Algorithmique

Partie 1

Première NSI

Lycée Vaugelas, Chambéry

Plan du chapitre

Plan du chapitre

Introduction

Utilisation des ressources

Rechercher une valeur dans un tableau

Définition

Un algorithme est un ensemble d'étapes qui permettent d'accomplir une tâche.

Que peut-on attendre d'un algorithme informatique ?

Un algorithme résoud un problème

- proposer une solution juste/exacte
  à partir des données en entrée
  peu importe l'ordre de ces données
- utiliser les ressources de calcul de manière efficace

Que peut-on attendre d'un algorithme informatique ?

Un algorithme résoud un problème

- proposer une solution juste/exacte
  à partir des données en entrée
  peu importe l'ordre de ces données
- utiliser les ressources de calcul de manière efficace

On va commencer par le 2ème point ...

Plan du chapitre

Le **temps d'exécution** de l'algorithme est une première mesure de l'efficacité de l'algorithme

Si le GPS met 1h à calculer l'itinéraire, on sera souvent arrivé avant que le GPS fournisse le trajet.

Le temps d'exécution dépend de facteurs **internes** et **externes** à l'algorithme :

- **externes** : rapidité de l'ordinateur, langage de programmation utilisé, compilateur / interprétateur utilisé, ce qui tourne en même temps sur l'ordinateur, de la taille des entrées à traiter
- **internes** : comment le programme a été écrit

Comment évaluer / estimer la rapidité d'un algorithme ?

1) estimer la durée d'un algorithme en fonction de la taille des données en entrée (n = nb d'éléments à traiter)
2) comment évolue cette durée avec la taille des données en entrée (n) ?

Exercice 1

On considère 2 algorithmes dont l'exécution prend un certain nombre de cycles processeur :

nb de cycles processeur	ordre de grandeur qd n grand	taux de croissance retenu
$50n + 125$		
$20n^3 + 100n^2 + 212$		

Compléter le tableau

Exercice 1 : correction

nb de cycles processeur	ordre de grandeur qd n grand	taux de croissance retenu
$50n + 125$	$50\,n$	n
$20n^3 + 100n^2 + 212$	$20\,n^3$	n^3

Les facteurs 50 et 20 ont une importance mais dépendent tellement des paramètres externes à l'algorithme qu'on ne peut pas les comparer d'un ordinateur à un autre.
Soit A et B 2 implémentations différentes du même algorithme. Il se peut tout à fait que l'algo A soit plus rapide que l'algo B sur une certaine combinaison de machine, langage de programmation, compilateur / interprêteur alors qu'une autre combinaison favorise l'algo B. Donc on ne tient pas compte de ces facteurs, on ne considère que le taux de croissance !

Définition du logarithme en base 2

On note $\lg n$ le logarithme de n en base 2 qui est l'inverse de la fonction exponentielle 2^n :

$$n = 2^x \qquad \Leftrightarrow \qquad x = \lg n$$

Rmq : vous n'avez pas cette fonction directement sur votre calculatrice, mais vous avez le logarithme népérien $\ln x$ qui permet de calculer le logarithme en base 2 par la relation

$$\lg x = \frac{\ln x}{\ln 2}$$

Exercice 2

Compléter le tableau

x	$\lg x$
$1024 = 2^{10}$	
$1048576 = 2^{20}$	

Exercice 2 : correction

$$n = 2^x \qquad \Leftrightarrow \qquad x = \lg n$$

x	$\lg x$
$1024 = 2^{10}$	10
$1048576 = 2^{20}$	

Exercice 2 : correction

$$n = 2^x \qquad \Leftrightarrow \qquad x = \lg n$$

x	$\lg x$
$1024 = 2^{10}$	10
$1048576 = 2^{20}$	20

Exercice 3 et 4

exercice 3 à faire sur ordinateur
exercice 4 : énoncé sur `nsivaugelas.free.fr`

Distribuer le fichier `01_recherche_tableau_eleve.pdf`

- les algorithmes seront décrits en pseudo code plutôt que dans un langage particulier pour ne pas se focaliser sur des points de détails du langage et donc mettre l'accent sur les étapes importantes de l'algorithme.
- les tableaux possèdent une caractéristique importante : il faut le même temps pour accéder à n'importe quel élément du tableau (accès direct à la bonne case mémoire grâce à une table de hasahge)

Faire les exercices de la fiche distribuée.

Plan du chapitre

On va maintenant apprendre à caractériser le temps d'exécution d'un algorithme en fonction de la taille des entrées (n) !

Hypothèses simples sur la durée des opérations

On va maintenant apprendre à caractériser le temps d'exécution d'un algorithme en fonction de la taille des entrées (n) !

