Algorithmes et complexité

Étienne Lozes

25 août 2016
Cachan

Sommaire

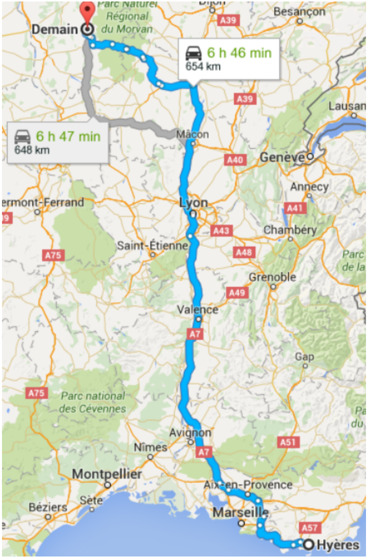
Sommaire

Définition

Un algorithme est une description
non ambiguë d'une méthode effective
permettant d'obtenir un résultat en réalisant des
opérations élémentaires.

Exemple 1

1. Prendre la direction ouest sur Rue des Porches vers Rue Portalet
2. Prendre à droite sur Rue Portalet
3. Prendre à gauche sur Place Massillon
4. Prendre légèrement à droite sur Rue du Vieux Cimetière
5. Prendre à droite sur Rue Sainte-Catherine
6. Prendre à gauche sur Place Saint-Paul
7. Place Saint-Paul tourne à droite et devient Rue de la Croix
8. Prendre à droite sur Montée Sainte-Croix
9. Prendre à droite sur Avenue Edith Wharton
10. Au rond-point, prendre la 2e sortie sur Rue Seré de Rivières
11. Prendre à gauche sur Avenue Alexis Godillot/D554
    Traverser le rond-point



Exemple 2

1. Une clé de détermination simplifiée des grands groupes d'animaux

- Pas de squelette interne osseux ou cartilagineux
  - Pas de coquille ou de squelette externe
    - Corps plutôt globuleux, aquatique
      - Vit fixé, forme variable, nombreux orifices …… **Spongiaires** (*éponge*)
      - Corps translucide avec des tentacules urticants, forme fixée pouvant former des constructions calcaires, ou libre …… **Cnidaires** (*anémone, méduse*)
      - Présence de 8 tentacules munies de ventouses autour de la tête …… se reporter à *
    - Corps allongé, pas de pattes ni de tentacules
      - Corps plat, avec ou sans anneaux, souvent parasite …… **Plathelminthes** (*ténia, douve*)
      - Corps rond, sans anneaux, souvent parasite …… **Némathelminthes** (*ascaris, nielle*)
      - Cours rond et annelé …… **Annélides** (*lombric, sangsue*)
  - Présence d'une coquille ou d'un squelette externe
    - Corps à symétrie radiaire d'ordre 5, en forme d'étoile, ou globuleux avec des piquants …… **Échinodermes** (*oursin, étoile de mer*)
    - Corps mou généralement entouré d'une coquille …… **Mollusques**
      - Une coquille enroulée en hélice …… **Gastéropodes** (*escargot, patelle, limace*)
      - Deux coquilles reliées par une charnière …… **Lamellibranches** ou **Bivalves** (*huître*)
      - * Coquille enroulée ou restreinte à un "os" ou absente, présence de 8 ou 10 tentacules autour de la tête …… **Céphalopodes** (*nautile, seiche, pieuvre*)
    - Squelette externe qui recouvre entièrement le corps, corps segmenté, appendices articulés …… **Arthropodes**
      - Corps en deux parties, pas d'antennes, 4 paires de pattes …… **Arachnides** (*araignée, acarien*)
      - Très nombreux segments portant chacun 1 ou 2 paires de pattes …… **Myriapodes** (*iule, gloméris*)
      - Deux paires d'antennes, tête et thorax soudés en un céphalothorax …… **Crustacés** (*crevette, daphnie, cloporte*)
      - 1 paire d'antennes, 3 paires de pattes, généralement présence d'ailes …… **Insectes** (*criquet, abeille*) voir p. 13
- Présence d'un squelette interne osseux ou cartilagineux …… **Vertébrés**
  - Peau recouverte d'écailles libres (non soudées) …… **Poissons** (*requin, truite*)
  - Peau nue contenant de nombreuses glandes …… **Amphibiens** (*grenouille, triton*)
  - Peau recouverte d'écailles soudées, jamais de plumes …… **Reptiles** (*serpent, tortue*)
  - Peau recouverte de plumes et d'écailles soudées sur les membres postérieurs …… **Oiseaux** (*pinson, manchot, autruche*)
  - Peau recouverte de poils, présence de mamelles …… **Mammifères** (*souris, éléphant, dauphin*)

Exemple 3


2
L'AMOUR EST UN OISEAU REBELLE
Habanera
extrait de l'opéra « Carmen »
Georges BIZET
(1838-1875)
Arrangt : Hans Günter HEUMANN
Allegretto quasi andantino ♩ = 54
pp
1.
2.

Limites de la définition

Un algorithme est une description *non ambiguë* d'une *méthode effective* permettant d'obtenir un *résultat* en réalisant des *opérations élémentaires*.

- **non ambiguïté** un algorithme est destiné à un humain
- **opérations élémentaires** elles-mêmes susceptibles de dépendre d'algorithmes à un autre niveau de granularité
- **effectivité** notion subjective, dépend des ressources disponibles
- **résultat** il s'agit de répondre correctement à un problème

Problèmes algorithmiques

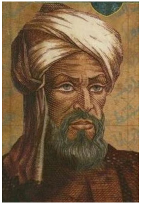
- **Donnée** un entier a
  Résultat la somme $1 + 2 + \cdots + a$
- **Donnée** une liste de mots
  Résultat la liste triée par ordre alphabetique
- **Donnée** une carte, deux villes A et B
  Résultat un plus court chemin de A à B
- **Donnée** un programme P
  Résultat est-il vrai que le programme P ne termine pas ?

Un algorithme doit résoudre toutes les instances d'un problème.

Etymologie : les algorithmes et l'algèbre

Le mot algorithme dérive du nom du mathématicien perse Al Khawarizmi.

Son *kitab al-mukhtasar fi hisab al-jabr wa-l-muqabala* a donné le mot *algèbre*.



Algèbre signifie réduction, une notion fondamentale en algorithmique et en complexité.

Un algorithme peut simplifier ou transformer un problème de sorte que l'on se ramène à un problème trivial.

Sommaire

Notre premier problème

Donnée : une liste d'entiers
Résultat le plus grand entier présent dans cette liste

Des instances : [2] , [1, 2, -3, 0, 2, 2]

Comment trouver le plus grand *en général* ?

Notre premier problème

Donnée : une liste d'entiers
Résultat le plus grand entier présent dans cette liste

Des instances : [2] , [1, 2, -3, 0, 2, 2]

Comment trouver le plus grand *en général* ?

- on parcours la liste
- on note sur une ardoise veleda le plus grand qu'on a vu
- si en cours de parcours passe un plus grand, on efface l'ardoise et on inscrit le nouveau plus grand
- quand on est au bout, on annonce ce qui est écrit sur l'ardoise

Notre premier algorithme

Formalisons...

Recherche du maximum

Données : liste non vide d'entiers L

Résultat : entier m le plus grand de L

début

   $m \leftarrow L[1]$;

   pour chaque i *allant de* 2 *à* $|L|$ **faire**

      si $L[i] > m$ **alors** $m \leftarrow L[i]$;

   retourner m

Un problème... des algorithmes ?

Pour un problème donné, il peut y avoir plusieurs algorithmes différents... ou aucun.

Lequel choisir ?

- le plus efficace
- le plus facile à programmer
- le plus facile à exécuter
- le plus résistant (aux bruits, aux pannes, etc)
- ...

Comment mesurer et comparer l'efficacité des algorithmes ?

Un problème... des algorithmes ?

Pour un problème donné, il peut y avoir plusieurs algorithmes différents... ou aucun.

Lequel choisir ?

- le plus efficace
- le plus facile à programmer
- le plus facile à exécuter
- le plus résistant (aux bruits, aux pannes, etc)
- ...

Comment mesurer et comparer l'efficacité des algorithmes ?

La complexité en temps

On compte le nombre d'opérations élémentaires effectuées. Par exemple, pour l'algorithme précédent, sur une liste de n éléments

- on lit et écrit n fois la variable i
- on lit n fois une case de la liste L
- on fait $n-1$ comparaisons
- on écrit *au moins* une fois et *au plus* n fois la variable m

On a donc effectué entre $4n$ et $5n-1$ opérations.

Les opérations élémentaires ne prennent pas toutes le même temps. Si l'on voulait calculer le temps d'exécution (ce que fait un ingénieur qui conçoit un système informatique), il faudrait pondérer la somme.

Complexité asymptotique dans le pire cas

Pour savoir si un algorithme peut passer à l'échelle on s'intéresse à sa complexité asymptotique.

- on ignore les différences de temps entre opérations élémentaires
- on ignore les parties de l'algorithme qui prennent un temps constant
- on se focalise sur ce qui prend du temps pour de très grandes instances

- dans cet exposé, on regarde la complexité dans le pire cas (plus facile à évaluer).

Notation $\mathcal{O}(\dots)$

En math, une fonction $g(n)$ à valeurs entières est en $\mathcal{O}(f(n))$ si f majore g vers l'infini à une constante multiplicative près.

Exemples :

- $12n$ est en $\mathcal{O}(n)$
- $72872n + 827480103239$ est en $\mathcal{O}(n)$
- la fonction constante à $10000^{10000000000}$ est en $\mathcal{O}(1)$
- la fonction $n^2 + n + \sqrt{n}$ est en $\mathcal{O}(n^2)$
- la fonction $\log_2(7n^{15})$ est en $\mathcal{O}(\log(n))$
- ...

Algorithmes efficaces... ou pas

- les algorithmes sous-linéaires, par exemple en $\mathcal{O}(\log(n))$
- les algorithmes linéaires en $\mathcal{O}(n)$,comme celui de la recherche du maximum
- les algorithmes en $\mathcal{O}(n \log n)$, par exemple de type "diviser pour régner".

- les algorithmes en temps polynomial $\mathcal{O}(n^k)$, i.e. quadratique, cubique, etc

- les algorithmes en temps exponentiel $\mathcal{O}(2^n)$, souvent de type résolutions de contraintes
- les algorithmes en temps doublement exponentiel $\mathcal{O}(2^{2^n})$
- ...

