UNIVERSITA’ DEGLI STUDI DI UDINE



**Progetto di Algoritmi e Strutture Dati**

**(Prima parte)**

*Autori*: *Professori:*

**Dalla Torre Josè Zito Piazza Carla**

123456 **Puppis Gabriele**

[123456@spes.uniud.it](mailto:123456@spes.uniud.it)

**Tracanelli Alessandra**

143342

[143342@spes.uniud.it](mailto:143342@spes.uniud.it)

**Indice**

**1 Il problema 3**

1.1 Enunciato3

**2 Soluzione Naive3**

2.1 Significato 3

2.2 Metodo3

2.3 Pseudocodice4

2.4 Calcolo dei tempi 4

**3 Soluzione Smart5**

3.1 Significato 5

3.2 Metodo5

3.3 Pseudocodice5

3.4 Calcolo dei tempi 6

**4 Considerazioni complessive 6**

1. **Il problema** 
   1. **­Enunciato**

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

*(⋆) ∀i=1,…, n−ps(i)=s(i+p)*

*dove n denota la lunghezza della stringa s.*

Il problema richiede l’implementazione di due algoritmi (*PeriodNaive* e *PeriodSmart*) per il calcolo del *periodo frazionario minimo* di una stringa e l'analisi dei loro tempi medi di esecuzione.

1. **Soluzione Naive**
   1. **Significato**

Naive (o naif) è un aggettivo il cui nome deriva da un termine francese. Sta ad indicare un tipo di ragionamento istintivo, primitivo ed elementare. Successivamente verrà presentato l’algoritmo naïve relativo al problema in questione, più semplice, ma anche meno efficiente.

* 1. **Metodo**

Il metodo utilizzato dall’algoritmo naïve è il seguente:

l’algoritmo utilizza un ciclo *for* con un numero intero *p* i cui valori spaziano tra 1 e *n*. Alla prima iterazione che soddisfa la proprietà *(⋆)* l’algoritmo termina restituendo il valore *p.*

E’ stato poi implementato un secondo ciclo for interno per controllare che la proprietà *(⋆)* sia soddisfatta, o no, da una certa stringa che controlla direttamente che s(i) = s(i+p) per ogni i=1,…,n−p.

**2.3 Pseudocodice**

Di seguito lo pseudocodice che realizza l’algoritmo naïve:

for p=1 to n {

for j=1 to n-p {

if (j == n-p)

return p

}

}

return n

Complessità: O(n2)

**2.4 Calcolo dei tempi**

**3 Soluzione Smart**

**3.1 Significato**

Si consideri la seguente definizione: un *bordo* di una stringa *s* è una qualunque stringa *t* che sia, allo stesso tempo, prefisso proprio e suffisso proprio di *s.*

Il *prefisso* di una stringa c0c1 … cm-1 è una sottostringa iniziale (eventualmente vuota) c0 … ck-1 (con 0 ≤ k ≤ m) della stringa; k è la lunghezza del prefisso.

Il *suffisso* di una stringa c0c1 … cm-1 invece è una sottostringa finale (eventualmente vuota) cm-k … cm-1 con 0 ≤ k ≤ m della stringa; k è la lunghezza del suffisso.

Il *prefisso* o *suffisso* *proprio*: è un prefisso o suffisso che non coincide con l’intera stringa (è più corto), cioè k < m.

Nota: la stringa vuota è un bordo di qualunque stringa; la stringa vuota non ha bordo.

Si osservi quindi che *p* è un periodo frazionario di una stringa *s* se e solo se *p = |s| - r,* dove *r* è la lunghezza di un bordo di *s.* Il problema del calcolo del periodo frazionario minimo di una stringa s viene quindi ridotto al calcolo della lunghezza massima di un bordo di s.

**3.2 Metodo**

Per realizzare l’algoritmo *PeriodSmart* si è proceduto 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 calcolare la lunghezza massima di un bordo di *s* si è usata 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.

**3.3 Pseudocodice**

for (int i = 1; i <= n-1; i++) //per calcolare r[i+1]

{

int z=r[i];

while ((z>0) &&(s[i+1] != s[z+1]))

{

z=r[z];

}

if (s[i+1]==s[z+1])

{

r[i+1]= z+1;

}else{

r[i+1]=0;

}

return n - r[n];

}

Complessità: Θ(n)

**3.4 Calcolo dei tempi**

**4 Considerazioni complessive**