Projet S4 2017/2018 *Université Paul Sabatier*

SOLUTION

GODINO Pierre & ROLLAND Florian *09/05/2018*

Personnel investi :

GODINO Pierre

Etudiant en Informatique

Université Paul Sabatier

Responsable Projet & JAVA

ROLLAND Florian

Etudiant en Informatique

Université Paul Sabatier

Responsable C

Sommaire :

- Algorithmes

- Retours

1. Conception des Algorithmes

Parmi les algorithmes utilisez dans le programme, beaucoup sont utilitaires et n'ont aucuns besoins d'être préprogrammés en pseudo-code.

Parmi les algorithmes qui faisaient prévue d'un minimum de complexité, nous avons relevés :

- Algorithme calculant le hash de Merkle d'un bloc
- Algorithme de calcul de hash
- Algorithme de vérification de difficulté

Arbre de Merkle

```
Liste_des_hashs = {…}
Pointeur_tableau_hashs = 0
statut_calcul = Nombre_De_Transaction ( Constante propre au bloc)

Tant que (statut_calcul != 1) Alors
        Si (statut_calcul est impair) Alors
                statut_calcul + 1

        Pour (indice = 0 tant que indice strictement inférieur à (statut_calcul – 1) Alors
                Liste_des_hashs[pointeur_tableau_hashs] = hash de (liste_des_hashs Conc. liste_des_hashs + 1)
                Indice + 2

        Statut_calcul / 2
        Pointeur_tableau_hashs = 0
```

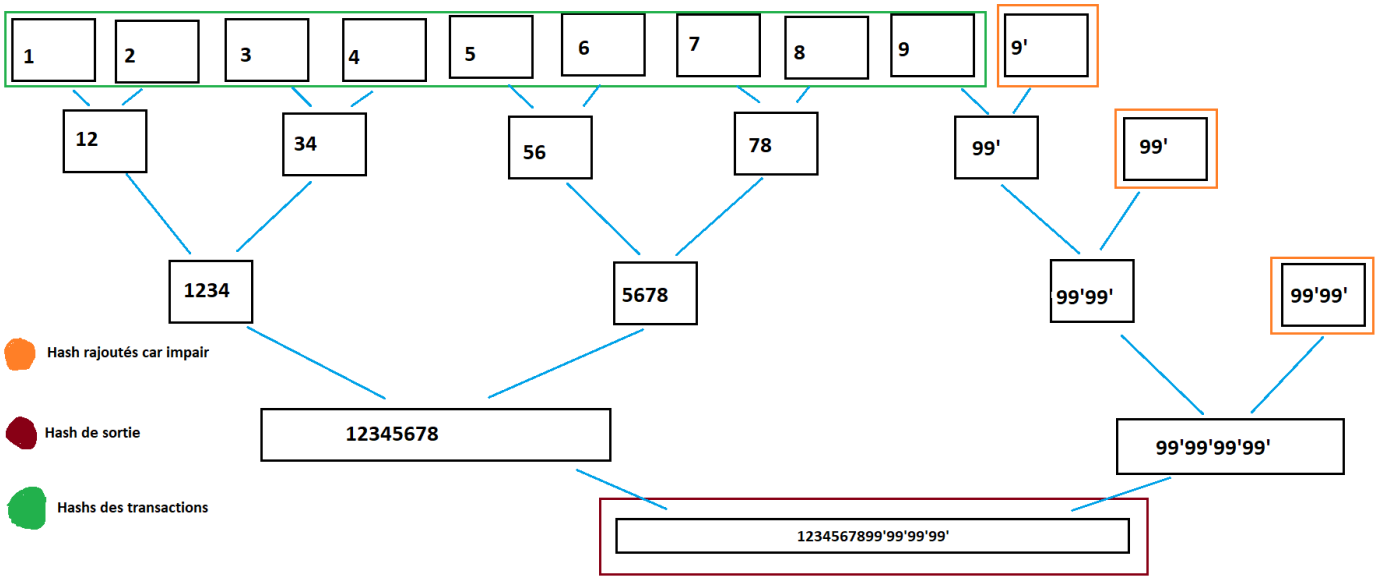


Schéma du fonctionnement de l'algorithme avec un cas de 9 transactions

Vérification de la difficulté

Hash_A_Verifier = {…}
Difficulté = 5

Le_Hash_est_il_bon = Vrai

Pour (Indice = 0 et tant que Indice est strictement inférieur à la Difficulté) Alors

