Hashing Puzzle

Stefano Scarcelli

April 8, 2023

Contents

1 Struttura della soluzione

La soluzione sviluppata si basa sull'idea di suddividere in modo equo lo spazio di ricerca della stringa K sui i vari thread, valutando la validità della proprietà richiesta dal problema in parallelo.

1.1 Definizione dello spazio di ricerca

Per definire lo spazio di ricerca complessivo bisogna determinare il numero di stringhe che possono essere formate usando un alfabeto di N caratteri con lunghezza da 1 a P.

Questo può essere calcolate sommando il numero di disposizioni con ripetizioni delle varie stringhe di lunghezza da 1 a P, che risulta essere più semplicemente calcolabile tramite la formula della somma dei primi P termini della successione geometrica (partendo da 1 invece che 0)

$$N_k = \sum_{i=1}^{P} D'_{N,i} = \sum_{i=1}^{P} N^i = \frac{N - N^{P+1}}{1 - N}$$

In questo modo possiamo rappresentare lo spazio di ricerca (tutte le possibili stringhe K) tramite un numero

$$Index \in [1, N_k]$$

1.2 Suddivisione dello spazio di ricerca

Per suddividere equamente lo spazio di ricerca sui vari thread basta passare ad essi due indici (*indice di inizio* e *indice di fine*) generati tramite queste due formule

$$Index_{start}(i) = (i + 1)\frac{N_k}{n_{thread}}$$

$$Index_{end}(i) = i\frac{N_k}{n_{thread}} + 1$$

dove i rappresenta *i-esimo* thread (che va da 0 a n_{thread}) e n_{thread} il numero di thread creati.

1.3 Funzione di generazione delle stringhe

L'ultimo passo per completare la soluzione è quello di determinare una funzione che prende in input un indice di una delle possibili stringhe K e ne restituisce una sua rappresentazione facilmente decodificabile dal computer.
La scelta più naturale ricade su quella di rappresentare la stringa K come un array di byte che usano la codifica `UTF-8`.

Questo array avrà una dimensione variabile da 1 a P che dipenderà esclusivamente dall'indice, ed è possibile determinare la lunghezza verificando la seguente condizione

$$Len_k \in \mathbb{N} : N_k(Len_k - 1) <= Index <= N_k(Len_k)$$

testando in successione i vari valori che Len_k può assumere.

Una volta determinata la lunghezza dell'array è possibile decodificare l'indice della stringa in una sequenza di byte eseguendo quella che è una conversione di base da base 10 a base N, tenendo però conto di azzerare il valore dell'indice ogni volta che si passa da una stringa di lunghezza Len_k a Len_k+1, sommando come offset il valore in byte (in codifica `UTF-8`) del carattere più piccolo dell'alfabeto[1].

[1] Questo è possibile sfruttando la caratteristica che i caratteri dell'alfabeto usati per la costruzione della stringa K sono consecutivi tra di loro.

2 Implementazione

2.1 HashFinder class

L'implementazione della soluzione si base sulla classe `HashFinder` che implementa l'interfaccia `Runnable`.

2.1.1 Stato

La classe implementa le seguenti variabili di stato:

private final int n Numero di caratteri dell'alfabeto della stringa K;

private final byte cMin Rappresentazione a byte (UTF-8) del 1° carattere dell'alfabeto di K;

private final int d Numero di zeri con cui la stringa T deve iniziare per soddisfare la condizione di soluzione;

private final String zeros Stringa di D zeri;

private final byte[] s Rappresentazione a byte (UTF-8) della stringa S definita dal problema;

private final long end Indice dell'ultima stringa K che questa istanza deve valutare;

private long index Indice della stringa K da valutare;

private int lastKLen = 0 Lunghezza dell'ultima stringa K valutata (Inizializzata a 0 quando nessuna stringa K è stata ancora valutata);

private long strOffset Numero di stringhe K con lunghezza minore a quella correntemente in valutazione;

2.1.2 Costruttori

La classe implementa un solo costruttore che definisce l'istanza del thread impostando le varie variabili di stato `final` e inizializza `index`.

