**Structures de données et algorithmes**

**Travail #2**

**Par**

Dylan Sicard-Smith

Firaas Esso-Ninam Ewetola

Miriam Davydov

Scott LeClair

Yannick Poirier

**Présenté à**

Adam Joly

28 avril 2023

**Table des matières**

[**1. Répartition**](#_m6018ijzgzl0) **des tâches 1**

[**2.**](#_ql7w3qg0llwb) **Guide utilisateur 2**

2.1 Source des noms 2

[2.2 Générer aléatoirement des noms (opt](#_xyf4vlvh8sio)ion 1) 4

[2.3 Ajout](#_gthxc31k24fo) de noms et réinitialisation (option 2) 5

2.4 Ajout d’une banque de noms (option 3) 7

2.5 Afficher la matrice de probabilité (option 4) 8

2.6 Quitter (option 5) 9

**3**[**. Stratégies**](#_ohfzhwe79j76) **conceptuelles et algorithmiques principales 9**

[3.1](#_rzl0qystosj4) La classe « App » 9

[3.2](#_rnwz50m34emq) La classe « Interval » 14

[3.3](#_xyf4vlvh8sio) La classe « MarkovNameGenerator » 16

**4. Commentaires 24**

1. **Répartition des tâches**

Notre façon de communiquer était par écrit, par appel et par partage d’écran sur un groupe Discord créé pour le projet. Nous avons également créé un espace de travail sur GitHub afin que tout le monde puisse avoir accès au code et à l’avancement de celui-ci.

**Dylan Sicard-Smith :** Codage de l’algorithme qui génère un nom aléatoirement à l’aide de la matrice des intervalles des probabilités. Codage de l’algorithme permettant d’insérer une banque de nom. Révision, test et ajustement du programme.

**Firaas Esso-Ninam Ewetola :** Application de commentaires sur le code et la forme, révision, test et ajustement du programme.

**Miriam Davydov :** Codage des fonctions principales et des algorithmes principaux (compteur des occurrences des lettres uniques pour la liste de nom, statistique de ces occurrences et création des intervalles des probabilités, …) révision, test et ajustement du programme.

**Scott LeClair :** Révision, test et ajustement du programme.

**Yannick Poirier :** Rédaction du présent rapport, révision, test et ajustement du programme.

1. **Guide utilisateur**

Avant tout, l’utilisateur est présenté au menu suivant lorsqu’il exécute le programme et il y a cinq options qui lui sont offertes. Chacune d’entre-elles vont être aborder plus en détail dans les sous-sections suivantes.

Une image contenant texte, Appareils électroniques, capture d’écran, logiciel

Description générée automatiquement

* 1. **Source des noms**

Il est possible pour l’utilisateur d’insérer sa propre liste de nom dans le programme, comme on va le voir plus en détail dans la section 3. Par défaut, nous avons utilisé une liste de nom enregistrer sous format texte. Cette liste contient un total de 100 nom différents et variés. Elle peut être retrouver sous le nom « names.txt ».

Une image contenant texte, capture d’écran, Police

Description générée automatiquement Une image contenant texte, capture d’écran, Police

Description générée automatiquement Une image contenant texte, Police, capture d’écran, typographie

Description générée automatiquement

* 1. **Générer aléatoirement des noms (option 1)**

Lorsque l’utilisateur sélectionne l’option 1, le programme va générer aléatoirement des noms selon la quantité qui a été demander. Il est donc possible de générer un seul nom ou bien des centaines. Aussi, le générateur se base sur la dernière liste de nom qui as été mise à jour dans le programme. Dans l’exemple ci-dessous, j’ai demandé au programme de me générer 20 noms selon la liste « names.txt ».

Une image contenant texte, capture d’écran

Description générée automatiquement

* 1. **Ajout d’un nom et réinitialisation (option 2)**

Si l’utilisateur sélectionne l’option 2, le programme va lui permettre d’ajouter un nom dans la liste de nom et ainsi réinitialisé la liste de nom précédente qui était stocker. Dans l’exemple ci-dessous, j’ai ajouté le nom « Gorgio ». Le programme confirme que l’action a été un succès.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

* 1. **Ajout d’une banque de nom (option 3)**

L’option 3 permet à l’utilisateur d’insérer un fichier contenant sa propre banque de nom dans le programme. Ainsi, le fichier choisis va réinitialiser le générateur à partir de cette nouvelle liste. Si le nom de fichier est inexistant dans le système de l’utilisateur, le programme envoie une erreur.

