

Récapitulatif des Tps Informatique Info0903

Binôme : B

Remplir la colonne « état » de chacun des items des Tps sous la convention suivante :

A : Mieux que demandé et/ou exercices facultatifs fait.

B : Testé et validé.

C : Fait mais moins bien que demandé.

D : Fait mais Buggé.

E : pas fait !!

Attention d'ajouter systématiquement un complément dans le champs commentaire, notamment pour **expliciter l'intégralité de vos sources.**

Vous pouvez ajouter des lignes mais évidemment pas en enlever.

Attention surtout à faire soigneusement votre projet et la présente feuille : le niveau requis en Master2 est celui d'un produit prêt à être commercialisé.

TP1

	Etat A,B,C,D,E	Commentaires et questions
Une Fonction de hachage naïve	B	Additionne chaque octet dans un entier, et reconvertis
Une Fonction de hachage sophistiquée	A	Elle passe les 3 tests (khi2, permutation, rang)
Une Utilisation de SHA256	B	On a testé l'exemple donné et on a appliqué SHA256 sur 10000 chaînes aléatoires
Un test de ces trois fonctions de hachage par le test de la fréquence d'une valeur donnée ($f=p+-\sqrt{n}$)	B	Les tests sont lancés sur hachage naïf, hash sophistiquée, sha256 (voir TP1.py)
Test de ces trois fonctions pour voir si les valeurs des octets des sorties suivent la loi uniforme avec le Khi ²	B	Pareil qu'au dessus
Test des permutations	B	Pareil qu'au dessus
Optionnel : Test du rang ou autre test	B	Pareil qu'au dessus

TP2

	Etat A,B,C,D,E	Commentaires et questions
Fonctions simulant une seule partie d'un jeu vu en cours en fonction des probabilités d'action des deux joueurs	B	Fonction simulant le tir de 10000 penalty entre le gardien et le tireur.
Fonctions renvoyant l'espérance de Gain du joueur 1 en fonction des probabilités d'action des deux joueurs	B	La fonction renvoie le gain en fonction du tableau de gain du cours (L1, L2).
Fonction renvoyant la nouvelle probabilité d'action du joueur 1 après simulation sur deux valeurs et choix de la meilleure par le joueur.	B	La fonction main adapte la probabilité alpha et beta en fonction du score.
Simulation complète de l'évolution des probabilités des deux joueurs : retrouve-t-on les résultats du cours ?	C	La fonction pour tracer le graphique est faite mais le losange est presque imperceptible. Cependant, on remarque bien que cela tourne en spirale.

[47,95] [63,15]

main () :

- Initialise \perp , B , $l1$, $l2$, pas
- while True :

simuler(1000)

grader, adapter(score_gardien)

simuler(10000)

tirer, adapter(score_tireur)

renvoie des zones de tir
de gardien

lancer()

- cannes - score

- pas

- direction $\leftarrow -1 \text{ ou } 1$

+ adapter()

simuler() :

Nod_gardien = 0

Nod_tireur = 0

for i in range(t) :

Si rand() < L :

\leftarrow garde gardien

Si rand() < B :

\leftarrow garde tireur

Nod_gardien += l1[0]

Nod_tireur += l2[0]

) on ajoute les points de la

gauche cas

...

return Nod_gardien, Nod_tireur

adapter(S) :

Si S > cannes - score :

pas += direction * pas

Sinon :

