Université de Nice – Sophia Antipolis
Outils Formels Pour l'Informatique

Licence 2 Informatique
20012–2013

TD 1

Complexité des algorithmes

1 Echauffement

Exercice 1) Réviser l'écriture d'un nombre entier sous forme décimale, sous forme binaire, sous forme hexadécimale. Ecrire les nombres 189 (en base 10), 10101011 (en base 2), A1B2 (en base 16) dans les différentes bases.

Rien de bien compliqué ici

Exercice 2) Vous connaissez certainement les algorithmes qui permettent de décider si un nombre entier est divisible par 2, par 3, par 5, en n'utilisant que l'addition d'entiers et la comparaison de deux entiers. Quelle est la complexité de chacun de ces algorithmes, en raisonnant sur le nombre de chiffres nécessaires à l'écriture décimale du nombre et en prenant comme opérations élémentaires prises l'addition et la comparaison.

Pour la division par 2 ou 5, complexité en $O(1)$, pour la division par 3, complexité en n, nombre de décimales du nombre.

Exercice 3) Adapter l'algorithme de multiplication *à la russe* aux nombre binaires.

Les seuls changements par rapport à la base 10 sont le fait que la multiplication et la division par 2 sont très simples en base 2 (décalage à droite ou à gauche)

---

2 Exercices d'entrainement

Exercice 4) Si une instruction est traitée en 10^{-7} seconde, quelle est la taille maximale que peut traiter un algorithme en $\log(n)$, n, $n\log(n)$, n^2, 2^n si on dispose de une seconde / une minute / une heure / une journÈe ? Même questions si chaque instruction est traitée en $10^{-20} seconde$

Il faut résoudre dan chaque cas l'équation $C(n) * 10^{-7} = temps$ où $C(n)$ désigne la complexité.

Exercice 5)

1. Prouver que $n^2 - n = O(n^2)$.

Dans ce premier exemple, il suffit de remarquer que $n^2 - n < n^2$ pour n entier.

---

2. Prouver que $n^2 + n = O(n^2)$.

Ici on remarque que que $n^2 + n = n^2(1 + \frac{1}{n})$ et que $\lim_{n \to +\infty}(1 + \frac{1}{n}) = 1$ et que par conséquent il existe un entier n_0 à partir duquel, pour $n > n_0$ on a $1 + \frac{1}{n} < 1.1$ par exemple. On en dénduit alors que pour $n > n_0$ on a $n^2 + n < 1.1n^2$. Un raisonnement similaire permettrait de montrer que $n^2 + n = \Theta(n^2)$

---

3. Donner l'ordre de grandeur des expressions suivantes:

   (a) $\dfrac{n^2 + 3n + 1}{n + 1}$

   (b) $\dfrac{n\log(n) + n^2 + \log(n)^2}{n + 1}$

Avec la même technique, on montre que (a) et (b) sont en $\Theta(n)$.

---

4. Si $f(n)$ en une fonction en $O(n)$, les affirmations suivantes sont-elles vraies ?

   (a) $(f(n))^2 = O(n^2)$;

   (b) $2^{f(n)} = O(2^n)$.

S'il existe une constante $c > 0$ telle que pour n assez grand, $f(n) < cn$, il est clair, puisque la fonction x^2 est croissante que $(f(n))^2 < c^2 n2$ et que par conséquent $(f(n))^2 = O(n^2)$.

Par contre, on a, pour la même raison (2^x est également croissante), $2^{f(n)} < 2^{cn} = (2^c)^n$. Si $c > 1$, alors $2^c > 2$ et $2^{f(n))}$ n'est pas forcément en 2^n. Par exemple, $2n$ est en $O(n)$ et pourtant 2^{2n} est en $O(4^n)$, et $4^n >> 2^n$, c'est à dire que 4^n ne peut pas être majoré par $c'2^n$ où c' est une constante.

Exercice 6) Itérations. Calculer la complexité des programmes suivants. Vous donnerez une borne supérieure avec un $O()$ dans un premier temps et affinerez votre calcul en utilisant $\Theta()$.

listing 1

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j = j++) {
    x = x + 3; }
  }
```

listing 2

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j = j++) {
    x = x + 3; }
  }
```

listing 3

```
for (int i = 0; i < n; i++) {
    for (int j = i; j < n-i; j++) {
    x = x + 3; }
  }
```

listing 4

```
for (int i = 5; i < n; i++) {
    for (int j = i-5; j < i+5; j++) {
    x = x + 3; }
  }
```

listing 5

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i*n; j = j++) {
    x = x + 3; }
  }
```

listing 6

