L’applicazione prevede l’individuazione delle chiavi segrete dell’algoritmo Diffie-Hellman key exchange. Le uniche informazioni a nostra disposizione sono che la coppia di chiavi da individuare (a,b) sono:

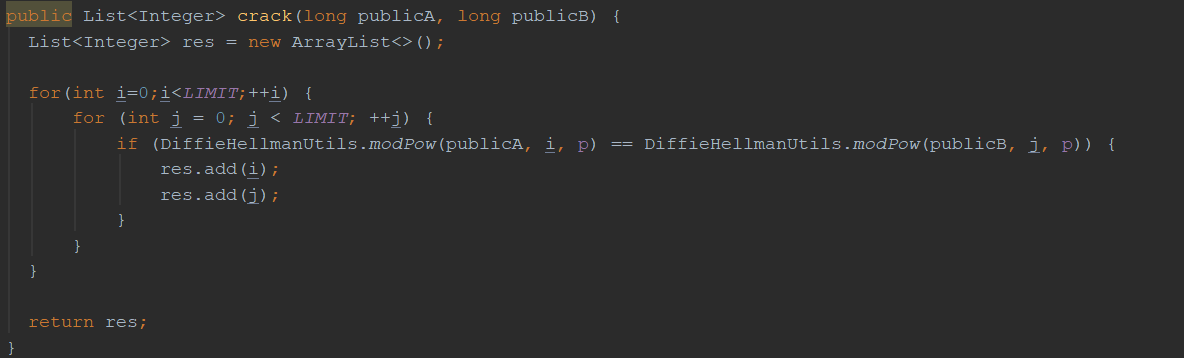
1. 0 <= a <= 65535;
2. 0 <= b <= 65535.

L’esercizio richiede che venga eseguito tale ricerca attraverso un attacco di forza bruta ovvero provare ogni possibile combinazione e salvare tutte le coppie che soddisfano la seguente condizione:

B^a mod p = A^b mod p

Dove p e g sono dati di pubblico dominio e A = g^a mod p, B = g^b mod p sono i messaggi che vengono scambiati. Quando la condizione è soddisfatta si prendono i valori a e b e li si inseriscono nella lista res.

Un semplice codice potrebbe essere il seguente:



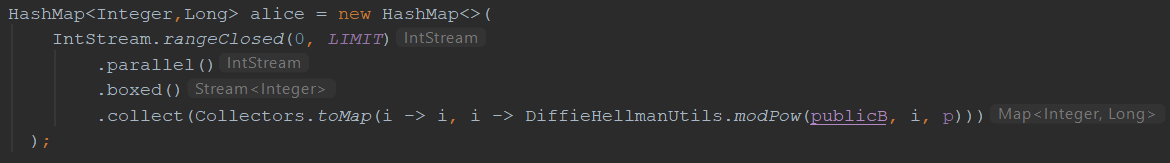
Tuttavia, si dovrebbero eseguire 65535x65535 operazioni consecutive costringendo uno dei thread a lavorare in modo esclusivo su di esso fino al suo termine con tempi troppo elevati.

L’esercizio richiede che vengano introdotti i concetti appresi a lezione di concorrenza e distribuzione in modo da ridurre i tempi di esecuzione.

Il programma richiede che vengano sfruttati tutti i core del computer adottato per questo motivo ho optato per l’utilizzo dei ParallelStream. In questo modo non è necessaria la creazione thread personalizzati, non si crea l’esigenza di studiare quanti thread mandare in esecuzione perché non si crei un effetto contrario a quello desiderato e non deve essere gestita alcuna eccezione.

Per poter sfruttare i ParallelStream ho deciso di adottare due HashMap<Integer,Long>, uno per i dati di A e l’altro per i dati di B, che assumono rispettivamente i nomi di alice e bob.

I dati di A sono stati creati usando la classe IntStream in modo tale che fosse possibile definire un rangeClosed, ovvero un insieme di numeri compresi tra 0 e 65535 su cui operare. Quindi per restare conforme alla scelta di usare i paralellStream ho inserito il metodo paralell(). A seguire con boxed() si è convertito l’IntStream in uno Stream<Integer> e per finire all’interno del metodo collect usando la classe Collectors ed il metodo toMap ho creato la mappa di A. Nell’immagine seguente è possibile visualizzare quanto descritto sopra:



Quello che cambia tra A e B oltre al nome della variabile è il primo valore del metodo statico modPow, in questo caso publicB, nell’altro publicA.

Arrivati a questo punto abbiamo due mappe con 65535 entry ciascuno, resta da confrontarle in modo parallelo. Prendo una delle due HashMap richiamo qundi il metodo entrySet per ottenere un Set<Entry<Integer,Long>> qundi richiamo il metodo parallelStream() in modo da indicare al programma di parallelizzare l’esecuzione del codice e con il terminale foreach indico che le operazioni a seguire devono essere eseguite per ogni singolo elemento. All’interno del foreach con una lambda espressione passo un set quindi prendo la seconda mappa applico come in precedenza entrySet e parallelStream() quindi filtro ovvero indico quali sono i risultati da prendere in considerazione che vengono poi convertiti in Stream<Integer> nel metodo map e anche in questo caso ho come terminale un foreach. All’interno di quest’ultimo foreach, poiché i valori a e b devono essere salvati una di seguito all’altra e c’è il rischio che con la parallelizzazione ciò non avvenga ho inserito un blocco syncronized in modo da indicare che quella sezione di codice deve essere eseguita in modo sequenziale.

Il codice di quanto descritto è il seguente:

