Place de l'algorithmique en mathématiques

Luc Sanselme, Lycée Henri Poincaré à Nancy

UMR 8623 LRI (Laboratoire de recherche en informatique)

Résumé

Dans cet exposé, nous allons essayer de comprendre la place de l'algorithmique en mathématiques. Nous commencerons par un historique de l'algorithmique : nous chercherons entre autre à replacer son histoire dans celle des mathématiques et de l'informatique. Nous continuerons avec une introduction à l'algorithmique qui aura pour but de poser les bases de ce qu'est cette science aujourd'hui .Ensuite, nous essaierons de dégager les liens et les différences entre les branches plus classiques des mathématiques et l'algorithmique : nous tenterons de mettre en avant comment ces mathématiques plus traditionnelles et l'algorithmique peuvent se nourrir mutuellement. Nous terminerons en exposant quelques pistes et quelques idées sur la façon de présenter l'algorithmique au grand publique et en particulier aux lycéens.

Le texte présenté ci dessous reprend les diapositives de l'exposé réalisé lors du colloque.

Structure de l'exposé

- Un peu d'histoire...
- Les machines de Turing, modèle de référence en calculabilité et en complexité
- Vers un formalisme simplifié
- Eléments didactiques

La démarche algorithmique au fil des siècles

- Averroès (12ème siècle) : méthode de raisonnement qui s'affine étape par étape jusqu'à une certaine convergence
- Descartes : "diviser chacune des difficultés que j'examinerois, en autant de parcelles qu'il se pourroit, et qu'il seroit requis pour les mieux résoudre"
- la descente infini de fermat au 17ème siècle

- Constructions géométriques

Historique

- Quelques éléments d'histoire des algorithmes
- Quelques repères dans l'évolution des machines de calculs
- L'algorithmique, de sa naissance à nos jours
  - Naissance de la calculabilité
  - Naissance de la complexité
  - Etat de l'art et perspectives

Les algorithmes au fil des siècles

"Algorithme" de Algorismi, surnom latin du mathématicien arabe Abu Ja'far Mohammed Ben Mussa Al-Khwarismi (780-850).

Algorithmes célèbres

- calculs commerce et impôts chez les Babyloniens (vers -1800/-1600)
- Algorithme d'Euclide (4ème siècle avant JC)
- Algorithme d'Archimède (3ème siècle avant JC)
- Crible d'Eratosthènes (3ème siècle avant JC)
- Méthode du Pivot de Gauss (1er siècle chez les Chinois)
- Code de Vigenère (16ème siècle)
- Machine enigma (Deuxième guerre mondiale)
- RSA (1977)

L'ère de l'informatique et l'omniprésence des algorithmes

Raisons

- Intérêt de systématiser certaines opérations
- Puissance de calcul des ordinateurs

Tous les domaines

- Economie : stratégies d'investissement, gestion des stocks,...
- Industrie : procédés de fabrication,...
- Simulation : météorologie, aérodynamisme, ... (Sciences)
- Cryptographie et compressions de données
- Gestion de grosses masses d'informations : bases de données,...
- Traitement d'images et de sons

Histoire des machines de calcul

- Boulier Chinois (3000 avant JC)
- 1642 Calculatrice de Blaise Pascal
- 1940 "Bombes" de Turing
- 1946 ENIAC (Electronic Numerical Integrator and Computer)
- 1947 Invention du transistor
- 1948 : Manchester Mark I
- 1953 IBM lance son premier ordinateur commercial en série, l'IBM 650
- 1965 Loi de Moore : capacité sur les puces double tous les ans
- 1971 Intel met en vente le premier microprocesseur
- 1975 Loi de Moore : le nombre de transistor dans un microprocesseur double tous les deux ans (densité double)
- 1975 Texas Instruments présente sa première calculatrice de poche programmable, la TI SR 52
- 1981 IBM lance son 5150 Personal Computer

Naissance de la calculabilité

- Les fondements de l'algorithmique datent des années 1930
- Liés à la construction formelle des mathématiques et de la logique
- Donner un cadre formel à la théorie de la démonstration
- Définir ce qu'est une fonction calculable

- Thèse de Church (Kleene en 1943)/Turing-Church (années 1990)
  - 1933 Church et le λ-calcul
  - 1936 Turing et les "Machines de Turing"

La thèse de Turing-Church

La thèse

Les machines de Turing (les fonctions λ-définissables, les fonctions récursives ...) formalisent correctement la notion de méthode effective de calcul.

