

Algorithmique et Complexité

Nicole Bidoit

Université Paris XI, Orsay

Année Universitaire 2008–2009

Algorithmique et Complexité

1. Introduction

Nicole Bidoit
Université Paris XI, Orsay

Année Universitaire 2007–2008

Introduction

● Algorithme ?

DESCRIPTION D'UNE SUITE/ENCHAÎNEMENT D'ÉTAPES DE CALCUL/ACTION POUR RÉSOUDRE UN PROBLÈME DONNÉ

↪ problème : général, spécifié par une fonction (entrées → sorties)

ex : multiplier, PGCD, primalité, ..., tri, soudure par robot, ...

↪ résoudre : solution constructive, correction, efficacité

↪ calcul/action : "plus ou moins" élémentaire

ex : addition, comparaison, ..., algorithme connu, ...

↪ suite/enchaînement : réalisation mécanique, suite finie,

ex : séquençement, action conditionnelle, itération,...

↪ description : précise, concise, abstraite, ... réutilisable

ex : multiplication "comme à l'école", multiplication à la russe, algorithme d'Euclide, algorithme de tri rapide, ...

Introduction : Correction

Problème **P** – Algorithme **A**

Correction : Pour toute instance légale **I** du problème **P**, l'algorithme **A** retourne la solution **P(I)**
ie on a toujours **A(I)=P(I)**.

Ex de la multiplication: l'algorithme "marche" même pour des entiers grands ...

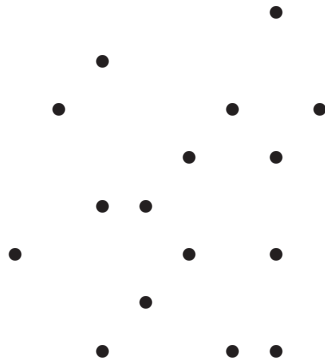
Ex du tri : l'algorithme "marche" même si l'entrée est déjà triée ou contient des répétitions d'éléments.

Montrer la correction d'un algorithme = difficile !

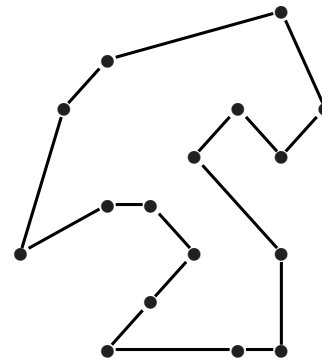
Trouver un algorithme correct (et efficace) = difficile !

Problème 1 : Bras-robot effectuant des soudures pour fabriquer des circuits intégrés. Pour réaliser un circuit, il faut donner au robot les points de contacts dans un ordre qui servira au bras à visiter ces points pour effectuer les soudures.

Problème (d'optimisation) = l'ordre des points doit minimiser la distance parcourue par le bras-robot.



Voici une instance du problème



et une solution (-: !

