Bryce Fuertes 16/11/2022

Programmation Efficace

Table des matières

Exercice 1 : casser une pierre en deux

La signature algorithmique du Stone.split() :

Stone = Stone * Stone

La méthode java qui implémente cette opération à un effet de bord car cette méthode renvoie une valeur mais modifie aussi la pierre sur laquelle on fait le .split()

Exercice 2 : user une pierre jusqu'à un certain « diamètre »

La classe de complexité de notre code est O log(n) car le temps ne double pas si les données doubles mais à bien un rapport au données présentes.

Si les données double, le temps augmente un peu.

Exercice 3 : fragmenter une pierre jusqu'à un certain diamètre

La classe de complexité de notre méthode est O(n^2) car la pierre sera divisée par 2 puis encore par 2 tant que le diamètre n'est pas atteint.

Exercice 4 : mesure et amélioration des performances

Le temps le plus court obtenue est

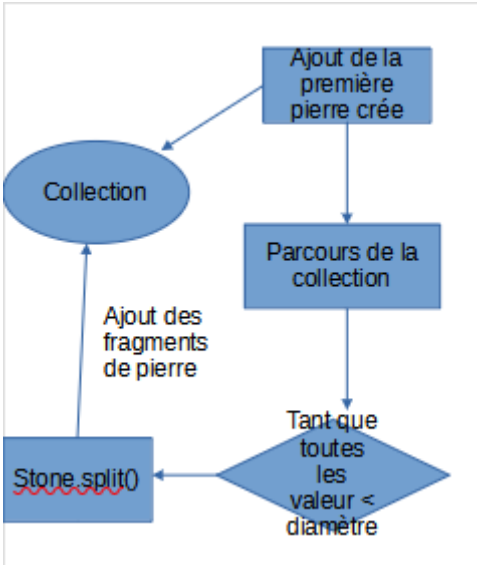
```
Fin du test : 3969148 fragments obtenus en 1228334300 nanoseconds (1228 ms)
```

```java
public Collection<Stone> grind(Stone stone, int diameter) {
    ArrayList<Stone> collection = new ArrayList<Stone>() ;

    collection.add(stone);

    for (int indice = 0 ; indice < collection.size(); indice++) {
        while (collection.get(indice).diameter() > diameter) {
            collection.add(collection.get(indice).split());
            }
        }
    return collection;
}
```

Exercice 5 : description des structures de données employées



Dernier exercice : Exercice A : fragmenter une pierre, en récursif