Tarefa T03 – Ponteiros, arrays e malloc

Seja a multiplicação de matrizes $C = A \times B$, onde os tamanhos de linhas e colunas são: $A(n,m)$, $B(m,w)$ e $C(n,w)$:

$$c_{ij} = \sum_{k=0}^{m-1} a_{ik} b_{kj} \quad onde\ i = 0, \cdots, n-1 \quad e \quad j = 0, \cdots, w-1$$

O algoritmo mais fácil está ilustrado na Fig.1(i) (que envolve um *loop* em i, outro em j e o mais interno em k, isto é: para $i = 0$ e $j = 0$, você calcula $a_{0,0}b_{0,0} + a_{0,1}b_{1,0} + a_{0,2}b_{2,0} = 22$). Mas, considerando que o armazenamento na memória privilegia linhas, o algoritmo da Fig.1 (ii) é mais eficiente. Para escrever esse algoritmo mais eficiente, basta trocar a ordem dos três *loops*.

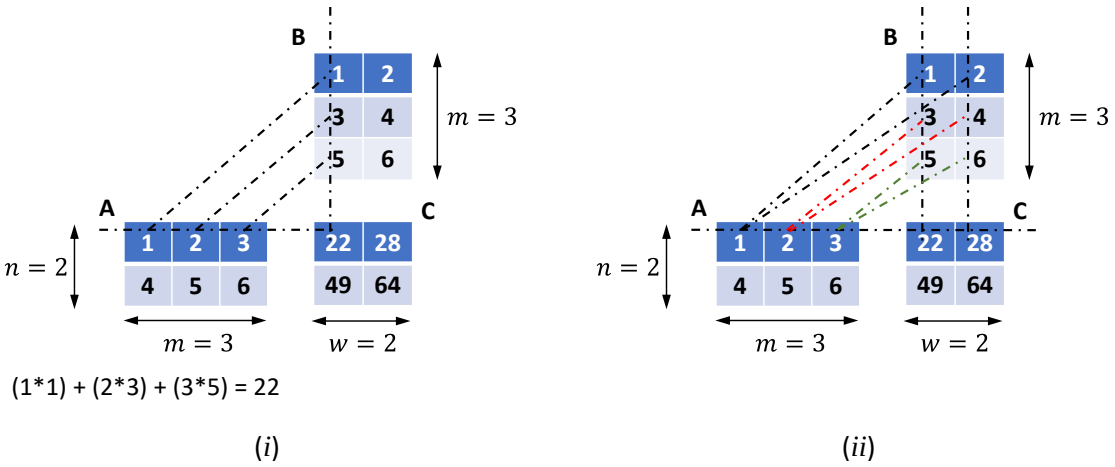

Fig1. Algoritmos de multiplicação de matrizes: (i) por coluna e (ii) por linha.

Faça um programa que onde as matrizes A e B são criadas com *malloc*, definindo uma função que recebe o número de linhas e o número de colunas e retorna um array 2D já inicializado (e.g., unsigned long long ** cria2D(int row, int col)). A matriz C deve ser alocada dinamicamente na *main*. Libere todas as memórias no final.

Um outro desafio é você medir o tempo aproximado de execução usando a função clock() antes e depois da chamada da função que você quer medir (para isto inclua o *time.h*).[1] Teste no modo **Release** (e, para isto, use o comando system("pause") antes do return 0;. Esse comando requer o stdlib.h). Você deve testar com o exemplo da Fig.1 e depois testar com $A(2048,1024)$ e $B(1024,4096)$. Gere matrizes que armazenam números em sequência (1, 2, 3, ..., n*m).

O enunciado acima é suficiente para você escrever o programa. Mas, para lhe ajudar na organização do seu programa, segue um roteiro:

- unsigned long long ** cria2D(int row, int col): cria memoria para matriz 2D e inicializa com a sequência 1, 2, ..., row*col. Retorne NULL se não houver espaço de memória.
- void print2D(unsigned long long **v, int row, int col, int max): imprime matriz 2D
- void mult1(unsigned long long ** a, unsigned long long ** b, unsigned long long ** c, int n, int m, int w): C = AxB com Loop Tradicional percorrendo B por coluna
- void mult2(unsigned long long** a, unsigned long long** b, unsigned long long** c, int n, int m, int w): C = AxB com Loop otimizado percorrendo B por linha
- void libera2D(unsigned long long** v, int row): libera memoria de um array 2D
- na *main*, crie, inicialize e imprima a matriz a e a matriz b, aloque memória para a matriz c com *malloc*, calcule c, imprima o tempo gasto (em milisegundos, segundos e minutos) e imprima apenas uma submatriz de no máximo 5 linhas e 5 colunas da matriz c. Trate adequadamente problemas de memória. Libere memórias no final.

Na *main*, isole as opções de chamada de mult1, mult2, ... com /* e */, deixando apenas uma das opções ativa (para facilitar a compilação e a execução por quem for corrigir a tarefa). Mas você precisa enviar um arquivo pdf com um *print screen* de todos os seus 4 testes, isto é, os dois algoritmos de multiplicação da figura 1 e também com as grandes matrizes. Para ter ideia de tempos aproximados (para as matrizes grandes):

C = AxB CASO 1 - Loop Tradicional percorrendo B por coluna:
27229.000000 milisegundos = 27.229000 segundos = 0.453817 minutos

C = AxB CASO 2 - Loop otimizado percorrendo B por linha:
9089.000000 milisegundos = 9.089000 segundos = 0.151483 minutos

[1] Primeiro defina `clock_t t0, t1; double elapsed;` depois faça `t0 = clock();` *chame a função*; `t1 = clock();`. Depois calcule o seguinte valor *double*: `elapsed = 1000 * ((double)t1 - (double)t0)/CLOCKS_PER_SEC;`. A varável `elapsed` conterá o tempo decorrido em milisegundos.