Algoritmo di ***Aho-Corasick***
Report - Progetto di algoritmi e laboratorio

Rosario Mirabella

0.1 Introduzione

Gli algoritmi di pattern matching su stringhe, anche chiamati algoritmi di confronto tra stringhe, sono una classe importante di algoritmi su stringhe che individuano la posizione di una o più stringhe solitamente più piccole (dette anche pattern) all'interno di una stringa più grande o di un testo.

0.1.1 Descrizione formale del problema

Si assuma che il testo sia un array $\mathcal{T}[1 \ldots n]$ di lunghezza n e che il pattern sia un array $\mathcal{P}[1 \ldots m]$ di lunghezza m, con $m \leq n$. Supponiamo, inoltre, che gli elementi di $\mathcal{T}$ e $\mathcal{P}$ siano dei caratteri appartenenti ad un alfabeto finito indicato con Σ.
Un esempio potrebbe essere il seguente: $\Sigma = \{0,1\}$ oppure $\Sigma = \{a,b \ldots z\}$. $\mathcal{T}$ e $\mathcal{P}$ sono chiamati ***stringhe*** di caratteri. Si dice che il pattern (sottostringa) $\mathcal{P}$ è presente a cominciare dalla posizione $s+1$ nel testo $\mathcal{T}$ se $0 \leq s \leq n-m$ e $\mathcal{T}[s+1 \ldots s+m] = \mathcal{P}[1 \ldots m]$.

0.2 Tipi di algoritmi

Gli algoritmi che si occupano di individuare l'occorrenza di una certa sequenza, detta pattern, all'interno di una sequenza più lunga, possono essere classificati in varie categorie:

- ***Algoritmi di string matching esatti:*** gli algoritmi di string matching esatti consistono nel trovare una, più di una o tutte le occorrenze di un pattern in un testo, facendo in modo che ogni corrispondenza sia esatta.
- ***Algoritmi di string matching approssimativi:*** questa classe di algoritmi agisce nel modo seguente: Supponiamo di avere il testo $\mathcal{T}[1 \ldots n]$ e il pattern $\mathcal{P}[1 \ldots m]$, lo scopo è quello di trovare tutte le occorrenze del pattern nel testo la cui distanza di editing dal pattern sia, al massimo, un certo valore k.
- ***Algoritmi di string matching multipli:*** il pattern matching multiplo serve a cercare più pattern in un testo. Quindi, dati un insieme di pattern ed un testo, l'obiettivo è quello di trovare tutte le occorrenze dei pattern nel testo. In input si hanno k pattern $\mathcal{P}_1 \ldots \mathcal{P}_k$ e un testo $\mathcal{T}$. In output si avranno le posizioni del testo in cui iniziano i k pattern.

0.3 L'algoritmo di Aho-Corasick

L'algoritmo di Aho-Corasick è uno degli algoritmi di pattern matching multipli su stringhe. Esso venne ideato da *Alfred Vaino Aho e Margaret J. Corasick* nel 1975. L'algoritmo utilizza **un auotma a stati finiti** per trovare uno specifico

insieme di stringhe di pattern in un testo.
Un semplice algoritmo di ricerca di stringhe confronterebbe il pattern dato con tutte le posizioni nel testo. Ciò comporterebbe una complessità di $O(nm)$ dove n è lunghezza del testo ed m è lunghezza del pattern. L'algoritmo di Aho-Corasick riesce a fare di meglio ottenendo una complessità pari a

$$O(n + m + z)$$

dove $\boldsymbol{n}$ è la lunghezza del testo, $\boldsymbol{m}$ è la somma del numero di caratteri di ogni pattern e $\boldsymbol{z}$ è il numero totale di occorrenze di pattern nel testo.

