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

Prima parte del progetto di ASD

Enrico Martin martin.enrico@spes.uniud.it 145175
Luca Bazzetto bazzetto.luca@spes.uniud.it 144760
Andrea Bordignon bordignon.andrea001@spes.uniud.it 142295

Anno accademico 2020/2021

Indice

Capitolo 1

Consegna

Lo scopo del progetto di laboratorio è verificare che lo studente sia in grado di progettare, implementare e analizzare la complessità di un programma usando gli strumenti illustrati durante le lezioni del corso di algoritmi e strutture dati. I programmi prodotti dovranno risolvere alcuni problemi classici di natura computazionale e dovranno essere formalmente corretti. Ogni progetto di laboratorio consiste nella produzione di uno o più programmi che risolvano un problema computazionale assegnato, nella stima empirica della complessità di tali programmi al variare della dimensione dell'input e di eventuali altri parametri, e nella redazione di una relazione in cui si discutano puntualmente alcune scelte implementative e si faccia un'analisi della stima di complessità.

Di seguito sono riportate le indicazioni per la prima parte del progetto di laboratorio, da consegnare necessariamente prima dell'iscrizione all'esame finale di teoria.
La prima parte del progetto richiede l'implementazione di due algoritmi per il calcolo del periodo frazionario minimo di una stringa e l'analisi dei loro tempi medi di esecuzione. Il periodo frazionario minimo di una stringa s è il più piccolo intero p>0 che soddisfa la proprietà seguente:

- $\forall i = 1, \ldots, n - ps(i) = s(i + p)$

dove n denota la lunghezza della stringa s. Di seguito si riportano alcuni esempi di istanze di input

(stringa s) con il corrispondente output desiderato (periodi frazionario minimo di s):

- input=abcabcab output=3
- input=aba output=2
- input=abca output=3

I due algoritmi da implementare, denominati PeriodNaive e PeriodSmart, sono descritti di seguito (si veda anche l'Esercizio 11).

PeriodNaive
L'algoritmo utilizza un ciclo for con un intero p che varia da 1 a n. Alla prima iterazione che soddisfa la proprietà (Stella) l'algoritmo termina restituendo p. La verifica della proprietà (Stella) può essere implementata in uno dei due modi seguenti:

- utilizzando un ciclo secondario e controllando direttamente che s(i) = s(i+p) per ogni i=1,..., n-p
- oppure utilizzando le funzioni strcmp e strsub del linguaggio C (o analoghe funzioni per altri linguaggi) per confrontare la congruenza fra il prefisso S[1,n-p] e il suffisso S[p+1,n]

Entrambe le implementazioni richiedono, nel caso pessimo*, tempo quadratico nella lunghezza n della stringa

Nota: il primo metodo di implementazione dell'algoritmo naive potrebbe facilmente avere dei tempi medi di esecuzione lineari nella lunghezza n della stringa. Ciò significa che generando input in maniera pseudo-casuale e misurando i tempi di esecuzione, si potrebbero non rilevare particolari differenze con l'algoritmo PeriodSmart, anch'esso lineare in complessità. È anche possibile generare input in modo tale che la complessità misurata abbia un andamento super-lineare; tuttavia la costruzione di tali input e l'analisi di complessità non è banale ed è lasciata ai più volenterosi e curiosi!

PeriodSmart
Si consideri la seguente definizione: un bordo si una stringa s è una qualunque stringa t che sia, allo stesso tempo, prefisso proprio di s e suffisso proprio di s. Ad esempio, t=abab è un bordo di s=ababaababababab, ed è anche il bordo di s di lunghezza massima.

Si osservi quindi che p è un periodo frazionario di s se e solo se p=|s|-r ed r è la lunghezza di un bordo di s. Ciò permette di ridurre il problema del calcolo del periodo frazionario minimo di s al problema del calcolo della lunghezza massima di un bordo di s.
Per risolvere quest'ultimo problema si proceda per induzione, calcolando per ogni prefisso s[1...i], dal più corto al più lungo, la lunghezza r(i) del bordo massimo di s[1...i]. Per implementare il passo induttivo da i ad i+1, si consideri la sequenza r(i)>r(r(i))>r(r(r(i)))>...>rk(i)=0 e si osservi che nel calcolo di r(i+1) solamente i due casi seguenti possono darsi:

- Per qualche indice $j \leq k$ vale l'uguaglianza s[i+1]=s[$r^j(i)$+1] in tal caso r(i+1)=$r_j(i)$+1, dove j è il primo indice per cui vale la suddetta uguaglianza;
- Non esiste alcun indice $j \leq k$ che soddisfi l'uguaglianza s[i+1]=s[$r^j(i)$+1]; in tal caso r(i+1)=0.

