Articulations preuves – algorithmes – programmes

Questions didactiques à l'interface mathématiques – informatique

Antoine Meyer[1] Simon Modeste[2]

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

[2]I3M / DÉMa, Université Montpellier II

Journée "Interface math-info", 17 janvier 2016

UNIVERSITÉ
DE MONTPELLIER

Avant de commencer...

Question

C'est quoi, pour vous, le rapport entre l'informatique et les math ?

Avant de commencer...

Question

C'est quoi, pour vous, le rapport entre l'informatique et les math ?

Micro-trottoir fantasmé :

- "Je ne veux pas en (entendre) parler"

Avant de commencer...

Question

C'est quoi, pour vous, le rapport entre l'informatique et les math ?

Micro-trottoir fantasmé :

- "Je ne veux pas en (entendre) parler"
- "Geogebra et LibreOffice"

Avant de commencer...

Question

C'est quoi, pour vous, le rapport entre l'informatique et les math ?

Micro-trottoir fantasmé :

- "Je ne veux pas en (entendre) parler"
- "Geogebra et LibreOffice"
- "Algobox, HP et Casio"

Avant de commencer...

Question

C'est quoi, pour vous, le rapport entre l'informatique et les math ?

Micro-trottoir fantasmé :

- "Je ne veux pas en (entendre) parler"
- "Geogebra et LibreOffice"
- "Algobox, HP et Casio"
- "Piaget, Papert et Logo"

Avant de commencer...

Question

C'est quoi, pour vous, le rapport entre l'informatique et les math ?

Micro-trottoir fantasmé :

- "Je ne veux pas en (entendre) parler"
- "Geogebra et LibreOffice"
- "Algobox, HP et Casio"
- "Piaget, Papert et Logo"
- "Scratch (c'est nul, et je déteste ce chat)"

Avant de commencer...

Question

C'est quoi, pour vous, le rapport entre l'informatique et les math ?

Micro-trottoir fantasmé :

- "Je ne veux pas en (entendre) parler"
- "Geogebra et LibreOffice"
- "Algobox, HP et Casio"
- "Piaget, Papert et Logo"
- "Scratch (c'est nul, et je déteste ce chat)"
- "Encore un truc que nous impose qui vous savez"

Avant de commencer...

Question

C'est quoi, pour vous, le rapport entre l'informatique et les math ?

Micro-trottoir fantasmé :

- "Je ne veux pas en (entendre) parler"
- "Geogebra et LibreOffice"
- "Algobox, HP et Casio"
- "Piaget, Papert et Logo"
- "Scratch (c'est nul, et je déteste ce chat)"
- "Encore un truc que nous impose qui vous savez"
- "Aucun, l'info c'est l'affaire des profs de techno"

Avant de commencer...

Question

C'est quoi, pour vous, le rapport entre l'informatique et les math ?

Micro-trottoir fantasmé :

- "Je ne veux pas en (entendre) parler"
- "Geogebra et LibreOffice"
- "Algobox, HP et Casio"
- "Piaget, Papert et Logo"
- "Scratch (c'est nul, et je déteste ce chat)"
- "Encore un truc que nous impose qui vous savez"
- "Aucun, l'info c'est l'affaire des profs de techno"
- "L'info est une branche des math"

Interactions math – info

Dans la recherche :

- Histoire partagée, apports mutuels ?
- Notions et thématiques communes, voisines, spécifiques ?
- Préoccupations, approches, méthodes, raisonnements ?

Dans l'enseignement, la formation :

- Math et info dans le secondaire ? Dans le supérieur ?
- Informatique outil vs. discipline informatique ?
- Provenance et formation des enseignants ?

Dans la classe (de math, ou ailleurs) :

- Quelles notions, à quel niveau, avec quels prérequis ?
- Quelles activités mêlant les deux disciplines ?
- Pour quels objectifs d'apprentissage ?

