Philippe Gambette

INF120 - Algorithmique

Page web du cours : http://tinyurl.com/INF120-2011S1

- **Contact**

- Courriel : philippe.gambette@gmail.com
  (INF120 doit apparaître dans le sujet du courriel)
- Avant ou après le cours

- **Matériel**

- Ordinateur portable : interdit pendant les TP, a priori inutile en cours et TD.
- Pas de téléphone portable pendant cours/TD/TP

- **Déroulement des enseignements**

- Pause (méritée) de 5 minutes entre deux heures de TD
- Passages au tableau pour les corrections d'exercices en TD

- **Notes et devoirs**

- Interrogations QCM en début de cours ou TD
  (signalement des absences pour rattrapage, voir intranet)
- Un devoir maison

- **Note finale**

- Prévision : environ 2/3 “compétences”, environ 1/3 “motivation”
- Compétences : 2/3 devoir final (11 janvier 2012), 1/3 QCM
- Motivation : devoir maison, exercices, TP

- **Exercices supplémentaires d'entraînement**

- Sur demande, par courriel
- Sur demande, possibilité d'organiser une séance d'exercices ou de préparation au devoir final.

- **Sources**

Le livre de Java premier langage, A. Tasso
http://www.pise.info/algo/introduction.htm
Cours INF120 de J.-G. Luque
http://xkcd.com, http://xkcd.free.fr
http://serecom.univ-tln.fr/cours/index.php/Algorithmie
Cours de J. Henriet :
http://julienhenriet.olympe-network.com/Algo.html

A quoi sert un algorithme ?

- À décrire les étapes de résolution d'un problème : ("**étapes**" aussi appelées "**pas de l'algorithme**")
  → de façon structurée et compacte (méthode **facile à comprendre, facile à transmettre**)
  → à partir d'opérations de base (méthode a**daptée aux moyens à disposition, adaptée aux connaissances de celui qui l'utilise**)
  → indépendamment d'un langage de programmation (méthode **adaptée pour des problèmes qui se traitent sans ordinateur, compréhensible sans apprendre un langage de programmation**)

- Problème : données en entrée —*algorithme*→ résultat en sortie

Algorithmes sans ordinateurs :
- Euclide (vers -300) : calcul du PGCD de 2 nombres
- Al-Khuwārizmī (825) : résolution d'équations
- Ada Lovelace (1842) : calcul des nombres de Bernoulli sur la *machine analytique* de Charles Babbage

