Introduction à l'algorithmique **Hiver 2024**

IFT2125-6001 Évaluation (7.5%)

Devoir 1 - Les nombres premiers, quel jeu d'enfant!

Consignes générales : À faire en équipe de 2 personnes maximum. Remettre un seul pdf et les fichiers de code python et C++. Indiquez vos noms et matricules sur tous les fichiers (pdf et code).

Votre code doit obligatoirement passer les tests fournis avec le devoir, sinon une note de 0 sera attribué aux exercices dont les tests de base ne sont pas réussis. D'autres tests seront rajoutés lors de la correction. Seulement les librairies standards de Python et C++ sont permises, sauf si mentionné autrement.

Il va de soit que le code doit être clair, bien indenté et bien commenté.

1 Notation asymptotique (25 points)

Question 1: (10 points) En utilisant les définitions de O, Ω et Θ vu en classe (sans utiliser les limites), montrer ou infirmez les énoncés suivants :

1. $2^n \in \Theta(3^n)$
2. $2^{n+b} \in \Theta(2^n)$ où $b \in \mathbb{N}^{\geq 2}$

Question 2: (10 points) En utilisant la règle de la limite, déterminez l'ordre relatif (O, Ω ou Θ) des fonctions suivantes :

1. $f(n) = 2^n$ et $g(n) = 3^n$
2. $f(n) = \frac{n}{\ln n}$ et $g(n) = \sqrt{n}$

Question 3: (5 points) Montrez que la fonction suivante est lisse :

$$f(n) = 2n^2 - 3n - 4$$

2 Nombres premiers - Code C++ (10 points)

Trouvez le n-ième nombre premier en C++. Votre algorithme devrait être efficace pour avoir tous les points.

Code
Exemple d'appel :

```
Compilation :
g++ -o nth_prime.exe nth_prime.cpp PrimeCalculator.cpp

Execution :
.\nth_prime.exe 100
```

Remise
Compléter les fichiers *PrimeCalculator.cpp* et *PrimeCalculator.h* fournis, et ne remettre **uniquement** que ces fichiers. Ne **pas** remettre le fichier *nth_prime.cpp*.

3 ACM - Code Python (20 points)

Calculez le poids d'un Arbre Couvrant Minimal (ACM) en Python.

Code
Votre code doit lire un fichier spécifié en premier argument qui contient n problèmes. Chaque problème contient un nombre de coordonnées m suivi de ces coordonnées. Les coordonnées sont toutes sur un plan euclédien. Toutes les arêtes entre chaque paire de sommets sont possibles. Le poid d'une arête est la distance euclédienne entre les deux coordonnées. La réponse retournée doit être seulement le poids de l'ACM. Écrire les réponses des n problèmes, ligne par ligne, dans un fichier spécifié en second argument.

Exemple d'appel :

```
python3 acm.py input.txt output.txt
```

Un fichier *test_acm.py* et des fichiers d'entrées vous sont fournis pour vous aider à tester votre code.

Exemple d'appel :

```
python3 test_acm.py
```

Remise
Compléter le fichier *acm.py* fournis, et ne remettre **uniquement** que ce fichier. Ne **pas** remettre le fichier *test_acm.py*, ou n'importe quel autre fichier de test.

4 Jeu Dobble - Code Python (45 points)

Il faut recréer le jeu de carte Dobble (ou Spot It!). C'est un jeu de vitesse et d'observation.

Le jeu se base sur le concept des plans projectifs finis en mathématique. Chaque carte contient un certain nombre de symboles. Chaque paire de carte a toujours un et un seul symbole en commun.