**Une image contenant texte, capture d’écran, Police

Description générée automatiquement**

**Une image contenant texte, capture d’écran, Police

Description générée automatiquement**

* 1. **Afficher la matrice de probabilité (option 4)**

Lorsque l’utilisateur sélectionne l’option 4, la matrice de probabilité actuelle est affichée à la console.

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

Une image contenant texte, capture d’écran, ligne

Description générée automatiquement

* 1. **Quitter (option 5)**

Permet à l’utilisateur de mettre fin au programme. On confirme à la console que le programme c’est correctement terminer.

Une image contenant texte, Appareils électroniques, capture d’écran, logiciel

Description générée automatiquement

1. [**Stratégies**](#_ohfzhwe79j76) **conceptuelles et algorithmiques principales**

**3.1 La classe « App »**

Cette classe, « App » permet de créer un générateur de noms aléatoires à partir d’un fichier de noms qui est donnée en entrée, à l’aide d’une matrice de probabilité. La méthode main, principale de la classe, s’exécute lorsque le programme est lancé. Elle contient la boucle du menu qui permet à l’utilisateur de choisir différentes options, qui ont déjà été détailler dans le guide utilisateur dans la section 2.

À l’intérieur de la méthode main, la variable « names » est initialisée à partir du fichier « names.txt » en appelant la méthode « readNames() ». Ensuite, un objet « MarkovNameGenerator() » est initialisé à partir de cette liste de noms.

La boucle « do-while » permet de répéter le menu tant que l’utilisateur ne choisit pas l’option «Quitter». Un switch-case est alors utilisé pour exécuter les différentes actions offertes dans le menu.

Dans la première case, on demande à l’utilisateur d’entrer le nombre de noms aléatoires qu’il souhaite générer. Un message d’erreur est affiché si le choix est invalide. D’une autre part, si le choix est valide, la méthode « generateRandomName() » de l’objet « MarkovNameGenerator() » est appelée pour générer aléatoirement le nombre de noms spécifié.

Dans la deuxième case, l’utilisateur est invité à inscrire un nom qu’il veut ajouter à la liste de noms. Ce nom est ajouté à la liste et un nouvel objet « MarkovNameGenerator() » est créé à partir de cette liste mise à jour.

Dans la troisième case, l’utilisateur doit préciser le nom d’un fichier contenant une liste de noms. Assumant que le fichier existe, les noms sont lus à partir du fichier et un nouvel objet « MarkovNameGenerator() » est créé à partir de cette liste de noms.

Dans la quatrième case, la méthode « showMatrix() » de l’objet « MarkovNameGenerator() » est appelée pour afficher la matrice de probabilité.

La cinquième et dernière case met fin au programme.

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Afin que le programme soit apte à lire un fichier externe, il y a deux méthodes qui sont utilisés. La méthode « readNames() » sert à lire le contenu d’un fichier de noms et le stocker sont contenue dans une liste d’objets String. La méthode « fileExists() » est utilisée pour vérifier si un fichier avec un tel nom existe dans le système de fichiers de l’utilisateur. Si un tel fichier existe, la méthode retourne vrai, dans le cas contraire la méthode retourne faux.

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Les méthodes « getString() » et « getInt() » sont utilisées respectivement pour lire l’entrée utilisateur à partir de la console et renvoyer une chaîne de caractères ou bien un entier.

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

**3.2 La classe « Interval »**

Cette classe nommée « Interval » contient deux variables « max » et « min » qui représentent respectivement la borne supérieure et la borne inférieure de l’intervalle. En gros, cette classe représente un intervalle de nombres réels.

