4 Complexité

4.1 Indécidabilité de l'arrêt

Problème de l'arrêt

Définition Le problème de l'arrêt est **indécidable**. Cela signifie qu'il n'existe pas de méthode systématique (d'algorithme) qui permet de savoir si un algorithme donné se termine.

Il existe d'autres problèmes indécidables (dans l'arithmétique de Peano) :
- Une équation diophantienne admet-elle des solutions ?
- Problème d'Hercule contre l'hydre ;
- Suites de Goodstein...

4.2 Complexité ?

Complexité

Complexité d'un algorithme
- en temps : exprime la manière dont le temps d'exécution évolue en fonction de la **taille** des données d'entrée
- en espace : exprime la manière dont la taille mémoire occupée par l'algorithme évolue en fonction de la **taille** des données d'entrée

Nous ne parlerons que de la complexité en temps

Auparavant, il faut définir ce qu'est la taille d'une donnée

4.3 Taille des données

Définition La taille d'une donnée est l'espace mémoire qu'elle occupe, en bits.

Changement de base
- le nombre de chiffres dépend de la base d'écriture
- la base unaire est «spéciale» et inefficace
- nombre de chiffres de n en base k : $log_k(n)$
- rapport des longueurs dans les bases a et b :

$$log_a(n) = \frac{ln(n)}{ln(a)}$$

$$\frac{log_a(n)}{log_b(n)} = \frac{ln(b)}{ln(a)}$$

Ce nombre ne dépend pas de n
⇒Taille des données définie à une constante multiplicative près

Tailles fixes vs. taille variable

Certains types sont à taille fixe. D'autres à taille variable. Quand utiliser l'un ou l'autre ?

Taille fixe
- calculs sur des entiers de 4 octets.
- nombres flottants sur 4 ou 8 octets

Taille variable
- manipulation de grands entiers (en cryptographie par exemple)
- calcul rationnel exact
- tableaux d'entiers de taille fixe
- ...

4.4 Calcul de complexité

Complexité en temps

- L'idée est de compter le nombre d'instructions élémentaires
- Qu'est-ce qu'une intruction élémentaire ?
  - instruction qui s'exécute en un temps indépendant de la taille des données
  - exemple : addition, multiplication, affectation, dans le cas de codages à taille fixe

Attention La modèle choisi pour la taille des données n'est parfois pas évident. Aussi, il est préférable de préciser «en fonction de quoi» est donnée la complexité :
- en fonction du nombre n passé en paramètre ?
- en fonction du nombre de chiffres du nombre n passé en paramètre ?
- en fonction de la taille n (nombre de cases) du tableau ?
- ...

4.5 Exemples

Recherche d'un élément dans un tableau trié (6.2)

Problème : Recherche d'un élément dans un tableau trié

Entrées Un tableau **trié (croissant)** `tab` d'entiers et un entier `k`

Sorties Le plus grand élément de `tab` inférieur ou égal à `k` s'il existe ou le plus petit élément de `tab` sinon

```
fonc rech1(t:tableau,k:entier):entier
| i : entier
| i←0
| repeter t.q. i<longueur(t) et t[i]≤k
| |  i ←i+1
|
| si i=0
| | retourner t[0]
| sinon
| | retourner t[i−1]
```

```
fonc rech2(t:tableau,k:entier):entier
| d,f,m : entiers
| d←0
| f←longueur(t)−1
| repeter tant que d≤f
| | m ←(d+f)/2;
| | si t[m]≤k
| | | d←m+1
| | sinon
| | | f←m−1
| if f<0
| | f←0
| retourner t[f]
```

```
fonc rech1(t:tableau,k:entier):entier
| i : entier
| i←0
| repeter t.q.  i<longueur(t) et t[i]≤k
| |  i ←i + 1
|
| si i=0
| | retourner t[0]
| sinon
| | retourner t[i−1]
```

- Taille données : taille n du tableau
- Nombre d'opérations :

$$1 + \text{boucle}(\approx 4) + 2$$

- La boucle fait au pire n tours
- Nombre d'opérations :

$$T(n) = 1 + 4n + 2 = 4n + 3$$

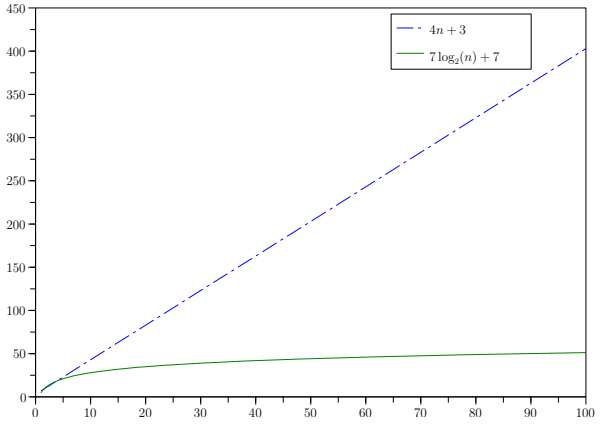
```
fonc rech2(t:tableau,k:entier):entier
| d,f,m : entiers
| d←0
| f←longueur(t)−1
| repeter tant que d≤f
| | m ←(d+f)/2;
| | si t[m]≤k
| | | d←m+1
| | sinon
| | | f←m−1
|
| si f<0
| | f←0
| retourner t[f]
```