Méthode effective

1. Ensemble fini d'instructions simples et précises, décrites avec un nombre fini de symboles,
2. nombre fini d'étapes,
3. suivi par un humain avec seulement du papier et un crayon,
4. l'exécution ne requiert pas d'intelligence de l'humain sauf celle nécessaire pour comprendre et exécuter les instructions.

La thèse de Turing-Church

Succès de la thèse

- Depuis 1940, notion de fonction calculable est bien définie
  - ↪ Début du XXe siècle : expressions informelles comme "effectivement réalisable"
- Tous formalisme "raisonnable" est équivalent à une Machine de Turing : Thèse renforcée
  - ↪ Encore vraie aujourd'hui

↪ Précise la notion d'algorithme et fonde l'algorithmique (calculabilité)

Naissance de la complexité

Années 1960 – 1970, pour mesurer l'efficacité d'un algorithme :

"Il s'exécute en tant de temps sur une machine"

Problème

- Dépend de la machine
- Dépend du langage de programmation
- Dépend de la façon dont est implanté l'algorithme

Notion de complexité

- Modèle théorique (Machines de Turing,...)
  - ↪ En temps : nombre d'opérations pour arriver au résultat
  - ↪ En espace : nombre de cases nécessaires sur le ruban
- Le plus souvent : évaluation asymptotique en fonction de l'augmentation de la taille des données
  - ↪ En pratique : bon modèle
- Etudie aussi la complexité de problèmes

Etat de l'art et perspectives

- Machines de Turing toujours modèle utilisé en calculabilité
  - ↪ Thèse de Turing-Church toujours valable
- Domaine de recherche très actif
- Divers scénarios qui font évoluer les modèles de complexité :
  - Plusieurs ordinateur, qui communiquent (lien avec la théorie de l'information)
  - Source aléatoire
  - scénarios proposés par la cryptographie...

- Problème de la miniaturisation des transistors (effet tunnel)
  - ↪ Comment lutter contre cet effet ?
  - ↪ Tirer bénéfices des propriétés de la mécanique quantique
- Calcul quantique : plus rapide ?
  - Algorithme de Grover : gain quadratique prouvé
  - Factorisation de Shor (1994) : gain sous-exponentiel supposé

Les machines de Turing, modèle de référence en calculabilité et en complexité

- Présentation des machines de Turing
- Introduction à la théorie de la complexité

Un modèle de calculabilité : les machines de Turing

- Présentation intuitive des machines de Turing
- Un premier exemple
- Définition formelle des machines de Turing
- Un deuxième exemple
- Machines de Turing non déterministes
- Variantes des machines de Turing

Présentation intuitive des machines de Turing

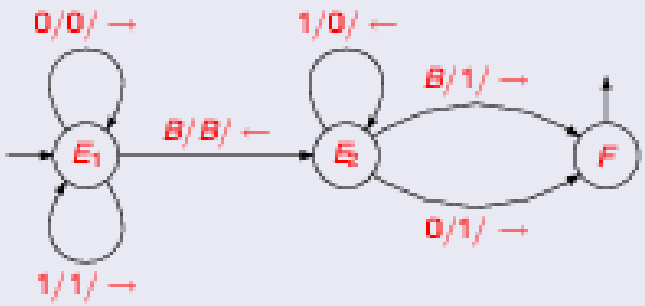
Qu'est-ce qu'une machine de Turing ?

Etant donné un alphabet qui contient un symbole "Blanc".

1. Un "ruban" : "longueur infinie",déplacement gauche ou droite, divisé en cases;
2. Une "tête de lecture/écriture" : lit, écrit et se déplacer à gauche ou à droite,
3. Nombre fini d'états : un initial et des finaux,
4. Un "registre d'état" : mémorise l'état courant,
5. Une "table d'actions" : quel symbole écrire, direction de la tête de lecture (gauche ou droite), et le nouvel état, en fonction du symbole lu.

