N° d'anonymat:

Année Universitaire 2017-2018
Session 1 de printemps

**Parcours/Étape:** L3 Informatique **Code UE:** 4TIN602U
**Épreuve:** Techniques Algorithmiques et Programmation
**Date:** 26/04/2018 **Heure:** 9h00 **Durée:** 1h30
Documents: une seule feuille A4 recto-verso autorisée.
Épreuve de M. Cyril Gavoille

université de BORDEAUX
**Collège Sciences et Technologie**

Solution. Épreuve sur un total de 56 points.

# Intervalle croissant

Soit $S = (s_0, \ldots, s_{n-1})$ une suite de $n$ valeurs réelles (`double`). On considère le problème PLIVC qui consiste à trouver le Plus Long Intervalle de Valeurs Croissantes de $S$, c'est-à-dire trouver l'intervalle d'indices $[i, j] \subseteq [0, n-1]$ tel que $j - i$ soit maximum et $s_i \leqslant s_{i+1} \leqslant \cdots \leqslant s_j$.

**Question 1.** *Donnez une solution au problème PLIVC pour la suite $S_0 = (2, 4, 3, 5, 7, 1, 8, 6, 10, 9)$, c'est-à-dire l'intervalle d'indices $[i, j]$. Précisez aussi la sous-suite correspondante.*

Solution. [3 pts] $[i, j] = [2, 4]$ et $(3, 5, 7)$.

**Question 2.** *Quelle serait la complexité d'un algorithme basé sur une méthode exhaustive ? Justifiez.*

Solution. [3 pts] A priori $O(n^3)$ car il y a $O(n^2)$ intervalles d'indices $[i, j] \subseteq [0, n-1]$ possibles, et cela prend un temps $O(j - i) = O(n)$ pour vérifier la croissance de chacun de ces intervalles.

**Question 3.** *Trouvez un algorithme de complexité linéaire qui résout PLIVC pour une suite $S$ de longueur $n$. Sans donner le code précis, expliquez le principe.*

Solution. [5 pts] Au début $i = j = 0$. On fait augmenter $j$ tant que les éléments rencontrés sont croissants. Si on trouve un élément décroissant $s_j$ ou si $j = n$, alors on mémorise $i_{\max} = i$ et $j_{\max} = j - 1$ (si l'intervalle $[i, j-1]$ est effectivement plus long que l'intervalle $[i_{\max}, j_{\max}]$ précédent), puis on réinitialise $i = j$ et on continue. L'intervalle recherché est $[i_{\max}, j_{\max}]$ lorsque $j$ atteint $n$.

# Sous-suite croissante

Dans cette partie on considère une variante du problème PLIVC. Dans cette variante, appelée PLSSC, il s'agit trouver la Plus Longue Sous-Suite Croissante d'une suite $S = (s_0, \ldots, s_{n-1})$ de longueur $n$, c'est-à-dire une sous-suite $(s_{i_1}, \cdots, s_{i_k})$ de $S$ de longueur maximum (soit l'entier $k$) telle que $s_{i_1} \leqslant \cdots \leqslant s_{i_k}$ et $0 \leqslant i_1 < \cdots < i_k < n$. Donc valeurs et indices de la sous-suite doivent être croissant, et sa longueur $k$ la plus grande possible. La différence avec le problème PLIVC est que les indices $i_1, \ldots, i_k$ de la solution pour PLSSC ne sont pas forcément consécutifs, et donc ne forment pas nécessairement un intervalle. Pour simplifier, on va supposer que toutes les valeurs de $S$ sont distinctes et $> 0$.

Par exemple, en reconsidérant la suite $S_0$ question 1, $(s_5, s_7, s_8) = (1, 6, 10)$ est une sous-suite croissante de longueur 3. Ce n'est pas la plus grande ! $(s_0, s_1, s_2) = (2, 4, 3)$ n'est pas une sous-suite croissante de $S_0$, de même que $(s_5, s_0, s_7) = (1, 2, 6)$.