0.3.1 Descrizione algoritmo

Automa di Aho-Corasick

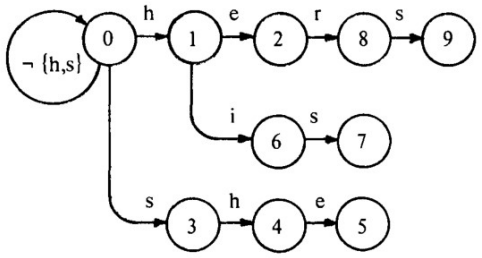
Gli automi a stati finiti possono essere considerati sia come modelli semplificati di macchine che come meccanismi usati per specificare linguaggi. Entrambi gli aspetti sono utili al fine del problema del pattern matching.
Un automa a stati finiti deterministico è una quintupla $A = \{\Sigma, Q, g, q, F\}$ dove

1. $\Sigma = \{a_0 ... a_n\}$ è l'alfabeto di input.
2. $Q = \{q_0 ... q_n\}$ è un insieme finito e non vuoto di stati.
3. $F \subseteq Q$ è l'insieme degli stati finali.
4. q_0 è lo stato iniziale
5. $g : Q \times \Sigma \to Q$ è la funzione di transizione che ad ogni coppia (carattere, stato) associa uno stato successivo.

L'automa di Aho-Corasick è un automa di pattern matching. Si parla di automa di pattern matching quando dato un pattern P, tale automa A_P accetta l'insieme di tutte le parole contenenti P come suffisso.
Detto questo diamo allora la definizione dell'automa di Aho-Corasick. L'automa viene definito mediante tre funzioni: **goto, fail, output**. Esse verrano indicate rispettivamente con *g, f, out.*

1. Sia $q \in Q$ ed $a \in \Sigma$, la funzione $g(q, a)$ restituisce il prossimo stato dell'auotoma se esiste (**match**), oppure, in caso contrario, l'insieme vuoto (**mismatch**). La funzione g è implementata utilizzando una **matrice bidimensionale** dove si memorizza, per ogni posizione, lo stato successivo partendo dallo stato q facendo il matching del carattere a. Fondamentale è ricordare che $g(0, a) = 0$ per ogni carattere $a \in \Sigma$ che non etichetta un arco fuori dalla radice.
   Il grafo riportato in basso rappresenta un'ipotetica funzione *goto*. Per esempio, l'arco etichettato con h da 0 ad 1 indica che $g(0, h) = 1$. L'assenza di tale arco avrebbe indicato un $fail$ (mismatch). Quindi, per questo esempio, $g(1, \sigma) = fail$ per ogni carattere $\sigma \neq e, i$.

Fig. 1.



(a) Goto function.

2. La funzione $fail$ $f : Q \to Q$. La funzione $f(q)$ entra in gioco in caso di mismatch con il carattere corrente a nello stato q, ossia nel caso in cui $g(q, a) = \emptyset$. Si definisce etichetta di un nodo v come la concatenazione delle etichette sugli archi che compongono il percorso dal nodo radice (quello presente allo stato 0) a v, e si indica con *L(v)*. Diremo che il valore $f(q)$ è il nodo etichettato dal più lungo suffisso proprio w di *L(q)* tale che w sia prefesso di almeno uno dei pattern $\mathcal{P}_1$ . . . . $\mathcal{P}_k$ da ricercare nel testo $\mathcal{T}$. Per questo motivo una transizione della funzione f (*fail transition*) non tralascia nessuna potenziale occorrenza. La funzione f è implementata, invece, utilizzando un **array unidimensionale**.

3. La funzione output, $out(q)$, restituisce l'insieme dei pattern riconosciuti quando ci si trova in uno stato q, indicando l'indice di inizio del pattern $\mathcal{P}$ all'interno del testo $\mathcal{T}$. Gli stati dell'automa sono rappresentati mediante i nodi di un trie. La funzione *out* è implementata, come la funzione *f*, utilizzando un semplice **vettore ad una dimensione**.

