Université du Québec à Montréal

INF4100

Devoir 3

Par :
Guillaume Lahaie
LAHG04077707

Remis à :
Louise Laforest

Date de remise :
Le 1^{er} avril 2014

Table des matières

1 Numéro 1.

a Concevez une fonction qui prend en entrée deux matrices et qui retourne le resultat de la multiplicationd des matrices

Voir la fonction `multiplierMatrice` du fichier `matrices.py`.

b Implantez l'algorithme naïf

Voir la fonction `trouverParenthesageOptimalNaif` du fichier `matrices.py`.

c Implantez l'algorithme naïf avec stockage des calculs déjà effectués

Voir la fonction `trouverParenthesageOptimalAvecStockage` du fichier `matrices.py`.

d Implantez l'algorithme de programmation dynamique

Voir la fonction `trouverParenthesageOptimalDynamique` du fichier `matrices.py`.

e Comparez les temps d'exécution des trois algorithmes

J'ai d'abord fait un premier test pour des chaines de matrices contenant de 5 à 20 matrices à multiplier.

Le premier graphique présente les resultats pour les trois algorithmes, pour une chaine de matrices variant de 5 à 20 matrices. Le second graphique présente les résultats pour l'algorithme de programmation dynamique et l'algorithme diviser pour régner avec stockage.

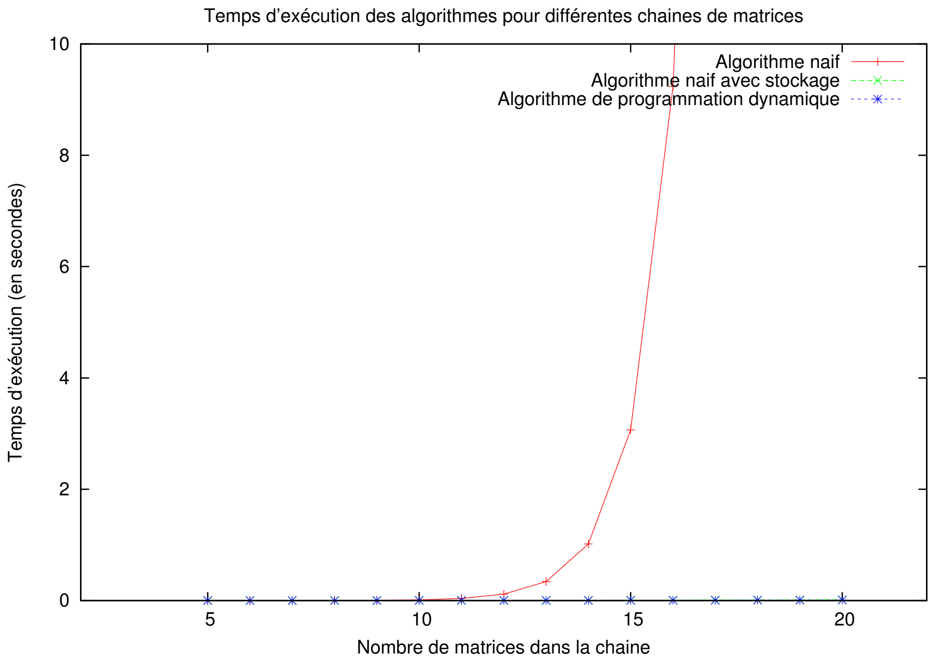
Le premier graphique démontre que pour l'algorithme naïf, le temps d'exécution s'accroit de façon exponentiel, et donc après 20 matrices, le temps d'exécution devient trop long pour représenter sur le graphique.

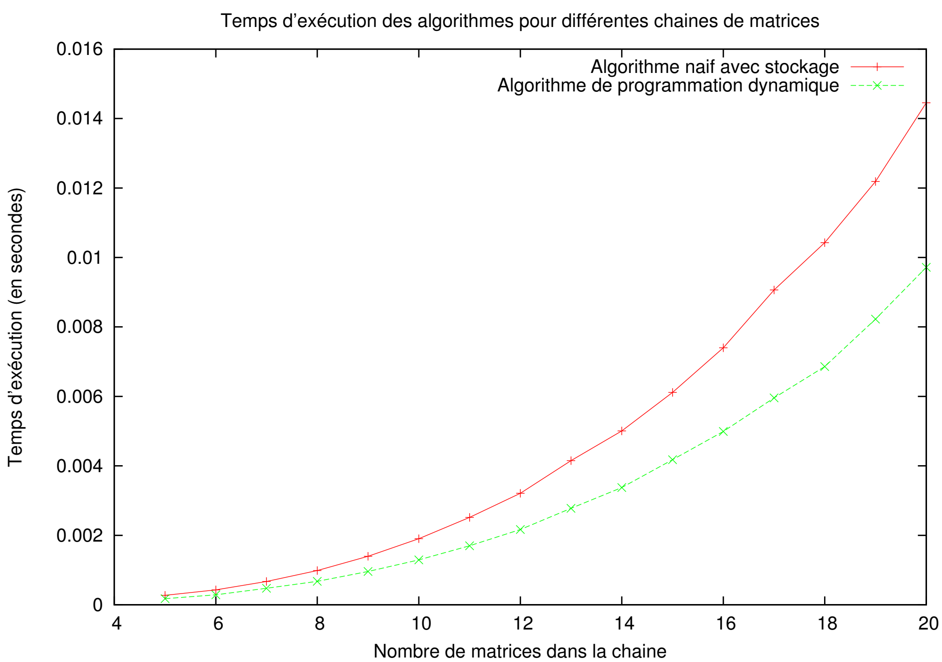
On peut voir ensuite que le temps d'exécution pour l'algorithme diviser pour régner avec stockage et l'algorithme en programmation dynamique croit à un rythme similaire.

