TD: threads 2

Exercice 1. Calcul de logarithmes discrets

On souhaite implémenter un serveur de calcul de logarithmes discrets dans un groupe cyclique. Un/e utilisateur/trice devra pouvoir soumettre des requêtes au serveur via un terminal, et celui-ci se chargera de les transmettre à un thread de calcul. Lorsqu'un résultat a été trouvé, le thread de calcul doit le communiquer au serveur. Les contraintes sont les suivantes :

- Le nombre de threads de calcul est spécifié à la création du serveur.
- Tant qu'aucun calcul n'est à effectuer, les threads de calcul doivent rester inactifs.
- Si une requête est faite au serveur et que tous les threads de calcul sont occupés, le serveur doit rendre la main à l'utilisateur/trice (qui doit alors être en mesure de soumettre d'autres requêtes) et transmettre la requête à un thread de calcul dès que possible.

a. Écrire une interface `CyclicGroup` qui fournira les fonctions permettant de calculer un logarithme discret avec un algorithme d'énumération naïf. On dotera notamment cette interface d'une fonction d'exponentiation $x \mapsto g^x$, où g est un générateur du groupe.

b. Écrire une classe `ZPStar` qui implémente l'interface ci-dessus pour le groupe multiplicatif $\mathbb{Z}/134217757\mathbb{Z}^\times$ d'ordre 134217756 (on pourra utiliser le fait que 5 est un générateur de ce groupe).

c. Écrire une classe `Pool` qui permet de gérer la soumission à des threads de calcul. Celle-ci devra posséder les deux méthodes suivantes :

- `void compDlog(CyclicGroup G, long target)`, dont le but est de soumettre un calcul à un thread disponible.
- `void queryDone(long target, long res)`, qui sera appelée par un thread de calcul ayant fini son travail et affichera le résultat.

Vous devez également réfléchir aux attributs à ajouter à cette classe, aux éventuelles règles d'accès des méthodes ci-dessus, à de possibles arguments supplémentaires pour les méthodes ci-dessus, à de possibles méthodes supplémentaires, aux possibilités d'héritage, etc.

d. Écrire une classe `WorkerDlog` qui implémente les threads de calcul. Celle-ci devra posséder au minimum les méthodes suivantes :

- `long findDlog(CyclicGroup G, long target)`
- `void doWork(CyclicGroup G, long target)`

Vous devez également réfléchir aux attributs à ajouter à cette classe, aux éventuelles règles d'accès des méthodes ci-dessus, à de possibles arguments supplémentaires pour les méthodes ci-dessus, à de possibles méthodes supplémentaires, aux possibilités d'héritage, etc.

e. Écrire une classe `SubmissionThread` qui permet d'implémenter des soumissions non bloquantes.

f. Écrire la méthode `main` du programme.