Esiee-Paris - unité d'algorithmique - Feuille d'exercices numéro 7

Novembre-décembre 2023 – R. Natowicz, I. Alame, A. Çela, X. Hilaire, T. Wu, W. Xu

Rappel des notations. L'intervalle $[i : j]$ est l'ensemble d'entiers positifs $\{i, i+1, ..., j-1\}$. C'est l'ensemble des $j-i$ entiers positifs supérieurs ou égaux à i. Cet intervalle est de *longueur* $j-i$. En particulier l'intervalle $[i : i] = [] = \emptyset$ est de longueur $i - i = 0$. Par extension, tout intervalle $[i : j],\ j - i \leq 0$, est vide.

Soit $T = T[0 : n] = [t_0, t_1, ..., t_{n-1}]$ un tableau de n valeurs. Le sous-tableau $T[i : j] = [t_i, ..., t_{j-1}]$ contient les $j - i$ valeurs de T d'indices supérieurs ou égaux à i. Ce sous-tableau est de longueur $j - i$. Si $j - i$ est inférieur ou égal à zéro le sous-tableau est vide ($T[i : j] = [\,].$).

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

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

Exercice 11. Négatifs, positifs. Le tableau d'entiers $T[0 : n]$ contient des valeurs négatives (< 0) ou positives (≥ 0) en nombres quelconques, éventuellement nuls.
– Écrire une fonction `int np1(int[] T)` qui calcule une permutation des éléments de T dont la sortie vérifie $T[0 : p] < 0$ et $T[p : m] \geq 0$. Elle retourne la valeur d'indice p. Elle sera construite sur l'invariant $I_1(p, q)$: " $T[0 : p] < 0$ et $T[p : q] \geq 0$. " Avec cet invariant le sous-tableau $T[q : n]$ est celui qui reste à traiter.
Le fait que $T[0 : n]$ est une permutation des valeurs $[t_0, ..., t_{n-1}]$ est sous-entendu, ici et pour toutes les fonctions de cette feuille d'exercices.
– Écrire une fonction `int np2(int[] T)` qui fait le même calcul et retourne l'indice p. Elle sera construite sur l'invariant $I_2(p, q)$: " $T[0 : p] < 0$ et $T[q : n] \geq 0$. Avec cet invariant le sous-tableau $T[p : q]$ est celui qui reste à traiter.
Chacune de ces deux fonction sera commentée par son invariant : initialisation, condition d'arrêt, et conditions de progression. Le calcul doit être fait *sur place* c'est-à-dire sans utiliser d'autre tableau. Il doit être en complexité $\Theta(n)$.

Exercice 12. Negatifs, nuls, positifs – Écrire une fonction `int nnp1(int[] T)` qui calcule une permutation des éléments de T vérifiant $T[0 : p] < 0$ et $T[p : q] = 0$ et $T[q : n] > 0$. Elle retourne les valeurs d'indice p et q dans un tableau de longueur 2 par l'instruction `return new int[] p,q`. Cette fonction sera construite sur l'invariant $I_1(p, q, r)$: "$T[0 : p] < 0$ et $T[p : q] = 0$ et $T[q : r] > 0$ ". Le sous-tableau $T[r : n]$ est celui qui reste à traiter.
– Même question pour une fonction `int nnp2(int[] T)` construite sur l'invariant $I_2(p, q, r)$: "$T[0 : p] < 0$ et $T[p : q] = 0$ et $T[r : n] > 0$ ". Le sous-tableau $T[q : r]$ reste à traiter.
Chacune de ces deux fonction sera commentée par son invariant : initialisation, condition d'arrêt, et conditions de progression. Le calcul doit être fait *sur place* c'est-à-dire sans utiliser d'autre tableau. Il doit être en complexité $\Theta(n)$.

Exercice 13. Segmentation en trois parties. Le sous-tableau d'entiers $T[i : j]$, $2 \leq j - i$, contient des valeurs négatives (< 0), nulles ou positives (≥ 0) en nombres quelconques, éventuellement nuls.

– Écrire une fonction `void segmenter1(int[] T, int i, int j)` qui calcule une permutation des valeurs de $T[i : j]$ qui vérifie $T[i : k1] < T[k1 : k2] < T[k2 : j]$, où $T[k1 : k2]$ est un sous-tableau constant. Elle retourne les valeurs d'indice k_1 et k_2.
Invariant : $I1(k1, k2, k)$: $T[i : k1] < T[k1 : k2] < T[k2 : k]$ où $T[k1 : k2]$ est un tableau constant. Le sous-tableau $T[k : j]$ est celui qui reste à traiter.

– Écrire une fonction `void segmenter2(int[] T, int i, int j)` qui fait le même calcul sur l'invariant $I1(k1, k2, k)$: $T[i : k1] < T[k1 : k2] < T[k : j]$. Le sous-tableau $T[k2 : k]$ est celui qui reste à traiter.

Chacune de ces deux fonction sera commentée par son invariant : initialisation, condition d'arrêt, et conditions de progression. Le calcul doit être fait *sur place* c'est-à-dire sans utiliser d'autre tableau. Il doit être en complexité $\Theta(j - i)$.

– 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 il ne faut que 35 ms pour trier un tableau de $n = 10^6$ valeurs choisies au hasard dans l'intervalle $[0 : 10]$ (chaque valeur du tableau est en moyenne répétée 100 000 fois.)