String Marching

Algoritmo do Knuth Morris Pratt

A.A. 2016-2017

Backtracking: string matching

Data una stringa P, chiamata pattern e una stringa (piu` lunga) T, chiamata testo, il problema del pattern matching esatto e` quello di trovare tutte le occorrenze del pattern P nel testo T.

Per esempio: P = aba, T = bbabaxababay
P occorre in T tre volte, a partire dalle posizioni 3, 7 e 9.

A cosa serve?

- Programmi che trovano informazione testuale
- Programmi che cercano in cataloghi librari
- Internet browser
- Dizionari on-line
- Ricerche in basi dati che contengono parecchie centinaia di stringhe di DNA, RNA e aminoacidi.

Backtracking: string matching

Per esempio il genoma del batterio *Escherichia coli* di 4.6 milioni di nucleotidi ha milioni di repeat di lunghezza 12 e solo 8.000 di lunghezza massima 20 o piu`.
(il piu` frequente ACGCCGCATCCG di lunghezza 12 compare 94 volte).

La ricerca (usando GCG) in Genbank di una stringa di 30 caratteri ha richiesto 4 ore per verificare la sua non occorrenza. La ripetizione del test usando l'algoritmo di Boyer-Moore ha richiesto meno di 10 minuti, molti dei quali dedicati a swapping, meno di 1 minuto per la ricerca vera e propria.

Backtracking: string matching

***Metodo naif**:* Allineare i caratteri di sinistra di P e T e confrontarli da sinistra a destra finche' si trova un'occorrenza di P in T o si trovano due caratteri diversi. P viene spostato a destra di un carattere e il confronto ricomincia; termina quando il carattere piu' a sinistra di P supera quello piu' a sinistra di T.

P = aaa, T = aaaaaaaaaa vengono effettuati 24 confronti !

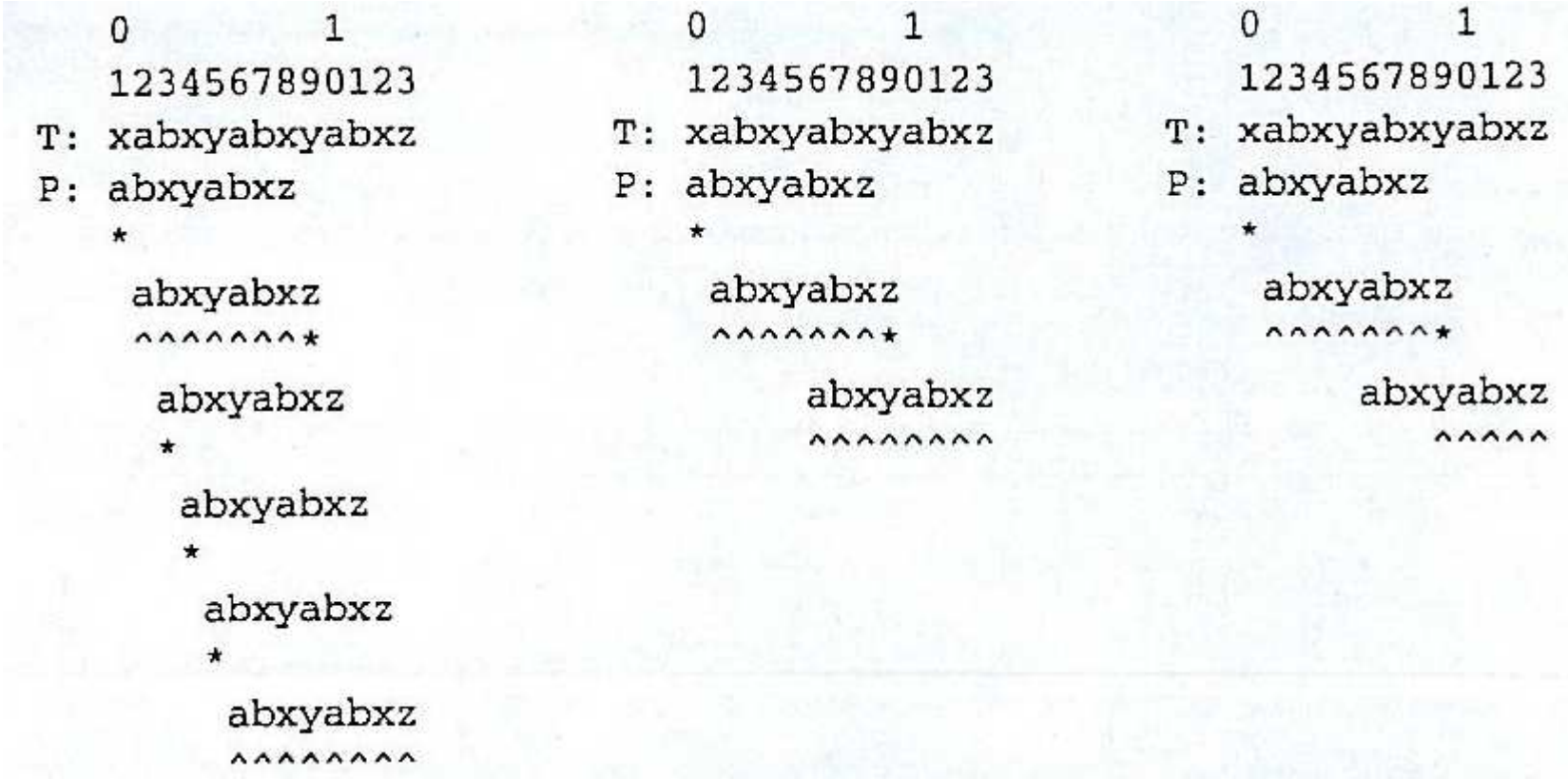
Sia $n = |P|$ e $m = |T|$, il numero di confronti nel caso peggiore e' $\Theta(n\,m)$