Per calcolare la lunghezza massima di un bordo di s si consiglia quindi di usare una procedura iterativa con un array di supporto che raccolga i valori r(i), calcolati progressivamente e in modo induttivo, per ogni i=1...n.

Modalità di consegna
Si richiede di implementare in un linguaggio a scelta (ad esempio, C, C++, Java) entrambi gli algoritmi descritti sopra, in modo che siano formalmente corretti. Per agevolare la verifica di correttezza da parte del docente sono stati predisposti tre moduli "Virtual Programming Laboratory" (VPL) da utilizzare per caricare il codice degli algoritmi. Una condizione necessaria alla valutazione dell'elaborato è il superamento di tutti i test previsti, per tutti e tre gli algoritmi.

Si richiede di implementare inoltre un programma per la misurazione dei tempi medi di esecuzione dei due algoritmi, al variare della lunghezza n della stringa fornita in input.

- La lunghezza n della stringa deve essere compresa in un range di valori fra 1000 e 500000, con una distribuzione preferibilmente esponenziale. Ad esempio, n potrebbe essere definito da una funzione esponenziale in i del tipo |A * Bi|, dove i=0,1,2,...,99 e A e B sono costanti in virgola mobile calcolate opportunamente in modo da avere che n=1000 quando i=0 ed n=500000 quando i=99.
- La stringa s di lunghezza n deve essere generata su un alfabeto binario o ternario (es. 'a','b'), utilizzando uno dei metodi seguenti (a scelta):
  - le lettere s(i) della stringa, per ogni i=1,...,n, vengono generate in modo pseudo-casuale e indipendentemente una dall'altra;
  - un parametro q compreso fra 1 ed n viene generato in modo pseudo-causale; in seguito vengono generate le lettere s(i) per ogni i=1,...,q, seguendo il metodo 1.; infine la parte rimanente di s viene generata seguendo la formula s(i)=s((i-1) mod q+1) per ogni i=q+1,...,n;
  - una variante del metodo 2. viene applicata, con l'eccezione che ad s(q) viene assegnata una lettera special (ad esempio, 'c') differente da tutte quelle generate in modo pseudo-casuale;
  - in aggiunta (e opzionalmente) è possibile misurare i tempi di esecuzione per una costruzione deterministica di s che possa fungere da caso pessimo nell'analisi di complessità dell'algoritmo PeriodNaive.

Opzionalmente, è possibile studiare la distribuzione di probabilità del periodo frazionario minimo per le stringhe generate con i metodi sopra indicati e per un n fissato a piacere (es. 100).

Per ogni stringa s di lunghezza n generata, occorre quindi stimare il tempo di esecuzione per ognuno dei due algoritmi, PeriodNaive o PeriodSmart. Si richiede una stima del tempo medio di esecuzione al variare della lunghezza n della stringa che garantisca un errore relativo massimo pari a 0.001. A tal fine si procede nel modo seguente:

- Per tutte le misurazioni del tempo trascorso è necessario utilizzare un clock di sistema monotono.
- Il primo passo consiste nello stimare la risoluzione del clock di sistema, utilizzando un ciclo while per calcolare l'intervallo minimo di tempo misurabile. A tale scopo è possibile utilizzare uno dei seguenti frammenti di codice forniti dal professore.
- Successivamente, in funzione della risoluzione stimata R e dell'errore relativo massimo ammissibile (E=0.001), si calcola il tempo minimo misurabile $Tmin = R * (1E + 1)$.
- Infine, si utilizza un ciclo while per iterare l'esecuzione dell'algoritmo, misurando il tempo trascorso dall'inizio dell'iterazione, fino a quando tale tempo non risulti superiore a Tmin. Il tempo medio di esecuzione per una singola istanza di input sarà quindi dato dal rapporto fra il tempo trascorso e il numero di iterazioni eseguite.

