Plan du cours

- **Structures de données**
  - Algorithmes, preuve, complexité
  - Récursivité
  - Types abstraits, listes, ensembles
  - Classements
  - Recherches

- **Algorithmique**
  - Graphes et leurs traitements
  - Algorithmes sur les langages et automates
  - Traitement des chaînes de caractères

Algorithmique

Conception de méthodes pour la résolution de problèmes

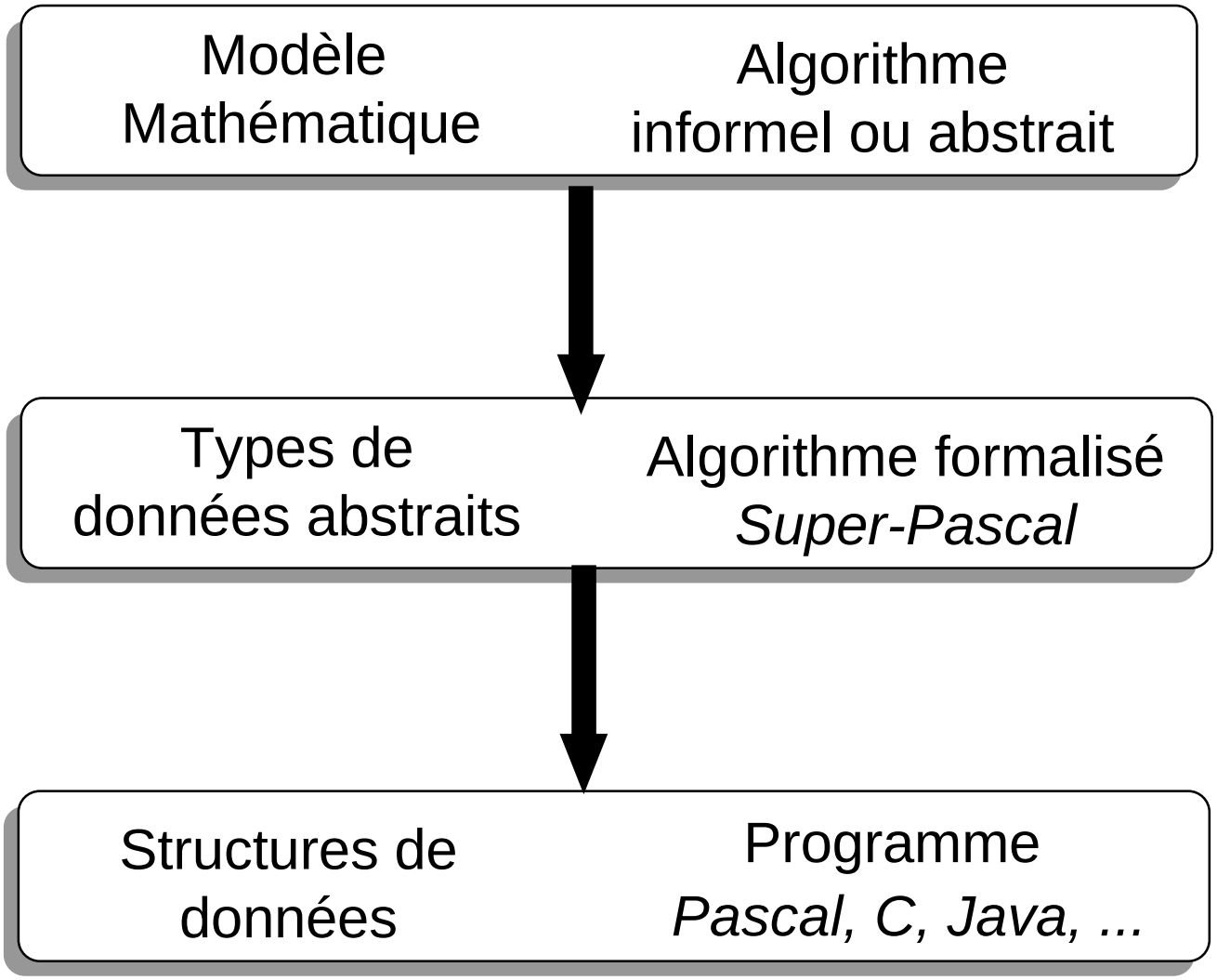
Description des données
Description des méthodes
Preuve de bon fonctionnement