**Question 4.** *Trouvez une solution au problème PLSSC pour la suite $S_0$ de longueur 10 de la ques-*

*tion 1*

SOLUTION. [4 pts] On peut faire 6 avec $(2, 3, 5, 7, 8, 10)$. Il y a d'autres solutions.

Pour résoudre PLSSC on se propose, dans un premier temps, d'appliquer l'algorithme consistant à considérer toutes les sous-suites croissantes possibles de $S$ et de n'en retenir que la plus longue. On va aussi se contenter de déterminer la longueur de la plus longue sous-suite croissante plutôt que la sous-suite elle-même.

Pour cela on considère la fonction `int SSC1(int i, double s)` supposée renvoyer la longueur de la plus longue sous-suite croissante de $(s_i, \dots, s_{n-1})$ ayant des valeurs $> s$. La longueur de la PLSSC pour $S$ est donc `SSC1(0,0)` puisque $S = (s_0, \dots, s_{n-1})$ et que tous les $s_i > 0$. (Pour alléger le code, on suppose que `int n;` et `double S[n];` ont été déclarées comme variables globales quelque part dans le programme si bien qu'il n'est pas nécessaire de les passer comme paramètres.)

Par exemple, si $S = S_0$, la suite de la question 1, `SSC1(5,0) = 3` car $(1, 8, 10)$ est une sous-suite croissante de $(s_5, \dots)$ de longueur maximum (il n'y en a pas de plus grande) où toutes les valeurs sont $> 0$. Cependant, `SSC1(5,7) = 2` seulement.

Il s'agit de donner le code de `SSC1()` que l'on peut définir récursivement. Le principe est le suivant. Chaque fois qu'on appelle `SSC1(i,s)` c'est qu'on a déjà produit une certaine sous-suite croissante pour $(s_0, \dots, s_{i-1})$, disons $A$, dont la plus grande valeur est `s`. L'appel à `SSC1(i,s)` est donc chargé de produire toutes les sous-suites croissantes de $S$ dont le début est précisément $A$. (C'est subjectif puisque `SSC1()` ne construit réellement aucune sous-suite, elle calcule juste leur longueur.) Il y a alors essentiellement deux cas, correspondant aux deux futurs possibles de $A$ : soit on ajoute $s_i$ à $A$, soit pas. Dans tous les cas il faudra poursuivre la construction avec `SSC1(i+1,a)` où `a` dépendra du choix d'ajoûter $s_i$ à $A$. Si on l'ajoute `a` $= s_i$, sinon `a` $=$ `s`. Évidemment, si $s_i <$ `s`, l'ajoût de $s_i$ à $A$ ne se pose pas. Bien sûr, l'ajoût produit une sous-suite plus longue d'une valeur, mais d'un autre coté il fait passer le paramètre `s` à une valeur plus grande. La longueur renvoyée par `SSC1(i,s)` doit être le maximum des deux options : l'ajoût de $s_i$ ou pas.

Pour revenir à l'exemple précédent, si l'on veut calculer la longueur `SSC1(5,7)`, associée à la sous-suite $(s_5, \dots) = (1, 8, 6, 10, 9)$, il faut appeller `SSC1(6,7)` car $s_5 = 1 <$ `7` ne peut pas être ajouté. Ensuite, il faut considérer les deux cas : `SSC1(7,8)+1` (on ajoute $s_6$) et `SSC1(7,7)` ($s_6$ n'est pas ajouté), et retenir la plus grande des deux longueurs.

