Distance de KENDALL et tri par insertion

Antoine DEQUAY

21 septembre 2022

Notes

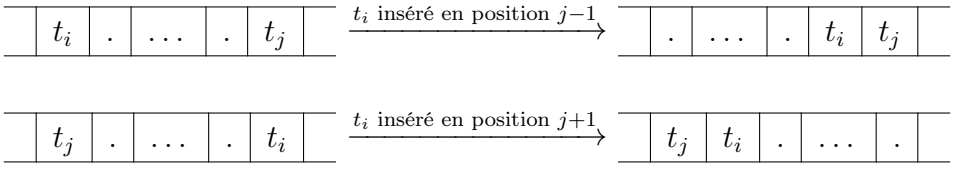
— Prof : .

— Leçon : 903, 927, 931.

— Références :

  — LE BARBENCHON.

On cherche à trier un tableau avec uniquement les opérations suivantes (cas tri par insertion ou sélection) :



On s'autorise à insérer en première et dernière position également...

Proposition 1 Supposons qu'on se donne T un tableau à trier et S une plus longue sous-séquence croissante (PLSC) de T. Alors, on a besoin d'au moins $n - \#S$ insertions pour trier le tableau.

Preuve. En effet, soit T' le tableau obtenu à partir de T par une insertion de t_i en position j et S' une PLSC de T'. Alors :

- Soit t_i n'est pas dans S', et dans ce cas S' est une sous-séquence croissante de T, donc, pas maximalité, $\#S' \leq \#S$,
- Soit t_i est dans S', et dans ce cas $S' \backslash \{t_i\}$ est une sous-séquence croissante de T, donc, pas maximalité, $\#S' - 1 \leq \#S$.

Ainsi, lors d'une insertion, on augmente au plus le cardinal d'une PLSC d'un. Comme un tableau trié s'admet lui-même comme PLSC (qui est de taille n), il faut donc au minimum $n - \#S$ insertions pour obtenir un tableau trié.

□

Proposition 2 En connaissant S une PLSC, on a en fait besoin d'exactement $n-\#S$ insertions pour trier T.

Preuve. En effet, pour trier T, il suffit de parcourir les éléments de $T \backslash S$ et de les insérer à "leur place" dans S de façon à augmenter d'une unité la longueur d'une PLSC du tableau :

Algorithm 1: TRI(T, S)

Entrée: T un tableau et S une PLSC de T.

Sortie : T trié.

```
1  for i ← 0 to n − 1 do
2      if t_i ∉ S then
3          for j ← 0 to #S − 1 do
4              S' ← [];
5              if t_i < s_i et qu'il n'y a pas encore eu d'insertion then
6                  Insérer t_i juste avant s_j dans T;
7                  S' ← S' + [t_i]
8              S' ← S' + [s_j]
9          if Aucune insertion n'a eu lieu then
10             Insérer t_i en dernière position (n − 1) de T;
11             S' ← S' + [t_i]
12         S ← S'
13 return T.
```

Montrons que l'algorithme est correct (il termine bien au vu de sa structure) :

On va montrer qu'à chaque étape de la boucle principale, S est une PLSC de T (invariant de boucle). On reprends les notations introduites S, T, S', T'.

On a différents cas possibles :

— Si $t_i \in S$, ni S ni T ne sont modifiés, donc OK,

— Si $t_i \notin S$ et qu'il n'y a pas d'insertion lors de la seconde boucle, alors t_i est supérieur à tous les éléments de S, donc S est une sous-séquence croissante de $T\backslash\{t_i\}$, et S' une sous-séquence croissante de T'.

Par la propriété 1, comme S est une PLSC de T et que S' est une sous-séquence croissante de T' de taille $\#S+1$ avec T' obtenu après une insertion sur T, on en conclut que S' est une PLSC de T'.

— Si $t_i \notin S$ et qu'il y a eu une insertion au rang j dans la seconde boucle, on a donc :

$$s_0 < \cdots < s_{j-1} < t_i < s_j < \cdots < s_{\#S-1}.$$

Comme précédemment, on a S' sous-séquence croissante de T' de taille $\#S+1$, donc PLSC de T'.

Comme l'invariant "S est une PLSC de T" est vérifié avant l'exécution de la boucle, il l'est à la fin de l'algorithme. Comme chaque élément de $T\backslash S$ est inséré dans S lors de l'exécution

de l'algorithme, à la fin de l'algorithme, S est une PLSC de T de même taille que T, donc T est trié, d'où le résultat !

□

Pour avoir un algorithme effectif, il ne reste plus qu'à savoir comment calculer une PSLC.

Proposition 3 On peut calculer une PLSC par programmation dynamique.

Preuve. On part de la remarque suivante : Tout préfixe d'une PLSC de T est une PLSC d'un préfixe de T. Ainsi, avec M_i la longueur d'une sous-séquence croissante de T maximale finissant pas t_i, on a :

$$\forall i \in [\![1, n-1]\!],\ M_i = 1 + \max\{M_k, k \in [\![0, i-1]\!],\ t_k < t_i\},$$

avec la convention $\max \emptyset = 0$ et la condition initiale $M_0 = 1$.

On a donc accès à l'algorithme (en $O(n^2)$ en temps) :

Algorithm 2: PLSC(T)

Entrée: T un tableau.

Sortie : Une PLSC de T.

```
1  M_0 ← 1;
2  P_0 ← 0;
3  for i ← 0 to n − 1 do
4  |   M_i ← 1;
5  |   P_i ← i;
6  |   for k ← 0 to i − 1 do
7  |   |   if t_k < t_i et M_i < 1 + M_k then
8  |   |   |   M_i ← 1 + M_k;
9  |   |   |   P_i ← k
10 j ← indice du maximum des M_i;
11 S ← [t_j];
12 while P_j ≠ j do
13 |   j ← P_j;
14 |   S ← t_j :: S
15 return S.
```

□