1. Introduction

Toi le jeune, va apprendre à coder les algorithmes de ce livre, puis gagne des concours de programmation, fais-toi accepter dans des entretiens d'embauche, et augmente le produit intérieur brut avec tes manches retroussées.

À tort, les informaticiens sont encore souvent considérés comme les magiciens des temps modernes. Les ordinateurs se sont petit à petit introduits dans les entreprises, les maisons, les machines, devenant des moteurs importants du fonctionnement de notre monde. Et de nombreuses personnes utilisent ces machines sans les maîtriser, sans qu'elles puissent entièrement répondre à leurs besoins. Savoir programmer permet alors d'exploiter au mieux leurs possibilités pour résoudre des problèmes de manière efficace. Algorithmique et programmation sont devenus un bagage incontournable dans bien des métiers. Maîtriser ces techniques permet alors de proposer des solutions informatiques créatives et efficaces aux problèmes de tous les jours.

Cet ouvrage expose différentes techniques algorithmiques pour des problèmes classiques, décrit des situations de la vie où ces problèmes surgissent, et montre des implémentations simples écrites dans le langage Python. Implémenter correctement un algorithme n'est pas toujours facile, il y a bien des pièges à éviter et des bonnes techniques à appliquer pour garantir la complexité en temps annoncée. Les implémentations de ce livre sont agrémentées d'explications sur les détails importants à respecter.

Depuis des dizaines d'années, des concours de programmation sont organisés dans le monde entier à tous les niveaux, afin de promouvoir une culture algorithmique large. Les problèmes proposés dans ces concours sont souvent des variantes de problèmes algorithmiques classiques, sous la forme d'énigmes agaçantes à résoudre qui ne vous lâcheront plus.

1.1 Les concours de programmation

Dans un concours de programmation, les candidats doivent résoudre plusieurs problèmes dans un temps imparti. Un problème est souvent une variante d'un problème classique, comme ceux abordés dans ce livre, et enrobé d'une petite histoire. Les entrées des problèmes sont appelés *instances*. Une instance peut par exemple

être la matrice d'adjacence d'un graphe pour un problème de plus court chemin. En général, un petit exemple d'instance avec sa solution est donné. Le code source d'une solution peut être transmis au serveur par une interface web, où il est compilé puis testé contre des instances cachées au public. Pour certains problèmes, le code est appelé pour chaque instance, pour d'autres l'entrée débute par un entier indiquant le nombre d'instances qui figureront dans l'entrée. Le programme devra alors dans une boucle lire chaque instance, la résoudre et afficher le résultat. La soumission est acceptée si elle donne les bonnes réponses dans un temps limite de l'ordre de la seconde.

FIGURE 1.1 – Le logo du concours ACM montre joliment les étapes de la résolution d'un problème. Un ballon gonflé à l'hélium est distribué aux équipes pour chaque problème résolu.

Donner ici une liste de tous les concours de programmation et sites d'entraînement est simplement impossible et serait vite dépassé. Néanmoins, nous allons passer en revue les plus importants.

ACM/ICPC Le plus ancien des concours est organisé par l'association ACM (*Association for Computing Machinery*) depuis 1977. Ce concours, portant le nom de ICPC pour *International Collegiate Programming Contest*, est organisé sous la forme d'un tournoi. Le point de départ pour les Français est le concours régional SWERC (*South-West European Regional Contest*) où les deux meilleures équipes se qualifient pour la finale mondiale. La particularité de ce concours est que chaque équipe de 3 personnes n'a qu'un seul ordinateur à sa disposition. Elles ont 5 heures pour résoudre un nombre maximum de problèmes parmi les 10 proposés. Le premier critère de classement est le nombre de solutions soumises et acceptées (testées contre un jeu de tests inconnu). Le deuxième, la somme pour chaque problème du temps écoulé entre le début du concours et le moment de la soumission acceptée. Pour chaque soumission erronée, une pénalité de 20 minutes est ajoutée.

