Algorithmique et structures de données linéaires
HLIN301

P. Janssen : philippe.janssen@lirmm.fr

Université de Montpellier - Faculté des Sciences

Organisation de l'UE

Heures

- 15 heures de cours
- 22,5 heures de TD
- 9 heures de TP

Contrôle des connaissances

La note finale de l'UE est calculée à partir de 2 notes

- une note de contrôle continu (CC) composée de 2 notes :
  - une note de contrôle en cours (début novembre)
  - une note de contrôle en TD basée sur des interrogations, l'activité en TD et TP, la présence en TD/TP.
- une note d'examen (Exam) ($1^{ère}$ session en Janvier, $2^{ème}$ session en juin)

selon la formule :
note UE = Max(note_Exam, (2*note_CC + 3*note_Exam)/5))

Objectifs

Analyse et conception d'algorithmes et Étude de Structures de données

- Preuve d'algorithmes
- Complexité des algorithmes
- Structures de données simples : listes chaînées, piles, files.
- Structures de données arborescentes : arbres binaires, tas binaires.
- Algorithmes de tri

Livres

- *Introduction à l'algorithmique*,
  T. Cormen, C. Leiserson, R. Rivest ;
  Ed. Dunod

Rappels du langage algorithmique

Schéma d'algorithme

Algorithme : nom(paramètres)
Données : description des paramètres-donnée de l'algorithme
Résultat : description du résultat : valeur renvoyée par l'algorithme
(ou paramètres modifiés par l'algorithme)
Déclaration des variables;
début
| Partie instructions
fin algorithme

Types

Type = Domaine de valeurs + opérations

- Réels : opérations : +, -, *, /
- Entiers : opérations : +, -, *,
  division euclidienne : quo, mod
- Booléens : opérations : non, et, ou
  opérateurs de comparaisons : <, >, =, $\neq$
- Tableaux : séquence de longueur fixe, d'éléments de même type
  opérateur d'accès à un élément : [ ]
  opération taille renvoie le nombre d'éléments d'un tableau.
  ex : `T` Tableau de 5 entiers,
  `T[2]` désigne le 3ème élément de `T`,
  `taille(T)` vaut 5

Variables

Variable = Nom + Type + éventuellement valeur
La déclaration d'une variable consiste à donner son nom et son type
Pas de valeur par défaut

Environnement

Ensemble d'associations *(Nom, Valeur)*
L'environnement est modifié par l'affection d'une valeur à un nom

Expressions

- *Constante*
- *Nom de Variable*
- *Nom de Variable Tableau*[*Expression*]
- *Expression opération Expression*
- *Opération(Expression)*

L'évaluation dans un environnement d'une expression composée de sous-expressions provoque l'évaluation de toutes les sous-expressions puis l'application de l'opération à ces valeurs.
Exception : l'évaluation des expressions booléennes est *paresseuse* :

Valeur(A et B)	=	si valeur(A)=Faux alors Faux sinon valeur(B)
Valeur(A ou B)	=	si valeur(A)=Vrai alors Vrai sinon valeur(B)

Conséquence : *B* n'est pas toujours évalué ;
ex :

Instructions, Structures de contrôle

- Affectation :←
- L'instruction `Renvoyer(Expression)` correspond à la directive " `Le résultat est Expression`" utilisée en première année.
- Conditionnelles :

si `Cond1` **alors**
| Inst1
sinon si `Cond2` **alors**
| Inst2
sinon
| Inst3
fin si

où `Cond1` et `Cond2` sont des expressions à valeur booléenne
Les parties `sinon` et `sinonSi` sont optionnelles.

Structures Répétitives

- **pour** `K` **de** `E1` **à** `E2` **faire**
  | Inst
  fin pour

  `K` est l'indice de boucle de type entier
  `E1, E2` 2 expressions à valeur entière.
  - `E1,E2` sont évaluées une fois pour toute avant l'itération.
  - le corps de l'itération ne peut pas modifier la valeur de la variable `K`
  - en sortie de l'itération `Pour` la variable de contrôle `K` n'a pas de valeur.

- **tant que** `Cond` **faire**
  | Inst
  fin tq

  où `Cond` est une expression à valeur booléenne

  Contrairement au `Tant que` , le nombre d'itérations de la structure `Pour` est indépendant de l'instruction itérée `Inst`.