**Question 5.** *Donnez le code* `C` *de la fonction récursive* `int SSC1(int i, double s)`. *(Vous pourrez supposer que la fonction* `max(u,v)` *donnant la maximum des valeurs* `u` *et* `v` *existe déjà.)*

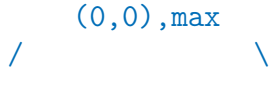
SOLUTION. [5 pts]

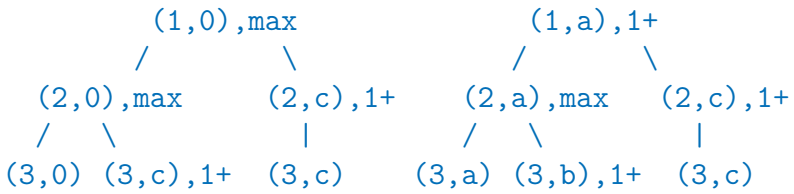
```
int SSC1(int i, double s){
  if(i==n) return 0;
  if(S[i]<s) return SSC1(i+1,s);
  return max( SSC1(i+1,s), 1+SSC1(i+1,S[i]) );
}
```

On rappelle que l'arbre des appels est un arbre dont les noeuds internes correspondent aux appels d'une fonction récursive et les feuilles aux valeurs terminales. (En option, près des noeuds internes, on peut également représenter l'opération à réaliser à partir de ses fils ce qui peut être pratique pour calculer la valeur renvoyée par la fonction qui se lit à la racine.)

**Question 6.** *On se donne trois valeurs* $0 < a < b < c$. *Dessinez l'arbre des appels pour* `SSC1(0,0)` *avec* $S = (a, c, b)$. *(Aide : l'arbre devrait contenir autant de branches et de feuilles que de sous-suites croissantes possibles pour* $(a, c, b)$ *qui sont* $()$, $(a)$, $(b)$, $(c)$, $(a, c)$, $(a, b)$.*)*

SOLUTION. [6 pts]






**Question 7.** *Indiquez la valeur renvoyée par* `SSC1(0,0)` *pour* $S = (a, c, b)$. *Donnez la sous-suite croissante correspondante.*

SOLUTION. [3 pts] Valeur renvoyée : 2, longueur des sous-suites croissantes $(a, b)$ ou $(a, c)$, correspondant aux deux branches DGD ou DD.

**Question 8.** *Donnez le nombre de noeuds internes de l'arbre des appels de* `SSC1(0,0)` *pour une suite* $S$ *de longueur* $n$ *dans le cas où* $s_0 < s_1 < \cdots < s_{n-1}$. *En déduire la complexité de* `SSC1(0,0)` *en fonction de* $n$. *Justifiez.*

SOLUTION. [3 pts] La hauteur de l'arbre est $n$. Il est binaire car les valeurs sont croissantes, on a donc exactement deux fils à chaque fois. Il a donc $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1 = O(2^n)$ noeuds internes.

**Question 9.** *Calculez, pour l'arbre des appels de* `SSC1(0,0)`, *le nombre maximum de noeuds internes* $(i, s)$ *différents pour une suite* $S$ *de longueur* $n$ *? Justifiez.*

SOLUTION. [3 pts] $O(n^2)$ car il y a $n$ valeurs différentes pour $i$ et $n$ valeurs différentes $s_i$.

On désire implémenter plus efficacement `SSC1()`, ce qui est suggéré par les réponses aux questions 8 et 9. Pour cela, on utilise une liste chaînée `L` contenant les différents appels $(i, s)$ rencontrés ainsi que la longueur résultante. On ajoutera par exemple le triplet `(i,s,k)` à `L` si `k=SSC1(i,s)` et si `(i,s,.)` n'est pas déjà présent dans `L`. On note `SSC2()` cette nouvelle implémentation dont le code n'est pas demandé.

**Question 10.** *Donnez la complexité de* `SSC2(0,0)` *pour une suite* $S$ *de longueur* $n$. *(Aide : la longueur maximum pour* `L` *est donnée par la question 9.)*

SOLUTION. [3 pts] C'est $O(n^4)$. En effet, le calcul d'un noeud particulier $(i, s)$ prend un temps constant (dans un cas terminal) ou bien $O(n^2)$ puisqu'il consiste à chercher $(i, s)$ dans une liste de longueur au plus $O(n^2)$. Maintenant, les appels récursifs de la fonction consiste à parcourir l'arbre des appels (selon un parcours en profondeur). Et, l'utilisation de la liste à pour effet de supprimer un sous-arbre (et tous ses noeuds) dès qu'on rencontre un appel déjà rencontré. Au final on ne parcours que des appels différents et leur voisins dans l'arbre, soit $O(n^2)$ puisque que chaque noeud a au plus deux voisins. Le coût total est donc $O(n^2 \times n^2) = O(n^4)$.

