**Prova sobre Recursão, Divisão e Conquista e Programação Dinâmica**

**Instruções:**

* Esta prova contém 30 questões, divididas em teóricas e de código.
* Leia atentamente cada questão antes de responder.
* Para as questões de múltipla escolha, marque apenas uma alternativa correta.
* Justifique suas respostas nas questões discursivas e de análise de código quando solicitado.

**Questões Teóricas (15 pontos) 7 de 8**

**Múltipla Escolha (8 questões, 1 ponto cada)**

1. Qual a condição essencial para que uma função recursiva não cause um loop infinito? ✅  
     
   a) Utilizar variáveis globais.   
   b) Chamar outra função dentro da função recursiva.   
   **c) Possuir um caso base que interrompa a recursão.**   
   d) Utilizar estruturas de repetição dentro da função.
2. A técnica de Divisão e Conquista geralmente envolve quais três passos principais? ✅  
     
   a) Inicialização, processamento e finalização.  
   b) Declaração, atribuição e comparação.   
   **c) Divisão, Conquista e Combinação.**   
   d) Entrada, saída e armazenamento.
3. Algoritmos baseados em Divisão e Conquista frequentemente possuem qual característica em sua estrutura? ✅  
     
   a) São puramente iterativos.   
   b) Utilizam principalmente estruturas de dados lineares.   
   **c) São, em geral, recursivos.**d) Dependem fortemente de operações de ordenação prévias.
4. Qual das seguintes situações indica que a estratégia de Divisão e Conquista pode ser utilizada com sucesso? ✅  
     
   a) Quando a combinação dos resultados dos subproblemas é computacionalmente cara.   
   b) Quando os subproblemas gerados são de tamanhos significativamente diferentes.   
   **c) Quando é possível decompor uma instância em sub-instâncias.**d) Quando não há subestrutura ótima no problema.
5. A Programação Dinâmica é particularmente útil para problemas onde a abordagem de Divisão e Conquista tradicional pode levar a: ❌  
     
   a) Um número linear de subproblemas.   
   b) Subproblemas independentes. **c) Um número exponencial de subproblemas.**  
    d) Uma combinação trivial de soluções.
6. Qual a principal diferença entre a abordagem "top-down" (memorização) e "bottom-up" (tabulação) na Programação Dinâmica? ✅  
     
   a) A complexidade de tempo da solução.  
    b) A necessidade de definir um caso base.   
   **c) A ordem em que os subproblemas são resolvidos.**   
   d) A quantidade de memória utilizada.
7. Qual das seguintes propriedades é fundamental para que um problema possa ser eficientemente resolvido com Programação Dinâmica? ✅  
     
   a) A inexistência de subproblemas sobrepostos.  
   b) A possibilidade de dividir o problema em um número infinito de subproblemas.   
   c) A garantia de que a solução ótima global não depende das soluções ótimas dos subproblemas.   
   **d) A existência de subproblemas sobrepostos.**
8. No contexto do problema da Distância de Edição, o preenchimento da tabela de programação dinâmica considera principalmente os custos de quais operações para transformar uma string em outra? ✅  
     
   a) Apenas inserção e remoção.   
   b) Apenas substituição.   
   **c) Inserção, remoção e substituição.**   
   d) Apenas operações de comparação.

**Discursivas (7 questões, 1 ponto cada) 5,3**

1. Explique brevemente o conceito de **recursão indireta**. 1.0  
     
   A recursão indireta ocorre quando uma função chama outra função, por exemplo, função A chama a função B, em que contenha uma chamada de volta, ou seja, contém uma chamada que devolve diretamente para a função A. Por fim, podemos concluir que a recursão indireta é feita através de uma cadeia de chamadas entre duas ou mais funções até que ocorra uma condição de parada.  
     