- Taille données : taille n du tableau
- Nombre d'opérations :

$$3 + \text{boucle}(\approx 7) + 2$$

- La boucle fait au pire $\log_2(n)$ tours
- Nombre d'opérations :

$$T(n) = 5 + 7\log_2(n)$$



Notation $\Theta(.)$

- Pour une fonction $g(n)$ donnée, on note :

$$\Theta(g(n)) = \{f(n), \exists(c_1, c_2, n_0) \in R^2 \times N/$$
$$\forall n > n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

- Pour une fonction $g(n)$ donnée, on note :

$$O(g(n)) = \{f(n), \exists(c, n_0) \in R \times N/$$
$$\forall n > n_0, 0 \leq f(n) \leq cg(n)\}$$

À retenir On recherche :
- le comportement asymptotique
- à une constante multiplicative près
- à une constante additive près

Autrement dit :
- $\Theta(af(n)) \Rightarrow \Theta(f(n))$
- $\Theta(f(n) + k) \Rightarrow \Theta(f(n))$
- $\Theta(f(n) + g(n)) \Rightarrow \Theta(max(f(n), g(n)))$

Résultats attendus :
- 1, $\log(n)$,
- n, $n\log(n)$,
- n^2,...,n^k,
- 2^n
- ...

Recherche d'un élément dans un tableau trié Complexités dans le cas le pire :
- rech1, recherche linéaire, $T(n) = 4n + 3 = \Theta(n)$
- rech2, recherche dichotomique, $T(n) = 7\log_2(n) + 3 = \Theta(\log(n))$

Exercice (6.6) Une opération = 1 nanoseconde
Comparer les temps d'exécutions d'algorithmes de complexité :

$$\log(n), n, n\log(n), n^2, n^4, 2^n$$

pour les valeurs de n : 100,1000,10000

Algorithme récursif

```
fonc rech3(t:tableau,d,f,k:entiers):entier
    m : entiers
    si d>f
        si f<0
            retourner t[0]
        sinon
            retourner t[f]

    m ←(d+f)/2
    si t[m]≤k
        retourner rech3(t,m+1,f,k)
    sinon
        retourner rech3(t,d,m−1,k)
```

- taille données : taille n du tableau
- Nombre d'opérations : $T(n)$. L'algorithme nous donne :

$$T(n) = 6 + T(n/2) = 6 + T(n/2)$$

- On veut calculer $T(n)$ en fonction de n.
- On a $T(1) \approx a$ (peu importe la valeur exacte)

Cas particulier $n = 2^k$ (c-à-d $k = \log_2(n)$), $u_k = T(2^k)$. Alors $u_k = 6 + u_{k-1}$, $u_0 = a$. On déduit : $u_k = 6k + a$. Donc $T(2^k) = 6k+a$ et $T(n) = 6\log_2(n)+a$. Finalement, on retrouve bien : $T(n) = \Theta(\log(n))$

Algorithmes de tri

```
insertion (t [] : tab d'entiers)
    i,j : entiers
    val : entier
    repeter pour j de 1 à taille (t)−1
        val← t[j]
        i← j−1
        repeter tant que i≥0 et t[i]>val
            t[i+1]← t[i]
            i← i−1

        t[i+1]← val
```

Le principe ressemble à la méthode utilisée pour trier une poignée de cartes à jouer.
Le tableau est parcouru de gauche à droite (boucle extérieure). Pour chaque valeur (clé) on cherche à l'insérer à sa gauche (qui est supposée triée). Pour cela, on parcourt le tableau de droite à gauche, à partir de la case située juste à gauche de la clé (boucle intérieure). Dès qu'on a trouvé la position d'insertion, on insère la clé.

Exercice 6.4 Calculez la complexité en temps dans le cas moyen en fonction de n

```
rapide(t [] : tab d'entiers, p, r : entiers)
    q : entier
    si p<r
        q←partitionner(t,p,r)
        rapide(t,p,q)
        rapide(t,q+1,r)
```

Exercice 6.5 Calculez la complexité en temps dans le cas moyen en fonction de n

```
partitionner (t [] : tab d'entiers, p, r : entiers)
    cle,i,j : entiers
    cle ←t[p]
    i← p−1
    j← r+1
    repeter indéfiniment
        j← j−1
        repeter tant que t[j]>cle
            j← j−1

        i←i+1
        repeter tant que t[i]<cle
            i← i+1

        si i<j
            echanger(t,i,j)
        sinon
            renvoyer j
```