Modes de passage des Paramètre(s) et Résultat(s)

Un algorithme décrit un calcul permettant d'obtenir un résultat à partir de données.
Les **paramètres formels** de l'algorithme correspondent aux données et résultats de l'algorithme.
Un paramètre formel peut être :

- une donnée de l'algorithme, précédé par la lettre d
- un résultat de l'algorithme, précédé par la lettre r
- une donnée et un résultat de l'algorithme, précédé par les lettres dr

Pour appliquer un algorithme, il faut indiquer son nom et ses **arguments** ("valeurs associées aux paramètres formels ").

Modes de passage des Paramètre(s) et Résultat(s)

Algorithme de style fonction

Tous les paramètres sont des données,
Le résultat est déterminé par l'instruction **renvoyer**

Algorithme : nom de l'algo(nom, nature et type des paramètres formels) : type du résultat
Données : description des paramètres données
Résultat : description du résultat
Déclaration des variables;
début
| Partie instructions ;
| **renvoyer** Expression ;
fin algorithme

Le corps de l'algorithme doit contenir l'instruction **renvoyer** `Expression`.
Le résultat calculé par l'algorithme est la valeur de l'expression `Expression`.
L'application d'un algorithme de style fonction correspond à une expression dont la valeur est le résultat calculé par l'algorithme.

Exemple d'algorithme de style fonction

Algorithme : noteUE(d NoteExam : Réel, d NotePartiel : Réel) : Réel
Données : NoteExam et NotePartiel sont 2 réels de l'intervalle [0 ;20]
Résultat : Renvoie la note finale du module sur 20 obtenue à partir des notes d'examen et de partiel
Variables Res : Réel;
début
|
|
|
|
fin algorithme

Modes de passage des Paramètre(s) et Résultat(s)

Algorithme de style procédure

Données et résultats sont des paramètres de l'algorithme.

Algorithme : nom de l'algo(nom, nature et type des paramètres formels)
Données : description des paramètres données
Résultat : description des paramètres résultat
Déclaration des variables;
début
| Partie instructions
fin algorithme

Le corps de l'algorithme ne doit pas contenir l'instruction **renvoyer**
L'application d'un algorithme de style procédure correspond à une instruction modifiant les valeurs des arguments associés aux paramètres résultats (et données–résultats).

Exemple d'algorithme de style procédure

Algorithme : échanger(dr a : Entier, dr b : Entier)
Données : a et b 2 entiers ; `(a=x, b=y)`
Résultat : Echange les valeurs de a et b ; `(a=y , b=x)`
Variables c : Entier;
début
|
|
fin algorithme

Exemple avec toutes les formes de paramètre formel

Algorithme : CalculFinal(dr NoteUE : Réel, d PtJury : Réel, r ECTS : Entier)
Données : `NoteUE` $\in$ [0 ;20] et `PtJury` $\geqslant$ 0 ; (`NoteUE=n,PtJury=p`)
Résultat : Ajoute les Points Jury à la note de l'UE ; Si la note obtenue est inférieure à 10 le nombre d'ECTS est 0, sinon il vaut 5 ; (`NoteUE=p+n; Si p+n <10 alors ECTS=0 sinon ECTS=5`)
début
|
|
fin algorithme

Modes de passage des Paramètre(s) et Résultat(s)

Correspondance entre paramètre formel et argument lors de l'application d'un algorithme

- Avant l'exécution du corps de l'algorithme :
  Pour chaque paramètre donnée et donnée-résultat, la valeur de l'argument est affectée au paramètre formel correspondant.
- Exécution du Corps de l'algorithme
- Après l'exécution du corps de l'algorithme :
  Pour chaque paramètre résultat et donnée-résultat, la valeur du paramètre formel est affectée à l'argument correspondant.

Algorithme : monAlgo(d A : ..., dr B : ..., r C : ...)
Données : A et B
Résultat : B et C
début
| Corps de l'algorithme
fin algorithme

monAlgo(E1,E2,E3)

A←E1 ;B←E2;

début
| Corps de l'algorithme
fin algorithme

E2←B ;E3←C;

Modes de passage des Paramètre(s) et Résultat(s)

Le principe de passage de paramètre décrit n'est qu'un modèle. Les mécanismes réels dépendent des langages et compilateurs.

