Algorithmique et Complexité

Étienne Lozes

formation informatique au collège
ENS Cachan
25 août 2016

Exercice 1 La complexité de se garer

Vous vous trouvez sur un parking ayant n places en tout, et vous voulez vous garer. Combien de temps cela vous prend-il dans le pire cas, asymptotiquement (notation $\mathcal{O}(\dots)$), en fonction de n ?

1. Vous êtes dans une rue, sans indications.
2. Vous êtes au supermarché, sur un parking "carré", et pour chaque rangée, un panneau vous dit combien il reste de places dans la rangée.
3. Vous êtes dans un parking souterrain "cubique", et à l'entrée on vous dit quels sont les niveaux (carrés) qui sont complets.

Exercice 2 Le cadet de mes entiers

Dans une liste d'entiers, tout entier qui n'est pas le plus grand est un cadet. Dans cet exercice, on s'intéresse au problème du calcul d'un cadet (notez qu'il peut y en avoir plusieurs, n'importe lequel répond au problème).

1. Quelle est la donnée du problème (attention, il y a une ou deux subtilités) ?
2. On considère l'algorithme suivant : on commence par chercher le plus grand élément de la liste L, puis on parcours à nouveau la liste et on s'arrête dès que l'on voit un élément différent du plus grand. Quelle est la complexité asymptotique en temps dans le pire cas ? On attend une réponse en notation $\mathcal{O}(\dots)$.
3. Décrivez précisément en une ou deux phrases un algorithme plus efficace.
4. Quelle est sa complexité asymptotique ? La réponse peut dépendre de votre réponse à la première question.

Exercice 3 Bien programmer, c'est couper les cheveux en deux (voire plus).

Dans cet exercice, vous allez *déverminer* (en franglais "debugger") un algorithme en l'exécutant **pas à pas** sur des données bien choisies.

L'algorithme qui contient des erreurs est une recherche par dichotomie.

Algorithme buggé

Données : liste non vide triée d'entiers L de longueur n, entier c
Résultat : position de c dans L si elle existe
début
  $g \leftarrow 1$;
  $d \leftarrow n$;
  tant que $g \neq d$ **faire**
    $m \leftarrow \lfloor \frac{g+d}{2} \rfloor$;
    si $L[m] = c$ **alors**
      retourner m
    sinon si $L[m] < c$ **alors**
      $g \leftarrow m$
    sinon
      $d \leftarrow m$
  si $L[g] = c$ **alors retourner** m;
  sinon retourner *Non trouvé*;

1. Complétez ci-dessous l'exécution pas à pas de cet algorithme sur le jeu de données $L = [2, 4, 6, \ldots, 20]$ et $c = 7$.

itération	g	d	m	L[m]
1	1	10	5	10
2	1	5	3	6
3	...	...	...	...
4	...	...	...	...
5	...	...	...	...

2. Quel est le problème avec cet algorithme ?
3. Corrigez l'algorithme. Indication : c'est possible en changeant seulement la partie si... alors... sinon en ayant l'idée de faire gagner un chouia de temps à l'algorithme à chaque fois (si vous séchez, regardez les transparents).

Exercice 4 Recherche d'un motif dans une image

On considère deux images pixelisées, une "image motif" M et une "image texte" T. On note $M[3][5]$ la couleur du pixel en ligne 3 et colonne 5 dans l'image

M. On note aussi $|M|$ le nombre de colonnes de M et $\overline{\underline{M}}$ le nombre de lignes de M.

On dit que M apparaît dans T en ligne l et colonne c si $M[1][1] = T[l][c]$, et $M[1][2] = T[l][c+1]$, et ... et $M[1][|M|] = T[l][c+|M|-1]$ et... et $M[\overline{\underline{M}}][|M|] = T[l + \overline{\underline{M}} - 1][c + |M| - 1]$.

1. Complétez l'algorithme ci-dessous.

Test d'apparition d'une image motif à une position donnée

Données : image motif M, image texte T, ligne l, colonne c
Résultat : Vrai si M apparait dans T en (l, c), Faux sinon
début
  pour chaque *i allant de 1 à* $\overline{\underline{M}}$ **faire**
    pour chaque *j allant de 1 à ...* **faire**
      si *M[i][j]...* **alors retourner** ...;
  retourner ...

2. En faisant appel à cet algorithme, écrivez un algorithme de recherche d'une image motif dans une image texte.

Exercice 5 Tri sélection en place

L'algorithme de tri par sélection vu durant l'exposé présente un défaut : il faut créer une nouvelle liste R différente de la liste de départ L pour stocker le résultat au fur et à mesure.
Une autre solution est la suivante :

- on trouve tout d'abord le plus grand élément, par exemple 7 dans la liste $L = [2, 7, 3, 1, 4]$,
- on l'échange **dans** L avec l'élement en fin de liste ; on obtient par exemple la liste $L = [2, 4, 3, 1, 7]$
- on cherche à nouveau le plus grand élément, mais cette fois-ci on omet l'élément en fin de la liste, donc sur l'exemple on trouve 4
- on l'échange à nouveau, mais cette fois-ci avec l'avant-dernier en fin de liste, donc $L = [2, 1, 3, 4, 7]$
- on continue ainsi de suite jusqu'à avoir mis à la bonne place tous les éléments.

Formalisez cet algorithme en pseudo-code ou en Scratch.