Complexité des méthodes

Efficacité : temps de calcul, espace nécessaire, ...
Complexité intrinsèque, optimalité
Solutions approchées

Réalisation - implémentation

Organisation des objets
Opérations élémentaires


Modèle
Mathématique
Algorithme
informel ou abstrait
Types de
données abstraits
Algorithme formalisé
Super-Pascal
Structures de
données
Programme
Pascal, C, Java, ...

Algorithme

Al Khowarizmi, Bagdad IXè siècle.

Encyclopedia Universalis :

« Spécification d'un schéma de calcul sous forme d'une suite finie d'opérations élémentaires obéissant à un enchaînement déterminé. »

DONNÉES  RÉSULTATS, ACTIONS

Composition d 'un nombre fini d'opérations dont chacune est :

- définie de façon rigoureuse et non ambiguë
- effective sur les données adéquates (exécution en temps fini)

Bibliographie

Beauquier, Berstel, Chrétienne
Éléments d'algorithmique, Masson, 1993.

Sedgewick
Algorithmes en C, InterÉditions, 1992.

Aho, Hopcroft, Ullman
Structures de données et algorithmes, InterÉditions, 1987.

Cormen, Leiserson, Rivest
Algorithmes, Dunod, 1994.

Bien distinguer !

Spécification d ’un algorithme :

ce que fait l’algorithme

cahier des charges du problème à résoudre

Expression d ’un algorithme :

comment il le fait

texte dans un langage de type *Pascal / C*

Implémentation d ’un algorithme :

traduction du texte précédent

dans un langage de programmation réel

Éléments de méthodologie

- Programmation structurée
- Modularité
- Programmation fonctionnelle
- Récursivité

- Types abstraits de données
- Objets
- Réutilisabilité du code

Exemple : classement

Suite $s = (7, 3, 9, 4, 3, 5)$
Suite classée $c = (3, 3, 4, 5, 7, 9)$
(en ordre croissant)

Spécification

suite → suite classée

Méthode informelle (Bulles)

tant que il existe i tel que $s_i > s_{i+1}$
faire échanger (s_i, s_{i+1})

Opérations élémentaires

comparaisons
transpositions d'éléments consécutifs

Preuve

Inversion : (i,j) tels que $i < j$ et $s_i > s_j$

Inv(s) = nombre d'inversions dans la suite s
Inv(7, 3, 9, 4, 3, 5) = 8

$s = (s_1, \ldots, s_{i-1}, s_i, s_{i+1}, s_{i+2}, \ldots, s_n)$

échange de s_i et s_{i+1}

$s' = (s_1, \ldots, s_{i-1}, s_{i+1}, s_i, s_{i+2}, \ldots, s_n)$

Note : si $s_i > s_{i+1}$, alors $Inv(s') = Inv(s) - 1$

c suite classée  $Inv(c) = 0$

Formalisation

Algorithme Bulles1

répéter
{ *i* : = 1 ;
tant que *i* < *n* et $s_i \leq s_{i+1}$ **faire** *i* := *i* + 1 ;
si *i* < *n* **alors**
{échanger (s_i, s_{i+1}) ; *inversion* := vrai ; }
sinon
inversion := faux ;
}
tant que *inversion* = vrai ;

Temps de calcul

Complexité en temps

T(*algo, d*) = temps d'exécution de l'algorithme *algo* appliqué aux données *d*

Éléments de calcul de la complexité en temps

T(opération élémentaire) = constant

T(**si** *C* **alors** *I* **sinon** *J*) $\leq \mathrm{T}(C) + \max(\mathrm{T}(I), \mathrm{T}(J))$

T(**pour** $i \leftarrow e_1$ **à** e_2 **faire** I_i) $= \mathrm{T}(e_1) + \mathrm{T}(e_2) + \Sigma\, \mathrm{T}(I_i)$

Temps de calcul de procédures récursives

$\rightarrow$ solution d'équations de récurrence

Temps de calcul (suite)

$T(algo, d)$ dépend en général de d.