En C, Scheme, Maple, Caml tous les paramètres sont des paramètres-données, donc non modifiables. Cependant en C, il est possible d'avoir une adresse comme paramètre. Dans ce cas, l'adresse-paramètre ne peut être modifiée, mais son contenu si. On peut ainsi par effet de bord simuler un paramètre résultat et/ou donnée-résultat.

exemple : traduction de l'algorithme `échanger` en langage C

```
void echanger(int *a,int *b)
{ int c; c=*a; *a=*b; *b=c; }
```

```
...
echanger(&X,&Y);
...
```


Modes de passage des Paramètre(s) et Résultat(s)

Contraintes liant paramètres et arguments

	Paramètre donnée	Paramètre résultat	paramètre donnée-résultat
Argument	Expression quelconque	Expression de variable	Expression de variable
Avant l'appel	doit avoir une valeur		doit avoir une valeur
Après l'appel	même valeur qu'avant		
Dans le corps de l'algorithme	ne doit pas être modifié	doit être modifié	

Expression de variable : expression pouvant être en partie gauche d'affection
Exemple : `A, T[i+1]`
Contre-exemple : `3, A+1`

Mécanisme de passage de paramètres en C++

Par défaut, le passage de paramètre en C++ est comme en C, un passage par valeur. Il existe également un passage de paramètre **par référence**, qui correspond au mode de paramètre *donnée/résultat* de notre langage algorithmique.

Traduction en C++ de la fonction CalculFinal

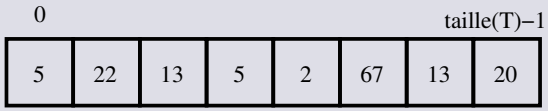
```
void CalculFinal(float& NoteUE, float PtJury, int& ECTS)
{
NoteUE = NoteUE + PtJury;
if(NoteUE < 10) ECTS=0;}
else ECTS=5;}
}

int main() {
int credit; float ue,pj;
ue=9.5; pj=0.5;
CalculFinal(ue, pj, credit);
... }
```

Algorithmes de Recherche dans un tableau

Le problème

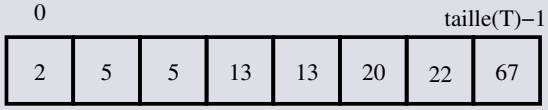
Algorithme : Recherche(d T : tableau de *type*, d e : *type*) : Booléen
Données : T un tableau, e
Résultat : Renvoie Vrai si e est élément de T, renvoie Faux sinon



Algorithmes de Recherche dans un tableau

Cas du tableau trié

Algorithme : Recherche(d T : tableau de *type*, d e : *type*) : Booléen
Données : T un tableau trié ↗, e
Résultat : Renvoie Vrai si e est élément de T, renvoie Faux sinon



Algorithmes de Recherche dans un tableau

Recherche Séquentielle

Algorithme : Recherche(d T : tableau, d e) : Booléen
Données : T un tableau, e
Résultat : Renvoie Vrai si e est élément de T, renvoie Faux sinon
variables : $i \in \mathbb{N}$;
début
  $i \leftarrow 0$;
  tant que **faire**
    $i \leftarrow i + 1$
  fin tq

fin algorithme

Algorithmes de Recherche dans un tableau

Recherche Séquentielle

Algorithme : Recherche(d T : tableau, d e) : Booléen
Données : T un tableau trié ↗, e
Résultat : Renvoie Vrai si e est élément de T, renvoie Faux sinon
variables : $i \in \mathbb{N}$;
début
  $i \leftarrow 0$;
  tant que **faire**
    $i \leftarrow i + 1$
  fin tq

  renvoyer
fin algorithme

Algorithmes de Recherche dichotomique

Algorithme : RechercheDicho(d T : Tableau, d e) : Booléen
Données : T trié $\nearrow$ et e
Résultat : renvoie Vrai si e est élément de T, Faux sinon
Variables : Deb, Fin, $Mil \in \mathbb{N}$, $Trouve$: $Booleen$
début
  $Deb \leftarrow 0$; $Fin \leftarrow taille(T) - 1$; $Trouve \leftarrow Faux$;
  tant que **faire**
    $Mil \leftarrow (Deb + Fin)$ *quo* 2 ;
    si $T[Mil] = e$ **alors**
    sinon si $e < T[Mil]$ **alors**
    sinon
    fin si
  fin tq
  renvoyer $Trouve$
fin algorithme

Algorithmes de Recherche dichotomique

Version récursive

Algorithme : RechDicho2(d T ,d e, d Deb : Entier, d Fin : Entier) : Booléen
Données : T trié $\nearrow$, e
Résultat : renvoie Faux si $\forall i \in [Deb \dots Fin]\ T[i] \neq e$ Vrai sinon
Variables : $Mil \in \mathbb{N}$
début
  si $Deb > Fin$ **alors renvoyer** `Faux`
  sinon
    $Mil \leftarrow (Deb + Fin)$ *quo* 2 ;
    si $T[Mil] = e$ **alors**
    sinon si $e < T[Mil]$ **alors**
    sinon
    fin si
  fin si
fin algorithme

Pour savoir si e appartient au tableau T on applique

Analyse d'un algorithme

- **Preuve de l'algorithme**
  Un algorithme est correct si pour toute valeur des paramètres-donnée vérifiant les spécifications des données :
  - l'exécution de la partie instructions s'arrête
  - le résultat calculé par l'algorithme correspond à la solution du problème, vérifie les spécifications du résultat.
- **Évaluation du coût de l'algorithme**

Preuve de l'arrêt d'un algorithme

Le problème se pose pour les itérations `Tant que`
(et pour la `récursivité`).

Montrer que le nombre d'itérations est fini : exhiber une expression
- dont la valeur décroît (ou croît) strictement à chaque itération
- montrer que cette valeur ne peut pas décroître (ou croître) indéfiniment.

Ex : expression strictement décroissante à valeur entière ($\in \mathbb{N}$)

Exemple

Algorithme : algo1(d A, d B, r Z)
Données : $A, B \in \mathbb{N}$
Résultat : $Z = ?$

Variables : X et $Y \in \mathbb{N}$
début
  $X \leftarrow A$; $Y \leftarrow B$; $Z \leftarrow 0$;
  tant que $X \neq 0$ **faire**
    si X `est impair` **alors**
      $Z \leftarrow Z + Y$; $X \leftarrow X - 1$;
    fin si
    $Y \leftarrow 2 * Y$; $X \leftarrow X/2$;
  fin tq
fin algorithme

Arrêt de l'algorithme

Exemple

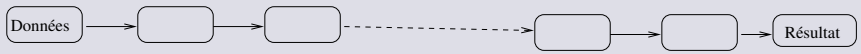
Algorithme : rechDicho(d T : tableau ,d e,r present : Bool, r ind : entier)
Données : T tableau trié $\nearrow$, e
Résultat : Si $e \notin T$ alors $present = Faux$ sinon $present = Vrai$ et ind est l'indice maximum de e dans T ($ind = max\{i \in [1..taille(T)[\ : \ T[i] = e\}$)
Variables : Deb, Fin, $Mil \in \mathbb{N}$
début
  $Deb \leftarrow 0$; $Fin \leftarrow taille(T) - 1$;
  tant que $Deb \leq Fin$ **faire**
1 $Mil \leftarrow (Deb + Fin)$ *quo* 2 ;
2 **si** $T[Mil] \leq e$ **alors** $Deb \leftarrow Mil + 1$
    sinon
      $Fin \leftarrow Mil - 1$
    fin si
  fin tq
  si . . . **alors** $present \leftarrow Vrai$; $ind \leftarrow ...$
  sinon
    $present \leftarrow Faux$
  fin si
fin algorithme

Arrêt de l'algorithme RechercheDichotomique

Preuve du résultat d'un algorithme

Vérifier que le résultat correspond aux spécifications de l'algorithme

- L'algorithme décrit un enchaînement d'états à parcourir pour passer de l'état de départ correspondant aux données, à l'état d'arrivée correspondant au résultat.
- Etat : valeurs des variables (environnement)
- Pour montrer qu'un tel cheminement est correct, on décrit les états intermédiaires.
- Pour décrire ces états on donne des propriétés vérifiées par les variables (assertions).
- Problème avec les itérations



Invariants de boucle

Définition (Invariant)

Invariant d'une itération : propriété vérifiée à chaque itération par les valeurs de certaines variables.

