

Pariel du 21/04/2005 — Durée : 2 heures — Aucun document n'est autorisé
 Le sujet comporte deux parties indépendantes.

1 Tas de sable

On utilise le modèle suivant pour simuler l'éboulement d'un tas de sable : le tas de sable est représenté par un tableau P d'entiers positifs et décroissants, chaque entier correspond à la hauteur de la pile de grains de sable. Par exemple, le tableau $P = [10; 7; 7; 5; 3; 3; 2; 0]$ représente le tas de sable à gauche de la figure 1. Chaque carré représente un grain de sable.

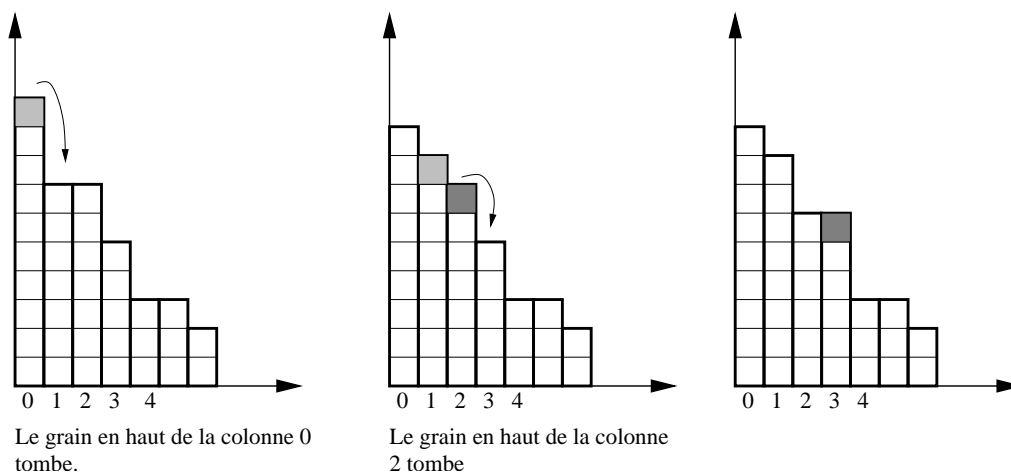


FIG. 1 – Tas de sable

Règle d'éboulement : Si la colonne i dépasse d'au moins 2 la colonne qui la suit, le grain au sommet de la colonne i tombe sur la colonne $i + 1$. Ceci est valable même si la colonne suivante est à zéro. (Par contre, on suppose qu'il y a un mur à la fin du tableau : on ne sort donc pas du tableau.)

Exercice 1 Écrire une méthode `static void unEboulement(int[] tas)` qui cherche le premier éboulement possible et transforme le tableau `tas` en conséquence. S'il n'y a pas d'éboulement possible, on ne change rien.

Exercice 2 Écrire une méthode `static boolean eboulementpossible(int[] tas)` qui retourne `true` si un éboulement est possible et `false` sinon.

Exercice 3 Écrire une méthode `static void dessin(int[] tas)` qui dessine le tas en mode texte, par exemple le premier tas sera dessiné :

```
*
*
*
* * *
* * *
* * * *
* * * *
* * * * *
* * * * *
* * * * *
```

Exercice 4 Écrire une méthode `static void eboulement(int[] tas)` qui éboule le tas jusqu'à ce qu'aucun éboulement ne soit plus possible.

2 Manipulation sur les mots

On souhaite représenter et manipuler les mots, non comme des données de type `String`, mais comme des listes simplement chaînées (de caractères) telles qu'elles ont été introduites en cours. La structure de base de la classe sera donc la suivante :

```
public class Mot{
    public char lettre;
    public Mot suivant;
}
```

On rappelle que la concaténation de deux mots u et v , que l'on note $u \cdot v$, est le mot constitué du mot u puis du mot v : par exemple, la concaténation des mots *bar* et *bu* est le mot *barbu*. Le mot vide est noté ε .

Exercice 5 Écrire une fonction `static Mot insere(char c, Mot u)` qui renvoie le mot obtenu en insérant le caractère c en tête du mot u , et une fonction `static boolean estVide(Mot u)` qui teste si le mot u est vide.

Exercice 6 Écrire une fonction `static Mot append(Mot u, Mot v)` qui renvoie la concaténation des mots u et v .

Exercice 7 Écrire une fonction `static int occurrences(char c, Mot u)` qui retourne le nombre de fois qu'un caractère c apparaît dans le mot v .

Exercice 8 Un mot u est *préfixe* d'un mot v si v s'écrit $u \cdot w$ pour un certain mot w . Écrire une fonction *récursive* `static boolean prefixe(Mot u, Mot v)` qui teste si le mot u est préfixe du mot v .

Exercice 9 Un mot u est *facteur* d'un mot v si u apparaît à l'intérieur de v : v s'écrit $w_1 \cdot u \cdot w_2$ pour certains mots w_1 et w_2 . Par exemple, *tatou* est un facteur de *ratatouille*.

Pour tester si un mot u est facteur d'un mot v , on propose la méthode récursive suivante : si $u = \varepsilon$ ou $v = \varepsilon$, la réponse est triviale ; sinon, on a $u = au'$ et $v = bv'$, où a et b sont des caractères. Dans ce cas, si $a = b$ et si u' est *préfixe* de v' alors u est facteur de v , sinon il faut regarder si u est facteur de v' .

Écrire une fonction `static boolean facteur(Mot u, Mot v)` qui implémente cette méthode.