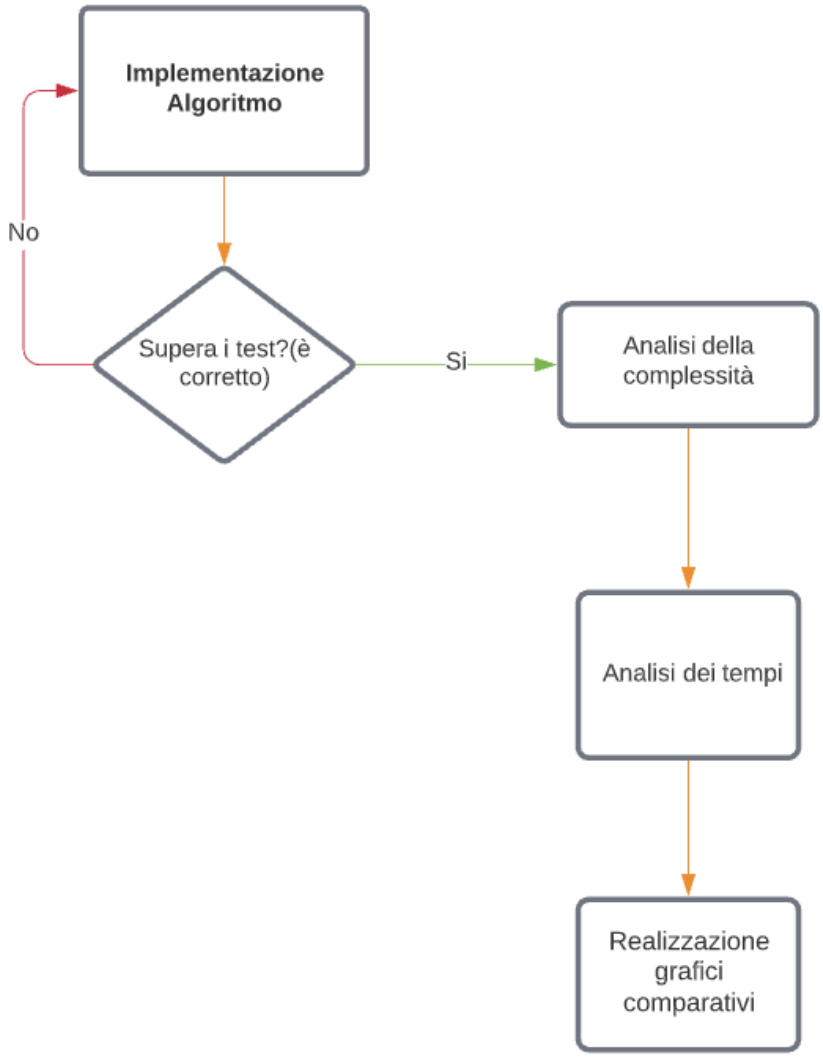
  Nel caso si utilizzi il linguaggio di programmazione Java, occorre prestare attenzione a non allocare ripetutamente grandi strutture dati (esempio, array o stringhe) in modo dinamico (ad esempio, con l'istruzione new). Tale pratica potrebbe esaurire in breve tempo la memoria RAM disponibile e attivare il garbage collector, creando picchi nei tempi di esecuzione misurati.
- Opzionalmente, è possibile stimare anche la deviazione standard dei tempi di esecuzione rispetto al tempo medio, in funzione del parametro n. In tal caso si procederà effettuando un certo numero di misurazioni (ad esempio, un centinaio) sui tempi di esecuzione per una lunghezza n fissata, facendo però variare la stringa di lunghezza n generata in modo pseudo-casuale. Ogni misurazione sarà effettuata seguendo il metodo sopra descritto, e il risultato di tale misurazione viene memorizzato in un opportuno array. Una volta popolato l'array con le misurazioni, si procederà nel modo classico calcolando media e scarto quadratico medio.

Capitolo 2

Approccio al progetto

Il primo approccio al progetto è stato quello di eseguire un'attenta lettura della consegna al fine di evidenziare in primis le varie richieste e contemporaneamente di effettuare una valutazione preliminare sugli obiettivi che sarebbero potuti risultare più difficili da implementare.

