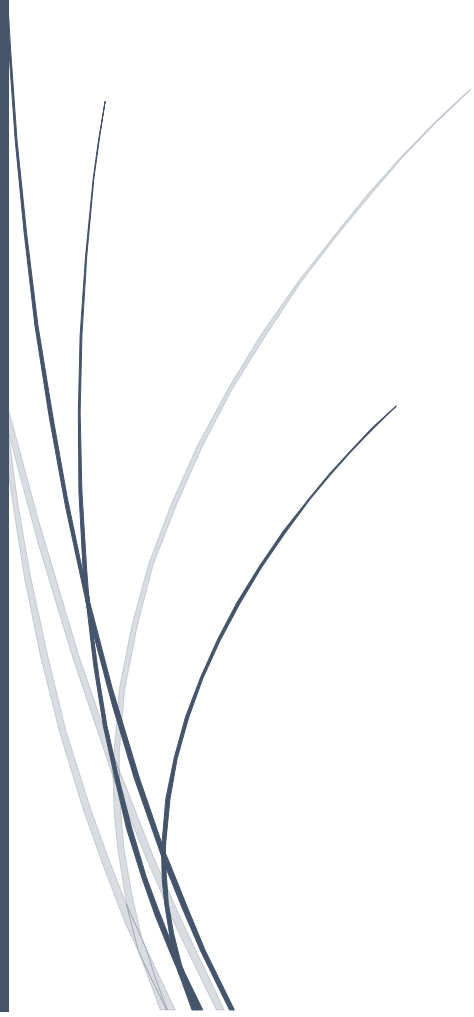
SCIENCES SORBONNE UNIVERSITÉ

12/04/2022

Rapport de projet

LU2IN006 – Structure de données



Floria Lim
Xiaohang Fang

Blockchain appliquée à un processus électoral

Sujet

Le but de ce projet est de modéliser l'organisation d'un processus électoral à l'aide de structure de données. Nous suivrons rigoureusement tout au long du projet un protocole spécifique afin de garantir la fiabilité et la sécurité du vote, qu'il est impérativement nécessaire de vérifier au vu de l'importance d'un vote électoral.

Description du code

Nous avons décidé de séparer le code par partie. Ainsi pour chaque partie, on a un fichier .c contenant le code des fonctions (part.c), un fichier .h contenant les définitions des structures et les entêtes des fonctions (part.h), ainsi qu'un dernier fichier .c contenant les jeux de test (main.c). Voici la liste de ces derniers :

- part1.c, part1.h, main1.c pour le développement d'outils cryptographiques. On y a rajouté une fonction puissance calculant la puissance afin de faciliter le retour d'un nombre premier entre 2^{t-1} et 2^t-1. Les fonctions encrypt et decrypt sont utilisées pour chiffrer un message de façon asymétrique.
- part2.c, part2.h, main2.c pour la création d'un système de déclarations sécurisées, auquel on a rajouté les fonctions free_signature et free_protected permettant de libérer les structures Signature et Protected pour faciliter la libération de la mémoire à la fin de l'exécution et ainsi éviter les fuites mémoires. Chaque citoyen est représenté par une clé publique et secrète, que l'on va utiliser afin pour sécuriser les déclarations de vote à l'aide d'une signature. La fonction generate_random_data permet de générer aléatoirement une liste d'électeurs, de candidats et de déclarations de votes que l'on conservera dans les fichiers keys.txt, candidates.txt et declarations.txt.
- part3.c, part3.h, main3.c pour la manipulation d'une base de déclarations centralisée. On récupère la liste des déclarations, des électeurs et des candidats et on les stocke dans des listes chaînées. On utilise ensuite la fonction fonction compute_winner qui utilise la structure de table de hachage pour comptabiliser les votes et on en ressortir le gagnant de l'élection grâce à la.
- part4.c, part4.h, main4.c pour l'implémentation d'un mécanisme de consensus avec des blockchains, pour sécuriser l'élection et éviter les fraudes. Le mécanisme de consensus utilisé est le proof of work qui fait appel à une fonction de hachage difficile à inverser. La fonction compute_proof_of_work permet de rendre un bloc valide.
- part5.c, part5.h, main5.c pour la manipulation d'une base décentralisée de déclarations, auquel on a rajouté une fonction max calculant le maximum entre deux entiers pour faciliter la détermination de l'enfant le plus grand. Une structure

