Devoir maison Algo et complexity

etudiant: Bouzara Zakaria
matricule: 212138069681

---

1. Algorithme iteraif pour calculer le produit de deux matrices:

```
fonction produit_matrices_carres(E: A, B [n][n]entier, S: [n][n]entier){
    pour i=0 ; i<n ; i++ faire:
        pour j=0 ; j<n ; j++ faire:
            pour k=0 ; k<n ; k++ faire:
                C[i][j] += A[i][k] * B[k][i];
            fait;
        fait;
    fait;
    retourner C
fin
```

- **La complexity de cette algorithme:**
  On a 3 boucles `pour` chacune itère de 0 à n (n est la taille de la matrice carrée A), alors **O(n^3)**.

- Pour cet algorithme de multiplication de matrices, la complexité temporelle **O(n^3)** est obtenue dans tous les cas (pire, meilleur et cas moyen). car l'algorithme est fixe et dépend uniquement de la taille des matrices.

2. Modifiant l'algorithme pour diffrent tailles des matrices:

```
fonction produit_matrices_non_carres(E: A [m][n]entier, B [n][p]entier S: [m][p]entier){
    pour i=0 ; i<m ; i++ faire:                    //m <- nbr de lignes de A
        pour j=0 ; j<p ; j++ faire:                //p <- nbr de colomns de B
            pour k=0 ; k<n ; k++ faire:        //n <- nbr de lignes de B
                C[i][j] += A[i][k] * B[k][i];
            fait;
        fait;
    fait;
    retourner C
fin
```

- **La complexity de cette algorithme:**
  on a 3 boucle `pour` qui itère de 0 a differents valeur m, p et m alors **O(n.m.p)**

Algorithme recursive pour calculer le produit

```
fonction produit_recursive(E: A [m][n]entier, B [n][p]entier S: [m][p], i, j, k entier){
    si i >= m:
        retourner C //condition d'arret, en retourne C comme resultat
    fsi;
    si j >= p:
        retourner produit_recursive(A, B, C, i+1, j, 0) //passer au prochaine colomn
    fsi;
    si k >= n:
        retourner produit_recursive(A, B, C, i, j+1, 0) //passer au prochaine ligne
    fsi;

    C[i][j] = A[i][k] * B[k][j]
    retourner produit_recursive(A, B, C, i, j, k+1)
fin
```

- **La complexity de cette algorithme:**

La fonction `produit_recursive` itère de `0` à `m` (condition d'arrêt), incrémentant `i`. À chaque fois, elle s'appelle avec une incrémentation de `j`, qui est responsable d'itérer sur les colonnes de `B`. Dans chaque appel, cette dernière appelle elle-même en incrémentant `k` jusqu'à `k = n`.

Pour chaque élément du résultat, la fonction effectue une série de multiplications et d'additions. Étant donné les trois niveaux de récursion, la complexité temporelle totale de cette fonction est **O(m × n × p)**.

la Multiplications des matrices utilisant la decomposition

Étape 1 : Diviser les Matrices

Tout d'abord, nous divisons chaque matrice en quatre sous-matrices de (**n/2 × n/2**). Par exemple, si nous avons la matrice (A), elle est divisée en :

```
A11 = [1  2]   A12 = [3  4]
      [5  6]         [7  8]

A21 = [9  10]  A22 = [11 12]
      [13 14]        [15 16]
```

Étape 2 : Multiplication Récursive

Ensuite, nous multiplions récursivement les sous-matrices pour calculer les nouvelles sous-matrices du résultat. Nous calculons :

```
C11 = A11 * B11 + A12 * B21
C12 = A11 * B12 + A12 * B22
C21 = A21 * B11 + A22 * B21
C22 = A21 * B12 + A22 * B22
```

Étape 3 : Fusionner les Sous-matrices

Enfin, nous fusionnons les sous-matrices (C11), (C12), (C21), et (C22) pour former la matrice résultat finale.

l'algorithme correspondant:

```
// Étape 1 : Diviser les Matrices
fonction  decompositionMatrice(E: A [n][n]entier
S: [n/2][n/2]eniter, [n/2][n/2]eniter, [n/2][n/2]eniter, [n/2][n/2]eniter)
    A11  = A[:mid][:mid]
    A12  = A[mid:][:mid]
    A21  = A[:mid][mid:]
    A22  = A[mid:][mid:]
    retourner  A11, A12, A21, A22,
fin
```

```
// Étape 2 : Multiplication Récursive
fonction  multiplication(A, B [n][n]entier) ([n][n]entier)
    Si n = 1 Alors
        retourner [[A[0][0] * B[0][0]]]
    Sinon
        A11, A12, A21, A22  = decompositionMatrice(A)
        B11, B12, B21, B22  = decompositionMatrice(B)
        C11  =  ajoutDeuxMatrices(multiplication(A11, B11), multiplication(A12, B21))
        C12  =  ajoutDeuxMatrices(multiplication(A11, B12), multiplication(A12, B22))
        C21  =  ajoutDeuxMatrices(multiplication(A21, B11), multiplication(A22, B21))
        C22  =  ajoutDeuxMatrices(multiplication(A21, B12), multiplication(A22, B22))
        retourner  fusionMatrices(C11, C12, C21, C22)
    FinSi
fin
```

```
// Étape 3 : Fusionner les Sous-matrices

fonction  ajoutDeuxMatrices(A, B [n][n]entier) ([n][n]entier)
    n = longueur(A)
    retourner [[A[i][j] + B[i][j] pour j de 0 à n-1] pour i de 0 à n-1]
fin

fonction  fusionMatrices(C11, C12, C21, C22 [n][n]entier) ([n*2][n*2]entier)
    retourner [C11[i] +  C12[i] pour i de 0 à n-1] + [C21[i] +  C22[i] pour i de 0 à n-1]
fin
```

Complexité de l'algorithme

1. **Division des matrices (SplitMatrix)** :
   Diviser une matrice (n * n) en quatre sous-matrices prend (O(1)) temps, car il s'agit simplement de réorganiser les pointeurs ou les indices.

2. **Multiplication récursive (RecursiveMatrixMult)** :
   Pour multiplier les sous-matrices, nous effectuons 8 multiplications récursives de matrices de taille (**n/2 × n/2**). Chaque multiplication récursive a une complexité **T(n/2)**.

3. **Addition de matrices (AddMatrices)** :
   L'addition de deux matrices de taille (n * n) a une complexité de **O(n^2)**.

Formule de récursion

L'algorithme suit la relation de récurrence suivante pour la multiplication de matrices :

T(n) = 8.T(n/2) + O(n^2)