**MATh.en.JEANS**

# SOSTITUZIONI

**Liceo Scientifico Statale "E. Curiel"**

Caterina Alessi, Eleonora Filira, Matteo Forin, Lorenzo Gamba, Mircea Muntean, Stefano Pietrogrande, Emanuele Quaglio, Marco Venuti, Federico Vettore.

Prof. Giorgio Ciociano, Prof. Alberto Zanardo, Prof. Riccardo Colpi.

Padova, 01/05/16

Sia $A = \{a, b\}$ un alfabeto e si considerino le stringhe $m$ con esso costruite.

Indichiamo con $\sigma$ una sostituzione, una funzione cioè che associa a $m$ una parola $m' = \sigma(m)$.
Ad esempio, se $\sigma: \begin{cases} a \to ab \\ b \to a \end{cases}$, allora: $abba \to abaaab$, $aa \to abab$.

Nell'esempio precedente, potete trovare una stringa $m$ invariante rispetto a $\sigma$? Data una $\sigma$ qualsiasi, è sempre possibile trovare invarianti? È possibile costruire un algoritmo che le generi?

**1. Caratterizzazione $\sigma \rightarrow$ invariante**

Data una sostituzione $\sigma$ generica ($A \rightarrow N$; $B \rightarrow N'$; $C \rightarrow N''$... con $N$, $N'$, $N''$ stringhe di lettere dell'alfabeto usato), possiamo affermare:

(1) Una $\sigma$ qualsiasi applicata ad una sequenza di lettere $S$ qualsiasi darà una sequenza $S'$, definita e unica (essendo $\sigma$ una funzione):

$$\text{e.g.: } \sigma: \begin{cases} A \rightarrow AB \\ B \rightarrow A \end{cases} : ABA \rightarrow ABAAB;\ BBA \rightarrow AAAB$$

(2) Condizione necessaria (e sufficiente se si ammettono stringhe di lunghezza infinita) per avere una stringa invariante rispetto a $\sigma$ è che la trasformazione di almeno una delle lettere dell'alfabeto cominci con la stessa lettera, dopo l'applicazione di $\sigma$.

$$A \rightarrow AN\ e/o\ B \rightarrow BN'\ e/o\ C \rightarrow CN''$$

(con $N$, $N'$, $N''$ sequenze di lettere dell'alfabeto usato).

Dimostrazione per assurdo:

Sia:

$$\sigma: \begin{cases} A \rightarrow CN \\ B \rightarrow AN' \\ C \rightarrow BN'' \\ \quad ... \end{cases}$$

Ora qualsiasi lettera si scelga come lettera iniziale della stringa $S$, $\sigma(S)$ inizierà con una lettera diversa e non sarà uguale alla parola originaria, non costituendo dunque un invariante.

Quindi condizione necessaria di $\sigma$ per avere un invariante è avere almeno una lettera la cui variazione cominci per la lettera stessa.

Se una o più lettere verificano questa condizione, allora per ciascuna di esse si può procedere con l'algoritmo illustrato nella sezione 3. per formare una (diversa) parola invariante.

(3) Se almeno una delle lettere è invariante rispetto a $\sigma$ ($\sigma(\mathrm{L}) = \mathrm{L}$) allora vi sono infinite stringhe invarianti, ossia tutte le possibili sequenze di lettere ottenute dalla concatenazione di $n$ lettere L, con $n \in \mathbb{N} \cup \{+\infty\}$:

$$\mathrm{L} \rightarrow \mathrm{L};\ \mathrm{LL} \rightarrow \mathrm{LL}\ ;\ \mathrm{LLL} \rightarrow \mathrm{LLL}$$

## 2. Caso $A \to AB$, $B \to A$

$$\sigma(c) := \begin{cases} AB \; se \; c = A \\ A \quad se \; c = B \end{cases}$$

Osserviamo che σ prende in input un singolo carattere mentre restituisce una stringa composta da 1 o 2 caratteri. Dobbiamo dunque definire una estensione di σ che ci permetta di applicare la trasformazione a una stringa più lunga di un carattere.

Definiamo innanzi tutto le operazioni tra stringhe e in seguito $\bar{\sigma}$.

Operazioni tra stringhe

$S_1 \cdot S_2$: concatenazione tra stringhe;

Π: "concatenatoria" di stringhe;

|S|: lunghezza della stringa S;

S[i]: lettera i-esima della stringa S, $i \geq 0$.

