TD 4: Algorithmique fonctionnelle
Complexité

Nga Nguyen - Stefan Bornhofen - Peio Loubière

1 Puissance

Voici 2 versions différentes permettant de calculer la puissance x^n vues en TD 3. Comparer leur complexité :

```
1  Fonction puissance(x, n : Entier) : Entier
2  Début
3    Si n = 0 Alors
4      retourner 1
5    Sinon
6      retourner x*puissance(x,n-1)
7    FinSi
8  Fin
9
10 ================
11
12 Fonction puissance-dic(x, n : Entier) : Entier
13 Début
14   Si n = 0 Alors
15     retourner 1
16   SinonSi (n mod 2 = 0)
17         retourner puissance-dic(x*x,n/2)
18   Sinon
19         retourner x*puissance-dic(x,n-1)
20   FinSi
21 Fin
```

Solution :
On choisit la multiplication comme opération élémentaire. On note C(n) le nombre de multiplications au rang n.

– Version 1 :

$$\begin{cases} C(0) = 0 \\ C(n) = 1 + C(n-1) \end{cases}$$

⇒ complexité linéaire (O(n)).

– Version 2 :

$$\begin{cases} C(0) = 0 \\ C(n) = 1 + C(n/2) \text{ , si n pair (1)} \\ C(n) = 1 + C(n-1) \text{ , si n impair (2)} \end{cases}$$

Dans le pire des cas, n aura la forme $2^k + 2^{k-1} + ... + 2^1 + 1$ pour que l'on passe par (2) puis (1) puis (2) puis (1) ... On a :

$$\begin{aligned} &C(n) = C(2^{k+1} - 1) = 1 + C(2^{k+1} - 2) \\ &C(2^{k+1} - 2) = 1 + C(2^k - 1) \\ &C(2^k - 1) = 1 + C(2^k - 2) \\ &C(2^k - 2) = 1 + C(2^{k-1} - 1) \\ &... \\ &C(2^2 - 1) = 1 + C(2^2 - 2) \\ &C(2^2 - 2) = 1 + C(2^1 - 1) \\ &C(2^1 - 1) = 1 + C(0) = 1 \\ &\Rightarrow C(2^{k+1} - 1) = 2k + 1 \\ &\Rightarrow C(n) = 2log(n+1) - 1 \end{aligned}$$

⇒ complexité logarithmique O(logn)

2 Jeu par élimination

Soit le jeu suivant consistant à tirer au sort un "vainqueur" parmi un groupe d'enfants, au moyen d'une pièce de monnaie :

- S'il n'y a qu'un enfant, il est évidemment vainqueur.
- S'il y en a plusieurs, on lance la pièce. Si c'est face qui sort, on élimine au hasard (par un jeu de type "chaises musicales" par exemple) un des enfants ; si c'est pile on en élimine deux (à moins qu'il n'en reste que deux, auquel cas on n'en élimine qu'un).
- Le gagnant est le dernier enfant non éliminé.

Question 1 : Si l'on considère le lancer d'une pièce comme opération fondamentale, trouver l'équation de récurrence correspondant à ce jeu dans le cas moyen.

Question 2 : Montrer par récurrence, à partir de cette équation, que l'algorithme précédent est en O(n), où n est le nombre initial d'enfants.

Solution :

- Équation de récurrence dans le cas moyen :
  On note C(n) le nombre de lancers avec n enfants. À chaque étape, on a une chance sur deux de retirer un enfant et une chance sur deux d'en retirer 2. En outre, on effectue un lancer par étape. Cela donne l'équation suivante, dans le cas moyen : $C(n) = 1/2 * C(n-1) + 1/2 * C(n-2) + 1$
  Le premier terme de la somme correspond au cas où l'on retire un enfant, le second à celui où l'on en retire deux. On sait en outre que $C(1) = 0$ (pas de lancer s'il n'y a qu'un enfant) et que $C(2) = 1$ (après un lancer, quel que soit le résultat, on se retrouve avec le gagnant).

$$\begin{cases} C(1) = 0 \\ C(2) = 1 \\ C(n) = 1 + 1/2 * C(n-1) + 1/2 * C(n-2) \end{cases}$$