On dit que l'ordre d'un jeu est n. Le nombre de symboles par carte est $n+1$. Le nombre total de symboles et aussi de cartes dans le jeu est $n^2 + n + 1$. Pour ce devoir, nous allons nous intéresser seulement aux ordres qui sont des nombres premiers. Exemple :

- ordre = 2, symboles par carte = 3, total de symboles et de cartes dans le jeu= 7
- ordre = 3, symboles par carte = 4, total de symboles et de cartes dans le jeu= 13
- ordre = 5, symboles par carte = 6, total de symboles et de cartes dans le jeu= 31
- ordre = 7, symboles par carte = 8, total de symboles et de cartes dans le jeu= 57
- etc.

Explications de l'algorithme de génération des cartes

Pour générer correctement les cartes pour un jeu d'ordre n, on peut construire un tableau $n \times n$ de cartes vides et un second tableau $(n+1) \times 1$ de cartes vides qui va représenter "l'horizon" du plan projectif. Pour chaque direction possible et pour chaque carte de début, on va assigner un symbole commun à toutes les cartes sur cette ligne et pour la carte à l'horizon qui représente cette direction. Finalement, nous devons assigner un symbole pour toutes les cartes sur l'horizon.

Voir l'exemple illustré en annexe.
Référence wiki pour les intéressé.e.s : Projective_plane#Finite_projective_planes

Code
Trois classes seront utilisées :

- Generator (dobble_generator.py) : Reçoit l'ordre du jeu et construit les cartes symboliquement. Le résultat est écrit dans le fichier "cartes.txt" où chaque ligne représente une carte. Les symboles sont simplement écrit comme une liste de nombres séparés par des espaces. Un mélange aléatoire des symboles d'une carte sera apprécié.
- Verificator (dobble_verificator.py) : Reçoit un fichier contenant les cartes puis vérifie la validité et l'optimalité du jeu. Avec un ordre n, il devrait y avoir $n+1$ symboles par carte et n^2+n+1 cartes et symboles au total pour que le jeu soit considéré optimal. Le nombre de symboles par cartes devrait être le même pour toutes les cartes pour que le jeu soit valide. Chaque paire de cartes partagent toujours un et un seul symbole en commun pour que le ju soit valide. La vérification retourne 2 si le jeu n'est pas valide, 1 si le jeu est valide mais n'est pas optimal et 0 si le jeu est valide et optimal.
- Creator (dobble_creator.py) : Reçoit un fichier "cartes.txt" contenant les cartes puis créer les cartes visuelles du jeu. Les images sont prises du dossier "images" : "1.png2, "2.png", "3.png", ... "⟨N⟩.png". Les cartes visuelles sont enregistrées dans le dossier "results" : "card1.jpg", "card2.jpg", "card3.jpg", ... "card⟨N⟩.jpg". Sur chaque carte, les images correspondantes aux symboles de la carte sont placées. Les rotations d'images seront appréciées. Des images tests seront fournies.

La librairie PIL sera utilisée pour faire les manipulations d'images :

```
from PIL import Image
```

Exemple d'appel :

```
python3 cartes_dobble.py 5
```

Remise
Compléter les fichiers *dobble_generator.py*, *dobble_verificator.py* et *dobble_creator.py* fournis, et ne remettre **uniquement** que ces fichiers. Ne **pas** remettre le fichier *cartes_dobble.py*, ou n'importe quel autre fichier de test/image.

Bonus 10%
Nous allons accorder un bonus maximal de 10% (pour le devoir) aux équipes qui vont présenter en personne une version imprimée et personalisée de leur jeu de cartes. Le jeu doit être produit par votre propre algorithme et les images doivent être différentes des images fournie en exemple avec le devoir puis différentes des autres équipes. Le jeu doit être minimalement d'ordre 5 et doit être optimal (6 symboles par carte, 31 symboles au total, 31 cartes au total). Les cartes doivent être cartonnées, donc pas de papier standard d'imprimante. Vous pourrez l'offrir en cadeau ensuite à vos ami.e.s ou à votre famille.

Vous pouvez utilisez les services d'impression du SIUM à l'Université de Montréal ou d'une entreprise. Des détails pour les intéressé.e.s seront fournis plus tard.

Annexe