ESERCIZIO: SEMINARIO

Implementazione Java di un bruteforcer di chiavi AES di 16 caratteri (da 0 a Integer.MAX_VALUE)

Marco Macrì (MAT. 220070)

Idea d’approccio

L’idea è di suddividere la ricerca della chiave su n thread scelti dal chiamante del metodo di bruteforcing.
Siccome si opera in un intervallo specifico, quello che ho pensato di fare è di progettare un modo per assegnare i sotto intervalli di ricerca ai thread.

Prima di affrontare il problema sollevato, serve sapere come adattare un Integer affinché sia compatibile con lo standard della chiave richiesta.

Come sappiamo essa dovrà essere composta da 16 cifre le cui «significative» dovranno essere degli Integer. Per farlo abbiamo bisogno di un Padding, ossia un’aggiunta di z 0 prima del valore dell’Integer.

Ho voluto fornire la classe di utilità «Metodi», che contiene i metodi necessari affinché l’intero meccanismo sia avviato da una sola chiamata, per esempio, in un main.

String dammiStringa(Integer intero) e byte[] dammiChiave(Integer intero)

Il metodo riceve un intero (Integer), ne calcola la lunghezza della relativa stringa e appende tante volte 0 e il toString() di Integer fino ad arrivare alla lunghezza finale di 16 caratteri.
Per fare ciò mi avvalgo di uno StringBuilder e mi assicuro che il valore dato in input dia idoneo, se così non fosse sollevo un'eccezione

```
static String dammiStringa(Integer intero) {
    if (intero < 0) throw new IllegalArgumentException("Valore negativo");
    StringBuilder stringa = new StringBuilder();
    for(int i = 0; i< 16 - intero.toString().length(); i++, stringa.append("0"));
    stringa.append(intero);
    return stringa.toString();
}
```

Il secondo metodo si occupa di ottenere i byte dalla stringa relativa all'intero passato come input.
Si serve della funzione dammiStringa(…)

```
static byte[] dammiChiave(Integer intero) {
    return dammiStringa(intero).getBytes();
}
```

contenutoValido(byte[] daVerificare, byte[] obiettivo)

Il metodo riceve una possibile decifrazione del file in bytes, dopodiché cerca al suo interno un sottoarray che corrisponde all'array contiguo dei bytes della stringa dell' indizio. È usato nella classe BForcer.
Evito di fare troppi cicli e ne uso uno solo che nel caso peggiore gira il primo array una volta.
Per far ciò tengo all'esterno del ciclo l'indice dell'array obiettivo, nel caso di valore trovato, il cilco si ferma.
È usato nella classe BForcer.

```
static boolean contenutoValido(byte[] daVerificare, byte[] obiettivo) {
    if (daVerificare.length < obiettivo.length) throw new IllegalArgumentException("Arrays in input non validi o invertiti");
    int indiceObiettivo = 0;
    for (int i = 0; i < daVerificare.length; i++) {
        if (daVerificare[i] != obiettivo[indiceObiettivo]) {
            indiceObiettivo = 0;
        } else {
            indiceObiettivo++;
        }
        if (indiceObiettivo == obiettivo.length) return true;
    }
    return false;
}
```

leggiFile(File input) e scriviFile(String pathNome, byte[] outputArray)

Il metodo riceve in input un file che legge e restituisce sotto forma di array di byte.

```
static byte[] leggiFile(File input) throws IOException {
   FileInputStream fIS = new FileInputStream(input);
   byte[] ret = new byte[(int) input.length()];
   fIS.read(ret);
   fIS.close();
   return ret;
}
```

Il metodo riceve una stringa che rappresenta il nome o il path del file che dovrà scrivere e un array di output.
È invocato nella classe BForcer.

```
static void scriviFile(String pathNome, byte[] outputArray) throws Exception {
   File file = new File(pathNome);
   FileOutputStream outputStream = new FileOutputStream(file);
   outputStream.write(outputArray);
   outputStream.close();
}
```

decifra(byte[] chiave, byte[] input)

Questo metodo sostituisce quello di decifrazione della classe CryptoUtils.
Riceve un array di byte come chiave e un altro come input.
Crea un oggetto chiave, avvia il cifratore in modalità decriptazione.
Restituisce un' array decifrato secondo la chiave avuta.
È usato nella classe BForcer.

```
static byte[] decifra(byte[] chiave, byte[] input) {
   try {
      Key chiaveSegreta = new SecretKeySpec(chiave, "AES");
      Cipher cipher = Cipher.getInstance("AES");
      cipher.init(Cipher.DECRYPT_MODE, chiaveSegreta);
      return cipher.doFinal(input);
   } catch (Exception e) {
   }
   return null;
}
```

void bruteForce(int nThread, File input, String indizio)