Exemple en pratique

un algorithme traite en une heure un film de taille T ; si on accède à une machine 1000 fois plus rapide, on peut traiter en une heure un film de taille

- $1000 \cdot T$ si c'est un algorithme en temps linéaire
- $31,6 \cdot T$ si c'est un algorithme en temps quadratique
- $T + 9,97$ si c'est un algorithme en temps exponentiel

Retour sur la recherche du maximum

Un puzzle

- la population mondiale augmente exponentiellement
- le nombre d'équipes participant à la coupe du monde de foot aussi
- à quelle vitesse augmente la durée de la coupe du monde ?

Retour sur la recherche du maximum

Un puzzle

- la population mondiale augmente exponentiellement
- le nombre d'équipes participant à la coupe du monde de foot aussi
- à quelle vitesse augmente la durée de la coupe du monde ?

Il existe plusieurs notions de complexité :

- complexité en temps (taille de circuit)
- complexité en temps parallèle (profondeur de circuit)
- complexité en espace
- complexité en communication
- ...

Notion de liste triée

On considère une liste de données triées par rapport à une clé.

Exemples :

- un répertoire téléphonique (clé=nom)
- un agenda (clé=date)
- des copies d'élèves (clé=note)
- un dictionnaire (clé=mot)
- ...

La liste de données $[d_1, \ldots, d_n]$ est triée si clé$(d_1)<$clé$(d_2)< \cdots <$clé(d_n)

A noter :

- chaque donnée est identifiée par sa clé (liste sans répétition)
- on doit pouvoir comparer les clés (ordre total)

Recherche d'une donnée par rapport à sa clé

Problème : étant donnée une clé c et une liste triée L, trouver (lorsque c'est possible) une donnée d de L dont la clé vaut c.

Algorithme naif :

Recherche par balayage

Données : liste de données L de taille n, clé c
Résultat : $d \in L$ tel que clé(d)=c
début
  pour chaque *i allant de* 1 *à* n **faire**
    si *clé*($L[i]$)$= c$ **alors retourner** $L[i]$;
  retourner *Non trouvé*

Complexité : $\mathcal{O}(n)$ (dans le pire cas, on parcourt toute la liste).

Algorithme de recherche dichotomique

Principe :

- on cherche la clé c dans un intervalle
- on regarde la clé c' de la donnée au milieu de l'intervalle
- si $c = c'$, on s'arrête,
- sinon on continue avec le sous-intervalle gauche/droite selon que $c < c'$ ou $c > c'$

Exemple : On recherche la clé 63.

10	12	16	21	26	49	50	55	61	63	69	71	73	77	81

Algorithme de recherche dichotomique

Principe :

- on cherche la clé c dans un intervalle
- on regarde la clé c' de la donnée au milieu de l'intervalle
- si $c = c'$, on s'arrête,
- sinon on continue avec le sous-intervalle gauche/droite selon que $c < c'$ ou $c > c'$

Exemple : On recherche la clé 63.

10	12	16	21	26	49	50	55	61	63	69	71	73	77	81

Algorithme de recherche dichotomique

Principe :

- on cherche la clé c dans un intervalle
- on regarde la clé c' de la donnée au milieu de l'intervalle
- si $c = c'$, on s'arrête,
- sinon on continue avec le sous-intervalle gauche/droite selon que $c < c'$ ou $c > c'$

Exemple : On recherche la clé 63.

10	12	16	21	26	49	50	55	61	63	69	71	73	77	81

Algorithme de recherche dichotomique

Principe :

- on cherche la clé c dans un intervalle
- on regarde la clé c' de la donnée au milieu de l'intervalle
- si $c = c'$, on s'arrête,
- sinon on continue avec le sous-intervalle gauche/droite selon que $c < c'$ ou $c > c'$

Exemple : On recherche la clé 63.

10	12	16	21	26	49	50	55	61	63	69	71	73	77	81

Algorithme de recherche dichotomique

Principe :

- on cherche la clé c dans un intervalle
- on regarde la clé c' de la donnée au milieu de l'intervalle
- si $c = c'$, on s'arrête,
- sinon on continue avec le sous-intervalle gauche/droite selon que $c < c'$ ou $c > c'$

Exemple : On recherche la clé 63.

10	12	16	21	26	49	50	55	61	63	69	71	73	77	81

Algorithme de recherche dichotomique

Principe :

- on cherche la clé c dans un intervalle
- on regarde la clé c' de la donnée au milieu de l'intervalle
- si $c = c'$, on s'arrête,
- sinon on continue avec le sous-intervalle gauche/droite selon que $c < c'$ ou $c > c'$

Exemple : On recherche la clé 63.

10	12	16	21	26	49	50	55	61	63	69	71	73	77	81

Trouvé !

Algorithme de recherche dichotomique

Principe :

- on cherche la clé c dans un intervalle
- on regarde la clé c' de la donnée au milieu de l'intervalle
- si $c = c'$, on s'arrête,
- sinon on continue avec le sous-intervalle gauche/droite selon que $c < c'$ ou $c > c'$

Exemple : On recherche la clé 52.

10	12	16	21	26	49	50	55	61	63	69	71	73	77	81

Algorithme de recherche dichotomique

Principe :

- on cherche la clé c dans un intervalle
- on regarde la clé c' de la donnée au milieu de l'intervalle
- si $c = c'$, on s'arrête,
- sinon on continue avec le sous-intervalle gauche/droite selon que $c < c'$ ou $c > c'$

Exemple : On recherche la clé 52.

10	12	16	21	26	49	50	55	61	63	69	71	73	77	81

Algorithme de recherche dichotomique

Principe :

- on cherche la clé c dans un intervalle
- on regarde la clé c' de la donnée au milieu de l'intervalle
- si $c = c'$, on s'arrête,
- sinon on continue avec le sous-intervalle gauche/droite selon que $c < c'$ ou $c > c'$

Exemple : On recherche la clé 52.

