

Al-Khwarizmi (vers 780 – vers 850)



Mathématicien persan du IX^e siècle. Pour aider ses contemporains, ce sage a écrit un livre regroupant des méthodes claires, à suivre pas à pas, pour résoudre des problèmes mathématiques. De plus, le titre de ce livre contient l'expression « Al Jabr » (lisez-le à haute voix) ce qui explique que son nom soit associé à l'arrivée de l'algèbre en Europe. Son nom fut traduit par Algoritmi en latin.

Bref, avec Al-Khwarizmi, on voit apparaître des algorithmes complexes !

Source : [1]

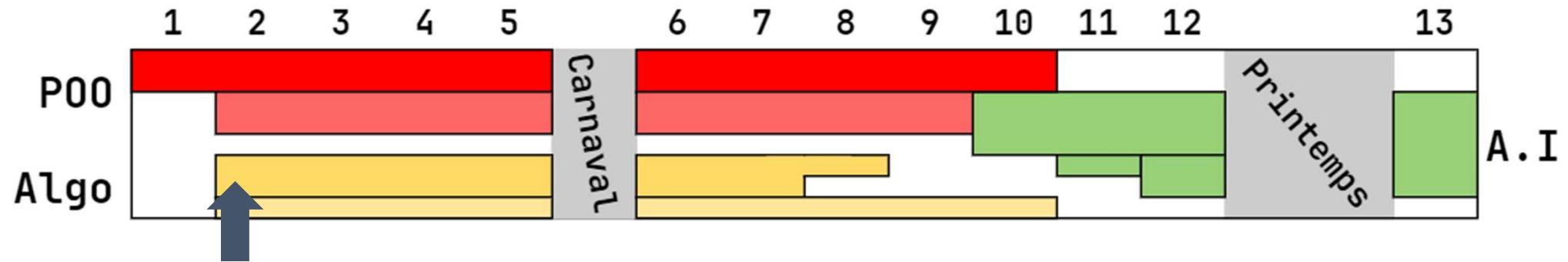
UE09 – Algorithmique Introduction

Simon LIÉNARDY

Bienvenue !

- Présentation de l'enseignant.

Vous êtes ici !



Qu'allons-nous faire ?

- Acquis d'apprentissages :
 1. **Programmer** des énoncés de conception en Java **selon les principes de la POO**.
 2. **Valider** les comportements des objets programmés par des **tests unitaires**.
 3. **Documenter** son code et rédiger un rapport technique spécifiant un problème donné, incluant un **plan de test**, une évaluation de **complexité** et une solution correspondante.
 4. **Justifier** le choix d'une **structure de données** particulière (tableaux, listes, files, piles, arbres) et d'une implémentation spécifique
 5. **Présenter et défendre** le fruit de son travail de réflexion, de comparaison, de création oralement selon un schéma/canevas spécifique.

Plan du cours – 2 thèmes principaux

Thème 1 – Analyse critique de la qualité d'un programme, appliquée

- à des algorithmes célèbres ;
- à vos propres programmes.

Thème 2 – Utiliser la récursivité et des structures de données complexes (piles, files, listes, arbres, images...) sur des exemples d'algorithmes célèbres (tri, chemins, jeux...)

Avant de commencer – Règles

- Arriver à l'heure au cours
 - Si le cours a lieu en début de journée **et que vous avez un peu de retard dû aux transports en commun**, alors entrez discrètement.
 - Panne de réveil ? Attendez la pause en allant boire un café.
- Se munir de son ordinateur portable suffisamment chargé
- Se munir d'un **bloc de feuilles de papier** pour dessiner et **réfléchir**

Faisons connaissance

- Instructions de connexion au Wooclap diffusée sur l'écran.

Quelques conseils

- Travailler régulièrement
 - Participation au cours et au labo
- Se poser régulièrement les questions :
 - Qu'est-ce que j'ai compris ?
 - Serai-je capable de le réexpliquer à quelqu'un ?
- Collaborer entre vous
 - Intelligence collective
 - **# recopier / plagier un·e pote ou un site internet**

Et l'IA ?