A vous de jouer pour trouver un algorithme résolvant le problème général

Introduction : Correction

Une première tentative : **voisin le plus proche**

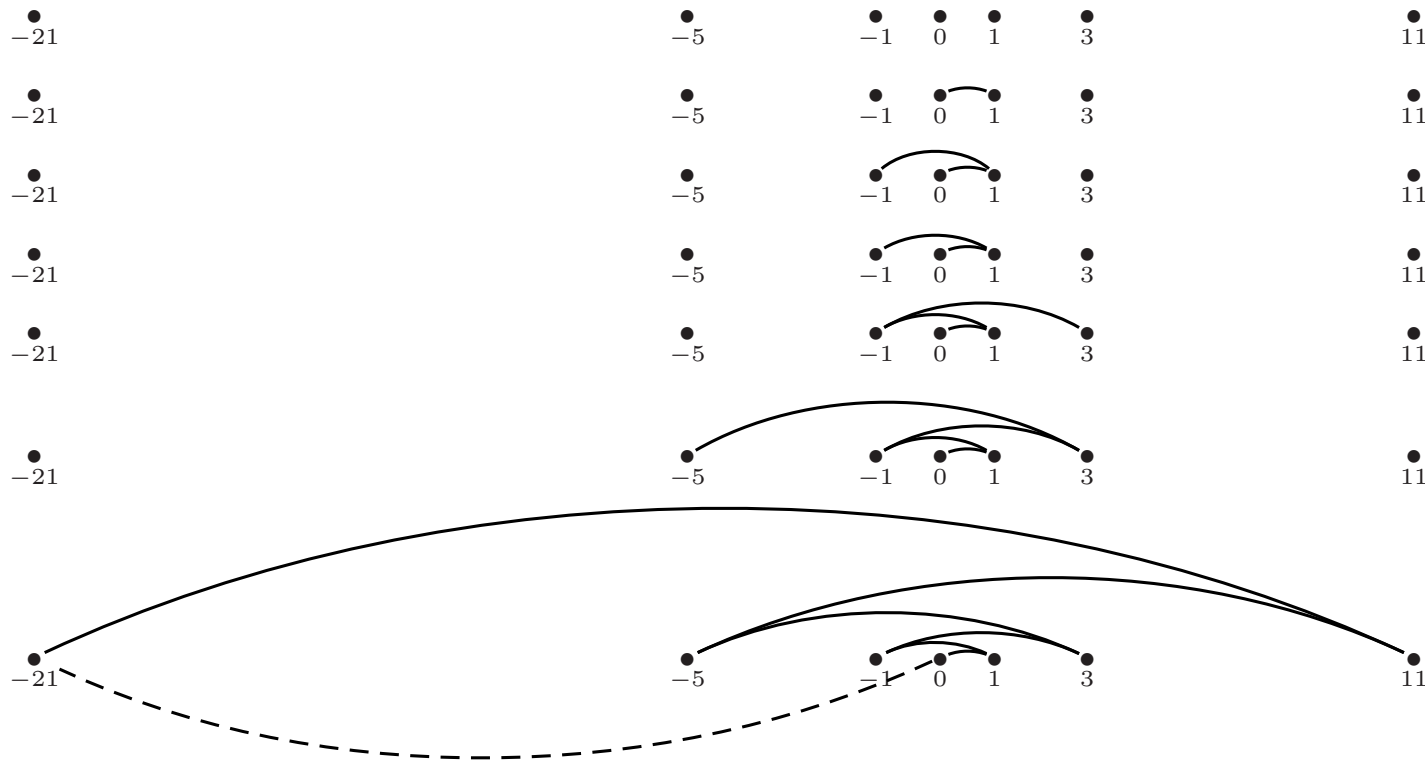
(Nearest Neighbour Tour)

Choisir, aléatoirement[†], un point p_0 , rejoindre son plus proche voisin p_1 ;

répéter ceci à partir de p_1 (éliminer au fur et à mesure les points déjà visités) ... jusqu'à la "fin";

rejoindre le point p_0 .

Cet algorithme n'est pas correct. Voici une instance témoin.



Exercice : Écrire cet algorithme.

[†] au hasard.

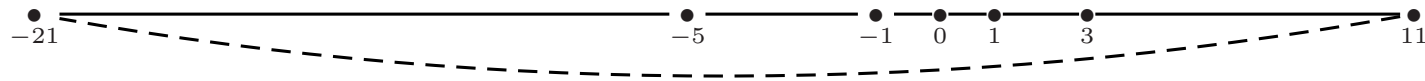
Introduction : Correction

Une deuxième tentative : **couple de points les plus proches**

(Closest Pair Tour)

Autre idée : connecter les couples de points les plus proches sans créer de cycle ou de branchement.