Il y a plusieurs théories sur la composition d'une bonne équipe. En général il faut au moins un bon codeur et un bon algorithmicien, et des spécialistes dans différents domaines tels les graphes, la programmation dynamique, etc. Et il faut bien s'entendre, même en situation de stress. Lors du concours, on peut apporter 25 pages de code imprimées en taille 8 points pour référence. On dispose également de la documentation en ligne de l'API de Java et de la bibliothèque standard de C++.

Google Code Jam Contrairement au concours ACM qui est limité aux étudiants jusqu'au master, le Google Code Jam est ouvert à tous. Ce concours annuel est plus récent et la participation est individuelle. Chaque problème vient en général avec un jeu de petites instances dont la résolution donne déjà quelques points et un jeu de grandes instances où il est vraiment important d'avoir trouvé

un algorithme avec une bonne complexité. On est informé de l'acceptation de sa solution pour les grandes instances seulement à la fin du concours. Mais son véritable point fort est qu'il est possible après coup de consulter les solutions soumises par tous les participants, ce qui est extrêmement instructif. Le concours *Facebook Hacker Cup* est de nature similaire.

Prologin L'association française Prologin organise chaque année un concours destiné aux étudiants de vingt ans et moins. Leur capacité à résoudre des problèmes d'algorithmique y est mise à l'épreuve au cours de trois étapes : une sélection en ligne, des épreuves régionales, et la finale. Cette dernière est une rencontre atypique de trente-six heures, au cours desquelles les participants doivent faire face à un sujet d'intelligence artificielle. Chaque candidat doit programmer un champion qui joue à un jeu dont les règles sont définies par les organisateurs, et à l'issue de l'épreuve, un tournoi confronte leurs champions et détermine le palmarès. Leur site web `prologin.org` comporte des annales très complètes, où l'on peut tester ses algorithmes.

France-ioi L'association France-ioi prépare chaque année des collégiens ou lycéens aux olympiades internationales d'informatique. Depuis 2012, ils organisent le concours Castor informatique, qui s'adresse aux jeunes de la 6e à la terminale (228 000 participants en 2014). Leur site web `france-ioi.org` héberge une quantité phénoménale de problèmes d'algorithmique (plus de 1 000).

Il existe également de nombreux concours de programmation organisés dans le but de sélectionner des candidats pour des offres d'emploi. Le site *TopCoder* par exemple comporte aussi des tutoriels algorithmiques, qui sont souvent bien écrits. Pour l'entraînement nous recommandons particulièrement *Codeforces*, un site très respecté dans la communauté de la programmation en compétition, car il propose des problèmes soignés et clairs.

1.1.1 Sites d'entraînement

Plusieurs sites web proposent des problèmes issus d'annales de concours, avec la possibilité de tester ses solutions pour s'entraîner. C'est notamment le cas de Google Code Jam et Prologin. Les annales des concours ACM sont collectées à différents endroits.

Juges en ligne traditionnels Les sites `uva.onlinejudge.org`, `icpcarchive.ecs.baylor.edu` et `livearchive.onlinejudge.org` contiennent entre autres beaucoup de problèmes issus du concours ACM/ICPC.

Juges en ligne chinois Il existe maintenant de nombreux sites d'entraînement en Chine comme `poj.org`, `acm.tju.edu.cn` et `acm.zju.edu.cn`. Ils ont une interface plus épurée que les juges traditionnels. Cependant, des pannes sporadiques ont été observées.

Sphere Online Judge Le site `spoj.com` a l'avantage d'accepter la soumission de solutions en beaucoup de langages dont Python.

Sur le site `tryalgo.org`, qui accompagne cet ouvrage, vous trouverez une sélection de problèmes qui reposent sur le contenu des différentes sections. Vous aurez

ainsi l'occasion de mettre en pratique les algorithmes décrits dans ce livre, en testant votre implémentation sur un juge en ligne.

Les langages utilisés pour les concours de programmation sont principalement C++ et Java. Le concours Google Code Jam accepte tous les langages — car la résolution est à faire localement — et le juge SPOJ accepte aussi le Python, comme mentionné ci-haut. Pour faire face aux différences entre les vitesses d'exécution dues au choix du langage, les juges en ligne varient en général le temps limite en fonction du langage utilisé. Mais cette adaptation n'est pas toujours faite avec soin, et il est souvent difficile de faire passer une solution correctement écrite en Python. Nous espérons que cette situation changera dans les années à venir. Certains juges aussi travaillent avec une ancienne version de Java, pour laquelle des classes pratiques comme *Scanner* par exemple ne sont pas disponibles.

