Informatica - Area scientifica
Dipartimento di Scienze matematiche, informatiche e multimediali
Università di Udine

Progetto di Algoritmi (Prima parte)

Agnoletti Pierre (150426) agnoletti.pierre@spes.uniud.it
Da Re Davide (141976) dare.davide@spes.uniud.it
Gazzola Elia (147575) gazzola.elia@spes.uniud.it
Moro Martina (147592) moro.martina001@spes.uniud.it
Pasin Angelica (149479) pasin.angelica@spes.uniud.it

Anno accademico 2020/2021

Indice

Introduzione

Il nostro progetto consiste nella produzione di più programmi che risolvono il problema del calcolo della stima empirica della complessità al variare della dimensione dell'input e di eventuali altri parametri.
Nella prima parte del progetto viene richiesta l'implementazione di due algoritmi entrambi che risolvono lo stesso problema, ma aventi complessità differenti. Il problema da risolvere è il calcolo del periodo frazionario minimo di una stringa.

Periodo frazionario minimo: Il periodo frazionario minimo di una stringa s è il più piccolo intero i>0 che soddisfa la proprietà seguente:

$$\forall j = 1, ..., n - i \text{ vale che s(j)=s(j+i) con n=|s|}$$

Esempi di periodo frazionario minimo sono:

- abcabcab => 3 ("abc" si ripete 3 volte contando anche l'ultima tripletta contenente "ab" che può continuare con c)
- aba => 2 (stesse caratteristiche della precedente)

Per risolvere tale problema vengono implementati due algoritmi, con i seguenti tempi asintotici di esecuzione: PeriodNaive $\theta(n^2)$ e PeriodSmart $\theta(n)$.

Inoltre, abbiamo calcolato il valore della deviazione standard del tempo medio di esecuzione dei due algoritmi data una lunghezza delle stringhe da generare decisa a priori.

1 Algoritmi implementati

1.1 PeriodNaive

L'algoritmo utilizza un ciclo for con un intero i che varia da 1 a n, con n la lunghezza della stringa considerata. Tale algoritmo continua ad iterare fino a quando viene soddisfatta la proprietà mostrata in precedenza, in tal caso restituirà i (lunghezza del periodo minimo). Per verificare la propietà viene utilizzata la funzione java equals() per confrontare la congruenza tra il prefisso della stringa [0,n-i] e il suffisso [i, n].
L'implementazione di tale algorito richiede, nel caso pessimo, tempo quadratico nella lunghezza n della stringa.

1.2 PeriodSmart

L'algoritmo riduce il problema del calcolo del periodo frazionario minimo di s al problema della lunghezza massima di un bordo di s.
Un bordo di una stringa s è una qualunque stringa che sia allo stesso tempo prefisso e suffisso proprio di s.
Nell'implementazione il vettore dei bordi r viene istanziato a priori, con la lunghezza massima possibile. Questo fa in modo che il vettore non venga istanziato ad ogni esecuzione dell'algoritmo, evitando così che venga attivato il garbage collector, ciò creerebbe dei picchi nelle misurazioni dei tempi.
L'implementazione di tale algorito richiede, nel caso pessimo, tempo lineare nella lunghezza n della stringa.

2 Classi utili

2.1 Generatore di stringhe pseudocasuali

L'algoritmo genera una stringa di lunghezza "length", passata come parametro. Viene utilizzata inizialmente la funzione "ints" della classe "Random", tale funzione genera uno stream illimitato di interi compresi tra "leftLimit" e "rightLimit", i caratteri che comporranno le stringhe saranno compresi tra tali limiti, intesi come codici ASCII. La funzione "limit", andrà appunto a limitare la dimensione dello stream alla lunghezza della stringa da generare, avremo quindi un codice ASCII per ogni carattere della stringa. La funzione "collect" andrà infine a trasformare tale sequenza di codici ASCII in oggetto di tipo "String". E' stata eseguita la presa dei tempi anche per tale algoritmo vedendo che anche questo ha complessità lineare.

```
public class GenerazioneStringhe {

    public static String generaStriga(int length) {
        int leftLimit = 97; // letter 'a'
        int rightLimit = 99; // letter 'c'
        Random random = new Random();

        String generatedString = random.ints(leftLimit, rightLimit + 1)
          .limit(length)
          .collect(StringBuilder::new, StringBuilder::appendCodePoint,
              StringBuilder::append)
          .toString();

        return generatedString;
    }

}
```

2.2 Misurazione dei tempi

La classe MisurazioneTempi, visibile nel modulo VPL dedicato, è composta da cinque metodi:

1. **timesOnFile(Algortimo alg)**
   La funzione timesOnFile calcola un determinato numero di volte (nel nostro caso 100) il tempo medio di esecuzione di un determinato algoritmo. La scelta dell'algoritmo da eseguire viene effettuata tramite un'enumerazione passata come parametro alla funzione. La lunghezza delle stringhe da utilizzare (n) viene generata tramite una funzione esponenziale in i, dove i è il numero di iterazioni a cui il programma è arrivato (0<=i<100). Tale funzione si basa su due costanti A e B calcolate in modo che la lunghezza sarà 1000 quando i=0 e 500000 quando i=99.
   Ad ogni iterazione del ciclo for l'algoritmo scelto viene iterato con stringhe diverse di lunghezza n, finchè il suo tempo di esecuzione non supera il tempo minimo misurabile dall'elaboratore, calcolato tramite la risoluzione stimata R e tramite l'errore massimo relativo E = 0.001.
   Viene poi creata una stringa denominata "tmp" dove viene inserita la lunghezza delle stringhe processate seguita da uno spazio e dal tempo medio di esecuzione dell'algoritmo appena calcolato. Tale stringa viene poi scritta in un file di testo, opportunamente nominato in base all'algoritmo utilizzato.
   Se ci troviamo nella prima iterazione del ciclo for (prima riga da scrivere), viene abilitato il flag "create" nella funzione "writeFile", che andrà a creare da zero il file di destinazione, altrimenti la riga verrà scritta in coda al file. La funzione "writeFile" è inoltre in grado di gestire il caso in cui il file di destinazione è già stato utilizzato in un'esecuzione precedente del programma, il file esistente non viene sovrascritto, ma viene generato un nuovo nome per il file da utilizzare aggiungendo un'appendice progressiva al nome desiderato.

2. **standardDeviation(int length, Algoritmo alg)**
   Calcola la deviazione standard dei risultati di un determinato algoritmo, testato con un numero prefissato di stringhe di lunghezza predefinita. I risultati delle varie esecuzioni dell'algoritmo vengono salvati in un vettore, tale vettore viene successivamente passato alla funzione calcolaSD che calcola la deviazione standard di tutti i valori in esso contenuti. Length è la lunghezza prefissata delle stringhe e Alg è il tipo di algoritmo da utilizzare.

3. **getResolution()**
   Stima la risoluzione del clock di sistema utilizzando un ciclo while per calcolare l'intervallo minimo di tempo misurabile.

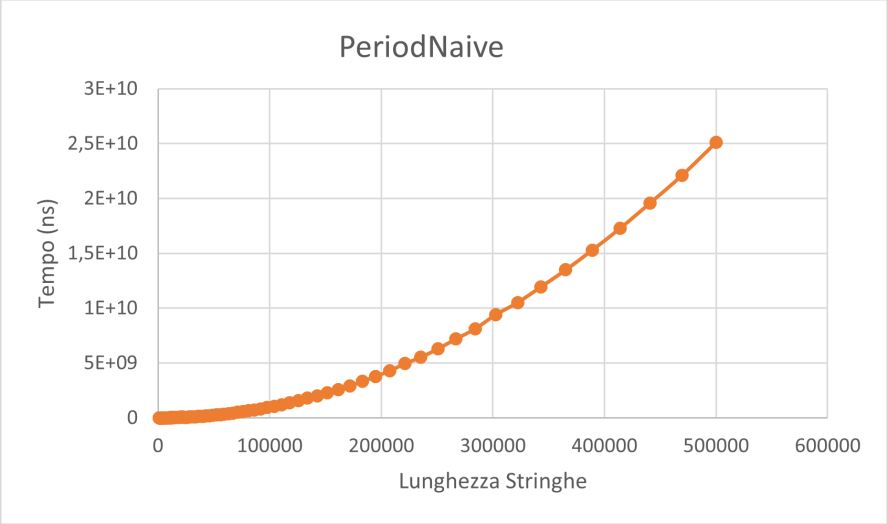
4. **calculateSD(double[ ] numArray)**
   Calcola la deviazione standard di un array di valori

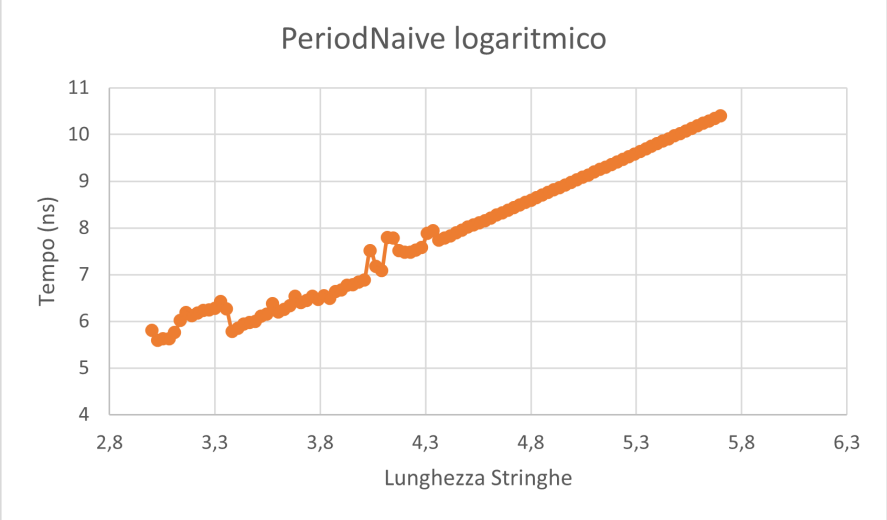
5. **writeFile(String path, String text, boolean create)**
   Scrive su file una stringa passata come parametro, è in grado di creare il file se esplicitamente richiesto. Se il file è gia esistente, modifica il nome del file finchè non ne trova uno inutilizzato.

