Pedro Henrique Andrade Trindade

0.1 Questão 2: (Plantação de tomates!)

0.1.1 Item A

(a) Estratégia gulosa. Considere a seguinte estratégia gulosa para resolver esse problema: "enquanto houver espaço disponível para plantar um pé de tomate, plante naquele que oferece a maior deliciosidade". Lembre-se de que um espaço só está disponível para plantio se seus vizinhos imediatos não estiverem ocupados. Construa um exemplo em que essa estratégia é ótima e outro em que ela não produz a solução ótima.

Resposta:

Um exemplo de vetor T aonde esta estratégia descrita traz a solução ótima é o vetor

$$T = [5, 1, 10, 1, 11]$$

Porque seguindo passo à passo, inicialmente encolhemos o valor 11 na última posição, então partimos para o 10 (como não há nenhum tomate adjacente, podemos plantar lá sem problemas) e depois para o 5 (que também está igualmente espaçado, portanto havendo a possibilidade de plantarmos um tomate na posição respectiva).
Neste processo a deliciosidade total seria D = 11 + 10 + 5 = 26. Que é a deliciosidade máxima possível a ser alcançada (podemos checar por exaustão, neste pequeno exemplo).

Um caso aonde esta estratégia não nos dá a solução ótima é quando temos, por exemplo,

$$T = [6, 10, 8, 1].$$

Neste caso, temos que escolhemos plantar na posição 10, então nossa única alternativa é plantar na posição de valor 1, assim temos a deliciosidade total como D = 10 + 1 = 11. Enquanto pode-se notar que plantar primeiro na posição de valor 8, então plantar na primeira posição de valor 6, encontramos uma deliciosidade total de D = 8 + 6 = 14, maior que a solução gulosa.

0.1.2 Item B

(b) Programação dinâmica. Construa um algoritmo recursivo que resolva esse problema, isto é, forneça a deliciosidade ótima total e o padrão ótimo de plantio para um dado vetor de deliciosidades T . Explique como a subestrutura ótima é explorada pelo seu algoritmo. A solução pode ser resolvida de forma iterativa e ordenada para melhorar a eficiência? Justifique.

Resposta:

O algoritmo planejado para resolver o problema explora a subestrutura ótima através de uma relação de recorrência que considera duas opções em cada posição i: plantar (adicionando T_i à solução ótima até $i-2$) ou não plantar (herdando a solução ótima até $i-1$), maximizando a deliciosidade total.

$$\mathrm{s}[i] = \begin{cases} T_0 & \text{se } i = 0, \\ \max(T_0, T_1) & \text{se } i = 1, \\ \max(T_i + \mathrm{s}[i-2], \mathrm{s}[i-1]) & \text{se } i \geq 2. \end{cases}$$

A reconstrução do padrão de plantio ótimo é feita percorrendo s de trás para frente, identificando as posições onde $s[i] \neq s[i-1]$ (indicando plantio em i) e pulando posições adjacentes.

O algoritmo roda em complexidade $\mathcal{O}(2^n)$ pois cada chamada ao problema adiciona mais duas novas chamadas em sequência, assim a árvore de decisão cresce exponencialmente com cada chamada.

Existe a posibilidade de fazer uma versão iterativa desse algoritmo, mais eficiente, preenchendo uma tabela "s" de maneira *bottom-up*, começão do inicio s_0 que seria o caso base para um array com 1 espaço para plantio (com deliciosidade T0) assim como s_1, e então construímos $s_2, \dots, s_n$ pois cada s_i pois eles depende apenas da solução do problema para dois indices anteriores, este algoritmo possuiria complexidade $\mathcal{O}(n)$, sendo mais eficiente que o algoritmo recursivo ingenuo.

Um detalhe importante de mencionar, é que se utilizarmos as tecnicas de memoização e salvarmos resultados para não repetirmos operações em chamadas repetidas, nosso algoritmo também se torna $\mathcal{O}(n)$ (que foi como eu implementei).