Un premier exemple

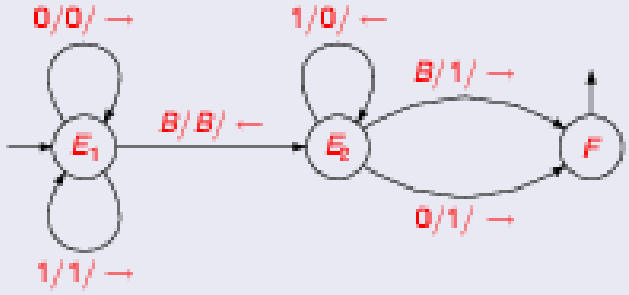
La fonction $f : n \mapsto n+1$ Alphabet : $\{0, 1, B\}$



B	B	B	1	0	1	B	B	B	B	E_1
B	B	B	1	0	1	B	B	B	B	E_1
B	B	B	1	0	1	B	B	B	B	E_1
B	B	B	1	0	1	B	B	B	B	E_1
B	B	B	1	0	1	B	B	B	B	E_2
B	B	B	1	0	0	B	B	B	B	E_2
B	B	B	1	1	0	B	B	B	B	F

Un premier exemple : en langage courant...

La fonction $f : n \mapsto n+1$ Alphabet : $\{0, 1, B\}$



L'algorithme

1. Tant que $CC \neq B$, se déplacer vers la droite,
2. Se déplacer vers la gauche,
3. Tant que $CC = 1$:
   1. Remplacer CC par 0,
   2. Se déplacer vers la gauche,
4. Remplacer CC par 1,
5. Se déplacer vers la droite.

Définition formelle d'une machine de Turing

Definition

Une machine de Turing déterministe est un septuplet $(Q, \Gamma, B, \Sigma, q_0, \delta, F)$ où

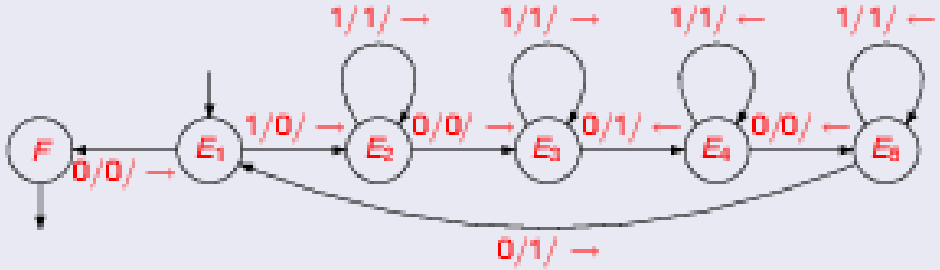
- Q est un ensemble fini d'états,
- Γ est l'alphabet de travail des symboles de la bande,
- $B \in \Gamma$ est un symbole particulier (dit blanc),
- Σ est l'alphabet des symboles en entrée ($\Sigma \subseteq \Gamma \setminus \{B\}$),
- $q_0 \in Q$ est l'état initial,
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ est la fonction de transition,
- $F \subseteq Q$ est l'ensemble des états acceptants (ou finaux, terminaux).

Un deuxième exemple

La fonction

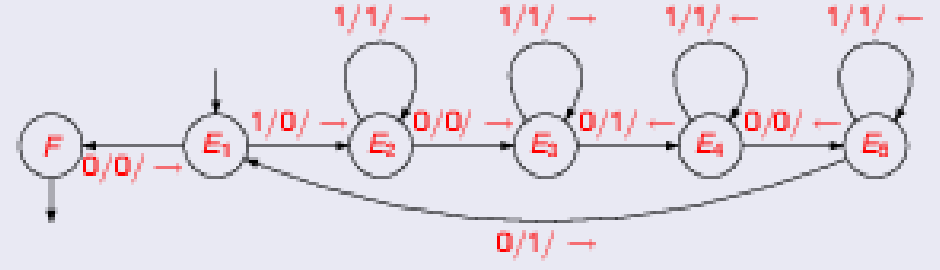
- Alphabet $\{0, 1\}$, 0 étant le "blanc",
- Le ruban contient une série de 1,
- la tête de lecture/écriture se trouve initialement au-dessus du 1 le plus à gauche,
- Cette machine a pour effet de doubler le nombre de 1, en intercalant un 0 entre les deux séries.
- Exemple : 111 devient 1110111.

