# Descrição do Problema e da Solução

Este código implementa uma solução para otimização de corte de uma placa principal em várias peças, visando obter o maior valor possível. Utilizando programação dinâmica, a matriz dp é utlizada para armazenar os valores máximos alcançados até o momento para diferentes subproblemas.

A leitura das dimensões da placa principal e o número de tipos de peças é seguida pela criação da matriz e pela entrada das dimensões e preços de cada tipo de peça. Através de dois loops aninhados, a matriz *dp* é preenchida considerando todas as combinações possíveis de cortes horizontais e verticais, atualizando os valores na matriz para representar o máximo entre o valor atual e a soma dos subproblemas resultantes dos cortes. O resultado, que representa o valor máximo possível, é printed.

# Análise Teórica

Pseudo-código:

Função getMaxPrice(dimx, dimy, dp):

Para cada i de 1 até dimx:

Para cada j de 1 até dimy:

Para cada k de 1 até j:

Se dp[i][j] < dp[i][k] + dp[i][j - k]:

dp[i][j] = dp[i][k] + dp[i][j - k]

Para cada k de 1 até i:

Se dp[i][j] < dp[k][j] + dp[i - k][j]:

dp[i][j] = dp[k][j] + dp[i - k][j]

Retornar dp[dimx][dimy]

Leitura dos Dados de Entrada:

Leitura de dimx, dimy, ntypes: O(1) - operações constantes para ler três valores.

Preenchimento da matriz dp com os preços das peças:

O(ntypes) - loop para cada tipo de peça.

Preenchimento da Matriz dp:

Dois loops aninhados, cada um percorrendo até dimx e dimy: O(dimx \* dimy).

Para cada célula (i, j), dois loops não aninhados para cortes horizontais e verticais: O(j) + O(i) = O(max(dimx, dimy)).

Cálculo e Retorno do Máximo Preço:

A função getMaxPrice tem três loops aninhados: O(dimx \* dimy \* max(dimx, dimy)).

Complexidade Global da Solução:

O(ntypes) + O(dimx \* dimy \* max(dimx, dimy)) = O(n) + O(n\*m\*k) = O(n) + O(n^3) = O(n^3)

# Avaliação Experimental dos Resultados

Foi utilizado um script gensript.sh (que utiliza o gerador fornecido) para gerar 15 instâncias de teste. De seguida, foi modificado o código do projeto no ficheiro p1\_time para *printar* o tempo de execução do programa em cada teste. O script run\_tests.sh foi utilizado para guardar os tempos de execução do projeto com as 15 instâncias de teste num ficheiro results.txt. Estes tempos foram utilizados com os valores das instâncias de teste para gerar o gráfico seguinte:

O tempo de execução não é linear nas dimensões da chapa. Assim, vamos pôr o eixo dos XX a variar com a quantidade prevista pela análise teórica. Sendo que esta é O(n^3), o tempo será posto a variar em função de n^3. Visto que nos testes realizados X e Y são sempre iguais, X^2 \* Y = X \* Y^2 = n^3.

Ao mudarmos o eixo dos XX para X^2 \* Y, 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(n^3).