

# Pattern Matching

Giuseppe Persiano

Università di Salerno

Primo Semestre

# Pattern Matching

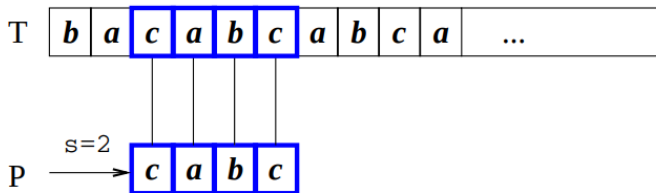
Abbiamo due input:

- una stringa  $T$  di lunghezza  $n$ , il *testo*
- una stringa  $P$  di lunghezza  $m$ , il *pattern*  
tipicamente,  $m \ll n$

Vogliamo trovare tutte le occorrenze di  $P$  in  $T$

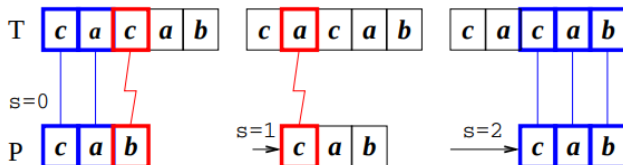
- cioè tutti gli indici  $0 \leq s \leq n - m$  tale che

$$T[s : s + m] = P$$



# Pattern Matching: Algoritmo di Forza Bruta

- Prova tutti i possibili valori di  $s$ 
  - ▶ per ogni valore prova se tutti i caratteri di  $P$  e  $T$  sono uguali.
  - ▶ se sì  $s$  è parte della soluzione
  - ▶ altrimenti prova il prossimo  $s$

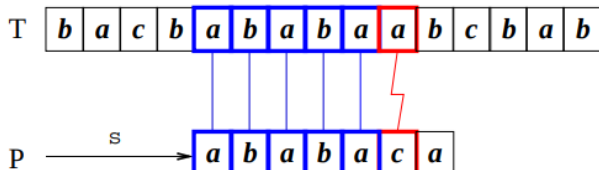


## Pattern Matching: Algoritmo di Forza Bruta

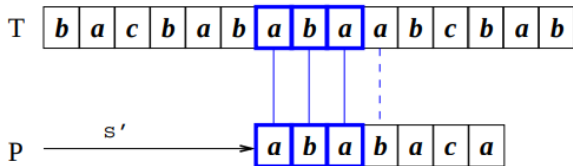
```
def bfPM(T,P):  
    res=[]  
    n=len(T)  
    m=len(P)  
  
    for s in range(n-m+1):  
        found=True  
        for i in range(m):  
            if P[i]!=T[s+i]:  
                found=False  
                break  
        if found:  
            res.append(s)  
    return res
```

# Pattern Matching: Knuth-Morris-Pratt

- L'algoritmo di forza bruta spreca delle informazioni guadagnate con i confronti
- Esempio:



- Inutile fare lo shift di una posizione



# La funzione next

- Abbiamo un match parziale

$$P[0 : q] = T[s : s + q]$$

e poi  $P[q] \neq T[s + q]$

- Cerchiamo il più piccolo indice  $\sigma > s$  tale che

$$P[0 : k] = T[\sigma : \sigma + k]$$

con  $\sigma + k = s + q$  e quindi

$$P[0 : k] = T[\sigma : s + q]$$

- Nota che  $T[\sigma : s + q]$  è uguale ad un suffisso di  $P[0 : q]$  quindi cerchiamo

- ▶ il più grande  $k < q$  tale

$$P[0 : k] \text{ è suffisso di } P[0 : q]$$

# La funzione next

- Per un pattern  $P[0 : m]$  la funzione

$$\text{next} : \{0, \dots, m-1\} \rightarrow \{0, \dots, m-1\}$$

è tale che

$$\text{next}(q) = \max\{k < q+1 : P[0 : k] \text{ suffisso di } P[0 : q+1]\}$$

- Se abbiamo un match parziale

$$P[0 : q] = T[s : s+q]$$

con  $P[q] \neq T[s+q]$ , ricominciamo da

$$P[\text{next}(q-1)]$$

```

def kmpPM(T,P):
    N=len(T)
    M=len(P)
    NXT=calcolaNext(P)
    res=[]
    i=0
    j=0
    while(i<N):
        if (P[j]==T[i]):
            j=j+1
            i=i+1
        if (j==M):
            res.append(i-M)
            j=NXT[j-1]
            continue

        if i<N and P[j]!=T[i]:
            if j!=0:
                j=NXT[j-1]
            else:
                i=i+1

    return res

```