

Lista de exercícios 2 (Programação Dinâmica)

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

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

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?

2. Uma subsequência contígua de uma sequência  $S$  é uma subsequência de elementos consecutivos de  $S$ . Por exemplo, se  $S = (5 \ 15 \ -30 \ 10 \ -5 \ 40 \ 10)$ , então  $(15 \ -30 \ 10)$  é uma subsequência contígua de  $S$ , mas  $(5 \ 15 \ 40)$  não é. Escreva um algoritmo linear para a seguinte tarefa: receba como entrada uma sequência de números  $(a_1, a_2, \dots, a_n)$  e devolva a subsequência contígua cuja soma é máxima (uma subsequência de tamanho zero tem soma zero). No exemplo anterior, a resposta seria a subsequência  $(10 \ -5 \ 40 \ 10)$  cuja soma é 55. (Dica: Para cada  $j \in \{1, 2, \dots, n\}$ , considere subsequências contíguas terminando exatamente na posição  $j$ ).

3. Você recebe uma sequência  $S[1 \dots n]$  com  $n$  dígitos de 0 a 9 e deseja saber se é possível quebrá-la em números que sejam quadrados ou cubos perfeitos. Por exemplo, se  $S = 125271448164$ , então a resposta é SIM, pois  $S$  pode ser quebrada da seguinte forma  $125, 27, 144, 81, 64$ , cujos números são quadrados ou cubos perfeitos ( $125 = 5^3$ ,  $27 = 3^3$ ,  $144 = 12^2$ ,  $81 = 9^2$ ,  $64 = 8^2$ ). Outra possibilidade seria:  $1, 25, 27, 144, 8, 16, 4$ . Escreva um algoritmo de programação dinâmica que determina se sua sequência  $S$  satisfaz ou não esta condição. A complexidade deve ser no máximo  $O(n^2)$ , onde você pode assumir que existe um algoritmo que retorna em tempo constante se um número dado é quadrado ou cubo perfeito. Caso a resposta seja SIM, faça seu algoritmo escrever a sequência correta de quadrados e/ou cubos perfeitos.

4. Uma subsequência é palíndroma se ela é igual lendo da direita para esquerda ou lendo da esquerda para direita. Por exemplo, a sequência  $(ACGTGTCAAATCG)$  possui muitas subsequências palíndromas, como  $(ACGCA)$  e  $(AGTGA)$ . Mas a subsequência  $(ACT)$  não é palíndroma. Escreva um algoritmo  $O(n^2)$  que recebe uma sequência  $S[1 \dots n]$  e retorna a subsequência palíndroma de tamanho máximo.

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

6. Você recebe  $n$  números reais positivos  $X = (x_1, x_2, \dots, x_n)$  e uma sequência de  $n - 1$  operadores  $op = (op_1, \dots, op_{n-1})$  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 possível. Também desejamos colocar parêntesis na expressão de modo que o resultado final seja o máximo possível. Por exemplo, o máximo e o mínimo do exemplo são respectivamente  $(0.3 + 1) \times (4 + (0.7 \times 0.2)) = 5.382$  e  $(0.3 + (1 \times 4) + 0.7) \times 0.2 = 1$ . Escreva 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  $O(n^3)$ .

**7. [algoritmos gulosos]** Considere um conjunto de livros numerados de 1 a  $n$ . Suponha que o livro  $i$  tem peso  $p_i$  e que  $0 < p_i < 1$  para cada  $i$ . Problema: Dado  $n$  e os números  $p_1, \dots, p_n$ , acondicionar os livros no menor número possível de envelopes de modo que cada envelope tenha no máximo 2 livros e o peso do conteúdo de cada envelope seja no máximo 1. Escreva um algoritmo eficiente que resolva esse problema. Aplique seu algoritmo a um exemplo interessante. Mostre que seu algoritmo está correto.

**8. [algoritmos gulosos]** Escreva um algoritmo eficiente que receba como entrada um conjunto de variáveis  $x_1, \dots, x_n$  e dois conjuntos  $I$  e  $D$  de pares  $(x_i, x_j)$  de variáveis. Os pares  $(x_i, x_j)$  em  $I$  representam restrições de igualdade  $x_i = x_j$  e os pares  $(x_a, x_b)$  em  $D$  representam restrições de desigualdade  $x_a \neq x_b$ . Seu algoritmo deve responder se é possível ou não satisfazer todas as restrições em  $I$  e em  $D$ . Por exemplo, a seguinte entrada não é satisfável:  $I = \{(x_1, x_2), (x_2, x_3), (x_3, x_4)\}$  e  $D = \{(x_1, x_4)\}$ .

**9. [algoritmos gulosos]** Miguel deseja fazer uma festa e está decidindo quem deve chamar. Ele tem  $n$  amigos para convidar e tem uma lista dos pares de amigos que se conhecem. Ele quer que ninguém se sinta deslocado, mas também quer que a festa seja interessante e que pessoas que não se conhecem façam amizade. Assim ele se colocou as seguintes restrições: Para cada convidado, devem existir pelo menos dez pessoas na festa que ele conhece e dez pessoas na festa que ele não conhece. Faça um algoritmo que receba uma lista com  $n$  pessoas e uma lista com os pares dessas pessoas que se conhecem e devolva o maior número de pessoas que poderão ser convidadas sob estas restrições. (a) Descreva com palavras como será o seu algoritmo. (b) Escreva o algoritmo em pseudo-código.

**10. [algoritmos gulosos]** Seja  $1, \dots, n$  um conjunto de tarefas. Cada tarefa consome um dia de trabalho; durante um dia de trabalho somente uma das tarefas pode ser executada. Os dias de trabalho são numerados de 1 a  $n$ . A cada tarefa  $T$  está associado um prazo  $P_T$ : a tarefa deveria ser executada em algum dia do intervalo  $1, \dots, P_T$ . A cada tarefa  $T$  está associada uma multa não-negativa  $M_T$ . Se uma dada tarefa  $T$  é executada depois do prazo  $P_T$ , sou obrigado a pagar a multa  $M_T$  (mas a multa não depende do número de dias de atraso). Problema: Programar as tarefas (ou seja, estabelecer uma bijeção entre as tarefas e os dias de trabalho) de modo a minimizar a multa total. Escreva um algoritmo guloso para resolver o problema. Prove que seu algoritmo está correto. Analise o consumo de tempo.