Esiee-Paris - unité d'algorithmique - Feuille d'exercices numéro 1

Décembre 2020 – R. Natowicz, I. Alame, A. Çela, D. Courivaud, D. Garrigue

Une *séquence* de TD est une séance "papier" de deux heures suivie d'une séance "machine" de deux heures. Il y a 8 séquences dans l'unité.

Rappel des notations. L'intervalle $[i:j]$ est constitué des entiers allant de i inclus à j exclus, c'est-à-dire $[i:j] = \{i, i+1, ..., j-1\}$. Cet intervalle d'entiers contient les $j-i$ valeurs entières supérieures ou égale à i. On dit que cet intervalle est de *longueur* $j-i$. En particulier l'intervalle $[i:i]$ de longueur $i-i=0$ est vide ($[\,] = \emptyset$.) Par extension, tout intervalle $[i:j]$, $j-i \leq 0$, est vide.

Soit $T = [t_0, t_1, ..., t_{n-1}]$ un tableau de n valeurs. La notation $T[0:n]$ désigne donc ce tableau T.
De façon générale $T[i:j] = [t_i, ..., t_{j-1}]$. Ce sous-tableau contient les $j-i$ valeurs de T d'indices supérieurs ou égaux à i. Le sous-tableau $T[i:j]$ est de longueur $j-i$. Si $j-i$ est inférieur ou égal à zéro le sous-tableau est vide ($T[i:j] = [\,]$.)
Exemple : avec $T = [8, 6, 2, 7, 1, 2, 4, 8, 6]$ et $i = 3$ et $j = 7$ on a $[i:j] = \{3, 4, 5, 6\}$ et $T[i:j] = [t_3, t_4, t_5, t_6] = [7, 1, 2, 4]$.

Préfixes du tableau $T[0:n]$: le sous-tableau $T[0:k]$ est le k-*préfixe* du tableau T. Le 0-préfixe est vide. Le n-préfixe est le tableau T.

Suffixes du tableau $T[0:n]$: le sous-tableau $T[k:n]$ est le k-*suffixe* du tableau T. Le 0-suffixe est le tableau T. Le n-suffixe est vide.

Exercices.

1. **Premier plus long sous-tableau constant.** Écrire un programme qui calcule les indices d et f du premier plus long sous-tableau constant du tableau d'entiers $T[0:m]$. On demande un temps de calcul linéaire en fonction de la taille m du tableau T : $\alpha \times m + \beta$.

   Donner la propriété $I(...)$ sous-jacente à votre méthode, son initialisation, sa condition d'arrêt, ses conditions de progression.

   Écrire la fonction `int[] pplstc(int[] T)` qui retourne les indices d et f dans un tableau de deux cases : `return new int[] {d,f}`. Cette fonction sera commentée par la propriété $I(...)$ que vous avez choisie.

2. **Segmentation en trois parties.** Écrire une fonction `segmenter(int[] T, int i, int j)` qui calcule une permutation des valeurs du sous-tableau $T[i:j]$ vérifiant $T[i:k1] < T[k1:k2] < T[k2:j]$, où $T[k1:k2]$ est un sous-tableau constant.

   En déduire une nouvelle version du tri rapide, encore plus rapide que la version vue en cours dans les situations où le tableau T contient beaucoup de répétitions de valeurs.
   Exemple : sur mon ordinateur portable qui date de 2012, il ne faut que 51 ms pour trier un tableau de $n = 2^{20} \approx 10^6$ valeurs choisies au hasard dans l'intervalle $[0:10]$ (chaque valeur est en moyenne répétée 100 000 fois.)

3. **Intersection.** Les tableaux $A[0:n_a]$ et $B[0:n_b]$ sont tous deux <u>strictement</u> croissants. On demande d'écrire une fonction `int[] inter(int[] A, int[] B)` qui retourne un tableau strictement croissant contenant l'intersection des valeurs de A et B. On donne l'invariant de la méthode :

   $$I(k, k_a, k_b): \quad A \cap B = C[0:k] \cup (A[k_a:n_a] \cap B[k_b:n_b])$$

   Avec cette propriété, l'intersection $A \cap B$ que nous voulons calculer se décompose en deux parties :

   (a) ce qui a déjà été calculé : ce résultat partiel est le k-préfixe de C ;
   (b) ce qui reste à calculer : c'est l'intersection des k_a et k_b suffixes de A et B.