Un deuxième exemple



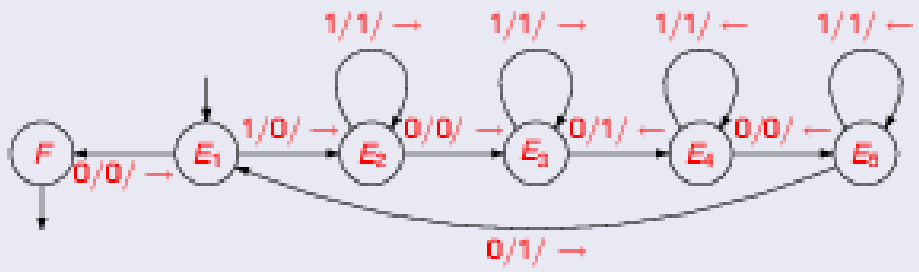
0	0	1	1	0	0	0	E_1
0	0	0	1	0	0	0	E_2
0	0	0	1	0	0	0	E_2
0	0	0	1	0	0	0	E_3
0	0	0	1	0	1	0	E_4
0	0	0	1	0	1	0	E_5
0	0	0	1	0	1	0	E_5

Un deuxième exemple



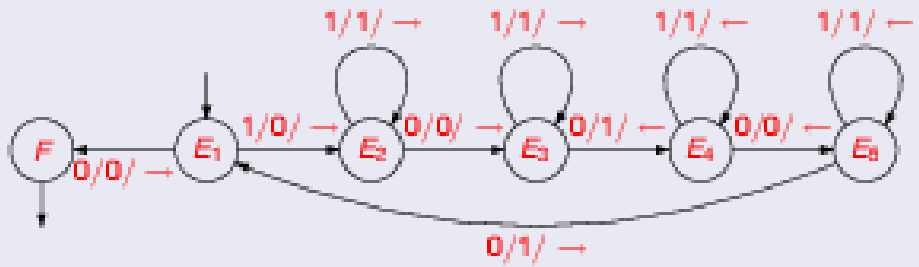
0	0	0	1	0	1	0	E_5
0	0	1	1	0	1	0	E_1
0	0	1	0	0	1	0	E_2
0	0	1	0	0	1	0	E_3
0	0	1	0	0	1	0	E_3
0	0	1	0	0	1	1	E_4
0	0	1	0	0	1	1	E_4

Un deuxième exemple



0	0	1	0	0	1	1	E_4
0	0	1	0	0	1	1	E_5
0	0	1	1	0	1	1	E_1
0	0	1	1	0	1	1	F

Un deuxième exemple : en langage courant...



L'algorithme

1. Tant que $CC \neq 0$
   1. Remplacer CC par 0/Aller à droite
   2. Aller à droite Jusqu'à ce que $CC = 0$
   3. Aller à droite Jusqu'à ce que $CC = 0$
   4. Remplacer CC par 1/Aller à gauche
   5. Aller à gauche Jusqu'à ce que $CC = 0$
   6. Tant que $CC = 1$ Aller à gauche
   7. Remplacer CC par 1/Aller à droite
2. Aller à droite

Un deuxième exemple : en langage courant...

0	1	1	1	1	...	1	1	1	0	0	0	0	...	0	0	0
0	0	1	1	1	...	1	1	1	0	0	0	0	...	0	0	0
0	0	1	1	1	...	1	1	1	0	1	0	0	...	0	0	0
0	1	1	1	1	...	1	1	1	0	1	0	0	...	0	0	0
0	1	0	1	1	...	1	1	1	0	1	0	0	...	0	0	0
0	1	0	1	1	...	1	1	1	0	1	1	0	...	0	0	0
0	1	1	1	1	...	1	1	1	0	1	1	0	...	0	0	0
0	1	1	0	1	...	1	1	1	0	1	1	0	...	0	0	0
⋮					⋮								⋮			⋮
0	1	1	1	1	...	1	0	1	0	1	1	1	...	1	0	0
0	1	1	1	1	...	1	1	1	0	1	1	1	...	1	0	0
0	1	1	1	1	...	1	1	0	0	1	1	1	...	1	0	0
0	1	1	1	1	...	1	1	0	0	1	1	1	...	1	1	0
0	1	1	1	1	...	1	1	1	0	1	1	1	...	1	1	0