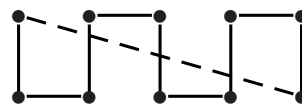
Cet algorithme traite l'exemple précédent avec succès:



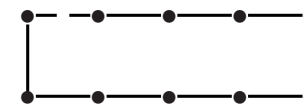
mais voici une instance qui montre que **ce nouvel algorithme n'est pas correct**.



instance



résultat algorithme



solution

Exercice : Écrire cet algorithme.

Introduction : Correction

Un algorithme correct : **recherche exhaustive**

(Exhaustive Search)

Considérer tous les ordonnancements possibles;

Sélectionner l'ordonnement minimisant le déplacement.

Exercice : Écrire cet algorithme.

La recherche exhaustive est trop "lente" !

Si le circuit nécessite n points de soudure, le nombre d'ordonnements à considérer est

$$n! = 1 \times 2 \times \dots \times (n - 1) \times n.$$

À partir de 10-20 points, la recherche exhaustive est beaucoup trop inefficace.

Le problème difficile que nous venons d'évoquer est plus connu sous le nom de

problème du voyageur de commerce

(Traveling Salesman)

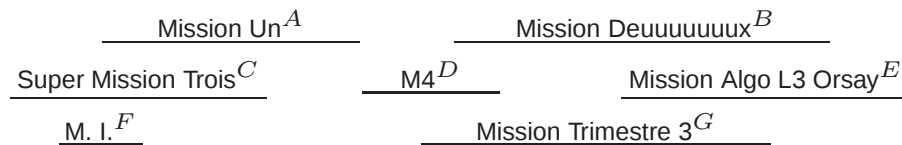
Il existe pleins de variantes de ce problème liées à des applications industrielles.

Introduction : Correction

Avoir un énoncé clair/formalisé du problème est une première étape importante pour rechercher un algorithme et prouver sa correction.

Ex: il est possible de trouver le plus court chemin mais pas le meilleur chemin!

Problème 2-a : Un intérimaire souhaite organiser son emploi du temps de sorte à effectuer le plus grand nombre possible de missions, sans que deux missions se recouvrent dans le temps.



Voici une instance du problème

Énoncé formalisé du problème 2-a :

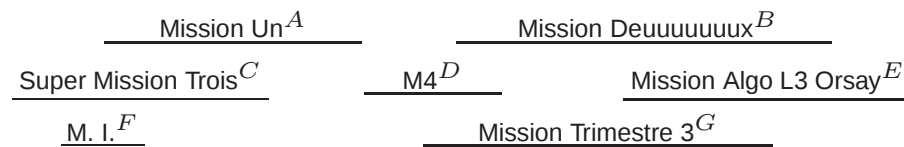
Entrée = un ensemble I de n intervalles

Sortie = un plus grand[†] sous ensemble d'intervalles de I deux à deux disjoints ?

[†] plus grand au sens de la cardinalité des ensembles.

Introduction : Correction

Problème 2-b : Un intérimaire souhaite sélectionner les missions qui maximisent son temps total de travail, sans que deux missions se recouvrent dans le temps.



Voici une instance du problème

Énoncé formalisé du problème 2-b :

Entrée = un ensemble I de n intervalles

Sortie = un plus grand[†] sous ensemble d'intervalles de I deux à deux disjoints ?

[†] I_1 plus grand que I_2 si la somme des tailles des intervalles dans I_1 est supérieure à celle faite pour les intervalles de I_2

Introduction : Correction

Donner une **description précise de l'algorithme** est une deuxième étape importante pour prouver sa correction.

Comment présenter un algorithme ?

en français : étape nécessaire mais une étape seulement ; insuffisant

notation dédiée : pseudo-Pascal, jeux d'instructions limitées avec parfois quelques "phrases".

notation présentée en cours et en TD au fur et à mesure!

