Mathématiques pour l'algorithmique et la programmation
L'écrivain numérique

— La version finale du projet doit être rendue sur gitlab le **vendredi 28 avril à midi**. La soutenance aura lieu le vendredi 05 mai.
— Le projet s'effectue en groupe de 4, puisqu'il contient 4 parties. Chaque partie est idéalement gérée par un des membres du groupe.
— **Il est interdit d'avoir d'avoir recours à une aide extérieure aux membres de votre groupe**, humaine ou IA.
— **Tout soupçon de fraude sera sanctionné par un** 0 **ou un conseil de discipline pour toutes les parties**. Faites donc attention à ne pas partager votre code et à le protéger des curieux. Les justifications du type "On en a discuté ensemble" ou "On a travaillé ensemble" ne sont pas valables hors de votre groupe de travail.
— **Un programme qui ne compile pas n'est pas corrigé et vaut automatiquement** 0. Il vaut mieux faire "moins de choses", tant que cela fonctionne.
— Vous serez noté sur la partie que vous avez faite, il est donc possible que vous n'ayez pas la même note que vos autres binômes.
— La lisibilité du code comptera dans la notation.
— La gestion de la mémoire comptera dans la notation : toute allocation mémoire doit être libérée avant la fin du programme.
— Nous devons pouvoir tester rapidement les fonctionnalités de votre programme. Par les tests unitaires et par le programme principal.
— Au début de chaque fichier, vous devez écrire le nom de la personne (ou des personnes) ayant travaillé dessus. **Il est interdit de rendre un devoir et de dire "on a tout fait ensemble"**. Vous devez répartir les tâches.
— **Pour chaque partie, il faut impérativement terminer les tâches d'un niveau avant d'en commencer un autre.**
— Toutes les structures nécessaires au développement du projet ne sont pas nécessairement spécifiées dans ce sujet. Il est donc normal d'être amené à développer des modules complémentaires afin de compléter ceux déjà spécifiés.

1 Partie 1 : les chaînes de Markov

1.1 Niveau 1

Écrire la déclaration d'une liste de fonction permettant de manipuler une chaîne de Markov, ainsi que les contrats associés dans un fichier `markov.h`. Ces fonction manipulent un type `markov_t *` que vous définirez par la suite.

On veut pouvoir :

— créer et détruire une chaîne de Markov. On passera en paramètre de la fonction de création une graîne du générateur aléatoire.

— ajouter des états dans la chaîne.

— dans la phase d'apprentissage, augmenter le poids entre deux états.

— après la phase d'apprentissage, utiliser les poids stockées pour calculer la probabilité de passer d'une état à un autre. Soit E l'ensemble des états de la chaînes de Markov. Soit la fonction $poids : E \times E \to \mathbb{N}$ qui prend en entrée un couple d'états (e_1, e_2) et renvoie le poids de e_1 vers e_2. Soit la fonction $\pi : E \times E \to [0, 1]$ qui prend en entrée un couple d'états (e_1, e_2) et renvoie la probabilité de passer e_1 à e_2. On a

$$\pi(e_1, e_2) = \frac{poids(e_1, e_2)}{\sum_{e \in E} poids(e_1, e)}$$

— se déplacer aléatoirement d'un état de départ à un état aléatoirement choisi parmi ses voisins, en fonction des probabilités calculées par la fonction précédente.

1.2 Niveau 2

Écrire une liste de fonction permettant de manipuler une chaîne de Markov, ainsi que les contrats associés dans un fichier `markov.h`. Attention, s'il y a peu de fonction à programmer, il faudra faire particulièrement attention à l'efficacité en temps et en espace de votre structure et de vos fonctions.

2 Partie 2 : la table de hachage

2.1 Niveau 1

Écrire la déclaration d'une liste de fonction permettant de manipuler une table de hachage, ainsi que les contrats associés dans un fichier `hachage.h`. Ces fonction manipulent un type `hachage_t *` que vous définirez par la suite. On veut pouvoir :

— créer et détruire une table de hachage.

— avoir une fonction de hachage efficace pour les mots définis sur $\{a, \ldots, z\}$.

— ajouter des mots dans la table.

— connaître la position d'un mot dans la table de hachage.

— associer chaque mot à un identifiant unique (un entier). Il s'agit d'une bijection entre l'ensemble des mots stockés dans la table de hachage et l'ensemble des états de la chaîne de Markov.
— retrouver un mot à partir de son identifiant unique.

Attention, s'il y a peu de fonction à programmer il faudra faire particulièrement attention à leur efficacité en temps et en espace.

2.2 Niveau 2

— Écrire une liste de fonctions permettant de manipuler une table de hachage, ainsi que les contrats associés dans un fichier `hachage.c`. Votre table devra minimiser les collisions et être capable de gérer le fait qu'elles puissent survenir.