direction *= -1

pas += direction * pas

TP3,4 et 5: Simulation de la blockchain

	Etat A,B,C,D,E	Commentaires et questions
Classe Block	B	Fait
- calcul du hash d'un bloc	B	Fonction Bloc.hash() se basant sur le hash précédent, les transactions, la date, et le nonce.
- vérification du bloc	B	On vérifie que le hash du bloc commence par un certain nombre de 0 (configurable)
Classe Serveur	B	Fait
- Recherche d'un nouveau Bloc par itération du nonce afin d'avoir un hash comportant au moins 3 zéros	B	Génération d'un nonce aléatoire à chaque création de bloc. Puis vérification du bloc créé.
- Recevoir	B	Permet de recevoir un bloc ou une blockchain, vérification et validation du bloc ou de la blockchain reçue.
- Afficher blockchain	B	Affiche les 5 derniers blocs en détail (configurable). Actuellement désactivé car envoie trop de messages dans la console.
Simulation de plusieurs acteurs sur la blockChain	B	A chaque itération de la boucle principale, on sélectionne un serveur au hasard (pondéré par la puissance) pour tenter de trouver un bloc. Il l'envoie ensuite à tout le monde.
Simulation d'un tricheur	B	Le tricheur refuse tous les messages tant qu'il n'a pas trouvé de bloc. Une fois trouvé, il envoie aux autres son bloc et accepte les blocs de ceux qui ont participé à sa chaîne.
Tests de plusieurs simulations	C	<p>Pour chaque simulation, nous suivons l'évolution du nombre de blockchains, évolution de la taille de la plus grande blockchain, la répartition des blockchains honnêtes/malveillantes, puis le nombre de blocs trouvés par serveur.</p> <p>Nous avons testé 3 simulations différentes :</p> <ol style="list-style-type: none"> 1. 10 serveurs sans tricheurs, puissance de calculs aléatoire entre 10 et 100 (pas de 25) : les évolutions des métriques sont bien ce qu'on attend (voir blockchain_simulation_results_1.png). 2. 10 serveurs + 1 tricheur avec 20% de la puissance totale : 15 blocs sont trouvés avant que le tricheur trouve son premier. Il y a alors deux blockchains mais le tricheur n'arrivera jamais à rattraper la chaîne honnête (voir image). 3. 10 serveurs + 1 tricheur avec 35% de la puissance totale : Deux chaînes différentes sont créées rapidement, le tricheur arrive à convaincre

les autres serveurs au bout de 5 blocs.
Tous les serveurs ont récupéré la blockchain malhonnête.
(voir image)

Idée d'amélioration : ajouter un 2ème tricheur.

SERVEUR	BLOC																			
<ul style="list-style-type: none"> - Blockchain : List[Block] - Transactions à intégrer - Mémoriser : DB[SERVEUR] - puissance ? ? ? 	<ul style="list-style-type: none"> - hash Bloc précédent - DB des Transactions - Date - momes 																			
<ul style="list-style-type: none"> + broadcast + recevoir + recherche Bloc 																				
\Rightarrow fonction (P_i): <ul style="list-style-type: none"> On prend le meilleur serveur On calcule $\sum P_i$ On prend le meilleur serveur 	<ul style="list-style-type: none"> qui a fait le bloc (pas une transaction pour l'instant) 																			
	<ul style="list-style-type: none"> - Calcul de la somme des puissances - On prend un nombre atte 0 et la somme (s'ajoute) - On sélectionne le serveur qui possède la puissance cumulée est le plus proche de l'obtention : <table border="1"> <tr> <td>P_1</td> <td>P_2</td> <td>P_3</td> <td>P_4</td> <td>P_5</td> </tr> <tr> <td>0</td> <td>P_0</td> <td>↓</td> <td>250</td> <td>350</td> <td>400</td> <td>550</td> </tr> <tr> <td></td> <td></td> <td></td> <td>213</td> <td></td> <td></td> <td></td> </tr> </table>	P_1	P_2	P_3	P_4	P_5	0	P_0	↓	250	350	400	550				213			
P_1	P_2	P_3	P_4	P_5																
0	P_0	↓	250	350	400	550														
			213																	
	recevoir (Bloc) <ul style="list-style-type: none"> - vérifie hash Bloc précédent - vérifie Transaction - calcul hash du Bloc - commence par NB 0 \Rightarrow si le Bloc est bon # si - ce qu'il de renvoyer à tous les serveurs 																			