Si Hash_A_verifier[i] est different de '0' Alors

Le_hash_est_il_bon = Faux

Indice + 1

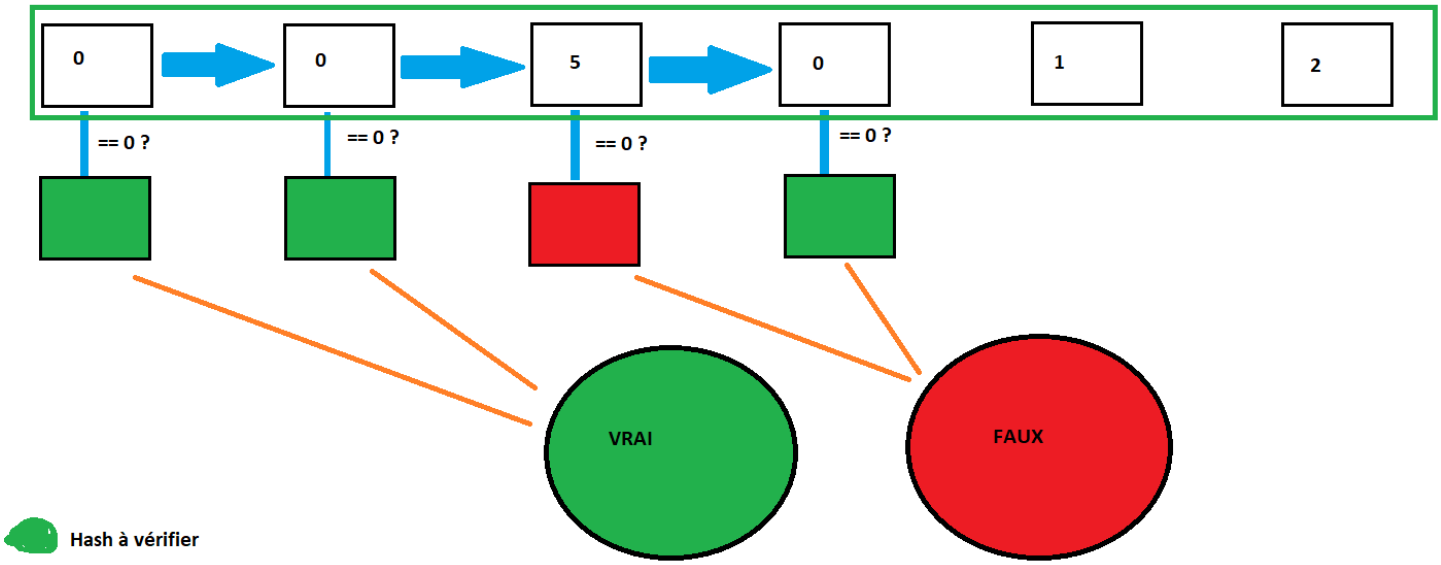


Schéma de la vérification pour un cas de difficulté égale à 4

Calcul du hash de bloc

Nonce = 0

Calcul_terminé = Faux

Hash = {vide}

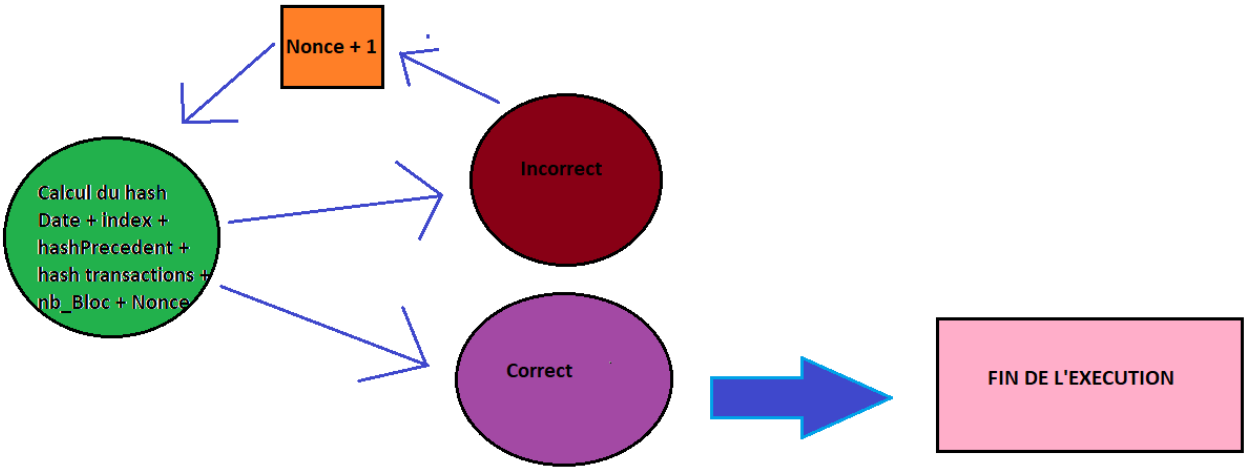
Tant que Calcul terminé est Faux

Hash = hash de (index + Date_Creation + hash_precedent + Hash_de_merkle + nombre_de_transactions + Nonce)

Si Hash correspond à la difficulté, Alors

Calcul_terminé = Vrai

Nonce + 1



2. Retours

Chaque algorithme a été élaboré sur un tableau, grâce à des schémas similaires à ceux présentés.

L'algorithme de Merkle a été retravaillé pour arriver au stade présenté précédemment :

- V1 : Algorithme utilisant deux listes
  Problèmes : Mémoire et espace de stockage pas optimisés

- V2 : Algorithme n'utilisant qu'une seule liste, à la fin de la V2 du programme, nous imaginions difficilement comment optimiser plus la fonction.

Beaucoup d’erreurs ont été à corriger pour cet algorithme.

L’algorithme de vérification de difficulté marche à 100% dès le premier test et est optimisé autant que possible.

L’algorithme de calcul de hash marche à 100% dès le premier test lui aussi et est optimisé autant que possible.