Costruzione dell'Automa di Aho-Corasick

Affinché sia possibile eseguire la ricerca di un pattern con l'algoritmo di Aho-Corasick è necessario che l'automa di Aho-Corasick sia stato costruito inizializzando correttamente le funzioni *g, f* e *out*.
La costruzione dell'automa per un insieme di pattern fissato avviene in due fasi:

1. Durante la prima fase si determinano gli stati dell'automa, il keyword tree relativo alla funzione g e, parzialmente, i valori della funzione *out*.

2. Nella seconda fase si definisce la funzione f e si determinano i valori definitivi della funzione *out*.

Per realizzare la ***funzione g*** si costruisce il keyword tree. La costruzione del trie è strutturata nel modo seguente. Si inizia creando il nodo radice, non etichettato in alcuna maniera particolare; si prosegue aggiungendo all'albero ogni pattern $\mathcal{P}_i$ come segue: partendo dal nodo radice si segue il cammino etichettato con i caratteri di $\mathcal{P}_i$ e, se il cammino termina prima di $\mathcal{P}_i$ si aggiunge un nuovo

arco e un nuovo nodo per ogni carattere rimanente di $\mathcal{P}_i$, etichettando il nuovo arco con il carattere aggiunto di $\mathcal{P}_i$. Alla fine si memorizza l'identificatore i di $\mathcal{P}_i$ nell'ultimo nodo del cammino. Per completare la funzione g, vengono creati dei cicli sul nodo radice settando $g(0, a) = 0$ per ogni carattere a, appartenente all'alfabeto, diverso dai caratteri iniziali delle keyword nell'insieme.

La ***funzione f*** è calcolata attraverso una visita in ampiezza (BFS) in cui ci si occupa anche di calcolare i valori definitivi della funzione *out*. Il valore della funzione f verrà calcolato per ogni stato s ad eccezione dello stato 0 per il quale la funzione f non è definita.
Si definisce la profondità di uno stato s nel grafo g come la lunghezza del cammino minimo che va dalla radice (stato 0) allo stato s. La funzione f è calcolata nel modo seguente:

1. Si mettono in coda i nodi a profondità 1 (ovvero i nodi per i quali è previsto un arco diretto con la radice) e vi si associa il valore di *fail* uguale a 0.

2. Il valore della funzione f associato al generico nodo a profondità d è calcolato sulla base dei valori della funzione g dei nodi a profondità minore.

Nello specifico, per calcolare i valori della funzione f per gli stati a profondità $\boldsymbol{d}$, si considerano tutti gli stati $\boldsymbol{r}$ a profondità $d - 1$ e si esegue tale ragionamento:

1. Se $g(r, a) = fail$ per ogni valore di $a \in \Sigma$, allora non eseguiamo alcuna operazione.

2. Altrimenti, per ogni valore di $a \in \Sigma$ per cui $g(r, a) = s$, si procede nel seguente modo:

   - si pone $state = f(r)$
   - si esegue l'operazione $state \leftarrow f(state)$ zero o più volte, finché non si ottiene un valore per cui $g(state, a) \neq fail$.
     Si noti che $g(0, c) \neq fail$ per ogni carattere c appartenente all'alfabeto, quindi il ciclo terminerà sempre. Intuitivamente la procedura $state \leftarrow f(state)$ serve a trovare il nodo *state* più profondo tale che *f(state)* sia un suffisso proprio di *L(s)* e *g(state,a)* sia definita.
   - si pone, infine, $f(s) = g(state, a)$

Una volta calcolata la funzione f del generico nodo s, l'algoritmo aggiornerà la ***funzione out*** unendo l'insieme di output di s con quello di *f(s)*. Questo viene fatto perché *f(s)*, se esiste, è un suffisso proprio di *L(s)* e dovrà quindi essere riconosciuto anche allo stato s.