Les valeurs des variables peuvent changer à chaque itération, mais la propriété est « invariante ».
Intérêt : la propriété « invariante » est vérifée en fin d'itération

Exemple

Algorithme : Recherche(d T : tableau, d e) : Booléen
Données : `T` un tableau trié ↗, `e`
Résultat : Renvoie Vrai si `e` est élément de `T`, renvoie Faux sinon
variables : $i \in \mathbb{N}$;
début
  $i \leftarrow 0$;
  tant que $i < taille(T)$ `et` $T[i] < e$ **faire**
    $i \leftarrow i + 1$
  fin tq

  renvoyer $(i < taille(T)$ `et` $T[i] = e)$
fin algorithme

Preuve d'Invariant

début
1
  tant que `Cond` **faire**
2
    Inst ;
3
  fin tq
4
fin algorithme

Pour montrer que `Rel` est invariante pour une itération `Tant que` on utilise le schéma de preuve par récurrence sur le nombre d'itérations : il faut montrer que :

- `Rel` est vérifiée avant la première itération (ligne 1)
- si à la ligne 2 `Cond` et `Rel` sont vérifiées, alors `Rel` est vraie à la ligne 3.

Utilisation d'Invariant

début
  tant que `Cond` **faire**
    `/* Rel */`
    Inst ;
  fin tq
4
fin algorithme

- Si `Rel` est une propriété invariante pour une itération
- alors en fin d'itération (ligne 4), `non Cond` et `Rel` sont vérifiées.

Que calcule algo1 ?

Exemple

Algorithme : algo1(d A,d B,r Z)
Données : $A, B \in \mathbb{N}$
Résultat : $Z = ?$

Variables : X et $Y \in \mathbb{N}$
début
  $X \leftarrow A$; $Y \leftarrow B$; $Z \leftarrow 0$;
  tant que $X \neq 0$ **faire**
    si X `est impair` **alors**
      $Z \leftarrow Z + Y$; $X \leftarrow X - 1$;
    fin si
    $Y \leftarrow 2 * Y$; $X \leftarrow X/2$
  fin tq
fin algorithme

Un exemple

$A = 6, B = 5$.

Num Itér	A	B	X	Y	Z
0	6	5	6	5	0
1	6	5	3	10	0
2	6	5	1	20	10
3	6	5	0	40	30

Résultat

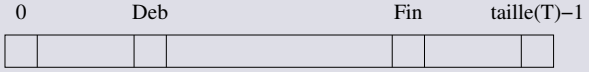
Z=

Invariant

Preuve de l'invariant

Invariant pour la recherche dichotomique

Algorithme : rechDicho(d T : tableau ,d e,r present : Bool, r ind : entier)
Données : T tableau trié $\nearrow$, e
Résultat : Si $e \notin T$ alors $present = Faux$ sinon $present = Vrai$ et ind est l'indice maximum de e dans T ($ind = max\{i \in [0..taille(T)[\ : T[i] = e\}$)
Variables : Deb, Fin, $Mil \in \mathbb{N}$
début
  $Deb \leftarrow 0$; $Fin \leftarrow taille(T) - 1$;
  tant que $Deb \leq Fin$ **faire**
    $Mil \leftarrow (Deb + Fin)$ *quo* 2 ;
    si $T[Mil] \leq e$ **alors** $Deb \leftarrow Mil + 1$
    sinon $Fin \leftarrow Mil - 1$
  fin tq
  si . . . **alors** $present \leftarrow Vrai$; $ind \leftarrow ...$
  sinon $present \leftarrow Faux$
fin algorithme



Preuve de l'invariant

En fin d'itération

Algorithme de recherche dichotomique complet

Algorithme : rechDicho(d T : tableau ,d e,r present : Bool, r ind : entier)
Données : T tableau trié $\nearrow$, e
Résultat : Si $e \notin T$ alors $present = Faux$, sinon $present = Vrai$ et ind est l'indice maximum de e dans T
Variables : Deb, Fin, $Mil \in \mathbb{N}$
début
  $Deb \leftarrow 0$; $Fin \leftarrow taille(T) - 1$;
  tant que $Deb \leq Fin$ **faire**
    $Mil \leftarrow (Deb + Fin)$ *quo* 2 ;
    si $T[Mil] \leq e$ **alors** $Deb \leftarrow Mil + 1$
    sinon
      $Fin \leftarrow Mil - 1$
    fin si
  fin tq
  si **alors** $present \leftarrow Vrai$;
  sinon
    $present \leftarrow Faux$
  fin si
fin algorithme

Invariant pour les itérations `Pour`

Même principe. Pour régler le problème des *instructions cachées*, on traduit l'itération `Pour` en itération `Tantque`.

