Di Antonio Pio Ricciardi
e Matteo Prata

Homework 1

➤ Modalità di compilazione ed esecuzione da terminale.

Compilazione ed esecuzione:

```
javac MainDecoder.java
java  MainDecoder <nome file> <numero thread>[facoltativo]
```

Un esempio:

```
javac MainDecoder.java
java  MainDecoder easy.txt 4
```

Queste linee di codice eseguiranno la decodifica dei digest presenti nel file **easy.txt** dividendo il lavoro in 4 thread. Se non viene definito il numero di thread da stdin, verrà utilizzato un numero di thread pari al numero dei processori (fisici e logici) disponibili sulla macchina.

➤ Calcolo delle permutazioni

Per decodificare le password è stato creato un algoritmo **ricorsivo ed esaustivo**, esso genera tutte le possibili permutazioni di lunghezza k a partire da un alfabeto di dimensione 36.

L'algoritmo prende come argomenti: un **prefisso** (stringa) ovvero la parte iniziale della permutazione; un **limite** (intero) che indica la lunghezza massima della permutazione.

Se la lunghezza del prefisso è uguale al limite allora è stata ottenuta una permutazione (la cui codifica in MD5 viene confrontata ai digest in input), altrimenti ogni lettera dell'alfabeto in input viene accodata al prefisso e la funzione viene chiamata ricorsivamente.

➤ Decodifica parallela

Abbiamo utilizzato la **classe thread** di Java per parallelizzare il calcolo delle permutazioni in parallelo.

Parallelamente vengono create permutazioni di parole che iniziano con caratteri diversi: ogni thread si occupa della generazione di permutazioni di caratteri. Le permutazioni iniziano con i caratteri presenti nel sottoalfabeto determinato dai due delimitatori (interi) che il thread prende in input.

Esempi:

Eseguire il programma con 2 thread significa quindi che l'alfabeto "a, b, …, z, 0, 1, …, 8, 9" viene diviso in due sottoalfabeti di dimensione 18 ciascuno, uno sarà "a, b, …, q, r" e l'altro "s, t, …, 8, 9".

I due thread opereranno parallelamente: il primo genererà le permutazioni di parole che iniziano per "a", poi per "b" e così via fino ad "r"; il secondo genererà le permutazioni di parole che iniziano per "s", poi per "t" e così via fino a "9".

Eseguire il programma con 36 thread significa che ogni thread parallelamente si occuperà di generare permutazioni di parole che iniziano con un solo carattere.

➤ Ottimizzazioni

Per evitare che vengano generate tutte le possibili permutazioni, il numero di digest presenti nel file input viene inserito in un contatore. Per ogni password trovata il contatore viene decrementato di uno. Quando il contatore arriva a zero, l'esecuzione termina.

La ricerca delle password nel file di input avviene parallelamente: per ogni nuova parola generata si verifica che la sua codifica sia contenuta in un HashSet (contenente tutti i digest del file di input), in caso affermativo viene stampato il digest e la sua decodifica. La ricerca del digest nell'HashSet risulta più efficiente rispetto all'ArrayList poiché il metodo *contains* nella prima struttura dati impiega tempo costante O(1) rispetto all'altra che impiega O(n).

➤ Caratteristiche delle piattaforme ed esperimenti

Le prove sono state effettuate su due macchine diverse:

- MacBook Pro 15" con processore Intel Core i7 a 2,2 GHz dotato di 4 core fisici e 4 logici.
- MacBook Pro 13" con processore Intel Core i5 a 2,4 GHz dotato di 2 core fisici e 2 logici.

Per ogni file contenente digest sono state effettuate diverse prove d'esecuzione, ognuna utilizzando un numero crescente di thread: 1, 2, 4, 8, 16, 36.

Si è usato **1 thread** per valutare l'esecuzione dell'algoritmo in modo seriale. E' stato inoltre studiato il caso particolare con **36 thread**; l'alfabeto delle password da decifrare infatti è composto da 36 caratteri, questo permette ad un singolo thread di concentrarsi sulle permutazioni di parole che iniziano con un solo carattere.

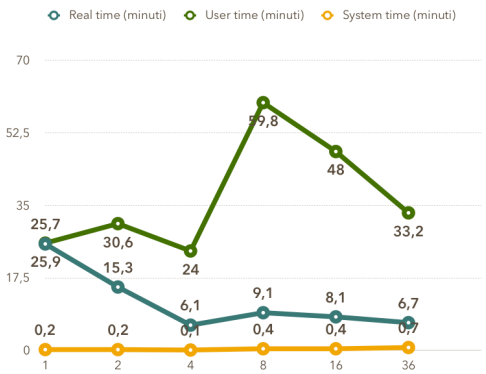
Prove sul file easy.txt: I tempi si sono mantenuti molto bassi su entrambe le macchine e si è ottenuto uno speedup massimo di 2,6 sul MacBook Pro 15".

Prove sul file hard.txt: I tempi sono stati molto elevati a causa delle grandi dimensioni delle parole da cercare, sono state trovate le seguenti parole: haydn, brahms, 1mozart, strauss, brahms9, 2wagner, debussy in circa 3 ore e 30 minuti, usando 4 thread su MacBook Pro 13".