3 Partie 3 : la gestion des répertoires et fichiers

Vous écrirez le contrat de toutes vos fonctions en prenant bien garde à gérer tous les cas qui peuvent être problématiques.

3.1 Niveau 1

La déclaration des fonctions suivantes se fera dans le fichier *sauvegarde.h* et leur définition dans *sauvegarde.c*.

— Écrire une fonction prenant en entrée un texte et un chemin (relatif ou absolu) qui crée un fichier au chemin donné, contenant le texte passé en entrée.
— Écrire une fonction qui prend en entrée un chemin, ouvre un fichier en mode "ajout" ou concaténation et renvoie le descripteur de fichier.
— Écrire une fonction qui prend en entrée un descripteur de fichier ouvert en mode "ajout" ou concaténation et une chaîne de caractère. La fonction écrit la chaîne de caractère dans le fichier.

3.2 Niveau 2

La déclaration des deux premières fonctions se fera dans le fichier *repertoire.h* et leur définition dans *repertoire.c*.

1. Écrire un itérateur des mots dans un fichier texte. Les mots renvoyés sont tous en minuscule. La ponctuation et les caractères spéciaux doivent être retirés. Lorsqu'un mot est renvoyé, l'itérateur indique aussi s'il s'agit ou non d'une fin de phrase.
2. Écrire une fonction prennant en entrée le chemin d'un répertoire et renvoie une chaîne de Markov initialisée grace aux mots trouvés dans la chaîne.
   — pour chaque fichier dans ce répertoire,
   — pour chaque phrase dans ce fichier,

— pour chaque mot dans cette phrase,
  (a) si le mot n'est pas dans la table de hachage, on l'y ajoute.
  (b) on ajoute incrémente le poids entre le mot est celui qui le précède (le début de phrase est le mot de clé 0).
— Une fois que tous les fichiers sont lus, on finalise l'initialisation de la chaîne de Markov en transformant les poids en probabilités.

3. Écrire un *programme* (main.c) qui initialise une chaîne de Markov en supposant toutes les fonctions nécessaires ont déjà été programmées. La chaîne sera initialisée grâce aux fichiers situés dans un répertoire dont le chemin est indiqué à l'exécution, comme paramètre du programme. Si l'utilisateur passe en argument une graine de générateur aléatoire, on l'utilise pour initialiser la chaîne de Markov, sinon, on utilise l'heure. Le programme va ensuite générer un texte aléatoire et le sauvegarder dans un nouveau fichier. Attention : vous devrez vous servir au maximum des fonctions à votre disposition. N'hésitez pas à découper ce programme en sous-fonctions, pour plus de lisibilité, si besoin est.

4 Partie 4 : Les tests unitaires

Votre partie consiste à vérifier que les parties qui ont été développées par les autres fonctionnent correctement. Si vous jugez que leurs contrats ne sont pas assez exigeants, vous pouvez/devez leur demander de les réécrire.

4.1 Niveau 1

Écrire un ensemble de tests unitaires pour les fonctions de manipulation de la chaîne de Markov et de la table de hachage

4.2 Niveau 2

— Écrire un ensemble de tests unitaires pour les fonctions de manipulation des répertoires et des fichiers.
— Trouver un moyen de tester si le programme final libère bien tout l'espace mémoire alloué par le programme. Faire en sorte que git ne valide pas les tests unitaires si ce n'est pas le cas (voir les fichiers Makefile et .gitlab-ci : vous ne devrez rien retirer de ces fichiers mais vous pouvez y ajouter des lignes).

5 Précisions inter-rôle

— Comme vous pouvez le remarquer dans ce sujet, il semble parfois nécessaire que d'autres personnes aient avancées dans leur partie, afin de pouvoir finaliser et tester la sienne. Toutefois, dans le cas où un membre du projet ne ferait pas sa partie, cela ne vous empêche en AUCUN CAS d'écrire la

vôtre, en supposant que les fonctions dont vous avez besoin ont déjà été écrites.

— La remarque précédente ne vaut pas pour la partie 4. En tant que personne responsable des tests unitaires, vous êtes responsables du bon déroulement du projet. Il faut donc demander à vos camarades d'écrire à minima leurs contrats, afin d'avancer sur votre partie. Les contrats doivent être écrits AVANT que les fonctions soient programmées. Si vous avez le sentiment que les contrats ne sont pas suffisant, vous pouvez rajouter des tests que le contrat ne demande pas. L'idéal étant d'en discuter entre vous.

— La table que hachage peut être stockée dans la chaîne de Markov, ou en dehors. Les deux sont possibles et valables, c'est un choix de développement que vous devrez faire.