P = AAAT, T = AAA…AAAAA se | T| = 1 milione la ricerca richiede circa 5 milioni di confronti e "disappointing" finisce senza successo.

La complessita` puo` essere ridotta a $O(n + m)$

- Cercare di spostare P di piu` di un carattere quando si presenta un mismatch, mai pero` in modo da perdere un'occorrenza di P.
- Ridurre i confronti saltando alcune parti del pattern dopo lo shift.

Backtracking: string matching



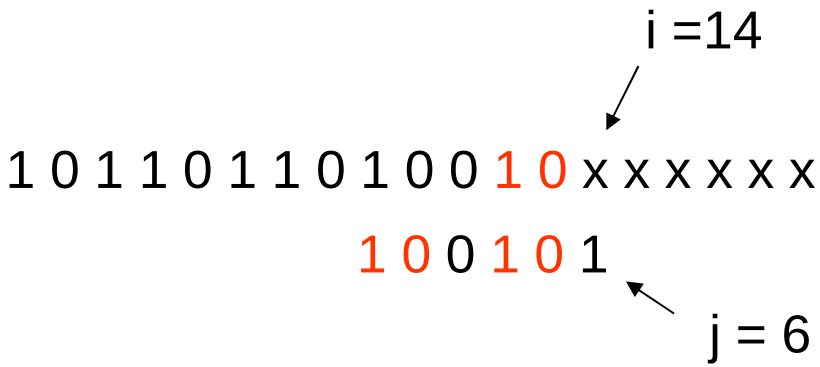
20 confronti 17 confronti 14 confronti

Backtracking: string matching

E' facile disegnare un algoritmo di ricerca bruta.

```
Ricerca-bruta (P, T, n, m)
        i, j, k: integer
        i ← 1
        j ← 1
        k ← 1
        while (i ≤ n and j ≤ m)
            if T[i] = P[j]
                    i ← i+1;   j ← j+1
            else k ← k+1;   i ← k;    j ← 1
        if j > m  return k
        else return i
```

Un backtrack meno brutale



T[9] = P[1]

T[10] = P[2]

T[11] = P[3]

T[14] ≠ P[6]

T[12] = P[4]

T[13] = P[5]

e P[1] P[2] = P[4] P[5]

T[12] T[13] = P[1] P[2]

Si può far ripartire il confronto da T[14] e P[3]

Backtracking: string matching

Essenziale: solo il pattern aiuta a calcolare l'indice da cui ricominciare il confronto, mentre sul testo non è necessario nessun backtracking.