2.1.3 Metodi

La classe implementa un metodo statico e tre metodi dinamici, tra cui il metodo `public void run()` per l'esecuzione in multithread. Gli altri metodi invece sono le vari implementazioni della struttura della soluzione.

public void run() Esegue la valutazione delle stringe del thread testandone la proprietà;

private byte[] getNextK() Determina la stringa K dell'indice corrente;

private int kLen() Determina la lunghezza dell'array della stringa K dell'indice corrente;

public static long geometricSeries(int n, int p) Implementa la formula della successione geometrica;

2.2 App class

L'implementazione della soluzione viene eseguita tramite la classe `App` grazie a due implementazioni del metodo `public static void solvePuzzle`, uno dove i parametri rappresentano i dati minimi per definire il problema e un'altro (più generale) dove è possibile gestire il numero di thread da avviare, questo implementa la creazione ed avvio dei thread.

3 Analisi delle prestazioni

In aggiunta alla soluzione il codice implementa un sistema di misurazione del tempo di esecuzione che viene gestito tramite la creazione di un'istanza della classe `Timer`.

3.1 Timer class

La classe implementa due variabili di stato un costruttore ed un metodo usati per determinare il tempo di esecuzione complessivo del programma.

Alla creazione dell'istanza la variabile di stato `public Instant start` viene Inizializzata tramite la funzione `Instant.now()`. Il riferimento dell'oggetto viene passato a tutti i thread e non appena uno di loro risolve il puzzle esso chiama su di esso il metodo `public void endTimer()` che setta la variabile di stato `public Instant end` tramite la funzione `Instant.now()` e successivamente stampa la differenza tra l'inizio e la fine in millisecondi.

3.2 Prestazioni

Tramite il calcolo del tempo di esecuzione ho eseguito diverse run del programma con un diverso numero di thread per determinarne il numero ottimale.

3.2.1 Metodologia dell'analisi

Il tempo di esecuzione per ogni numero di thread è stato eseguito sulle seguenti istanza del problema (varie stringhe):

$\boldsymbol{S}$ = "Test-String-",
"Prova-Stringa-",
"String-Test-",
"Stringa-Prova-",
"SisOp-Course A-Hashing-Puzzle-" [2]

$\boldsymbol{D} = 6$ [3]

$\boldsymbol{Len_{k,\ max}} = 6$

$\boldsymbol{N} = 36$

$\boldsymbol{Char_0}$ = ':'

