L'informatique dans les programmes de mathématiques

Enjeux didactiques, besoins de formation

Antoine Meyer[1] Simon Modeste[2]

[1]LIGM, Université Paris Est – Marne-la-Vallée

[2]IMAG, Université de Montpellier

Colloque CORFEM, 11 juin 2018

Contexte institutionnel

Introduction de notions informatiques au secondaire

- Algorithmique et programmation au Cycle 4
- Algorithmique et programmation au lycée
- Informatique dans le projet de nouveau lycée

Ouverture du CAPES de mathématiques à l'informatique

option informatique apparue au concours 2017

Contexte institutionnel qui soulève des questions didactiques

Contexte de réflexion

Projet ANR DEMaIn (lancement : janvier 2017)

Didactique et Épistémologie des interactions entre Math. et Info.

- Deux axes de travail :
  1. Fondations scientifiques : logique, algorithmique, langage, preuve
  2. Objets et concepts : math discrètes et de l'informatique, représentation des objets

- Deux volets :
  1. Épistémologique (à but didactique) : champs, notions, conceptions, modes de pensée communs ou spécifiques aux deux disciplines et leurs effets l'une sur l'autre
  2. Didactique : ingénierie didactique, potentiel d'enseignement riches à l'interface math-info, au secondaire et au supérieur, ressources pour la formation

Contexte de réflexion

Projet de Groupe de Recherche CNRS : DEMIPS

Didactique et Épistémologie des Mathématiques, de l'Informatique et de la Physique dans le Supérieur

- Thème 3 : Enseignement de l'arithmétique, des mathématiques discrètes et de l'Algorithmique, relations avec l'Informatique
- Thème 4 : Logique, langage, raisonnement, preuves, et apprentissages mathématiques et informatiques

Intérêt : objets potentiels pour le lycée, transition secondaire-supérieur, formation initiale disciplinaire des enseignants

Contexte de réflexion

IREM

- Groupe algorithmique de l'IREM de Paris
- Groupe algorithmique de l'IREM de Montpellier (en création)
- Commission Inter-IREM Informatique (C3i – octobre 2017)

Portée de l'exposé

Interactions math-info en cours de mathématiques

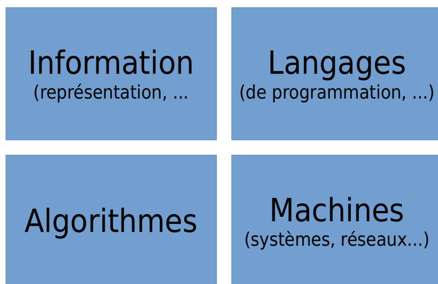
- Enjeux didactiques
- Prise en compte dans les ressources d'enseignement et de formation
- Pistes pour la formation

Trois exemples d'enjeux :

- Autour de l'idée de généralisation
- L'objet algorithme et l'objet programme
- Validation et preuve

Les mathématiques et l'informatique

Cartographie simplifiée de l'informatique scolaire :



Accent sur information (un peu), algorithmique et programmation

- Thème machines à prendre en compte, mais exploré plutôt dans d'autres disciplines
- Cf. journée SIF du 13 juin sur l'enseignement de l'informatique

Les mathématiques et l'informatique

Liens mathématiques-informatique :

- Fondements (logique, preuve...)
- Modes de pensée
- Algorithmique
- Modélisation et simulation
- Place du langage, rôle de la formalisation

Hypothèses

Étudier et identifier les liens mathématiques-informatique permet :

- de mettre en lumière certaines connaissances parfois implicites ou absentes habituellement...
- de multiplier les points de vue sur des objets, des problèmes ou des techniques...
- de favoriser certains apprentissages...

... en mathématiques et en informatique.