Prove sul file medium.txt:
Sul MacBook 15": Passare da 1 thread a 2 thread comporta uno speedup di 1,67; lo speedup massimo è 4,21 ed è stato ottenuto eseguendo il programma con 4 thread.

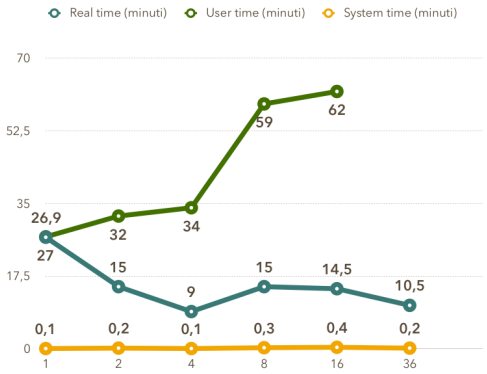
Sul MacBook 13": Passare da 1 thread a 2 thread comporta uno speedup di 1,79; lo speedup massimo è 2,99 ed è stato ottenuto eseguendo il programma con 4 thread.

Usando 4 thread, entrambe le macchine **ottengono il massimo speedup**. Tale risultato è determinato dal modo in cui opera l'algoritmo: la ricerca di alcune parole risulta infatti più veloce nei casi in cui il sottoalfabeto su cui opera il thread contiene nelle prime posizioni i caratteri iniziali della password da decodificare. Questo consente al MacBook 15" di ottenere con 4 thread speedup superlineare.



ESECUZIONE DI MEDIUM.TXT (MACBOOK PRO 15")

Threads	Real time (minuti)	User time (minuti)	System time (minuti)
1	25,7	25,9	0,2
2	15,3	30,6	0,2
4	6,1	24	0,1
8	9,1	59,8	0,4
16	8,1	48	0,4
36	6,7	33,2	0,7



ESECUZIONE DI MEDIUM.TXT (MACBOOK PRO 13")

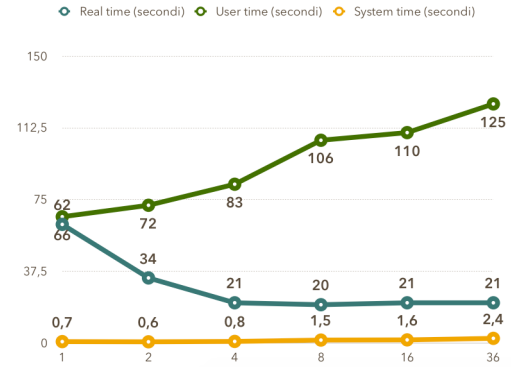
Threads	Real time (minuti)	User time (minuti)	System time (minuti)
1	26,9	27	0,1
2	15	32	0,2
4	9	34	0,1
8	15	59	0,3
16	14,5	62	0,4
36	10,5	39.1	0,2

Prove sul file test0.txt:
Il file test0.txt contiene un unico digest corrispondente all'ultima permutazione di 5 caratteri generabile.

Sul MacBook 15": Passare da 1 thread a 2 thread comporta uno speedup di 1,82; lo speedup massimo è 3,11 ed è stato ottenuto eseguendo il programma con 8 thread.

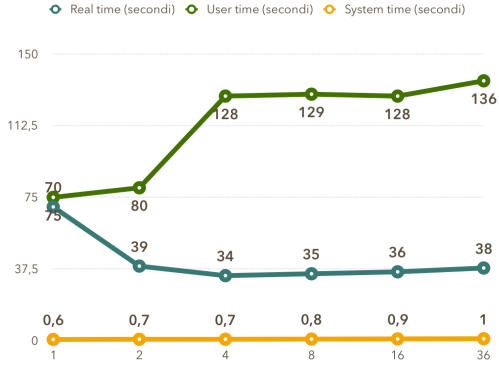
Sul MacBook 13": Passare da 1 thread a 2 thread comporta uno speedup di 1,8; lo speedup massimo è 2,1 ed è stato ottenuto eseguendo il programma con 4 thread.

Da questo test si nota che utilizzando un solo thread, le due macchine generano tutte le permutazioni in tempi simili. Aumentando il numero di thread i benefici apportati dalla presenza di più processori sul MacBook 15" sono evidenti in quanto riesce ad ottenere speedup maggiori rispetto al MacBook 13".



PERMUTAZIONI DI DIMENSIONE 5 (MACBOOK PRO 15")

Threads	Real time (secondi)	User time (secondi)	System time (secondi)
1	62	66	0,7
2	34	72	0,6
4	21	83	0,8
8	20	106	1,5
16	21	110	1,6
36	21	125	2,4



PERMUTAZIONI DI DIMENSIONE 5 (MACBOOK PRO 13")

Threads	Real time (secondi)	User time (secondi)	System time (secondi)
1	70	75	0,6
2	39	80	0,7
4	34	128	0,7
8	35	129	0,8
16	36	128	0,9
36	38	136	1

➤ Considerazioni finali

Si è notato quindi che eseguire algoritmi opportunamente scritti per supportare l'esecuzione in parallelo, porta vantaggi crescenti in termini di tempo all'aumentare del numero di processori disponibili sulle macchine.