Aix*Marseille université
Initiative d'excellence
Année 2018-19

Diplôme Inter-Universitaire
Enseigner l'informatique au lycée

Bloc 1 : Séance 7

Complexité des algorithmes

1 Tris par permutation

Un des algorithmes les moins efficaces que l'on puisse imaginer pour trier une suite de n entiers consiste à énumérer toutes les permutations de ces n entiers jusqu'à trouver celle pour laquelle la liste est triée.

Question 1. Décrire un algorithme qui prend en entrée une liste d'entiers et renvoie vrai si elle est triée et faux sinon. Une liste vide sera considérée triée.

Pour énumérer toutes les permutations d'une liste d'entiers, nous allons traiter le problème plus général consistant à compléter une liste partielle R avec toutes les permutations possibles de la liste courante L.

L'appel initial de la procédure se fera avec une liste vide pour R et la liste initiale pour L : `TriParPermutations([], L)`.

L'algorithme est le suivant

```
TriParPermutations(R, L)
début
si L est vide alors
    si R est triée alors
        afficher(R)
sinon
    pour chaque élément e de L,
        soit R' la liste obtenue en ajoutant e à la fin de R
        soit L' la liste obtenue à partir de L en supprimant e
        TriParPermutations(R', L')
fin
```

Question 2. Quel est la complexité en temps de cet algorithme ?

Question 3. Voyez-vous comment en inversant deux tests on peut améliorer l'efficacité pratique de cet algorithme ? Cette inversion modifie-t-elle la complexité théorique de l'algorithme ?

Question 3. Ecrire cet algorithme en Python avec une complexité linéaire en mémoire en utilisant un tableau pour représenter L et un autre pour représenter R.

Question 4. Réécrire la procédure en utilisant un seul tableau dans lequel la liste R est stockée au début et la liste L est stockée à la fin.

2 Complexité du tri par sélection

Etant donné un tableau de n entiers `T[0], T[1], ..., T[n-1]` le principe du tri par sélection est le suivant.

On suppose que la première partie du tableau contient les `k` plus petits éléments triés. Au départ `k=0`. On choisit l'élément minimal parmi ceux qui ne sont pas encore triés (les éléments de rang `k` à `n-1`) et on échange sa position avec celle de l'élément de rang `k`. On incrémente `k` et on recommence tant que `k<n`.

Question 1. Quel est la complexité en temps de cet algorithme ?

Question 2. Quel est la complexité en espace de cet algorithme ?

3 Autres exercices sur la complexité

On considère une suite de $S = (s_1, s_2, \ldots, s_n)$ d'entiers tous distincts, avec $n \geq 2$.

Question 1. Décrivez un algorithme en $\mathcal{O}(n)$ pour trouver deux éléments s_i et s_j de S tels que $s_i - s_j$ soit maximal.

Question 2. Décrivez un algorithme en $\mathcal{O}(n \log n)$ pour trouver deux éléments distincts s_i et s_j de S tels que $|s_i - s_j|$ soit minimal.

Question 3. Soit K un entier arbitraire. Comment déterminer s'il existe deux entiers s_i et s_j dont la somme est égale à K. Quelle est la complexité de votre algorithme ?

Question 4. Même problème que celui de la question 3, mais cette fois-ci, on suppose que la liste est triée. Trouver un algorithme en $\mathcal{O}(n)$.

4 Retour sur l'exponentiation

On souhaite comparer les complexités des deux algorithmes d'exponentiation étudiés en TD 4 :

```
expo
entrée : x un entier strictement positif, y un entier naturel
sortie : x^y
début
    i = 0;
    résultat = 1;
    tant que (i != y) {
        résultat = résultat * x;
        i = i + 1;
    }
    retourner résultat;
fin

expo_rapide
entrée : x un entier strictement positif, y un entier naturel
sortie : x^y
début
    puissance = x;
    résultat = 1;
    tant que (y != 0) {
        si y est impair alors {
            résultat = résultat * puissance;
        }
        puissance = puissance * puissance;
        y = y // 2;
    }
    retourner résultat;
fin
```

Question 1. Donner le nombre de produits réalisés par l'algorithme `expo`.

Question 2. Donner une majoration du nombre de produits réalisés par l'algorithme `expo`.

Ces complexités sont indépendantes de l'entier x, ce qui ne semble finalement pas si naturel puisque les produits sont plus coûteux dès lors que x est grand. Pour un entier naturel n, notons $|n|$ le nombre de bits nécessaires pour représenter n. Dans ce cas, si n et m sont des entiers naturels non nuls,

— le produit $n \times m$ peut être représenté par $|n| + |m|$ bits ;

— la complexité du calcul naïf du produit est en $|n| \times |m|$ opérations.

Intéressons-nous à la longueur de la représentation du contenu de la variable `résultat` au long de l'exécution de l'algorithme `expo`, c'est-à-dire le nombre maximum de bits nécessaires pour représenter l'entier dans `résultat`. Au regard de la boucle et de l'affectation `résultat = résultat * x`, il est évident que cette longueur augmente tout au long de cette exécution.

Notons $L(\textit{résultat}, i)$ la longueur de la représentation du contenu de la variable `résultat` après la i-ième itération de la boucle. Pour $i = 0$, il s'agit de la longueur juste avant la boucle.

Question 3. Que vaut $L(\textit{résultat}, 0)$? Exprimer $L(\textit{résultat}, i+1)$ en fonction de $L(\textit{résultat}, i)$ et de $|x|$. En déduire une expression de $L(\textit{résultat}, i)$ en fonction de $|x|$ et de i.

Notons maintenant $C(\textit{résultat}, i)$ le nombre d'opérations nécessaires pour calculer le produit `résultat * x` lors de la i-ième itération de la boucle. Notons que ce nombre d'opérations est fonction notamment de la longueur de la représentation du contenu de la variable `résultat` à cet instant.

Question 4. Exprimer $C(\textit{résultat}, i+1)$ en fonction de $L(\textit{résultat}, i)$ et de $|x|$. En déduire une expression de $C(\textit{résultat}, i)$ en fonction de $|x|$ et de i.

Question 5. Le coût total $T(x, y)$ lié aux produits dans l'algorithme `expo` peut alors s'exprimer comme

$$T(x, y) = \sum_{i=1}^{y} C(\textit{résultat}, i)$$

Exprimer $T(x, y)$ en fonction de $|x|$ et de y. Sachant que $|n| = \lfloor \log_2(n) \rfloor + 1$, l'algorithme `expo` est-il polynomial en $|x|$ et $|y|$?

Question 6. Que devient la complexité de l'algorithme `expo_rapide` si l'on considère le coût de chaque multiplication ?