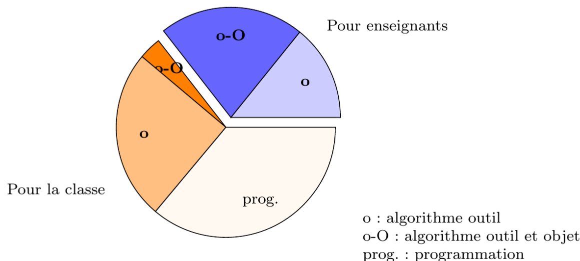
Il est possible et important de faire vivre les champs à l'interface (fidélité de la transposition didactique).

Panorama de quelques ressources existantes

Ressources

Un point sur les ressources IREM produites entre 2009 et 2012

- Objectif : pour la classe, l'initiation des enseignants, la formation
- Langages : naturel, algorithmique/pseudo-code, Scratch, Python, Basic, Xcas, CaML
- Algorithme comme outil ou comme objet (preuve, complexité)



Quelques ressources des IREM

- Méthode pour élaborer des algorithmes itératif (Aix-Marseille)
- Algorithmes gloutons (Lyon)
- Diviser pour régner (Lyon)
- Terminaison (Lyon)
- Ouvrage de formation à l'algorithmique et la logique (Aix-Marseille)
- Document d'accompagnement des stages de formation à l'algorithmique (Grenoble)

Quelques ressources des IREM

- La complexité c'est simple comme la dichotomie. Guillaume Connan, IREM de Nantes (Repères IREM 102, 2016)
- Algorithmique et programmation au cycle 4. Commentaires et recommandations du groupe Info. de la CII Lycée. 2017.
- L'évaluation de l'algorithmique dans l'enseignement des mathématiques au lycée. P. Lac et M. More. IREM de Clermont-Ferrand et CII Lycée (Repères IREM 106, 2017)

Prise en compte croissante des aspects « objet » et de la discipline informatique

Ressources à paraître

- Repères IREM, numéro spécial 116 (juillet 2019). *Quelles interactions entre l'informatique et les mathématiques ?*
- Analyse des exercices d'algorithmique du Diplôme National du Brevet 2017. Commission Inter-IREM Informatique (C3i). *En préparation.*
- Pourquoi est-il difficile d'enseigner la variable informatique ? C3i. *En préparation, titre provisoire.*

Ressources pour le formateur

Ressources issues de la recherche :

- Des thèses récentes autour de l'algorithmique : Modeste, Briant, Laval
- Recherches de l'époque de l'informatique des lycées (Samurçay, Rogalski, Arsac (J),...) et des premières introductions d'informatique en mathématiques, Papert...
- Aujourd'hui, encore des recherches dans cette veine (psychologie de la programmation, ...)
- Pensée computationnelle, algorithmique, informatique...
- Colloquium ARDM-CFEM sur l'enseignement de l'informatique, ...

Exemples d'enjeux pour l'enseignement et la formation

Exemples d'enjeux pour l'enseignement et la formation

- Généralisation : une dimension centrale de l'algorithmique, proche d'enjeux mathématiques, qui met en avant la résolution de problème – rôle particulier de la notion de fonction
- Algorithme / programme : entre modèle théorique et mise en oeuvre pratique, exemple de concepts à savoir différencier
- Preuve et validation : enjeux partagés avec les mathématiques, questionne les spécificités épistémologiques de l'informatique

Extrait du document-ressource lycée

Il s'agit de consolider les acquis du cycle 4 autour de deux idées essentielles : la notion universelle de fonction d'une part et la programmation comme production d'un texte dans un langage informatique d'autre part.

- *Les notions mathématique et informatique de fonction relèvent du même concept universel. En informatique, une fonction prend un ou plusieurs arguments et renvoie une valeur issue d'un calcul.*
- *Le choix d'un langage textuel, comme Python, au lieu d'un langage par blocs, comme Scratch, permet aux élèves de se confronter à la précision et la rigidité d'une syntaxe proche de celle des expressions mathématiques, avec l'avantage de pouvoir bénéficier du contrôle apporté par l'analyseur syntaxique.*