[27]:
```python
def max_deliciosidade(T, i, memo):
    # Caso base
    if i < 0:
        return 0

    # Verifica se já calculamos este subproblema
    if memo[i] != -1:
        return memo[i]

    # Opção 1: Não plantar na posição i
    opcao_nao_plantar = max_deliciosidade(T, i-1, memo)

    # Opção 2: Plantar na posição i (só possível se i-1 não foi plantado)
    opcao_plantar = T[i] + max_deliciosidade(T, i-2, memo)

    # Escolhe a melhor opção
    memo[i] = max(opcao_nao_plantar, opcao_plantar)
    return memo[i]

def plantar_tomates_dp(T):
    n = len(T)
    memo = [-1] * n  # Tabela de memoização
    deliciosidade_total = max_deliciosidade(T, n-1, memo)

    # Reconstruir a solução (padrão de plantio)
    plantio = []
```

```python
    i = n - 1
    while i >= 0:
        if i == 0:
            if memo[i] == T[i]:  # Se o valor veio de plantar aqui
                plantio.append(i)
            break
        if memo[i] == memo[i-1]:  # Não plantou em i
            i -= 1
        else:  # Plantou em i
            plantio.append(i)
            i -= 2  # Pula o vizinho

    plantio.reverse()  # Para manter a ordem original
    return deliciosidade_total, plantio

# Exemplo do enunciado
T = [21, 4, 6, 20, 2, 5]

deliciosidade_total, plantio = plantar_tomates_dp(T)

print(f"Deliciosidade total: {deliciosidade_total}")
print(f"Indices do plantio : {plantio}")
print(f"Deliciosidades dos tomates plantados: {[T[i] for i in plantio]}")
```

```
Deliciosidade total: 46
Indices do plantio : [0, 3, 5]
Deliciosidades dos tomates plantados: [21, 20, 5]
```

(c) Programação linear inteira. Modele esse problema como um problema de otimização linear inteira (PLI). Detalhe a escolha das variáveis de decisão e das restrições.

Resposta:

Nosso problema do plantio será resolvido utilizando a seguinte modelagem com PLI

$$
\begin{aligned}
\text{maximize} \quad & \sum_{i=1}^{n} T_i x_i \\
\text{sujeito a} \quad & x_i + x_{i+1} \leq 1, \quad i = 1, \dots, n-1 \\
& x_i \in \mathbb{B}, \quad i = 1, \dots, n
\end{aligned}
$$

Temos nesta formulação que as variáveis de decisão x_i no problema de plantio são variáveis binárias que representam se um pé de tomate é plantado ($x_i = 1$) ou não ($x_i = 0$) na posição i. Já o vetor Ti, como descrito no enunciado, contém os valores de deliciosidade associados a cada posição possível de plantio. A função objetivo $\sum_{i=1}^{n} T_i x_i$ busca maximizar a deliciosidade total dos pés plantados, somando apenas as deliciosidades das posições onde efetivamente se planta.

As restrições de vizinhança $x_i + x_{i+1} \leq 1$ garantem que não haja plantio em posições adjacentes, se uma posição i for escolhida, ou seja, $x_i = 1$, a sua vizinha é forçada a não ser escolhida por conta da desigualdade ser com variaveis binárias. Isso assegura que cada pé plantado tenha espaço livre ao redor.

