Tutorial: Knapsack Problem

1 Solução do Problema

A solução do problema da mochila 0/1 pode ser abordada por meio de *programação dinâmica*. A ideia central é decompor o problema em subproblemas menores, de forma que a solução ótima total seja composta pelas soluções ótimas desses subproblemas.

1.1 Definição do Subproblema

Seja $dp[i][w]$ o valor máximo que pode ser obtido ao considerar os i primeiros itens, com capacidade máxima da mochila igual a w.

1.2 Função de Transição

Para cada item i (com peso p_i e valor v_i), temos duas opções:

- Não incluir o item i: o valor máximo permanece igual ao do subproblema anterior, isto é, $dp[i-1][w]$.
- Incluir o item i: caso o peso p_i caiba na mochila ($p_i \leq w$), o valor máximo será o valor do item v_i somado ao melhor valor possível com a capacidade restante, $dp[i-1][w-p_i]$.

Assim, a função de transição pode ser expressa como:

$$dp[i][w] = \begin{cases} dp[i-1][w], & \text{se } p_i > w \\ \max(dp[i-1][w],\ dp[i-1][w-p_i] + v_i), & \text{caso contrário} \end{cases}$$

1.3 Casos Base

As condições iniciais são:

$$dp[0][w] = 0, \quad \forall w \geq 0$$
$$dp[i][0] = 0, \quad \forall i \geq 0$$

Esses casos representam que, com zero itens ou capacidade zero, o valor máximo obtido é zero.

1.4 Implementação Dinâmica

O algoritmo preenche uma tabela dp de tamanho $(n+1) \times (W+1)$, onde n é o número de itens e W é a capacidade máxima da mochila. Cada célula é calculada a partir das decisões descritas na função de transição.

```
for i in 1..n:
    for w in 1..W:
        if peso[i] <= w:
            dp[i][w] = max(dp[i-1][w], dp[i-1][w - peso[i]] + valor[i])
```

```
else:
    dp[i][w] = dp[i-1][w]
```

O resultado final é dado por $dp[n][W]$, que representa o maior valor possível que pode ser obtido sem ultrapassar a capacidade da mochila.