Les savoirs mathématiques mobilisés par l'algorithmique

Emmanuel Beffara

Institut de Mathématiques de Marseille
Université de d'Aix-Marseille
IREM d'Aix-Marseille et IREM de Grenoble

XXVIe colloque CORFEM, Strasbourg, 11 juin 2019

Introduction

Résumé

L'activité algorithmique consiste en deux choses : d'une part à élaborer des algorithmes pour apporter des solutions calculatoires à un problème donné, d'autre part à étudier les algorithmes pour démontrer leur correction et évaluer leur efficacité. Chacun de ces aspects met en œuvre des savoirs et des compétences mathématiques : travail sur les énoncés logiques pour élaborer des spécifications, méthodes de démonstration (dont la récurrence sous de nombreuses formes) et étude des suites lors de l'analyse de complexité. Dans cet exposé, je mettrai en évidence ces différents aspects en les développant dans le cas d'un algorithme tiré du programme d'informatique.

Thème

Algorithmique, programmation, savoirs informatiques. Quels liens avec l'enseignement des mathématiques?

Comment comprendre cette question?

- Est-ce que les différents sujets d'informatique peuvent nous aider à enseigner le reste des mathématiques?
  — *dans le cadre des programmes de mathématiques*
- Est-ce que NSI peut nous apporter des choses pour enseigner les mathématiques?
  — *une sorte d'interdisciplinarité*
- Que peut-on apprendre comme mathématiques quand on enseigne l'informatique en tant que telle?
  — *ou la pensée mathématique au delà des contraintes disciplinaires*

Reformulons la question:

Logique, calcul, savoirs mathématiques. Quels liens avec l'enseignement de l'informatique?

L'informatique

On connaît bien la dichotomie usuelle:

Outil informatique, dans la longue lignée de l'instrumentation en mathématiques et au delà, qui mobilise des savoirs liés à l'outil et au domaine auquel il s'applique.

Science informatique, qui s'intéresse à la façon de penser propre à l'informatique, étudie les notions propres à ce domaine et leur mise en œuvre.

On se place ici résolument dans le second point.

- Quand on fait de la science informatique, on fait avant tout de la science. *esprit critique, précision, intuition, expérimentation, épreuve de réalité. . .*
- Le versant théorique est avant tout une pratique de nature mathématique, même si motivée par des préoccupations liées au monde réel.

L'algorithmique

Un algorithme est une méthode *effective* et *non ambigüe* permettant de résoudre un problème calculatoire donné.

Élaborer consiste à préciser une méthode jusqu'à la rendre effective et non ambigüe.

- choix des données et de leur représentation
- détermination des étapes de calcul et des situations intermédiaires

Analyser consiste à s'assurer de son effectivité pour résoudre le problème considéré.

- démontrer que le calcul se termine dans tous les cas à considérer
- démontrer que le résultat obtenu est le bon
- évaluer l'efficacité de la méthode employée

Chaque étape met en jeu une démarche et des savoirs mathématiques.

Illustration par l'exemple

On détaille tout cela avec des exemples d'algorithmes:

- recherche d'une « tranche minimale » dans un tableau,
- tri par insertion.

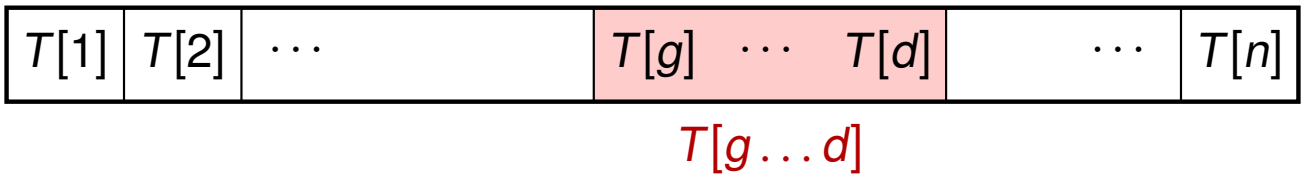
Tranche de somme minimale

La question

On se donne un tableau de valeurs (des entiers relatifs):

$$T = \Big[T[1], T[2], \ldots, T[n]\Big]$$

On demande de trouver une *tranche* de T, c'est-à-dire un suite d'éléments consécutifs du tableau, dont la somme est minimale:



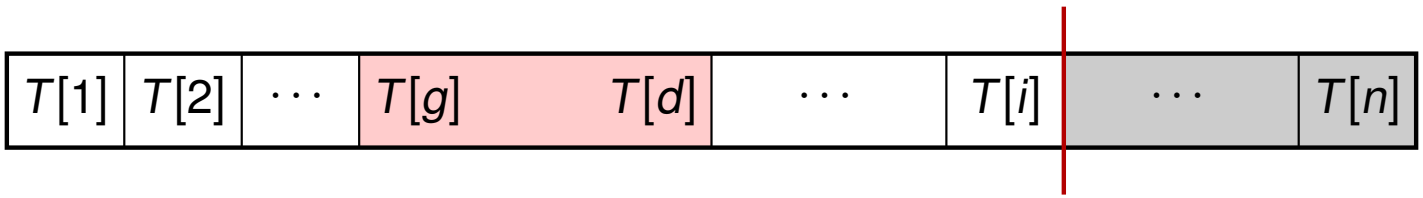
On demande de résoudre le problème en parcourant le tableau une seule fois.

Élaboration: états intermédiaires

On demande de résoudre le problème en parcourant le tableau une seule fois.

L'indication guide et contraint l'élaboration de l'algorithme:

- Après avoir traité les éléments jusqu'au rang i, on doit avoir extrait assez d'information pour répondre au sous-problème associé à la tranche $T[1 \ldots i]$.



On est donc amené à se poser la question suivante:

- Supposons que $T[g \ldots d]$ est une tranche minimale de $T[1 \ldots i]$. Comment en déduire une tranche minimale de $T[1 \ldots i+1]$?

Élaboration: étape fondamentale

On se place dans un état intermédiaire: le problème est traité pour la tranche $T[1 \dots i]$ pour un certain i avec $1 \leq i < n$, on supposons trouvés g et d tels que $T[g \dots d]$ est une tranche minimale de $T[1 \dots i]$.

Élaboration: étape fondamentale

On se place dans un état intermédiaire: le problème est traité pour la tranche $T[1 \dots i]$ pour un certain i avec $1 \leq i < n$, on supposons trouvés g et d tels que $T[g \dots d]$ est une tranche minimale de $T[1 \dots i]$.

Pour trouver une tranche minimale de $T[1 \dots i+1]$, on distingue deux cas:

- Soit $T[g \dots d]$ est aussi une tranche minimale pour $T[1 \dots i+1]$.
- Soit il y a dans $T[1 \dots i+1]$ une tranche de somme strictement inférieure.

Comment déterminer si on est dans le second cas?

Élaboration: étape fondamentale

On se place dans un état intermédiaire: le problème est traité pour la tranche $T[1 \dots i]$ pour un certain i avec $1 \leq i < n$, on supposons trouvés g et d tels que $T[g \dots d]$ est une tranche minimale de $T[1 \dots i]$.

Pour trouver une tranche minimale de $T[1 \dots i+1]$, on distingue deux cas:

- Soit $T[g \dots d]$ est aussi une tranche minimale pour $T[1 \dots i+1]$.
- Soit il y a dans $T[1 \dots i+1]$ une tranche de somme strictement inférieure.

Comment déterminer si on est dans le second cas?

- Si $T[g \dots d]$ est minimale dans $T[1 \dots i]$ mais pas dans $T[1 \dots i+1]$, c'est qu'une tranche minimale dans $T[1 \dots i+1]$ n'est pas une tranche de $T[1 \dots i]$.
- Donc elle contient $T[i+1]$ et s'écrit donc $T[k \dots i+1]$ pour un $k \leq i+1$.
- La borne k minimise $T[k] + \cdots + T[i+1]$, donc aussi $T[k] + \cdots + T[i]$.

Ainsi k peut être déterminé avant de connaître $T[i+1]$.

