Algorithmique
J.M. Dischler
Professeur d'informatique

Master Biologie Structurale et Bioinformatique

CC3 – 10h - 11h30, lundi 12 Décembre 2016

Implémenter les algorithmes à l'aide de Java. En fin de séance, envoyer le fichier contenant les codes sources par e-mail à l'adresse: dischler@unistra.fr.

Calculer avec des très grands nombres entiers

Dans cet exercice on se propose de faire des calculs avec des nombres entiers très grands. On suppose les entiers positifs. Pour cela on définit une classe *BigNum*. Cette classe modélise un entier sous la forme d'un vecteur où chacun des éléments du vecteur correspond à un chiffre:

```
public class BigNum {

Vector number;

public:
...
}
```

Par exemple, pour modéliser le chiffre 123, on exécute la séquence suivante:

```
number.addElement(new Integer(3));
number.addElement(new Integer(2));
number.addElement(new Integer(1));
```

En effet, l'entier 123 correspond à la suite de chiffres 1, 2 et 3. Le premier élément du vecteur "number" de la classe *BigNum* correspond toujours aux unités, le second aux dizaines, le troisième aux centaines, etc.

1.1 Initialiser un *BigNum* à partir d'un entier.

Ecrire une opération qui permet de créer un *BigNum* à partir d'un entier (de type *int*) passé en paramètre. Par exemple, l'appel:

```
BigNum  n = new BigNum(512);
```

initialisera le vecteur "number" de la classe à la séquence 5, 1 et 2 (le premier élément étant 2, le second 1 et le troisième 5).

Note: utiliser pour cela des divisions successives par 10. En Java, le reste d'une division entière est donnée par l'opérateur % (modulo). Par exemple, 14%4 => 2.

1.2 Afficher un *BigNum*.

Ecrire une opération "afficher" qui permet d'afficher la valeur d'un *BigNum* à l'écran.

1.3 Additionner deux *BigNum*

Ecrire une opération "somme" permettant de faire la somme de deux *BigNum*.
Par exemple les opérations:

```
BigNum a = new BigNum(123);
BigNum b = new BigNum(512);
BigNum c = a.somme(b);
c.affiche();
```

auront pour résultat d'afficher "635". Pour réaliser cette opération, on "posera" simplement l'addition.
Note: le type Java *Integer* peut être convertit en *int* à l'aide de l'opération *intValue()*.

1.4 Multiplication naïve de deux *BigNum*.

Pour multiplier deux *BigNum* nous pouvons également simplement la poser: le premier nombre est successivement multiplié par les chiffres du second. Tous ces résultats sont ensuite additionnés entre eux, sans oublier de faire des décalages. Implanter en Java une telle opération de multiplication. Vérifier sur un exemple.

1.5 Multiplication optimisée de deux *BigNum*.

La technique précédente génère trop de multiplications couteuses en temps de calcul. Nous souhaitons réduire ce nombre en exploitant le principe du « diviser pour régner ».

Pour multiplier deux *BigNum* x et y nous allons nous servir de la propriété suivante:
soit $x = u\,10^n + v$ et $y = s\,10^n + t$, et en posant $a = us$, $b = vt$ et $c = (u+v)(s+t)$,
nous avons $xy = a\,10^{2n} + (c-a-b)10^n + b$.

L'algorithme suivant permet d'appliquer ce principe du:

```
fonction mult (x,y)
   créer u,v,s,t en fonction de n choisi comme m/2, m étant le
   nombre de chiffres du plus grand entiers entre x et y
   a=mult (u,s)
   b=mult (v,t)
   c=mult (u+v, s+t)
   renvoyer   a 10^2n + (c-a-b)10^n + b
```

Transcrire cet algorithme, dit de *Karatsuba*, en Java sur la classe BigNum.