141-INF4100-10
Devoir 2

Louise Laforest

Date de remise : Mardi le 1er avril 2014 avant minuit

Le devoir peut être fait en équipe d'au plus deux personnes

DIRECTIVES

1. Vous devez remettre vos fichiers dans une archive `devoir3.zip` via le système **oto** dans la boîte **INF4100** de l'enseignant ℓ**aforest**_ℓ (toutes des lettres minuscules) à l'adresse suivante `http://oto.uqam.ca/application-web/connexion`
2. **Politique concernant les retards** : pénalité de $\frac{m}{144}$ points sur 100 où m est le nombre de minutes de retard (donne 10 points pour 24 heures).

Nous avons vu au cours un algorithme permettant de déterminer l'ordre de multiplication d'une suite de matrices qui minimise le nombre de multiplications scalaires.

(a) Concevez un sous-programme (procédure, fonction, méthode) qui prend en paramètre deux matrices (et leurs dimensions si nécessaire) et qui retourne une troisième matrice (via un paramètre de sortie ou comme résultat retourné par le sous-programme) qui est le résultat de la **multiplication** des deux premières matrices.

(b) Implantez l'**algorithme naïf** (il utilise la méthode « diviser pour régner » et est récursif) qui, étant donné un nombre de matrices ainsi que leurs dimensions, retourne la matrice $frontiere$ ainsi que le nombre minimal de multiplications de nombres (voir le pseudo-code de la fonction trouverParenthésageOptimalRec dans le document `parenthesage.pdf`).

(c) Modifiez l'**algorithme naïf** en faisant un appel récursif seulement lorsque la valeur n'a pas déjà été calculée. Cela suppose que vous stockiez les valeurs calculées. Votre algorithme devra avoir les mêmes paramètres qu'en (b).

(d) Implantez l'**algorithme de programmation dynamique** (fonction trouverParenthésageOptimal dans le document `parenthesage.pdf`) qui, étant donné un nombre de matrices ainsi que leurs dimensions, retourne la matrice m et la matrice $frontiere$.

(e) **Comparez les temps d'exécution** de vos trois algorithmes (b, c, d) (voir plus loin pour des exemples en Java, C++ et Python). Expliquez comment vous avez fait vos comparaisons. Discutez des résultats obtenus. Remarquez que vous n'avez pas à multiplier les matrices.

(f) Concevez un algorithme récursif qui prend en entrée les dimensions des matrices (le vecteur p) ainsi que la matrice $frontiere$ et qui **affiche l'expression**, avec des parenthèses représentant la multiplication optimale des matrices. Avec l'exemple du document `parenthesage.pdf`, votre algorithme devrait afficher quelque chose du genre $((a1(a2a3))a4)$ ou $(a1(a2a3))a4$.

(g) Concevez un algorithme récursif qui prend en entrée la suite de matrices, leurs dimensions si nécessaire ainsi que la matrice $frontiere$ et qui **multiplie la suite de matrices** en utilisant la matrice $fontiere$ (calculée par n'importe lequel des trois algorithmes) qui contient l'information sur le parenthésage donnant le minimum de multiplications de nombres entiers. Donnez la **complexité temporelle** de votre algorithme, avec justifications.

Pour tester votre programme, celui-ci devra lire les données dans un fichier texte, appelé `matrices.txt`, dont le format sera le suivant :
– première ligne : le nombre de matrices,
– deuxième ligne : le vecteur p,
– lignes suivantes : lignes de la première matrice
– lignes suivantes : lignes de la deuxième matrice
– ...
Tous les nombres seront séparés par un espace. **Veuillez respecter ce format.**

Votre programme devra lire ces données, faire les calculs nécessaires et écrire dans un fichier texte nommé `resultats.txt` le contenu de la matrice m, le contenu de la matrice $frontiere$ puis la matrice résultat.

Votre programme devra être écrit en Java, C++, Python ou Maple. Vous devez indiquer avec quel compilateur votre programme a été compilé et sur quelle plateforme. Vous devez me remettre le source et, si possible, l'exécutable.

C'est à vous de me convaincre que votre code fonctionne et non à moi de vous convaincre que votre code ne fonctionne pas, le cas échéant.

Java

```
long tempsAvant;
long tempsApres;
double tempsEcoule;

tempsAvant = System.currentTimeMillis();

// calcul ...

tempsApres = System.currentTimeMillis();
tempsEcoule = tempsApres - tempsAvant;
```

On peut aussi utiliser `System.nanoTime()` au lieu de `System.currentTimeMillis()`. Une nanoseconde est égale à 10^{-9} secondes.

C++

```
#include <time.h>
#include <iostream>
using namespace std;

    clock_t tempsAvant;  // temps avant le calcul
    clock_t tempsApres;  // temps apres le calcul
    double tempsEcoule;  // temps requis pur le calcul

    tempsAvant = clock();

    // calcul ...

    tempsApres = clock();

    tempsEcoule = (double)(tempsApres - tempsAvant) / (double)CLOCKS_PER_SEC;
```

Python

```
from time import time

tempsAvant = time(); // temps en millisecondes

// calcul ...

tempsApres = time();
```