Associamo ad ogni posizione j del pattern un indice che ricordi di quanto si deve "saltare all'indietro" se il confronto con il testo fallisce in quella posizione: back[j].

Nell'esempio precedente abbiamo back[6] = 3

Algoritmo di Knuth - Morris - Pratt

n: lunghezza del testo, m:lunghezza del pattern

```
Calcola_back (P, back, m)
        back[1] ← 0
        k ← 0
        for q ← 2 to m
            while  k>0  and P[k+1] ≠ P[q])
                    k = back[k]
            if P[k+1] = P[q]
                    k ← k +1
            back[q] ← k
        return
```

```
KMP (P, T, n, m)
    Calcola_back (P, back, m)
    q ← 0
    for i ← 1 to n
        while q > 0 and P[q+1]≠T[i]
                q ← back[q]
        if P[q+1] = T[i]
                q ← q+1
        if q = m
                print "pattern at i'
                q ←back[q]
    return
```

Backtracking: string matching

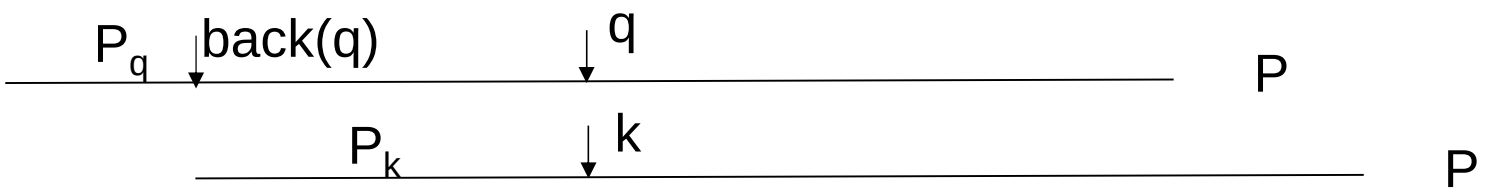
Perchè funziona? Proviamo a dare una spiegazione informale

Se P è una stringa chiamiamo P_q la sottostringa dei primi q caratteri
Indichiamo con $x \supset y$ se x =uy, cioè y è la parte terminale di x

Calcolando il vettore back su un pattern P si ha che:

$back[q] = \max \{k \mid k<q \text{ and } P_q \supset P_k\}$

In forma grafica:



Backtracking: string matching

```
Calcola_back (P, back, m)
        back[1] ← 0
        k ← 0
        for q ← 2 to m
            while  k>0  and P[k+1] ≠ P[q])
                    k = back[k]
            if P[k+1] = P[q]
                    k ← k +1
            back[q] ← k
            1
        return
```

Abbreviamo *back* con *b*
Quando il controllo passa dal punto 1 si ha sempre:

$\forall q' \leq q$:

$b[q'] = \max \{k \mid k<q' \text{ and } P_{q'} \sqsupset P_k \}$

Backtracking: string matching

Esempio

pattern =	**a**	**a**	**c**	**a**	**b**	**a**	**a**	**c**	**c**
	1	2	3	4	5	6	7	8	9