Complexité au pire

$$T_{MAX}(algo, n) = \max \{T(algo, d) ; d \text{ de taille } n\}$$

Complexité au mieux

$$T_{MIN}(algo, n) = \min \{T(algo, d) ; d \text{ de taille } n\}$$

Complexité en moyenne

$$T_{MOY}(algo, n) = \sum_{d \text{ de taille } n} p(d).T(algo, d)$$

$p(d)$ = probabilité d'avoir la donnée d

$$T_{MIN}(algo, n) \leq T_{MOY}(algo, n) \leq T_{MAX}(algo, n)$$

Algorithme Bulles2

```
{ pour j := n-1 à 1 pas -1 faire
    pour i := 1 à j faire
      si s_i > s_{i+1} alors échanger(s_i, s_{i+1}) ;
}
```

Temps d 'exécution T_{MAX} (Bulles2, n)

j donné $\longrightarrow$ temps $\leq k.j + k'$

$$T_{MAX}(\text{Bulles2}, n) \leq \sum_{j} (k.j + k') + k''$$

$$\leq k.\frac{n(n-1)}{2} + k'.(n\text{-}1) + k''$$

Nb comparaisons = $\frac{n(n-1)}{2}$ Nombre d'échanges $\leq \frac{n(n-1)}{2}$

Comparaisons de complexités

Ordres de grandeur asymptotique

"grand O"

$\mathrm{T}(n) = \mathrm{O}(f(n))$ ssi

$\exists\, c > 0 \;\; \exists\, N > 0 \;\; \forall\, n > N \;\;\; \mathrm{T}(n) \leq c.f(n)$

"grand oméga"

$\mathrm{T}(n) = \Omega\,(f(n))$ ssi

$\exists\, c > 0 \;\; \exists\, N > 0 \;\; \forall\, n > N \;\;\; \mathrm{T}(n) \geq c.f(n)$

"grand théta"

$\mathrm{T}(n) = \Theta(f(n))$ ssi

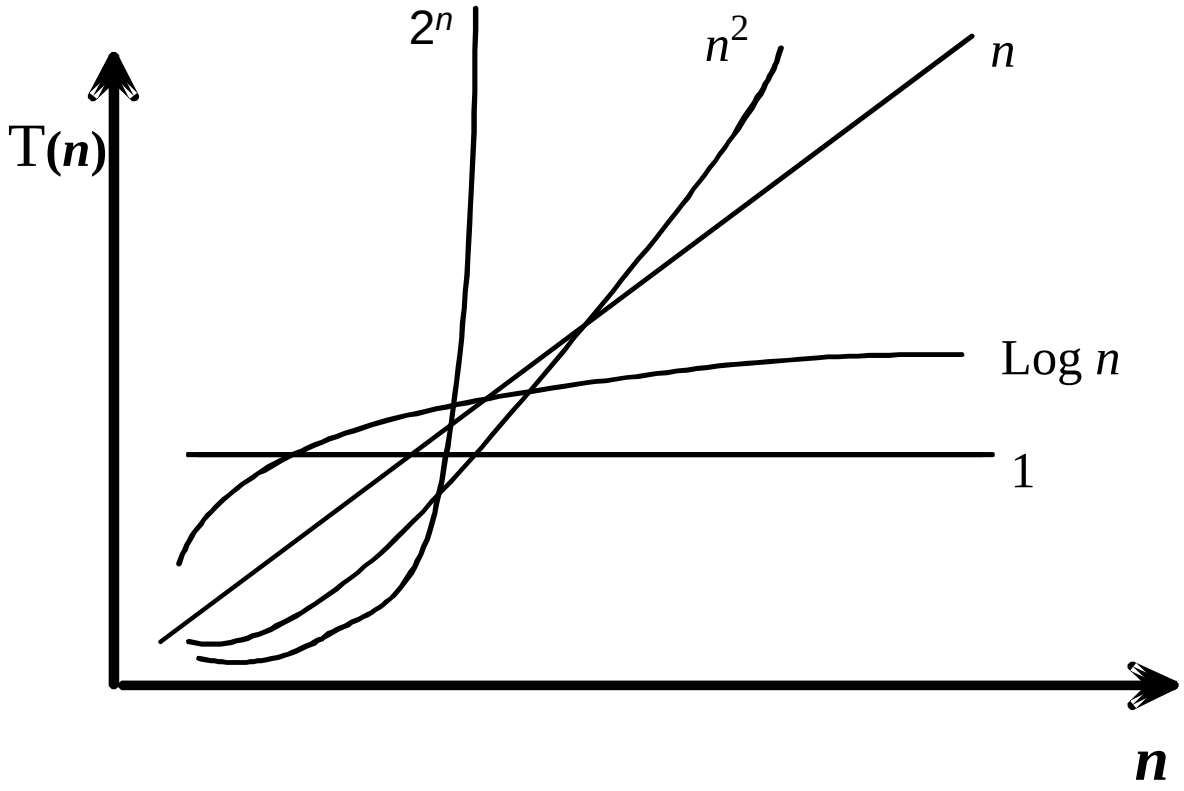
$\mathrm{T}(n) = \mathrm{O}(f(n))$ et $\mathrm{T}(n) = \Omega(f(n))$