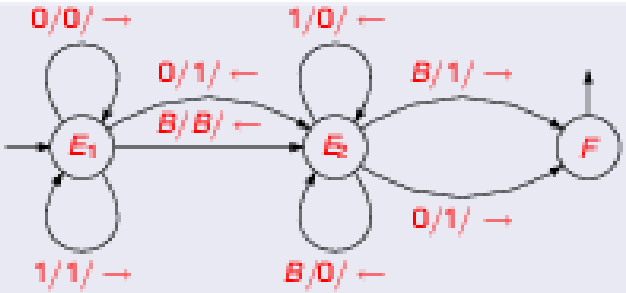
Machines de Turing non-déterministes

Definition

Une machine de Turing non-déterministe est un septuplet $(Q, \Gamma, B, \Sigma, q_0, \Delta, F)$ où

- Q est un ensemble fini d'états,
- Γ est l'alphabet de travail des symboles de la bande,
- $B \in \Gamma$ est un symbole particulier (dit blanc),
- Σ est l'alphabet des symboles en entrée ($\Sigma \subseteq \Gamma \setminus \{B\}$),
- $q_0 \in Q$ est l'état initial,
- $\Delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{\leftarrow, \rightarrow\})$ est une relation vérifiant : $\forall (q_1, x) \in (Q \times \Gamma)$,
$$\exists (q_2, y, f) \in (Q \times \Gamma \times \{\leftarrow, \rightarrow\});\ ((q_1, x), (q_2, y, f)) \in \Delta,$$
- $F \subseteq Q$ est l'ensemble des états acceptants (ou finaux, terminaux).

Machines de Turing non-déterministes



Remarques

- Ce n'est plus un modèle de calcul "réaliste"
- Par contre, si l'on nous donne un trajet, on peut vérifier si on arrive dans un état final et donc vérifier si la réponse à un un problème est bien oui (mais pas non)
- Mêmes fonctions calculables que pour une machine de Turing déterministe
- A la base du modèle des machines de Turing probabilistes : machines de Turing non-déterministe + répartition de probabilité pour choisir celle des différentes branches possibles
  - → Compliqué : pour simplifier on ne parlera que des non-déterministes

Variantes des machines de Turing

Machines de Turing universelles

Evolutions possibles

- Nombre de bandes
- Bandes en lecture seule ou en écriture seule
- Bande simplement infini

Evolution très utilisée

- Machine RAM (Random Access Machine)

Introduction à la théorie de la complexité

- De la complexité d'un algorithme à celle d'un problème algorithmique
- Quatre familles de classes
- Principales classes
- Réduction de problèmes
- Structures de ces classes

De la complexité d'un algorithme à celle d'un problème algorithmique

Modèle

Les deux modèles les plus utilisés en théorie de la complexité sont :

- La machine de Turing (déterministe ou non),
- La machine RAM (Random Access Machine).

→ Modèles "équivalents"

Complexité d'un algorithme

- Estimation, théorique, des ressources informatiques nécessaires.
- En temps : nombre d'étapes nécessaires pour effectuer le calcul
- En espace : nombre de cases nécessaires sur le ruban pour effectuer le calcul
- Fonction de la taille des données en entrée/ Etude asymptotique

Complexité d'un problème algorithmique

- Vise à savoir si la réponse à un problème peut être donnée efficacement
- Hiérarchies de difficultés entre les problèmes algorithmiques : "classes de complexité"

Quatre familles de classes

n : Taille des données en entrée

- $TIME(t(n))$: classe des problèmes résolus en temps de l'ordre de $t(n)$ sur une machine déterministe.
- $NTIME(t(n))$: classe des problèmes résolus en temps de l'ordre de $t(n)$ sur une machine non déterministe.
- $SPACE(s(n))$: classe des problèmes résolus avec un espace de l'ordre de $s(n)$ sur une machine déterministe.
- $NSPACE(s(n))$: classe des problèmes résolus avec un espace de l'ordre de $s(n)$ sur une machine non déterministe.