10	12	16	21	26	49	50	55	61	63	69	71	73	77	81

Algorithme de recherche dichotomique

Principe :

- on cherche la clé c dans un intervalle
- on regarde la clé c' de la donnée au milieu de l'intervalle
- si $c = c'$, on s'arrête,
- sinon on continue avec le sous-intervalle gauche/droite selon que $c < c'$ ou $c > c'$

Exemple : On recherche la clé 52.

10	12	16	21	26	49	50	55	61	63	69	71	73	77	81

Algorithme de recherche dichotomique

Principe :

- on cherche la clé c dans un intervalle
- on regarde la clé c' de la donnée au milieu de l'intervalle
- si $c = c'$, on s'arrête,
- sinon on continue avec le sous-intervalle gauche/droite selon que $c < c'$ ou $c > c'$

Exemple : On recherche la clé 52.

10	12	16	21	26	49	50	55	61	63	69	71	73	77	81

Algorithme de recherche dichotomique

Principe :

- on cherche la clé c dans un intervalle
- on regarde la clé c' de la donnée au milieu de l'intervalle
- si $c = c'$, on s'arrête,
- sinon on continue avec le sous-intervalle gauche/droite selon que $c < c'$ ou $c > c'$

Exemple : On recherche la clé 52.

10	12	16	21	26	49	50	55	61	63	69	71	73	77	81

Algorithme de recherche dichotomique

Principe :

- on cherche la clé c dans un intervalle
- on regarde la clé c' de la donnée au milieu de l'intervalle
- si $c = c'$, on s'arrête,
- sinon on continue avec le sous-intervalle gauche/droite selon que $c < c'$ ou $c > c'$

Exemple : On recherche la clé 52.

10	12	16	21	26	49	50	55	61	63	69	71	73	77	81

Algorithme de recherche dichotomique

Principe :

- on cherche la clé c dans un intervalle
- on regarde la clé c' de la donnée au milieu de l'intervalle
- si $c = c'$, on s'arrête,
- sinon on continue avec le sous-intervalle gauche/droite selon que $c < c'$ ou $c > c'$

Exemple : On recherche la clé 52.

10	12	16	21	26	49	50	55	61	63	69	71	73	77	81

Non Trouvé !

Complexité de la recherche par dichotomie

- dans le pire cas, on continue toujours la recherche dans un sous-intervalle jusqu'à un intervalle de taille 1
- la taille de l'intervalle diminue de moitié à chaque itération après k itération, la taille de l'intervalle est $\frac{|L|}{2^k}$
- on a donc $k = \log_2 |L|$ dans le pire cas l'algorithme a donc une complexité en temps en $\mathcal{O}(\log n)$.

Recherche par dichotomie

Données : liste de données L non vide, triée, de taille n, clé c
Résultat : $d \in L$ tel que clé(d)$= c$, s'il existe
début
    $g \leftarrow 1$; $d \leftarrow n$;
    tant que $g \neq d$ **faire**
        $i \leftarrow \lfloor \frac{g+d}{2} \rfloor$;
        si *clé(L[i])*$= c$ **alors retourner** *L[i]*;
        si *clé(L[i])*$< c$ **alors**
            $g \leftarrow i + 1$
        si *clé(L[i])*$> c$ **alors**
            $d \leftarrow i - 1$
    si *clé(L[g])*$= c$ **alors retourner** *L[d]*;
    sinon retourner *Non Trouvé*;

Rechercher un mot dans un texte

Problème : déterminer si un mot apparaît dans un texte, et si oui, a quelle position.

Nombreuses applications et variantes :

- recherche/remplacement
- filtrage
- complétion automatique
- ...
- distance d'édition (travail collaboratif, génomique,...)

Un domaine à part entière : l'algorithmique du texte.

Quelques définitions

Un “mot” = une liste de caractères.

On note $M[i]$ le i-ème caractère du mot M.

Par exemple, si M est le mot `algo`, alors $M[3] =$g.

Le mot M (appelé motif) apparait dans un mot T (appellé texte) à la position p si

- $M[1] = T[p]$, et
- $M[2] = T[p+1]$, et
- ..., et
- $M[m] = T[p+m-1]$

où m est la longueur du mot M, et où $p+m-1$ est une position valide de T.

La recherche de sous-chaîne

Données : le motif M, le texte T.
Problème : trouver la première position p à laquelle M apparait dans T, si elle existe.

Le premier algorithme qu'on peut imaginer

- on essaie toutes les valeurs de p une à une
- pour une valeur de p fixée, on vérifie en balayant simultanément M et T à partir de la position p

Exemple :

b	a	o	b	a	b
b	a	b			

La recherche de sous-chaîne

Données : le motif M, le texte T.
Problème : trouver la première position p à laquelle M apparait dans T, si elle existe.

Le premier algorithme qu'on peut imaginer

- on essaie toutes les valeurs de p une à une
- pour une valeur de p fixée, on vérifie en balayant simultanément M et T à partir de la position p

Exemple :

b	a	o	b	a	b
b	a	b			

La recherche de sous-chaîne

Données : le motif M, le texte T.
Problème : trouver la première position p à laquelle M apparait dans T, si elle existe.