Exemple (complexités au pire)

$\mathrm{T}(\mathrm{BULLES2},n) = \mathrm{O}(n^2) = \Omega(n^2)$ $\qquad$ $\mathrm{T}(\mathrm{BULLES1},n) = \mathrm{O}(n^3) = \Omega(n^3)$

Ordres de grandeur (exemples)

1, log n, n, n.log n, n^2, n^3, 2^n



	Évolution du temps quand la taille est **10 fois** plus grande	Évolution de la taille quand le temps est **10 fois** plus grand
1	t	∞
$\log_2 n$	$t + 3{,}32$	n^{10}
n	$10 \times t$	$10 \times n$
$n \log_2 n$	$(10 + \varepsilon) \times t$	$(10 - \varepsilon) \times t$
n^2	$100 \times t$	$3{,}16 \times n$
n^3	$1000 \times t$	$2{,}15 \times n$
2^n	t^{10}	$n + 3{,}32$

Optimalité

E (P) = ensemble des algorithmes qui résolvent un problème P
au moyen de certaines opérations élémentaires.

$A \in E$ (P) est optimal en temps (par rapport à E (P))
ssi $\forall B \in E$ (P) $\forall$ donnée d $\mathrm{T}(B,d) \geq \mathrm{T}(A,d)$

A $\in E$ (P) est **asymptotiquement optimal**
ssi $\mathrm{T}(A,n) = \mathrm{O}(\mathrm{T}(B,n))$

Exemple (complexité au pire)
BULLES 2 est asymptotiquement optimal parmi les algorithmes de classement qui utilisent les opérations élémentaires : comparaisons, transpositions d'éléments consécutifs.

Preuve : Inv((*n*,*n*-1, …,1)) = $\frac{n(n-1)}{2}$ = O (n^2)

Remarques

1. Temps de conception ne doit pas être >> temps d ’exécution.
2. La complexité de la description nuit à la mise à jour d ’un algorithme.
3. Les constantes de proportionnalité peuvent être importantes si les données sont réduites.
4. Réduire le temps peut conduire à augmenter l’espace nécessaire
5. Pour les algorithmes numériques, la précision et la stabilité des valeurs sont des critères importants.

P G C D

Spécification

Calculer pgcd(*n,m*), plus grand diviseur commun aux entiers ≥0, *n* et *m*.

Algorithme 1 (*n*≥*m*)

pour *i* : = *m* à 1 **pas** -1 **faire**

si *i* divise *n* et *m* **alors retour** (*i*)

pgcd (21,9) = 3 [21 = 3x7, 9 = 3x3] 7 étapes

Algorithme d 'Euclide (300 avant J-C)

division entière $n = q.m + r$, $0 \leq r < m$

propriété : pgcd (*n,m*) = pgcd (*m,r)*

pgcd(*n,m*) = **si** *m* = o **alors** *n*

sinon pgcd(*m*, *n* mod *m*)

