* **Descrição do Problema e da Solução**

A nossa solução utiliza a técnica de programação dinâmica. Ao processar o input, começamos por criar uma tabela de (comprimento da chapa) linhas e (largura da chapa) colunas com todos os elementos a 0. Depois, por cada peça de dimensões [, ] lida, atribui-se o preço de venda da peça ao elemento da tabela na posição [, ]. Passando agora para o algoritmo em si, a ideia é, quando acabar o algoritmo, termos os valores máximos a obter por uma chapa de dimensões [, ] precisamente na posição [, ] da tabela (a partir daqui vamos chamar a esta tabela de ).

Ao percorrer a tabela, , de cima para baixo, da esquerda para a direita, comparamos o valor de [, ], isto é, o preço da peça de dimensões [,], que é 0 caso não seja pedida, com o máximo entre [, ] + [, - ] (que corresponde ao corte vertical da chapa) e [, ] + [ - , ] (que corresponde ao corte horizontal da chapa). Desta forma, escolhemos para[, ] o maior valor entre entregar a peça ao cliente ou cortá-la.

* **Análise Teórica**

Função recursiva da solução proposta:

(,)

De notar que utilizámos os limites superiores /2+1 e/2+1, dado que o resultado de cortar a chapa [, ] considerando = em [, ] e [, - ] é igual a cortar a mesma chapa considerando *= j -* em[*, j -* ] e [*, j - (j - )*].

Pseudocódigo:

parseDimensions():

read , from input

= create 2 vector of size ( + 1) by (+ 1) initialized with zeros

parsePrice():

read from input

for from 0 to do

read ,, from input

if <= and <= then

[][] =

if j <= and <= then

[][] =

computeMaxValue():

for from 1 to do

for from 1 to do

for from 1 to /2 + 1 do

[][] = max([][], [][] + [ - ][])

for from 1 to /2 + 1 do

[][] = max([][j], [][] + [][ - ])

1. Leitura dos dados de entrada: Simples leitura do input, com um ciclo a depender linearmente de (número de peças). Logo, O().
2. Processamento da instância: Inserir o preço da peça na tabela . Logo, O(1).
3. Aplicação do algoritmo indicado para cálculo da função recursiva: Percorre-se a tabela utilizando dois ciclos *for* apropriados, cujo custo é . Para cada entrada da tabela, são percorridas, em média, /4 linhas e /4 colunas, sendo o custo por entrada /4+/4 = O(+). Logo, o custo total da aplicação do algoritmo é O(y(+)).
4. Apresentar o resultado final: é feito um print do valor que se encontra na última posição da tabela, [][]. Corresponde a uma complexidade O(1).
5. Complexidade global da solução: O() + O(1) + O((+y)) + O(1) = O((+)). Vale notar que pode haver um caso raro em que > (+), mas de forma geral, admitindo que isso não acontece, o custo está na aplicação do algoritmo e não no processamento do input.

* **Avaliação Experimental dos Resultados**

Neste gráfico, apresentamos o tempo de execução do algoritmo em função do tamanho (+) da entrada. Para tal, utilizámos 16 instâncias espaçadas 100 de tamanho entre si.

A graph with blue dots

Description automatically generated

O tempo de execução não é linear nas dimensões da chapa. Assim, para determinar se a previsão pela análise teórica é correta, vamos pôr o eixo dos XX a variar com a quantidade prevista pela análise teórica.

A graph with blue dots

Description automatically generated

Ao mudarmos o eixo dos XX para (, ) = O((+)), vemos que temos uma relação linear com os tempos no eixo dos YY, confirmando que a nossa implementação está de acordo com a análise teórica de O((, )).