3 Analisi dei tempi

3.1 Analisi PeriodNaive

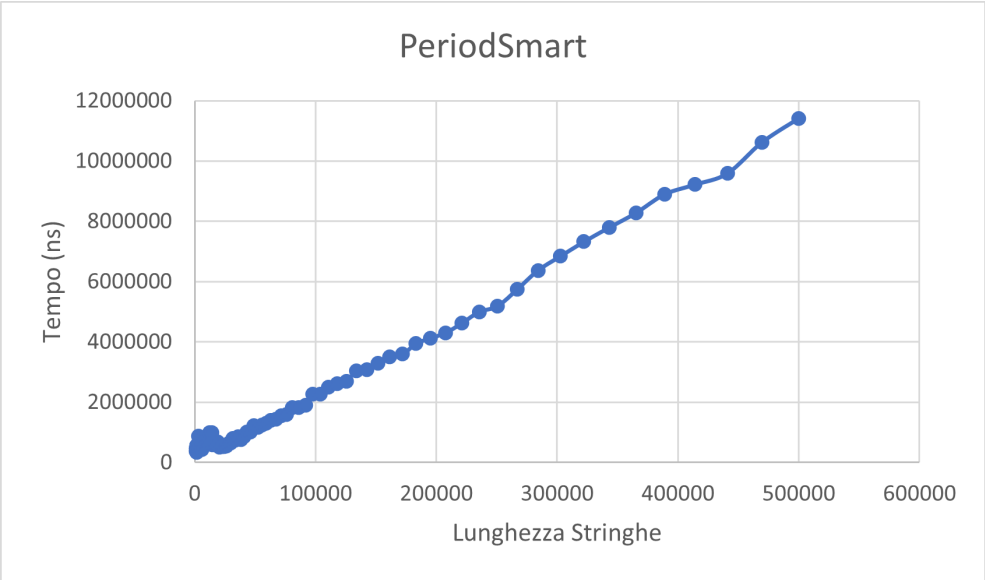
Riportiamo di seguito i grafici in scala normale e in scala logaritmica, generati dall'algoritmo del PeriodNaive dopo l'esecuzione di cento iterazioni. Come possiamo osservare dal primo grafico, all'aumentare della dimensione della stringa aumenta il tempo di esecuzione in maniera quadratica.

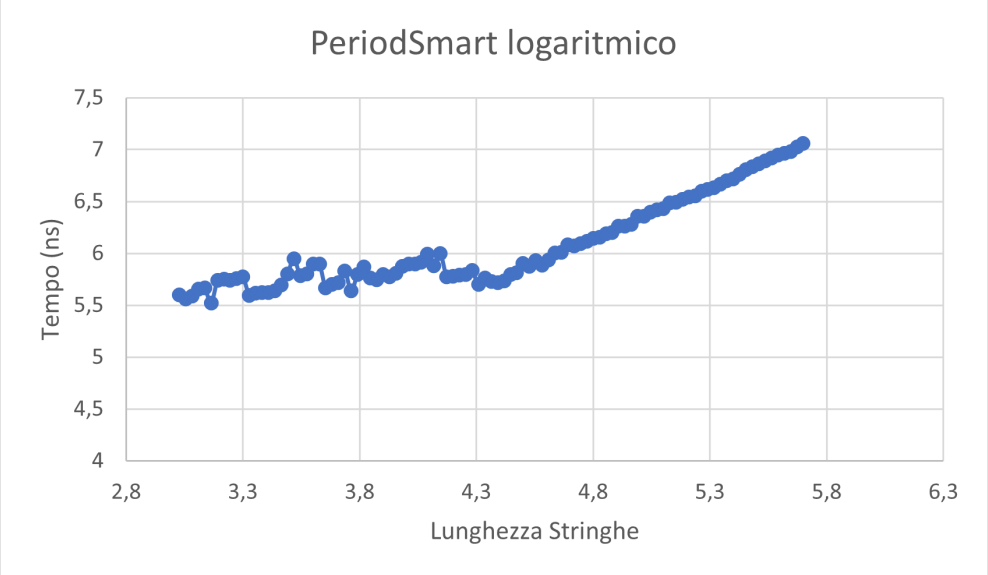




3.2 Analisi PeriodSmart

Riportiamo di seguito i grafici in scala normale e in scala logaritmica, generati dall'algoritmo del PeriodSmart dopo l'esecuzione di cento iterazioni. Come possiamo osservare dal primo grafico, l'algoritmo ha un andamento lineare all'aumentare della dimensione della stringa.





4 Conclusioni

Dalle osservazioni tratte in precedenza è possibile notare, dai grafici comparativi riportati in seguito, come l'algoritmo periodSmart sia più efficiente dell'algoritmo periodNaive all'aumentare della lunghezza della stringa passata in input agli algoritmi.