I21: Introduction à l'algorithmique

Cours 1: Conception des algorithmes

Nicolas Méloni

Licence 1: 2ème semestre

(2017/2018)

À propos

Objectifs

- Connaitre et comprendre les concepts essentiels à l'analyse et la conception d'algorithmes.
- Connaitre et savoir réutiliser quelques algorithmes et structures fondamentales
- Appliquer les méthodes générales de conceptions des algorithmes à des problèmes nouveaux et étudier la validité et la compléxité de la solution.

À propos

Déroulement

- 10 cours magistraux
- 8 séances de TD
- 8 séances de TP

Évaluations

- 1 contrôle terminal
- 2-4 contrôles continus
- 8 TP notés (moodle)
- Note finale $= \max(0.7\text{CT}, 0.5\text{CT}+0.2\text{CC})+0.3\text{TP}$

Algorithmes

Définition

Procédure permettant de résoudre un problème donné, en un nombre fini d'étapes, par application d'une série de règles prédéfinies.

- Un algorithme est sensé résoudre un problème **général**.
- Le problème doit être défini en spécifiant l'ensemble des instances sur lesquelles ils portent.

Problème / instance

Un problème est défini par son entrée et sa sortie

Problème : Tri

Entrée : Un ensemble de $n \geq 1$ données $a_1, a_2, \ldots, a_n$.
Sortie : Une permutation des données telle que
$a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Problème / instance

Certaines spécifications peuvent être trop vagues ou trop générales, auquel cas il devient difficile ou impossible d'écrire un algorithme de résolution.

Problème : Meilleur chemin

Entrée : Une carte routière, deux points A et B sur la carte.
Sortie : Le meilleur chemin entre A et B.

Il est important de bien définir les objets et leur relations.

Problème / instance

Une instance du problème est la donnée d'une entrée spécifique.

Instances du problème de tri

- $235, 564, 12, 325, 95, 631, 223$
- `Nicolas, Jean-Pierre, Pascal, Thierry, Valerie, Joseph`

Un algorithme de résolution d'un problème doit fonctionner pour toute instance du problème.

Langages de descriptions

- Langage naturel
- Pseudocode
- Langage de programmation

Chaque façon représente un certain compromi entre facilité d'expression et précision.

Description d'un algorithme

Problème : Somme des entiers de 1 à n

Entrée : Un entier $n \geq 0$
Sortie : $S = \sum_{i=1}^{n} i = 1 + 2 + 3 \cdots + (n-1) + n$

Langage naturel

On énumère les nombres entre 1 et n que l'on additionne au fur et à mesure.

Pseudocode

```
ALGORITHME Somme(n): entier
DONNEES:
  n: entier
VARIABLES:
  i: entier
  s: entier
DEBUT
  i ← 1
  s ← 0
  TQ i ≤ n FAIRE
    s ← s + i
    i ← i + 1
  FTQ
  RENVOYER s
FIN
```

Description d'un algorithme

Langage de programmation (Python)

```python
def somme(n):
  s = 0
  for i in range(n+1):
    s = s + i
  return s
```

Description d'un algorithme

- Quelle que soit la méthode choisie, c'est la clarté de la description qui doit primer.
- Tout algorithme repose sur une **idée clé**, la description de l'algorithme doit permettre de la faire apparaître clairement.

Description d'un algorithme

Pseudocode

- C'est un langage "universel".
- Reprend la structure d'un programme classique sans référence à un langage particulier.
- Permet de décrire précisément un algorithme en faisant abstraction de certaines difficultés techniques.
- Il n'y a pas de convention générale pour son écriture.

Description d'un algorithme

Pseudocode (dans ce cours)

- Décomposé en trois blocs :
  1. Données : la ou les entrées à traiter par l'algorithme
  2. Variables : l'ensemble des différentes variables que manipule l'algorithme
  3. Corps de l'algortihme : la séquence des instructions à éxécuter, encadré par les mots clés **DEBUT et FIN.**
- Mots clés : **DEBUT, FIN, SI, ALORS, FSI, SINON, ET,OU,TQ, FTQ, FAIRE, RENVOYER**

Exemple 1 : Somme d'une liste d'éléments

Exemple 1 : Somme d'une liste d'éléments

```
ALGORITHME ListeSomme(L): liste d'entiers
DONNEES:
  L: liste d'entiers
VARIABLES:
  i: entier
  s: entier
DEBUT
  i ← 1
  s ← 0
  TQ i ≤ #L FAIRE
    s ← s + L[i]
    i ← i + 1
  FTQ
  RENVOYER s
FIN
```