Estensione di σ

$$\bar{\sigma}(S) := \prod_{i=0}^{|S|-1} \sigma(S[i])$$

$\bar{\sigma}$ prende in input una stringa, applica la trasformazione ai singoli caratteri che la compongono e concatena dunque tutti gli output.

Stringa invariante: approccio euristico

Si noti innanzi tutto che se la parola $m$ cominciasse per $B$, la sua trasformazione restituirebbe come primo carattere $A$, il che non verifica la condizione di invarianza.

Dunque, condizione necessaria per l'invarianza è che $m$ cominci per $A$.

Dimostrazione

| **Ipotesi: $\bar{\sigma}(m) = m$** | | **Tesi: $m[0] = A$** |
|---|---|---|
| (i) | $m[0] \neq A$ | Per assurdo. |
| (ii) | $m[0] = B$ | Poiché si dispone dell'alfabeto {A, B}. |
| (iii) | $\bar{\sigma}(m[0]) = \bar{\sigma}(B) = AB$ | Applicazione di $\bar{\sigma}$. |
| (iv) | $\bar{\sigma}(m)[0] = m[0]$ | Segue dall'ipotesi (proprietà delle stringhe). |
| (v) | $A = B$ | Segue da (iv), da (iii) e da (ii). |

Assurdo ■

La stringa deve cominciare per $A$, lettera la cui trasformazione è $AB$. Dunque la lunghezza complessiva della parola, una volta trasformata, deve necesariamente aumentare (anche ipotizzando che le lettere seguenti la prima $A$ fossero tutte $B$, vi sarebbe comunque una lettera in eccesso all'inizio della parola), quindi non vi possono essere invarianti finite per questa sostituzione. Proviamo dunque a costruire una parola invariante infinita.

Sappiamo che $m$ comincia per $A$: applicando la trasformazione, otteniamo $AB$, una stringa diversa da quella di partenza:

| $m$ | $A$ |
|---|---|
| $\bar{\sigma}(m)$ | $AB$ |

Possiamo dunque continuare la costruzione della parola concatenandole in coda una $B$ in maniera tale da eguagliare la stringa nella seconda riga con quella nella prima:

| $m$ | $AB$ |
|---|---|
| $\bar{\sigma}(m)$ | $AB$ |

Così facendo, però, si dovrà accodare alla seconda riga la trasformazione corrispondente alla $B$ appena aggiunta alla prima riga:

| $m$ | $AB$ |
|---|---|
| $\bar{\sigma}(m)$ | $ABA$ |

Il processo si deve dunque ripetere all'infinito, per ottenere due stringhe illimitate tali che a ogni lettera della stringa non trasformata corrisponda la stessa lettera nella stringa trasformata.

| $m$ | $ABAABABAABA\ldots$ |
|---|---|
| $\bar{\sigma}(m)$ | $ABAABABAABAABABAAB\ldots$ |

Si noti che l'incremento che di volta in volta si va a riportare alla riga superiore è, ad eccezione delle prime due lettere, la trasformazione dell'incremento precedente.

Dunque, considerando una sola riga, possiamo suddividere la parola $m$ in sottostringhe in cui ognuna, ad eccezione delle prime due, è la trasformazione di quella precedente. Chiamiamo queste sottostringhe $a_n$:

| $m$ | $a_0$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ | … |
|---|---|---|---|---|---|---|---|---|
| | $A$ | $B$ | $A$ | $AB$ | $ABA$ | $ABAAB$ | $ABAABABA$ | … |

Si definisce dunque $a_n$ ricorsivamente:

$$a_n := \begin{cases} a_0 = A \\ a_1 = B \\ a_n = \bar{\sigma}(a_{n-1}) \end{cases}$$

La parola $m$ è definita come la concatenazione delle infinite stringhe $a_n$:

$$m := \prod_{i=0}^{\infty} a_i$$

Dimostrazione dell'invarianza di $m$

**Ipotesi: Definizioni precedenti** **Tesi: $\bar{\sigma}(m) = m$**

$$\bar{\sigma}(m) = \bar{\sigma}\left(\prod_{i=0}^{\infty} a_i\right) = \prod_{i=0}^{\infty} \bar{\sigma}(a_i) =$$ Definizione di $m$ e linearità di $\sigma$.

$$= \bar{\sigma}(a_0) \cdot \bar{\sigma}(a_1) \cdot \prod_{i=2}^{\infty} \bar{\sigma}\,(a_i) =$$ Esplicitati i primi termini della concatenazione.

$$= AB \cdot A \cdot \prod_{i=2}^{\infty} a_{i+1} =$$ Applicazione di $\bar{\sigma}$.

$$= ABA \cdot \prod_{i=3}^{\infty} a_i =$$ Concatenazione e shift dell'indice.

$$= a_0 \cdot a_1 \cdot a_2 \cdot \prod_{i=3}^{\infty} a_i =$$ Segue da definizione di $a_n$ e dal calcolo manuale di $a_2$.

$$= \prod_{i=0}^{\infty} a_i = m$$ Raccolti nel segno di concatenazione i primi termini; Definizione di m.