Nos questions de recherche

Projet en deux parties non indépendantes

- Partie épistémologique : contribuer à identifier les champs, notions, conceptions, modes de pensée communs ou spécifiques aux deux disciplines et leurs effets l'une sur l'autre
- Partie didactique : contribuer à élaborer et expérimenter des ressources de formation et d'enseignement riches à l'interface math-info, au secondaire et au supérieur

Axes d'analyse

Valeurs mathématiques et valeurs informatiques

- Nombre mathématique ($\mathbb{N}$, $\mathbb{R}$...) et nombre informatique (`int`, `float`...)
- Types et structures de données, codage, représentation
- Types de données abstraits et jeu d'opérations
- Algorithmes idéaux (précision arbitraire) ou implémentés

Rôles et statuts des variables

- Math : inconnue, variable substitutive, variable formelle
- Logique : variable libre ou muette, quantification, portée
- Programmation : portée, durée de vie, *temporalité*, paramètres formels ou effectifs

Axes d'analyse

Logique, langage et nature du raisonnement

- Décomposition en sous-problèmes : parallèle entre lemme et sous-programme, encapsulation
- Récurrence / induction / récursion
- Preuves constructives et complexité / preuves d'existence
- Question de l'implicite et de l'explicite
- Niveau de langue / niveau de langage (de programmation)
- Question de l'abstraction

Dans cet exposé (restriction n°1)

Quelles interactions entre programme, algorithme et démonstration ?

Définitions

Démonstration ou preuve

Suite d'arguments visant à établir une proposition, à partir d'un ensemble de propositions déjà établies ou admises

Une preuve peut faire appel à différents types de raisonnements (disjonction de cas, preuve par l'absurde, récurrence...) en principe formalisables en logique

Preuve formelle

Preuve exprimée dans un système de déduction formelle (incluant axiomes et règles de déduction), contenant toutes les étapes logiques élémentaires permettant d'aboutir à la proposition souhaitée

Définitions

Algorithme

Procédure systématique de résolution de problème, qui permet de résoudre toute instance du problème en un nombre fini d'étapes et à l'aide d'un ensemble fini d'opérations élémentaires

Programme

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

Preuves, algorithmes, programmes

Programme $\leftrightarrow$ algorithme

- Un algorithme *peut* être implémenté sous forme de programme
- Un programme (ou « quasi-programme ») *peut* servir à décrire un algorithme

Algorithme $\leftrightarrow$ preuve

- L'analyse d'un algorithme (ou d'un programme) peut requérir une ou plusieurs démonstrations
- Une démonstration mathématique « constructive » peut contenir ou mener à un algorithme (implicite ou explicite)

Preuve formelle $\leftrightarrow$ programme

- Cas des assistants de preuve (en France : Coq)
- Vérification formelle de programmes

Preuves, algorithmes, programmes

Plus généralement :

- Dialectique outil / objet, diverses conceptions de l'algorithme vis-à-vis des mathématiques (Modeste, 2012)
- Quels concepts, méthodes, types de raisonnements, champs d'application... communs ou spécifiques ?
- Quelles conséquences didactiques ?

Dans cet exposé (restriction n°2)

Quelques exemples de preuves d'algorithmes

Divers types de preuves d'algorithmes

Preuve de terminaison :

- L'algorithme s'arrête sur toute entrée annoncée comme valide
- Techniques : décroissance dans un ordre bien fondé, convergence de suite...

Preuve de correction :

- L'algorithme fournit le bon résultat quelle que soit l'instance
- Techniques : preuve d'invariant, récurrence / induction

Complexité :

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

Exemple 1 : algorithme d'Euclide

Problème : Soient $a \geq b \geq 0$ deux naturels, calculer le p.g.c.d. de a et b (plus grand naturel c divisant a et b ou 0 si a et b nuls)

Exemple 1 : algorithme d'Euclide

Problème : Soient $a \geq b \geq 0$ deux naturels, calculer le p.g.c.d. de a et b (plus grand naturel c divisant a et b ou 0 si a et b nuls)

Algorithme A_1

Soit la suite $(r_i)_{i \geq 0}$ telle que

$$\begin{cases} r_0 = a, r_1 = b \\ r_{i+2} = r_i \bmod r_{i+1}. \end{cases}$$

Résultat : premier r_n t.q. $r_{n+1} = 0$.

Exemple 1 : algorithme d'Euclide

Problème : Soient $a \geq b \geq 0$ deux naturels, calculer le p.g.c.d. de a et b (plus grand naturel c divisant a et b ou 0 si a et b nuls)

Algorithme A_1

Soit la suite $(r_i)_{i \geq 0}$ telle que

$$\begin{cases} r_0 = a, r_1 = b \\ r_{i+2} = r_i \bmod r_{i+1}. \end{cases}$$

Résultat : premier r_n t.q. $r_{n+1} = 0$.