Exemples d'enjeux pour l'enseignement et la formation

Enjeu 1 : la question de la généralité

La question de la généralité

Enjeux pour l'élève

- Identifier l'algorithme comme une « machine à fabriquer des solutions » qui « marche à tous les coups »
- Distinguer cas général et cas particuliers d'un même problème
- Reconnaître des solutions algorithmiques semblables et en tirer un schéma général

Enjeux pour l'enseignant

- Construire une conception claire de la notion de problème algorithmique, et de l'impératif de généralité d'une solution
- Construire progressivement des schémas de résolution élaborés
  parcourir une liste, la filtrer, sommer ses éléments, énumérer un ensemble... voire introduire des paradigmes algorithmiques

Définitions

Definition (Problème algorithmique)

Un problème est défini par un couple (I, Q) :
I : ensemble des instances du problème
Q : une question que l'on pose pour toute instance du problème

Definition (Solution algorithmique)

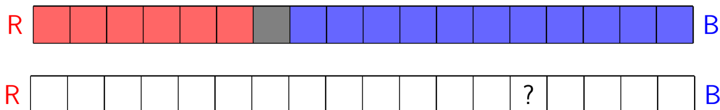
Résoudre algorithmiquement un problème, c'est proposer un algorithme qui peut prendre en entrée toute instance de I et renvoie en sortie une réponse à la question Q pour l'instance donnée

Motivations

Ce point de vue permet :

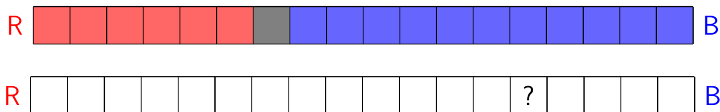
- de formaliser ce qu'est un algorithme
- d'étudier l'algorithme en tant qu'objet (preuve, complexité, ...)
- d'adopter un point de vue fonctionnel : algorithme comme une fonction
- de mettre en avant la dimension de généralité

Exemple : un jeu de frontières



- On peut interroger une case à la fois.
- Comment retrouver la frontière en interrogeant le moins de cases possible ?
- Algorithme de recherche par dichotomie :
  On teste la case qui sépare le territoire de la manière la plus équilibrée.
  On réitère sur le territoire restant jusqu'à trouver la frontière.

Exemple : un jeu de frontières



Variante :

- On ne peut tomber plus d'une fois sur une case bleue.
- Comment retrouver la frontière en interrogeant le moins de cases possible ?

  Partager en deux territoires équilibrés ne produit plus l'algorithme optimal

Problèmes instanciés et problèmes généraux

Exemple d'« algorithme instancié » (ressource IREM) :

début
  Donner à res la valeur 1
  pour *i de 1 à 10* **faire**
    Donner à res la valeur $res * i$
  fin
  Afficher res
fin

Algorithme 7 : Factorielle "Pour"

Un formalisme pour quoi faire ?

Exemple (ressource IREM) :

Algorithme 1 : Tri d'une série statistique

Entrées :
A : suite de nombres réels (série statistique)
Sorties :
B : suite obtenue en ordonnant A
début
  Trier A
  Afficher B
fin

Exemple de problème : recherche dans une liste

Problème (Indice d'insertion triée)

Étant donnés une liste croissante $S = (a_i)_{0 \leq i < n}$ de n éléments comparables deux à deux et une valeur b, déterminer le plus petit indice $0 \leq k < n$ tel que $b \leq a_k$, ou $k = n$ si $b > a_{n-1}$.

Autrement dit :
Étant donnée une liste croissante S, quelle est la plus petite position où on peut insérer un élément b en préservant l'ordre ?

Solutions à suivre...

Algorithmes fonctionnels au lycée

Objet central au lycée : l'algorithme comme fonction

- Comprendre en quoi un algorithme peut être vu comme réalisant une fonction mathématique
- Distinction claire entre saisie / affichage, entrée / sortie, paramètre / valeur renvoyée, instance / résultat
- Possibilité de récursivité
- Possibilité de généralisation par l'ajout de nouveaux paramètres

Exemples d'enjeux pour l'enseignement et la formation

Enjeu 2 : l'objet algorithme et l'objet programme

Algorithmes et programmes

Enjeux pour l'élève

- Reconnaître la description d'un algorithme dans divers registres
- Modifier ou formuler un algorithme dans divers registres
- Différencier la méthode de son implémentation par un programme (et compendre les enjeux associés)
- Construire une image mentale raisonnable de la « machine »
  opération élémentaire, modèle mémoire, état...

