

I21: Introduction à l'algorithmique

Cours 1: Conception des algorithmes

Nicolas Méloni

Licence 1 (2017-2020)

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 complexité de la solution.

Déroulement

- ❖ 14 cours magistraux
- ❖ 16 séances de TD
- ❖ 10 séances de TP

Évaluations

- ❖ 1 contrôle terminal
- ❖ 2-4 contrôles continus
- ❖ 9 TP notés (moodle)
- ❖ 1 examen de TP
- ❖ Note finale = $\max(0.7CT, 0.5CT+0.2CC)+0.3TP$

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.

Un problème est défini par son entrée et sa sortie

Problème : Tri

Entrée : Un ensemble de $n \geq 1$ données a_1, a_2, \dots, a_n .

Sortie : Une permutation des données telle que

$$a'_1 \leq a'_2 \leq \dots \leq a'_n.$$

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.

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 compromis entre facilité d'expression et précision.

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

```
1  ALGORITHME Somme(n): entier
2  DONNEES:
3      n: entier
4  VARIABLES:
5      i: entier
6      s: entier
7  DEBUT
8      i ← 1
9      s ← 0
10     TQ  $i \leq n$  FAIRE
11         s ← s + i
12         i ← i + 1
13     FTQ
14     RENOYER s
15 FIN
```

Langage de programmation (Python)

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

- ❖ 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.

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.

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'algorithme : la séquence des instructions à exécuter, 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

```
1  ALGORITHME ListeSomme(L): liste d'entiers
2  DONNEES:
3    L: liste d'entiers
4  VARIABLES:
5    i: entier
6    s: entier
7  DEBUT
8    i ← 1
9    s ← 0
10   TQ i ≤ #L FAIRE
11     s ← s + L[i]
12     i ← i + 1
13   FTQ
14   RENOYER s
15 FIN
```

Exemple 2 : primalité d'un nombre

```
1  ALGORITHME EstPremier(n): booleen
2  DONNEES:
3    n: entier
4  VARIABLES:
5    d: entier
6  DEBUT
7    d ← 2
8    TQ d < n FAIRE
9      SI n mod d = 0 ALORS
10       RENOYER FAUX
11     FSI
12     d gets d+1
13   FTQ
14   RENOYER VRAI
15 FIN
```


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.

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

```
1  ALGORITHME Somme(n): entier
2  DONNEES:
3    n: entier
4  VARIABLES:
5    i: entier
6    s: entier
7  DEBUT
8    i ← 1
9    s ← 0
10   TQ i ≤ n FAIRE
11     s ← s + i
12     i ← i + 1
13   FTQ
14   RENOYER s
15 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

```
1  ALGORITHME Somme(n): entier
2  DONNEES:
3    n: entier
4  VARIABLES:
5    i: entier
6    s: entier
7  DEBUT
8    i ← 1
9    s ← 0
10   TQ i ≤ n FAIRE
11     s ← s + i
12     i ← i + 1
13   FTQ
14   RENOYER s
15 FIN
```

Preuve de validité

Il faut montrer que les valeurs des différentes variables sont bien en rapport avec le résultat attendu.

On cherche une propriété dépendante des variables qui reste vraie tout au long de l'algorithme. Ici par exemple, en début de boucle, on a toujours $s = \sum_{k=1}^{i-1} k = 1+2+\dots+(i-1)$.

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).

❖ 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 $\sum_{i=0}^{n+1} i = \frac{(n+1)(n+2)}{2}$.

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

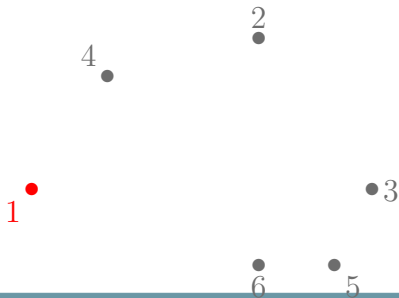
- ❖ Faire une preuve de validité peut être difficile.
- ❖ Avant de se lancer dans une preuve on vérifie que l'algorithme n'est pas trivialement faux.
- ❖ Montrer qu'un algorithme n'est pas valide est en général plus simple : il suffit d'exhiber un contre exemple.

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

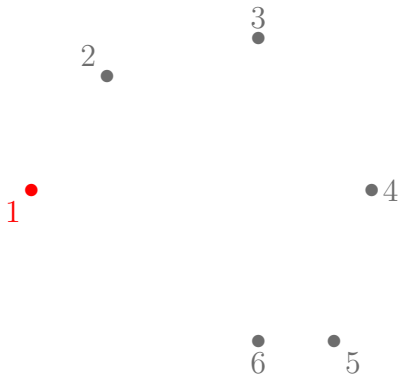


Invalider un algorithme

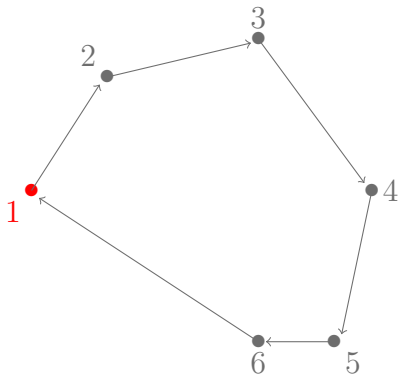
```
1 ALGORITHME RamassePlots()  
2 DEBUT  
3   Ramasser le plot 1  
4   TQ (il reste des plots) FAIRE  
5     ramasser le plot le plus proche  
6   FTQ  
7     revenir au debut  
8 FIN
```

Invalider un algorithme

Instance favorable

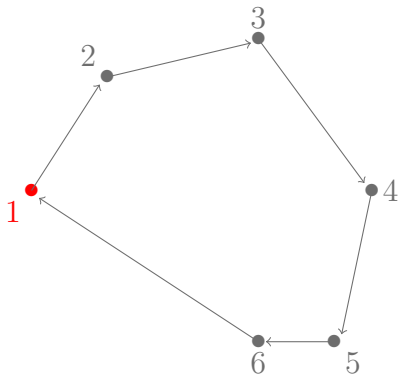


Instance favorable



Invalider un algorithme

Instance favorable

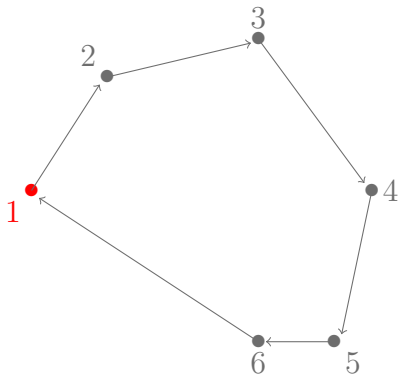


Instance défavorable

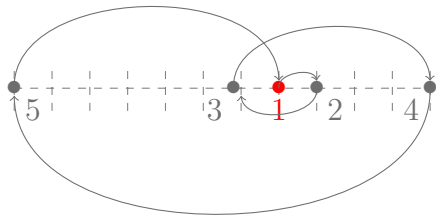


Invalider un algorithme

Instance favorable



Instance défavorable



Invalider un algorithme

```
1 ALGORITHME RamassePlotsCorrect()  
2 DEBUT  
3   Calculer la longueur de tous les parcours possibles  
4   Choisir le parcours le plus court  
5 FIN
```

Invalider un algorithme

```
1 ALGORITHME RamassePlotsCorrect()  
2 DEBUT  
3   Calculer la longueur de tous les parcours possibles  
4   Choisir le parcours le plus court  
5 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.