1.1.2 Réponses des juges

Lorsque vous soumettez un code à un juge en ligne pour un problème, celui-ci l'évalue via un jeu de tests privé et vous renvoie une réponse particulièrement succincte. Les principaux codes de retour sont les suivants.

Accepted Votre programme a donné les bonnes réponses dans le temps imparti. Félicitations.

Presentation Error C'est presque un programme accepté, mais vous affichez des espaces ou retours de ligne en trop ou en moins. Ce message est rare.

Compilation Error La compilation de votre programme a produit des erreurs. Souvent en cliquant sur ce message, vous aurez la nature de cette erreur. Comparez la version du compilateur utilisé par le juge avec celle du vôtre.

Wrong Answer Relisez l'énoncé, un détail a dû vous échapper. Êtes-vous sûr d'avoir testé tous les cas limites ? Auriez-vous laissé des affichages de débogage dans votre code ?

Time Limit Exceeded Vous n'avez probablement pas implémenté l'algorithme le plus efficace pour ce problème, ou bien vous avez une boucle infinie quelque part. Vérifiez les invariants de boucle pour vous assurer de la terminaison. Générez une grande instance et testez en local la performance de votre code.

Runtime Error En général, il peut s'agir d'une division par zéro, d'un dépassement de limite de tableau, ou bien d'un *pop()* sur une pile vide. Mais d'autres situations peuvent générer ce message, comme l'utilisation de *assert* en Java, qui n'est souvent pas acceptée.

Le comportement taciturne des juges permet néanmoins d'extraire certaines informations sur les instances. Voici une astuce qui a été utilisée lors du concours ACM/ICPC/SWERC. Dans un problème concernant les graphes, l'énoncé précisait que l'entrée était constituée de graphes connexes. Une équipe a eu un doute et a écrit un test de connexité. Dans le cas positif le programme entrait dans une boucle infinie et dans le cas négatif, il faisait une division par zéro. Le code d'erreur généré par le juge (Time Limit Exceeded ou Runtime Error) a permis à l'équipe de détecter que certains graphes de l'entrée n'étaient pas connexes.

1.2 Notre choix : Python

Le langage de programmation Python a été choisi pour ce livre, pour sa lisibilité et simplicité d'utilisation. En industrie, il est parfois utilisé pour produire des prototypes de programmes. Python est également le langage retenu pour des projets importants comme SAGE par exemple dont les parties critiques sont alors écrites dans des langages plus rapides comme C++ ou C.

Donnons quelques précisions sur ce langage. En Python, il existe quatre types de base : les booléens, les entiers, les nombres à décimale flottante et les chaînes de caractères. Contrairement à la plupart des autres langages de programmation, les entiers ne sont pas limités à un nombre de bits fixé, mais utilisent une représentation à précision arbitraire.

Les principales structures avancées sont les dictionnaires, les listes et les tuplets. La différence entre les derniers est que les tuplets sont des données immuables et peuvent être utilisés comme clés dans un dictionnaire.

Des bonnes introductions à Python sont disponibles en ligne, par exemple sur *python.org*. David Eppstein a mis à disposition des implémentations en Python très instructives dans une bibliothèque nommée *Python Algorithms and Data Structures* (PADS).

Pour la rédaction du code de ce livre, nous avons suivi la norme PEP8, qui fournit des recommandations précises sur l'usage des blancs, du choix des noms de variables, etc. Nous suggérons au lecteur de suivre également ces indications.

