Algorithmique – série d'exercices

Test, boucle, procédure, fonction, tableau, structure

Optimisation et complexité

Table des matières

Méthode de base d'analyse algorithmique

La méthode de base pour écrire un algorithme suit les 4 étapes suivantes :

1. Comprendre le problème : bien lire le sujet et bien comprendre ce qu'il y a à faire.
2. Lister ce dont on a besoin pour résoudre le problème (les données) et ce qu'on va produire (les résultats).
3. Trouver un principe de résolution : se donner les grandes lignes, en français, de la méthode de résolution. Eventuellement, se donner des procédures ou des fonctions (des actions générales).
4. Ecrire l'algorithme en détail.

Tests, boucle, procédure, fonction

Exercice 1 - année bissextile

Ecrire un programme qui affiche le nombre de jours d'un mois d'une année donnée. On rappelle que les années bissextiles sont celles qui sont divisibles par 4. Toutefois, les années divisibles par 100 ne sont pas bissextiles. Mais les années divisibles par 400 sont quand même bissextiles. Par exemple : 2008 est bissextile, 2010 n'est pas bissextile, 2000 est bissextile, 1900 n'est pas bissextile.

Exercice 2 – racine Newton

Ecrire un programme qui calcule la racine carré d'un nombre en appliquant la méthode de Newton.

Racine(A) est donnée par le calcul de la suite suivante :

$S(0) = 1$

$S(n+1) = 0,5 * (S(n) + A / S(n))$

Quand deux valeurs successives de la suite sont identiques, on a trouvé la racine.

On n'utilisera pas de tableau.

Exercice 3 – Fonctions des précédents

Pour les deux exercices précédents, écrire une procédure (ou une fonction) qui traite le problème et le programme qui utilise cette procédure.

Exercice 4 – équation du second degré

Écrire une procédure (ou une fonction) qui résout dans IR une équation du second degré en traitant tous les cas possibles (si a=0, etc.)

Trouver l'algorithme le plus modulaire possible (pensez à la résolution d'une équation du premier degré).

Ecrire le programme qui teste cette procédure

Tableaux

Exercice 1 – décaler un tableau vers le bas

Écrire une procédure (ou une fonction) qui permet de redescendre tous les éléments d'un tableau (le dernier passant en premier).

Quelle est la complexité de l'algorithme ?

Ecrire une variante qui permet de remonter tous les éléments d'un tableau (le dernier passant en premier).

Exercice 2 – rechercher première occurrence

Écrire une procédure (ou une fonction) qui recherche la première occurrence d'une valeur dans un tableau d'entiers non triés avec doublons.

Quelle est la complexité de l'algorithme ?

Exercice 3 : recherche dichotomique

Écrire une procédure (ou une fonction) qui recherche une valeur dans un tableau trié sans doublon en utilisant la méthode de recherche dichotomique.

Quelle est la complexité de l'algorithme ?

Exercice 4 – suppression dune valeur

Écrire une procédure (ou une fonction) qui supprime une valeur dans un tableau trié sans doublon.

Quelle est la complexité de l'algorithme ?

Exercice 5 – ajout d'une valeur

Écrire une procédure (ou une fonction) qui ajoute une valeur dans un tableau trié sans doublon.

Quelle est la complexité de l'algorithme ?

Exercice 6 : tri à bulles

Ecrire l'algorithme de tri d'un tableau par la méthode du tri à bulles.

Le principe est d'inverser les couples de valeurs successives du tableau en fonction de l'ordre du tri, en parcourant le tableau d'un bout à l'autre et en répétant l'opération autant de fois que nécessaire.

Exemple avec le tableau : 6 10 5 8 12 4

On teste 6 et 10 et on ne fait rien

On teste 10 et 5 et on inverse : 6 5 10 8 12 4

On teste 10 et 8 et on inverse : 6 5 8 10 12 4

On teste 10 et 12 et on ne fait rien

On teste 12 et 4 et on inverse : 6 5 8 10 4 12

Le plus grand (12) est tout en bas.