back[1] = 0

back[2] = 1

back[3] = 0

back[4] = 1

back[5] = 0

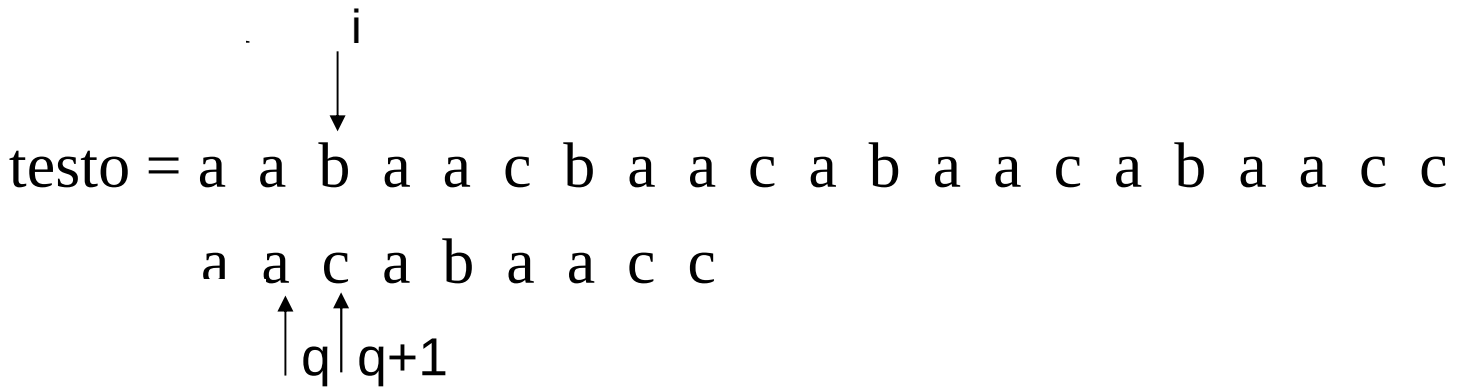
back[6] = 1

back[7] = 2

back[8] = 3

back[9] = 0

Backtracking: string matching



back[1] = 0

back[2] = 1

back[3] = 0

back[4] = 1

back[5] = 0

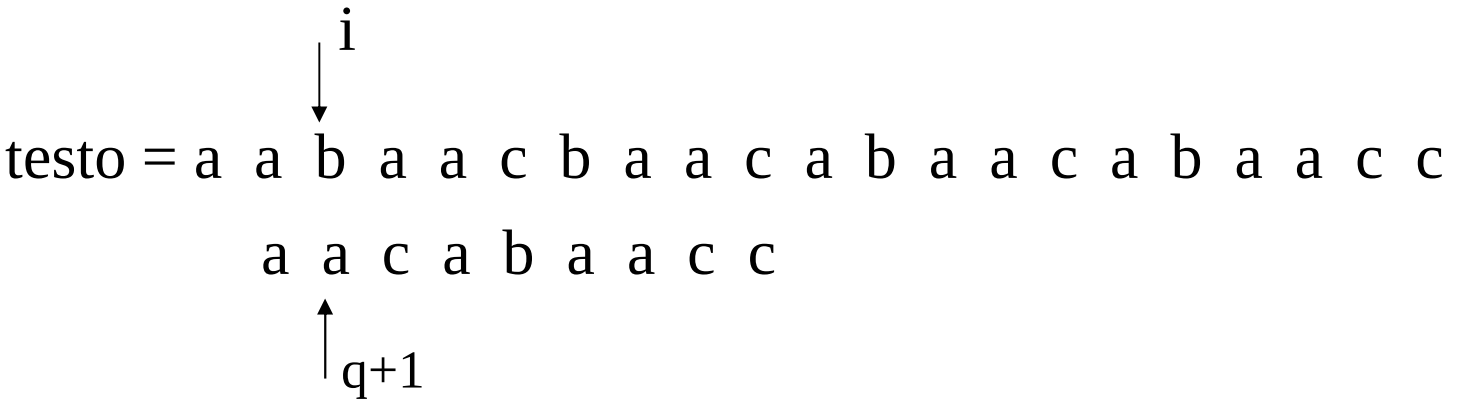
back[6] = 1

back[7] = 2

back[8] = 3

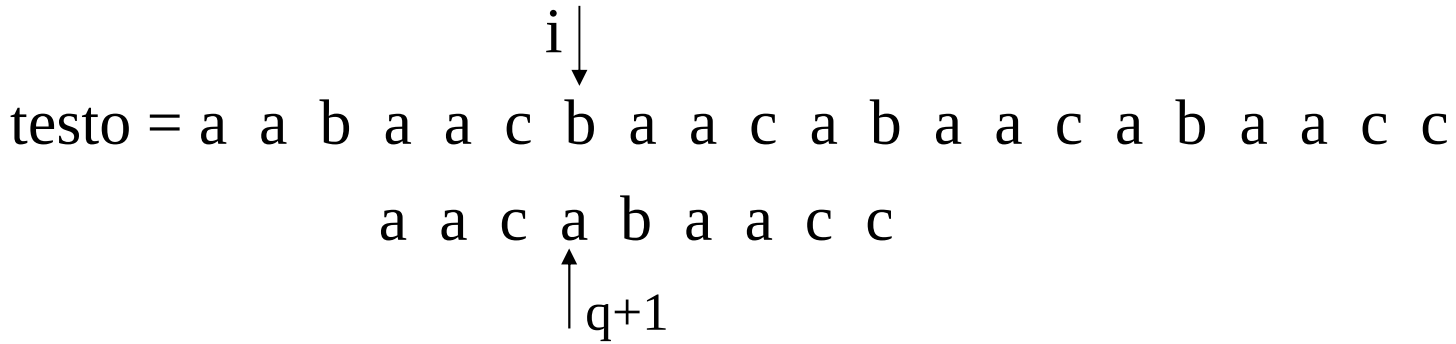
back[9] = 0

back[2] = 1 “il confronto deve avvenire tra il primo carattere del pattern e il carattere successivo del testo, cioè tra gli elementi in posizione i e q = 1



back[1] = 0 “il confronto fallisce e si resta nel while.
si deve ricominciare dal primo carattere del pattern”
e dal carattere successivo dell'input