Python 2 ou 3 ? La version 3 de Python est sortie en 2008, mais encore aujourd'hui de nombreux développements se font en Python 2, car de nombreuses bibliothèques n'ont pas été portées en Python 3. Malgré ce fait, nous avons choisi d'implémenter les algorithmes en Python 3. Les principaux changements qui affectent les codes proposés dans ce livre portent sur *print* et la division entre entiers. En Python 3, pour deux entiers a et b, l'expression a / b renvoie le nombre flottant qui résulte de la division tandis que $a // b$ en renvoie le quotient euclidien (la partie entière). Et contrairement à Python 2, où *print* est un mot clé, en Python 3 c'est une fonction, qu'il faut donc appeler avec des paramètres entre parenthèses.

Si votre programme a des problèmes de performance il peut être intéressant de l'exécuter avec l'interpréteur *pypy* ou *pypy3*, qui est un compilateur à la volée. Ceci veut dire que votre code Python est d'abord traduit dans un code machine, qui ensuite est exécuté nettement plus rapidement. L'inconvénient est que *pypy* est encore en cours de développement et que certaines parties de la bibliothèque Python ne sont pas encore accessibles.

Infini Python travaille en précision arbitraire et ne limite pas les entiers à un nombre de bits fixé. Donc il n'existe pas d'équivalent entier pour représenter $-\infty$ et $+\infty$. Pour les nombres flottants en revanche, on peut utiliser *float('-inf')* et *float('+inf')*.

Quelques conseils Une erreur souvent commise par des débutants en Python concerne la copie de tableaux. Dans l'exemple suivant, le tableau B est juste une référence vers A. Une modification de $B[0]$ modifie $A[0]$ également.

```
A = [1,2,3]
B = A
```

Pour que B soit une copie distincte de A, on peut utiliser la syntaxe suivante.

```
A = [1,2,3]
B = A[:]
```

La notation [:] permet de faire une copie d'une liste. On peut également faire par exemple une copie amputée du premier élément $A[1:]$, ou du dernier élément $A[:-1]$, ou en ordre inverse $A[::-1]$. Par exemple, le code suivant crée une matrice M dont toutes les lignes sont les mêmes, et la modification de $M[0][0]$ modifiera toute la première colonne de M.

```
M = [[0]*10]*10
```

On peut initialiser une matrice carrée correctement d'une des manières suivantes.

```
M1 = [[0]*10 for _ in range(10)]
M2 = [[0 for j in range(10)] for i in range(10)]
```

Une manipulation aisée de matrices est possible avec le module *numpy* mais nous avons choisi de ne pas en tirer profit dans cet ouvrage, afin d'obtenir un code générique qu'on peut facilement traduire en Java ou C++.

Une autre erreur typique concerne l'utilisation de la fonction *range*. Par exemple, le code suivant traite les éléments du tableau A entre les indices 0 et 9 inclus dans l'ordre.

```
for i in range(0, 10):  # 0 inclus, 10 exclu
   traite(A[i])
```

Pour traiter les éléments dans l'ordre décroissant, il ne suffit pas d'inverser les arguments. Car *range(10, 0, -1)* — le 3e argument indique le pas — est la liste des éléments de 10 (inclus) à 0 (exclu). Il faut donc écrire le traitement ainsi :

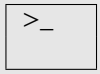
```
for i in range(9, -1, -1):  # 9 inclus, -1 exclu
   traite(A[i])
```

1.3 Entrées-sorties

1.3.1 Lire l'entrée standard

Pour la plupart des problèmes issus des concours de programmation, les données sont à lire depuis l'entrée standard, et la réponse à afficher sur la sortie standard. Si le fichier d'entrée s'appelle *test.in* par exemple et votre programme *prog.py*, vous pouvez diriger le contenu du fichier vers votre programme par la commande suivante, à lancer dans une console :

```
python prog.py < test.in
```



Généralement sous Mac OS X, une console peut être obtenue en tapant *Commande-Espace Terminal* et sous Windows, via *Démarrer → Exécuter → cmd*. Si vous êtes sous Linux, le raccourci clavier est *Alt-F2*, mais ça, vous le saviez déjà.