Exemple

Algorithme : rechercheSequentielle(d T : tableau , d e, r present : Bool)
Données : T tableau, e
Résultat : $present = Vrai$ si $e \in T$, $present = Faux$ sinon
Variable $i \in \mathbb{N}$
début
  $present \leftarrow Faux$;
  pour i `de 0 à` $taille(T) - 1$ **faire**
    si $T[i] = e$ **alors** $present \leftarrow Vrai$
  fin pour
fin algorithme

Version Pour

Données : T, e
Résultat : $present = (e \in T)$
Variable $i \in \mathbb{N}$
début
  $present \leftarrow Faux$;
  pour i `de 0 à taille(T)-1`
  faire
    si $T[i] = e$ **alors**
      $present \leftarrow Vrai$
    fin si
  fin pour
fin algorithme

Version Tant que

Données : T, e
Résultat : $present = (e \in T)$
Variable $i \in \mathbb{N}$
début
  $present \leftarrow Faux$; $i \leftarrow 0$;
  tant que $i \leqslant taille(T) - 1$ **faire**
    si $T[i] = e$ **alors**
      $present \leftarrow Vrai$
    fin si
    $i \leftarrow i + 1$;
  fin tq
fin algorithme

Invariant

$present = Vrai$ si et seulement si

Complexité d'un algorithme

Objectif

- Définir des critères pour la comparaison d'algorithmes ou pour évaluer la qualité d'un algorithme.
- Critères : temps, mémoire utilisés pour le calcul décrit par l'algorithme.
- Évaluation du temps de calcul d'un algorithme : **nombre d'opérations élémentaires** exécutées par l'algorithme en fonction de la **taille de la donnée** et **dans le plus mauvais des cas**.

opération élémentaire

Définition

Opération élémentaire : opération dont le temps d'exécution est borné par une constante (indépendant de la donnée).

Exemples

- Affectation de variables simples
- Accés à un élément de tableau
- Opérations booléennes, comparaisons
- Opérations arithmétiques (lorsque opérandes et résultats sont bornés)

Contre-Exemples

- Initialisation, comparaison, affectation de tableau
- Algorithme
- Arithmétique sur grand nombre, exponentiation

taille de la donnée

Définition

Taille de la donnée : taille du codage (nombre de mots mémoire) de l'ensemble des paramètres-donnée.

Simplification

- pour un booléen, caractère, nombre borné : 1
- pour un tableau ou un ensemble : nombre d'éléments (si la valeur des éléments est bornée)
- pour un arbre : nombre de noeuds et hauteur
- (pour un graphe : nombre de sommets et nombre d'arêtes)
- (pour un problème de « nature numérique » (ex : test de primalité) : taille du codage des nombres)

Types d'analyse

Pour une même taille de donnée le nombre d'opérations élémentaires exécutées par 1 algorithme peut varier selon les données ; on peut effectuer plusieurs types d'analyse :

- dans le plus mauvais des cas : on compte $t^A_{max}(n)$, le nombre maximum d'opérations élémentaires exécutées par l'algorithme `A` pour les données de taille n.
- **en moyenne** : $t^A_{moy}(n)$: moyenne du nombre d'opérations élémentaires exécutées par l'algorithme `A` pour toutes les données de taille n

Exemple

Algorithme : rechSeq(**d** T, **d** e) : Bool
Données : T tableau trié $\nearrow$ et e
Résultat : Renvoie Vrai ssi $e \in T$
Variables : $i \in \mathbb{N}$
début
  $i \leftarrow 0$;
  tant que $i < taille(T)$ `et` $T[i] < e$ **faire**
    $i \leftarrow i + 1$;
  fin tq
  si $i \geqslant taille(T)$ `ou` $T[i] > e$ **alors**
    renvoyer Faux
  sinon
    renvoyer Vrai
  fin si
fin algorithme

- Taille du problème :
- Meilleur cas :
  affectations
  additions
  accès tableau
  comparaisons
  opérations bool.
  renvoyer
  ---
  en tout
- Pire cas :

  affectations
  additions
  accès tableau
  comparaisons
  opérations bool.
  renvoyer

  $t^{RS}_{max}(N+1) =$
  $t^{RS}_{max}(N) =$

Ordre de grandeur

Rôle des constantes

Dans l'expression de $t_{max}(n)$ les constantes ont :

- peu de sens : par exemple l'exécution d'une opération booléenne est plus rapide que l'accès à un élément de tableau.
- peu d'importance : Si pour un même problème 2 algorithmes `A` et `B` ont les complexités $t^A_{max}(n) = n^2 + 1$ et $t^B_{max}(n) = 4.n + 3$, on préfèrera l'algorithme `B` même si $t^A_{max}(n) < t^B_{max}(n)$ pour $n < 5$.