Et on recommence :

On teste 6 et 5 et on inverse : 5 6 10 8 12 4

On teste 6 et 10 et on ne fait rien

Etc.

Quelle est la complexité de l'algorithme ?

Regarder dans le cours les résultats des tests selon les différents algorithmes.

Exercice 7 : matrice – création – tri par colonne ou par ligne

Ecrire un programme qui permette de créer une matrice de taille choisie (nombre de lignes et nombre de colonnes). On pourra créer une matrice avec des valeurs de 0 à N ou avec des valeurs aléatoires.

On veut ensuite pouvoir trier la matrice selon une colonne donnée et aussi selon une ligne donnée (on utilise la méthode du tri à bulles).

On veut un programme qui permette d'accéder à toutes les options.

Exercice 8 : Tours de Hanoi

- ***Description du jeu***

  Le jeu est constitué d'une plaquette de bois où sont plantées trois tiges : les tours.

  Sur ces tiges sont enfilées des disques de diamètres tous différents.

  Les seules règles du jeu sont que l'on ne peut déplacer qu'un seul disque à la fois, et qu'il est interdit de poser un disque sur un disque plus petit.

  Au début tous les disques sont sur la tige de gauche, et à la fin sur celle de droite.

- ***Méthode de résolution itérative***

  Déplacer le 1 à gauche (« faire le tour » et revenir sur la tour de droite si on est déjà complètement à gauche).

  Déplacer celui des deux qui n'est pas 1 et qu'on peut déplacer.

  Répéter les deux opérations jusqu'au déplacement complet de la tour.

  Subtilité : Si le nombre total de disques est impair, il faut déplacer le 1 à gauche. Mais si le nombre total de disques est pair, il faut déplacer le 1 à droite !

- ***Premier objectif***

  Ecrire un programme qui affiche les déplacements dans des tours de Hanoï.

  A chaque coup, on affichera le numéro du coup pour arriver au nombre de coups joués pour arriver au but.

  Dans un premier temps on fera un affichage très simplifié.

- ***Méthode***

  1) Bien comprendre le fonctionnement du jeu et le principe de résolution : faites-le « tourner » à la main. On peut tester le jeu en ligne, par exemple ici.
  2) Quelles données envisagez-vous d'utiliser pour traiter le problème ?
  3) Quel algorithme général pouvez-vous écrire qui traduise le principe de résolution et l'objectif, et qui utilise vos données ? L'algorithme général doit utiliser des fonctions
  4) Une fois l'algorithme général conçu, il faut écrire le détail de chaque fonction, éventuellement en créant d'autres fonctions.

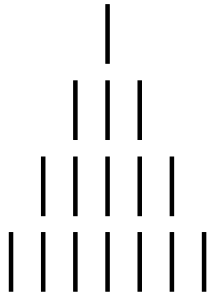
Exercice 9 : Jeu des allumettes ou jeu de Marienbad

- ***Test du jeu***

  Par exemple, ici.

- ***Description du jeu***

  On a 4 lignes d'allumettes avec 1, 3, 5 et 7 allumettes.



  Le but du jeu est de ne pas ramasser la dernière allumette : qui prend la dernière perd.

  Le jeu se joue à 2. A son tour, chaque joueur peut prendre autant d'allumettes qu'il veut mais sur une seule ligne.

- ***Comment gagner à tous les coups ?***

  D'abord, il ne faut pas commencer.

  Ensuite, il faut laisser le jeu dans une configuration particulière expliquée maintenant.

  Chaque ligne contient un nombre d'allumettes : 1, 3, 5 et 7 au départ. Il faut traduire ce nombre en binaire.

  Situation de départ :

Nombre d'allumettes	Nombre d'allumettes en binaire		
1	0	0	1
3	0	1	1
5	1	0	1
7	1	1	1
Total en base 10 de chaque colonne binaire :	2	2	4

  Le joueur 1 qui commence est dans une configuration telle que le total en base 10 de chaque colonne binaire est un nombre pair. C'est cette configuration qui fait que le joueur va perdre, si l'adversaire ne fait pas d'erreur.