Le premier algorithme qu'on peut imaginer

- on essaie toutes les valeurs de p une à une
- pour une valeur de p fixée, on vérifie en balayant simultanément M et T à partir de la position p

Exemple :

b	a	o	b	a	b
b	a	b			

La recherche de sous-chaîne

Données : le motif M, le texte T.
Problème : trouver la première position p à laquelle M apparait dans T, si elle existe.

Le premier algorithme qu'on peut imaginer

- on essaie toutes les valeurs de p une à une
- pour une valeur de p fixée, on vérifie en balayant simultanément M et T à partir de la position p

Exemple :

b	a	o	b	a	b
	b	a	b		

La recherche de sous-chaîne

Données : le motif M, le texte T.
Problème : trouver la première position p à laquelle M apparait dans T, si elle existe.

Le premier algorithme qu'on peut imaginer

- on essaie toutes les valeurs de p une à une
- pour une valeur de p fixée, on vérifie en balayant simultanément M et T à partir de la position p

Exemple :

b	a	o	b	a	b
		b	a	b	

La recherche de sous-chaîne

Données : le motif M, le texte T.
Problème : trouver la première position p à laquelle M apparait dans T, si elle existe.

Le premier algorithme qu'on peut imaginer

- on essaie toutes les valeurs de p une à une
- pour une valeur de p fixée, on vérifie en balayant simultanément M et T à partir de la position p

Exemple :

b	a	o	b	a	b
			b	a	b

La recherche de sous-chaîne

Données : le motif M, le texte T.
Problème : trouver la première position p à laquelle M apparait dans T, si elle existe.

Le premier algorithme qu'on peut imaginer

- on essaie toutes les valeurs de p une à une
- pour une valeur de p fixée, on vérifie en balayant simultanément M et T à partir de la position p

Exemple :

b	a	o	b	a	b
			b	a	b

La recherche de sous-chaîne

Données : le motif M, le texte T.
Problème : trouver la première position p à laquelle M apparait dans T, si elle existe.

Le premier algorithme qu'on peut imaginer

- on essaie toutes les valeurs de p une à une
- pour une valeur de p fixée, on vérifie en balayant simultanément M et T à partir de la position p

Exemple :

b	a	o	b	a	b
			b	a	b

La recherche de sous-chaîne

Données : le motif M, le texte T.
Problème : trouver la première position p à laquelle M apparait dans T, si elle existe.

Le premier algorithme qu'on peut imaginer

- on essaie toutes les valeurs de p une à une
- pour une valeur de p fixée, on vérifie en balayant simultanément M et T à partir de la position p

Exemple : Trouvé !

b	a	o	b	a	b
			b	a	b

Recherche de sous-chaîne “naïve”

Données : motif M de longeur m, texte T

Résultat : position p à laquelle M apparait dans T

début

```
p ← 1;
tant que p + m − 1 < |T| faire
    /* on teste en partant de p                                  */
    i ← 1;
    tant que i ≤ m et M[i] = T[p + i − 1] faire
        i ← i + 1;
    si i = m + 1 alors retourner p;
    sinon p ← p + 1;/* on passe au p suivant                     */
retourner Non trouvé
```

Complexité : $\mathcal{O}(|M| \cdot |T|)$ (linéaire en $|T|$).

Notion de backtracking

Cet algorithme est un exemple d'algorithme à *reprise sur échec*.
En anglais (et en informatique), on dit plutôt backtracking.

- on cherche la solution en énumérant tous les candidats
- pour chacun, on teste s'il est solution
- si ce n'est pas le cas, on échoue et on reprend avec un autre candidat
- dans l'idéal pas tout à fait à zéro
- et en ayant éliminé au passage d'autres candidats

Autre exemple : résoudre une grille de sudoku.

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
b	a	r	b	a	r	a			

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
b	a	r	b	a	r	a			

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
b	a	r	b	a	r	a			

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
b	a	r	b	a	r	a			

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
b	a	r	b	a	r	a			

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
b	a	r	b	a	r	a			

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
b	a	r	b	a	r	a			

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
			b	a	r	b	a	r	a

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
			b	a	r	b	a	r	a

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
			b	a	r	b	a	r	a

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
			b	a	r	b	a	r	a

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
			b	a	r	b	a	r	a

L'algorithme de Boyer-Moore : sauter des positions sans les lire !

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e
e	x	a	m	p	l	e																	

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
			b	a	r	b	a	r	a

L'algorithme de Boyer-Moore : sauter des positions sans les lire !

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e
							e	x	a	m	p	l	e										

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b a r b a r b a r a
b a r b a r a

L'algorithme de Boyer-Moore : sauter des positions sans les lire !

h e r e i s a s i m p l e e x a m p l e
e x a m p l e

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
			b	a	r	b	a	r	a

L'algorithme de Boyer-Moore : sauter des positions sans les lire !

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e
									e	x	a	m	p	l	e							

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
			b	a	r	b	a	r	a

L'algorithme de Boyer-Moore : sauter des positions sans les lire !

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e
									e	x	a	m	p	l	e								

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
			b	a	r	b	a	r	a

L'algorithme de Boyer-Moore : sauter des positions sans les lire !

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e
									e	x	a	m	p	l	e							

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
			b	a	r	b	a	r	a

L'algorithme de Boyer-Moore : sauter des positions sans les lire !

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e
									e	x	a	m	p	l	e							

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b a r b a r b a r a
b a r b a r a

L'algorithme de Boyer-Moore : sauter des positions sans les lire !

h e r e i s a s i m p l e e x a m p l e
e x a m p l e

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
			b	a	r	b	a	r	a

L'algorithme de Boyer-Moore : sauter des positions sans les lire !