Dunque, $\bar{\sigma}(m) = m$ ■

Interpretazione con la sequenza di Fibonacci

Vale, come prima, la proprietà $\sigma(MN) = \sigma(M)\sigma(N)$.

(e.g.: $\sigma(ABAAB) = ABAABABA$
$= \sigma(ABAA) \cdot \sigma(B) = ABAABAB \cdot A = ABAABABA$ )

Ossia per trasformare una stringa si può scomporre quest'ultima in più sequenze a ciascuna delle quali si applica in seguito $\sigma$.

| $a_n$ | $a_0$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ | … |
|---|---|---|---|---|---|---|---|---|
| $m$ | $A$ | $B$ | $A$ | $AB$ | $ABA$ | $ABAAB$ | $ABAABABA$ | … |

Si noti che $a_3$, cioè AB, è composto nell'ordine dalla concatenazione di A e B, cioè $a_2$ e $a_1$, i due blocchi precedenti.

Dunque: $a_3 = a_2 a_1$ *(Passo base)*

Ciò è vero anche per i blocchi successivi, ad esempio:

$$a_4 = ABA = AB \cdot A = a_3 a_2$$

$$a_5 = ABAAB = ABA \cdot AB = a_4 a_3$$

Completiamo ora la dimostrazione con il *passo induttivo*, mostrando che $a_n = a_{n-1}a_{n-2}$ è vera $\forall n \geq 3$:

Ipotesi: $a_n = a_{n-1}a_{n-2}$ Tesi: $a_{n+1} = a_n a_{n-1}$

$$a_{n+1} = \sigma(a_n) = \sigma(a_{n-1}a_{n-2}) = \sigma(a_{n-1})\sigma(a_{n-2}) = a_n a_{n-1}$$

Quindi, $a_n = a_{n-1}a_{n-2} \forall n \geq 3$ ■

Ma questa relazione è analoga alla definizione della successione di Fibonacci ($F_n = F_{n-1} + F_{n-2} \forall n \geq 3$: l'elemento n-esimo è la somma dei due elementi che lo precedono), poiché $\forall n \geq 3$ ogni blocco è la concatenazione dei due precedenti (prima l'(n-1)-esimo e poi l'(n-2)-esimo).

Poiché $|MN| = |M| + |N|$, si può affermare che $|a_n| = |a_{n-1}| + |a_{n-2}|$, che è proprio la definizione di $F_n$.

Ad esempio, $a_1$ e $a_2$ contengono solo 1 lettera mentre $a_3$ ne contiene 2 (cfr. le prime tre cifre della successione di fibonacci: 1,1,2).

Algoritmo in C++ (http://cpp.sh/5jxx)

```cpp
// cpp.sh/5jxx

#include <iostream>

using namespace std;

string sigma (char l)
{
  if (l == 'A') return string("AB");
  if (l == 'B') return string("A");
}

string sigma_segnato (string str)
{
  string toReturn = "";
  for (unsigned i=0; i <= str.length()-1; i++)
    toReturn.append( sigma(str[i]) );
  return toReturn;
}

string a (unsigned n)
{
  if (n == 0) return string("A");
  if (n == 1) return string("B");
  if (n >= 2) return sigma_segnato(a(n-1));
}


int main()
{
  string P;
  for (int i=0; i<8 /*estremo superiore*/; i++)
    P.append (a(i));
  cout << "m   : " << P << endl;
  cout << "s(m): " << sigma_segnato (P);

  return 0;
}
```

Output:

```
P    : ABAABABAABAABABAABABAABAABABAABAAB
s(P): ABAABABAABAABABAABABAABAABABAABAABABAABABAABAABABAABABA
```

## 3. Algoritmo per la costruzione di invarianti infinite

Il problema in questione consiste nella generalizzazione di quello posto nella sezione 2..