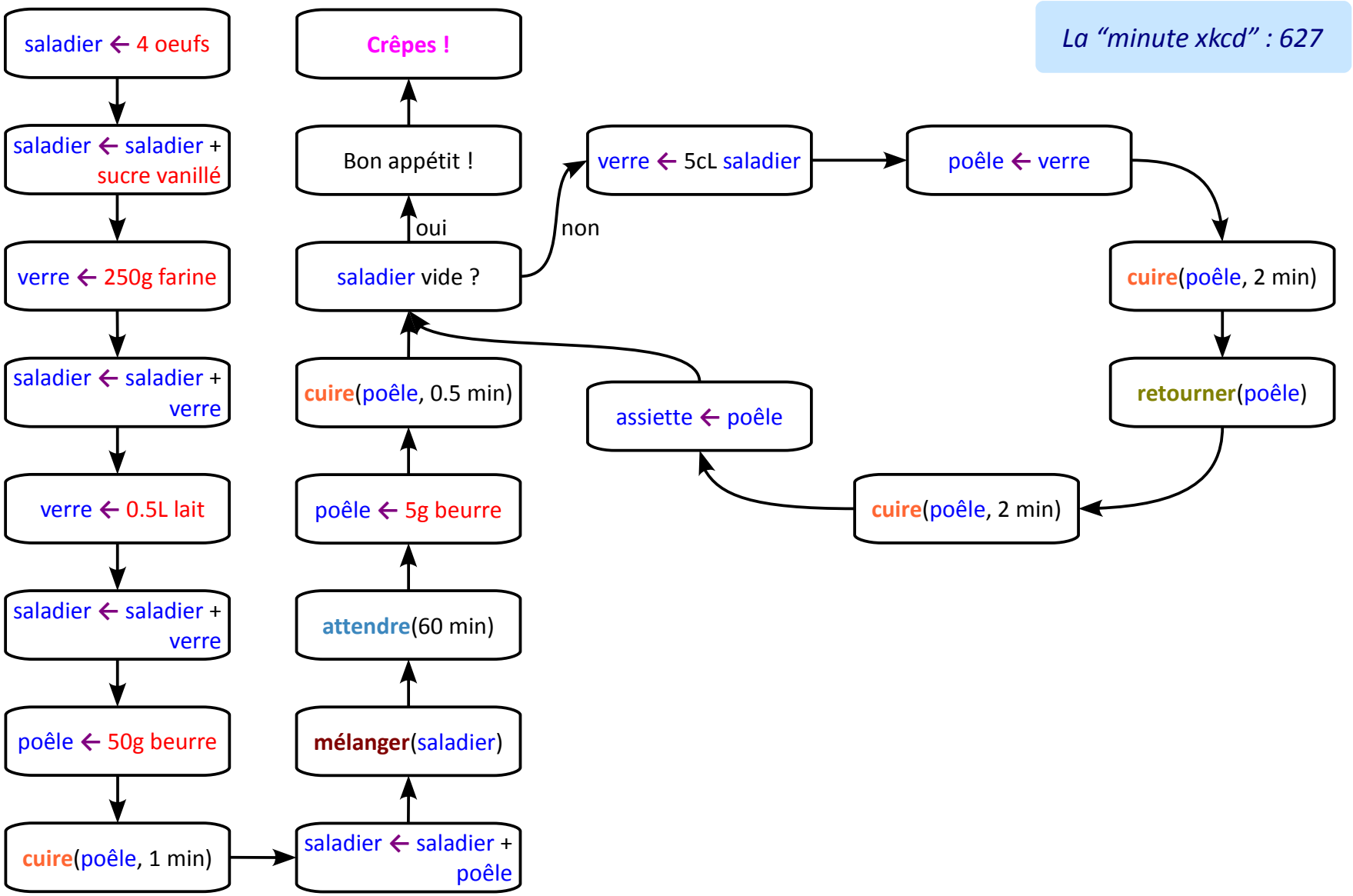
- *L' « algorithme des crêpes »*

Ingrédients : beurre, oeufs, sachets de sucre vanillé, farine, lait, sel
Récipients : saladier, verre mesureur, poêle, assiette,
Opérations de base : ***mettre dans un récipient***, ***mélanger***, ***attendre pendant … minutes***, ***retourner***, ***laisser cuire pendant ... minutes***

Algorithme des crêpes :

Mettre 4 oeufs **dans** le saladier
Mettre 1 sachet de sucre vanillé **dans** le saladier
Mettre 250 g de farine **dans** le verre mesureur
Mettre le contenu du verre mesureur **dans** le saladier
Mettre 0,5 litre de lait **dans** le verre mesureur
Mettre le contenu du verre mesureur **dans** le saladier
Mettre 50 grammes de beurre **dans** la poêle
Laisser cuire la poêle **pendant** 1 **minute**
Mettre le contenu de la poêle **dans** le saladier
Mélanger le contenu du saladier
Attendre pendant 60 **minutes**
Mettre 5 grammes de beurre **dans** la poêle
Laisser cuire la poêle **pendant** 0.5 **minute**
Tant que le saladier n'est pas vide :
  Mettre 5 cL du contenu du saladier **dans** le verre mesureur
  Mettre le contenu du verre mesureur **dans** la poêle
  Laisser cuire la poêle **pendant** 2 **minutes**
  Retourner le contenu de la poêle
  Laisser cuire la poêle **pendant** 2 **minutes**
  Mettre le contenu de la poêle **dans** l'assiette

Organigramme de la recette des crêpes



Enjeux algorithmiques : terminaison, correction, complexité

Terminaison : L'algorithme se termine-t-il ?