Exemple 2 : primalité d'un nombre

Exemple 2 : primalité d'un nombre

```
ALGORITHME EstPremier(n): booleen
DONNEES:
   n: entier
VARIABLES:
   d: entier
DEBUT
   d ← 2
   TQ d < n FAIRE
      SI n mod d = 0 ALORS
         RENVOYER FAUX
      FSI
      d gets d+1
   FTQ
   RENVOYER VRAI
FIN
```

Justesse et terminaison

Pour être valide un algorithme doit satisfaire 2 conditions :

- être correct (ou juste)
- se terminer

Vérifier la justesse ou la terminaison d'un algorithme se fait à l'aide d'une preuve.

Justesse et terminaison

Preuve d'arrêt

Il s'agit de vérifier que quelle que soit l'instance du problème qui est traitée, l'algorithme fini toujours par s'arrêter.

Preuve de justesse

Il s'agit de vérifier que quelle que soit l'instance du problème qui est traitée, l'algorithme retournera toujours le résultat attendu.

Exemple de preuves

```
ALGORITHME Somme(n): entier
DONNEES:
   n: entier
VARIABLES:
   i: entier
   s: entier
DEBUT
   i ← 1
   s ← 0
   TQ i ≤ n FAIRE
      s ← s + i
      i ← i + 1
   FTQ
   RENVOYER s
FIN
```

Preuve d'arrêt

Il faut montrer que chaque boucle de l'algorithme voit sa condition de continuation invalidée au bout d'un certain nombre d'étapes.
Ici il suffit de montrer que la variable i finie toujours par prendre la valeur $n+1$.

Exemple de preuves

```
ALGORITHME Somme(n): entier
DONNEES:
   n: entier
VARIABLES:
   i: entier
   s: entier
DEBUT
   i ← 1
   s ← 0
   TQ i ≤ n FAIRE
      s ← s + i
      i ← i + 1
   FTQ
   RENVOYER s
FIN
```

Preuve de validité

Il faut montrer que les valeurs des différentes variables sont bien en rapport avec le résultat attendu.
On cherche une proriété dépendante des variables qui reste vraie tout au long de l'algorithme. Ici par exemple, en début de boucle, on a toujours $s = \sum_{k=1}^{i-1} k = 1+2+\cdots+(i-1)$.

Preuve par récurrence

La preuve par récurrence est un des outils les plus utiles pour démontrer la validité d'un algorithme. Elle permet de montrer qu'une propriété $\mathcal{P}(n)$ est vraie pour tout n plus grand qu'une certaine constante (en général 0). Pour cela il faut montrer 2 points :

- la propriété $\mathcal{P}(0)$ est vraie (initialisation) ;
- si la propriété $\mathcal{P}(n)$ est vraie alors la propriété $\mathcal{P}(n+1)$ l'est aussi (hérédité).

Preuve par récurrence

- Montrer que $\forall n \geq 0, \sum_{i=0}^{n} i = \frac{n(n+1)}{2}$.

Ici on a $\mathcal{P}(n)$: $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$.

Initialisation

$\mathcal{P}(0)$ est vraie car $\sum_{i=0}^{0} i = 0 = \frac{0(0+1)}{2}$

Hérédité

Soit $n \geq 0$. Supposons que $\mathcal{P}(n)$ est vraie, montrons que $\sum_{i=0}^{n+1} i = \frac{(n+1)(n+2)}{2}$.

Preuve par récurrence

On a

$$
\begin{aligned}
\sum_{i=0}^{n+1} i &= \sum_{i=0}^{n} i + (n+1) \\
&= \frac{n(n+1)}{2} + (n+1) \text{ (par hypothèse de récurrence)} \\
&= \frac{n(n+1) + 2n + 2}{2} \\
&= \frac{(n+1)(n+2)}{2}
\end{aligned}
$$

L'initialisation ne doit pas être oubliée.

- Une propriété peut être héréditaire dans jamais n'être vraie pour aucun $n \in \mathbb{N}$.
- Ex : si 10^n est divisible par 3 alors 10^{n+1} est divisible par 3 mais quel que soit n 10^n n'est jamais divisible par 3.