L'algorithme naif avec stockage est un peu moins efficace, car il est toujours récursif. De plus, il faut toujours vérifier si le résultat partiel est déjà disponible ou non.

J'ai ensuite créer un script pour tester la performance des algorithmes (`test_performance.py`). Ce script prend en entrée un nombre de matrices dans une chaine, et calcule le parenthésage optimal avec les trois algorithmes. Si les temps d'exécution dépasse 5 secondes, le calcule est annulé.

On peut remarquer ici que plus le même résultat que pour le premier test, le temps d'exécution de l'algorithme naïf croit exponentiellement, alors que les deux autres algorithmes croissent au même rythme.





f Algorithme qui affiche l'expression de parenthésage

Cet algorihtme vient du livre *Introduction to Algorithms, 3rd edition*, de Cormen, Leiserson, Rivest, Stein :

fonction: imprimerParenthesage($frontiere, i, j$)
donnees: *frontiere* : matrice n*n indicé de 1 à n
i : un indice entre 1 et n
j : un indice entre 1 et n
conséquents: Le parenthésage optimal est affiché à l'écran
début
 si $i == j$ **alors**
  imprimer("A"+i)
 fin
 sinon
  imprimer("(")
  imprimerParenthesage($frontiere$, i, $frontiere[i][j]$)
  imprimerParenthesage($frontiere$, $frontiere[i][j] + 1$, j)
  imprimer(")")
 fin
fin

L'algorithme est implémenté dans la fonction `afficherParenthesageOptimal` du fichier `matrices.py`.

g Algorithme qui multiplie la suite de matrices selon la matrice frontiere

g.1 Algorithme

Cet algorithme est basé sur l'algorithme de parenthésage en f.

fonction: multiplierChaineMatrices($matrices, frontiere, i, j$)
donnees: *matrices* : chaine de matrices à multiplier *frontiere* : matrice n*n indicé de 1 à n
i : un indice entre 1 et n
j : un indice entre 1 et n
Sorties: Le résultat de la multiplication de la chaine de matrices
début
 si $i == j$ **alors**
  retourner $matrices[i]$
 fin
 sinon
  $c \longleftarrow$ multiplierChaineMatrices($matrices, frontiere, i, frontiere[i][j]$)
  $d \longleftarrow$ multiplierChaineMatrices($matrices, frontiere, frontiere[i][j] + 1, j$)
  retourner *multiplierMatrice(c, d)*
 fin
fin

Cet algorithme est implémenté par la fonction `multiplierChaineMatrice` du fichier `matrices.py`.

g.2 Analyse de la complexité temporelle de l'algorithme

On connait déjà le nombre de multiplications totales à effectuer pour mutiplier la chaine de matrices. En effet, il s'agit du résultat de l'algorithme de parenthésage utilisé.

On peut donc dire de façon précise le nombre de multiplications scalaires totales qui seront effectuées. Toutefois, cela ne donne pas la complexité temporelle de l'algorithme.

Une autre approche serait de considérer que la multiplication de matrices à comme complexité temporelle $O(n^3)$, où n représente la grandeur maximale d'une des deux matrices à multiplier (par exemple, le nombre de ligne de la matrice A, le nombre de colonnes de B, ou le nombre de colonnes de A).

Si nous avons m matrices dans la chaine, nous aurons donc $m-1$ multiplications de matrices à effectuer. Donc, si chaque multiplication a comme complexité temporelle $O(n^3)$, alors la complexité temporelle de la multiplication de la chaine sera :

$T(n) = O((m-1)n^3) = O(mn^3)$, où m est le nombre de matrices à multiplier, et n la dimension maximale d'une matrice dans la chaine.