

Corrigé du TD n°4

Où il est question de balance, de crêpes et d'un monstre

1 Des pièces et une balance

Exercice 1

1. Deux pesées suffisent dans tous les cas
2. La balance possède trois états : penche à gauche, penche à droite ou équilibré.
Si on met deux tas A et B de poids différents sur les plateaux, on n'apprend rien. Par contre si $|A| = |B|$ alors
 - Si la balance penche vers A la fausse pièce est dans B
 - Si la balance penche vers B la fausse pièce est dans A
 - Sinon (équilibre) la fausse pièce n'est ni en A ni en BOn ne peut avoir d'autre information ! Un algorithme efficace consiste à couper le tas de pièces en TROIS tas A , B et C de même taille (très exactement deux tas A et B de taille $\lceil n/3 \rceil$ et C le reste). Puis on recommence.
3. Si l'algorithme dit que la pièce P est fausse alors on vient de faire une pesée à 2 pièces seulement, on vérifie facilement les 3 cas possibles. Dans le sens inverse (montrer que si P est fausse alors l'algorithme sort bien P) on remarque que l'invariant "la fausse pièce est dans l'un des trois tas A , B ou C considérés" est maintenu.
4. Grâce à l'algorithme ci-dessus, on se ramène à un problème trois fois plus petit (n est divisé par 3 à chaque fois).
Il faut $\lceil \log_3(n) \rceil = O(\log n)$ pesées en tout ! 3 pour $n \leq 27$ etc.
5. On ne peut faire moins : en effet il y a n possibilités. Pour tout algorithme qui fait k pesées il y a 3^k exécutions possibles. Si $3^k < n$ nécessairement deux possibilités tombent dans la même exécution de l'algorithme, qui n'arrive donc pas à les distinguer.

2 Tri-crêpes

Exercice 2 On remarque que au pire il faut $n - 1$ retournements. En effet, on appelle "voisines" des crêpes qui doivent se retrouver ensemble dans le tableau trié. Chaque retournement crée au plus une paire de crêpes voisines. Il existe des piles de crêpes sans voisines (par exemple : $(2, 4, 1, 3, 6, 8, 5, 7, \dots, 4i + 2, 4i + 4, 4i + 1, 4i + 3, \dots)$)

Voici un algorithme qui y arrive en $2n - 2$ retournements (on est donc à un facteur multiplicatif 2 de l'optimum !). Il maintient l'invariant que la pile entre i et n (le fond) est triée.

Pour i de n à 2

Placer la crêpe i à sa place

En effet, des crêpes de numéro plus petit que i se trient sans toucher à en-dessous de i . On s'arrête à 2 car la dernière crêpe est alors bien placée ! Formellement cela donne :

```

Pour i de n à 2
  Soit p la position de la ieme crepe
  Si p != i // on a alors p < i, par invariant
    Retourner [1..p] // amene la ieme crepe en sommet de pile
    Retourner [1..i] // puis la met à sa place
  Fin si
Fin Pour

```

Noter qu'on peut faire des algorithmes plus complexes qui trient en kn avec $k \in [1, 2[$: voir article de Bill Gates¹ et Christos Papadimitriou.

3 Encore un tri

Exercice 3 On utilise un tas-min de p éléments. Le tas contient le premier élément de chaque liste. On est donc sûr que le minimum global s'y trouve. On le place donc en tête de la liste de sortie, en l'extrayant du tas. Ce faisant, on l'a aussi extrait de la liste où il était, la i ème liste. Il faut alors replacer dans le tas la nouvelle tête de la i ème liste. On fait cela n fois. Notez que si une liste se vide, on ne réinsère rien. Quand le tas est vide on a terminé. Chaque extraction et insertion dans le tas est en $O(\log p)$ d'où la complexité. La correction vient du maintien de l'invariant «le minimum global est tête d'une des listes et est sommet du tas-min».