Principales classes

n taille des données

- Classe $P = TIME(n^k, k \in \mathbb{N})$ (déterministe)
  - → Exemples : connexité dans un graphe, test de primalité (2002),...
  - → Stable par "composition".
  - → Classe des problèmes que l'on peut résoudre "efficacement".
  - → Différents "modèles" équivalents pour cette classe

- Classe NP équivalent non-déterministe de P
  - → SAT (Boolean Satisfiability Problem) : problème de savoir si une formule logique admet une instance pour laquelle elle est vraie est NP
  - → Autres exemples : Tester si deux graphes sont isomorphes, sac-à-dos, voyageur de commerce, factorisation,...
  - → Classe des problèmes pour lesquels on peut "tester si une solution est vraie en temps polynomial"

- Classe $CoNP$ (complémentaire de NP)
  - → Exemple : Une formule logique donnée est-elle toujours vraie ?
- Classe $PSPACE$, $NPSPACE$, $EXPTIME$

Réduction de problèmes

Réduction de problèmes

- Ramener la résolution d'un problème à la résolution d'un autre
- Algorithme de réduction
- Cette réduction doit être polynomiale et déterministe (le plus souvent)
- A se réduit à un problème B : A est plus facile que B

→ Également, réduction d'un "modèle" à un autre

Problèmes C-complets et C-difficiles

- C une classe de complexité (P, NP, $PSPACE$, ...).
- Un problème est C-difficile s'il est au moins aussi difficile que tous les problèmes dans C.
- Un problème C-difficile qui appartient à C est dit C-complet.

Théorème de Cook-Levin

- SAT est NP-complet

Structure de ces classes

Hiérarchie

- $P \subseteq NP \subseteq PSPACE \subseteq NPSPACE \subseteq EXPTIME$,
- $P \subseteq CoNP \subseteq PSPACE \subseteq NPSPACE \subseteq EXPTIME$,
- $PSPACE = NPSPACE$ (Théorème de Savitch)
- $P \subsetneq EXPTIME$.

Problèmes ouverts

- $P = NP$?
- $NP \neq CoNP$?

Vers un formalisme simplifié

- Notions élémentaires en algorithmique :
  - Problème
  - Algorithmes et syntaxe
  - Preuve de terminaison et de correction
  - Complexité
- Objets élémentaires en algorithmique
- Stratégies élémentaires en algorithmique

Notions élémentaires : le problème

Notion de problème

- un problème est une question générique
- chaque instance du problème a une réponse
- la notion de problème est indépendante de la notion de programme

Exemples de problèmes

- Déterminer si un nombre est pair ou impair est un problème,
- Trier un tableau est un problème
- Déterminer si un programme écrit dans un langage s'arrête est un problème appelé problème de l'arrêt.
- Déterminer si un polynôme (en plusieurs variables) à coefficients entiers a des racines entières est un problème (dixième problème de Hilbert)

Notion élémentaire : l'algorithme

Notion d'algorithme

Un algorithme est

- un processus systématique de résolution, par le calcul, d'un problème
- présente les étapes vers le résultat à une autre personne physique (un autre humain) ou virtuelle (un calculateur).
- une suite finie et non-ambigüe d'opérations "élémentaires" permettant de donner la réponse à un problème.

Notion de syntaxe

- Un algorithme est un "texte" écrit dans un langage.
- "Textes" dans ce langage : suivent des règles.
- Le "texte" représentant un algorithme est la syntaxe de cet algorithme.

→ Syntaxe doit être correcte.

Notion élémentaire : l'algorithme

Exemples d'algorithmes

- Déterminer si un nombre est pair ou impair :
  Regarder son dernier chiffre en binaire pour savoir si il est pair ou impair est un algorithme
- Trier un tableau :
  Tri insertion, tri bulle, tri rapide, tri fusion, tri par tas.. sont des algorithmes
- Déterminer si un programme écrit dans un langage s'arrête :
  Aucun algorithme ne résoud ce problème
  Problème de Syracuse :

$$f : n \mapsto \begin{cases} 0 & \text{si } n = 1 \\ f(n/2) & \text{si } n \text{ pair} \\ f(3n+1) & \text{si } n \text{ impair} \end{cases}$$

- Déterminer si un polynôme (en plusieurs variables) à coefficients entiers a des racines entières est un problème (dixième problème de Hilbert)
  Aucun algorithme ne résoud ce problème