   **Correção:** A recursão indireta ocorre quando uma função chama outra função, que por sua vez chama a função original, formando um ciclo de chamadas entre duas ou mais funções. Esse processo continua até que uma condição de parada seja atingida.
2. Descreva uma **desvantagem** da utilização da técnica de Divisão e Conquista em algoritmos. 0.8  
     
   Uma das principais desvantagens da divisão e conquista é a repetição de subproblemas. Esse problema pode ser resolvido com memorização em programação dinâmica   
     
   **Correção:** Uma desvantagem da técnica de Divisão e Conquista é o alto custo de chamadas recursivas e alocação de memória, especialmente quando o número de subproblemas cresce rapidamente, podendo impactar o desempenho e causar estouro de pilha.
3. Cite um exemplo clássico de problema que pode ser resolvido utilizando a técnica de Divisão e Conquista. 1  
     
   Karatsuba, algoritmo utilizado para a multiplicação de grandes números inteiros.  
     
   **Correção**: Um exemplo clássico é o algoritmo de Karatsuba, utilizado para multiplicar grandes números inteiros. Ele divide os números em partes menores, resolve multiplicações parciais recursivamente e combina os resultados, reduzindo a complexidade do algoritmo em comparação com o método tradicional.
4. Qual a motivação principal por trás da criação da técnica de Programação Dinâmica? 1   
     
   A motivação principal por trás da programação dinâmica é o armazenamento das resoluções dos subproblemas (de maneira ótima) em memória (array ou dicionário) para evitar resoluções desnecessárias.
5. Explique o conceito de **memorização** na Programação Dinâmica. 1  
     
   O conceito de memorização consiste em armazenar em memória, podendo ser um dicionário ou um array a solução ótima dos subproblemas. Essas soluções são armazenadas para que o programa faça consultas nessa memória e evite soluções desnecessárias diminuindo o custo computacional

No problema da Mochila (Knapsack), qual o objetivo de construir uma tabela maxTab? 0,7  
  
A tabela maxTab é uma forma de resolver o problema da mochila utilizando a abordagem de tabulação em que são alocados a capacitada da mochila e valor dos objetos que irão entrar nessas mochilas. O Max tab é a reunião de várias possibilidades e ver qual delas ocupa o menor espaço e agrega o maior valor.  
  
**Correção:** A tabela maxTab armazena o valor máximo que pode ser alcançado para cada subcapacidade da mochila, permitindo resolver o problema incrementalmente e evitando recalcular soluções.  
  
Em geral, o problema não é "ocupar o menor espaço", mas **otimizar o valor** **sem ultrapassar** a capacidade.

1. Descreva brevemente como o **caso base** é importante no algoritmo recursivo para o cálculo do fatorial. 0.8  
     
   O caso base é crucial nos algoritmos recursivos para evitar que sejam criadas funções infinitas até que ocorra um estouro de pilha. Os casos base são condições de parada com relação aos algoritmos recursivos.  
     
   **Correção:** O caso base é essencial em algoritmos recursivos, como no cálculo do fatorial, para interromper as chamadas recursivas. Por exemplo, ao definir que o fatorial de 0 é 1, garantimos que a recursão pare nesse ponto e evite chamadas infinitas que causariam estouro de pilha.

**Questões de Código (15 pontos)**

**Análise de Código (8 questões, 1 ponto cada)**

1. O que a seguinte função recursiva calcula para uma entrada n inteira não negativa?  
     
   O algoritmo acima calcula o fatorial do número passado por parâmetro

def misterio(n):

if n == 0:

return 0

else:

return n + misterio(n - 1)

1. Qual será a saída da seguinte função recursiva quando chamada com funcao\_teste(4)?  
     
   funcao\_teste(4)  
    não entra no if  
    função\_teste(2)+2  
      
   função\_teste(2)  
    não entra no if  
    função\_teste(0) + 2

def funcao\_teste(n):

if n <= 1:

return 1

else:

return funcao\_teste(n - 2) + 2

1. O código a seguir implementa uma etapa de qual algoritmo de Divisão e Conquista? O que ele faz?  
     
   Ele cria um array e instancia duas variáveis de posição (i e j) com o valor de 0. Ele abre um loop que só se encerra depois que os índices fiquem menores que o tamanho dos arrays recebidos por parâmetro (esquerda e direita). Caso o valor da esquerda seja menor, adiciona o valor da esquerda no array resultado na posição i e vice-versa.

def combinar(esquerda, direita):

resultado = []

i = j = 0

while i < len(esquerda) and j < len(direita):

if esquerda[i] < direita[j]:

resultado.append(esquerda[i])

i += 1

else:

resultado.append(direita[j])

j += 1

resultado.extend(esquerda[i:])

resultado.extend(direita[j:])

return resultado

1. A seguinte função recursiva implementa a exponenciação. Identifique um possível problema de eficiência nesta implementação.  
     