Il metodo riceve in input il numero di thread da utilizzare, il file da decifrare, e una striga indizio che dovrà essere ricercata dentro i files decifrati, se essa sarà presente. Leggo il file una sola volta e lo converto in bytes, così facendo risparmio tempo. Creo un array di threads e calcolo i passi da fare per singolo thread mediante divisione intera. Tengo conto del resto, che delego all'ultimo thread.
Comincio a fare un ciclo sull'array incrementando il conto e popolando gli spazi. . .
Dopo mando in attesa il main di tutti i thread. Al termine dell'attività dei thread restituisco il tempo impiegato dalla ricerca.

```
public static void bruteForce(int nThread, File input, String indizio) throws Exception{
    Instant start = Instant.now();
    byte[] letturaOriginale = leggiFile(input);
    if(nThread <= 0) throw new IllegalArgumentException("Numero thread non valido!");
    if(input == null) throw new IllegalArgumentException("File == null!");
    BForcer[] bForcers = new BForcer[nThread];
    int passo = Integer.MAX_VALUE / nThread;
    int resto = Integer.MAX_VALUE % nThread;
    int conto = 0;
    for (int i = 0; i < nThread; i++) {
        if (i == nThread-1) bForcers[i] = new BForcer(letturaOriginale, conto, conto + passo + resto, i, indizio.getBytes(), bForcers);
        else {bForcers[i] = new BForcer(letturaOriginale, conto, conto + passo, i, indizio.getBytes(), bForcers);
        conto += passo;}
        bForcers[i].start();
    }
    for (int i = 0; i < nThread; i++) bForcers[i].join();
    Instant end = Instant.now();
    System.out.println("Tempo impiegato: " + Duration.between(start, end).toMinutes() + " minuti");
}
```

Class Bforcer 1/2

```
public class BForcer extends Thread{
    private final int inizio, fine, id;
    private BForcer[] bForces;
    private byte[]  outputArray;
    private final byte[] inputArray, indizioArray;

    public BForcer(byte[] inputArray, int inizio, int fine, int id, byte[] indizioArray,
BForcer[] bForces) {
        super();
        this.inputArray = inputArray;
        this.inizio = inizio;
        this.fine = fine;
        this.id = id;
        this.indizioArray = indizioArray;
        this.bForces = bForces;
    }

    @Override
    public void run() {
        System.out.println("Il thread "+ id + " comincia da "+inizio+" a "+fine);
        for (int i = inizio; i < fine && !this.isInterrupted(); i++) {
            try {
                byte[] chiave = Metodi.dammiChiave(i);
                outputArray = Metodi.decifra(chiave, inputArray);
                if (Metodi.contenutoValido(outputArray, indizioArray)) {
                    Metodi.scriviFile("Decifrato.dec", outputArray);
                    System.out.println("La chiave trovata sembra essere "+
Metodi.dammiStringa(i));
                    System.out.println("Il file in output ha nome Decifrato.dec");
                    interrompiTuttiNotThis();
                    break;
                }
            } catch (Exception e) {
            }
        }
    }

    private void interrompiTuttiNotThis() {
        for(int i = 0; i < bForces.length; i++) if(i != this.id) bForces[i].interrupt();
    }
}
```

Class Bforcer 2/2

Il cuore è la classe BForcer che si occupa di provare tutte le chiavi da un inizio ad una fine.
Riceve un sacco di parametri: un array di input su cui avviene la decfrazione, degli interi che definiscono inizio dell'intervallo, fine e l'id del thread creato. Riceve inoltre un secondo array di byte contenete l'indizio da ricercare e un riferimento all'array dei Bforcers (utile per interrompere tutti i thread a chiave trovata).
Il run(…) comprende un ciclo che tenta le chiavi nell'intervallo.
Il ciclo al suo interno controlla che il thread non sia interrotto.
In caso di chiave trovata viene creato un file dall'array di output contenente l'indizio, stampa l'avvenuta creazione del file e la chiave giusta, interrompe i Bforcers in esecuzione.

Metodo interrompiTuttiNotThis()

Il metodo interrompe i thrad thranne quello in escuzione.
Tale scelta è giustificata dal fatto che eventualmente si possa
Voler eseguire qualcos'altro con il thread che ha trovato la chiave,
Spetta a lui interrompersi

```
private void interrompiTuttiNotThis() {
   for(int i = 0; i < bForces.length; i++) if(i != this.id) bForces[i].interrupt();
}
```

Post esecuzione

Calcolo Speedup

$$Sn = Ts / Tn$$

Nel nostro il tempo impiegato su 4 thread è di 12 minuti, 52 su singolo thread.
Lo Speedup è quidi di 4,33

Ho stimato che il codice eseguibile in modo parallelo è di circa il 97%.
Quindi, grazie a questo calcolatore ho applicato la legge di Amdahl al nostro caso.
È prodotto il seguente grafico