Algorithm 1 : BUILD-MATCHING-MACHINE($\mathcal{P}$, k)

```
1: let be g and f initialized with each of its values to −1
2: let be out initialized with each of its values to 0
3: states ← 1
4: for i ← 0 to k do
5:    string word ← P[i]
6:    currentState ← 0
7:    for j ← 0 to length[word] do
8:       ch ← word[j] − 'a'
9:       if g[currentState][ch] = −1 then
10:         g[currentState][ch] ← states
11:         states ← states + 1
12:      end if
13:      currentState ← g[currentState][ch]
14:   end for
15:   links the string in question to the position out[currentState]
16: end for
17: for each ch ∈ Σ do
18:    if g[0][ch] = −1 then
19:       g[0][ch] ← 0
20:    end if
21: end for
22: let be q a queue
23: for each ch ∈ Σ do
24:    if g[0][ch] ≠ 0 then
25:       f[g[0][ch]] ← 0
26:       q ← q ∪ g[0][ch]
27:    end if
28: end for
29: while q ≠ ∅ do
30:    let be r the next state in q
31:    q ← q − {r}
32:    for each ch ∈ Σ do
33:       if g[r][ch] ≠ −1 then
34:          q ← q ∪ g[r][ch]
35:          state ← f[r]
36:          while g[state][ch] ≠ 1 do
37:             state ← f[state]
38:          end while
39:          state ← g[state][ch]
40:          f[g[r][ch]] ← state
41:          out[g[r][ch]] ← out[g[r][ch]] ∪ out[state]
42:       end if
43:    end for
44: end while
45: return states
```

Ricerca del pattern

Dopo aver costrutio correttamente l'automa di Aho-Corasick, si può procedere con la ricerca del pattern all'interno del testo.
Si assuma che il testo sia un array $\mathcal{T}[1 \ldots n]$ di lunghezza n e che il pattern sia un array $\mathcal{P}[1 \ldots m]$ di lunghezza m, con $m \leq n$. La ricerca avviene nel modo seguente. Si utilizza un indice i con cui si scorre $\mathcal{T}$ da 0 ad $m-1$. Si esamina inizialmente il carattere in posizione 0 di $\mathcal{T}$ e si controlla se esiste nel keyword tree un arco uscente dallo *starting node* etichettato con il carattere in posizione i. Se esiste, vuol dire che la funzione g è definita per quel carattere, quindi si esegue una ***goto transition*** spostandoci allo stato successivo. Se tale stato ha una funzione di output non vuota, allora è stata trovata un'occorrenza e si calcolano gli indici a partire dai quali quella stringa si trova nel testo. Se tale stato, invece, ha una funzione di output vuota, si incrementa il valore di i passando al carattere successivo di $\mathcal{T}$ e si ripete nuovamente la procedura descritta prima.
Se si giunge ad uno stato (un nodo del grafo) per il quale la funzione g non è definita, si esegue una ***fail transition*** per non mancare nessuna potenziale occorrenza. Dal fail node si verifica se esiste un arco etichettato con il carattere in esame. Se esiste, si controlla se a quel determinato stato è associato un valore di output non vuoto. Se è così è stata trovata un'altra occorrenza. Dopo aver fatto questo, si esamina il carattere successivo di $\mathcal{T}$ e si continua nel modo sopra descritto finchè $i = m - 1$.

Algorithm 2 : FIND-NEXT-STATE ***(currentState, nextInput)***

```
1: answer ← currentState
2: ch ← nextInput - 'a'
3: while g[answer][ch] = −1 do
4:     answer ← f[answer]
5: end while
6: return g[answer][ch]
```

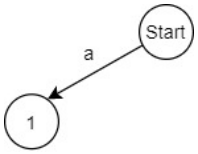
Algorithm 3 : SEARCH-WORDS (***P, T, k***)

```
1: buildMatchingMachine(P, k)
2: currentState ← 0
3: for i ← 0 to length[T] do
4:     currentState ← findNextState(currentState, T[i])
5:     if out[currentState] ≠ ∅ then
6:         print "word" P[j] "appears from" i − P[j].size + 1 "to" i
7:     end if
8: end for
```