   Le "*reste à calculer*" $A[k_a:n_a] \cap B[k_b:n_b]$ est de la même forme que $A \cap B$ puisque tout tableau est égal à son 0-suffixe.

   $$I(k, k_a, k_b): \quad A[\underline{\mathbf{0}}:n_a] \cap B[\underline{\mathbf{0}}:n_b] = C[0:k] \cup \left(A[\underline{\mathbf{k_a}}:n_a] \cap B[\underline{\mathbf{k_b}}:n_b]\right)$$

   La propriété $I(k, k_a, k_b)$ nous dit : " l'intersection des 0-suffixes de A et B est l'union du k préfixe de C et de l'intersection des k_a et k_b préfixes de A et B. "

   – <u>Condition d'arrêt</u> : $A[k_a:n_a] \cap B[k_b:n_b] = \emptyset$. On a alors $A \cap B = C[0:k] \cup \emptyset$. Donc $A \cap B = C[0:k]$.

   – <u>Initialisation</u> : rien n'a encore été calculé, $C[0:k] = \emptyset$, $A[k_a:n_a] \cap B[k_b:n_b] = A \cap B$.

   – <u>Progression</u> : à vous de jouer...

   Votre fonction `int[] inter(int[] A, int[] B)` calculera l'intersection dans un tableau $C[0:m]$, $m = \min\{n_a, n_b\}$, puis retournera le k-préfixe de C : `return Arrays.copyOfRange(C,0,k)`.

   Dans le pire des cas, le temps de calcul sera de la forme $\alpha \times (n_a + n_b) + \beta$, linéaire en fonction de la taille $n_a + n_b$ du problème.

   Remarque : le premier algorithme auquel on pense recherche chaque valeur de A dans l'ensemble B. Si les recherches sont séquentielles, le temps de calcul dans le pire des cas est de la forme $\alpha \times (n_a \times n_b) + \beta$. Si ces recherches sont dichotomiques (à voir dans un prochain cours) le temps de calcul est de la forme $\alpha \times n_a \times \log_2(n_b) + \beta$.
   Prenons $n_a = n_b = 2^{20} \approx 10^6$: les recherches séquentielles auront un temps de calcul dominé par le terme $\alpha \times n_a \times n_b = 2^{40}$, celui des recherches dichotomiques sera dominé par $\alpha \times n_a \times \log_2(n_b) = \alpha \times 2^{20} \times 20$, et celui de votre fonction sera dominé par $\alpha \times 2^{20}$. Votre fonction sera plus rapide. Respectivement : un million et 20 fois plus rapide.

4. **Premier sous-tableau de somme maximum.** Le tableau d'entiers $T[0:n]$ contient des valeurs entières quelconques : positives, négatives, nulles. Écrire une fonction `int [] pstsm(int[] T)` qui calcule les indices d et f de début et de fin du premier sous-tableau non vide de somme maximum, et la somme s des valeurs de ce sous-tableau. Ces trois valeurs seront retournées dans un tableau de trois cases : `return new int[] {d,f,s}`. On demande un calcul en temps linéaire : $\alpha \times n + \beta$. Et, ça va de soi, on demande la propriété $I(...)$, son initialisation, sa condition d'arrêt, ses conditions de progression. La fonction `int [] pstsm(int[] T)` sera commentée par la propriété $I(...)$.

   Exemple : avec $T = [-1, 2, 1, -4, 3, 4, -6, 2, 3, 2, -3, 1]$, votre fonction retournera $[d, f, s] = [4, 10, 8]$. (le premier sous-tableau de somme maximum est $T[4:10] = [3, 4, -6, 2, 3, 2]$, il est de somme $s = 8$.)

   Remarque : le premier algorithme auquel on pense consiste à examiner tous les sous-tableaux $T[i:j]$, $0 \leq i < j < n$; à calculer pour chacun d'eux la somme de ses valeurs ; et à retourner le premier sous-tableau de somme maximum. Le temps de calcul de la fonction correspondante est de la forme $\alpha \times n^3 + \beta \times n^2 + \gamma n + \delta$. Le terme dominant de ce calcul est $\alpha \times n^3$. Le terme dominant de votre fonction est $\alpha \times n$. Prenons $n = 2^{20} \approx 10^6$. Votre fonction sera $2^{40} = 10^{12}$ fois plus rapide[1].