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e
																	e	x	a	m	p	l	e

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
			b	a	r	b	a	r	a

L'algorithme de Boyer-Moore : sauter des positions sans les lire !

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e
																	e	x	a	m	p	l	e

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
			b	a	r	b	a	r	a

L'algorithme de Boyer-Moore : sauter des positions sans les lire !

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e
																	e	x	a	m	p	l	e

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
			b	a	r	b	a	r	a

L'algorithme de Boyer-Moore : sauter des positions sans les lire !

h	e	r	e		i	s		a		s	i	m	p	l	e	e	x	a	m	p	l	e
																e	x	a	m	p	l	e

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
			b	a	r	b	a	r	a

L'algorithme de Boyer-Moore : sauter des positions sans les lire !

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e
																	e	x	a	m	p	l	e

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
			b	a	r	b	a	r	a

L'algorithme de Boyer-Moore : sauter des positions sans les lire !

h	e	r	e		i	s		a		s	i	m	p	l	e		e	x	a	m	p	l	e
																	e	x	a	m	p	l	e

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
			b	a	r	b	a	r	a

L'algorithme de Boyer-Moore : sauter des positions sans les lire !

h	e	r	e		i	s		a		s	i	m	p	l	e	e	x	a	m	p	l	e
																e	x	a	m	p	l	e

Backtracking efficace : exemples

De nombreux algorithmes de recherche de sous-chaîne existent.

Principe : améliorer le backtracking.

L'algorithme de Knuth-Morris-Pratt : sauter les positions déjà lues

b	a	r	b	a	r	b	a	r	a
			b	a	r	b	a	r	a

L'algorithme de Boyer-Moore : sauter des positions sans les lire !

h	e	r	e		i	s		a		s	i	m	p	l	e	e	x	a	m	p	l	e
																e	x	a	m	p	l	e

Sommaire

Trier une liste par rapport à une clé

Problème : étant donnée une liste de données L et une clé pour ces données, calculer la liste triée en ordre de clés croissantes.

Pour simplifier : donnée = entier = clé.

- problème connu
- très utile en pratique (tableur, carnet d'adresse, agenda,...)
- très utile pour d'autres algorithmes (dichotomie, géométrie, ...)
- grande richesse conceptuelle

Tri sélection : principe

- on prend le plus grand et on le met à la fin

Donnée	12	3	8	9	7	1
Résultat						

Tri sélection : principe

- on prend le plus grand et on le met à la fin

Donnée	12 3 8 9 7 1
Résultat	

Tri sélection : principe

- on prend le plus grand et on le met à la fin

Donnée	12	3	8	9	7	1
Résultat						12

Tri sélection : principe

- on prend le plus grand et on le met à la fin
- on prend le plus grand dans ce qui reste et on le met juste avant

Donnée	12	3	8	9	7	1
Résultat						12

Tri sélection : principe

- on prend le plus grand et on le met à la fin
- on prend le plus grand dans ce qui reste et on le met juste avant

Donnée	12	3	8	9	7	1
Résultat					9	12

Tri sélection : principe

- on prend le plus grand et on le met à la fin
- on prend le plus grand dans ce qui reste et on le met juste avant
- on continue...

Donnée	12	3	8	9	7	1
Résultat					9	12

Tri sélection : principe

- on prend le plus grand et on le met à la fin
- on prend le plus grand dans ce qui reste et on le met juste avant
- on continue...

Donnée	12	3	8	9	7	1
Résultat				8	9	12

Tri sélection : principe

- on prend le plus grand et on le met à la fin
- on prend le plus grand dans ce qui reste et on le met juste avant
- on continue...

Donnée	12	3	8	9	7	1
Résultat				8	9	12

Tri sélection : principe

- on prend le plus grand et on le met à la fin
- on prend le plus grand dans ce qui reste et on le met juste avant
- on continue...

Donnée	12	3	8	9	7	1
Résultat			7	8	9	12

Tri sélection : principe

- on prend le plus grand et on le met à la fin
- on prend le plus grand dans ce qui reste et on le met juste avant
- on continue...

Donnée	12	3	8	9	7	1
Résultat			7	8	9	12

Tri sélection : principe

- on prend le plus grand et on le met à la fin
- on prend le plus grand dans ce qui reste et on le met juste avant
- on continue...

Donnée	12	3	8	9	7	1
Résultat		3	7	8	9	12

Tri sélection : principe

- on prend le plus grand et on le met à la fin
- on prend le plus grand dans ce qui reste et on le met juste avant
- on continue...

Donnée	12	3	8	9	7	1
Résultat		3	7	8	9	12

Tri sélection : principe

- on prend le plus grand et on le met à la fin
- on prend le plus grand dans ce qui reste et on le met juste avant
- on continue...

