Web development 2025/1

Global Solution	2o.SEM/2025	CURSO:	ENGENHARIA DE SOFTWARE
DISCIPLINA:	Dynamic Programming	PROFESSOR:	Marcelo Amorim

Global Solution FIAP 2025 – O Futuro do Trabalho

Dynamic Programming - Turmas: 2ESR | 2ESPG | 2ESA

Bem-vindos a este desafio empolgante que aplicará todo o conhecimento adquirido ao longo do semestre!

Otimização de Portfólio de Projetos

Tema: Otimização no Futuro do Trabalho

No cenário do Futuro do Trabalho, as empresas especializadas em consultoria ou desenvolvimento de tecnologia precisam otimizar a alocação de seu recurso mais valioso: o tempo e a *expertise* de seus colaboradores mais qualificados.

O Problema do Portfólio: Uma empresa de consultoria possui uma capacidade limitada de **Horas-Especialista** para o próximo trimestre. Eles têm uma lista de projetos potenciais, onde cada projeto exige um certo volume desse recurso limitado e oferece um valor (lucro ou impacto estratégico) específico.

Objetivo: Determinar o conjunto ideal de projetos a serem aceitos para **maximizar o valor total** (impacto/lucro) para a empresa, sem exceder a capacidade total disponível de Horas-Especialista.

Tarefa: Otimização em Quatro Fases

Você deve implementar o problema de Otimização de Portfólio de Projetos, que é uma aplicação direta do **Problema da Mochila 0/1 (0/1 Knapsack Problem)**, um conceito fundamental que exploramos em aula. A implementação deve ser em uma linguagem de sua escolha (preferencialmente Python ou JavaScript), seguindo **obrigatoriamente** as quatro estratégias de implementação abaixo:

Dados de Exemplo

Considere a seguinte capacidade máxima e lista de projetos, onde **E** é a **Hora-Especialista Requerida** (custo do projeto) e **V** é o **Valor** (lucro/impacto) do projeto:

- **Capacidade Máxima de Horas-Especialista (Limite de Recurso):** $C = 10$
- **Projetos (Nome, (V) Valor, (E) Horas-Especialista Requeridas):**
  - Projeto A: V=12, E=4
  - Projeto B: V=10, E=3
  - Projeto C: V=7, E=2
  - Projeto D: V=4, E=3

Fase 1: Estratégia Gulosa (Greedy)

Implemente uma função que tente resolver o problema usando uma abordagem **Guloso (Greedy)**.

- **Regra Gulosa Sugerida:** Priorize os projetos com a maior **Relação Valor/Horas-Especialista** (V/E).
- **Requisito:** A função deve selecionar projetos sequencialmente usando esta regra até que a capacidade seja esgotada.

Observação: Esta estratégia *não garante* a solução ótima e deve ser usada para demonstrar por que uma abordagem ingênua falha.

Fase 2: Solução Recursiva Pura

Implemente uma função recursiva pura para resolver o problema, explorando todas as combinações de projetos.

- **Fórmula de Recorrência (Conceitual):**

$$MaximoValor(i, c) = \max \begin{cases} MaximoValor(i-1, c) & \rightarrow \text{Não incluir} \\ Valor_i + MaximoValor(i-1, c - Horas - Especialista_i) & \rightarrow \text{Incluir o Pr} \end{cases}$$

- **Requisito:** A função deve retornar o valor máximo e **deve** sofrer de redundância, recalculando subproblemas.

Fase 3: Programação Dinâmica Top-Down (Memoização)

Otimize a solução da Fase 2, adicionando um mecanismo de **Memoização** (armazenamento de resultados).

- **Requisito:** Implemente a mesma lógica recursiva, mas armazene os resultados de `MaximoValor(i, c)` em uma estrutura de dados (ex: dicionário, matriz) antes de retorná-los. Se o resultado de um subproblema for solicitado novamente, ele deve ser consultado na estrutura de armazenamento em vez de ser recalculado.

Fase 4: Programação Dinâmica Bottom-Up (Iterativa)

Implemente a solução de Programação Dinâmica de forma **Iterativa (Bottom-Up)**.

- **Requisito:** Construa uma tabela (matriz T) onde $T[i][c]$ representa o valor máximo que pode ser obtido com os primeiros i projetos e uma capacidade c. O resultado final será o valor na última célula da tabela, $T[N][C]$.

Requisitos Avaliação

A entrega deve incluir um arquivo de código-fonte contendo as quatro funções distintas, com a seguinte documentação:

1. **Funções Claras:** Cada uma das quatro abordagens (Greedy, Recursiva, Memoização, PD Iterativa) deve ser uma função separada e bem nomeada.
2. **Comentários:** O código deve ser amplamente comentado, explicando a lógica, o caso base da recursão e o significado das células da matriz de PD.
3. **Análise de Desempenho (Teórica):** Inclua um pequeno bloco de comentários explicando as complexidades de tempo teóricas (O notação) das quatro abordagens e qual é a mais eficiente e por quê.

Critério	Peso	Descrição
Implementação Correta	50%	As quatro funções devem estar implementadas, sendo que as soluções de PD devem retornar o valor ótimo para pelo menos 4 casos de teste a sua escolha.

Critério	Peso	Descrição
Uso de Memoização/Tabela	20%	Correta aplicação da estrutura de armazenamento (memoização na Fase 3 e tabela na Fase 4).
Identificação da Solução	15%	Apresente algum caso de teste onde a solução Gulosa deve falhar, e as soluções de PD devem encontrar o resultado ótimo.
Clareza e Documentação	15%	Código limpo, bem comentado e com a análise de complexidade correta.

Entrega

- Repositório Github: Crie o projeto dentro de uma Organização com seu time como colaboradores
- Repositório **DEVE OBRIGATORIAMENTE** ter um arquivo `README.MD` contendo nome e RM dos alunos integrantes, descrevendo detalhes do projeto, instruções de uso, requisitos, dependências e demais informações relevantes ao projeto. Será avaliada a clareza e organização do conteúdo apresentado.
- Implementações em Python
- Apenas um membro da equipe deve realizar a entrega por meio da respectiva tarefa criada para esta atividade no Teams.