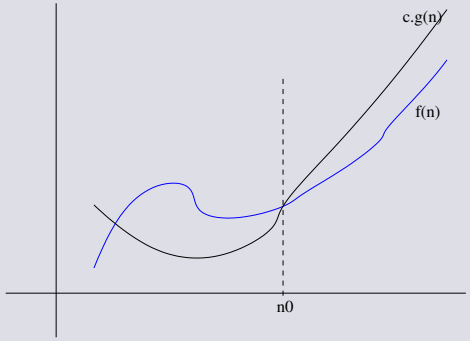
Ce qui nous intéresse

Comportement asymptotique de $t_{max}(n)$.

Ordre de grandeur

Définition

2 fonctions $f, g : \mathbb{R} \to \mathbb{R}$
$f(n) \in O(g(n))$ si $\exists c > 0, \exists n_0$ tels que $\forall n > n_0, f(n) \leqslant c.g(n)$
f est dominé asymptotiquement par g.
On note également $f = O(g)$ ou encore $f(n) = O(g(n))$.



Exemple

- $6n + 6 \in O(n)$. Noté aussi $6n + 6 = O(n)$. (par exemple $c = 12$ et $n_0 = 1$)
- $(n + 1)^2 = O(n^2)$ (par exemple $c = 3, n_0 = 1$)
- $4n^3 + 10n^2 + 8 = O(n^3)$.

Simplification des calculs

Propriétés

$O(f).O(g) = O(f.g)$
$O(f)+O(g) = O(f+g) = O(max(f,g))$ [1]

Exemple

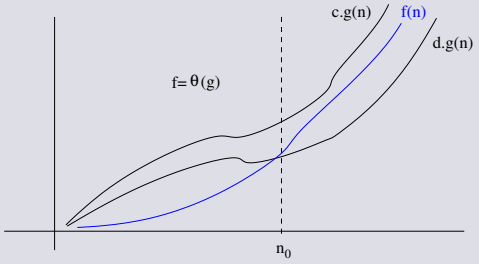
début
  $i \leftarrow 0$;
  tant que $i < N$ et $T[i] < e$ **faire**
    $i \leftarrow i+1$;
  fin tq
  si $i \geqslant N$ ou $T[i] > e$ **alors**
    renvoyer Faux
  sinon
    renvoyer Vrai
  fin si
fin algorithme

Notation θ

On a $6n = O(n)$ mais aussi $6n = O(n^2)$ car $O(n) \subset O(n^2)$.

Définition

2 fonctions $f, g : \mathbb{R} \to \mathbb{R}$
$f(n) \in \theta(g(n))$ si $\exists c, d \in \mathbb{R}^*, \exists n_0$ tels que $\forall n > n_0, d.g(n) \leq f(n) \leq c.g(n)$
On note également $f = \theta(g)$.



$t^{RS}_{max}(N) = 6N = \theta(N)$

Principaux ordres de grandeurs

- $\theta(1)$: constant
- $\theta(log_2 n)$: logarithmique
- $\theta(n)$: linéaire
- $\theta(nlog_2 n)$
- $\theta(n^2)$: polynomial
- $\theta(n^3)$
- $\theta(2^n)$: exponentiel

Des chiffres

Exemple

On exécute des algorithmes de complexité différentes sur une machine 1 Ghz, exécutant une opération élémentaire en $1\eta s$ ($10^{-9} s$) :

	$log_2 n$	n	$n.log_2 n$	n^2	n^3	2^n
10^2						
10^3						
10^4						
10^5						
10^6						
10^7						

Algorithme : rechDicho(**d** T,**d** e) : Bool
Données : T tableau trié $\nearrow$, e
Résultat : Renvoie $e \in T$
Variables : Deb, Fin, $Mil \in \mathbb{N}$
début
  $Deb \leftarrow 0$; $Fin \leftarrow taille(T) - 1$;
  tant que $Deb \leq Fin$ **faire**
    $Mil \leftarrow (Deb + Fin)$ *quo* 2 ;
    si $T[Mil] = e$ **alors renvoyer** `Vrai`
    sinon si $T[Mil] \leq e$ **alors**
      $Deb \leftarrow Mil + 1$
    sinon
      $Fin \leftarrow Mil - 1$
    fin si
  fin tq
  renvoyer `Faux`
fin algorithme