Donnée	12	3	8	9	7	1
Résultat	1	3	7	8	9	12

Tri fusion : principe

- on sépare la liste à trier en deux listes deux fois plus petites
- on trie chacune de façon récursive
- on fusionne les deux listes pour obtenir une liste triée

Exemple :

8	3	6	1	2	5	4	7

Tri fusion : principe

- on sépare la liste à trier en deux listes deux fois plus petites
- on trie chacune de façon récursive
- on fusionne les deux listes pour obtenir une liste triée

Exemple :

8	3	6	1

2	5	4	7

Tri fusion : principe

- on sépare la liste à trier en deux listes deux fois plus petites
- on trie chacune de façon récursive
- on fusionne les deux listes pour obtenir une liste triée

Exemple : $\boxed{8 \quad 3}$ $\boxed{6 \quad 1}$ $\boxed{2 \quad 5 \quad 4 \quad 7}$

Tri fusion : principe

- on sépare la liste à trier en deux listes deux fois plus petites
- on trie chacune de façon récursive
- on fusionne les deux listes pour obtenir une liste triée

Exemple : $\boxed{3 \quad 8}$ $\boxed{6 \quad 1}$ $\boxed{2 \quad 5 \quad 4 \quad 7}$

Tri fusion : principe

- on sépare la liste à trier en deux listes deux fois plus petites
- on trie chacune de façon récursive
- on fusionne les deux listes pour obtenir une liste triée

Exemple : $\boxed{3 \quad 8}$ $\boxed{1 \quad 6}$ $\boxed{2 \quad 5 \quad 4 \quad 7}$

Tri fusion : principe

- on sépare la liste à trier en deux listes deux fois plus petites
- on trie chacune de façon récursive
- on fusionne les deux listes pour obtenir une liste triée

Exemple :

1	3	6	8

2	5	4	7

Tri fusion : principe

- on sépare la liste à trier en deux listes deux fois plus petites
- on trie chacune de façon récursive
- on fusionne les deux listes pour obtenir une liste triée

Exemple : $\boxed{1 \quad 3 \quad 6 \quad 8}$ $\boxed{2 \quad 5}$ $\boxed{4 \quad 7}$

Tri fusion : principe

- on sépare la liste à trier en deux listes deux fois plus petites
- on trie chacune de façon récursive
- on fusionne les deux listes pour obtenir une liste triée

Exemple :

1	3	6	8

2	5

4	7

Tri fusion : principe

- on sépare la liste à trier en deux listes deux fois plus petites
- on trie chacune de façon récursive
- on fusionne les deux listes pour obtenir une liste triée

Exemple : $\boxed{1 \quad 3 \quad 6 \quad 8}$ $\boxed{2 \quad 5}$ $\boxed{4 \quad 7}$

Tri fusion : principe

- on sépare la liste à trier en deux listes deux fois plus petites
- on trie chacune de façon récursive
- on fusionne les deux listes pour obtenir une liste triée

Exemple :

1	3	6	8

2	4	5	7

Tri fusion : principe

- on sépare la liste à trier en deux listes deux fois plus petites
- on trie chacune de façon récursive
- on fusionne les deux listes pour obtenir une liste triée

Exemple :

1	2	3	4	5	6	7	8

Fusionner deux listes triées

Principe :

- on a un curseur dans chaque liste
- on compare les valeurs aux deux curseurs
- on recopie la plus petite et on avance le curseur correspondant
- on répète

Exemple :

Données : 2 5 8 9 1 3 4

Résultat

Fusionner deux listes triées

Principe :

- on a un curseur dans chaque liste
- on compare les valeurs aux deux curseurs
- on recopie la plus petite et on avance le curseur correspondant
- on répète

Exemple :

Données : 2 5 8 9 1 3 4

Résultat

Fusionner deux listes triées

Principe :

- on a un curseur dans chaque liste
- on compare les valeurs aux deux curseurs
- on recopie la plus petite et on avance le curseur correspondant
- on répète

Exemple :

Données : 2 5 8 9 1 3 4

Résultat 1

Fusionner deux listes triées

Principe :

- on a un curseur dans chaque liste
- on compare les valeurs aux deux curseurs
- on recopie la plus petite et on avance le curseur correspondant
- on répète

Exemple :

Données : 2 5 8 9 1 3 4

Résultat 1

Fusionner deux listes triées

Principe :

- on a un curseur dans chaque liste
- on compare les valeurs aux deux curseurs
- on recopie la plus petite et on avance le curseur correspondant
- on répète

Exemple :

Données : 2 5 8 9 1 3 4

Résultat 1 2

Fusionner deux listes triées

Principe :

- on a un curseur dans chaque liste
- on compare les valeurs aux deux curseurs
- on recopie la plus petite et on avance le curseur correspondant
- on répète

Exemple :

Données : 2 5 8 9 1 3 4

Résultat 1 2

Fusionner deux listes triées

Principe :

- on a un curseur dans chaque liste
- on compare les valeurs aux deux curseurs
- on recopie la plus petite et on avance le curseur correspondant
- on répète

Exemple :

Données : 2 5 8 9 1 3 4

Résultat 1 2 3

Fusionner deux listes triées

Principe :

- on a un curseur dans chaque liste
- on compare les valeurs aux deux curseurs
- on recopie la plus petite et on avance le curseur correspondant
- on répète

Exemple :

Données : 2 5 8 9 1 3 4

Résultat 1 2 3

Fusionner deux listes triées

Principe :

- on a un curseur dans chaque liste
- on compare les valeurs aux deux curseurs
- on recopie la plus petite et on avance le curseur correspondant
- on répète

Exemple :

Données : 2 5 8 9 1 3 4

Résultat 1 2 3 4

Fusionner deux listes triées

Principe :

- on a un curseur dans chaque liste
- on compare les valeurs aux deux curseurs
- on recopie la plus petite et on avance le curseur correspondant
- on répète

Exemple :

Données : 2 5 8 9 1 3 4

Résultat 1 2 3 4

Fusionner deux listes triées

Principe :