Preuve de terminaison :

- critère qui diminue à chaque boucle de l'algorithme
  → quantité de pâte à crêpes
- plus formellement : suite **à valeurs entières positives, strictement décroissante** à chaque boucle de l'algorithme

La "minute mathématique"

Théorème :

Toute suite à valeurs entières positives, strictement décroissante, ne peut prendre qu'un nombre fini de valeurs.

Application :

Trouver la suite qui convient pour prouver la terminaison d'un algorithme.

La "minute votes SMS"

Problème : *aller en voiture de Châtelet à la Tour Montparnasse*

Algorithme "du repère visuel" :

A tout instant on sait où se trouve la Tour Montparnasse
→ prendre la rue qui s'en rapproche le plus

NE TERMINE PAS NECESSAIREMENT

Exige-t-on que tout bon algorithme se termine ?
Oui en général (exceptions : algorithme de contrôle d'un ascenceur, de transactions financières, ...)

Correction : L'algorithme donne-t-il le résultat attendu ?

Preuve de correction :

- « invariant » : propriété vraie tout au long de l'algorithme
  - vraie à la première étape
  - si vraie à une étape, vraie à l'étape suivante
  - ⇨ vrai à la fin

En pratique, pour débuter :

- vérifier sur les "cas de base"
- vérifier sur des exemples aléatoires

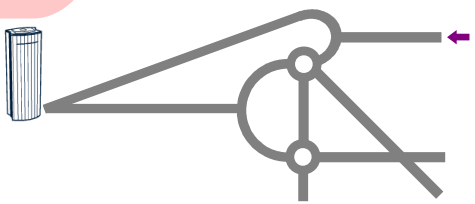
Complexité : Combien de temps l'algorithme prend-il pour se terminer ?

Théorie de la complexité :

- évaluer le nombre d'opérations en fonction de la taille du problème, dans le pire cas
- prouver qu'on ne peut pas utiliser moins d'opérations pour résoudre le problème, dans le pire cas

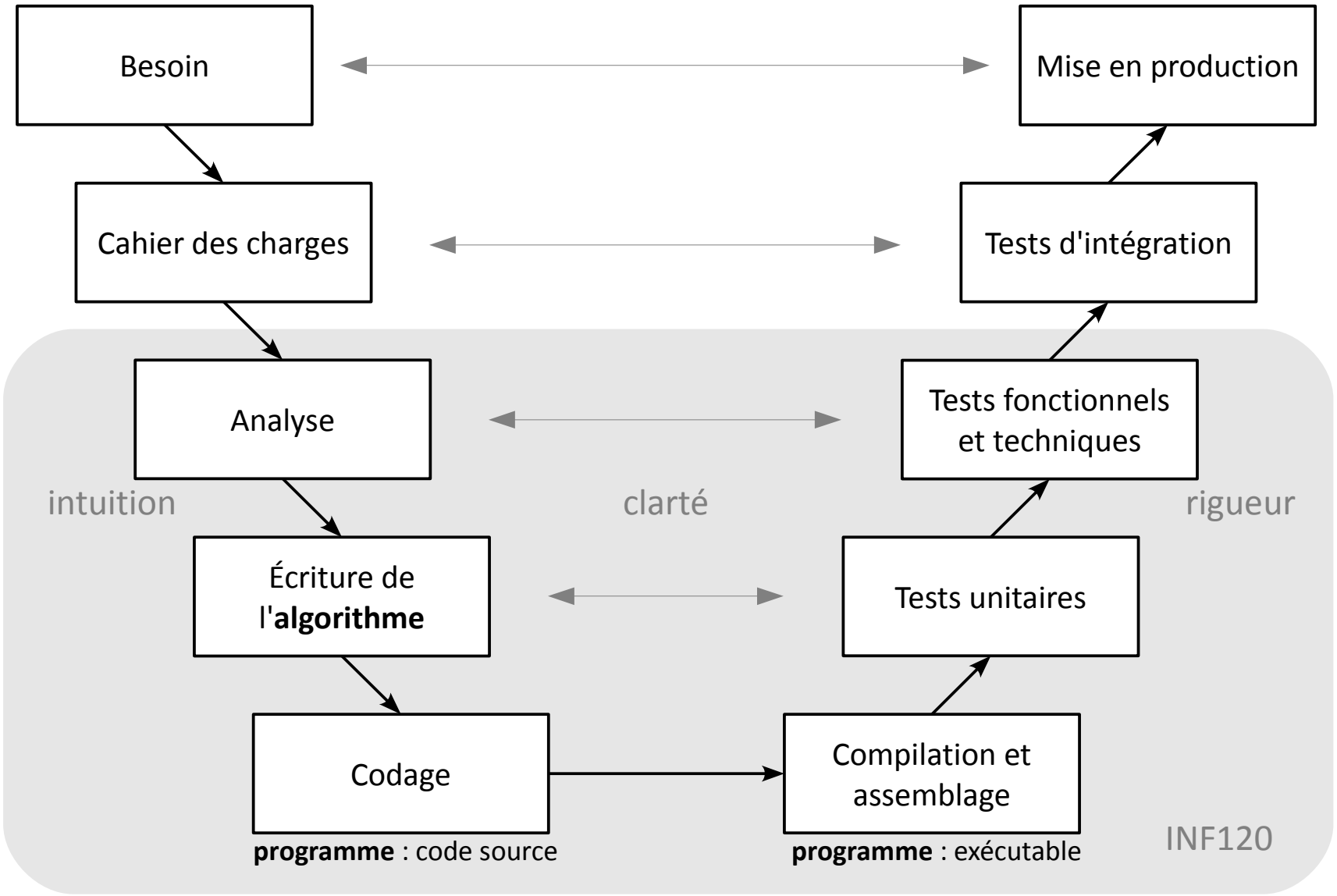
En pratique :