Il y a d’abord un constructeur de la classe qui prend en entrée deux paramètres : « min » et « max ». Ensuite, le début de la boucle est initié afin de vérifier que les valeurs des bornes sont différentes. Si les bornes (min et max) sont égales, cela veut donc dire que l’intervalle est vide, et signifie que les valeurs min et max sont toutes deux initialisées à 0.0 (donc, la transition est impossible). Sinon, les attributs « min » et « max » prennent les valeurs passées en paramètres, où « min » est inclus et « max » est exclu.

La classe possède une méthode « toString() » afin de permettre la conversion de l’intervalle en une chaîne de caractères qui représente l’intervalle à la console sous la forme souhaité (« [X; X] »). À l’aide de la méthode « Math.round() », les valeurs « min » et « max » sont arrondies à deux décimales en multipliant par 100, puis en divisant par 100. Cela permet d’obtenir une représentation plus lisible de l’intervalle.

De surcroît, si la borne supérieure est supérieure à 0, l’intervalle est considéré comme non vide et la chaîne de caractères qui est retourné va contenir les bornes de l’intervalle généré. En revanche, si la borne supérieure est inférieure ou égale à 0, cela signifie que l’intervalle est vide et la chaîne de caractères qui est retourné va simplement contenir un tiret (« - ») afin d’indiquer que la transition n’est pas possible.

En résumé, cette classe permet de créer des objets pour représenter des intervalles contenant des nombres réels et aussi d’effectuer des opérations sur ces intervalles, comme l’affichage à la console sous la forme souhaitée.

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

**3.3 La classe « MarkovNameGenerator »**

Cette classe est en fait la classe qui permet d’implémenter le générateur de nom de Markov, une partie essentielle du programme. Afin que cela soit possible, la classe possède les attributs suivants : « TOTALCHAR » qui représente la fin d’un nom à l’aide d’un caractère spécial, « names » qui est une liste pour stocker les noms, « headers » qui est une liste de caractères unique extraits des noms de base, puis finalement « matrix » qui contiendra la matrice de transition pour le modèle de Markov. Cette matrice est en fait une table de hachage imbriquée, où les clés sont des caractères et les valeurs sont des tables de hachage de caractères et d’objets « Interval ».

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Le constructeur permet de créer une nouvelle instance de la classe « MarkovNameGenerator() » en utilisant une liste de noms fournie en entrée par l’utilisateur. Dans un premier temps, il affecte la valeur de la liste de noms fournie en entrée a « names ». Ainsi, cela nous permet d’utiliser cette liste de noms pour entraîner le modèle de Markov.

Dans un deuxième temps, la « matrix » appelle la méthode « generateMatrixWithInterval() » qui retourne une matrice de transition pour le modèle de Markov basée sur les noms fournis en entrée. Cette matrice est alors stockée dans « matrix ».

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

La première méthode de la classe, « extractUniqueCharacters() » est utilisée pour extraire les caractères uniques des noms de base dans la banque de noms. Tout d’abord, une liste « uniqueCharacters » est initialisée avec le caractère d’espace comme premier élément. Cet espace représente l’absence de caractère et son utilité est de s’assurer que chaque nom commence par un caractère. Ensuite, à l’aide d’une boucle, la méthode parcours chaque nom de la liste de noms et convertit chaque nom en minuscules à l’aide de la méthode « toLowerCase() » afin que les caractères soient insensibles à la case. Aussi, la méthode utilise une autre boucle pour vérifier que chacun des caractères dans tous les noms sont des lettres de l’alphabet (« name.toCharArray() ») et également si la liste (« uniqueCharacters ») ne contient pas déjà ce caractère. Si c’est le cas, le caractère est ajouté à la liste « uniqueCharacters ». Finalement, la méthode retourne la liste « uniqueCharacters » qui contient tous les caractères uniques présents dans la liste de noms fournie, à l’exception de l’espace qui a été ajouté comme premier élément de la liste.