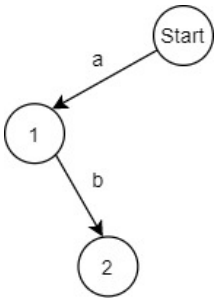
Spiegazione dell'algoritmo di Aho-Corasick con un esempio pratico

Supponiamo che $\mathcal{P}$ = {a, ab, bc, aac, aab, bd} e che $\mathcal{T}$ = {bcaab}.
La prima fase dell'algoritmo consiste nella costruzione dell'automa, realizzando il grafo *g*, la funzione *f* e la funzione *out*.

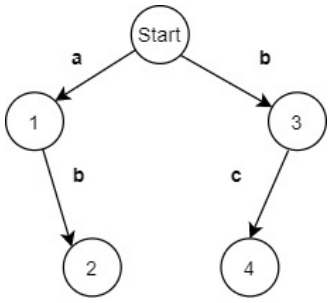
Si inserisce la prima keywoord, **"a"**, all'interno del grafo. Il cammino dallo stato 0 (lo *start state*) allo stato 1 mostra la stringa **'a'**. Si associa, quindi, la stringa **"a"** con lo stato 1 alla funzione *out*.



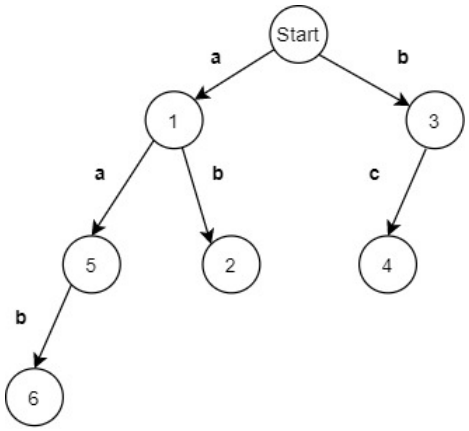
Aggiungendo la seconda keyword **"ab"**, si ottiene la seguente configurazione. La stringa **"ab"** è una delle stringhe da ricerca all'interno del testo, quindi si associa **"ab"**, nella funzione *out*, con lo stato 2.



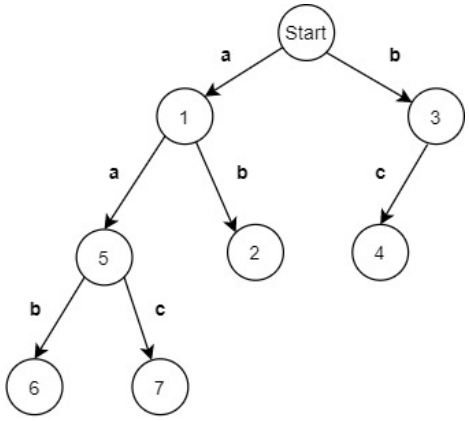
Per aggiungere la stringa **"bc"** si crea un cammino che inizia dallo *start state*, poichè, all'interno del grafo, prima dell'inserimento di tale stringa, non era presente una keyword che inizia con il carattere b. Dopo aver inserito la stringa, si associa l'output **"bc"** con lo stato 4.



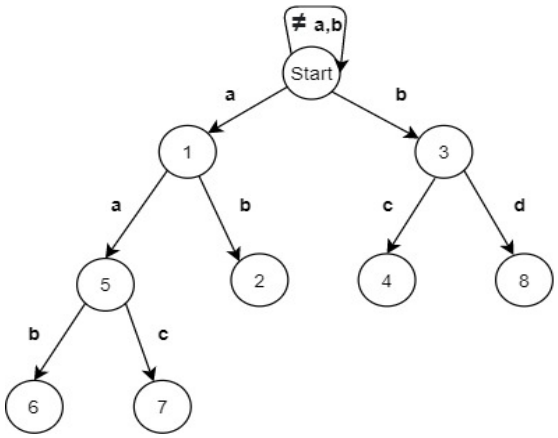
Si procede aggiungendo la stringa **"aab"**. L'output **"aab"** viene associato allo stato 6. È importante notare che, al momento dell'inserimento della stringa, esisteva già un arco etichettato con "a" partente dallo *start state*, per cui non era necessario aggiungere nuovamente $g(0, a) = 1$.