Preuve : si *n*<*m* première étape = échange, sinon propriété arithmétique

Terminaison : *m* décroît à chaque étape (strictement)

pgcd(9,21) = pgcd(21,9) = pgcd(9,3) = pgcd(3,0) = 3 3 étapes

Complexité

Algorithme 1

T(Algo1, (n,m)) = O(min(m,n))

moins de 2.min(n,m) divisions entières

Algorithme d'Euclide

Théorème de Lamé (1845) : ($n \geq m$)

Si l 'algorithme d 'Euclide nécessite k étapes pour calculer pgcd (n,m) on a $n \geq m \geq Fib_k$

[$Fib_0 = 0$, $Fib_1 = 1$, $Fib_k = Fib_{k-1} + Fib_{k-2}$, pour $k>1$]

pgcd(21,13) = pgcd(13,8) = pgcd(8,5) = pgcd(5,3)
= pgcd(3,2) = pgcd(2,1) = pgcd(1,0) = 1

T(Euclide, (n,m)) = O(log min(n,m))

Version récursive

```
function  PGCD(n,m: integer): integer;
{n>=0, m>=0}
begin
     if m = 0 then return (n)
     else return (PGCD(m, n mod m));
end;
```

Version itérative :

```
function  PGCD(n,m: integer): integer;
{n>=0, m>=0}
     var temp;
begin
     while m > 0 do begin
          temp := m;
          m := n mod m;
          n := temp;
     end;
     return (n);
end;
```

Version récursive :

```
int PGCD(int n, int m){
/* n>=0, m>=0 */
     if (m == o) return (n);
     else return ( PGCD(m, n % m));
}
```

Version itérative :

```
int PGCD(int n, int m){
/* n>=0, m>=0 */
     int temp;
     while (m > o){
          temp = m;
          m = n % m;
          n = temp;
     }
     return (n);
}
```

Récursivité terminale

```
fonction  F(x) ;
début
    si  C(x)  alors  {I;  retour (y); }
    sinon  {J;  retour (F(z)); }
fin.
```

Élimination (par copier-coller) :

```
fonction F(x) ;
début
    tant que non C(x) faire
        {J;  x := z; }
    {I;  retour (y); }
fin.
```

Factorielle

```
fonction FAC(n) ;
début
        si n=0 alors retour (1)
        sinon retour (n* FAC(n-1)) ;
fin.
```

RÉCURSIVE

```
fonction FAC'(n, m)
début
        si n=0 alors retour (m)
        sinon retour (FAC'(n-1, n*m)) ;
fin.
```

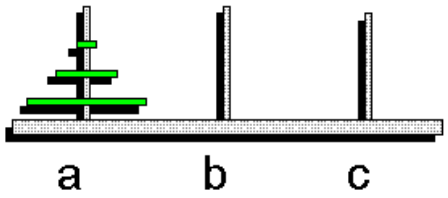
RÉCURSIVITÉ TERMINALE

```
fonction FAC(n) ;
début
    m : = 1 ;
    tant que n > 0 faire
    { m := n* m ; n := n-1 ; }
    retour (m) ;
    fin.
```

ITÉRATIVE

```
fonction FAC'(n, m) ;
début
        tant que n > 0 faire
        { m : = n* m ; n : = n-1 ; }
        retour (m) ;
fin.
```

Tours de Hanoï



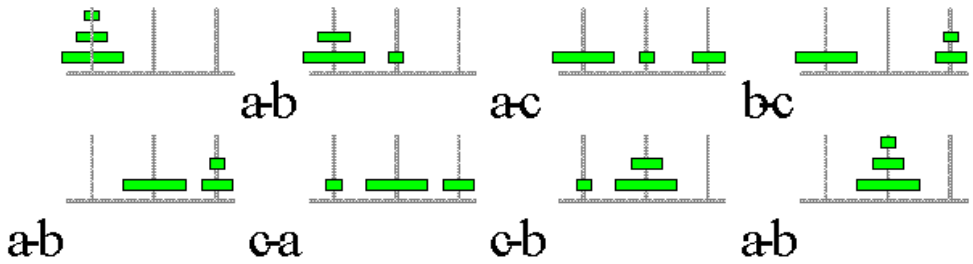
RÈGLES (opérations élémentaires)

1. déplacer un disque à la fois d'un bâton sur un autre
2. ne jamais mettre un disque sur un plus petit

BUT

transférer la pile de disques de a vers b

EXEMPLE (3 disques)



Spécification

Calculer H(*n, x, y, z*) = suite des coups pour transférer *n* disques de *x* vers *y* avec le bâton intermédiaire *z* ($n \geq 1$).

