atividade-02

Pedro Henrique Andrade Trindade

Questão 1: Modele este problema como um problema de otimização, identificando:

- as variáveis de decisão;
- a função objetivo;
- as restrições envolvidas, incluindo as de integralidade, se houver.

Resposta:

- **Variáveis de decisão:** x_i, que é o número de cortes de tamanho i.
- **Função objetivo:** variaveis de decisão ponderando o lucro do corte. (maximizar lucro total)
- **Restrições envolvidas:** Temos que garantir que as variáveis sejam inteiras, positivas e que o conjunto dos cortes seja factível (todos os cortes devem somar a ser o comprimento da barra)

$$
\begin{aligned}
\max_{x_i} \quad & \sum_{i \in \{1 \ldots n\}} x_i \times l(i) \\
\text{s.t.} \quad & \sum_{i \in \{1 \ldots n\}} i \times x_i = n, \\
& x_i \geq 0, \\
& x_i \in \mathbb{N}.
\end{aligned}
$$

.

Questão 2: Em uma estratégia de busca por soluções de forma enumerativa, podemos enumerar os padrões de corte e avaliar o seu lucro obtido. Pensando que uma barra de tamanho n pode (ou não) ser cortada em n − 1 posições distintas (cada uma espaçada de 1 unidade de medida), quantas soluções devem ser testadas (considere repetições)?

Resposta: Sabendo que temos n-1 escolha binárias para a barra (cortar ou não cortar) o número de casos que precisamos testar é $2^{(n-1)}$ possíveis soluções (complexidade exponencial).

Questão 3: Projete, por indução, um algoritmo recursivo que encontre a solução ótima para este problema. Para tanto, as seguintes perguntas devem ser respondidas pelo seu projeto: - Qual é o caso base para a recursão? - Como construir a solução ótima para a barra de tamanho k se conhecermos as soluções ótimas para as barras de tamanho 1, · · · , k − 1. - Ao final da execução, como reconstruir a solução ótima? Lembre de memorizar a melhor escolha a cada passo!

Resposta:

A ideia é a seguinte: Seja $T(k) \in \mathbb{N}^k$ a solução do nosso problema para a barra de tamanho k, onde $T_i(k)$ representa o número de cortes de tamanho i.

Para o caso base podemos tomar que em uma barra de tamanho 1 a solução é fazer apenas um corte de tamanho 1, então $T_1(1) = 1, \; T_i(1) = 0, \; \forall i \neq 1$. Ou seja $T(1) = \delta[1] = [1, 0 \ldots, 0]$. Também vamos tomar $T(k) = [0,0 …0],   k 0 $

Se soubermos a solução para $\{1, 2, 3 \ldots k-1\}$, para montar a solução ótima podemos fazer da seguinte maneira recursiva

$$T(k) = \arg \max_{i \in \{1,2 \ldots k\}} \left(l(i) + \sum_{j \in \{1,2,\ldots k\}} l(j) \times T_j(k-i) \right) + \delta[i]$$

(Ps: eu tentei usar notação de vetor e ficou meio feio, né?) Eu quis dizer "some 1 no indice que corresponde ao corte ótimo". Acho que no exercício abaixo que eu retrato o problema como um loop ficou mais default a representação.

Assim para encontrar o valor máximo do total de lucro podemos fazer:

$$L_{\max} = \sum_{i \in \{1,2,\ldots k\}} l(i) \times T_i(k)$$

(Ps: No código prefiri implementar com dicts ao invés de listas, mas ainda é a mesma ideia)

[174]:
```
def bar_cut_problem_recursive(n, l_dict):
    max_value = 0
    solution_assignments = {i: 0 for i in range(1,n+1)}
    chosen_assignments = {}
    chosen_cut = 0

    if n == 0:
        return 0, {}

    for idx in range(1,n+1):
        current_value, previous_assigments = bar_cut_problem_recursive(n-idx,␣
↪l_dict)
        current_value = current_value + l_dict[idx]

        if current_value > max_value:
            max_value = current_value
            chosen_cut = idx
            chosen_assignments = previous_assigments

    for key in chosen_assignments.keys():
        solution_assignments[key] = chosen_assignments[key]

    solution_assignments[chosen_cut] = solution_assignments[chosen_cut] + 1

    return max_value, solution_assignments
```

```python
# Exemplo de uso
n = 4
l_dict = {1: 4, 2: 8, 3: 13, 4: 15}
#l_dict = {1: 1, 2: 5, 3: 8, 4: 9}

max_value, x_opt = bar_cut_problem_recursive(n, l_dict)

print("Atribuição ótima de Xi:")
for i in sorted(x_opt.keys()):
    print(f"x_{i} = {x_opt[i]}")

print(f"Lucro Máximo: {max_value}")
```

```
Atribuição ótima de Xi:
x_1 = 1
x_2 = 0
x_3 = 1
x_4 = 0
Lucro Máximo: 17
```

Questão 4: Um problema que pode acontecer em algoritmos recursivos como o projetado acima é o re-cálculo de soluções de subproblemas durante as chamadas recursivas. Isto é, por exemplo, para resolver o problema com n = 4, o problema com n = 2 é resolvido em mais de uma chamada recursiva. Isso acontece com o seu algoritmo? Se isto acontece, o seu algoritmo pode facilmente se aproximar da simples enumeração e testagem de soluções.