Anche qui, condizione necessaria all'invarianza di $m$ è che $m$ cominci con una lettera la cui trasformazione comincia con la stessa lettera.

Si può ripercorrere la costruzione della parola fatta nella sezione 2..

Supponiamo che la lettera iniziale generica $K$ verifichi la condizione necessaria all'invarianza rispetto alla sostituzione generica $\sigma$. Avremo dunque:

| $m$ | $K$ |
|---|---|
| $\bar{\sigma}(m)$ | $\bar{\sigma}(K)$ |

Ora bisogna aggiungere alla riga superiore i caratteri che le mancano per essere uguale a quella inferiore, ossia $\bar{\sigma}(K)$ cui è stato rimosso il carattere iniziale $K$. Indichiamo la stringa ottenuta $\bar{\sigma}(K) - K$. L'operatore "$-$", dunque, elimina la lettera $K$ a partire dall'inizio della stringa (la parola comincia con $K$ per definizione):

| $m$ | $K \cdot (\bar{\sigma}(K) - K)$ |
|---|---|
| $\bar{\sigma}(m)$ | $\bar{\sigma}(K)$ |

Ora si procede concatenando alla seconda riga la trasformazione dell'incremento appena aggiunto alla prima, $\bar{\sigma}(\bar{\sigma}(K) - K)$, e iterando all'infinito il processo.

Come nella sezione 2., riportiamo il tutto su una riga singola suddividendo ogni passaggio in sottostringhe $a_n$:

| | | | | | | |
|---|---|---|---|---|---|---|
| $m$ | $a_0$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ | … |
| | $K$ | $\bar{\sigma}(K) - K$ | $\bar{\sigma}(\bar{\sigma}(K) - K)$ | $\bar{\sigma}(\bar{\sigma}(\bar{\sigma}(K) - K))$ | $\bar{\sigma}^3(\bar{\sigma}(K) - K)$ | … |

Si ricorda la notazione: $f^n(x_0) \equiv f(x_{n-1})$.

Si definisce dunque $a_n$ ricorsivamente:

$$a_n := \begin{cases} a_0 = K \\ a_1 = \bar{\sigma}(a_0) - a_0 \\ a_n = \bar{\sigma}^{n-1}(a_1) \equiv \bar{\sigma}(a_{n-1}) \end{cases}$$

La parola $m$ è definita come la concatenazione delle infinite stringhe $a_n$:

$$m := \prod_{i=0}^{\infty} a_i$$

Algoritmo in C++ (http://cpp.sh/37gh2)

```cpp
// cpp.sh/37gh2
#include <iostream>
using namespace std;

string sigmaA, sigmaB, K;

string sigma (char l)
{
    if (l == 'A') return sigmaA;
    if (l == 'B') return sigmaB;
}


string sigma_segnato (string str)
{
    string toReturn = "";
    for (unsigned i=0; i <= str.length()-1; i++)
        toReturn.append( sigma(str[i]) );
    return toReturn;
}


string a (unsigned n)
{
    if (n == 0) return K;
    if (n == 1) return sigma_segnato(K).erase(0,1);
    if (n >= 2) return sigma_segnato(a(n-1));
}


int main()
{
    cout << "Definire sostituzione\n"
         << "A -> ";
    cin >> sigmaA;
    cout << "B -> ";
    cin >> sigmaB;
    cout << "Carattere iniziale (deve valere la condizione necessaria all'invarianza): ";
    cin >> K;

    for (unsigned i=0; i<sigmaA.length(); i++) // Tutto in maiuscolo
        sigmaA[i] = toupper(sigmaA[i]);
    for (unsigned i=0; i<sigmaB.length(); i++)
        sigmaB[i] = toupper(sigmaB[i]);
    for (unsigned i=0; i<K.length(); i++)
        K[i] = toupper(K[i]);

    cout << "Invariante:\n";

    string P = "";
    for (int i=0; i<5 /*estremo superiore*/; i++)
        P.append (a(i));

    cout << "m   : " << P << endl;
    cout << "s(m): " << sigma_segnato (P);

    return 0;
}
```