  Quand le joueur 1 va joueur, quoi qu'il fasse, il laissera un ou plusieurs totaux impairs.

  Le joueur 2 devra jouer de telle sorte que chaque total soit à nouveau un nombre pair.

- ***Vérifier que vous avez compris comment gagner***

  Essayer de jouer en appliquant la méthode pour gagner à tous les coups pour vérifier que vous avez compris la méthode.

- ***Coder un programme qui permette de jouer contre l'ordinateur***

  Pour ça, vous devez d'abord réfléchir aux grandes lignes du programme : les grandes fonctions.

  Vous devez aussi réfléchir aux structures de données manipulées.

  Ensuite, pour chaque grande fonction identifier dans votre algorithme général, essayer d'écrire l'algorithme en le divisant en fonctions. C'est particulièrement vrai pour l'algorithme qui s'occupera de faire jouer l'ordinateur.

Exercice 10 : Bucket Sort

Principe du « bucket sort » :

Le « bucket sort » ordonne un tableau donné de N éléments. Il a besoin pour cela de tableaux auxiliaires. Le principe est de répartir les valeurs dans plusieurs tableaux puis de fusionner en commençant par les unités.

Voyons sur un exemple le fonctionnement du « bucket sort » :

- ***Soient les 11 nombres suivants à trier :***

112	97	156	253	744	514	266	634	800	1	333

- ***étape 1 : on range les nombres dans 10 sous tableaux (de T0 à T9) en fonction de leur unité :***

T0	T1	T2	T3	T4	T5	T6	T7	T8	T9
800	1	112	253	744		156	97		
			333	514		266			
				634					

Ensuite on remplace ces sous tableaux dans le tableau principal :

800	1	112	253	333	744	514	634	156	266	97

- ***étape 2 : ce tableau est à nouveau réparti mais cette fois-ci par rapport aux dizaines :***

T0	T1	T2	T3	T4	T5	T6	T7	T8	T9
800	112		333	744	253	266			97
1	514		634		156				

Ensuite on fusionne de nouveau et on obtient :

800	1	112	514	333	634	744	253	156	266	97

- ***Dernière étape : on répartit par rapport aux centaines :***

T0	T1	T2	T3	T4	T5	T6	T7	T8	T9
1	112	253	333		514	634	744	800	
97	156	266							

Ensuite on fusionne de nouveau et on obtient :

1	97	112	156	253	266	333	514	634	744	800

- ***Test des tris***

Créer un tableau de 2000 nombres < 1000.

Code python pour avoir un entier aléatoire entre 0 et 999

```
from random import *
randint(0, 999)
```

Mesurer le temps d'exécution avec un Bucket Sort et avec un tri à bulles.

Pour mesurer le temps d'exécution, deux méthodes :

Compter le nombres d'itérations dans les boucles

Mesurer le temps au début et à la fin (ça ne donne pas forcément, en première approche, une durée avec une unité connue, mais ça donne une base de comparaison).

```
import time
tmps1=time.clock()
# code
tmps2=time.clock()
print (tmps2-tmps1)
```