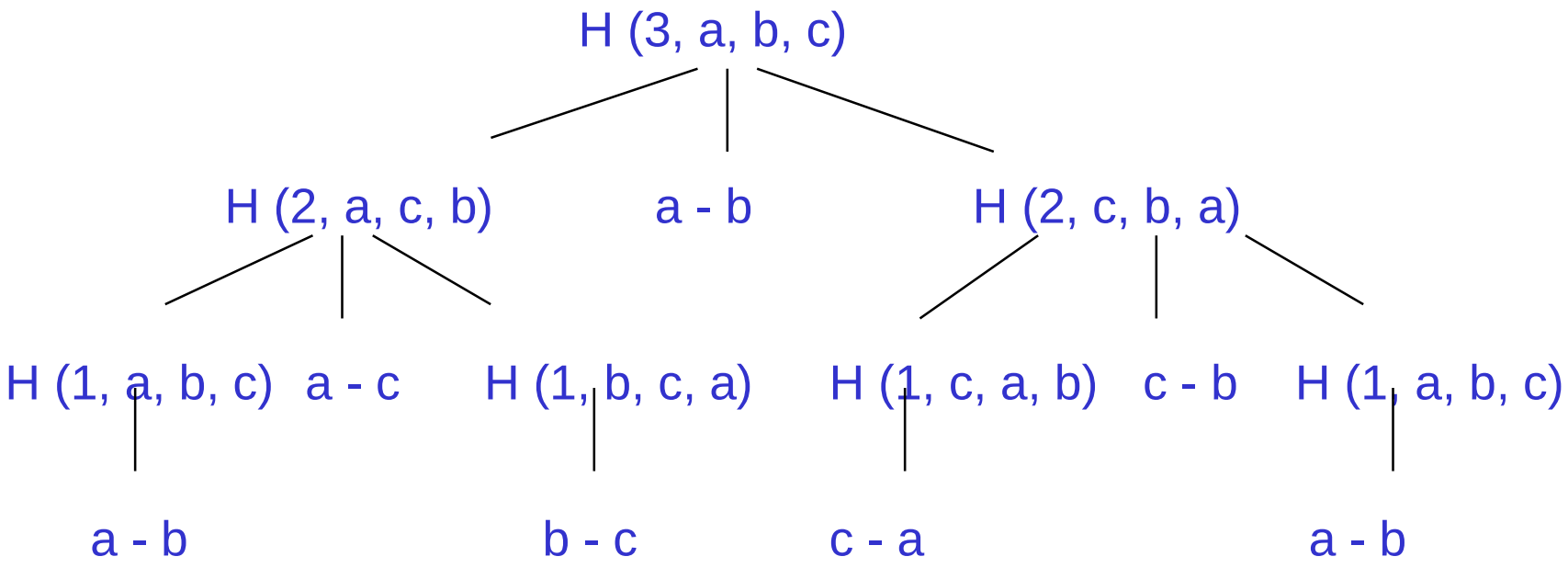
Relations

$$H(n, x, y, z) = \begin{cases} x\text{-}y \quad \text{si } n = 1 \\ H(n\text{-}1, x, z, y)\ .\ x\text{-}y\ .\ H(n\text{-}1, z, y, x) \quad \text{si } n > 1 \end{cases}$$

Algorithme ($n \geq 1$)

fonction H(*n, x, y, z*) ;
début
si *n* = 1 **alors** écrire (*x*-*y*) ;
sinon { H(*n*-1, *x, z, y*); écrire(*x*-*y*); H(*n*-1, *z, y, x*); }
fin.

