ATELIER 1 : IMPLEMENTATION COMPLETE D'UNE BLOCKCHAIN SIMPLIFIEE EN C++

Exercice 1 : Arbre de Merkle

- Ecrire un programme permettant l'implémentation de l'arbre de Merkle basique en C++, from scratch.
- Vérifiez le bon fonctionnement de l'arbre en donnant des exemples d'exécution.

Exercice 2 : Proof of Work

- Implémentation de Proof of Work en C++.
- Une implémentation des blocs et chaines sera nécessaire.
- Changer à chaque fois la difficulté du **hashage** et calculer le **temps d'exécution** pour chaque niveau de difficulté.
- Vérifiez le bon fonctionnement du concept en donnant des exemples d'exécution.

Exercice 3 : Proof of Stake

- De même, faire l'implémentation de Proof of Stake en C++
- Donnez un exemple de validation avec Proof of stake et Proof of work et calculer le temps d'exécution de chacune pour en déduire le plus rapide. *[ce paramètre de rapidité dépendra de votre implémentation]*

Exercice 4 : Manipuler une mini-blockchain from scratch

Dans les exercices précédents, vous avez construit pas à pas les principaux éléments d'une blockchain :

- **Exercice 1 :** Arbre de Merkle pour résumer les transactions.
- **Exercice 2 :** Proof of Work pour sécuriser un bloc.
- **Exercice 3 :** Proof of Stake comme alternative plus rapide à PoW.

Dans cet exercice, vous allez intégrer ces éléments pour créer une **mini blockchain simple** en C++.

Partie 1 – Structure des blocs et de la chaîne

1. Définissez une classe **Transaction** (par exemple : id, sender, receiver, amount).
2. Réutilisez votre implémentation de **Merkle Tree** pour calculer le *Merkle Root* à partir d'une liste de transactions.
3. Définissez une classe **Block** contenant :
   - un identifiant,
   - un timestamp,
   - le hash du bloc précédent,
   - le Merkle root des transactions,
   - un nonce (utilisé pour PoW),
   - le hash du bloc.
4. Définissez une classe Blockchain permettant :
   - d'ajouter un bloc,
   - de vérifier l'intégrité de la chaîne.

Partie 2 – Proof of Work (PoW)

1. Intégrez dans la classe **Block** la fonction **mineBlock(difficulty)** qui calcule un hash respectant la difficulté (par ex. hash commençant par *n* zéros).
2. Ajoutez des blocs en utilisant le **PoW** et mesurez le temps de minage pour différentes valeurs de difficulté (comme fait dans l'exercice 2).
3. Vérifiez que la chaîne reste valide.

Partie 3 – Proof of Stake (PoS)

1. Implémentez un mécanisme simple de sélection de validateur (ex : choisir aléatoirement parmi une liste de validateurs avec des poids proportionnels à leur "stake").
2. Intégrez PoS dans la création d'un bloc : au lieu de miner (PoW), le bloc est validé par le validateur sélectionné.
3. Comparez expérimentalement le temps de validation entre PoW et PoS (comme fait dans l'exercice 3).

Partie 4 – Analyse comparative

1. Ajoutez plusieurs blocs à la chaîne avec PoW.
2. Ajoutez plusieurs blocs à la chaîne avec PoS.
3. Comparez les résultats :
   - Rapidité d'ajout des blocs.
   - Consommation de ressources (approximative : temps CPU).
   - Facilité de mise en œuvre.