Si vous souhaitez enregistrer la sortie de votre programme dans un fichier intitulé *test.out*, tapez :

```
python prog.py < test.in > test.out
```

Petite astuce, si vous souhaitez afficher la sortie en même temps que vous l'enregistrez dans un fichier *test.out*, tapez la ligne suivante (la commande *tee* n'est pas présente par défaut sous Windows) :

```
python prog.py < test.in | tee test.out
```

L'entrée est à lire ligne par ligne via la commande *input()*, qui renvoie la prochaine ligne de l'entrée sous la forme d'une chaîne de caractères, en excluant le ou les caractères de fin de ligne[1]. Il existe dans le module *sys* une fonction similaire *stdin.readline()*, qui ne supprime pas les marqueurs de fin de ligne, et qui est 4 fois plus rapide selon nos expériences.

Si la ligne lue est censée contenir un entier, on convertira la chaîne avec la fonction *int* (si c'est un nombre à virgule, on utilisera *float*). Dans le cas où elle contient plusieurs entiers séparés par des espaces, on découpe d'abord la chaîne en différentes parties avec *split()*, qu'on convertit toutes en entiers avec la méthode *map*. Par exemple, dans le cas de deux entiers *hauteur* et *largeur* à lire sur la même ligne, séparés d'un espace, la commande suivante convient :

```
import sys

hauteur, largeur = map(int, sys.stdin.readline().split())
```

Si votre programme a des problèmes de performance lors de la lecture, vous pourrez gagner un facteur 2 selon nos expériences en lisant toute l'entrée avec un seul appel système. L'instruction suivante suppose que l'entrée n'est composée que d'entiers, éventuellement sur plusieurs lignes. Le paramètre 0 à la fonction *os.read* indique le flux d'entrée standard, et la constante M doit être une borne supérieure sur la taille du fichier. Par exemple si celui-ci contient 10^7 entiers chacun entre 0 et 10^9, alors comme chaque entier s'écrit sur au plus 10 caractères et qu'il y a au plus 2 caractères de séparation entre les entiers (`\n` et `\r`), on peut choisir $M = 12 \cdot 10^7$.

```
import os

entree = list(map(int, os.read(0, M).split()))
```

1. Suivant les systèmes d'exploitation, la fin de ligne est codée par les caractères *\r*, *\n* ou les deux, mais c'est sans importance lors d'une lecture avec *input()*. Notez aussi qu'en Python 2 le comportement de *input()* est différent, aussi il convient d'utiliser la fonction équivalente *raw_input()*

Exemple – lire trois matrices A, B, C et tester si $AB = C$ Dans cet exemple, l'entrée est de la forme suivante : une ligne contenant un unique entier n, suivie de $3n$ lignes contenant chacune n entiers séparés par des espaces. Ces lignes codent les valeurs contenues dans trois matrices $n \times n$ A, B, C données ligne par ligne. Le but est de tester si le produit de A par B est égal à la matrice C. Une approche directe aura une complexité $O(n^3)$, par la multiplication de matrices. Mais il existe une solution en $O(n^2)$ probabiliste, qui consiste à choisir au hasard un vecteur x et à tester si $A(Bx) = Cx$ (c'est le *test de Freivalds* [8]). Quelle est la probabilité que la procédure affiche égalité alors que $AB \neq C$? Si jamais les calculs se font modulo d, la probabilité d'erreur est au plus $1/d$. Cette probabilité d'erreur peut être rendue arbitrairement petite en répétant le test plusieurs fois. Le code suivant a une probabilité d'erreur majorée par 10^{-6}.

```
from random import randint
from sys import stdin


def readint():
    return int(stdin.readline())


def readarray(typ):
    return list(map(typ, stdin.readline().split()))


def readmatrix(n):
    M = []
    for _ in range(n):
        row = readarray(int)
        assert len(row) == n
        M.append(row)
    return M


def mult(M, v):
    n = len(M)
    return [sum(M[i][j] * v[j] for j in range(n)) for i in range(n)]


def freivalds(A, B, C):
    n = len(A)
    x = [randint(0, 1000000) for j in range(n)]
    return mult(A, mult(B, x)) == mult(C, x)


if __name__ == "__main__":
    n = readint()
    A = readmatrix(n)
    B = readmatrix(n)
    C = readmatrix(n)
    print(freivalds(A, B, C))
```