Place de l'IA dans le cours

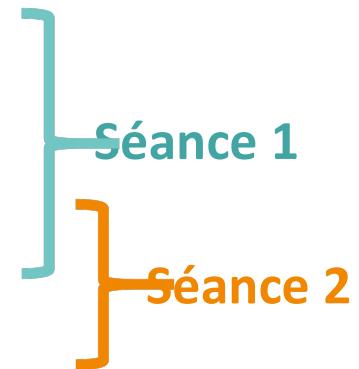
- La question à se poser est la suivante :

« L'utilisation de l'IA pour générer directement le code qu'on me demande de concevoir est-elle favorable au développement de mes capacités cognitives ? »

- **Pour l'instant**, de nombreuses **communications scientifiques** alertent sur une réponse **défavorable**.

Thème 1 – Analyse de la qualité d'un programme

1. Un premier exercice
2. Définitions
3. Exemples d'algorithme
4. Caractéristiques (domaine, exactitude, efficacité)
5. La notion de complexité
6. Comparaison d'algorithmes polynomiaux
7. Types de problèmes



Premier exercice

Qu'est-ce qu'un bon programme ?

- Réfléchissez à votre définition d'un « bon programme » – ⏳ 3'
- Partagez cette définition avec votre voisin – ⏳ 2'
 - Aviez-vous oublié quelque chose ?
 - Y'a-t-il des qualités plus importantes que d'autres ?
- Avec le fichier [ExemplesAlgorithmes.java](#) (dispo. sur LEARN) – ⏳ 5'
 - Confrontez votre définition aux trois exemples de fonction
 - lesquels selon vous sont des bons programmes ? Et pourquoi ?
 - Votre définition doit-elle être modifiée ?

Premier exercice – Mise en commun

Sur Wooclap, pour chaque fonction, quelles qualités et défauts voyez-vous dans chacun des programmes ?

- Faites précéder les qualités d'un « + », ex. : « + savoureux »
- Faites précéder les défauts par un « - », ex. : « - dissolu »
- **Lisez d'abord** les réponses des autres ;
- Si une qualité est déjà mentionnée, « likez-là ».

Sommaire

1. Un premier exercice
2. Définitions
3. Exemples d'algorithme
4. Caractéristiques (domaine, exactitude, efficacité)
5. La notion de complexité
6. Comparaison d'algorithmes polynomiaux
7. Types de problèmes

Définitions informelles

« Procédure de calcul bien définie qui prend en **entrée** une valeur, ou un ensemble de valeur, et qui donne en **sortie** une valeur ou un ensemble de valeurs. Un algorithme est donc une séquence d'étapes de calcul qui transforment l'entrée en sortie. » [2]

ou « outil permettant de résoudre un **problème de calcul** bien spécifié. L'énoncé du problème spécifie, en termes généraux, la relation désirée entre l'entrée et la sortie. L'algorithme décrit une procédure de calcul spécifique permettant d'obtenir cette relation entrée/sortie. » [ibid.]