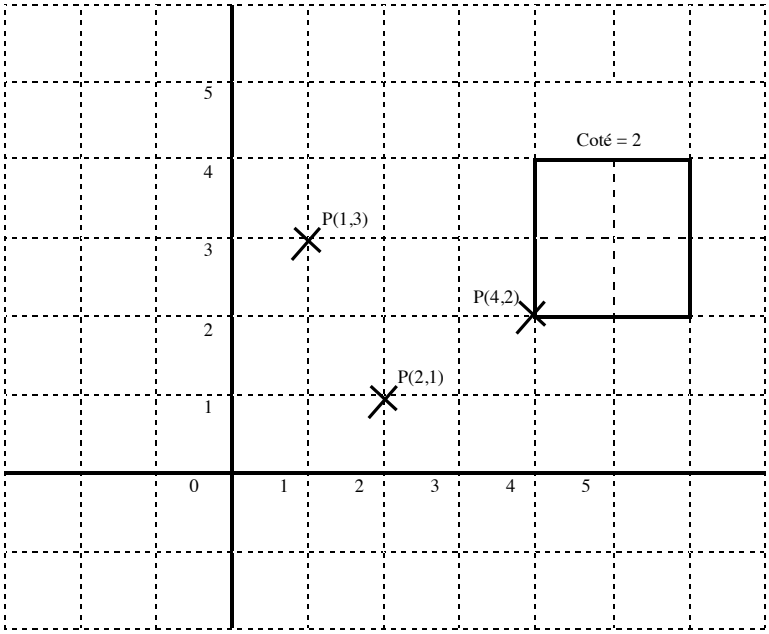
Type structuré

Exercice 1 – des points et des carrés

Un point est caractérisé par ses coordonnées x et y. Un carré à base horizontale est défini par les coordonnées de son point en bas à gauche et par son côté. Les coordonnées et le côté sont des réels.

On travaille sur un ensemble de carrés.

- Définir les structures qui permettent de gérer ce problème. Créer le carré du schéma ci-dessous.
- Ecrire une procédure qui calcule la surface d'un carré.
- Ecrire une procédure qui calcule les coordonnées du point en haut à droite d'un carré.
- Ecrire une procédure qui détermine si un carré est inclus dans un autre carré.
- Ecrire une procédure qui détermine quel est le plus grand nombre de carrés inclus dans un autre carré.



Exercice 2 – élève et classe d'élèves

Un élève est caractérisé par son nom, son prénom, sa date de naissance. Il y a 3 matières d'informatique : algo, C++ et SQL. Chaque matière donne lieu à 2 QCM et à 1 examen. Les QCM comptent pour 25%. L'examen pour 50%. On connaît les dates d'examen et de QCM. Chaque élève porte toutes les informations le concernant. On connaît les notes pour chaque examen, la note finale pour la matière et la moyenne des 3 matières.

- Définir la ou les structures de données permettant de gérer un élève.
- Définir la structure de donnée permettant de gérer une classe. On enregistre aussi la moyenne générale de la classe dans la structure de la classe.
- Ecrire une procédure ou une fonction qui permet de mettre à jour la note finale de chaque élève pour chaque matière et sa moyenne.
- Ecrire une procédure ou une fonction qui permet de calculer la moyenne générale de la classe.
- Ecrire une procédure ou une fonction qui affiche la liste des élèves avec leur moyenne générale triée par notes décroissantes.

Exercice 3 : tableau d'utilisateurs

1) On veut gérer des utilisateurs avec les caractéristiques suivantes :

Prénom et NOM (dans un seul champ), mail, motDePasse, annee de naissance (on peut utiliser un dictionnaire en python).

1) Créez une fonction newUser permettant de créer un utilisateur.

```
# creation d'un utilisateur
newUser( parametres à déterminer);
```

2) Créez une fonction qui permette d'afficher un utilisateur.

```
# affichage d'un utilisateur
printUnUser( parametres à déterminer);
```

3) Utilisez les deux fonctions pour créer un utilisateur et l'afficher.

4) A la place de l'année de naissance, on veut afficher l'âge. Adapter le programme. On se dote d'une fonction calculerAge(parametres à déterminer);

   Pour calculer l'âge, on fera une simple soustraction entre l'année en cours et l'année de naissance.

   En python, pour récupérer l'année du jour, on peut écrire :

```
from datetime import date
year=date.today().year
```

5) Créez une fonction qui permette de créer un tableau d'utilisateurs, vide dans un premier temps.

```
# création d'un tableau d'utilisateurs
créerLesUsers( parametres à déterminer );
```

6) Créez une fonction qui permette d'ajouter un utilisateur dans le tableau précédemment créé.

```
# création d'un tableau d'utilisateurs
ajouterDansLesUsers( parametres à déterminer );
```

7) Créez une fonction qui affiche les utilisateurs.

```
# affichage du tableau d'utitlisateur
printLesUsers ( parametres à déterminer );
```

8) Utilisez les fonctions pour créer un tableau de 5 utilisateurs et affichez-le.