Algorithme

On maintient à chaque étape

- les bornes (g, d) d'une tranche minimale de $T[1 \ldots i]$ de somme *min*,
- la borne k d'une tranche finale minimale $T[k \ldots i]$ de somme *fin*.

Poser $g = d = k = 1$.
Poser $min = fin = T[1]$.
Pour i **de** 2 **à** n, **faire**:
    si $fin + T[i] < T[i]$
        alors $fin := fin + T[i]$
        sinon $fin := T[i]$; $k := i$
    si $fin < min$ **alors** $min := fin$; $g := k$; $d := i$

Analyse: correction par invariant

Poser $g = d = k = 1$.
Poser $min = fin = T[1]$.
Pour i **de** 2 **à** n, **faire**:
    { *invariant satisfait pour* $i - 1$ }
    si $fin + T[i] < T[i]$
        alors $fin := fin + T[i]$
        sinon $fin := T[i]$; $k := i$
    si $fin < min$ **alors** $min := fin$; $g := k$; $d := i$
    { *invariant satisfait pour* i }

Analyse: correction par invariant

Poser $g = d = k = 1$.
Poser $min = fin = T[1]$.
Pour i **de** 2 **à** n, **faire**:
    { *invariant satisfait pour* $i - 1$ }
    si $fin + T[i] < T[i]$
        alors $fin := fin + T[i]$
        sinon $fin := T[i]$; $k := i$
    si $fin < min$ **alors** $min := fin$; $g := k$; $d := i$
    { *invariant satisfait pour* i }

L'invariant $P(i)$ décrit l'état intermédiaire en fonction du rang i:

- $T[g \ldots d]$ est une tranche minimale dans $T[1 \ldots i]$,
- $T[k \ldots i]$ est minimale parmi les tranches contenant $T[i]$.

Analyse: correction par invariant

Poser $g = d = k = 1$.
Poser $min = fin = T[1]$.
Pour i **de** 2 **à** n, **faire**:
    { *invariant satisfait pour* $i - 1$ }
    si $fin + T[i] < T[i]$
        alors $fin := fin + T[i]$
        sinon $fin := T[i]$; $k := i$
    si $fin < min$ **alors** $min := fin$; $g := k$; $d := i$
    { *invariant satisfait pour* i }

L'invariant $P(i)$ décrit l'état intermédiaire en fonction du rang i:

- $T[g \ldots d]$ est une tranche minimale dans $T[1 \ldots i]$,
- $T[k \ldots i]$ est minimale parmi les tranches contenant $T[i]$.

On le démontre par récurrence:

- l'élaboration de donne $P(i-1) \rightarrow P(i)$ pour chaque i tel que $2 \leq i \leq n$,
- on vérifie explicitement $P(1)$ après l'initialisation des variables.

Par récurrence, on en déduit que $P(n)$ est satisfait après la dernière itération $j = n$, donc $T[g \ldots d]$ est une tranche minimale en fin de calcul.

Analyse: efficacité

Poser $g = d = k = 1$.
Poser $min = fin = T[1]$.
Pour i **de** 2 **à** n, **faire**:
    si $fin + T[i] < T[i]$
        alors $fin := fin + T[i]$
        sinon $fin := T[i]$; $k := i$
    si $fin < min$ **alors** $min := fin$; $g := k$; $d := i$

Analyse: efficacité

Poser $g = d = k = 1$.
Poser $min = fin = T[1]$.
Pour i **de** 2 **à** n, **faire**:
    si $fin + T[i] < T[i]$
        alors $fin := fin + T[i]$
        sinon $fin := T[i]$; $k := i$
    si $fin < min$ **alors** $min := fin$; $g := k$; $d := i$

On peut compter les opérations:

- par itération: une addition, une comparaison, 0 à 5 affectations;
- en tout: $4n - 4$ additions et comparaisons, 5 à $20n$ affectations $\longrightarrow O(n)$.