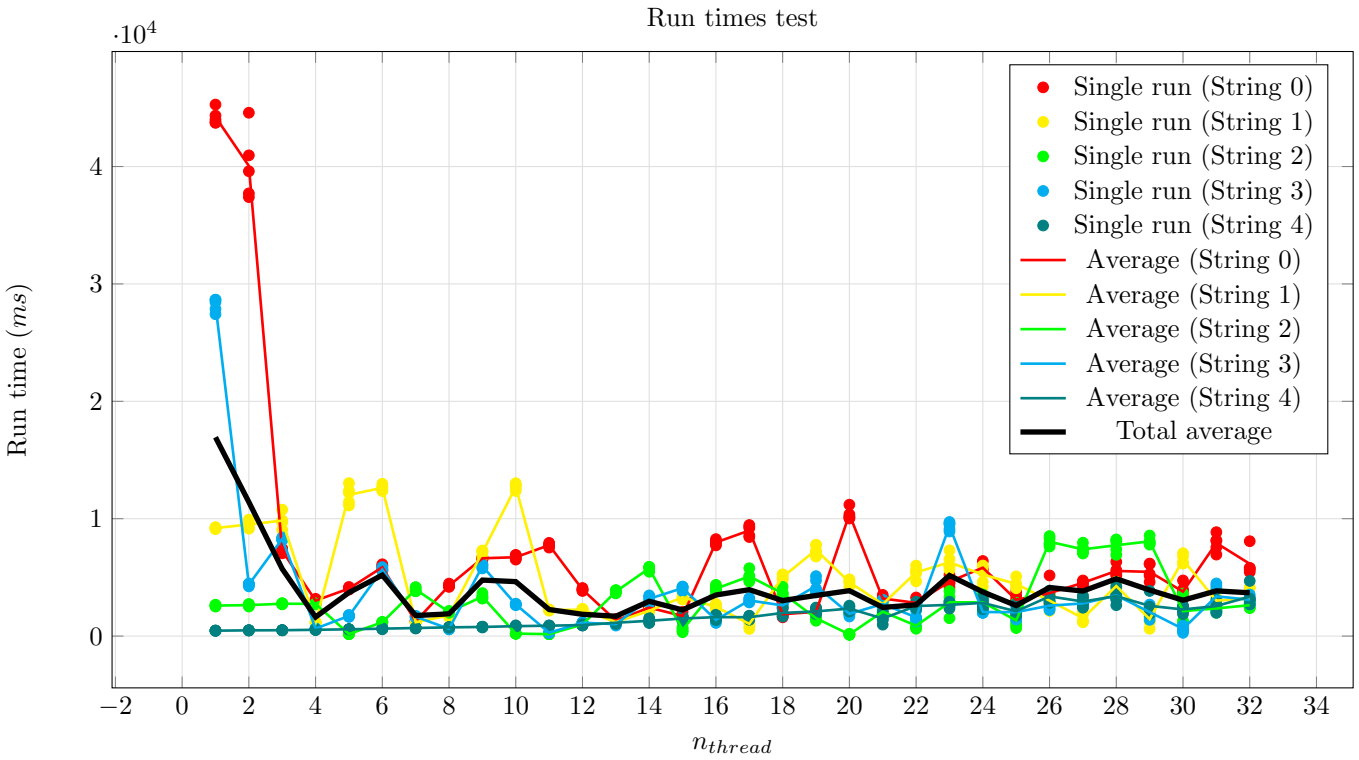
Per calcolare il tempo di esecuzione per ogni numero di thread si è calcolato la media su 5 run[4] per ciascuna stringa, calcolandone successivamente una media aritmetica tra le medie delle run delle varie stringhe.

[2] I test sono stati eseguiti su più stringhe per poter sfruttare la proprietà di equi-distribuzione delle stringhe K nello spazio dei risultati.

[3] Il numero di zeri dell'hash è stato impostato a 6 per velocizzare le run dell'applicazione presupponendo un aumento esponenziale del tempo di ricerca della soluzione all'umettare di questo valore.

[4] Questo per cercare di eliminare possibili variazioni relative all'ambiente di esecuzione.

3.2.2 Risultati



3.2.3 Analisi dei risultati

Dai test è visibile come il programma tragga molto vantaggio dalla frammentazione in più thread, pur non rimanendo sempre consistente. Questo è dovuto al fatto che pur essendo le stringhe distribuite in modo casuale nello spazio di ricerca, l'esecuzione del programma con un diverso numero di thread può modificare la distanza massima tra il punto di partenza del thread con la stringa corretta più vicina ad esso. Questo non è prevedibile a priori per tanto non vi è un modo per ottimizzare in maniera perfetta l'esecuzione del programma.

Dai test si può anche denotare come il content switch dei vari thread non risulta particolarmente pesante per l'esecuzione del programma[5] pur avendo un leggero effetto negativo sulle prestazioni generali del programma.

[5]Questo può essere dedotto dal fatto che il tempo di esecuzione non aumenta molto al superamento della soglia ottimale teorica del numero di thread $2 \cdot n_{core}$(nel test pari a 24).

In conclusione dal test il valore ottimale risulta essere 4 con 7, 12 e 13 anche molto vicini. Il test purtroppo non è però in grado di determinare un valore generale ottimale[6] ma è comunque in grado di darci una visone generale dell'andamento del problema e delle consistenze tra le varie stringhe testate.

I punti a minore variazione risultano essere 21 con 4, 13 e 12 a seguire. Ancora una volta i valori vicini al valore teorico ottimale sembrano essere i migliori, di conseguenza l'esecuzione standard del programma viene eseguito su $2 \cdot n_{core}$ della macchina.

4 Possibili ottimizzazioni

*TODO

[6] Il numero di stringhe analizzate risulta essere troppo piccolo rispetto allo spazio possibile di input.