+}8{+}6 = 24$, porque a primeira tora tinha comprimento 10, o que restou tinha 8 metros de comprimento e o último pedaço tinha comprimento 6. Se cortássemos na ordem 4, depois 2, depois 7, pagaríamos $10 + 4 + 6 = 20$, que é mais barato. Seu chefe encomendou um algoritmo de programação dinâmica que, dado o comprimento $L$ do tronco e $k$ pontos $p_1, \ldots, p_k$ de corte, encontre o custo mínimo para executar esses cortes na UFC.