   Essa potência cria uma função toda vez que for fazer o cálculo de multiplicação para efetuar a potencialização. Para corrigir esse problema o correto seria implementar a técnica de memorização da programação dinâmica.

def potencia\_ruim(a, n):

if n == 0:

return 1

else:

return potencia\_ruim(a, n // 2) \* potencia\_ruim(a, n // 2) \* (a if n % 2 == 1 else 1)

1. O seguinte trecho de código faz parte de uma solução de Programação Dinâmica para o problema de Fibonacci. O que representa o array memo?  
     
   O array memo representa uma estrutura para armazenar o resultado das operações dos subproblemas.

memo = [-1] \* (n + 1)

def fibonacci\_memoizado(n, memo):

if memo[n] != -1:

return memo[n]

if n <= 1:

memo[n] = n

else:

memo[n] = fibonacci\_memoizado(n - 1, memo) + fibonacci\_memoizado(n - 2, memo)

return memo[n]

1. No contexto do problema da Mochila com a tabela maxTab, o que o valor de maxTab[i][j] geralmente representa?
2. O seguinte trecho de código é parte da solução para o problema da Distância de Edição. Explique o que a linha matriz[i] = matriz[i-1] + 1 está calculando.

def distancia\_edicao(A, B):

m = len(A)

n = len(B)

matriz = [[0 for \_ in range(n + 1)] for \_ in range(m + 1)]

for i in range(m + 1):

matriz[i] = matriz[i-1] + 1 # Explique esta linha

for j in range(n + 1):

matriz[j] = matriz[j-1] + 1

# ... restante do código ...

return matriz[m][n]

1. O seguinte código recursivo calcula o máximo em um vetor usando Divisão e Conquista. Identifique o caso base.

def encontrar\_maximo\_recursivo(A, inicio, fim):

if fim - inicio <= 1:

return max(A[inicio], A[fim])

else:

meio = (inicio + fim) // 2

max\_esquerda = encontrar\_maximo\_recursivo(A, inicio, meio)

max\_direita = encontrar\_maximo\_recursivo(A, meio + 1, fim)

return max(max\_esquerda, max\_direita)

**Construção de Código (7 questões, 1 ponto cada)**

1. Escreva uma função recursiva em Python chamada soma\_digitos que receba um inteiro positivo e retorne a soma de seus dígitos. (Exemplo: soma\_digitos(123) deve retornar 6).
2. Implemente uma função em Python chamada busca\_binaria\_recursiva que realize uma busca binária recursiva em uma lista ordenada por um determinado elemento. A função deve retornar o índice do elemento se encontrado, ou -1 caso contrário.
3. Escreva uma função em Python chamada multiplicacao\_ingenua que implemente a multiplicação de dois inteiros positivos a e b utilizando recursão (apenas somas e subtrações são permitidas na chamada recursiva).
4. Implemente uma função em Python chamada fibonacci\_tabulacao que calcule o n-ésimo número de Fibonacci utilizando a abordagem de tabulação (bottom-up) da Programação Dinâmica.
5. Considere um problema onde você precisa encontrar o número mínimo de moedas para formar um determinado valor, dadas as denominações das moedas disponíveis. Escreva uma função em Python chamada troco\_recursivo\_ingenuo que resolva esse problema utilizando recursão (sem otimizações de Programação Dinâmica). Assuma que você tem uma lista de denominações moedas e o valor alvo valor.
6. Escreva uma função em Python que implemente o passo de **divisão** para a multiplicação de dois números grandes (representados como strings de dígitos) utilizando a abordagem de Divisão e Conquista, dividindo cada número em duas metades (assuma que o número de dígitos seja par).
7. Implemente uma função em Python que, dada uma tabela dp já preenchida para o problema da Distância de Edição e duas strings A e B, retorne a própria sequência de operações (inserção, remoção, substituição) que leva da string A para B com o custo mínimo (você não precisa implementar o preenchimento da tabela, apenas o "backtracking" para encontrar a sequência).