**Question 11.** *Quelle aurait été la complexité de* `SSC2(0,0)` *si on avait décidé de stocker les triplets* `(i,s,k)` *dans un arbre de recherche équilibré plutôt qu'une liste chaînée ?*

SOLUTION. [2 pts] Le temps de recherche et d'ajoût aurait été $O(\log(n^2)) = O(\log n)$, l'arbre ayant $O(n^2)$ noeuds. Donc au total on aurait eut une complexité de $O(n^2 \log n)$.

Pour aller plus vite encore, on va considérer la variable $L_S(i)$, définie pour tout indice $i \in [0, n[$, comme étant la longueur de la plus longue sous-suite croissante de $S$ se terminant par $s_i$. Pour résoudre le problème PLSSC pour $S$, il suffira de calculer $\max_{i \in [0,n[} L_S(i)$ puisque la plus longue sous-suite croissante de $S$ se termine nécessairement sur une certaine valeur $s_i$ de la suite $S$.

**Question 12.** *Calculez les valeurs* $L_{S_0}(i)$ *pour la suite* $S_0$ *de la question 1. (Présentez le résultat sous forme d'un tableau comportant trois lignes : une ligne pour les* 10 *indices, une ligne pour les valeurs de* $S_0$ *et une ligne pour les longueurs* $L_{S_0}(i)$.*)*

SOLUTION. [4 pts]

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| $S$ | 2 | 4 | 3 | 5 | 7 | 1 | 8 | 6 | 10 | 9 |
| $L_S$ | 1 | 2 | 2 | 3 | 4 | 1 | 5 | 4 | 6 | 6 |

**Question 13.** *Pour une suite $S$ de longueur $n$, donnez une formule de récurrence entre $L_S(i)$ et les valeurs $L_S(j)$ avec $j < i$. En particulier, déterminez $L_S(0)$.*

SOLUTION. [3 pts] $L_S(i) = 1 + \max_{\{j\in[0,i[:s_j<s_i\}} \{L_S(j)\}$ en posant $\max_{j\in\varnothing} \{L_S(j)\} = 0$ (ce qui revient à poser $L_S(i) = 1$ s'il n'existe pas de $s_j < s_i$ avec $j < i$).

**Question 14.** *Donnez le code de la fonction* `SSC3()`*, dont le début vous est fourni, permettant de calculer la longueur $k$ de PLSSC de $S$ en utilisant la programmation dynamique basée sur la variable $L_S(i)$. (Ici encore* `n` *et* `S[]` *sont supposées être des variables globales.)*

```
int SSC3(void){
  int L[n];
  ...
  return k;
}
```

SOLUTION. [4 pts]

```
int SSC3(void){
  int L[n];
  int i,j,m,k=1;     // k=longueur cherchée

  for(i=0;i<n;i++)
    for(j=m=0;j<i;j++)
      if(S[j]<S[i]) m=max(m,L[j]);
    L[i]=1+m;        // 1 ou 1+max L[j] tq s_j < s_i
    k=max(k,L[i]); // max des L[i]
  }

  return k;
}
```

**Question 15.** *Quelle est, en fonction de la longueur $n$ de $S$, la complexité de votre fonction* `SSC3()` *? Justifiez.*

SOLUTION. [3 pts] $O(n^2)$, chaque terme $L[i]$ prenant un temps $O(i)$.

**Question 16.** *Quelle variable faudrait-il introduire pour pouvoir récupérer la sous-suite solution de* `SSC3()` *? Expliquez comment la calculer.*

SOLUTION. [2 pts] Une variable `J[i]` donnant l'indice $j$ du maximum $\max_j$ dans la formule donnant $L_S(i)$. On peut poser `J[i]=-1` si cet indice n'existe pas.

**FIN.**