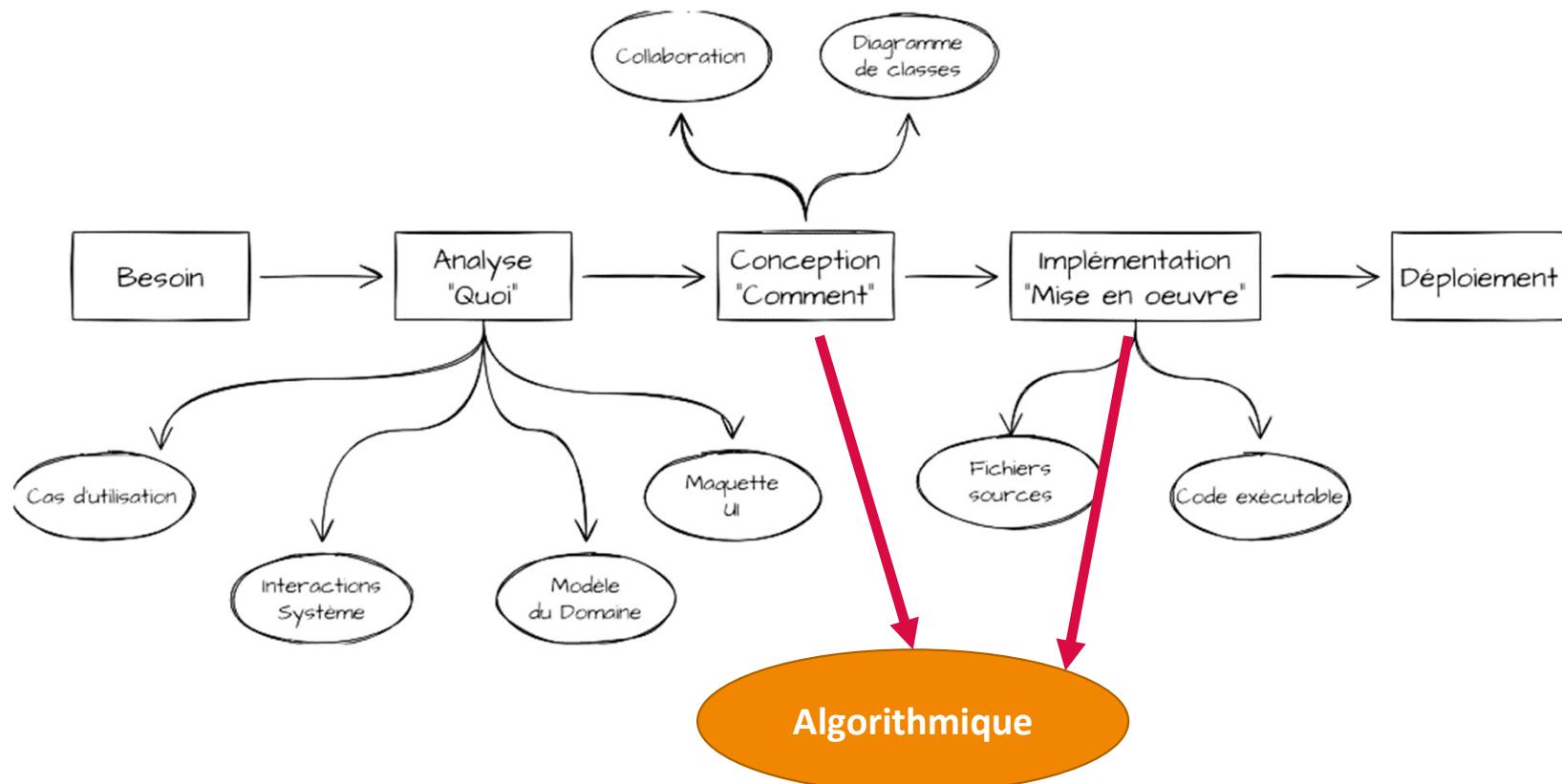
Exemples d'algorithmes

- Déjà rencontrés dans d'autres cours ?
- Rencontrés dans la vie de tous les jours ?
- Quelles en sont les entrées et sorties ?

Remarques

- Pour résoudre un problème, il peut exister **plusieurs** algorithmes
 - Lequel choisir ?
 - Dans quel langage de programmation le traduire (= l'implémenter) ?
- Parfois, **il n'existe pas** d'algorithme pour résoudre un problème !
 - À priori, difficile à concevoir
 - Mais c'est un résultat **mathématique** (d'une branche des math appelée la *Calculabilité*)
 - Exemple :
 - **input**: la description d'un programme et un paramètre d'entrée
 - **output**: **true** si le programme s'arrête si l'exécute, **false** sinon.

Développement logiciel - Algorithmique



Source : [3, p. 174]

Sommaire

1. Un premier exercice
2. Définitions
3. Exemples d'algorithme
4. Caractéristiques (domaine, exactitude, efficacité)
5. La notion de complexité
6. Comparaison d'algorithmes polynomiaux
7. Types de problèmes

Exemple d'algorithme : Bin-packing



Bin-packing – Contexte

Énoncé : On cherche à ranger un certain nombre de « boites » de tailles et volumes variés dans des conteneurs d'une certaine taille, de manière à optimiser l'espace (en utilisant le moins possible de conteneurs).

Exemples concrets:

- Répartir des groupes de touristes dans des bus sans séparer les membres d'un même groupe
- Agencer des containers de marchandise sur un/des bateau(x) de transport
- Expédier les commandes en ligne d'un client en un minimum de colis