Invalider un algorithme

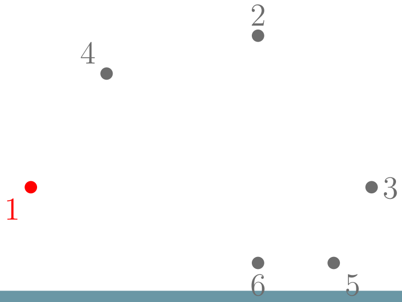
- Faire une preuve de validité peut être difficile.
- Avant de se lancer dans une preuve on vérifie que l'algorithme n'est pas trivialement faux.
- Montrer qu'un algorithme n'est pas valide est en général plus simple : il suffit d'exhiber un contre exemple.

Invalider un algorithme

Problème : Ramassage de plots

Entrée : n plots sur un terrain numérotés de 1 à n.
Sortie : Le plus court chemin pour ramasser tous les plots et revenir au point de départ en partant du plot 1.

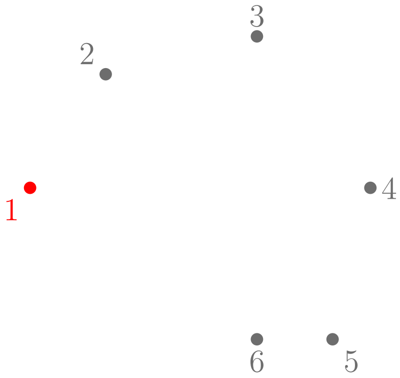
Exemple d'instance



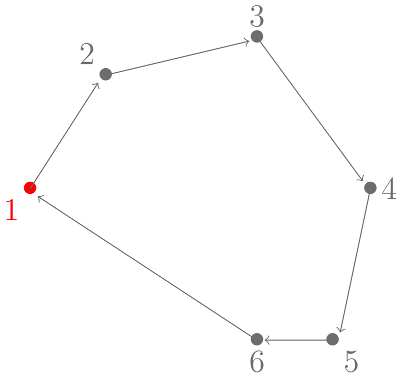
Invalider un algorithme

```
ALGORITHME RamassePlots()
DEBUT
    Ramasser le plot 1
  TQ (il reste des plots) FAIRE
     ramasser le plot le plus proche
  FTQ
     revenir au debut
FIN
```

Instance favorable

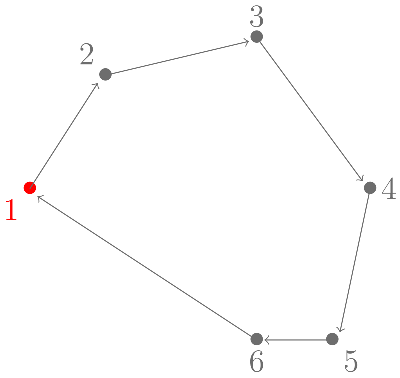


Instance favorable

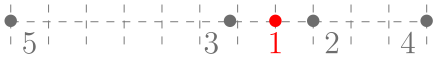


Invalider un algorithme

Instance favorable

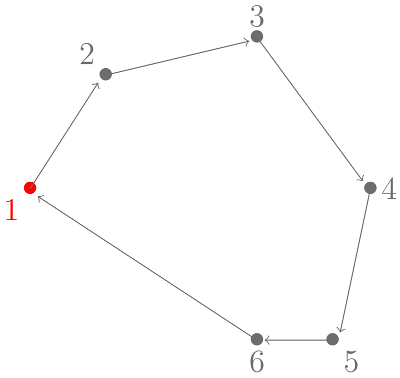


Instance défavorable

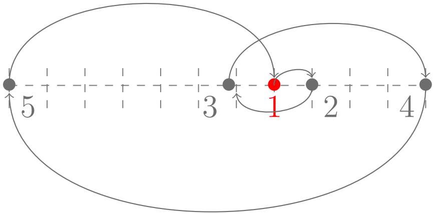


Invalider un algorithme

Instance favorable



Instance défavorable



Invalider un algorithme

```
ALGORITHME RamassePlotsCorrect()
DEBUT
   Calculer la longueur de tous les parcours possibles
   Choisir le parcours le plus court
FIN
```

Invalider un algorithme

```
ALGORITHME RamassePlotsCorrect()
DEBUT
   Calculer la longueur de tous les parcours possibles
   Choisir le parcours le plus court
FIN
```

Il y a $(n-1)!$ parcours possible !

Modélisation d'un problème

- C'est le processus qui consiste à reformuler un problème concret en termes d'objets abstraits pour lesquels on sait résoudre le problème.
- La modélisation détermine en particulier les opérations que l'on s'autorise ainsi que leur coût.
- Du bon choix de la modélisation dépendra en grande partie l'efficacité des solutions que l'on pourra concevoir.