Backtracking: string matching



back[1] = 0

back[2] = 1

back[3] = 0

back[4] = 1

back[5] = 0

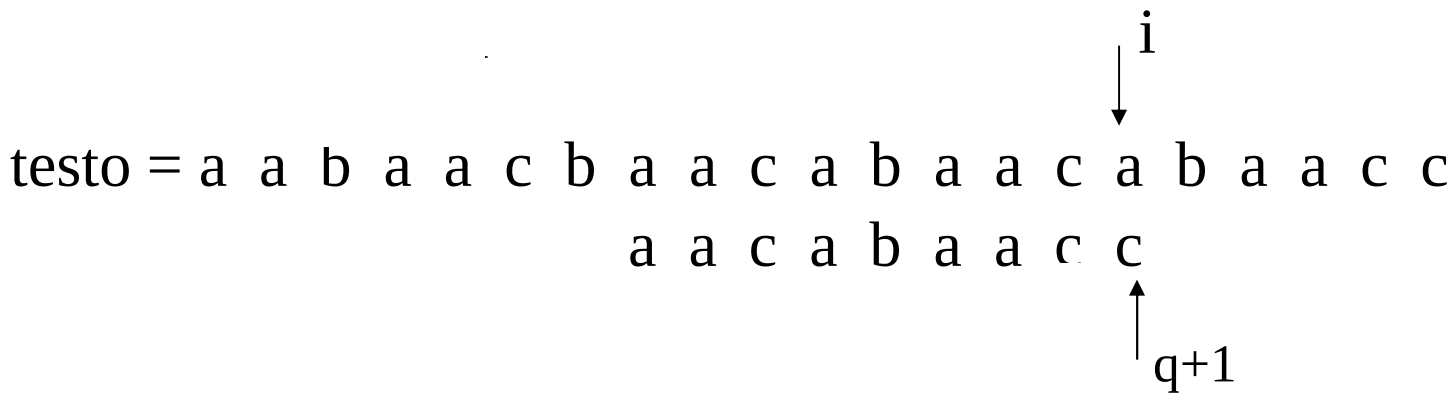
back[6] = 1

back[7] = 2

back[8] = 3

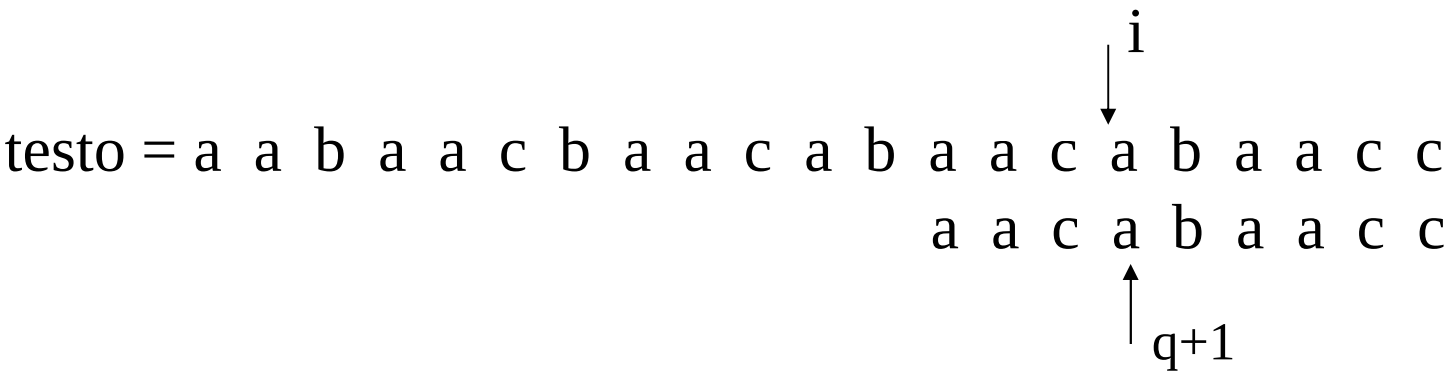
back[9] = 0

back[3] = 0 “il confronto deve avvenire tra il primo carattere del pattern e il carattere successivo del testo, cioè tra gli elementi in posizione i+1 e 1”



back[8] = 3 “il confronto deve ricominciare da i con q = 3 nel pattern”