Notions élementaires : preuve de terminaison et de correction

Validité d'un algorithme

Pour qu'un algorithme soit valide, il faut prouver que :

- sa syntaxe est correct
- il termine (toujours) : preuve de terminaison
- il donne la réponse au problème posé (toujours) : preuve de correction

Notions élementaires : preuve de terminaison et de correction

Pour la syntaxe

- Bonnes habitudes de présentation
- Débugger en mettant en commentaires

Pour la terminaison

- Pour une boucle : fonction valeurs entières strictement décroissante
- Tester différentes valeurs correspondant aux différents "circuits"

Pour la correction

- Pour une boucle : propriété vraie à chaque passage dont l'instance au dernier passage est la propriété voulue
- Comprendre ce que chaque variable "représente" (à chaque instant)
- Tester différentes valeurs correspondant aux différents "circuits"

↪ Tester les cas particuliers

Notions élementaires : la complexité

Notion de complexité

- Fonction qui à n associe $f(n)$, nombre maximum (ou moyen, mais plus compliqué) d'opérations pour les entrées de longueur n.
- Comportement asymptotique (constante multiplicative).
  Exemple : un entier n a une longueur de $\log_2(n)$ en binaire

Efficacité

Algorithme efficace : complexité polynomiale en la taille de l'entrée

Espace moins un problème que le temps : capacités des ordinateurs.

Objets élémentaires en algorithmique

Les structures de données

- constantes
- variables
- tableaux
- structures récursives (listes, arbres, graphes)

Les structures de contrôle

- séquences
- conditionnelles
- boucles

- Itérativité et récursivité
- "while" et "for", et bannir les "goto"

Stratégies élémentaires en algorithmique

L'algorithmique a développé quelques stratégies pour résoudre les problèmes

- Recherche exhaustive
- Algorithme glouton : rendu de pièce de monnaie (euro)
- Diviser pour régner : tri fusion
- Aléatoire : marches aléatoires
- Par approximations successives : dichotomie
- Décomposition top-down / bottom-up
- Pré-traitement (ou post-traitement) : crible d'ératosthène pour la primalité sur des petits entiers
- Heuristique : solution rapide, assez bonne, mais pas toujours optimale.

Eléments didactiques

- L'algorithmique, une des branches les plus récentes des mathématiques
- L'algorithmique comme outil didactique pour les mathématiques
- Didactique de l'algorithmique et évaluation
- Choix d'un langage/logiciel

L'algorithmique, une des branches les plus récentes des mathématiques

Deux domaines du 20ème siècle

- Les probabilités et les statistiques
- L'algorithmique et l'informatique théorique

Nouveaux modes de raisonnement

- Liés aux probabilités : algorithmes probabilistes,
- Notion d'induction,
- Mathématiques discrètes,
- Constructiviste,
- Formalisme et abstraction un peu différents,
- Introduit la notion de temps en mathématiques, qui était plutôt la science de l'éternité.

L'algorithmique comme outil didactique pour les mathématiques

Outil pour comprendre les objets traditionnels des mathématiques

- La notion de variable/inconnue
- La notion de fonction

Notion de variable et d'inconnue

- Pas de différence entre une variable "mathématique" et une variable "informatique"
- En mathématiques :
  - Pas de distinction entre $:=$ et $=$.
  - "Type" pas les variables explicitement.
  - Quantificateurs définissent le type.
  - Pas besoin de "prévoir" l'espace mémoire.
- En informatique :
  - Quantifie pas (ou peu).
  - Constructiviste : $\exists$ "disparait".
  - Ensemble qu'elle parcourt donné par son type (sinon binaire) : $\forall$ disparait.
- Différence due à l'utilisation : ne nécessite pas de la rigueur au mêmes endroits.

L'algorithmique comme outil didactique pour les mathématiques

Outil pour comprendre les objets traditionnels des mathématiques

- La notion de variable/inconnue
- La notion de fonction

Notion de fonction

- Permet de comprendre qu'une fonction peut être "n'importe quoi",
  - → En mathématiques : fonctions ont toujours des "bonnes propriétés"
  - → En informatique : natures très diverses
  - → Créativité : pas limité à combiner des fonctions usuelles
- Les propriétés étudiées sont différentes : calculabilité, complexité

L'algorithmique comme outil didactique pour les mathématiques

Améliorer la méthodologie

- Rigueur rédactionnelle/Présentation,
- Analyse et méthodologie de la résolution.