Hypothèses simples sur la durée des opérations
Chaque opération individuelle

- opération arithmétique (+,-,/,*)
- comparaison
- affectation
- indexation dans un tableau
- appel ou retour d'une fonction
- ...

se fait en un temps constant et indépendant de la taille n des entrées.

On va maintenant apprendre à caractériser le temps d'exécution d'un algorithme en fonction de la taille des entrées (n) !

Hypothèses simples sur la durée des opérations
Chaque opération individuelle

- opération arithmétique (+,-,/,*)
- comparaison
- affectation
- indexation dans un tableau
- appel ou retour d'une fonction
- ...

se fait en un temps constant et indépendant de la taille n des entrées.

chaque étape i se fait en un temps t_i qui est indépendant de n

Caractérisation du temps d'exécution de l'algo 1 : `recherche_lineaire(A,n,x)`

étape 1 : 1 fois t_1
étape 3 : 1 fois t_3
étape 2 : comparaison de i avec n : $(n+1)$ fois t'_2
incrémentation de i : $(n+1)$ fois t''_2
étape 2A : test `A[i]==x` : n fois t'_{2A}
`réponse` $\leftarrow$ `i` : on ne sait pas : t''_{2A} (*)

La dernière instruction (*) peut être exécutée 0 fois ($x \notin A$) ou jusqu'à n fois si $\forall$`i, A[i]=x`.

Caractérisation du temps d'exécution de l'algo 1 : `recherche_lineaire(A,n,x)`

étape 1 : 1 fois t_1
étape 3 : 1 fois t_3
étape 2 : comparaison de i avec n : $(n+1)$ fois t'_2
incrémentation de i : $(n+1)$ fois t''_2
étape 2A : test `A[i]==x` : n fois t'_{2A}
`réponse` $\leftarrow$ `i` : on ne sait pas : t''_{2A} (*)

$$\begin{array}{c} t_1+t_3 \\ +(n+1)t'_2 \\ +(n+1)t''_2 \\ +nt'_{2A} \\ +0\times t''_{2A} \end{array} \leq \text{temps d'exécution} \leq \begin{array}{c} t_1+t_3 \\ +(n+1)t'_2 \\ +(n+1)t''_2 \\ +nt'_{2A} \\ +n\times t''_{2A} \end{array}$$

Caractérisation du temps d'exécution de l'algo 1 : `recherche_lineaire(A,n,x)`

$$\begin{array}{c} t_1+t_3 \\ +(n+1)t_2' \\ +(n+1)t_2'' \\ +nt_{2A}' \\ +0\times t_{2A}'' \end{array} \leq \text{temps d'exécution} \leq \begin{array}{c} t_1+t_3 \\ +(n+1)t_2' \\ +(n+1)t_2'' \\ +nt_{2A}' \\ +n\times t_{2A}'' \end{array}$$

$$\begin{array}{c} (t_2'+t_2''+t_{2A}')\times n \\ +(t_1+t_2'+t_2''+t_3) \end{array} \leq \begin{array}{c} \text{temps} \\ \text{d'exécution} \end{array} \leq \begin{array}{c} (t_2'+t_2''+t_{2A}'+t_{2A}'')\times n \\ +(t_1+t_2'+t_2''+t_3) \end{array}$$

$$\begin{array}{c} (a)\times n \\ +(b) \end{array} \leq \begin{array}{c} \text{temps} \\ \text{d'exécution} \end{array} \leq \begin{array}{c} (a+t_{2A}'')\times n \\ +(b) \end{array}$$

Caractérisation du temps d'exécution de l'algo 1 : `recherche_lineaire(A,n,x)`

$$\begin{array}{c} (a)\times n \\ +(b) \end{array} \leq \begin{array}{c} \text{temps} \\ \text{d'exécution} \end{array} \leq \begin{array}{c} (a+t_{2A}'')\times n \\ +(b) \end{array}$$

fonction affine $an+b$ $\rightarrow$ taux de croissance en

Caractérisation du temps d'exécution de l'algo 1 : `recherche_lineaire(A,n,x)`

$$\begin{array}{c} (a)\times n \\ +(b) \end{array} \leq \begin{array}{c} \text{temps} \\ \text{d'exécution} \end{array} \leq \begin{array}{c} (a+t_{2A}'')\times n \\ +(b) \end{array}$$

fonction affine $an+b$ $\rightarrow$ taux de croissance en n
fonction an^2+bn+c $\rightarrow$ taux de croissance en

Caractérisation du temps d'exécution de l'algo 1 : `recherche_lineaire(A,n,x)`

$$\begin{array}{c} (a)\times n \\ +(b) \end{array} \leq \begin{array}{c} \text{temps} \\ \text{d'exécution} \end{array} \leq \begin{array}{c} (a+t_{2A}'')\times n \\ +(b) \end{array}$$

fonction affine $an+b$ $\rightarrow$ taux de croissance en n
fonction an^2+bn+c $\rightarrow$ taux de croissance en n^2
fonction $an\lg n+b$ $\rightarrow$ taux de croissance en