Bin-packing - Contexte

Énoncé

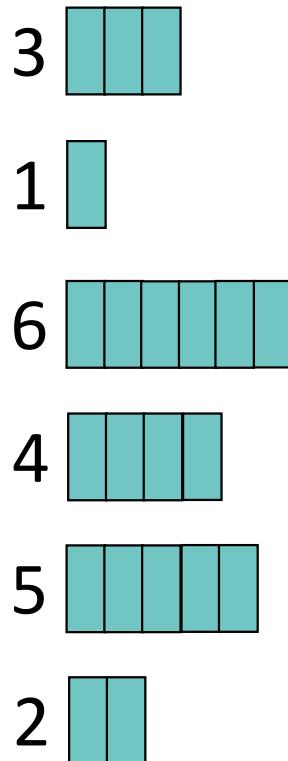
- Six groupes de personnes de tailles 3, 1, 6, 4, 5 et 2 doivent être déplacées à bord de minibus pouvant accueillir 7 passagers.
- Toutefois, les groupes ne peuvent pas être séparés !
- Trouver le nombre minimum de minibus pour les déplacer (sans diviser les groupes)

Bin-packing - Consigne

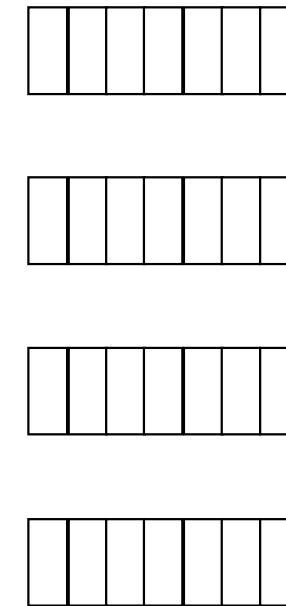
1. Prenez une feuille de papier et représentez la situation.
2. Imaginez une méthode de résolution et notez-la. Cette méthode doit être explicable à un être humain ( : 5')
3. Comparez votre réponse avec votre voisin. ( : 5')
 - Plus spécifiquement, chacun d'entre vous suit les consignes de l'autre
 - Arrivez-vous au même résultat ?
 - Rédiger un document ensemble (nous en rediscuterons dans la suite)

Bin-packing – Illustration

Groupes



Minibus



Ada Lovelace (1815 – 1852)



Femme de sciences anglaise. En anticipant le potentiel de la machine analytique, certainement plus que Charles Babbage lui-même, qui en était pourtant le concepteur, elle a proposé de nombreux travaux sur cette dernière, en particulier, la première série d'instructions exécutables par cette machine.

Bref, Ada Lovelace est la première codeuse au monde !

Pour en savoir plus : [Algorithmes, mode d'emploi](#)

Source : [1]

UE09 – Algorithmique Introduction

Simon LIÉNARDY

Sommaire

1. Un premier exercice
2. Définitions
3. Exemples d'algorithme
4. Caractéristiques (domaine, exactitude, efficacité)
5. La notion de complexité
6. Comparaison d'algorithmes polynomiaux
7. Types de problèmes

Exemple d'algorithme : Bin-packing



Bin-packing – Contexte

Énoncé : On cherche à ranger un certain nombre de « boites » de tailles et volumes variés dans des conteneurs d'une certaine taille, de manière à optimiser l'espace (en utilisant le moins possible de conteneurs).

Exemples concrets:

- Répartir des groupes de touristes dans des bus sans séparer les membres d'un même groupe
- Agencer des containers de marchandise sur un/des bateau(x) de transport
- Expédier les commandes en ligne d'un client en un minimum de colis

Bin-packing - Contexte

Énoncé

- Six groupes de personnes de tailles 3, 1, 6, 4, 5 et 2 doivent être déplacées à bord de minibus pouvant accueillir 7 passagers.
- Toutefois, les groupes ne peuvent pas être séparés !
- Trouver le nombre minimum de minibus pour les déplacer (sans diviser les groupes)

Bin-packing – « First-fit »