- vérifier sur des exemples aléatoires
- connaître les cas difficiles

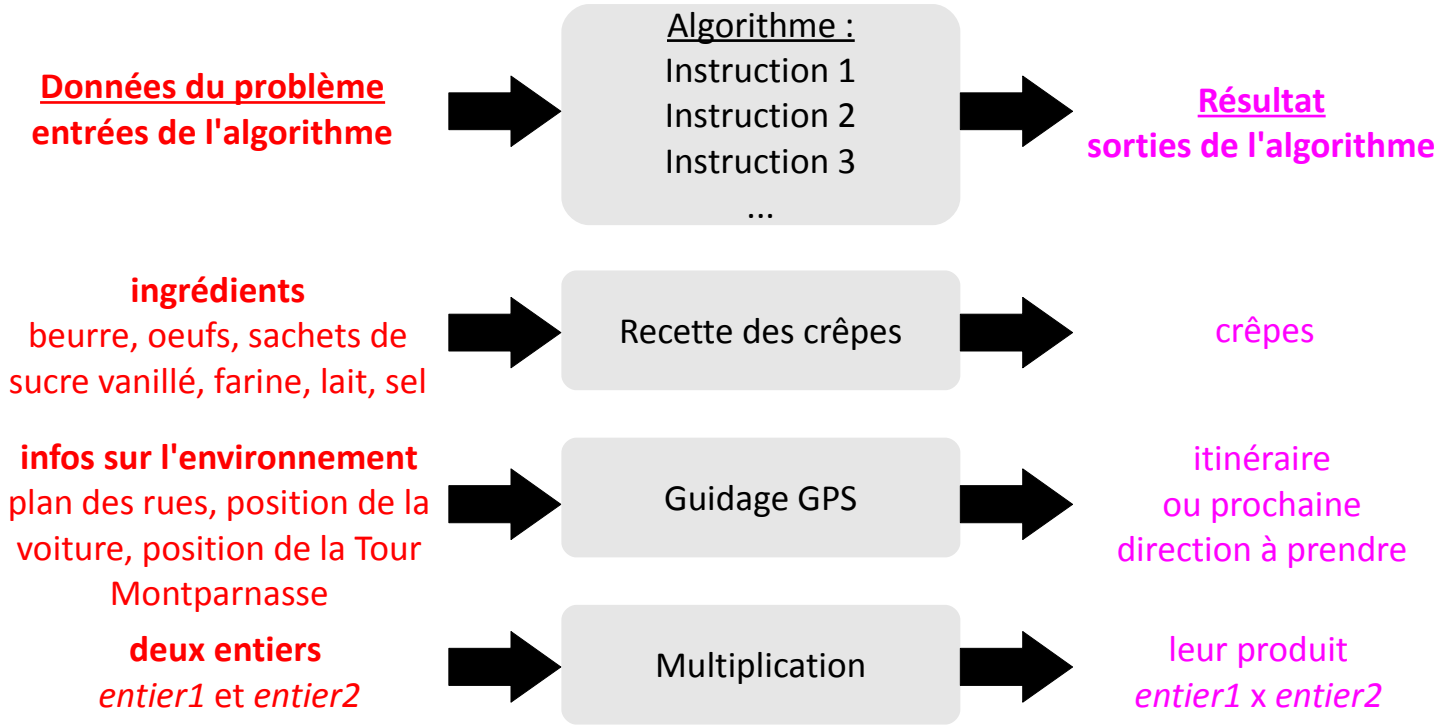


algorithme ***glouton*** :
toujours choisir le profit maximum.
Pas toujours la meilleure stratégie.

De l'algorithme au programme



Composants d'un algorithme



Divers **types d'instructions** d'un algorithme :
- déclaration d'un algorithme
- appel d'un algorithme
- déclaration d'une variable
- affectation d'une variable
- entrées / sorties
- boucle
- test

Variables et affectation

Dans un algorithme, une **variable** possède :
• un **nom**,
• une **valeur**,
• un **type** (ensemble des valeurs que peut prendre la variable).

La **valeur** d'une variable :
• est **fixe à un moment donné**,
• peut **changer au cours du temps**.

L'**affectation** change la valeur d'une variable :
• *a* ← 5 :
- la variable *a* prend la valeur 5
- la valeur précédente est perdue (“écrasée”)
• *a* ← *b* :
- la variable *a* prend la valeur de la variable *b*
- la valeur précédente de *a* est perdue (“écrasée”)
- la valeur de *b* n'est pas modifiée
- *a* et *b* doivent être de même type
(ou de type compatible)

Dans un **algorithme**, choisir pour les variables :
• un nom composé de **lettres** et éventuellement de **chiffres**
• un nom **expressif**, par exemple :
- *chaine*, *requête1*... pour une chaîne de caractères
- *n*, *a*, *compteur*, *nbOperations*, *longueur*... pour un entier
- *x*, *y*, *température* pour un réel
- *estEntier*, *testEntier*, *trouvé*... pour un booléen
