Homework 1: Password Cracking

In questo homework si richiede di progettare e implementare in Java un algoritmo parallelo per il cracking di password codificate attraverso MD5. MD5 è un algoritmo di hashing crittografico che codifica una stringa in input di lunghezza arbitraria (la *password*) in una stringa di 128 bit che ne rappresenta il *digest* (detto anche *firma* o *checksum*). Ad esempio:

MD5(Cantami o diva del pelide Achille l'ira funesta) = b4dd7f0b0ca6c25dd46cc096e45158eb

In questo metodo di codifica anche piccoli cambiamenti nella stringa di input comportano probabilmente un digest molto diverso. Ad esempio, sostituendo "Cantami" con "Contami" si ottiene:

MD5(Contami o diva del pelide Achille l'ira funesta) = f065b51db9c592bf6ecf66a76e39f8d0

Proposto nel 1991 da Ronald Rivest e standardizzato attraverso la RFC 1321, l'algoritmo MD5 è caratterizzato da uno schema di codifica molto veloce e, sebbene importanti vulnerabilità identificate nel corso degli anni ne sconsiglino l'impiego, la sua diffusione risulta ad oggi ancora molto estesa.

Input e output. Il programma implementato deve ricevere in input un file di testo contenente, per ciascuna riga, il digest di una password da decifrare. Il nome del file di input (eventualmente preceduto dal path di accesso) deve essere l'unico argomento ricevuto dal programma.

Per ciascun digest d nel file, il programma deve stampare in output il digest stesso seguito dalla password π che lo ha generato, ovvero la stringa tale che *MD5(π)=d*. Per gli scopi dell'homework, potete assumere che le password contengano solo lettere minuscole e numeri (ovvero, che l'*alfabeto* delle password contenga esclusivamente i simboli $\{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$). Non potete invece assumere che vi sia un limite alla lunghezza delle password (più lunghe sono le password, maggiori sono le difficoltà – e quindi il tempo di esecuzione richiesto – per decifrarle).

Implementazione di MD5. La classe `java.security.MessageDigest`* contiene un'implementazione di MD5 che potete utilizzare. Il seguente codice vi mostra un esempio d'uso di alcuni metodi di MessageDigest:

```
public class MD5HashingExample {
    public static void main(String[] args) throws Exception {
        String password = "123456";

        MessageDigest md = MessageDigest.getInstance("MD5");
        md.update(password.getBytes());
        byte byteData[] = md.digest();

        // convert the byte to hex format
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < byteData.length; i++) {
            sb.append(Integer.toString((byteData[i] & 0xff) + 0x100, 16).substring(1));
        }

        System.out.println("Digest(in hex format):: " + sb.toString());
    }
}
```

*Si veda ad esempio https://www.cis.upenn.edu/%7Ebcpierce/courses/629/jdkdocs/api/java.security.MessageDigest.html

Il codice genera e stampa il digest associato alla password "123456".

Scopo dell'homework. Utilizzate un algoritmo esaustivo, basato sull'enumerazione di tutte le possibili stringhe sull'alfabeto specificato (lettere minuscole e numeri) e sulla successiva verifica che una certa password π coincida con uno dei digest contenuti nel file di input. Per enumerare le password, suggerisco di lavorare su una lunghezza fissata, evitando di generare da principio password troppo lunghe senza aver tentato prima la generazione di password più corte: orchestrate la generazione in modo da generare password di lunghezza via via crescente.

Non utilizzate euristiche (ad esempio dizionari di password) o altre tecniche crittografiche avanzate. Pur essendo queste tecniche tipicamente molto efficaci, il solo scopo dell'homework è la parallelizzazione di un algoritmo computazionalmente pesante. Vale infatti la pena notare che il numero di password di una data lunghezza k su un alfabeto di n simboli è n^k: si tratta infatti delle disposizioni con ripetizione di n elementi di classe k[§]. Nel nostro caso $n = 36$, mentre non potete fare assunzioni sul valore di k (lunghezza delle password). Osservate che 36^k può comunque essere un valore molto alto anche per k piccolo.

Esperimenti. A corredo dell'implementazione, eseguite una serie di esperimenti e misurate il tempo di esecuzione richiesto da ciascuno di essi. Confrontate i tempi richiesti con quelli di una esecuzione sequenziale, valutando lo speedup ottenuto. A breve metterò a disposizione sul sito Web del corso una serie di benchmark su cui testare il codice e un generatore di digest.

Documentazione. A corredo dell'implementazione (codice Java), inviatemi una breve relazione in formato pdf che illustri:

- l'approccio algoritmico utilizzato e il modo in cui avete effettuato la parallelizzazione;
- le caratteristiche salienti dell'implementazione realizzata e le relative ottimizzazioni;
- le caratteristiche della/e piattaforma/e su cui avete eseguito l'analisi sperimentale;
- i risultati sperimentali ottenuti, tramite tabelle o grafici opportunamente commentati;
- la modalità per compilare ed eseguire il programma (da linea di comando).

Modalità di svolgimento: potete lavorare in gruppi composti da al più 3 persone.

Deadline: 13 dicembre 2015.

[§]Questo programma può essere utile per richiamare alcuni concetti:

`http://utenti.quipo.it/base5/combinatoria/calcombinat.htm`