Algo voisin proche (problème 1):

Algo NNT

entrée : ensemble P de n points et leurs distances 2 à 2

sortie : la suite $(p_i)_{i=0..(n-1)}$

choisir p dans P ;

initialisation %%%%

$i \leftarrow 0$; $p_0 \leftarrow p$; $P \leftarrow P - \{p_i\}$

while P non vide **do**

itération %%%%

soit p dans P tel que p est le plus proche possible de p_i

$i \leftarrow i+1$; $p_i \leftarrow p$; $P \leftarrow P - \{p_i\}$

quelle structure de données pour l'ensemble P ? pour la suite $(p_i)_{i=0..(n-1)}$?

comment est recherché p dans l'itération ?

Introduction : Correction

Prouver qu'un algorithme **A** est incorrect / correct pour le problème **P**

- Examiner des instances (exemples) "petits"
- Examiner des instances qui activent "fortement" les tests
- Examiner des instances eXtrêmes en taille, et aussi pour leurs caractéristiques ...

Revisiter l'exemple 1.

Problème 2-a :

- la stratégie "choisir la première mission qui se présente" est incorrecte
- la stratégie "choisir la mission la plus courte" est incorrecte
- "choisir la mission dont la fin est le plus tôt" est correcte

Ne pas trouver de contre-exemple pour un algorithme (instance montrant que l'algorithme est incorrect)
ne doit pas faire penser que l'algorithme est évidemment correct !

Induction et Récursion utilisent les mêmes idées:

- (1) traitement du cas de base
- (2) hypothèse ou propriété générale (d'induction)
- (3) traitement du cas général

L'induction mathématique est souvent utilisée pour montrer la correction d'algorithmes récursifs.

Introduction : Efficacité, Complexité

Pour un problème **P**, on dispose souvent de plusieurs algorithmes corrects **A**, **B** ...

Lequel est le plus efficace ?

Problème 3 : Multiplication de 2 entiers

- Algorithmes : comme à l'école, à la russe, méthode pour grands entiers, ...

Comme à l'école :

$$\begin{array}{r} 45 \\ \times 19 \\ \hline 405 \\ 45 \\ \hline 855 \end{array}$$

↪ tables de multiplication

↪ addition

Méthode russe[†] :

$$\begin{array}{r} 45 \quad 19 \quad * \\ 22 \quad 38 \\ 11 \quad 76 \quad * \\ 5 \quad 152 \quad * \\ 2 \quad 304 \\ 1 \quad 608 \quad * \\ \hline 19 + 76 + 152 + 608 = 855 \end{array}$$

↪ multiplication par 2

↪ division par 2

↪ addition

Exercice : Écrire les deux algorithmes

[†] méthode utilisée pour les circuits intégrés.

Comparer des algorithmes

Déterminer si l'algorithme est le meilleur possible

- Approche empirique : écrire un programme pour implémenter **A** et, pour un jeu de tests mesurer le temps d'exécution et/ou espace mémoire
↪ programmeur ↪ langage ↪ machine ↪ jeu de tests
- Approche a priori (indépendante de toute implémentation) :
 - ↪ machine / ordinateur "idéal" (RAM : Random Access Machine)
 - ↪ chaque opération élémentaire coûte 1 opération élémentaire ?
 - ↪ chaque accès mémoire coûte 1
 - ↪ la taille d'une instance I est la taille du codage binaire de I
NON, c'est inutilement compliqué : nombre "logique" d'éléments, valeur (cas numérique)
Tri : taille instance = nbre d'éléments à trier
Décomposition en facteurs premiers de n :
taille instance = valeur n ou taille de la représentation binaire de n
- Approche hybride
 - déterminer la complexité de l'algorithme (forme générale de la fonction)
 - préciser certains paramètres expérimentalement.

1. Introduction

2. Analyse et Complexité des algorithmes

- ↪ complexité en temps, en espace
- ↪ meilleur cas, pire cas, en moyenne

3. Structures de données avancées

- ↪ rappels , tableaux dynamiques, hachage,
- ↪ arbres binaires de recherche : AVL, 2-3-4, et variantes

4. Stratégie I : diviser pour régner

- ↪ multiplications de matrices
- ↪ Tri : tri-rapide, le tri-fusion, ...