arborescente de blocs est ici utilisée et on regarde la plus longue chaîne pour en ressortir le vainqueur de l'élection. Pour une suppression plus facile et complète d'un arbre, nous avons ajouté à la fonction delete_tree 2 booléens en paramètre pour spécifier si l'on veut oui ou non supprimer l'auteur et la liste des votes du bloc. Nous avons également ajouté un paramètre dans la fonction permettant d'afficher un nœud (print_noeud) afin de conserver sa profondeur et ainsi pouvoir le séparer de son parent en le décalant de quelques espaces. Tous les 10 votes soumis, un bloc est créé avec add_block et est stocké dans le dossier Blockchain. Dans le cas où le nombre de votes n'est pas un multiple de 10, on n'oublie pas d'ajouter les votes restants dans un dernier bloc. Une fois tous les votes soumis, un arbre est créé à partir de ces blocs et il est alors possible à partir de cet arbre de calculer le gagnant de l'élection en ignorant les fraudes présentes.

Réponse aux questions

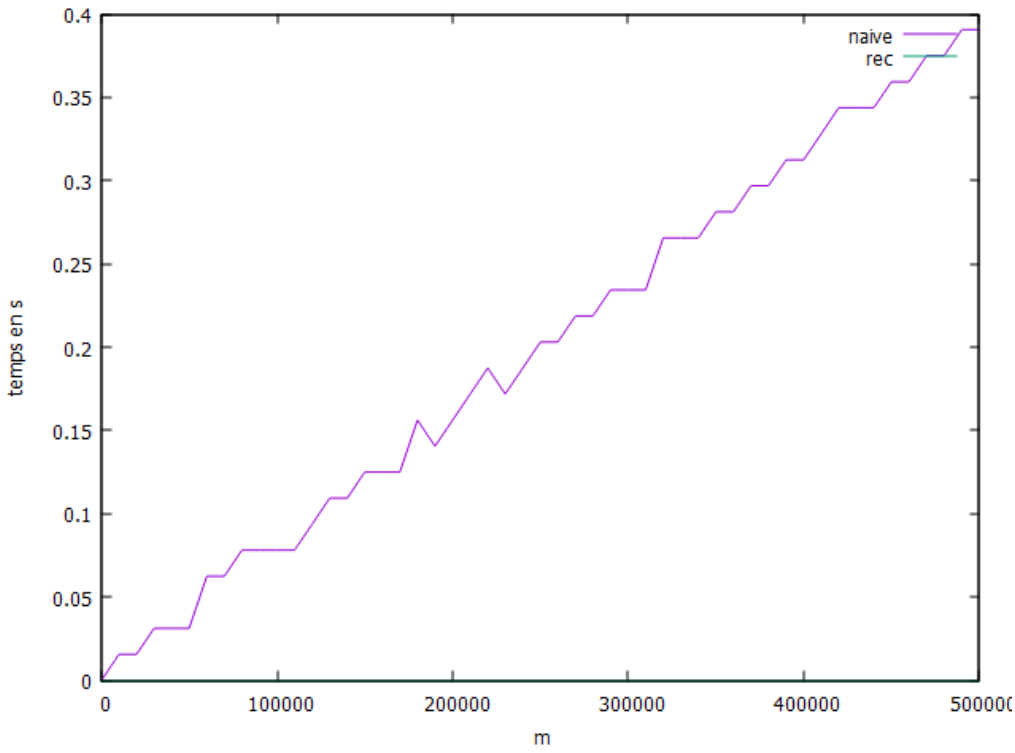
Exercice 1

1.1. Dans le pire des cas (p est premier), il est nécessaire de faire p-3 tours de boucles. La complexité est donc en O(p).

