

 Universidade Federal do Ceará Centro de Ciências Departamento de Computação	2ª Avaliação Parcial Construção e Análise de Algoritmos (ck0183/ck0203) - 2021.1 Profa. Ana Karolinna Maia karolmaia@ufc.br Aluno: Matrícula:	Nota
--	---	------

- A prova deve ser escrita a mão (no papel) e fotografada/escaneada para enviar para correção em um único arquivo no formato PDF.
- O upload do arquivo deve ser feito pelo SIGAA.
- Respostas em linguagens de programação não serão aceitas.

Questão 1 (2,0 pontos). No problema da pilha de caixas são dadas n caixas C_1, C_2, \dots, C_n . Cada caixa C_i possui peso p_i e quantidade de objetos k_i . O problema consiste em construir uma pilha P com todas as caixas, de modo a minimizar o esforço, calculado como $\text{esfor}(P) = \sum_{j=1}^n (k_j \times (\sum_{i=1}^j p_i))$. Prove ou apresente contra-exemplos para as seguintes ideias de como resolver o problema de forma gulosa: *crescente*

- (a) Ordenar as caixas de maneira crescente pelo peso; $\{(1, 1), (2, 5)\}$ $\text{est}(A) = 0 \cdot 1 + 1 \cdot (1+2) = 3$
- (b) Ordenar as caixas de maneira decrescente pela quantidade de objetos.

$$\left\{ \begin{array}{l} \{(1, 1), (10, 2)\} \text{ est}(B) = 20 + 11 \rightarrow \text{desc.} \\ \text{est}(A) = 1 + 22 \end{array} \right\} \text{ est}(B) = 1 \cdot 2 + 0 \cdot (1+2) = 2$$

Questão 2 (4,0 pontos). Seja $1, \dots, n$ um conjunto de tarefas. Em cada dia, é possível executar no máximo **OITO** tarefas, uma para cada hora de trabalho do dia. Os dias de trabalho são numerados como $1, 2, 3, \dots$ e as horas de trabalho são numeradas de 1 a 8. Cada tarefa T tem um prazo P_T : a tarefa deveria ser executada em algum dia do intervalo $1, \dots, P_T$. Cada tarefa T tem uma multa $M_T > 0$: se uma 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 informando em qual dia e hora cada tarefa deve ser executada de modo a minimizar a multa total. Descreva com palavras um algoritmo **guloso** para resolver o problema. Argumente porque seu algoritmo está correto.

Algoritmo: Ordenar as tarefas pelas multas de forma decrescente. Preencher a programação/escala da seguinte forma: Para a tarefa considerada no momento, colocar na primeira posição livre percorrendo a escala de trás para frente a partir do seu prazo. Se não houver nenhuma posição livre, colocar na ultima posição livre disponível.

Vamos representar uma solução qualquer S pela sequência $\langle a_1, a_2, \dots, a_n \rangle$, de acordo com os dias em que as tarefas são programadas, isto é, a_1, \dots, a_8 estão no dia 1, a_9, \dots, a_{16} no dia 2, e assim por diante. Seja S' uma solução diferente da fornecida pelo algoritmo acima. Então existe um elemento a_i em S' com multa menor que outro elemento a_j que impede que a_j seja executado em seu prazo. Dessa forma, $S' = \langle a_1, \dots, a_i, \dots, a_j, \dots, a_n \rangle$, a_j está fora do prazo mas estaria dentro do prazo se estivesse na posição de a_i . Considere a solução $S'' = \langle a_1, \dots, a_j, \dots, a_i, \dots, a_n \rangle$ construída a partir de S' , mas com a_i e a_j em posições invertidas. Queremos mostrar que $S' > S''$, ou seja, tal solução pode ser melhorada ao fazermos em um passo uma escolha semelhante a do algoritmo proposto. Observe que a multa de todos os elementos diferentes de a_i e a_j é mantida. Então $S' - S'' = M_{a_i}^{S'} + M_{a_j}^{S'} - M_{a_j}^{S''} - M_{a_i}^{S''} = M_{a_i}^{S'} + M_{a_j}^{S'} - 0 - M_{a_i}^{S''} = M_{a_i}^{S'} + M_{a_j}^{S'} - M_{a_i}^{S''}$. Se $M_{a_i}^{S'} > 0$, então $M_{a_i}^{S'} = M_{a_i}^{S''} > 0$ e $S' - S'' = M_{a_j}^{S'} > 0$. Se $M_{a_i}^{S'} = 0$, então $M_{a_i}^{S''} = M_{a_i}^{S'} = 0$ e $S' - S'' = M_{a_j}^{S''} > 0$, ou $M_{a_i}^{S''} > 0$ e $S' - S'' = M_{a_j}^{S''} - M_{a_i}^{S''} = M_{a_j}^{S''} - M_{a_i}^{S'} > 0$, uma vez que a multa de a_j é maior que a multa de a_i .

Questão 3 (4,0 pontos). 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, \dots, p_k de corte, encontre o custo mínimo para executar esses cortes na UFC.

- (a) Explique a propriedade de subestrutura ótima desse problema.

Considerando um ponto de corte p_i , a solução ótima em que p_i é o primeiro corte executado é calculada pelas soluções ótimas dos dois subproblemas que restam quando esse corte é executado: dois troncos menores de madeira os pontos de cortes recebidos como entrada, somados ao cumprimento inicial do tronco. Então, para calcular a solução ótima do problema, basta considerar cada ponto de corte como o inicial e retornar a solução de custo mínimo, calculada a partir de cada corte e seus respectivos subproblemas.

- (b) Escreva a equação de recorrência. $P_{ij} = p_j - p_i + \min_{i < k < j} \{P_{ik} + P_{kj}\}$

- (c) Escreva o algoritmo em pseudo-código.

Algoritmo Corte_UFC ($p[0], p_1, \dots, p_k, L$)

Criar matriz $P[K+1][K+1]$

Para $i \leftarrow 0$ até K

$P[i][i+1] \leftarrow 0$

Para $l \leftarrow 3$ até $K+1$

Para $i \leftarrow l$ até $K+1-l+1$

$P[i][i+l-1] \leftarrow \infty$

Para $m \leftarrow i+l$ até $(i+l-1)-1$

$\& P[i][i+l-1] \leftarrow (p_{i+l-1} - p_i) +$

$P[i][m] + P[m][i+l-1]$

$P[i][i+l-1] \leftarrow (p_{i+l-1} - p_i) +$

$P[i][m] + P[m][i+l-1]$

Retorna $P[0][K+1]$