Algorithme A_3

```
def pgcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a
```

Exemple 1 : algorithme d'Euclide

Problème : Soient $a \geq b \geq 0$ deux naturels, calculer le p.g.c.d. de a et b (plus grand naturel c divisant a et b ou 0 si a et b nuls)

Algorithme A_1

Soit la suite $(r_i)_{i \geq 0}$ telle que

$$\begin{cases} r_0 = a, r_1 = b \\ r_{i+2} = r_i \bmod r_{i+1}. \end{cases}$$

Résultat : premier r_n t.q. $r_{n+1} = 0$.

Algorithme A_2

Le résultat est a si $b = 0$, et le p.g.c.d. de b et $a \bmod b$ sinon.

Algorithme A_3

```
def pgcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a
```

Exemple 1 : algorithme d'Euclide

Problème : Soient $a \geq b \geq 0$ deux naturels, calculer le p.g.c.d. de a et b (plus grand naturel c divisant a et b ou 0 si a et b nuls)

Algorithme A_1

Soit la suite $(r_i)_{i \geq 0}$ telle que

$$\begin{cases} r_0 = a, r_1 = b \\ r_{i+2} = r_i \bmod r_{i+1}. \end{cases}$$

Résultat : premier r_n t.q. $r_{n+1} = 0$.

Algorithme A_2

Le résultat est a si $b = 0$, et le p.g.c.d. de b et $a \bmod b$ sinon.

Algorithme A_3

```
def pgcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a
```

Algorithme A_4

```
def pgcd(a, b):
    if b == 0:
        return a
    else:
        return pgcd(b, a % b)
```

Exemple 1 : algorithme d'Euclide

Terminaison : la suite des opérandes successifs est strictement décroissante dans l'ordre bien fondé $(\mathbb{N}, <)$

Correction (version récursive) : par récurrence complète sur b

- tout diviseur de a et b divise bien le pgcd de b et $a \bmod b$
- le pgcd de b et $a \bmod b$ divise bien a et b

Correction (version itérative) : par récurrence sur n

- pour tout $0 \leq i < n$, r_n divise r_i et r_{i+1} (en particulier a et b)
- tout c divisant a et b divise aussi tout r_i pour $i \leq n$

Exemple 1 : algorithme d'Euclide

Complexité – G. Lamé, 1844

Les plus petits nombres $a > b > 0$ tels que l'algorithme effectue n divisions sont les termes F_{n+2} et F_{n+1} de la suite de Fibonacci

Par récurrence sur n

- 0 divisions : $a = F_2 = 2$, $b = F_1 = 1$
- n divisions :
  - $\text{pgcd}(a, b) = \text{pgcd}(b, a \bmod b)$, obtenu en $n - 1$ divisions
  - par H.R., les plus petits b et $a \bmod b$ sont F_{n+1} et F_n
  - le plus petit a possible est atteint si $a \div b = 1$
  - on a alors $a = b + a \bmod b = F_{n+1} + F_n = F_{n+2}$

Exemple 1 : algorithme d'Euclide

Corollaire

Pour a et b quelconques, le nombre de divisions est borné par 5 fois le nombre de chiffres de b

- D'après le résultat précédent, la pire instance (c'est-à-dire la plus petite valeur de b) nécessitant n divisions est $b = F_{n+1}$
- Or $F_{n+1} = \frac{1}{\sqrt{5}}(\varphi^{n+1} - \varphi'^{n+1}) \geq \varphi^{n-1}$
- Donc $n - 1 \leq \log_\varphi b = \log_{10} b / \log_{10} \varphi$
- Comme $\log_{10} \varphi < 1/5$, on a $n \leq 5 \log_{10} b$

Exemple 2 : méthode de dichotomie

Approche majoritaire au lycée : recherche de zéro d'une fonction f par dichotomie

- Activité omniprésente (documents officiels, manuels...)
- Généralement accompagné de difficultés spécifiques :
  - Compréhension des notions d'analyse
  - Vérification des hypothèses d'applicabilité ?
  - Travail sur des nombres approchés
  - Problème parfois mal posé
  - Que calcule-t-on vraiment ?

Exemple 2 : méthode de dichotomie

Exercice tiré de Math'x 1ère S (Didier 2011) :

35 **ALGORITHMIQUE** @ TICE

1. Soit f la fonction définie sur $\mathbb{R}$ par $f(x) = x^3 - 3x^2 + 2$.

a. Étudier le sens de variation de f.

b. Calculer $f(-20)$, $f(30)$.

c. Déterminer le nombre de solutions de l'équation $f(x) = 0$ (justifier).