```
for (int i = n; i > 1; i=i/2) {
    for (int j = 0; j < i; j++) {
    x = x + 3; }
  }
```

Les algorithmes 1, 2 et 3 sont en $\Theta(n^2)$, le 4 en $\Theta(n)$, le 5 en $\Theta(n^3)$ et le 6 en $\Theta(n)$

---

3 Pour aller plus loin

Exercice 7) Écrire un algorithme qui calcule, pour un tableau T de taille n donné en paramètre, la somme :

$$\sum_{i,j=1}^{n} (T(i) - T(j))^2$$

1. Écrire l'algorithme "naïf" quadratique qui vient en premier à l'esprit.
2. Est-il possible de linéariser cet algorithme ?

L'algorithme naïf, constitué de deux boucles imbriqués est évidemment quadratique. On peut le linéariser en remarquant que $S = \sum_{i=1}^{n}\sum_{j=1}^{n}(T_i - T_j)^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} T_i^2 + \sum_{i=1}^{n}\sum_{j=1}^{n} T_j^2 - 2\sum_{i=1}^{n}\sum_{j=1}^{n} T_i T_j$ s'écrit donc $S = 2nS_2 - 2S_1^2$ où $S_2 = \sum_{i=1}^{n} T_i^2$ et $S_1 = \sum_{i=1}^{n} T_i$. A l'aide de ce calcul, on peut alors écrire un algorithme linéaire car il suffit d'une boucle simple pour calculer S_1 et S_2.

Exercice 8) Trouver un algorithme qui teste si un tableau de taille n représente une permutation (c'est-à-dire si tous ses éléments sont distincts et compris entre 1 et n).

1. Donner un premier algorithme naïf qui soit quadratique.
2. Donner un second algorithme linéaire utilisant un tableau auxiliaire.

Il faut vérifier que tout élément du tableau T est compris entre 0 et n et que les éléments du tableau sont deux à deux distincts. Cet algorithme est évidemment quadratique (deux boucles imbriqués). On peut linéariser l'algorithme en utilisant un tableau auxiliaire de même taille que l'on initialise à 1. On lit alors le tableau T à l'aide d'une boucle simple et on met à jour la valeur du tableau auxiliaire aux[T[i]] à 1. Si cette valeur avait été positioné à 1 auparavant, cela signifie que cette valeur apparait deux fois dans T qui n'est donc pas un tableau de permutation.

algorithme linéaire

```
for (int i = 1; i < n+1; i++) {
      aux[i]=0;
}
for (int i = 1; i < n+1; i++) {
      if(T[i]<1 || T[i]>n || aux[T[i]]>0,return(false); aux[T[i]]=1);
}

return(true)
```

Exercice 9) Ecrire un algorithme pour calculer p^n, qui soit de complexité $O(\log_2 n)$ (indication: considérer n écrit en binaire).

Il suffit de considérer la récurrence suivante

$$p^{2n} = (p^n)^2$$

$$p^{2n+1} = p(p^n)^2$$

On en dénduit alors l'algorithme suivant

algorithme en log(n)

```
if(n=0, 1,
   if((modulo(n,2)=0, (x^(n/2) )^2,
          x*(  x^((n-1)/2) )^2)
```

On remarque que cet algorithme peut aussi s'appliquer au produit de matrices.

Exercice 10) Etudiez le nombre d'additions réalisées par l'algorithme suivant dans le meilleur des cas, dans le pire des cas et enfin en moyenne en supposant que chaque test a une probabilité $\frac{1}{2}$ d'être vrai.

```
for (int i = 0; i < n; i++) {
    if(T[i]>a,s=s+T[i]);  }
```

- Si tous les éléments du tableau sont inférieurs à a, aucune addition n'est exécutée. L'algorithme est en $O(1)$.
- Si tous les éléments du tableau sont supérieurs ou égaux à a, n additions sont exécutées. L'algorithme est en $O(n)$.
- Le nombre d'additions dépend du nombre déléments du tableau qui sont $> a$. Pour choisir k éléments du tableau parmi n, on a $\binom{n}{k}$ possibilités avec probabilité $\frac{1}{2^n}$. La complexité en moyenne est donc $\sum_{i=1}^{n} k\binom{n}{k}\frac{1}{2^n}$.

  On calcule $\sum_{i=1}^{n} k\binom{n}{k}$ en remarquant que, puisque $(1+x)^n = \sum_{i=1}^{n}\binom{n}{k}x^k$, on obtient en dérivant par rapport à x, $n(1+x)^{n-1} = \sum_{i=1}^{n} k\binom{n}{k}x^{k-1}$.

  En fixant x à 1 dans cette égalité, on obtient $\sum_{i=1}^{n} k\binom{n}{k} = n2^{n-1}$. On en déduit alors que la complexité en moyenne est $\frac{n}{2}$.