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

La deuxième méthode de la classe, « generateMatrixWithCount() » génère une matrice qui stocke le nombre d’occurrences de chaque caractère dans les noms de base. En premier lieu, la fonction « generateMatrixWithCount() » commence par appeler la fonction « extractUniqueCharacters() », qui va retourner une liste contenant les caractères uniques présents dans les noms. Cette liste de caractères est stockée dans la variable « keys ». La fonction « generateMatrixWithCount() » initialise également une matrice de transition baptiser « matrix ». Cette matrice est en fait une carte de hachage de caractères à une autre carte de hachage de caractères avec des entiers. Les clés de la première carte de hachage sont les caractères de « keys ». Les clés de la deuxième carte de hachage sont également les caractères de « keys », et les valeurs sont initialement toutes à 0. La matrice est initialisée de sorte que toutes les transitions possibles entre les caractères soient représentées, même si certaines transitions sont impossibles.

Ensuite, pour chaque nom dans la liste « names », la fonction utilise une boucle qui parcourt chaque caractère dans le nom et calcule le nombre de fois que chaque paire de caractères se présente dans les noms. Pour accomplir cela, la fonction utilise une matrice représentée sous forme de « HashMap » imbriquée. La matrice est initialisée avec des compteurs à zéro et ce, pour chaque paire de caractères possibles, comprenant même le caractère spécial représentant le total de caractères (« TOTALCHAR »). La boucle principale itère à travers chaque nom dans la liste des noms. Pour chaque nom, une boucle imbriquée itère à travers chaque caractère dans le nom, en commençant par un caractère vide. En vue de chaque paire de caractères consécutifs, le compteur de cette paire de caractères est incrémenté dans la matrice en utilisant la méthode « get() » de la « HashMap » pour accéder à l’entrée de la matrice correspondant à la paire de caractères ciblé. Ainsi, pour chacun des caractères de la paire de caractères en cours, le code incrémente le compteur total pour le caractère précédent. Enfin, le caractère précédent est mis à jour pour être le caractère actuel, et la boucle continue avec le caractère suivant dans le nom. Lorsque la boucle principale est complétée, la matrice qui est maintenant mise à jour est retournée en résultat.

Une image contenant texte, capture d’écran, logiciel, affichage

Description générée automatiquement

La troisième méthode de la classe, « generateMatrixWithInterval() » utilise en premier lieux la matrice de comptage créée par « generateMatrixWithCount() » pour générer une matrice qui stocke les intervalles de probabilité de transition entre deux caractères. Ensuite, la méthode initialise une nouvelle « HashMap » également nommé « matrix » pour stocker la matrice de probabilités avec des intervalles. Pour pouvoir réaliser cet objectif, la méthode va utiliser une boucle qui parcours à travers chacune des entrées dans la matrice de compte qui a été créée ultérieurement. Pour chaque entrée, la nouvelle matrice va calculer la probabilité de chaque caractère suivant en divisant son compte par le nombre total de caractères qui suivent le caractère actuel. Elle initialise également les variables « min » et « max » à la valeur de 0.

Puis, pour chaque caractère suivant, la méthode calcule l’intervalle de probabilité en utilisant la formule : max = max + prob \* count ; « max » représente la borne supérieure de l’intervalle, « prob » est la probabilité du caractère suivant et « count » est le compte du caractère suivant. La borne inférieure « min » de l’intervalle est la valeur précédente de « max ».

La méthode va également stocker chaque intervalle calculé dans la « HashMap » de « matrix » en utilisant le caractère actuel comme clé externe et le caractère suivant comme clé interne. Une fois que toutes les entrées de données sont traitées, la méthode va retourner la « HashMap » contenant la matrice de probabilités qui va ainsi permettre de visualiser les intervalles pour chaque paire de caractères possible.

Une image contenant texte, capture d’écran

Description générée automatiquement

La quatrième méthode de la classe, « regenerateMatrix() » va prendre en paramètre une liste de noms et mettre à jour la propriété « names » de l’objet de l’instance de la classe. Ensuite, une nouvelle matrice est générée avec la méthode «generateMatrixWithInterval()» et met à jour la propriété « matrix » avec la nouvelle matrice. Grossièrement, cette méthode permet de réinitialiser la matrice et lui permettre une nouvelle banque de noms en en donnée. De ce fait, de nouveaux noms pourront être générée.