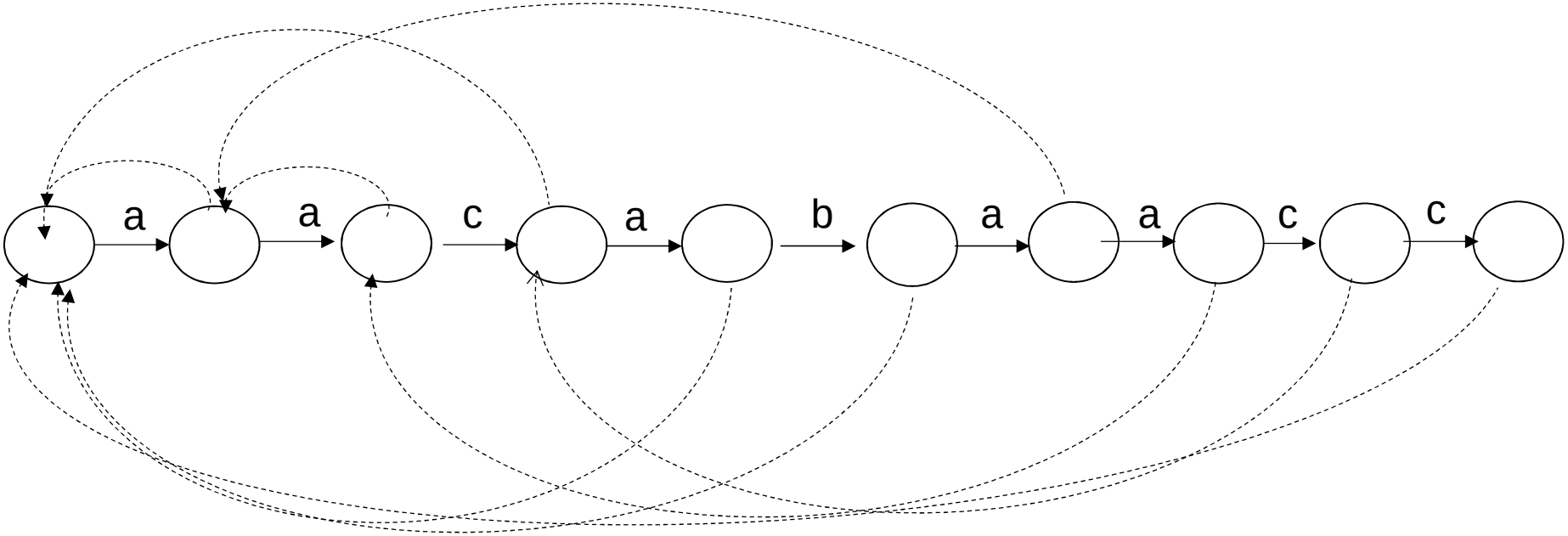
Backtracking: string matching



il pattern è trovato e si finisce

Backtracking: string matching

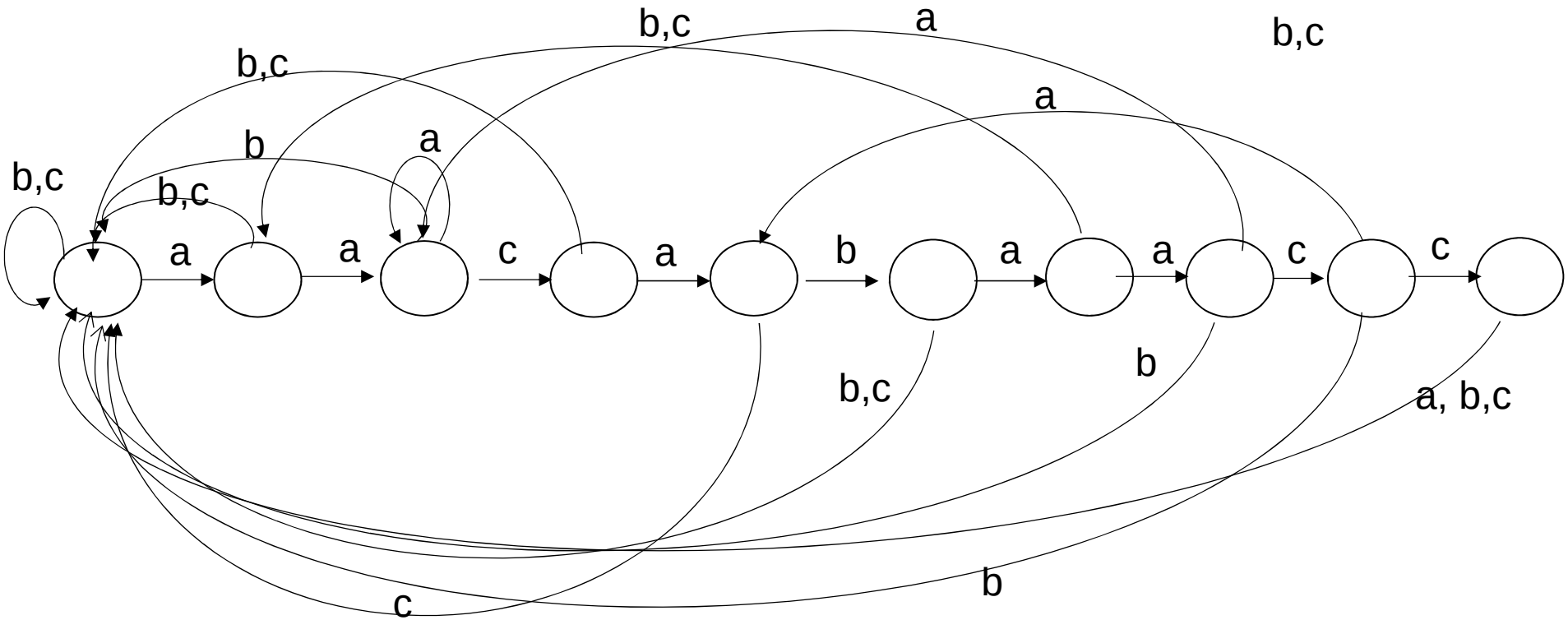
si può vedere come una specie di automa.



le transizioni all'indietro vengono eseguite per tutti quei simboli che non consentono di andare avanti, e si riparte (con lo stesso simbolo) dallo stato di arrivo.

Backtracking: string matching

Si può migliorare trasformandolo in un vero automa.



le transizioni all'indietro vengono eseguite per tutti quei simboli che non consentono di andare avanti, e si riparte (con lo stesso simbolo) dallo stato di arrivo.

Backtracking: string matching

Un altro esempio.

internal representation

	j	0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	1
	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

mismatch transition (back up)

match transition (increment)

graphical representation