Resposta: Sim, isto acontece nesse algoritmo.

Questão 5: Para evitar o recômputo de soluções já calculadas, proponha uma forma ordenada de resolver o problema, partindo de cortes menores e construindo a solução ótima para problemas maiores. Use esta forma de resolução para remover as chamadas recursivas, reescrevendo-o usando laços (loops). Quantas operações são necessárias, nesta versão, para resolver o problema?

Resposta: Reescrevendo apenas com loops sem recursão, podemos utilizar dois vetores $L \in \mathbb{N}^k$ e $C \in \mathbb{N}^k$, onde $L(i)$ e $C(i)$ representam, respectivamente, o lucro máximo e a melhor escolha ótima de próximo corte para uma barra de tamanho i.

A medida que vamos iterando o loop, preenchemos esses valores um por um, com a seguinte regra:

$$L(k) = \max_{j \in \{1,2\ldots k\}} (l(j) + L(k-j))$$

e

$$C(k) = \arg \max_{j \in \{1,2\ldots k\}} (l(j) + L(k-j))$$

Assim conseguimos com apenas dois loops, com $\frac{N(N+1)}{2}$ operações, isto é $\mathcal{O}(n^2)$ de complexidade. Podemos reconstruir a quantidade de cortes de cada tipo fazendo o caminho de cortes ótimos desde a barra inteira até quando não há mais o que cortar.

[172]:
```python
import numpy as np
def bar_cut_with_loop(n, l_dict):
    cut_choice = np.zeros(n+1)
    profit_array = np.zeros(n+1)

    l_array = np.array(list(l_dict.values()))

    for i in range(n + 1):
        max_profit = 0
        for j in range(1,i+1):
            profit_sum = l_array[j-1] + profit_array[i-j]
            if profit_sum > max_profit:
                cut_choice[i] = j
                max_profit = profit_sum

        profit_array[i] = max_profit

    return cut_choice, profit_array

# Exemplo de uso
n = 4
l_dict = {1: 4, 2: 8, 3: 13, 4: 15}
#l_dict = {1: 1, 2: 5, 3: 8, 4: 9}

cut_array, profit_array = bar_cut_with_loop(n, l_dict)

cut_numbers = np.zeros(n)

remaining_rod = n
while remaining_rod > 0 :
    cut_numbers[int(cut_array[remaining_rod]-1)] += 1
    remaining_rod = remaining_rod - int(cut_array[remaining_rod])

print("Atribuição ótima de Xi:")
for idx, value in enumerate(cut_numbers):
    print(f"x_{idx+1} = {int(value)}")

print(f"Lucro Máximo: {int(profit_array[n])}")
```

```
Atribuição ótima de Xi:
x_1 = 1
x_2 = 0
x_3 = 1
x_4 = 0
Lucro Máximo: 17
```

Questão 6: Use o algoritmo projetado para resolver o caso de teste fornecido no moodle.

Resposta:

[173]:
```python
# Exemplo de uso
n = 17
l_list = [5, 26, 40, 59, 66, 73, 75, 76, 85, 92, 102, 105, 105, 114, 125, 131, 140]
l_dict = {idx+1 : value for idx, value in enumerate(l_list)}
#l_dict = {1: 1, 2: 5, 3: 8, 4: 9}

max_value, x_opt = bar_cut_problem_recursive(n, l_dict)

print("\n ALGORITMO RECURSIVO:")
print("Atribuição ótima de Xi:")
print(list(x_opt.values()))

print(f"Lucro Máximo: {max_value}")

cut_array, profit_array = bar_cut_with_loop(n, l_dict)

cut_numbers = np.zeros(n)

remaining_rod = n
while remaining_rod > 0 :
    cut_numbers[int(cut_array[remaining_rod]-1)] += 1
    remaining_rod = remaining_rod - int(cut_array[remaining_rod])

print("\nALGORITMO COM APENAS LOOPS: ")
print("Atribuição ótima de Xi:")
print(cut_numbers)

print(" ")
print(f"Lucro Máximo: {int(profit_array[n])}")
```

```
 ALGORITMO RECURSIVO:
Atribuição ótima de Xi:
[0, 1, 1, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Lucro Máximo: 243

ALGORITMO COM APENAS LOOPS:
Atribuição ótima de Xi:
[0. 1. 1. 3. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

Lucro Máximo: 243
```

[ ]: