Lezione 17 06/12/2023

Occorrenza approssimata (recap)

DEFINIZIONE

Una posizione i del testo T tale che esista una sottostringa S=T[i-L+1,i] tale che ED(P,S) ≤ k, è un'occorrenza approssimata di P in T

ESEMPIO:

T = bbaccbbaac, k = 1 S_1 = ccbb

P = ccbb ED(P,S_1) = 0 ≤ k

i = 7 → occorrenza approssimata!

Testo T, pattern P, soglia di errore 1. Dove c'è la freccia è un'occorrenza approssimata? si.

ESEMPIO:

T = bbaccbbaac, k = 1 S_2 = cbb

P = cbb ED(P,S_2) = 1 ≤ k

i = 7 → occorrenza approssimata!

Anche qui è un'occorrenza approssimata perché abbiamo k=1

String matching approssimato

INPUT:

Un **testo T** di lunghezza n, un **pattern P** di lunghezza m, definiti su alfabeto Σ e una **soglia k di errore**

OUTPUT:

tutte le posizioni i di T in cui finisce una sottostringa S tale che ED(P,S) ≤ k

Ricerca esatta con WM

1. Preprocessing del pattern P in tempo $\theta(|\Sigma|+m)$ → calcolo di $|\Sigma|$ *words* di m bit

2. Scansione del testo T in tempo $\theta(kn)$ per cercare le occorrenze approssimate di P

Nella fase di preprocessing si calcolava una tabella di parole di m bit, una per ogni simbolo dell'alfabeto, che dicevano come si distribuiscono i simboli. Quindi 1) è uguale a prima.
Nella scansione del testo però ora abbiamo anche k (soglia di errore) quindi non è una sola scansione, sono più scansioni successive e quindi la complessità è più alta.

Definizione di *word* D^h_i $(0 \le h \le k, 0 \le i \le n)$

$P[1,j] = suff_h(T[1,i])$
significa
"P[1,j] è uguale a un suffisso di T[1,i] a meno di h errori"

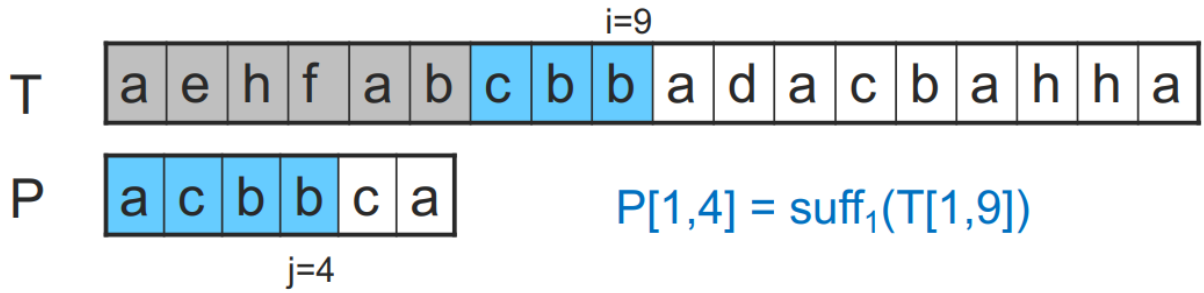
Cioè, riesco a trovare un suffisso S di T[1,i] tale che:

$ED(S,P[1,j]) \le h$

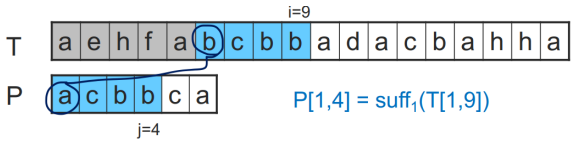
Dobbiamo definire delle parole di m bit ma più complicate. Avremo una parola, oltre alle posizioni del testo, per ogni possibile valore di k. Quindi i varia tra 0 e n, mentre h varia tra errore 0 e k. Avremo quindi (k+1)(n+1) parole.

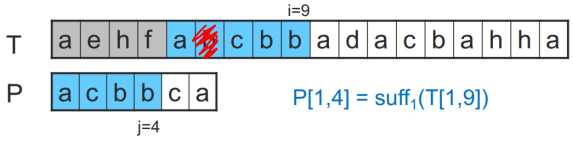
Quando scriviamo la prima riga blu, significa che possiamo trovare una sottostringa del testo che finisce in i che ha distanza di edit con il prefisso del pattern lungo j che è minore o uguale ad h.

Nell'ultima riga suffisso = sottostringa che finisce in i.



Troviamo una sottostringa che finisce in 9, che ha distanza di edit con il prefisso del pattern (blu del pattern) che è minore uguale a 1. cbb ha distanza di edit con acbb 1.





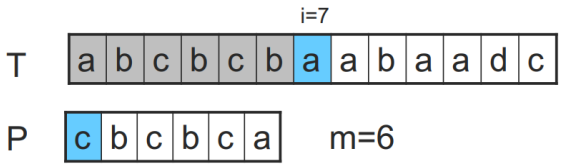
sono corrette anche queste due stringhe.

Tornando alla definizione di word D:

> Dati P lungo m e T lungo n e una soglia k di errore, D^h_i ($0 \le h \le k$, $0 \le i \le n$) è una *word* di m bit tale che:
>
> $D^h_i[j] = 1 \Leftrightarrow P[1,j] = suff_h(T[1,i])$

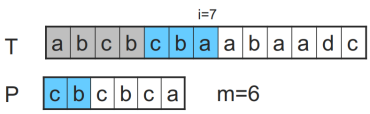
Esempio

$D^1_7 = \underline{1}11111$



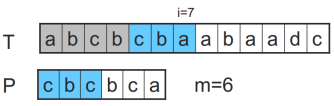
Il primo bit è 1 perchè il prefisso lungo 1 del pattern occorre come suffisso del prefisso lungo 7 nel testo con al più un errore. (Il prefisso lungo 1 del pattern ha distanza di edit con un sottostringa del testo che finisce in 7 che è minore di 1)
Il primo bit 1 è perché la distanza di edit è 1.

$D^1_7 = 1\underline{1}1111$

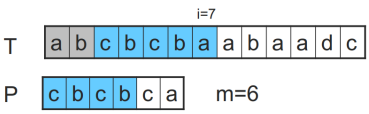


Anche qui abbiamo distanza di edit 1.

$D^1_7 = 11\underline{1}111$

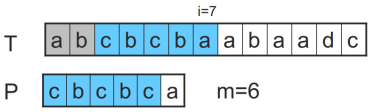


$D^1_7 = 111\underline{1}11$

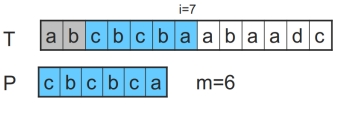


Cancelliamo la a (errore), siamo nella soglia.

$D^1_7 = 1111\underline{1}1$



$D^1_7 = 11111\underline{1}$



L'errore può essere sia sul patter che sul testo, può essere fatto con una cancellazione, un inserimento, una sostituzione.

Tornando alla definizione di word D

Dati P lungo m e T lungo n e una soglia k di errore, D^h_i ($0 \le h \le k$, $0 \le i \le n$) è una *word* di m bit tale che:

$D^h_i[j] = 1 \Leftrightarrow P[1,j] = suff_h(T[1,i])$

D^0_i (h=0) è per definizione la *word* D_i di BYG.

D a livello 0 abbiamo quindi tutte le parole esatte, senza errore. Quindi è lo stesso algoritmo BYG.

D^h_0 (i=0) è tale che:

$j \le h \Rightarrow D^h_0[j] = 1$ ($P[1,j] = suff_h(T[1,0])$)

$j > h \Rightarrow D^h_0[j] = 0$ ($P[1,j]$ non è $suff_h(T[1,0])$)

$D^h_0 =$ 1 1 .. 1 0 .. 0 0

h bit a 1 (m-h) bit a 0

Quando i=0 il bit j sarà 1 se $j \le h$. Anche se la porzione di testo analizzata è nulla, il bit sarà comunque 1 quando la lunghezza del prefisso del pattern è ≤ ad h, ovvero gli errori massimi accettati.

Se invece j>h abbiamo solo bit a 0.

$D^h_i[m] = 1 \Leftrightarrow P[1,m] = suff_h(T[1,i])$

In particolare:

$D^0_i[m] = 1 \Leftrightarrow P[1,m] = suff_0(T[1,i])$

$D^k_i[m] = 1 \Leftrightarrow P[1,m] = suff_k(T[1,i])$

Questo è il segnale per l'occorrenza approssimata.

$D^h_i[m] = 1 \Leftrightarrow$ i è occorrenza approssimata per soglia h

In particolare:

$D^0_i[m] = 1 \Leftrightarrow$ i-m+1 è occorrenza esatta

$D^k_i[m] = 1 \Leftrightarrow$ i è occorrenza approssimata

Per un h generico che varia tra 0 e k questa è l'occorrenza approssimata.

$D^h_i[m] = 1 \Rightarrow D^{h+1}_i[m] = 1 \Rightarrow D^{h+2}_i[m] = 1 \Rightarrow \ldots \Rightarrow D^k_i[m] = 1$

Se il bit m della parola dove h è 1, come sarà lo stesso bit della parola con h successiva? Sarà sempre 1 perché sto semplicemente concedendo un errore in più. E così via.

$D^h_i[m] = 1 \quad \Rightarrow \quad D^k_i[m] = 1$

Quindi se abbiamo una parola per un h intermedio che ha l'ultimo bit a 1, allora tutte quelle con h maggiore avranno l'ultimo bit a 1.

Quindi le occorrenze approssimate saranno trovate guardando tutte le parole al livello k.

Scansione del testo

Il testo T viene scandito per k+1 volte:

- **Iterazione 0** → calcolo delle *words* D^0_i
- **Iterazione 1** → calcolo delle *words* D^1_i
- **Iterazione 2** → calcolo delle *words* D^2_i
- …
- **Iterazione h** → calcolo delle *words* D^h_i
- …
- **Iterazione k** → calcolo delle *words* D^k_i
- **All'Iterazione k**, se $D^k_i[m] = 1$, viene prodotta in output l'occorrenza approssimata i

Ad ogni iterazione cambia la h.

Tutte le occorrenze trovate nei livelli precedenti saranno trovate all'ultimo livello.

Iterazione h=0: calcolo della *word* D^0_i per i da 0 a n
→ algoritmo BYG

Word D^h_0 per h da 1 a k ?
D^h_0 ha primi h bit a 1 e i restanti (m-h) bit a 0

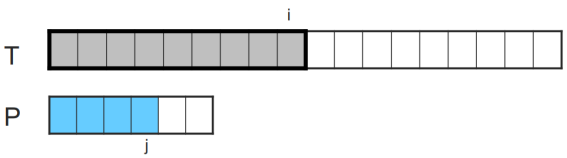
QUESTIONE DA RISOLVERE:
calcolare la parola D^h_i per i > 0 e h > 0

Come calcoliamo la parola generica per i>0 e h>0?

Calcolo di $D^h_i[j]$ (h > 0, i > 0)

j > 1

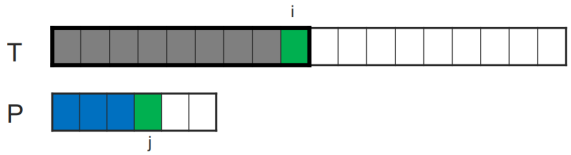
$D^h_i[j] = 1 \Leftrightarrow P[1,j] = suff_h(T[1,i])$



C'è una condizione che si verifica che mi rende vera l'affermazione in blu, e di conseguenza il bit 1?

$P[1,j-1] = suff_h(T[1,i-1])$ **AND** $P[j] = T[i]$

se il prefisso accorciato del pattern ha distanza di edit ≤ ad h e gli ultimi due caratteri devono essere uguali.



$$D^h_i[j] = 1 \Leftarrow P[1,j-1] = suff_h(T[1,i-1]) \textbf{ AND } P[j] = T[i]$$

Non è un sse perchè se l'AND è falso, il bit potrebbe essere comunque a 1, ci arriviamo.

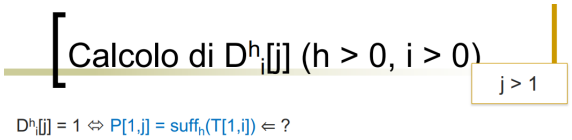
Possiamo sostituire il pezzo blu con la D di i-1 e il suo bit j-1.

$$D^h_i[j] = 1 \Leftarrow D^h_{i-1}[j-1] = 1 \textbf{ AND } B_{T[i]}[j] = 1$$

Sostituiamo il secondo pezzo con il bit j-esimo della parola B (di preprocessing)

$D^h_i[j] \Leftarrow D^h_{i-1}[j-1]$ **AND** $B_{T[i]}[j]$ **CASO1**

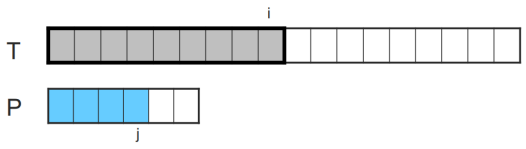
Arriviamo quindi a questa affermazione finale, che è il caso 1.