Caractérisation du temps d'exécution de l'algo 1 : `recherche_lineaire(A,n,x)`

$$\begin{matrix}(a)\times n\\ +(b)\end{matrix} \leq \begin{matrix}\text{temps}\\ \text{d'exécution}\end{matrix} \leq \begin{matrix}(a+t''_{2A})\times n\\ +(b)\end{matrix}$$

fonction affine $an+b$ $\rightarrow$ taux de croissance en n
fonction an^2+bn+c $\rightarrow$ taux de croissance en n^2
fonction $an\lg n+b$ $\rightarrow$ taux de croissance en $n\lg n$

Voyons l'algo 2 ...

Caractérisation du temps d'exécution de l'algo 2 : `meilleure_recherche_lineaire(A,n,x)`

Refaire ce qu'on vient de faire sur l'algo 2.

Caractérisation du temps d'exécution de l'algo 2 : `meilleure_recherche_lineaire(A,n,x)`

étape 2 : 1 fois t_2
étape 1 : comparaison de i avec n : q fois avec $1 \leq q \leq n+1$: t'_1
incrémentation de i : q fois avec $1 \leq q \leq n+1$: t''_1
étape 1A : test `A[i]==x` : q fois t'_{1A}
retourne la sortie : 1 fois : t''_{1A}

Le temps d'exécution est de l'ordre de
$$(t'_1+t''_1+t'_{1A})\times q+(t''_{1A}+t_2)$$
avec $1 \leq q \leq n+1$.

Caractérisation du temps d'exécution de l'algo 2 : `meilleure_recherche_lineaire(A,n,x)`

Le temps d'exécution est de l'ordre de
$$(t'_1+t''_1+t'_{1A})\times q+(t''_{1A}+t_2)$$
avec $1 \leq q \leq n+1$.

Au pire : $x \notin A$, temps d'exécution

Caractérisation du temps d'exécution de l'algo 2 : `meilleure_recherche_lineaire(A,n,x)`

Le temps d'exécution est de l'ordre de
$$(t'_1 + t''_1 + t'_{1A}) \times q + (t''_{1A} + t_2)$$
avec $1 \leq q \leq n+1$.

Au pire : $x \notin A$, temps d'exécution
$(t'_1 + t''_1 + t'_{1A}) \times n + (t''_{1A} + t_2 + t'_1 + t''_1 + t'_{1A})$
taux de croissance en n

Au mieux : `A[1]=x`, temps d'exécution

Caractérisation du temps d'exécution de l'algo 2 : `meilleure_recherche_lineaire(A,n,x)`

Le temps d'exécution est de l'ordre de
$$(t'_1 + t''_1 + t'_{1A}) \times q + (t''_{1A} + t_2)$$
avec $1 \leq q \leq n+1$.

Au pire : $x \notin A$, temps d'exécution
$(t'_1 + t''_1 + t'_{1A}) \times n + (t''_{1A} + t_2 + t'_1 + t''_1 + t'_{1A})$
taux de croissance en n

Au mieux : `A[1]=x`, temps d'exécution
$t'_1 + t''_1 + t'_{1A} + t''_{1A} + t_2$ on dit qu'on est en temps constant

Caractérisation du temps d'exécution de l'algo 2 : `meilleure_recherche_lineaire(A,n,x)`

Dans l'algo 2, on ne sait pas à l'avance combien de fois sera exécutée la boucle for :

entre 1 fois (`A[1]=x`)

et n fois au max ($x \notin A$)

Dans un cas comme celui-ci, bien que le temps d'exécution puisse être meilleur qu'une fonction affine, il ne sera jamais pire qu'une fonction affine. On dit alors que l'algo 2 est en $\mathcal{O}(n)$.

Définition

La notation $\mathcal{O}()$ est utilisée pour indiquer qu'un temps d'exécution n'est jamais pire qu'une constante multipliée par une fonction de n. Comportement asymptotique.

A retenir

fonction constante a	$\rightarrow$	$\mathcal{O}(1)$
fonction $a \lg n + b$	$\rightarrow$	$\mathcal{O}(\lg n)$
fonction affine $an + b$	$\rightarrow$	$\mathcal{O}(n)$
fonction $an^2 + bn + c$	$\rightarrow$	$\mathcal{O}(n^2)$
fonction $an \lg n + b$	$\rightarrow$	$\mathcal{O}(n \lg n)$
fonction exponentielle $a \times 2^n$	$\rightarrow$	$\mathcal{O}(2^n)$

exercice 3 à faire sur ordinateur
exercice 4 : énoncé sur `nsivaugelas.free.fr`
DM infogl algorithmique