Enjeux pour l'enseignant

- Question du langage : comment « doit-on » décrire un algorithme ?
- Distinction claire entre questions algorithmiques et questions de programmation

Algorithmes et programmes

Definition (Programme informatique)

Suite d'instructions ou d'opérations obéissant à la syntaxe d'un langage de programmation, destinée à être exécutée par une machine selon une sémantique bien définie.

Problématiques propres à la programmation :

- Contraintes syntaxiques
- Performance empirique, détails d'implémentation
- Gestion de la mémoire, représentation des données
- Saisie, affichage, accès aux fichiers...

Écueil possible : amalgame entre problématiques algo et prog

Exemple : évolution des attentes au bac

2009 : Séparation nette entre Variables, Entrée(s), [Initialisation], Traitement, Sortie(s)

- Conventions reprises dans la plupart des manuels et au bac
- Confusion fréquente : saisie / paramètre, affichage / résultat

2017 : Volonté de simplification – note de l'IGEN

- Refonte des conventions de rédaction pour le Bac
- Plus aucune entrée-sortie, pas de rôle spécifique des variables (paramètres / locales), pas de résultat explicite
- Le problème résolu et le rapport entre instance et résultat n'est plus explicite et doit être détaillé par ailleurs

Exemple : évolution des attentes au bac

BAC ES 2017

1. Recopier et compléter l'algorithme de façon qu'il affiche le montant total des cotisations de l'année 2017.

```
Variables        S est un nombre réel
                 N est un entier
                 U est nombre réel
Initialisation   S prend la valeur 0
                 U prend la valeur 900
                 Pour N allant de 1 à 12 :
                     Affecter à S la valeur ...
                     Affecter à U la valeur 0,75 U + 12
                 Fin Pour
```

On propose simplement un changement de forme : suppression des étiquettes « Variables » et « Initialisation », suppression de la déclaration des variables, remplacement de la syntaxe d'une affectation.

$S \leftarrow 0$
$U \leftarrow 900$
Pour N allant de 1 à 12
  $S \leftarrow \ldots$
  $U \leftarrow 0{,}75\, U + 12$
Fin Pour

Exemple : évolution des attentes au bac

BAC STMG 2017

On considère l'algorithme suivant :

Variables n est un nombre entier
u et k sont des nombres réels
Traitement Saisir k
n prend la valeur 0
u prend la valeur 3 081,45
Tant que $u < k$ Faire
u prend la valeur $1{,}04 \times u$
n prend la valeur $n + 1$
Fin Tant que
Afficher n

Si l'on choisit $k = 4\,000$, quelle valeur affichera cet algorithme ? Interpréter ce résultat dans le contexte étudié.

On propose la suppression de la déclaration de variables et des entrées-sorties, la simplification de la syntaxe. Par cohérence d'un sujet à l'autre on propose de ne garder que : Tant que ...
plutôt que : Tant que ... faire

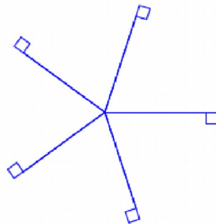
$n \leftarrow 0$
$u \leftarrow 3081{,}45$
Tant que $u<k$
$u \leftarrow 1{,}04 \times u$
$n \leftarrow n+1$
Fin Tant que

Quelle est la valeur de la variable n à la fin de l'exécution de cet algorithme si la valeur de la variable k en début d'exécution est égale à 4000 ? Interpréter ce résultat dans le contexte étudié.