• un nom **assez court** (il faut l'écrire !)
• éviter les **noms réservés** : *pour*, *tant que*, *si*...

Dans un **programme** :
• **éviter** les lettres accentuées et la ponctuation
• préférer l'**anglais** si votre code source est diffusé largement
• être **expressif** et **lisible** :
- *est_entier* ou *estEntier* plutôt que *estentier*

Votre code sera relu, par vous ou par d'autres...

Mon premier vrai algorithme

Je connais l'algorithme d'**addition** de deux entiers positifs.
Comment écrire un algorithme de **multiplication** de deux entiers ?

Intuition : $5 \times 3 = \underbrace{5 + 5 + 5}_{3 \text{ fois}}$

$$\textit{entier1} \times \textit{entier2} = \underbrace{\textit{entier1} + \textit{entier1} + \textit{entier1} + \ldots + \textit{entier1}}_{\textit{entier2 fois}}$$

Multiplication :

Entrées : deux entiers *entier1* et *entier2*
Sorties : le produit de *entier1* et *entier2*
Variables : entiers *compteur* et *produit*

```
Début
    compteur ← 0
    produit ← 0
    Tant que compteur < entier2 faire :
        produit ← addition(produit, entier1)
        compteur ← addition(compteur, 1)
    Fin tant que
    renvoyer produit
Fin
```

Terminaison ? Correction ?

Essayer avec *entier1* = 5 et *entier2* = 3

Preuve dans un prochain cours...

Valeurs avant la *i*-ième boucle *Tant que* :

i	1	2	3	4	5
compteur	0	1	2	3	-
produit	0	5	10	15	-

Remplir ce tableau, c'est **faire la trace de l'algorithme** :
savoir à chaque moment la valeur de toutes les variables