Algoritmos e Estruturas de Dados II
Prova P1 – 2022/1
29/07/2022

Nome completo: _____

Matrícula:_____

Questão 1 (2 pontos)
Considere o problema do troco visto em aula, no qual dados os m tipos de moedas de um país, determinar o número de maneiras distintas para dar um troco de valor n. Seja $m = 5, V = \{1,2,5,6,10\}$. Mostre, passo a passo, o uso do algoritmo de Programação Dinâmica apresentado em aula para encontrar o número de maneiras distintas de fornecer um troco de $n = 7$.

Resposta: O resultado produzido pelo algoritmo usando Programação Dinâmica visto em aula para $m = 5, V = \{1,2,5,6,10\}$ e $n = 7$ é

Moedas\valor	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1
2	1	1	2	2	3	3	3	4
3	1	1	2	2	3	4	5	6
4	1	1	2	2	3	4	6	7
5	1	1	2	2	3	4	6	7

Logo, existem 7 maneiras distintas de fornecer um troco de $n = 7$.

Critério de correção: nota proporcial a evolução do passo a passo para obtenção da resposta.

Questão 2 (2 pontos)
Explicar, em termos gerais, as principais diferenças entre os paradigmas Divisão e Conquista e Programação Dinâmica.

Resposta: Divisão e Conquista é uma técnica inerentemente recursiva: ela divide o problema de entrada em subproblemas menores e, em seguida, resolvemos recursivamente a cada subproblema, combinando suas soluções posteriormente. Nada é feito para evitar que o mesmo subproblema seja resolvido várias vezes.
A Programação Dinâmica não é uma técnica inerentemente recursiva: em sua forma básica, ela constrói vários subproblemas e os resolve, armazenando suas soluções em uma tabela para referência futura. Isso evita que o mesmo subproblema seja resolvido várias vezes.
A eficiência de um algoritmo de Divisão e Conquista geralmente vem do fato de que cada problema é dividido em subproblemas substancialmente menores, o que impede que a árvore de recursão fique muito grande. Por outro lado, a eficiência da Programação Dinâmica normalmente reside no fato de

que o número de subproblemas e suas interdependências são relativamente pequenos (por exemplo, polinômios), e nenhum subproblema é resolvido mais de uma vez.

Critério de correção: 100% quando destacado o principal aspecto abordado no gabarito. Para os demais casos, o aproveitamento será proporcial a clareza e precisão na resposta.

Questão 3. (2 pontos)

Considere uma prova com perguntas n questões. Para cada questão $i = 1, \dots, n$, a questão i tem valor de pontos $v_i > 0$ e requer $m_i > 0$ minutos para ser resolvida. Suponha ainda que nenhum crédito parcial seja concedido (ao contrário desta prova ;-). Você tem como objetivo escrever um algoritmo que, dado $v_1, v_2, \dots, v_n, m_1, m_2, \dots, m_n$ e V, calcula o número mínimo de minutos necessários para ganhar pelo menos V pontos na prova. Por exemplo, você pode usar esse algoritmo para determinar a rapidez com que pode obter 10 na prova. Denote por $M(i, v)$ o número mínimo de minutos necessários para ganhar v pontos quando você está restrito a selecionar entre as perguntas 1 a i. Escreva uma relação de recorrência para $M(i, v)$. Para facilitar, aqui está o caso base:

Para todo i e $v \leq 0$, $M(i, v) = 0$; para $v > 0$, $M(0, v) = \infty$

Resposta: A não atribuição de crédito parcial só permite escolher fazer ou não a questão i. Ao resolver a questão i, serão gastos m_i minutos, e deve-se escolher uma maneira ideal de ganhar os pontos restantes $v - v_i$ entre as outras questões. Se não resolvermos o problema, deve-se escolher uma maneira ideal de ganhar v pontos considerando as questões restantes. A escolha mais rápida é a ideal. Isso produz a recorrência:

$$\boldsymbol{M(i, v) = min\{m_i + M(i-1, v - v_i), M(i-1, v)\}}$$

Critério de correção: 100% somente para apresentação da relação de recorrência correta.

Questão 4. (2 pontos)

Suponha que você recebeu um array não ordenado contendo todos os inteiros no intervalo 0 a n, exceto por um inteiro o qual denotaremos como número ausente. Assuma que $n = 2^k - 1$. Escreva um algoritmo de Divisão e Conquista para encontrar o número ausente. Será dado crédito parcial para algoritmos que não sejam de Divisão e Conquista. Argumente (informalmente) que seu algoritmo está correto e analise seu tempo de execução.

Resposta: Considere uma função de seleção que retorna o elemento mediano x. Verifique se x está no array. Caso x não seja encontrado, isso significa que ele é o número que falta. Caso contrário, o array é particionado em torno de x em elementos $\leq x$ e $> x$. Se o primeiro tiver tamanho menor que $x + 1$, então resolvemos recursivamente este subarray. Caso contrário, resolvemos recursivamente o outro subarray.

Algoritmo InteiroFaltante($A, [i, j]$)

1. Determine o elemento mediano x no intervalo de i a j
2. Verifique se x está em A e retorna x se o mesmo não for encontrado
3. Particione A em duas partes B e C, onde B possui os elementos $\leq x$ e C possui os elementos $> x$
4. Se Tamanho$(B) < x + 1$
5. InteiroFaltante($B, [i, x]$)
6. Senão InteiroFaltante($C, [x + 1, j]$)

A complexidade do algoritmo é $O(n)$, já que a recorrência para o tempo de execução deste algoritmo é $T(n/2) + n$, que é $O(n)$ pelo Teorema Mestre.

Critério de correção: 100% somente para proposta do algoritmo usando Divisão e Conquista e apresentação da argumentação sobre o correto funcionamento do algoritmo e complexidade.
90% para apresentação do algoritmo argumentado sobre sua correção ou complexidade.
80% para apresentação somente do algoritmo correto com Divisão e Conquista.
60% se o algoritmo proposto resolver o problema e não usar Divisão e Conquista, além de apresentar a argumentação para sua corretude e complexidade.
Algoritmos que não resolvem completamente o problema serão parcialmente considerados.

Questão 5. (2 pontos)
Mostre, passo a passo, o uso do Quicksort para a ordenação da string de 6 letras: 3 primeiras letras do seu primeiro nome + 3 últimas letras do seu sobrenome.

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

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

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

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

Critério de correção: 100% para apresentação do passo a passo completo para a ordenação.
Na ordenação completa, será considerado 80% caso o pivô não seja apresentado em todos os passos ou caso os índices i e j não sejam apresentados.
Nos demais casos, a pontuação será proporcial a evolução do passo a passo para ordenação da entrada.
30% para ordenação usando MergeSort.

Boa sorte!