Esempi:

```
Definire sostituzione
A -> ABA
B -> BA
Carattere iniziale (deve valere la condizione necessaria all'invarianza): A
Invariante:
m   : ABABAABABAABAABABAABA
s(m): ABABAABABAABAABABAABABAABAABABAABAABABAABABAABAABABAABA
```

```
Definire sostituzione
A -> ABA
B -> BA
Carattere iniziale (deve valere la condizione necessaria all'invarianza): B
Invariante:
m   : BAABAABABAABA
s(m): BAABAABABAABAABABAABABAABAABABAABA
```

```
Definire sostituzione
A -> BABA
B -> BAA
Carattere iniziale (deve valere la condizione necessaria all'invarianza): B
Invariante:
m   : BAABABABABABAABABABAABABABAABABABAABA
s(m): BAABABABABABAABABABAABABABAABABABAABABABAABABABAABABABAABABABA…
```

## 4. $\sigma$ per invarianti finite

| | |
|---|---|
| $m$ | Parola finita iniziale; |
| $A=\{a,b\}$ | Alfabeto di cui è composta $m$; |
| $\sigma(m)$ | Parola trasformata; |
| $x,y,n$ | Indicano quante volte viene ripetuta la lettera, o la stringa di lettere, all'interno della parola; |
| $+$ | Rappresenta la concatenazione fra le lettere; |
| $\Lambda$ | Stringa nulla. |

Devo trovare le sostituzioni $\sigma$ tali che $m=\sigma(m)$:

<table>
<tr><td colspan="2">$xa+yb=m=\sigma(m)$</td></tr>
<tr><td>$\sigma:\begin{Bmatrix}a\to a\\ b\to b\end{Bmatrix}\quad xa+yb\to xa+yb$<br>$\sigma:\begin{Bmatrix}a\to a\\ b\notin m\end{Bmatrix}\quad xa\to xa$<br>$\sigma:\begin{Bmatrix}a\notin m\\ b\to b\end{Bmatrix}\quad yb\to yb$</td><td>$\sigma:\begin{Bmatrix}a\to \Lambda\\ b\to m\end{Bmatrix}\quad xa+yb\to xa+yb$<br>$\sigma:\begin{Bmatrix}a\to m\\ b\to \Lambda\end{Bmatrix}\quad xa+yb\to xa+yb$</td></tr>
<tr><td>$x(a+b)=m=\sigma(m)$</td><td>$x(b+a)=m=\sigma(m)$</td></tr>
<tr><td>$\sigma:\begin{Bmatrix}a\to ab\\ b\to \Lambda\end{Bmatrix}\quad x(a+b)\to x(a+b)$<br>$\sigma:\begin{Bmatrix}a\to \Lambda\\ b\to ab\end{Bmatrix}\quad x(a+b)\to x(a+b)$</td><td>$\sigma:\begin{Bmatrix}a\to ba\\ b\to \Lambda\end{Bmatrix}\quad x(b+a)\to x(b+a)$<br>$\sigma:\begin{Bmatrix}a\to \Lambda\\ b\to ba\end{Bmatrix}\quad x(b+a)\to x(b+a)$</td></tr>
<tr><td>$x(a+yb)=m=\sigma(m)$</td><td>$x(b+ya)=m=\sigma(m)$</td></tr>
<tr><td>$\sigma:\begin{Bmatrix}a\to a+yb\\ b\to \Lambda\end{Bmatrix}\quad x(a+yb)\to x(a+yb)$<br>$\sigma:\begin{Bmatrix}a\to a+(y-n)b\\ yb\to nb\end{Bmatrix}$<br>$x(a+yb)\to x[a+(y-n)b+nb]=x(a+yb)$</td><td>$\sigma:\begin{Bmatrix}a\to \Lambda\\ b\to b+ya\end{Bmatrix}\quad x(b+ya)\to x(b+ya)$<br>$\sigma:\begin{Bmatrix}a\to na\\ b\to b+(y-n)a\end{Bmatrix}$<br>$x(b+ya)\to x[b+(y-n)a+na]\to x(b+ya)$</td></tr>
</table>

## 5. $\sigma$ per invarianti periodiche

Notazione:

$$\prod_{i=1}^{k} S := S^k$$

Si dice periodica di periodo $T$ una parola del tipo:

$$m := T^k$$

Nel caso di parole infinite ($T^\infty$), si possono costruire invarianti secondo il seguente criterio:

$$\sigma(c) = \begin{cases} T^m \; se \; c = A \\ T^n \; se \; c = B \end{cases}, \quad m, n \in \mathbb{N} \setminus \{0\}$$

Ossia ogni lettera viene espansa in un multiplo del periodo della parola (non necessariamente lo stesso per i due caratteri) così da ottenere la stringa di partenza.