Aggiungendo la penultima stringa del pattern, si ottiene la seguente configurazione. Nella funzione *out* si associa l'output **"aac"** con lo stato 7. Anche in questo caso il prefisso "aa" della stringa "aac" era già presente all'interno del grafo, per cui è bastato porre $g(5, c) = 7$.



Per completare il grafo si inserisce l'ultima keyword **"bd"** ponendo $g(3, d) = 8$ e si aggiunge un arco dallo *start state* allo *start state* la cui etichetta dovrà essere diversa da a e b.



Quinid, dopo la fine della prima fase di costruzione dell'automa di Aho-Corasick, il keyword tree che si ottiene è quello riportato sopra, e, inoltre, la funzione di output presenta i seguenti valori:

- L'output **"a"** è asccoiato allo **stato 1.**
- L'output**"ab"** è associato allo **stato 2.**
- L'output **"bc"** è associato allo **stato 4.**
- L'output **"aab"** è associato allo **stato 6.**
- L'output **"aac"** è associato allo **stato 7.**
- L'output **"bd"** è associato allo **stato 8.**

La seconda fase di costruzione dell'automa di Aho-Corasick procede determinando i valori della funzione f e i valori definitivi della funzione *out*. Si inizia inserendo in una coda i nodi a profondità uno, che sono il nodo 1 e il nodo 3, e vi si associa un valore di *fail* uguale a 0. Quindi $\mathbf{f(1) = f(3) = 0}$.

Si estrae il primo nodo dalla coda, cioè il nodo 1. Si determina il valore di *fail* per ogni nodo raggiungibile da un arco uscente dal nodo 1. I nodi raggiungibili dal nodo 1 sono il nodo 5 e il nodo 2, i quali vengono inseriti in coda. Determiniamo per prima cosa il valore di *fail* relativo al nodo 5. L'arco che, dal nodo 1, ci permettere di raggiungere il nodo 5 è quello etichettato con "a" per

cui vale $g(1, a) = 5$. Poniamo una variabile *state* uguale al valore di *fail* associato al nodo estratto dalla coda. Per cui $state = f(1) = 0$. Poiché $g(0, a) = 1$ vale che il valore di *fail* associato allo stato 5 è **f(5) = g(0,a) = 1**.
Poiché allo stato 1 era associato l'output "a", questo sarà riconoscibile anche allo stato 5. Questo perché, ogni volta che si computa $f(s) = s'$, si calcola l'unione dell'output associato allo stato s con quello associato allo stato *s'*.

Si calcola adesso il valore di *fail* relativo al nodo 2. L'arco che dal nodo corrente, il nodo 1, ci permette di raggiungere il nodo 2 è quello etichettato con "b" per il quale vale $g(1, b) = 2$. Si pone *state* uguale al valore di *fail* associato al nodo estratto dalla coda, per cui $state = f(1) = 0$. Poiché $g(0, b) = 3$, allora il valore di *fail* associato allo stato 2 è **f(2) = g(0,b) = 3**.
Dato che allo stato 3 non viene riconosciuta alcuna stringa del pattern, allo stato 2 abbiamo solo la possibilità di individuare la stringa "ab". Per cui la funzione di output non subisce cambiamenti.