Rigueur rédactionnelle/Présentation

- La rigueur rédactionnelle est obligatoire sur un ordinateur,
  - → Importance de la syntaxe,
  - → Sanction directe,
  - → En mathématiques : pas une lubie pour "embêter" les élèves,
  - → Rigueur avec les objets $f \neq f(x)$,
- Présentation des programmes, en pratique, indispensable,
  - → Structurer les démonstrations de la même façon,
- Découper un programme en modules,
  - → Faire de même avec une démonstration (lemme),

L'algorithmique comme outil didactique pour les mathématiques

Améliorer la méthodologie

- Rigueur rédactionnelle/Présentation,
- Analyse et méthodologie de la résolution.

Analyse et méthodologie de la résolution

- Découper un programme en modules,
  - → Faire de même avec une démonstration (lemme).
- Réduire un problème à un autre grâce à un algorithme,
  - → Faire de même en mathématiques pour démontrer une proposition
- Techniques de génie logiciel : du cahier des charges au logiciel,
  - → De l'énoncé à l'algorithme,
  - → De l'énoncé à la démonstration sur feuille (en passant par l'analyse et la rédaction).

Didactique de l'algorithmique et évaluation

- Travail sur machine
  - → La sanction est directe,
  - → Apprendre à corriger ses erreurs : autonomie,
  - → ...mais écrire les programmes sur papier avant (rapidement).
- Evaluer la rigueur rédactionnelle,
- Evaluer la présentation/structure des programmes,
  - → Sortir de l'idée qu'un bon programme est incompréhensible,
  - → ...au contraire.

Didactique de l'algorithmique et évaluation

- Travailler sur des programmes déjà faits : que fait-il, quelle complexité ?
  - → ...ou faits en partie et à compléter,
- Projets,
- Méthodes d'analyse par "couches" (génie logiciel).

Didactique de l'algorithmique et évaluation

- Savoir tester un programme,
  - → Avec l'ordinateur (of avant),
  - → A la main (of après),
- Le tableau comme "machine de Turing"/simulateur d'ordinateur (écrire/effacer/réécrire),
  - → Dessiner les cases mémoires,
  - → Les modifier en faisant une exécution pas à pas,
- Expliquer "ce qui se passe en machine",
  - → Désacraliser l'ordinateur,
  - → Autre chose qu'une boîte noire.

Le problème du choix d'un langage/logiciel

Différents paramètres

- Structuré et syntaxe rigoureuse ou plus permissif
- Typé ou non
- Passage par valeur ou par adresse
- Itératif ou récursif
- Fonctionnel ou impératif
- Orienté objet ou non

Une proposition : Python/Sage

Sage : aspect pédagogique

- Programmé en python, utilisation en Python.
- Le "notebook" : interface agréable/possibilités graphiques ,
- Nombreuses bibliothèques spécialisées
- Très complet : algèbre, analyse, calcul numérique, calcul symbolique, probabilités, statistiques, géométrie,
- Peut être utilisé jusque dans le supérieur

Une proposition : Python/Sage

Sage : aspect logistique

- Logiciel licence GNU
- L'aspect "serveur".
- Une communauté qui se développe.

→ Logiciel en anglais uniquement (mais ébauche d'une version francisée),

→ Peu de ressources en français pour les enseignants,

- Utilisé par certaines académies : Académie Aix-Marseille
  http://sage.irem.univ-mrs.fr/

Une proposition : Python/Sage

Python

- Python : un vrai langage, utilisé aussi bien par la Nasa, Google, en bioinformatique, ...
- Multi-plateforme, portable,
- Langage orienté objet : bonne vision intellectuelle,
- Fortement typé,
- Syntaxe simple et intuitive,
- Licence libre.

Conclusion

Introduction de l'algorithmique dans les programmes

- Mini-révolution dans l'enseignement des mathématiques
- Occasion de diversifier les compétences demandées aux élèves
- Pas seulement outil indispensable en mathématiques appliquées