- Algorithme en O(N) ?
  Pour montrer que cet algorithme est en O(n), on va supposer que :

$$\forall i \leq n, C(i) \leq i$$

  Cette propriété est clairement vraie pour $n \leq 2$. Dans le cas général, on majore l'équation de C(n) par

$$C(n) \leq 1/2 * (n-1) + 1/2 * (n-2) + 1 = n - 1/2 \leq n$$

3 Mots décroissants

Soit A un alphabet fini, muni d'un ordre total $\leq$. On considère A^* l'ensemble des mots finis sur A. Soit m un mot de longeur l. Pour tout $i, 1 \leq i \leq l$, on note m_i la ième lettre de m. On dit qu'un mot m de longueur l est décroissant, s'il est vide ou si pour tout $i, 1 \leq i \leq l-1$, $m_{i+1} \leq m_i$. Si de plus il n'y a pas deux lettres égales dans le mot m, m est dit strictement décroissant.
Soit A^*_D le sous ensemble de A^* constitué des mots décroissants, et A^*_{SD} le sous-ensemble de A^*_D constitué des mots strictement décroissants.

Donner en pseudo-code un algorithme récursif pour chacune de ces fonctions puis déterminer sa complexité :

Question 3 : Decroissant : $A^* \rightarrow \{vrai, faux\}$
Cette fonction retourne vrai si et seulement si le mot considéré est décroissant.

Question 4 : Simplifie : $A^*_D \rightarrow A^*_{SD}$
Cette fonction retourne le mot de A^*_{SD} ayant le même ensemble de lettres que le mot initial, mais où les répétitions de lettres ont été éliminées.

Question 5 : Intersection : $A^*_{SD} \times A^*_{SD} \rightarrow A^*_{SD}$
Cette fonction retourne le mot de A^*_{SD} contenant les lettres qui sont dans les deux mots.

Solution :

```
1  Fonction Decroissant(m : Chaîne) : Booléen
2  Variables l : Entier
3  Début
4    l ← longueur(m)
5    Si (l ≤ 1) Alors
6      retourner vrai
7    Sinon
8      retourner (extrait(m,1,1) ≥ extrait(m,2,1)) et
           Decroissant(extrait(m,2,l-1))
9    FinSi
10 Fin
11
12 ======
13
14 Fonction Simplifie(m : Chaîne) : Chaîne
15 Variables l : Entier
16 Début
17   l ← longueur(m)
18   Si (l ≤ 1) Alors
19     retourner m
20   Sinon
21         Si (extrait(m,1,1) = extrait(m,2,1)) Alors
22         retourner  Simplifie(extrait(m,2,l-1))
23     Sinon
24         retourner extrait(m,1,1) & Simplifie(extrait(m,2,
               l-1))
25   FinSi
26 Fin
27
28 ======
29
30 Fonction Intersection(m1, m2 : Chaîne) : Chaîne
31 Variables l1, l2 : Entier, c1, c2 : Chaîne
32 Début
33   l1 ← longueur(m1)
34   l2 ← longueur(m2)
35   // si un de 2 mots est vide, on retourne un mot vide
36   Si (l1 = 0) ou (l2 = 0) Alors
37     retourner ""
38   Sinon // on compare les 2 premières lettres
39     c1 ← extrait(m1,1,1)
40     c2 ← extrait(m2,1,1)
41     Si (c1 = c2) Alors
42       retourner c1 & Intersection(extrait(m1,2,l1-1),
             extrait(m2,2,l2-1))
43     SinonSi (c1 > c2)      4
44       retourner Intersection(extrait(m1,2,l1-1),m2)
45     Sinon
46       retourner Intersection(m1,extrait(m2,2,l2-1))
47     FinSi
48   FinSi
49 Fin
```

- Decroissant : complexité linéaire O(l) où l est la longueur du mot
- Simplifie : complexité linéaire O(l) où l est la longueur du mot
- Intersection : complexité O(l1+l2) où l1 et l2 sont les longueurs de 2 mots