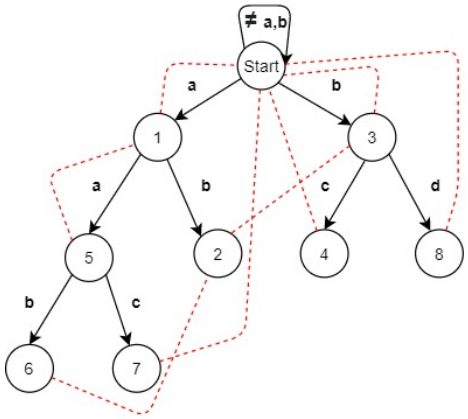
Si continua estraendo il prossimo nodo dalla coda, cioè il nodo 3. Si determina il valore di *fail* per ogni nodo raggiungibile da un arco uscente dal nodo 3, cioè i nodi 4 e 8.
Si calcola il valore di *fail* relativo al nodo 4. L'arco che, dal nodo 3, permettere di raggiungere il nodo 4 è quello etichettato con "c" per cui vale $g(3, c) = 4$. Si pone *state* uguale al valore di *fail* associato al nodo estratto dalla coda, quindi $state = f(3) = 0$. Poiché $g(0, c) = 0$, il valore di *fail* associato allo stato 4 è **f(4) = g(0,c) = 0**.
Pure in questo caso il valore di output, associato allo stato 4, calcolato durante la pirma fase di costruzione dell'automa, rimane invariato.

Si calcola adesso il valore di *fail* relativo al nodo 8. L'arco che, dal nodo corrente (il nodo 3), permette di arrivare al nodo 8 è quello etichettato con "d" per cui vale $g(3, d) = 8$. Si setta *state* uguale al valore di *fail* associato al nodo corrente, per cui $state = f(3) = 0$. Poiché $g(0, d) = 0$, allora il valore di fail associato allo stato 8 è **f(8) = g(0,d) = 0**. **Anche in questo caso il valore di output, associato allo stato 8, calcolato durante la pirma fase di costruzione dell'automa, rimane invariato.**

Si procede estraendo il prossimo nodo dalla coda, cioè il nodo 5. Si determina il valore di *fail* per ogni nodo raggiungibile da un arco uscente dal nodo 5, cioè i nodi 6 e 7, i quali vengono inseriti in coda.
Si calcola $f(6)$. L'arco che, dal nodo 5, permettere di raggiungere il nodo 6 è quello etichettato con "b" per cui vale $g(5, b) = 6$. Si pone *state* uguale al valore di fail associato al nodo estratto dalla coda, per cui $state = f(5) = 1$. Poiché $g(1, b) = 2$, il valore di *fail* associato allo stato 6 è **f(6) = g(1,b) = 2**. **Quindi, allo stato 6 riconosciamo, oltre alla stringa "aab", la stringa associata allo stato 2, ovvero "ab".**

Si calcola, per ultimo, $f(7)$. Si percorre l'arco per cui vale $g(5, c) = 7$. Si inizializza *state* uguale al valore di *fail* associato al nodo estratto dalla coda, per cui $state = f(5) = 1$. Poiché g(1,c) non è definito, si esegue il passo *state* $\leftarrow$ *f(state)* da cui si ottiene $state = f(1) = 0$. Dato che adesso $g(0, c) = 0$, il valore di fail associato allo stato 7 è **f(7) = g(0,c) = 0**. **Il valore di output associato allo stato 7 rimane invariato.**
Per i restanti nodi in coda, **2,4,8,6,7** non è previsto alcun arco uscente, quindi vengono estratti dalla coda senza eseguire alcun'altra operazione.

I valori di *fail* per ogni nodo x del grafo g sono indicati tramite una linea tratteggiata che va dal nodo x al nodo che rappresenta il suo valore di *fail*. Ecco il risultato ottenuto:



Dopo aver costruito correttamente l'automa di Aho-Corasick, si hanno i seguenti valori relativi alla funzione *out*:

- Allo **stato 1** riconosciamo la stringa **"a"**.
- Allo **stato 2** riconosciamo la stringa **"ab"**.
- Allo **stato 4** riconosciamo la stringa **"bc"**.
- Allo **stato 5** riconosciamo la stringa **"a"**.
- Allo **stato 6** riconosciamo le stringhe **"aab"** e **"ab"** .
- Allo **stato 7** riconosciamo la stringa **"acc"**.
- Allo **stato 8** riconosciamo la stringa **"bd"**.