Exemple : "rosaces au collège"

<u>Séance 1 :</u>

1) L'objectif de cette question est de dessiner la figure ci-contre (on l'appellera **figure 5**). Commence par te poser quelques questions…



Dessine à main levée le motif qui se répète dans cette figure (on l'appellera **M1**) :

Combien de fois est-il répété ?

……………………………………………………………………………………………………

Décris le plus précisément possible comment passer d'un motif à l'autre :

……………………………………………………………………………………………………
……………………………………………………………………………………………………

2) Ouvre le logiciel Scratch et construis ton propre programme pour dessiner le motif M1 (indication : la longueur du segment le plus grand est 100).
Attention, il faut que le lutin revienne à son point de départ, dans sa position initiale !

3) En t'aidant de la question 1), construis maintenant un programme qui permet de dessiner la **figure 5.** Teste-le.

Exemple : "rosaces au collège"

4) On veut maintenant construire une figure similaire mais avec 8 motifs M1.
Que doit-on changer dans le programme précédent?

…………………………………………………………………………………………………

Modifie ton programme et enregistre le sous la forme « rotation_figure8_prenom_nom ».

5) A la carte !!

On veut laisser l'utilisateur **choisir le nombre de motifs** dans la figure…

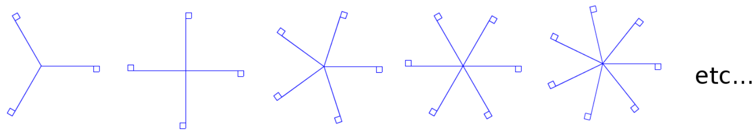
Reprends le dernier programme enregistré, puis programme ton lutin pour qu'il demande le nombre de motifs souhaités dans la figure finale :

- Insère une variable (on l'appellera « **nombre de motifs** ») dans ton programme.
- Comment peut-on calculer la mesure de l'angle entre deux motifs consécutifs M1 ?
  ……………………………………………………………………………………
- Modifie ton programme et enregistre-le sous la forme « rotation_figureN_prenom_nom ».

Fin de la séance 1

Exemple : "rosaces au collège"

Une alternative plus algorithmique :



Peut-on écrire un algorithme qui permet de produire cette famille de figures ?

Exemple (suite) : recherche dans une liste

Problème (Indice d'insertion triée)

Étant donnés une liste croissante $S = (a_i)_{0 \leq i < n}$ *de* n *éléments comparables deux à deux et une valeur* b*, déterminer le plus petit indice* $0 \leq k < n$ *tel que* $b \leq a_k$*, ou* $k = n$ *si* $b > a_{n-1}$*.*

Deux fois deux solutions :

- Recherche linéaire itérative
- Recherche linéaire récursive (en annexe)
- Recherche binaire (dichotomique) itérative
- Recherche binaire récursive (en annexe)

Solution 1 : recherche linéaire

Algorithme (Recherche linéaire itérative)

- *Pour i allant de 0 jusqu'à $n - 1$,*
  - *Si $b \leq a_i$, répondre i*
- *Si on n'a encore rien répondu, répondre n*

Autrement dit :

Parcourir la liste de gauche à droite indice par indice, et renvoyer l'indice courant dès qu'on atteint un élément supérieur ou égal à l'élément à insérer, ou la fin de la liste.

Solution 1 : recherche linéaire

```
def position(lst, elem):
    n = len(lst)
    for i in range(n):
        if elem <= lst[i]:
            return i
    return n
```

```
>>> position([], 42)
0
>>> position([1, 1, 1], 1)
0
>>> position([1, 2, 3, 4], 5)
4
>>> position(["allez", "marchez", "voyagez"], "volez")
2
```

Solution 2 : recherche binaire

Algorithme (Recherche binaire itérative)