Comparaison avec un algorithme naïf:

Poser $g = d = 1$ et $min = T[1]$.
Pour i **de** 1 **à** n,
    pour j **de** i **à** n,
        si $T[i] + \cdots + T[j] < min$ **alors** $min := T[i] + \cdots + T[j]$; $g := i$; $d := j$

- $n(n+1)/2$ itérations avec $j - i$ additions, une comparaison et 3 affectations;
- nombre total d'opérations: un polynôme de degré 3 en $n \longrightarrow O(n^3)$.

Tri par insertion

La question

Problème à résoudre:

- Trier les éléments d'un tableau.

Contraintes sur la méthode:

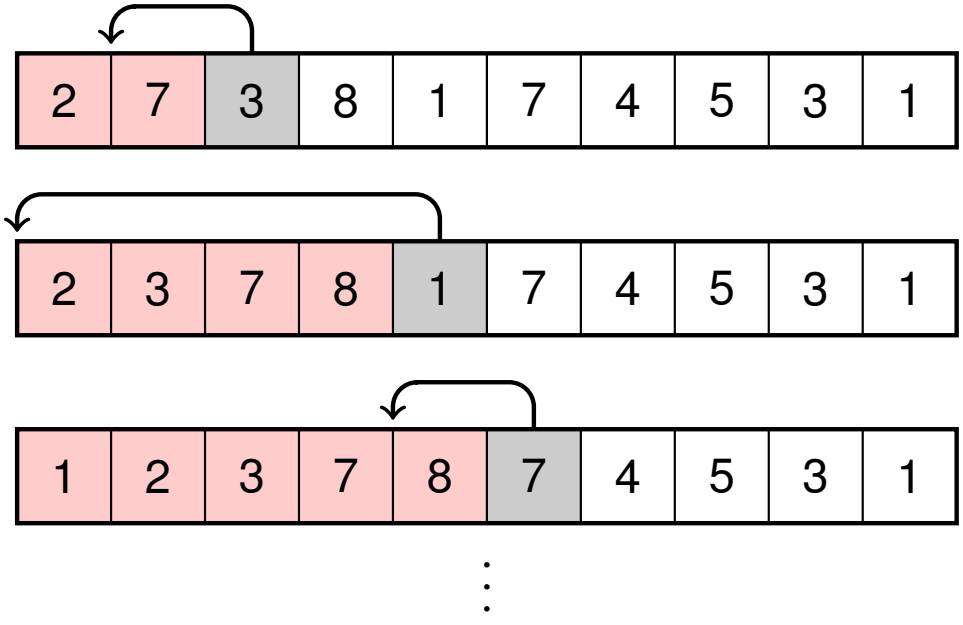
- en insérant les éléments l'un après l'autre dans un tableau trié,
- le tout sans utiliser de tableau intermédiaire.

Comment préciser ce que l'on fait?

Élaboration, structure de l'algorithme

Trier les éléments d'un tableau, en insérant les éléments l'un après l'autre dans un tableau trié, sans utiliser de tableau intermédiaire.

Pour rester dans un seul tableau, on utilise les premières cases pour la partie triée:



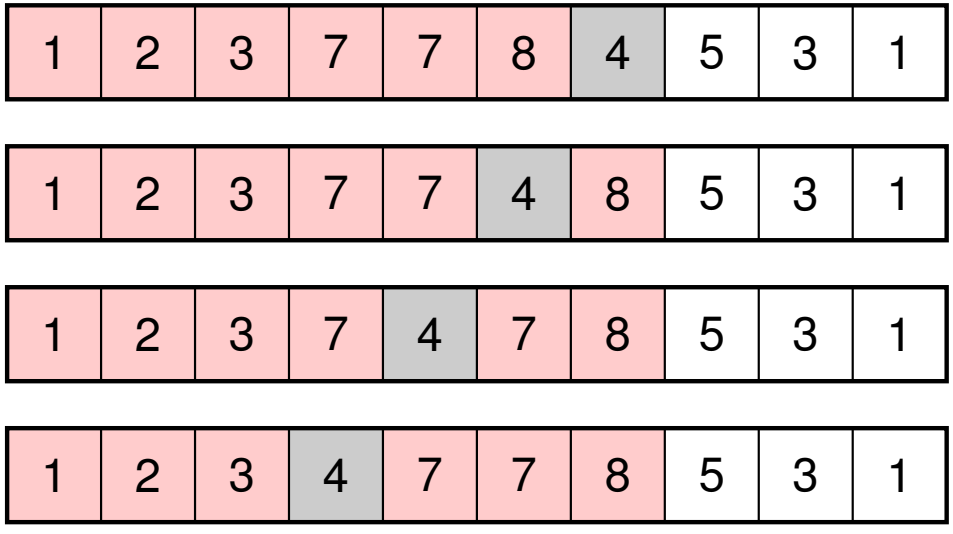
Situation à l'étape i:

- T est une permutation du tableau initial,
- la tranche $T[1 \ldots i]$ est triée.

Élaboration, insertion d'un élément

À chaque étape i, on veut insérer l'élément $T[i+1]$ à sa place k dans $T[1 \ldots i]$.

- Il faut décaler la tranche $T[k \ldots i]$ d'une case vers la droite.
- Tout se passe dans un seul tableau, on procède par échanges successifs.



Notons j la position de l'élément à placer dans la tranche $T[1 \ldots i+1]$.

Situation à l'étape j, avec i fixé:

- T est une permutation du tableau initial,
- les tranches $T[1 \ldots j-1]$ et $T[j \ldots i+1]$ sont triées.

Algorithme et invariants

Pour i **de** 1 **à** $n-1$,
    { $P(i)$ *est satisfait* }
    poser $j = i + 1$
    tant que $j > 1$ et $T[j-1] > T[j]$,
        { $Q(i,j)$ *est satisfait* }
        échanger $T[j]$ et $T[j-1]$,
        $j := j - 1$

Les invariants sont:

- T est toujours une permutation du tableau initial
- $P(i)$: la tranche $T[1 \ldots i]$ est triée
- $Q(i,j)$: les tranches $T[1 \ldots j-1]$ et $T[j \ldots i+1]$ sont triées

Algorithme et invariants

Pour i **de** 1 **à** $n-1$,
    { $P(i)$ *est satisfait* }
    poser $j = i + 1$
    tant que $j > 1$ et $T[j-1] > T[j]$,
        { $Q(i,j)$ *est satisfait* }
        échanger $T[j]$ et $T[j-1]$,
        $j := j - 1$

Les invariants sont:

- T est toujours une permutation du tableau initial
- $P(i)$: la tranche $T[1 \ldots i]$ est triée
- $Q(i,j)$: les tranches $T[1 \ldots j-1]$ et $T[j \ldots i+1]$ sont triées

Étapes de démonstration:

- poser $j = i + 1$ garantit $Q(i,j)$ en supposant $P(i)$ et $i < n$,
- l'échange pour $T[j]$ garantit $Q(i,j-1)$ en supposant $Q(i,j)$,
- la condition d'arrêt permet de montrer $Q(i,j) \rightarrow P(i+1)$.

Conclusion

Les compétences en jeu

Évaluer attribuer mentalement une valeur à un programme donné.

Anticiper se mettre en posture de programmeur pour décrire l'enchaînement des opérations, avant le début de l'exécution.

Décomposer transformer un problème complexe en un ensemble de problèmes plus simples équivalent au problème initial.

Généraliser inférer un problème général à partir d'une instance, repérer dans un problème particulier la répétition de traitements ou de données suivant un même schéma.

Abstraire "faire abstraction" des informations non pertinentes et créer des solutions où la manière de résoudre un problème peut être "abstraite" à l'aide d'une interface pertinente.

Bilan

Quels savoirs et techniques mathématiques?

- formuler précisément une situation
- travailler les énoncés
  - généraliser (pour intégrer un paramètre supplémentaire)
  - décomposer (pour comparer le cas i et le cas $i+1$)
  - reformuler...
- démontrer des énoncés
  - une récurrence « en action »
  - une erreur dans un algorithme est une démonstration invalide
  - les notions manipulées ici sont essentiellement combinatoires
- construire et analyser des suites