2. Un algorithme

VARIABLES : a, b nombres
ENTRÉES : Saisir a, b
TRAITEMENT :
  Si $a > b$ alors
    c prend la valeur b
    b prend la valeur a
    a prend la valeur c
  FinSi
  Si $f(b) \times f(a) \leqslant 0$ Alors
    Tantque $b - a > 10^{-5}$ Faire
      Si $f\left(\frac{a+b}{2}\right) \times f(a) \leqslant 0$
        Alors b prend la valeur $\frac{a+b}{2}$
        Sinon a prend la valeur $\frac{a+b}{2}$
      FinSi
    FinTantque
  FinSi
SORTIES : Afficher a et b

a. Que peut-on dire de $f(b)$ et de $f(a)$ lorsque $f(b) \times f(a) \leqslant 0$?

b. Que représente $\frac{a+b}{2}$ par rapport à a et b ?

c. Que fait cet algorithme ?

d. Programmer cet algorithme.

e. Que se passe-t-il si l'utilisateur entre les valeurs $a = -20$ et $b = 30$? $a = -20$ et $b = 3$? $a = 0$ et $b = 3$?

f. Comment peut-on s'assurer de trouver la solution souhaitée ?

Exemple 2 : méthode de dichotomie

En analyse :

- Stable mais non nécessairement optimal
  ("seulement" un chiffre par itération)
- Outil de démonstration (non constructif !)
  - Théorème des valeurs intermédiaires
  - Théorème de Bolzanno-Weierstrass...

Autres approches possibles :

- « Jeu de la devinette » : accessible à des élèves plus jeunes ou en approche préliminaire
- Recherche d'occurrence dans une liste triée : classique des cursus d'informatique, exploitation « objet » possible
- Autres applications du principe de dichotomie
- Principe général : "Diviser pour régner"

Exemple 2 : méthode de dichotomie

Problème *InsertionTriée* :

Étant donnée une liste croissante L et un élément e de L, déterminer le plus petit i tel que $L(i) \geq e$

Algorithme D

1. Calculer l'indice du milieu de la liste (arrondi par défaut)
2. Si l'élément à l'indice m est supérieur ou égal à e, rechercher e jusqu'à l'indice m, sinon le chercher à partir de l'indice $m + 1$

Affirmation : l'algorithme D résoud le problème *InsertionTriée*
(à démontrer... mais pas maintenant)

Exemple 2 : méthode de dichotomie

Programme récursif

```
def recherche(lst, e, debut, fin):
    if debut == fin:
        return debut
    else:
        milieu = (debut + fin) // 2
        if lst[milieu] >= e:
            return recherche(lst, e, debut, milieu)
        else:
            return recherche(lst, e, milieu+1, fin)
```

Exemple 2 : méthode de dichotomie

Programme itératif

```
def recherche(lst, e):
    debut = 0
    fin = len(lst)
    while debut != fin:
        milieu = (debut + fin) // 2
        if lst[milieu] >= e:
            fin = milieu
        else:
            debut = milieu+1
    return debut
```

Exemple 2 : méthode de dichotomie

Complexité dans le pire cas : borne supérieure

Sur une liste L de taille $0 < n < 2^k$, l'algorithme D effectue au plus k comparaisons

Preuve par récurrence sur $k > 0$

- Vrai pour $k = 1$ (soit $n = 1$) : 1 seule comparaison
- Si c'est vrai jusqu'à un certain k, alors :
  - Soit L une liste de longueur $n < 2^{k+1}$
  - On compare $L(\lfloor n/2 \rfloor)$ à l'élément recherché
  - Dans tous les cas on continue sur une liste de longueur $< 2^k$
  - On conclut en au plus k comparaisons soit $k + 1$ en tout

Exemple 2 : méthode de dichotomie

Complexité dans le pire cas : borne inférieure

Dans le modèle de l'arbre de comparaison, tout algorithme résolvant *InsertionTriée* sur une liste de longueur $n \geq 2^{k-1}$ effectue au moins k comparaisons

- Sur une liste de taille n, $n+1$ issues possibles (0 à n)
- Algo sur une telle liste : arbre binaire à au moins $n+1$ feuilles
- L'algorithme *garantissant* le nombre de comparaisons le plus petit correspond à l'arbre le plus "tassé"
- Si $2^{k-1} \leq n < 2^k$, cet arbre a au moins une branche de longueur k

Conclusion

L'algorithme D est optimal dans ce modèle de calcul