Exécution



Temps d ’exécution

T(H, n disques) = O(2^n)

Longueur(H(n, a, b, c)) = 2^n -1

H est optimal (parmi les algorithmes qui respectent les règles)

Jeu des chiffres

Spécification

Données : $w_1, w_2, \ldots, w_n$ entiers > 0 ; s

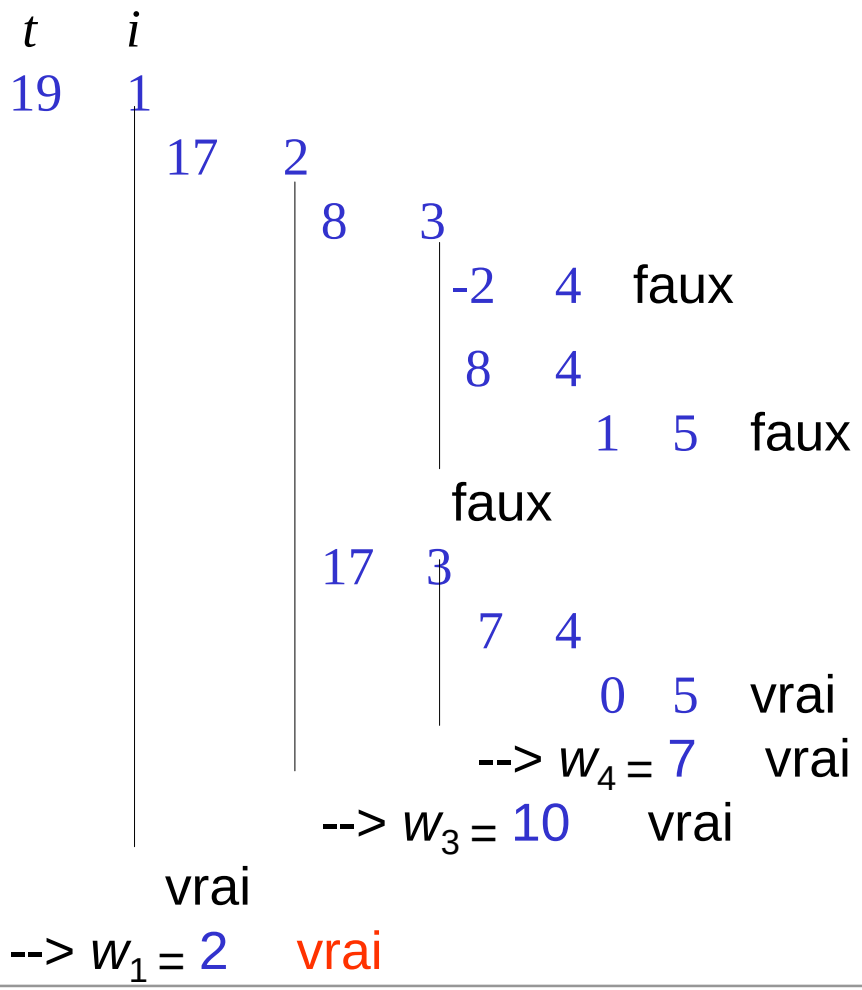
Résultat : une partie I de $(1, 2, \ldots, n)$ telle que $\sum_{i \in I} w_i = s$; $\varnothing$ sinon

Algorithme récursif

fonction CHIFFRES (*t, i*) ;
/* prend la valeur vrai ssi il existe une partie de (*i, i*+1, ..., *n*)
qui donne la somme *t* ; écrit les nombres correspondants */
début
si (*t* = 0) **alors retour** (vrai)
sinon si (*t* < 0) **ou** (*i* > *n*) **alors retour** (faux)
sinon si CHIFFRES (*t* - w_i, *i*+1) **alors**
{ écrire (w_i) ; **retour** (vrai) ; }
sinon retour (CHIFFRES (*t, i*+1)) ;
fin .

Exécution

sur $\begin{cases} w_1 = 2,\ w_2 = 9,\ w_3 = 10,\ w_4 = 7 \\ s = 19 \end{cases}$



Complexité en temps = $O(2^n)$

Preuve : on vérifie que si $s > \sum w_i$ l'algorithme parcourt tous les sous-ensembles de (1, 2, …, n). Il y en a 2^n.