5. Stratégie II : Programmation dynamique

- ↪ multiplication d'une liste de matrices, distance d'édition

6. Stratégie III : Backtracking

- ↪ Permutation d'une liste, les 8 reines, sudoku.

Objectif du cours

est Faire de l'algorithmique ... pas de la programmation

Découvrir des méthodes, des stratégies générales

Apprendre à se poser des questions

Aborder certains algorithmes classiques autrement

Découvrir d'autres algorithmes, leurs applications

n'est pas Apprendre (une fois de plus) le tri rapide, tri bulles, ... recherche dans une liste

Équipe pédagogique et contacts

Nicole Bidoit	nicole.bidoit@lri.fr	http://www.lri.fr/~bidoit/	LRI, Bât 490, 2ème étage
Najla Chamsed	chamsed@lsv.ens-cachan.fr		
Romaric Gaudel	romaric.gaudel@lri.fr		
Marc Kaplan	kaplan@lri.fr		
Nguyen Kim Thang	thang@lix.polytechnique.fr		

Séances de cours (1h45/semaine)

Poser des questions pendant le cours, après (15 mn à votre disposition)

Signaler les erreurs, faire les exercices suggérés en cours

Compléter par la consultation d'ouvrages (voir suggestions)

Attention : le cours ne sera pas refait en TD

Séances de travaux dirigés (3h/semaine)

Préparer les exercices à l'avance (Minimum = lire les sujets)

Rédiger les solutions proprement

Chercher avec votre crayon, votre gomme, ..., dessiner, gribouiller, ...

Soyez actif !!!!

Contrôle des connaissances

Contrôle continu : partiel et **Exercices d'application du cours**

Partiel et examen : document non autorisé, rappels de cours nécessaires inclus.

Les transparents sont distribués - ils sont aussi disponible sur ma page Web.

Les sources sont nombreuses, très nombreuses. Voici quelques classiques :

- **Introduction à l'algorithmique, Th. Cormen, Ch. Leiserson, R. Rivest, chez Dunod.**
- Types de données et algorithmes, C. Froideveaux, M-C. Gaudel, M. Soria, chez Ediscience.
- Algorithmes en langage C : cours et exercices, R. Sedgewick, chez Dunod.
- Algorithmique : conception et analyse, G. Brassad, P. Bratley, chez Masson.
- **The Art of Computer Programming**, D. Knuth, chez Addison-Wesley.
(-: Il y a 3 volumes publiés. Les volumes 4 et 5 sont en préparation.

Réfléchir au Problème du Sac à Dos

Le problème du **sac à dos** s'énonce comme ceci: étant donné un ensemble fini d'entiers $E = \{e_1, \dots, e_n\}$ et un nombre entier cible c , trouvez un sous-ensemble de E dont la somme des entiers soit égale à c . Voir e_i comme la taille d'un objet i à placer dans un contenant (sac à dos) dont la capacité est c .

Par exemple, pour $E = \{1, 2, 5, 9, 10\}$, un tel sous-ensemble existe pour $c = 22$ mais pas pour $c = 23$.

Trouvez des contre-exemples pour chacun des algorithmes suivants proposés pour le problème du sac à dos: donner une instance (E, c) du problème tel que la solution de cette instance ne corresponde pas au résultat de l'algorithme pour cette instance.

1. **premier servi (first fit)** considérer les éléments de E dans l'ordre droit à gauche et les mettre dans le sac à dos si ils y tiennent,
2. **mieux servi (best fit)** considérer les éléments de E dans l'ordre du plus petit au plus grand et les mettre dans le sac à dos si ils y tiennent,
3. **variante** considérer les éléments de E dans l'ordre du plus grand au plus petit et les mettre dans le sac à dos si ils y tiennent.