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

La cinquième méthode de la classe, « showMatrix() » est utilisée pour afficher la matrice de transition à l’écran. Tout d’abord, la méthode initialise des formats de chaîne de caractères (« formatChar » et « formatString ») qui seront utilisés pour bien aligner les valeurs de la matrice lorsqu’elles seront affichées à la console et ainsi avoir un affichage propre et lisible. Ensuite, la méthode « generateMatrixWithInterval() » est appelé pour pouvoir générer la matrice de probabilités à afficher. La méthode va par la suite créer une variable « test » pour stocker la première ligne de la matrice (qui est correspondante aux caractères de départ), en récupérant la liste des clés de la carte de caractères espace (‘ ‘) dans la matrice. En fait, la méthode parcourt chaque élément de « test » et affiche le caractère correspondant en utilisant le format de chaîne de caractères souhaité « formatChar ». Si l’élément est un caractère espace, la méthode affiche plutôt « vide ».

Pour chaque ligne, le programme commence par afficher le caractère correspondant à cette ligne. Encore une fois, s’il s’agit d’un espace, il affiche « vide » à la place. La variable « formatString » contient une chaîne de caractères qui définit le format d’affichage des chaînes de caractères. Ensuite, pour chaque élément de cette ligne (chaque objet « Interval »), le programme affiche la chaîne de caractères correspondante. La boucle interne parcourt les entrées imbriquées pour cette ligne. Comme précédemment, la ligne se termine par un retour à la ligne pour passer à la ligne suivante.

En fin de compte, l’ensemble de ces deux boucles affiche une matrice complète qui représente les probabilités de transition entre les différents caractères dans l’échantillon de noms qu’on lui a fourni.

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

La sixième et dernière méthode de cette classe, « generateRandomName() » génère un nom aléatoire en utilisant la matrice de transition qui a été définie précédemment.

La variable « randomName » définit une chaîne de caractères vide pour stocker le nom aléatoire qui vas être généré ultérieurement. Ensuite, « Random() » crée une instance de la classe « Random » qui sera utilisée pour générer des nombres aléatoires. La variable de caractère « current » représente le caractère actuel à considérer dans la génération du nom aléatoire. Elle est initialisée à un espace vide pour indiquer le début du nom qui sera aléatoirement généré. La variable « endOfName » définit un booléen qui indique si la fin du nom a été atteinte ou pas encore.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Ensuite, nous avons une boucle « while » qui est utilisée pour générer le nom aléatoire. Elle se poursuit tant et aussi longtemps que la fin du nom n’a pas été atteinte. À chaque itération de boucle, la méthode récupère les caractères possibles suivants et les probabilités associées à partir de la matrice de transition en utilisant la méthode « get(current) ». La variable « index » est ensuite initialisée avec un nombre aléatoire généré à l’aide de l’objet « Random » créé précédemment.

La boucle « for » parcourt chaque entrée de la carte récupérée dans la ligne précédente. La méthode vérifie si le nombre aléatoire généré précédemment se situe entre les bornes inférieure et supérieure de l’intervalle associé à cette entrée, et ce pour chacune des entrées. Advenant que tel est le cas, la méthode passe au caractère suivant et ajoute ce caractère à la chaîne « randomName ». Toutefois, si le caractère suivant est un espace vide, cela veut dire que la fin du nom est atteinte. La méthode définit donc la variable « endOfName » sur « vrai » et quitte la boucle. Finalement, la méthode retourne la chaîne « randomName » qui contient le nom aléatoire généré.

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

1. **Commentaires**

Les consignes et directives du travail étaient bien formulé et ce qui nous a permis de bien comprendre les exigences reliées au travail. Nous sommes confiants d’avoir bien répondue à ces attentes. Les plus grandes difficultés ont été de bien comprendre les algorithmes principaux et leurs fonctionnements, pour ceux d’entre-nous ayant moins d’expérience du côté pratique. En d’autres mots, traduire la théorie vue en classe en code Java fonctionnel et polyvalent. Néanmoins, ce travail nous a permis d’approfondir notre compréhension des concepts vus en classe et de développer nos compétences en programmation Java. Il n’y a toutefois pas eu de problème majeur et la réalisation du travail #2 s’est bien mener à terme.