---

[1]Dans les faits, la fonction en $\alpha \times n^3$ n'ira jamais au bout de son calcul. Vous l'arrêterez après quelques heures, lassée, lassé, d'attendre le résultat.

Exécution des programmes ci-dessus par le programme main() de la page suivante.

```
Premier plus long sous-tableau constant
T = [ 1,2,1,1,3,3,3,4,4,4 ]
T[d:f] pplstc, avec d,f = [ 4,7 ]

Tri rapide avec segmentation en 3 parties
Tableau T avant le tri
[ 3,3,3,4,0,1,4,3,4,0,0,2,3,0,0,4,3,0,1,1,0,2,0,1,2,0,3,0,2,1,1,3 ]
Tableau T après le tri
[ 0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,2,2,2,2,3,3,3,3,3,3,3,3,4,4,4,4 ]
Temps de calcul avec n = 10**20 et 10 valeurs différentes : 51ms

Intersection de deux ensembles
A = [ 0,1,4,6,7,8,9,11 ]
B = [ 1,2,3,4,5,7,10 ]
A inter B = [ 1,4,7 ]

Premier sous-tableau de somme maximum
[ -1,2,1,-4,3,4,-6,2,3,2,-3,1 ]
T[d:f] de somme s, maximum : d,f,s = [ 4,10,8 ]
```

```java
import java.util.Random;
import java.util.Arrays;
public class TD1 {
   static int[] pplstc(int[] T){ int n = T.length;
      ...
      return new int[] {d,f};
   }

   static int[] stp(int[] T, int i, int j ){
   // segmentation en trois parties T[i:k1]<T[k1:k2]<T[k2:j]
      ...
      return new int[]{k1,k2};
   }
   static void permuter(int[] T, int i, int j){int x = T[i];
      T[i] = T[j];
      T[j] = x;
   }
   static void qs(int[] T, int i, int j){
      ...
   }

   static int[] inter(int[] A, int[] B){
      ...
      return Arrays.copyOfRange(C,0,k);
   }

   static int min(int x, int y){ if (x <= y) return x; return y;}

   static int[] pstsm(int[] T){ int n = T.length;
     ...
     return new int[] {d,f,s};
   }

   static void afficher(int[] T){ int n = T.length;
        System.out.print("[ ");
        for (int i = 0; i < n-1 ; i++) System.out.print(T[i]+",");
        System.out.println(T[n-1] + " ]");
    }

   public static void main(String[] Args){
      {  System.out.println("Premier plus long sous-tableau constant");
         int[] T = {1,2,1,1,3,3,3,4,4,4};
         System.out.print("T = "); afficher(T);
         System.out.print("T[d:f] pplstc, avec d,f = ");
         afficher(pplstc(T));
         System.out.println();
      }
      {  System.out.println("Tri rapide avec segmentation en 3 parties");
      Random r = new Random();
        int n = (int)Math.pow(2,5);
        int[] T = new int[n];
        for (int i = 0; i<n; i++) T[i] = r.nextInt(3);
        System.out.println("Tableau T avant le tri");
        afficher(T);
        qs(T,0,n);
        System.out.println("Tableau T après le tri") ;
        afficher(T);
      }
      { System.out.print("Temps de calcul avec n = 10**20 et 10 valeurs différentes : ");
        Random r = new Random();
        int n = (int)Math.pow(2,20);
        int[] T = new int[n];
        for (int i = 0; i<n; i++) T[i] = r.nextInt(10);
        long avant = System.currentTimeMillis();
        qs(T,0,n);
        long apres = System.currentTimeMillis();
        System.out.println((apres-avant) + "ms");
        System.out.println();
      }
      { System.out.println("Intersection de deux ensembles");
        int[] A = {0, 1, 4, 6, 7, 8, 9, 11},
              B = {1, 2, 3, 4, 5, 7, 10};
        System.out.print("A = "); afficher(A);
        System.out.print("B = "); afficher(B);
        System.out.print("A inter B = "); afficher(inter(A,B));
        System.out.println();
      }
      { System.out.println("Premier sous-tableau de somme maximum");
        int[] T = {-1, 2, 1, -4, 3, 4, -6, 2, 3, 2, -3, 1};
        afficher(T);
        System.out.print("T[d:f] de somme s, maximum : ");
        System.out.print("d,f,s = "); afficher(pstsm(T));
      }
   }
}
```