2.2. Le plus grand nombre premier qu'on arrive à tester en moins de 2 millièmes de seconde est 3344.

1.3. A chaque fois il est nécessaire de faire m tour de boucle. La complexité est en θ(m).

1.5. Voici la courbe obtenue en traçant les courbes de temps des deux méthodes en fonction de m :

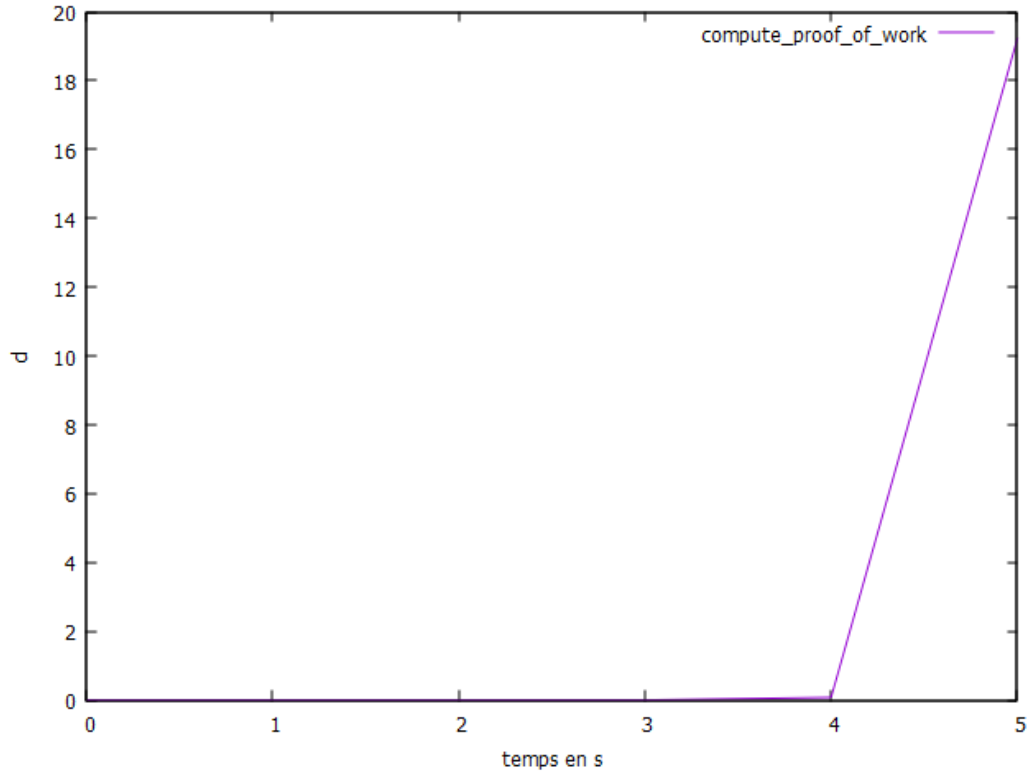


On peut observer que la fonction naïve croit linéairement, tandis que celle récursive envoisine quasiment constamment les zéros secondes. Cette dernière est donc beaucoup plus efficace que la première.

1.7. Si au moins $\frac{3}{4}$ des valeurs entre 2 et p-1 sont des témoins de Miller pour p, alors on a $\frac{1}{4}$ des valeurs qui le sont. La probabilité d'erreur est donc de $\frac{1}{4}$ si p n'est pas premier.

Exercice 7

7.8 Voici la courbe du temps moyen de la fonction compute_proof_of_work selon la valeur de d :



On peut voir que la courbe est exponentielle. Dès que d vaut 5, la fonction prend plus de 1 seconde pour s'exécuter (elle passe de 0.07s à 21.6s).

Exercice 8

8.8. La complexité de la fonction de fusion des deux listes chaînées est de O(n), avec n la longueur de la première liste. En effet, il est nécessaire de parcourir l'entièreté de la première liste pour ajouter le début de la deuxième liste à la fin de la première. Pour avoir une fusion en O(1), il suffit de ajouter à la structure de la liste chaînée un pointeur vers le dernier élément de la liste.

Exercice 9

9.7. La blockchain est réputée pour être une base de données rapide et très sécurisée, et est aujourd'hui utilisée dans de nombreux domaines tels que le bitcoin ou les NFT. Son utilisation est à mes yeux légitime pour le cadre d'un processus de vote aussi important que celui d'un

vote électoral. Cependant, le mécanisme de proof of work est réputé pour être très coûteux en énergie, ce qui pourrait être très contrariant étant donné que ce type de vote concerne l'entièreté de la population d'un pays. De plus, bien qu'il permette de limiter les fraudes étant beaucoup trop couteux en énergie pour le fraudeur, il n'en empêche toutefois pas la totalité à garanti.

Description des jeux d'essai

La plupart des jeux d'essai ont été fourni dans l'énoncé pour ce projet, cependant nous l'avons modifié afin de retirer les fuites mémoires qu'ils présentaient.

Pour la comparaison du temps de compute_proof_of_work en fonction de d, nous nous sommes arrêtées dès que le temps dépassait 1 seconde.

Enfin, afin de simuler des tentatives de fraudes, nous avons ajouté au jeu d'essai, juste après la création d'un bloc après 10 soumissions de votes, un bout de code permettant la création d'un bloc frauduleux avec une chance de 25%.