4 La fonction d'Ackerman

Exercice 4

- 0- $A_0(a, b) = b + 1$ est la fonction *successeur*
 - 1- $A_1(a, b) = a + b$ est la fonction *addition*
 - 2- $A_2(a, b) = a * b$ est la fonction *multiplication*
 - 3- $A_3(a, b) = a^b$ est la fonction *puissance*
 - 4- $A_4(a, b) = a^{(a^{(\dots^a)})}$ est la fonction *tour d'exponentielles*
- au-delà de 4 on ne sait plus nommer les fonctions

Exercice 5 On ne sait pas nommer cette classe de complexité! En effet, si $A_i(a, b) = c$ alors c est construits à coup d'appels à A_0 , qui est la fonction "+1". Il en faut donc au moins c . Or les nombres $A_0(10, 10)$, $A_1(10, 10)$, $A_2(10, 10)$, $A_3(10, 10)$, $A_4(10, 10)$ possède respectivement 2 chiffres; 2 chiffres; 3 chiffres; 10 chiffres, puis $10^{10^{10^{10^{10^{10}}}}$ chiffres ($A_4(10, 9)$ chiffres) ce qui est quand même beaucoup. On peut prouver qu'Ackermann croît plus vite que toute fonction récursive-primitive.

L'univers étant agé de 15 milliards d'années soit 10^{18} secondes environ, on en déduit qu'aucune machine ne peut esperer calculer $A_4(10, 10)$. D'ailleurs aucune n'aurait une mémoire suffisante pour le stocker car il n'y a que 10^{80} atomes dans l'univers observable.

1. comme quoi la bio-informatique mène a tout...

5 Complexité en moyenne

Exercice 6 Pour le premier point, il suffit de voir que dans l'unique permutation de $\{1\}$ dans $\{1\}$ (qui envoie 1 sur 1), il y a un seul maximum provisoire.

Pour le second point, il suffit de voir que les permutations ayant un unique maximum provisoire sont celles qui débutent par n . Il y en a $(n-1)!$.

Pour montrer $P_{n,k} = P_{n-1,k-1} + (n-1)P_{n-1,k}$, on va distinguer deux sortes de permutations à n éléments : celles dont le dernier élément est n , et les autres.

Dans le premier cas, en enlevant le dernier élément n , on obtient une permutation à $n-1$ éléments. Cette nouvelle permutations a $k-1$ maximums provisoires si et seulement si l'ancienne avait k maximums provisoires, puisque n est un maximum provisoire.

Et dans le second cas, en enlevant le dernier élément de la permutation, on n'enlève pas de maximum provisoire. Si le dernier élément est d , en remplaçant tous les éléments $x > d$ par $x-1$ et en enlevant le dernier élément, on obtient une permutation de $n-1$ éléments qui a autant de maximums provisoires. Il y a $n-1$ valeurs possibles pour d .

Formellement, en notant $A(d)_{n,k}$ le nombre de permutations à n éléments tels que le dernier élément est d et qui ont k maximum provisoires, on a $A(n)_{n,k} = P_{n-1,k-1}$ (premier cas) et $A(d)_{n,k} = P_{n-1,k}$ si $d < n$ (deuxième cas).

Puisque $P_{n,k} = \sum_{d=1}^n A(d)_{n,k}$, on a le résultat.

Enfin, calculons combien vaut S_n . On a bien $S_1 = 1$ et $S_2 = 3/2$.

On peut éviter de se lancer dans de grands calculs, mais juste regarder comment on passe de $n-1$ à n éléments en ajoutant un dernier élément d . Comme on l'a vu, pour chaque d on peut mettre en bijection les permutations à n éléments terminant par d et les permutations à $n-1$ éléments. Il y a un seul cas où l'on augmente le nombre de maximums locaux : celui où $d = n$. Cela arrive une fois sur n . Il y a donc une chance sur n d'augmenter la quantité de 1. On a donc

$$S_n = S_{n-1} + \frac{1}{n} = H_n = \Theta(\log n)$$

En effet le logarithme est la primitive de $1/n$ qui s'annule en 1. H_n en est une approximation.