Groupes

3

1

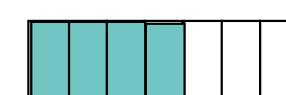
6

4

5

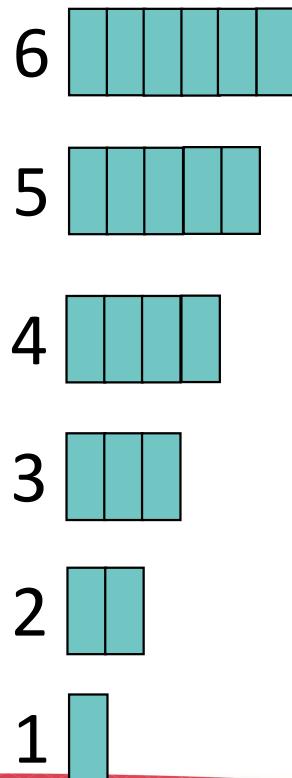
2

Minibus

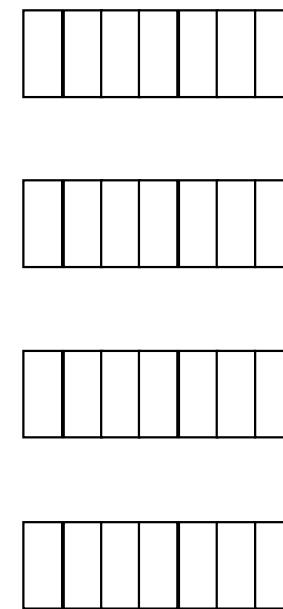


Bin-packing – « First-fit decreasing »

Groupes



Minibus



Bin-packing – « First-fit decreasing »

Groupes

6

5

4

3

2

1

Minibus



Sommes-nous sûr·e·s de nos constatations ?

« Le first-fit decreasing est *mieux* que le first-fit »

⚠ Si vous avancez ce genre de déclarations, il *vaut mieux* être en mesure de le **prouver** !

💡 Pensez à d'autres **instances** du problème

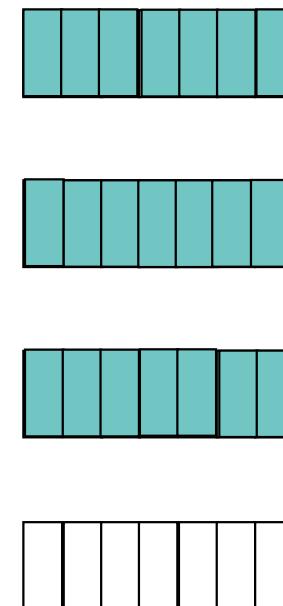
= le même problème mais avec d'autres groupes de personnes

Bin-packing – « Full-bin packing »

Groupes

3
1
6
4
5
2

Minibus



Bin-packing – Résumé

- Algorithme « first-fit »
 - + Rapide et facile à exécuter
 - + Fonctionne si les groupes arrivent au fur et à mesure
 - Ne mène généralement pas à une solution optimale
- Algorithme « first-fit decreasing »
 - + Généralement meilleur que le first-fit
 - + Rapide (mais demande plus de temps que le FF, pour le tri)
 - Ne mène pas toujours à une solution optimale
- Full-bin
 - + Mène généralement à une bonne solution
 - Difficile à exécuter, dépend des tailles et nombres de groupes

Conclusion

- Qu'avons-nous appris de cet exemple ?

Sommaire

1. Un premier exercice
2. Définitions
3. Exemples d'algorithme
4. Caractéristiques (Domaine, Exactitude, Efficacité)
5. La notion de complexité
6. Comparaison d'algorithmes polynomiaux
7. Types de problèmes

Sommaire

4. Caractéristiques

1. Domaine
2. Exactitude
3. Efficacité

Caractéristiques – 1. Domaine



- Quand vous écrivez un algorithme, devez-vous envisager toutes les entrées possibles et imaginables ?
 - Non, évidemment !
 - Parfois, parce que cela n'a pas de sens ?
 - Exemples ?

Caractéristiques – 1. Domaine

- **Domaine :** Vous avez déjà rencontré cette notion en base de données ou en mathématiques !
- Comment le définir ?
 - À l'aide d'une condition que doivent respecter les entrées : la **précondition**
 - Voir la 1^{re} partie de la **phase préparatoire**
- Exemple avec le Bin-packing ?
 - Réfléchissez-y individuellement – ⏳ 2'
 - Partagez vos réponses en petits groupes – ⏳ 5'

Bin-packing - Contexte

- Énoncé

Six groupes de personnes de tailles 3, 1, 6, 4, 5 et 2 doivent être déplacées à bord de minibus pouvant accueillir 7 passagers.

Toutefois, les groupes ne peuvent pas être séparés !

Trouver le nombre minimum de minibus pour les déplacer (sans diviser les groupes)

Préconditions ?