9) Créez une fonction qui permette de trier par nom un tableau d'utilisateurs et testez la fonction.

```
# initialisation d'un tableau d'utilisateurs
triParNom( parametres à déterminer );
```

   On utilisera un tri à bulles. Rappel de la méthode de résolution la plus simple : on parcourt le tableau. Si une valeur est plus grande que la suivante, on les inverse les deux valeurs pour que la plus grande passe en dessous. On répète l'opération autant de fois qu'il y d'éléments dans le tableau.

   On peut ensuite réfléchir à optimiser l'algorithme.

10) Créez une fonction qui permette de trier par âge un tableau d'utilisateurs et testez la fonction.

```
# initialisation d'un tableau d'utilisateurs
triParAge( parametres à déterminer );
```

11) Créez une fonction qui permette de trier par n'importe quel critère un tableau d'utilisateurs et testez la fonction.

```
# initialisation d'un tableau d'utilisateurs
triParCritere( parametres à déterminer );
```

Exercice 4 : Select SQL

Ecrire un algorithme qui recherche les employés qui gagne plus de 2000 et qui ont été embauché après 2016.

On part d'une structure employe (NE, nom, prenom, fonction, salaire, dateEmbauche, ND)

Avec NE, numéro de l'employé et ND numéro de son département.

On s'appuiera sur le code de l'exercice précédent (tableau d'utilisateurs).

Exercice 5 : Jointure SQL

Ecrire un algorithme qui recherche les employés parisiens qui gagne plus de 2000.

On part d'une table d'employés avec une structure employe (NE, nom, prenom, fonction, salaire, dateEmbauche, ND)

Avec NE, numéro de l'employé et ND numéro de son département.

On a aussi une table de départements avec une structure departement(ND, nom, ville).

On s'appuiera sur le code de l'exercice précédent (tableau d'utilisateurs).

Chaîne de caractères

Exercice 1 : Palindrome

Ecrire une fonction qui détermine si un mot est un palindrome ou pas. Un palindrome est un mot qui se lit pareil à l'endroit et à l'envers.

Exercice 2 : Anagramme

Ecrire une fonction qui détermine si un texte est une anagramme d'un autre ou pas. Une anagramme est un texte constitué avec les même lettres qu'un autre. Par exemple : « argent » et « gérant » ou encore : « connaître » et « actionner ».

Avec des expressions : « La gravitation universelle » devient « Loi vitale régnant sur la vie » et aussi « Entreprise Monsanto » devient «poison très rémanent ». Plus : ici.

Exercice 3 : Nombre d'occurrences d'un mot dans un texte

Ecrire une fonction qui calcule le nombre d'occurrence d'un mot dans un texte, les séparateurs de mot étant définis dans une chaine de caractères : sep = « ;/, etc. »

Exercice 4 : Nombre d'occurrences de tous les mots dans un texte

Ecrire une fonction qui calcule le nombre d'occurrence de tous les mots dans un texte, les séparateurs de mot étant définis dans une chaine de caractères

Exercice 5 : Algorithme de César

- ***Cryptage, chiffrement, codage***

Le **cryptage** ou **chiffrement** est un procédé pour rendre la compréhension d'un document impossible sans avoir la **clé de déchiffrement**.

On parle aussi de codage à la place de cryptage ou chiffrement, mais le codage est une notion plus large.

- ***Algorithme de César***

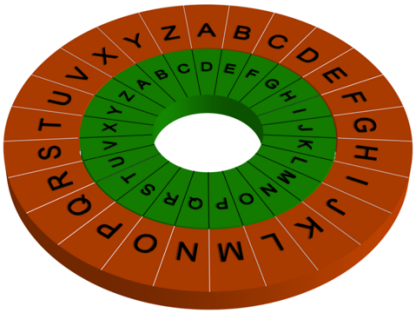
César utilisait une technique simple pour coder ses messages. Cette technique est connue sous le nom de « algorithme de César ». C'est un algorithme de cryptage. C'est un algorithme simple.