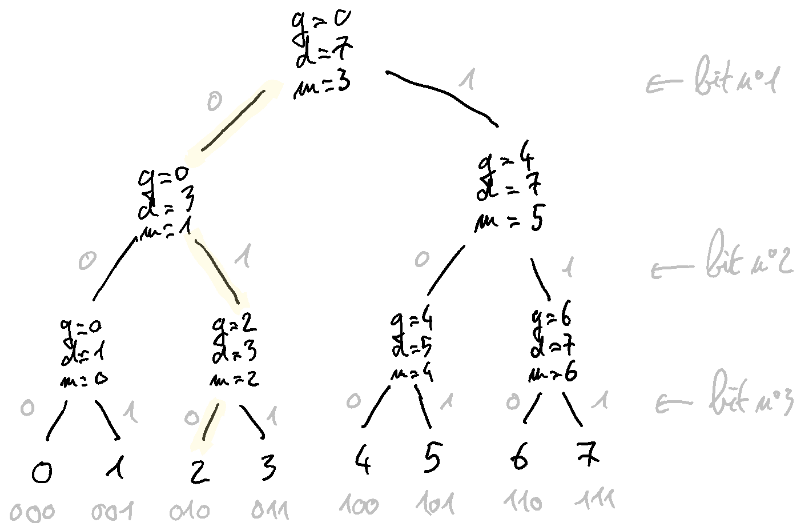
- *Poser* $g = 0$, $d = n$
- *Tant que* $d > g$ *:*
  - *Poser* $m = \lfloor (g + d)/2 \rfloor$
  - *Si* $b \leq a_m$, *poser* $d = m$
  - *Sinon, poser* $g = m + 1$
- *Répondre* g

Autrement dit :

Tant qu'il reste plus d'un indice possible, chercher l'indice médian. Si l'élément correspondant est plus grand que l'élément à insérer, éliminer les indices supérieurs, sinon éliminer les indices inférieurs. Renvoyer le dernier indice possible.

Solution 2 : recherche binaire

Exécutions possibles sur une liste à $7 = 2^3 - 1$ éléments :



Solution 2 : recherche binaire

```
def position(lst, elem):
    g, d = 0, len(lst)
    while g < d:
        m = (g + d) // 2
        if elem <= lst[m]:
            d = m
        else:
            g = m+1
    return g

>>> position([], 42)
0
>>> position([1, 1, 1], 1)
0
>>> position([1, 2, 3, 4], 5)
4
>>> position(["allez", "marchez", "voyagez"], "volez")
2
```

Exemples d'enjeux pour l'enseignement et la formation

Enjeu 3 : validation et preuve

Motivation

Au-delà de l'algorithmique comme panoplie d'outils :

- Prendre conscience de la nécessité d'une validation
- Pouvoir comparer plusieurs solutions du même problème
- Comprendre et pouvoir expliquer « pourquoi ça marche »
- Proximité de l'enjeu mathématique

Motivation

Au sujet de la boucle d'essai-erreur :

- Quels signes que la solution trouvée fonctionne ?
- Effectuer des tests : pas une tâche triviale !
  - Imaginer une instance qui peut montrer une erreur : semblable à la recherche de contre-exemple
  - S'assurer que l'ensemble des tests couvre tous les cas : semblable à une preuve par disjonction de cas
  - Comprendre une erreur constatée et la réparer
- Déjà un premier pas vers la preuve d'algorithme

Divers types de preuves d'algorithmes

Preuve de terminaison :

- L'algorithme s'arrête pour toute instance valide
- Techniques : décroissance dans un ordre bien fondé, convergence de suite...

Preuve de correction :

- Quand il s'arrête, l'algorithme fournit le bon résultat
- Techniques : preuve d'invariant, récurrence / induction

Complexité :

- Bornes supérieures et/ou inférieures sur les ressources
  en temps, en espace, au pire, en moyenne, ...
- Techniques : dénombrement, résolution de suites récurrentes, réductions, théorie de l'information...