Caractéristiques - 2. Exactitude

- **Exactitude** : pour toute instance du problème posé, l'algorithme produit le résultat escompté en un temps fini



- Instance du problème = 1 jeu de valeurs particulières pour chaque paramètre du problème, respectant la précondition
- Résultat correct : il respecte la **postcondition**

Caractéristiques - 2. Exactitude

- En résumé : on va vérifier que le programme est correct
- À ne pas confondre avec la **robustesse**
 - = la capacité d'un programme à s'exécuter sans erreur, même dans des cas extrêmes (pannes de courant, surchauffe, attaque de pirates informatiques, fichiers ou réseau non disponible...).
- Plutôt liée à un langage ou un environnement d'exécution qu'à un algorithme

Caractéristiques - 2. Exactitude

- **Exactitude** : pour toute instance du problème posé, l'algorithme produit le résultat escompté en un temps fini

Caractéristiques – 2. Exactitude

- **Exactitude :**

- pour toute instance du problème posé,

Complétude

- l'algorithme

- produit le résultat escompté

Correction partielle

- en un temps fini.

Terminaison

Caractéristiques – 2. Exactitude – Définitions

Complétude : Pour chaque entrée du domaine, l'algorithme, s'il termine, fournit une solution.

Correction partielle : Si l'algorithme se termine en produisant une solution, alors cette solution est correcte.

Terminaison : L'exécution de l'algorithme prend un temps fini.

Exactitude = **Correction partielle** + **Terminaison** + **Complétude**

Correction

Caractéristiques - 2. Exactitude

```
/*
 * Calcule |x|
 */
static int abs(int x) {
    return -x;
}
```

Est-ce correct ?

```
/*
 * Détermine si x est premier
 */
static boolean isPrime(int x){
    return x % 2 == 1;
}
```

Est-ce correct ?

Caractéristiques – 2. Exactitude – Tests

- Les tests permettent de montrer la **présence** d'erreur
- Ils ne permettent **pas** de prouver leur **absence**
- MAIS ils n'en restent pas moins **indispensables** !
 - Pour déceler des erreurs de programmation ;
 - Pour rassurer le client ;
 - Pour s'assurer qu'une modification d'une autre partie du code n'a pas créé une erreur ailleurs
 - ...
- Une méthode de développement logiciel se base sur l'écriture de test **avant** le reste du code (TDD – Test Driven Development).

Caractéristiques - 2. Exactitude - Tests

- Un bon plan de test doit contenir
 - ✓ Différents cas distingués par l'algorithme
 - ✓ Y compris les cas d'erreurs prévus
 - ✓ Les cas limites
 - Ensemble vide
 - Valeur 0
 - Référence null
 - ✓ Des valeurs extrêmes (dépassemens de capacité)
 - ✓ Des combinaisons de tous ces cas

Dépassemement de capacité ? N'oubliez pas de redémarrer votre !

- « En 2015, il a été découvert qu'un bogue informatique pouvait empêcher les pilotes de garder le contrôle du Boeing 787, possiblement en plein vol, a informé la Federal Aviation Administration (FAA) dans une directive destinée à toutes les compagnies aériennes utilisant cet avion. »
- Selon la FAA. « Si les quatre unités de contrôle des générateurs (associées aux générateurs du moteur) venaient d'être allumées simultanément, après 248 jours d'allumage, tous les générateurs entreront en mode sécurité-défaut, causant une perte de tout le courant électrique, quelle que soit la phase de vol. »
- La note de la FAA n'avait pas donné plus de détails sur ce bogue, mais il s'agirait d'un problème de débordement d'entier 32-bit déclenché après 2^{31} centisecondes (248,55 jours) de fonctionnement continu. 2^{31} étant le nombre de secondes dans 248 jours multipliés par 100 (un compteur en centièmes de s).

Caractéristiques - 2. Exactitude - Tests

- Exemple de plan de test pour le Bin-packing ?
- Réfléchissez-y – ⏳ 3'

Caractéristiques - 2. Exactitude - S'en assurer

Comment s'assurer de l'exactitude d'un algorithme ?

- Utiliser un algorithme de **référence**, connu
- Le **démontrer** mathématiquement
- Le concevoir avec **méthode**
- Avantages et inconvénients de ces approches ?