Le principe est de décaler l'alphabet dans le message crypté. **La valeur du décalage est la clé de cryptage**.

Ainsi, si la clé vaut 3, A devient D, B devient E, C devient F, …, X devient A, Y devient B, Z devient C.

« CESAR » est crypté par « FHVDU »

On peut représenter les choses ainsi :



L'alphabet rouge est celui du texte en clair (non crypté), l'alphabet vert est celui du texte crypté.

- ***Chiffrage***

On peut faire correspondre chaque lettre de l'alphabet à un entier. On définit une bijection « f » d'un ensemble de lettres vers un ensemble d'entiers avec l'image de chaque lettre.

f : {A, B, .., Y, Z} → {0, 1, …, 24, 25}

A → 0, B→ 1, …, Y→24, Z→25

Les lettres sont donc numérotées de 0 à 25 (A=0, B=1, …, Z=25).

Le **chiffrement** consiste donc à :

- Remplacer chaque lettre du texte par l'entier lui correspondant.
- Puis à utiliser la clé pour « chiffrer » la valeur de départ. Dans le cas de l'algorithme de César, il suffit d'ajouter la valeur de la clé (la valeur du décalage).
- Il reste à remplacer le nouveau nombre par la lettre qui lui correspond. A vaut 0. 0 chiffré devient 3. 3 vaut C.

➢ ***Ensemble des clés***

L'ensemble des clés, c'est l'ensemble des valeurs possibles pour la clé.

Dans l'algorithme de César, l'ensemble des clés = {0, 1, …, 24, 25}

Le cardinal de l'ensemble des clés donne la complexité de la clé.

Dans l'algorithme de César, la complexité de la clé vaut 26, ce qui est très peu.

➢ ***Décodage – Déchiffrement : craquer le code !!!***

Pour décoder un texte (pour le déchiffrer) il faut **connaître la valeur de la clé** (la valeur du décalage).

On peut aussi **tester toutes les clés possibles** : si le cardinal de l'ensemble des clés est petit, c'est une opération faisable.

De plus, dans le cas de l'algorithme de César, si on en connaît pas la valeur de d, on peut appliquer le principe suivant : **dans un texte, la lettre « e » est le plus souvent la plus représentée**. Ainsi, on peut trouver le décalage à partir du texte codé.

➢ ***Exercices « A la main »***

1. **Coder** « bonjour » avec une **clé codage de 6.**
2. **Décoder** «olssv dvysk » avec une **clé de codage de 7**
3. **Chercher « manuellement » à décoder** le texte suivant : « tew wm jegmpi uyi gipe pi gsheki hi giwev »
4. Combien y a-t-il de clés possibles pour l'algorithme de César ?

➢ ***Algorithmes à écrire***

1. Ecrivez une fonction qui permette de coder des textes non codés à partir d'une clé. On l'appellera « coderMessage() ». A vous de déterminer les paramètres et le code.
2. Ecrivez une fonction qui permette de décoder des textes codés à partir d'une clé.
3. Ecrivez une fonction de tests unitaires qui permettra de mettre en œuvre ces deux fonctions sur 1 ou 2 exemples.

4. On cherche maintenant à « craquer » le code avec le principe du « e » le plus représenté. Ecrire une fonction frequenceDesLettres () qui renvoie un tableau avec la fréquence de chaque lettre de l'alphabet dans le message.

5. Testez la fonction en affichant le tableau de fréquences des lettres du message codé et le tableau de fréquences des lettres du message décodé.
6. Une fonction cleDeDecodage() qui renvoie la clé de décodage probable du message (le décalage à appliquer pour décoder).
7. Testez la fonction pour vérifier que le décodage fonctionne.