Exemple de preuve d'algorithme : recherche linéaire

Algorithme (Recherche linéaire itérative)

- *Pour i allant de 0 jusqu'à $n-1$,*
  - *Si $b \leq a_i$, répondre i*
- *Si on n'a encore rien répondu, répondre n*

- Terminaison : trivial (boucle bornée)
- Correction : preuve d'invariant par récurrence
  à chaque début d'itération, indice recherché compris entre i et n
- Complexité : au pire n tours de boucle

Algorithme (Recherche binaire itérative)

- *Poser* $g = 0$, $d = n$
- *Tant que* $d > g$:
  - *Poser* $m = \lfloor (g + d)/2 \rfloor$
  - *Si* $b \leq a_m$, *poser* $d = m$
  - *Sinon, poser* $g = m + 1$
- *Répondre* g

Recherche binaire

Terminaison :

- La quantité $d - g$ décroît strictement à chaque itération car si $d > g$ alors $m < d$ et $m + 1 > g$
- Par un argument de descente infinie, il ne peut donc exister d'exécution infinie

Correction :

- Invariant de boucle : la position k recherchée est toujours comprise entre g et d
- Preuve par récurrence sur le nombre d'itérations

Recherche binaire

Complexité :

- Argument informel :
  - On coupe l'intervalle « en gros en deux » à chaque itération
  - On ne peut faire cela que $\log_2(n)$ fois environ
- Argument formel (cas simple) :
  - Supposons $n = 2^\ell$, au départ $d - g = n$
  - À chaque itération on divise $d - g$ exactement par deux
  - Après ℓ itérations $d - g = 1$, à l'étape suivante $d - g = 0$
  - Au total : $\ell + 1$ itérations
- Argument formel (cas général) : gestion des arrondis

Comparaison théorique

Exemple 1 : liste de 1000 éléments (pire cas)

- Recherche exhaustive : 1000 accès à la liste
- Recherche binaire : 10 accès à la liste
- Rapport : 100

Exemple 2 : liste de 10^6 éléments ($\times$1000)

- Recherche exhaustive : 10^6 accès à la liste ($\times$1000)
- Recherche binaire : 20 accès à la liste ($\times$2)
- Rapport : 50000 ($\times$500)

Différence exponentielle d'efficacité en théorie

Comparaison empirique

Exemple 1 : liste de 1000 éléments (pire cas)

- Recherche exhaustive : 64.4 µs en moyenne
- Recherche binaire : 2.28 µs en moyenne
- Rapport : environ 30

Exemple 2 : liste de 10^6 éléments (pire cas)

- Recherche exhaustive : 68.4 ms en moyenne ($\times$1000 environ)
- Recherche binaire : 4.49 µs en moyenne ($\times$2 environ)
- Rapport : environ 15000 ($\times$500 environ)

Observations compatibles avec le modèle théorique

Conclusion

Travaux en cours et à venir :

- Cartographie des notions, concepts et objets vivant à l'interface mathématiques-informatique
- Approche « diviser pour régner » : dichotomie et recherche binaire (une comparaison)
- Relations entre algorithme, preuve et programme pour l'enseignement
- Autour de la notion d'arbre syntaxique : travail sur les expressions algébriques

Conclusion

Hypothèses et propositions :

- Nécessité d'une prise en compte de l'épistémologie de l'informatique pour son enseignement et ses interactions avec les mathématiques
- Nécessité d'identifier les savoirs informatiques à enseigner
- Importance d'une formation suffisante des enseignants
- Difficulté à « plaquer » un contenu algorithmique sur certaines notions des programmes de mathématiques
- Identifier et faire vivre les champs à l'interface des mathématiques et de l'informatique

Annexe

Annexe

Recherche linéaire récursive

Recherche linéaire récursive

Algorithme (Recherche linéaire récursive)

- *Si la liste est vide, répondre 0*
- *Si* $b \leq a_0$*, répondre 0*
- *Sinon, soit* i *la position d'insertion dans* S *privée de son premier élément, répondre* $i + 1$

Autrement dit :

Si la liste est vide ou le premier élément est supérieur à l'élément à insérer, le résultat est 0, sinon c'est la position d'insertion dans la queue de la liste, plus 1.