Caractéristiques - 2. Exactitude - Méthode

```
static float puissance (float nombre, int exposant) {  
    assert(exposant ≥ 0 || nombre ≠ 0): "Erreur - division par zéro";  
    float resultat = 1;  
    int compteur = exposant;  
    while(compteur > 0) {  
        resultat *= nombre;  
        compteur -= 1;  
    }  
    while(compteur < 0) {  
        resultat /= nombre;  
        compteur += 1;  
    }  
    return resultat; Postcondition : resultat = nombreexposant  
}
```

Vérifie la précondition

```
static float puissance (float nombre, int exposant) {  
    assert(exposant >= 0 || nombre != 0): "Erreur - division par zéro";  
  
    float resultat = 1;  
    int compteur = exposant;  
  
    while(compteur > 0) {  
  
        resultat *= nombre;  
        compteur -= 1;  
    }  
  
    /* reste du calcul execute si exposant < 0 */  
    return resultat;  
}
```

- À chaque itération, `resultat` est multipliée par `nombre`
- `resultat` est une puissance de `nombre`
- On peut exprimer une propriété qui ne change pas durant les itérations de la boucle = son **invariant**
- `resultat = nombre(exposant-compteur)` et `0 <= compteur <= exposant`

Quelques mots sur l'invariant

Un **invariant de boucle** est une **condition** qui respecte les propriétés suivantes :

Initialisation : Il est vrai avant la première itération de la boucle

Conservation : S'il est vrai avant une itération de la boucle, il reste vrai avant l'itération suivante

Terminaison : Une fois la boucle terminée, l'invariant fournit une propriété utile qui aide à montrer la validité de l'algorithme.

```
static float puissance (float nombre, int exposant) {  
    assert(exposant >= 0 || nombre != 0): "Erreur - division par zéro";  
    // Précondition  
    float resultat = 1;  
    int compteur = exposant;  
    // res. = nombre(exposant-compteur) et 0 <= compteur <= exposant  
    while(compteur > 0) {  
        // res. = nombre(exposant-compteur) et 0 < compteur <= exposant  
        resultat *= nombre;  
        compteur -= 1;  
        // Invariant conservé ?  
    } // Ici, compteur <= 0 et Inv => compteur == 0  
    // resultat = nombre(exposant-0) => Postcondition  
    /* reste du calcul executé si exposant < 0 */  
    return resultat;  
}
```

Ok car $\text{nombre}^0 = 1 = \text{resultat}$

Caractéristiques - 2. Exactitude - Terminaison

- Qu'est-ce qui nous garantit que la boucle se termine ?

```
while(compteur > 0) {  
    resultat *= nombre;  
    compteur -= 1;  
}
```

Pour montrer la terminaison, on doit chercher une fonction :

1. À valeur entière
2. Qui combine des valeurs des variables utilisées dans la boucle
3. Strictement positive si le **gardien** est vrai
4. Dont la valeur décroît entre deux itérations consécutives

Caractéristiques - 2. Exactitude - Terminaison

- Si vous êtes capables de trouver une telle fonction, votre boucle se terminera
- On l'appelle « fonction de terminaison »
- Pourquoi cela fonctionne-t-il ?
 - Votre fonction met en correspondance l'état du programme à chaque itération et un nombre entier
 - D'une itération à l'autre, elle **doit** décroître
 - De plus, **il n'existe pas de suite infinie décroissante de nombres entiers positifs**
 - Donc, l'existence de votre fonction implique, puisqu'elle met en correspondance des itérations et des entiers, qu'il n'y a pas une suite infinie d'itération
 - La boucle se termine donc !
- Exemple pour l'algorithme de la puissance ?

Terminaison - Contrexemple

```
int valeur = Console.lireInt("Valeur d'entree ? ");

while (valeur > 1) {
    if (valeur % 2 == 1) {
        valeur = 3 * valeur + 1;
    }
    else {
        valeur /= 2;
    }
    System.out.println(valeur);
}
System.out.println("Fini !");
```

Caractéristiques – 3. Efficacité

- On veut mesurer les ressources à mettre en œuvre pour exécuter l'algorithme.
- Ressources ?
 - Espace mémoire ?
 - Temps de calcul ?
 - Bande passante ?
 - Nombre de transferts ?
- En faisant ***abstraction*** du matériel exact qui équipe l'ordinateur sur lequel est exécuté l'algorithme !

Dans ce cours

Sources

1. Maxime Amblard & Christine Leininger. (2019) *Famille « Algorithmes & programmation »*. <https://interstices.info/famille-algorithmes-programmation/> (Consulté le 27/01/24)
2. Thomas Cormen, Charles Leiserson, Ronald Rivest & Clifford Stein. (2010) *Algorithmique*. Dunod
3. Nicolas Hendrikx. (2023) *Programmation Orientée Objet*. Syllabus du cours