Preuve

- Arrêt :
  $Fin - Deb \geq -1$ et décroît strictement à chaque itération.
- Invariant :
  $\forall i \in [0 \ldots Deb[$
  $\forall j \in]Fin \ldots taille(T)[$
  $T[i] < e < T[j]$

début
  $Deb \leftarrow 0$; $Fin \leftarrow taille(T) - 1$;
  tant que $Deb \leq Fin$ **faire**
    $Mil \leftarrow (Deb + Fin)$ *quo* 2 ;
    si $T[Mil] = e$ **alors**
      renvoyer `Vrai`
    sinon si $T[Mil] \leq e$ **alors**
      $Deb \leftarrow Mil + 1$
    sinon
      $Fin \leftarrow Mil - 1$
    fin si
  fin tq
  renvoyer `Faux`
fin algorithme

Analyse de la complexité

- taille du problème : $N + 1$
  $(N = taille(T))$
- pire des cas :
- complexité d'une itération :
- la complexité de l'algo est

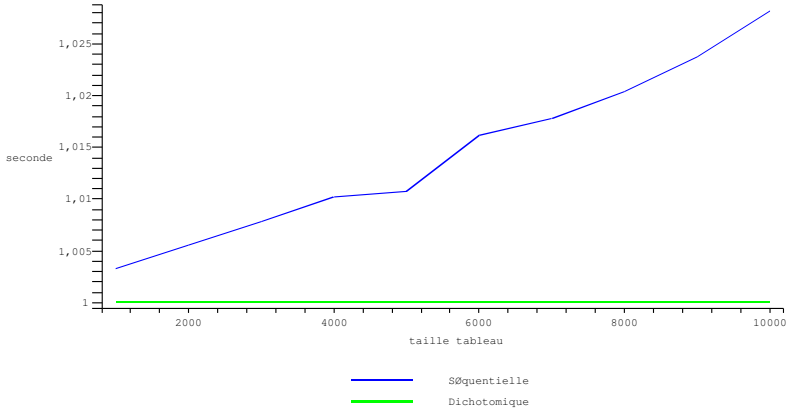
Calcul du nombre maximum d'itérations

- À chaque itération $Fin - Deb$ est divisé par 2. Soit Nbe le nombre d'éléments entre les indices Deb et Fin.

Améliorer un algorithme : changer sa classe de complexité !

Ex : Recherche d'un élément dans un tableau trié

Moyenne pour 100 tirages aléatoires



Exemple : multiplication de deux entiers par additions

Mesure de la complexité

On compte le nombre d'additions en fonction de la valeur des opérandes

Algorithme par additions successives

Algorithme : Mult1(**d** A, **d** B, **r** Z)
Données : $A, B \in \mathbb{N}$
Résultat : $Z = A \times B$
Variables : $X \in \mathbb{N}$
début
  $Z \leftarrow 0$;
  pour X `de 1 à` A **faire**
    $Z \leftarrow Z + B$;
  fin pour
fin algorithme

Calcul du nombre d'additions

Exemple : multiplication de deux entiers par additions

Multiplication « Russe »

Algorithme : mult2(**d** A, **d** B, **r** Z)
Données : $A, B \in \mathbb{N}$
Résultat : $Z = A \times B$
Variables : X et $Y \in \mathbb{N}$
début
  $X \leftarrow A$; $Y \leftarrow B$; $Z \leftarrow 0$;
  tant que $X \neq 0$ **faire**
    si X `est impair` **alors**
      $Z \leftarrow Z + Y$; $X \leftarrow X - 1$
    fin si
    $X \leftarrow X/2$; $Y \leftarrow Y + Y$;
  fin tq
fin algorithme

Calcul du nombre d'additions

Limites de l'analyse

- en prenant l'ordre de grandeur, on néglige les constantes qui peuvent être grandes !
  On peut faire une analyse plus fine.
- le plus mauvais des cas peut être très peu fréquent ; pour certains problèmes les algorithmes utilisés en pratique ne sont pas ceux de plus petite complexité dans le pire des cas.
  On peut faire une analyse en moyenne, mais il faut connaître la distribution des données

Tous ces cas sont rares.
L'analyse de la complexité asymptotique dans le pire des cas est une bonne mesure.