Recherche linéaire récursive

```python
def position(lst, elem):
    if len(lst) == 0 or elem <= lst[0]:
        return 0
    else:
        return 1 + position(lst[1:], elem)  # bof...
```

```python
>>> position([], 42)
0
>>> position([1, 1, 1], 1)
0
>>> position([1, 2, 3, 4], 5)
4
>>> position(["allez", "marchez", "voyagez"], "volez")
2
```

Recherche linéaire récursive

Défaut de la version précédente : `lst[1:]` n'est *pas* une opération élémentaire et prend un temps proportionnel à `len(lst)` !

Variante : on généralise légèrement en ajoutant un paramètre `debut`. Nouveau problème : calculer la position d'insertion *à partir de l'indice* `debut` *!*

```
def position(lst, elem, debut):
    if len(lst) == ... or elem <= lst[...]:
        return ...
    else:
        return ...
```

Appel principal (recherche dans la liste entière) :
`position([1, 2, 3], b, 0)`

Recherche linéaire récursive

Défaut de la version précédente : `lst[1:]` n'est *pas* une opération élémentaire et prend un temps proportionnel à `len(lst)` !

Variante : on généralise légèrement en ajoutant un paramètre `debut`. Nouveau problème : calculer la position d'insertion *à partir de l'indice* `debut` !

```
def position(lst, elem, debut):
    if len(lst) == debut or elem <= lst[debut]:
        return debut
    else:
        return position(lst, elem, debut+1)
```

Appel principal (recherche dans la liste entière) :
`position([1, 2, 3], b, 0)`

Recherche linéaire récursive

Algorithme (Recherche linéaire récursive)

- *Si la liste est vide, répondre 0*
- *Si* $b \leq a_0$*, répondre 0*
- *Sinon, soit* i *la position d'insertion dans* S *privée de son premier élément, répondre* $i + 1$

- Terminaison : la taille de la liste décroît strictement
- Correction : par récurrence sur le nombre d'appels
  - 1 appel : S est vide ou $b \leq a_0$, donc 0 est la bonne réponse
  - Plus d'un appel : par h.r., i est la bonne réponse pour S privée de a_0, donc $i + 1$ est la bonne réponse pour S
- Complexité : au pire $n + 1$ appels en comptant le premier

Annexe

Recherche binaire récursive

Recherche binaire récursive

On généralise légèrement le problème : recherche de position d'insertion dans S entre les indices g et d, $0 \leq g \leq d \leq n$

Algorithme (Recherche binaire récursive)

- *Si* $g = d$, *répondre* g
- *Posons* $m = \lfloor (g + d)/2 \rfloor$
- *Si* $b \leq a_m$, *chercher dans* S *entre* g *et* m *et répondre*
- *Sinon, chercher dans* S *entre* $m + 1$ *et* d *et répondre*

Autrement dit :
S'il ne reste plus qu'un indice possible, le renvoyer. Sinon, si l'élément d'indice médian est plus grand que l'élément à insérer, chercher jusqu'à cet indice inclus, sinon chercher au-delà.

Recherche binaire récursive

```
def position(lst, elem):
    def aux(g, d):
        if d == g:
            return g
        else:
            m = (g + d) // 2
            if elem <= lst[m]:
                return aux(g, m)
            else:
                return aux(m+1, d)

    return aux(0, len(lst))
```

```
>>> position([], 42)
0
>>> position([1, 1, 1], 1)
0
>>> position([1, 2, 3, 4], 5)
4
>>> position(["allez", "marchez", "voyagez"], "volez")
2
```

Recherche binaire récursive

Algorithme (Recherche binaire récursive)

- *Si* $g = d$, *répondre* g
- *Posons* $m = \lfloor (g + d)/2 \rfloor$
- *Si* $b \leq a_m$, *chercher dans* S *entre* g *et* m *et répondre*
- *Sinon, chercher dans* S *entre* $m + 1$ *et* d *et répondre*

- Terminaison : la taille de la liste décroît strictement
- Correction : par récurrence sur le nombre d'appels
- Complexité : au pire $\lceil \log_2(d - g + 1) \rceil$ appels