[29]:
```python
import cvxpy as cp
import numpy as np

def plantio_tomate_pli(deliciosidade):
    n = len(deliciosidade)

    # Variável de decisão binária
    x = cp.Variable(n, boolean=True)

    # Função objetivo - maximizar a deliciosidade total
    objetivo = cp.Maximize(deliciosidade @ x)

    # Restrições - não pode plantar em posições adjacentes
    restricoes = []
    for i in range(n-1):
        restricoes.append(x[i] + x[i+1] <= 1)

    # Formular e resolver o problema
    prob = cp.Problem(objetivo, restricoes)
    prob.solve()

    # Retornar a deliciosidade total e o padrão de plantio
    deliciosidade_total = prob.value
    padrao_plantio = np.round(x.value).astype(int)

    return deliciosidade_total, padrao_plantio

# exemplo do enunciado
T = np.array([21, 4, 6, 20, 2, 5])
total, padrao_plantio = plantio_tomate_pli(T)

print(f"Deliciosidade total máxima: {int(total)}")
print(f"Padrão de plantio ótimo: {padrao_plantio}")
print(f"Deliciosidades dos tomates plantados: {[T[i].item() for i, result in␣
↪enumerate(padrao_plantio) if result == 1]}")
```

```
Deliciosidade total máxima: 46
Padrão de plantio ótimo: [1 0 0 1 0 1]
Deliciosidades dos tomates plantados: [21, 20, 5]
```

(d) Valide as duas abordagens ótimas desenvolvidas para os vetores

$T1 = [5, 12, 10, 7, 15, 10, 11, 5, 8, 10]$ e $T2 = [10, 12, 5, 12, 20, 18, 5, 3, 2, 8]$.

[28]:
```python
T1 = np.array([5, 12, 10, 7, 15, 10, 11, 5, 8, 10])
total_dp, plantio_dp = plantar_tomates_dp(T1)
```

```python
total_pli, plantio_pli = plantio_tomate_pli(T1)

print("\nEXEMPLO T1!!!!!")

print("\n CASO COM PROGRAMACAO DINAMICA T1")
print(f"\tDeliciosidade total: {total_dp}")
print(f"\tIndices do plantio : {plantio_dp}")
print(f"\tDeliciosidades dos tomates plantados: {[T1[i].item() for i in␣
↪plantio_dp]}")

print("\n CASO COM PLI T1")
print(f"\tDeliciosidade total máxima: {total_pli}")
print(f"\tPadrão de plantio ótimo: {plantio_pli}")
print(f"\tDeliciosidades dos tomates plantados: {[T1[i].item() for i, result in␣
↪enumerate(plantio_pli) if result == 1]}")

T2 = np.array([10, 12, 5, 12, 20, 18, 5, 3, 2, 8])
total_dp, plantio_dp = plantar_tomates_dp(T2)
total_pli, plantio_pli = plantio_tomate_pli(T2)

print("\nEXEMPLO T2!!!!!")
print("\n CASO COM PROGRAMACAO DINAMICA T2")
print(f"\tDeliciosidade total: {total_dp}")
print(f"\tIndices do plantio : {plantio_dp}")
print(f"\tDeliciosidades dos tomates plantados: {[T2[i].item() for i in␣
↪plantio_dp]}")

print("\n CASO COM PLI T2")
print(f"\tDeliciosidade total máxima: {total_pli}")
print(f"\tPadrão de plantio ótimo: {plantio_pli}")
print(f"\tDeliciosidades dos tomates plantados: {[T2[i].item() for i, result in␣
↪enumerate(plantio_pli) if result == 1]}")
```

```
EXEMPLO T1!!!!!

 CASO COM PROGRAMACAO DINAMICA T1
        Deliciosidade total: 51
        Indices do plantio : [0, 2, 4, 6, 9]
        Deliciosidades dos tomates plantados: [5, 10, 15, 11, 10]

 CASO COM PLI T1
        Deliciosidade total máxima: 51.0
        Padrão de plantio ótimo: [1 0 1 0 1 0 1 0 0 1]
        Deliciosidades dos tomates plantados: [5, 10, 15, 11, 10]

EXEMPLO T2!!!!!
```

```
CASO COM PROGRAMACAO DINAMICA T2
       Deliciosidade total: 53
       Indices do plantio : [1, 3, 5, 7, 9]
       Deliciosidades dos tomates plantados: [12, 12, 18, 3, 8]

CASO COM PLI T2
       Deliciosidade total máxima: 53.0
       Padrão de plantio ótimo: [0 1 0 1 0 1 0 1 0 1]
       Deliciosidades dos tomates plantados: [12, 12, 18, 3, 8]
```