Successivamente è stato realizzato un diagramma, qui di seguito riportato, che sintetizza la metodologia con cui abbiamo realizzato il progetto. Partendo dall'implementazione degli algoritmi, alla verifica della loro correttezza per arrivare poi alla realizzazione dei grafici riportati poi nella relazione.



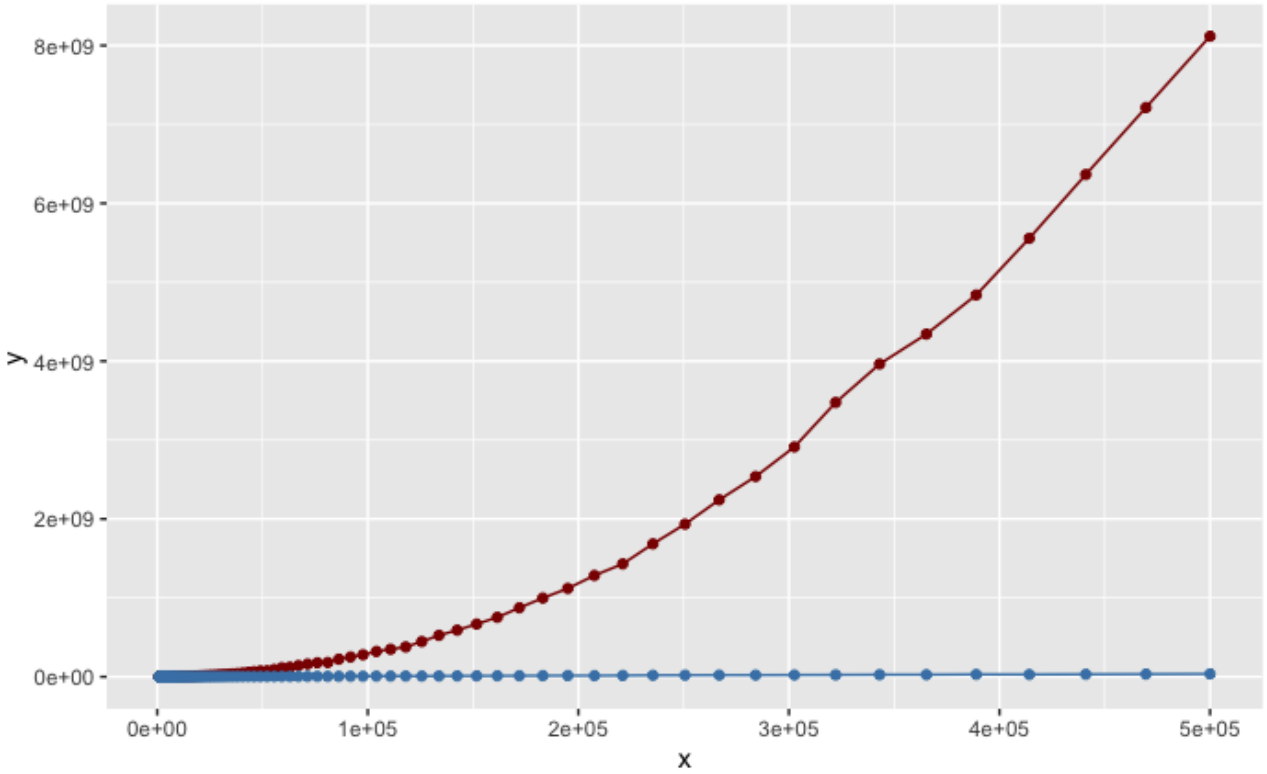
Capitolo 3

Analisi dei Tempi

In questa sezione della relazione si riporteranno i grafici delle misurazioni dei tempi.
I grafici mettono sempre in relazione i due algoritmi, distinguibili con il colore rosso per il PeriodNaive e con il colore blu per il PeriodSmart.

