Projet 1 d'algorithmique

Auteurs

- Jocelyn Caron
- François Soulié

Table des matières

Exercice 1

Fonction qselRecursif

On cherche à écrire une fonction qui permettrait de calculer la p-ème valeur d'un tableau `T` de taille n et d'ordre quelconque. On pense tout d'abord à trier le tableau puis à chercher la valeur de `T[p]`, cependant il est possible de réduire le temps d'éxecution en cherchant à ne s'intéresser qu'a un côté du tableau à chaque segmentation.

En effet, on pose:

- `m = j-i` la taille du sous-tableau `T[i:j]`
- k le résultat d'une segmentation du tableau T vérifiant `T[i:k] <= T[k:k+1] <= T[k+1:j]`
- `0 <= p < m` Pour commencer, si `m=1` `p` est forcément égal à 0, donc la p-ème valeur de `T[i:j]` est `T[i]`

Ensuite:

- Si `i <= p+i < k` alors la p-ème valeur de `T` est la p-ème valeur de `T[i:k]`
- Si `p+i == k` alors la p-ème valeur de `T` est `T[k]`
- Si `k+1 <= p+i < j` alors la p-ème valeur de `T` se trouve dans le sous-tableau `T[k+1:j]` et c'est sa `(p+i)-(k+1)`-ème valeur puisqu'on prend en compte la borne de gauche

```java
static int qselRecursif(int p, int[] T, int i, int j) {
  int m = j - i;
  if (m == 1) {
    // Si m =1, p est nécéssairement égal à 0. La p-ème valeur de
    // T[i:j] est T[i]
    return T[i];
  } else {
    // Si m > 1, p est nécéssairement dans l'intervalle [0:m]
    // On cherche une partition de T[i:j] pour comparer p+i
    // à son résultat
    int k = segmenter(T, i, j);
    int ppi = p + i;
    if (i <= ppi && ppi < k) {
      // Si i <= p+i < k la p-ème valeur de T est la p-ème valeur
      // de T[i:k]
      return qselRecursif(ppi - i, T, i, k);
    } else if (k <= ppi && ppi < k + 1) {
      // Si p+i == k la p-ème valeur de T est T[k]
      return T[k];
    } else {
      // Si k <= p+i < j la p-ème valeur de T est la
      // (p+i-(k+1))-ème valeur de T[i:k]
      return qselRecursif(ppi - (k + 1), T, k + 1, j);
    }
  }
}
```

Fonction qselIteratif

Pour la fonction équivalente itérative, on utilise simplement un `pprime` pour imiter le changement de paramètre p

```
static int qselIteratif(int p, int[] T) {
  int n = T.length, pprime = p, i = 0, j = n, k, pppi;
  while (!(i == j)) {
    // On cherche une partition de T[i:j] pour comparer p+i à son résultat
    k = segmenter(T, i, j);
    pppi = pprime + i;
    if (i <= pppi && pppi < k) {
      // Si i <= p+i < k la p-ème valeur de T est la p-ème valeur de T[i:k]
      j = k;
      pprime = p - i;
    } else if (k <= pppi && pppi < k + 1) {
      // Si p+i == k la p-ème valeur de T est T[k]
      return T[k];
    } else {
      // Si k <= p+i < j la p-ème valeur de T est la
      // (p+i-(k+1))-ème valeur de T[i:k]
      i = k + 1;
      pprime = p - i;
    }
  }
  return T[pprime];
}
```

Exercice 2

Fonctions sac, sacGloutonParValeur et sacGloutonParDensitesDeValeur

La fonction `sac` est une abstraction du comportement de `sacGloutonParValeur` et `sacGloutonParDensitesDeValeur` qui ont exactement le même fonctionnement mais une premìere étape différente. Comme leurs noms l'indiquent, l'un commence par trier le tableau par valeurs et l'autre par densités de valeurs.

```
static boolean[] sac(Objet[] objets, int c){
  // ...
}

static boolean[] sacGloutonParValeurs(Objet[] objets, int c) {
  qspvd(objets); // tri quicksort des objets par valeurs décroissantes
  return sac(objets, c); // sac glouton par valeurs.
}

static boolean[] sacGloutonParDensites(Objet[] objets, int c) {
  qspdd(objets); // tri quicksort des objets par densités décroissantes
  return sac(objets, c); // sac glouton par densités décroissantes
}
```

L'idée de `sac` est de construire un tableau booléen construit en parallèle au tableau d'objets `T` initial tel que si l'objet `T[i]` rentre dans le sac glouton (par valeur ou densité) alors `sac[i]` est `true`. Pour réaliser cette subtilité, on utilise l'attribut `i` des objets de `T` qui nous permettent de retrouver l'ordre initial du tableau même après son tri.

```
static boolean[] sac(Objet[] objets, int c) {
  // Objets triés par valeurs décroissantes ou par densités décroissantes.
  // Retourne un sac glouton selon le critère du tri.
  int n = objets.length, r = c;
  boolean[] sac = new boolean[n];
  // Sac est un tableau parallèle au tableau objets d'ordre initial
  //tel que si T[i] rentre dans le sac alors sac[i] = true

  // Pour chaque objet, si sa taille est inferieur à l'espace
  // restant, on assigne TRUE à l'indice initial de ce dernier
  // dans le tableau sac
  for (Objet obj : objets) {
    int taille = obj.t;
    if (taille <= r) {
      sac[obj.i] = true;
      r -= taille;
    }
  }
  return sac;
}
```

Fonction valeurDuSac

La fonction `valeurDuSac` utilise la fonction précédente `sac` pour savoir quel objet est présent dans le sac glouton et si il l'est, ajouter sa valeur à la valeur du sac.

```
static int valeurDuSac(boolean[] sac, Objet[] objets) {
  int vds = 0; // valeur du sac

  // Pour chaque objet dans le tableau objets, si ce dernier
  // rentre dans le sac, c'est-à-dire que la valeur de sac à son
  // indice initial est TRUE, alors on ajoute sa valeur à la valeur du sac
  for (Objet obj : objets) {
    if (sac[obj.i]) {
      vds += obj.v;
    }
  }
  return vds;
}
```

Fonction objetsAleatoires

On initialise simplement un les n objets d'un tableau `T` en utilisant la bibliothèque `java.util.Random` pour générer des valeurs aléatoires.

```
static Objet[] ObjetsAleatoires(int n, int vsup, int tsup) {
  Objet[] E = new Objet[n]; // ensemble de n objets
  for (int k = 0; k < n; k++) {
    // On assigne indice à la variable
    int i = k;
    // On assigne à v une valeur aléatoire dans [0:vsup+1]
    int v = random.nextInt(vsup + 1);
    // On assigne à t une taille aléatoire dans [1:tsup+1]
    int t = 1 + random.nextInt(tsup + 1);
    // On calcule la densité aléatoire par rapport aux t et v précédents
    float d = v / t;
    // On initialise l'objet aléatoire avec les propriétés précédentes
    E[k] = new Objet(i, v, t, d);
  }
  return E;
}
```

Résultats

Projet.java

```
medianes récursive et iterative et moyenne des valeurs des sacs :
Sacs glouton par valeurs : 1475, 1475, 1335
Sacs glouton par densité : 1641, 1641, 1518
Valeurs des sacs "gloutons par valeurs" dans le fichier valeursSGV.csv
Valeurs des sacs "gloutons par densités" dans le fichier valeursSGD.csv
```

```
mÃ©diane(valeursSGD) = 1641
moyenne(valeursSGD) = 1519

mÃ©diane(valeursSGV) = 1475
moyenne(valeursSGV) = 1335
```

Histogramme