Sfruttando tutte le informazioni ottenute fino a questo momento, si può procedere con la **ricerca** del pattern $\mathcal{P} = \{a, ab, bc, aac, aab, bd\}$ all'interno del testo $\mathcal{T} = \{bcaab\}$.

STEP 1
Inizialmente $i = 0$, quindi il primo carattere ad essere esaminato è quello in posizione $\mathcal{T}[i]$, ovvero "b".
Si controlla se dal nodo 0 (lo *start state*) esiste un arco etichettato con "b". Dato che questo esiste, si percorre il cammino indicato dall'arco in questione, il quale raggiunge il nodo 3. Poichè allo stato 3 dell'automa non viene riconosciuta alcuna stringa del pattern, si procede incrementando il valore di i di un'unità.

STEP 2
$i = 1$. Si esamina il carattere $\mathcal{T}[1] = c$. Dal nodo 3 si segue il cammino che porta al nodo 4 poichè $g(3, c) = 4$. Allo stato 4 dell'automa viene riconosciuta la stringa **"bc"** del pattern. La sua posizione all'interno del testo $\mathcal{T}$ viene calcolata nel modo seguente: **"bc"** è presente in $\mathcal{T}$ dalla posizione $i - length(\text{"}bc\text{"}) + 1$ alla posizione $\boldsymbol{i}$, quindi dalla posizione 0 alla posizione 1. Si continua con lo step successivo incrementando i di uno.

STEP 3
Adesso $i = 2$. Si esamina il carattere $\mathcal{T}[2] = a$. Dal nodo 4 non ci sono archi uscenti, per cui, per spostarsi al nodo successivo, si ricorre al valore suggerito dalla funzione *fail* per il nodo 4. Poichè $f(4) = 0$, si ritorna al nodo 0. Dato che $g(0, a) = 1$, ci si sposta al nodo 1. Allo stato 1 dell'automa è associato la stringa **"a"** del pattern, la cui posizione all'interno del testo va da $i - length(\text{"}a\text{"}) + 1$ a $\boldsymbol{i}$, quindi dalla posizione 2 alla posizione 2. Si passa allo step successivo incrementando i.

STEP 4
Adesso $i = 3$. Si esamina il carattere $\mathcal{T}[3] = a$. Il nodo corrente è il nodo 1. Dato che $g(1, a) = 5$, si segue il cammino che porta al nodo 5. Allo stato 5 riconosciamo nuovamente la stinga **"a"**, la quale è presente all'interno del testo dalla posizione $i - length(\text{"}a\text{"}) + 1$ alla posizione $\boldsymbol{i}$, quindi dalla posizione 3 alla posizione 3. Si incrementa nuovamente il valore di i.

STEP 5
Adesso $i = 4$. Si esamina il carattere $\mathcal{T}[4] = b$. Il nodo corrente è il nodo 5. Dato che $g(5, b) = 6$, si segue il cammino che porta dal nodo 5 al nodo 6. Allo stato 6 riconosciamo la stinga **"aab"** e la stringa **"ab"**. La stringa "aab" appare dalla posizione $i - length(\text{"}aab\text{"}) + 1$ alla posizione $\boldsymbol{i}$, quindi dalla posizione 2 alla posizione 4. La stringa "ab" appare dalla posizione $i - length(\text{"}ab\text{"}) + 1$ alla posizione $\boldsymbol{i}$, quindi dalla posizione 3 alla posizione 4. Dato che i ha scorso ogni carattere del testo l'algoritmo termina correttamente avendo trovato tutte le occorrenze.