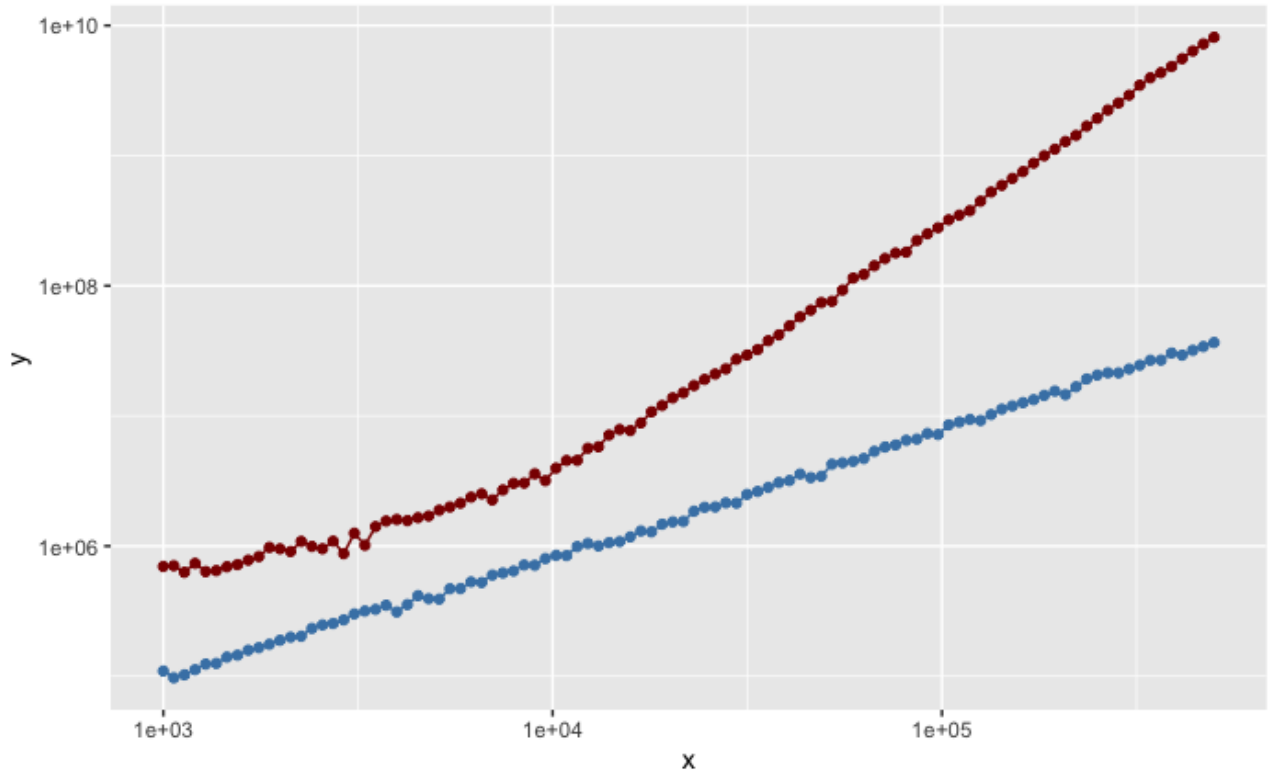
3.1 Grafico 1

Il primo grafico mostra il comportamento dei due algoritmi in funzione di n.
Nell'asse delle ascissa troviamo n mentre nell'ordinata il tempo espresso in nanosecondi
Osservando il grafico ci si accorge che i punti iniziali presentano una maggior densità e diventano sempre meno densi al crescere di n. Questo perchè i campioni sono distribuiti con una distribuzione esponenziale.



3.2 Grafico 2

Applichiamo al Grafico 1 una scala logaritmica per ogni asse. I campioni così risultano essere distribuiti in maniera ovviamente uniforme.
Da qui si nota la convenienza (traducibile in efficienza) del PeriodSmart rispetto al PeriodNaive, ad eccezione dei primi valori dove si evidenziano dei costi di overhead costanti.



Capitolo 4

Conclusioni

Al termine dell'implementazione degli algoritmi e dell'analisi dei tempi di ciascuno di essi si evince che il PeriodSmart è più efficiente rispetto al PeriodNaive.

Inoltre, come opzionalmente richiesto e per entrambi gli algoritmi, abbiamo stimato la deviazione standard dei tempi di esecuzione rispetto al tempo medio, in funzione del parametro n.
Il parametro n viene fissato a 70 e lunghezza stringa, di conseguenza, risulta pari a 76050.

I risultati per il PeriodSmart sono:

- Media: 0.00429699
- Deviazione standard: 0.000187

I risultati per il PeriodNaive sono:

- Media: 0.187628
- Deviazione standard: 0.00483076