- on a un curseur dans chaque liste
- on compare les valeurs aux deux curseurs
- on recopie la plus petite et on avance le curseur correspondant
- on répète

Exemple :

Données : 2 5 8 9 1 3 4

Résultat 1 2 3 4 5

Fusionner deux listes triées

Principe :

- on a un curseur dans chaque liste
- on compare les valeurs aux deux curseurs
- on recopie la plus petite et on avance le curseur correspondant
- on répète

Exemple :

Données : 2 5 8 9 1 3 4

Résultat 1 2 3 4 5

Fusionner deux listes triées

Principe :

- on a un curseur dans chaque liste
- on compare les valeurs aux deux curseurs
- on recopie la plus petite et on avance le curseur correspondant
- on répète

Exemple :

Données : 2 5 8 9 1 3 4

Résultat 1 2 3 4 5 8

Fusionner deux listes triées

Principe :

- on a un curseur dans chaque liste
- on compare les valeurs aux deux curseurs
- on recopie la plus petite et on avance le curseur correspondant
- on répète

Exemple :

Données : 2 5 8 9 1 3 4

Résultat 1 2 3 4 5 8

Fusionner deux listes triées

Principe :

- on a un curseur dans chaque liste
- on compare les valeurs aux deux curseurs
- on recopie la plus petite et on avance le curseur correspondant
- on répète

Exemple :

Données : 2 5 8 9 1 3 4

Résultat 1 2 3 4 5 8 9

Fusion de deux listes triées

Données : liste d'entiers L_1 et L_2 triées de longueurs n_1 et n_2
Résultat : liste triée R fusion de L_1 et L_2

début

$L_1[n_1+1] \leftarrow +\infty$; `/* ``béquilles'' (pour simplifier) */`
$L_2[n_2+1] \leftarrow +\infty$;
$i_1 \leftarrow 1$;
$i_2 \leftarrow 1$;
pour chaque j *allant de* 1 *à* n_1+n_2 **faire**
  si $L_1[i_1] < L_2[i_2]$ **alors**
    $R[j] \leftarrow L_1[i_1]$;
    $i_1 \leftarrow i_1+1$;
  sinon
    $R[j] \leftarrow L_2[i_2]$;
    $i_2 \leftarrow i_2+1$;
retourner R

Complexité : $\mathcal{O}(n_1+n_2)$

Tri fusion

Données : liste d'entiers L de longueur n

Résultat : liste triée R

début

- **si** $n > 1$ **alors**
  - $L_1 \leftarrow L[1 \cdots \lfloor \frac{n}{2} \rfloor]$;
  - $L_2 \leftarrow L[\lfloor \frac{n}{2} \rfloor + 1 \cdots n]$;
  - $R_1 \leftarrow \mathit{TriFusion}(L_1)$;
  - $R_2 \leftarrow \mathit{TriFusion}(L_2)$;
  - $R \leftarrow \mathit{Fusion}(R_1, R_2)$;
- **sinon** $R \leftarrow L$;
- **retourner** R

Complexité ?

Tri fusion

Données : liste d'entiers L de longueur n
Résultat : liste triée R
début
  si $n > 1$ **alors**
    $L_1 \leftarrow L[1 \cdots \lfloor \frac{n}{2} \rfloor]$;
    $L_2 \leftarrow L[\lfloor \frac{n}{2} \rfloor + 1 \cdots n]$;
    $R_1 \leftarrow TriFusion(L_1)$;
    $R_2 \leftarrow TriFusion(L_2)$;
    $R \leftarrow Fusion(R_1, R_2)$;
  sinon $R \leftarrow L$;
  retourner R

Complexité ? Soit t_n le temps dans le pire cas pour une liste de longueur n. On a $t_{2n} = 2t_n + 2n$, i.e. $t_n = n \log_2 n$ pour n une puissance de 2.

Complexité quasi-linéaire en $\mathcal{O}(n \log n)$.

Comparaison

- le tri sélection est en $\mathcal{O}(n^2)$
- le tri fusion est en $\mathcal{O}(n \log n)$
- le tri fusion est plus efficace
- on peut même montrer que tout algo de tri est au mieux en $\mathcal{O}(n \log n)$
- le tri fusion, la panacée ?

Comparaison

- le tri sélection est en $\mathcal{O}(n^2)$
- le tri fusion est en $\mathcal{O}(n \log n)$
- le tri fusion est plus efficace
- on peut même montrer que tout algo de tri est au mieux en $\mathcal{O}(n \log n)$
- le tri fusion, la panacée ?

- en fait, non, à cause des recopies
- on préfère parfois un tri "en place", qui échange les éléments à l'intérieur de la liste

Sommaire

Cinq messages à retenir

1. un problème, des algorithmes
2. plusieurs notions de complexité
3. de grands principes (diviser pour régner, backtracking,. . .)
4. faire une recherche dans des données, les trier, c'est facile
5. le sudoku, c'est difficile ($P \stackrel{?}{=} NP$)

Pour aller plus loin

- Vidéos sur Canal Inria
  - conférence de François Laroussinie sur l'algorithmique
  - conférence de Paul Gastin sur la calculabilité et la complexité
- Sites
  - Pixees et Class'Code
  - Computer Science Unplugged
  - Kahn Academy, cours sur les algorithmes
- Livres
  - Thomas H. Cormen. Algorithmes. Notions de base.
  - Donald Knuth. Éléments pour une histoire de l'informatique.