

Algorithmique avancée

Examen final, session n° 1

barème indicatif - durée 3h - documents interdits

Exercice 1 : Chemin de longueur minimum (2 points)

On considère un graphe G orienté non valué. Proposer un algorithme qui, étant donné G et deux sommets x et y de G , affiche le plus court chemin partant de x et arrivant en y . Si y est inaccessible depuis x , on affiche "erreur".

Exercice 2 : Chemin de capacité maximum (3 points)

On considère un graphe G non orienté et une fonction de capacité c sur les arêtes du graphe. Selon l'adage «la force d'une chaîne est celle de son maillon le plus faible», la capacité d'un chemin est la plus petite capacité d'une arête de ce chemin. Proposer un algorithme qui, étant donné G et deux sommets x et y de G , calcule et renvoie la capacité du meilleur chemin (=celui de plus grande capacité, parmi tous les chemins possibles) entre x et y .

Exercice 3 : Graphe biparti (2 points)

On considère un graphe connexe, non orienté, dont les sommets ont un champ `couleur` à valeur dans `{noir, blanc, ?}`. Initialement tous les sommets ont `x.couleur = ?`. Écrire une fonction *réursive* `color(sommet x)` qui colore, si possible, chaque sommet de sorte qu'à la fin toutes les arêtes aient une extrémité blanche et une noire.

Problème : la plus longue séquence croissante

Dans ce problème, on considère un tableau T de n nombres réels numérotés de $T[0]$ à $T[n-1]$.

- Une *séquence* de T est obtenue en "effaçant" certaines cases de T (on peut la voir comme un sous-tableau de T).
- Une séquence est un *intervalle* si elle est constituée de cases contiguës de T , sans "trou".
- Une séquence (ou un intervalle) est *croissante* si le sous-tableau correspondant est trié en ordre croissant (au sens large : les répétitions sont admises).
- La *longueur* d'une séquence (ou d'un intervalle) est son nombre de cases.

Les deux problèmes qui nous intéressent sont de rechercher la plus longue séquence croissante d'un tableau, et son plus long intervalle croissant. NB : s'il y en a plusieurs qui ont la longueur maximum, «le plus long» ou «la plus longue» veut dire «l'un(e) des plus long(ue)s».

Par exemple pour $T =$

2.5	-3	42	-3	1	11.17	3.14	10.35
-----	----	----	----	---	-------	------	-------

,
 une séquence de longueur 4 est

-3	-3	11.17	3.14
----	----	-------	------

,
 la plus longue séquence croissante est

-3	-3	1	3.14	10.35
----	----	---	------	-------

,
 et le plus long intervalle croissant est

-3	1	11.17
----	---	-------

.

Les exercices du problème sont indépendants, vous pouvez répondre à une question sans avoir répondu aux précédentes.

Exercice 4 : 1 point

Combien d'intervalles un tableau de n cases contient-il ? Et combien de séquences ?

Exercice 5 : 1 point

Écrire un tableau de n cases tel que

- tous les intervalles croissants ont longueur 1, et
- la plus longue séquence croissante est la plus longue possible.

Donner cette longueur en fonction de n . Ne pas justifier.

Dans la suite du sujet, on suppose que l'on dispose des fonctions

- `int longueur(double[] T)` qui donne la longueur de `T`
- `boolean estTrié(double[] T)` qui dit si `T` est trié en ordre croissant
- `double[] effacer(double[] T, int i)` qui alloue et renvoie un nouveau tableau qui est une copie de `T` sans la case d'indice `i` (il a donc une case de moins que `T`)
- `double[] plusLong(double[] T1, double[] T2)` qui renvoie le plus long des deux tableaux

Exercice 6 : 2 points

Proposer un algorithme qui calcule (et renvoie) le plus long **intervalle** croissant d'un tableau.

On considère le pseudo-code suivant, dont on admettra qu'il calcule bien la plus longue **séquence** croissante de `T`. On suppose que l'appel initial se fait toujours avec `i = 0`.

```
double[] plusLongueSeq(double[] T, int i)
si i == longueur(T) alors
    | si estTrié(T) alors
    | | return T
    | sinon
    | | return null
sinon
    | return plusLong( plusLongueSeq(T, i+1), plusLongueSeq(effacer(T,i), i) )
```

Exercice 7 : 2 points

Dessiner l'arbre des appels récursifs de `plusLongueSeq(

2	1	4
---	---	---

, 0)`

Exercice 8 : 1 point

Donner la complexité d'un appel de `plusLongueSeq(T,0)` en fonction de la longueur `n` de `T`. Justifier.

Exercice 9 : 2 points

Dans quel(s) cas cela ne sert-il à rien de faire l'un des deux appels récursif? En utilisant les principe de *branch and bound*, écrire un code amélioré de `plusLongueSeq`.

Exercice 10 : 2 points

On appelle l_i la *longueur* de la plus longue séquence croissante de `T` dont la *dernière case* est la case numéro `i` ($l_i \geq 1$ car `T[i]` appartient à toute séquence croissante terminant sur la case `i`).

Montrer que $l_i = 1 + \max\{ l_j \mid j < i \text{ et } T[j] \leq T[i] \}$.

Noter qu'il est possible que l'ensemble $\{ l_j \mid j < i \text{ et } T[j] \leq T[i] \}$ soit vide (par exemple pour $i = 0$), dans ce cas `max` retourne 0.

Exercice 11 : 1 point

En utilisant l'équation de l'exercice précédent (il n'est pas nécessaire d'avoir fait l'exercice!) écrire une fonction récursive `int l(int i)` qui calcule l_i (le tableau `T` et sa longueur `n` sont des variables globales).

Exercice 12 : 2 points

Améliorer le programme de l'exercice précédent grâce à la programmation dynamique. Donner la complexité.