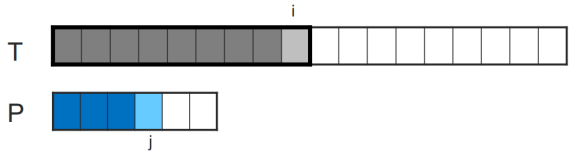


$D^h_i[j] = 1 \Leftrightarrow P[1,j] = suff_h(T[1,i]) \Leftarrow ?$

Il caso 2 richiede un errore in meno. Quindi l'ultimo bit può anche essere diverso, non lo consideriamo.

$$D^h_i[j] = 1 \Leftarrow P[1,j-1] = suff_{h-1}(T[1,i-1])$$





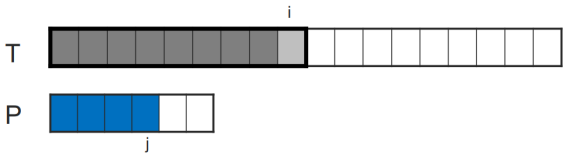
$$D^h_i[j] \Leftarrow D^{h-1}_{i-1}[j-1] \qquad \textbf{CASO2}$$

Troviamo quindi il secondo caso. C'è anche un terzo caso:

Calcolo di $D^h_i[j]$ ($h > 0$, $i > 0$)

$j > 1$

$D^h_i[j] = 1 \Leftrightarrow P[1,j] = suff_h(T[1,i]) \Leftarrow P[1,j] = suff_{h-1}(T[1,i-1])$



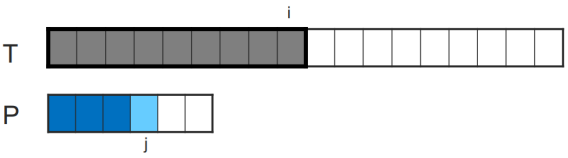
$$D^h_i[j] \Leftarrow D^{h-1}_{i-1}[j]$$

CASO3

Calcolo di $D^h_i[j]$ ($h > 0$, $i > 0$)

$j > 1$

$D^h_i[j] = 1 \Leftrightarrow P[1,j] = suff_h(T[1,i]) \Leftarrow P[1,j-1] = suff_{h-1}(T[1,i])$



$$D^h_i[j] \Leftarrow D^{h-1}_{i}[j-1]$$

CASO4

Riassumendo:

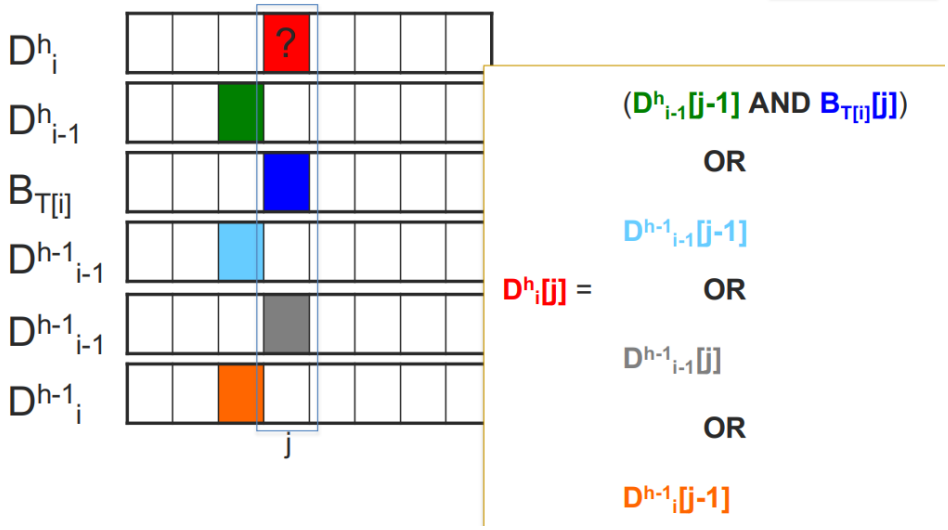
1. $D^h_i[j] \Leftarrow (D^h_{i-1}[j-1]$ **AND** $B_{T[i]}[j])$
   OR
2. $D^h_i[j] \Leftarrow D^{h-1}_{i-1}[j-1]$
   OR
3. $D^h_i[j] \Leftarrow D^{h-1}_{i-1}[j]$
   OR
4. $D^h_i[j] \Leftarrow D^{h-1}_{i}[j-1]$

Il bit j-esimo della parola h e i, sarà 1 quando almeno 1 di questi 4 casi è 1.

$$D^h_i[j] = (D^h_{i-1}[j-1] \textbf{ AND } B_{T[i]}[j]) \textbf{ OR } D^{h-1}_{i-1}[j-1] \textbf{ OR } D^{h-1}_{i-1}[j] \textbf{ OR } D^{h-1}_{i}[j-1]$$

Calcolo di $D^h_i[j]$ ($h > 0, i > 0$)

$j > 1$



Passiamo al calcolo del primo bit della parola generica (j=1)

Calcolo di $D^h_i[1]$ ($h > 0, i > 0$)

$j = 1$

$D^h_i[1] = 1$ per $h > 0, i > 0$

Se h>0 sarà sempre 1 perché gli errori sono concessi.

Quando invece siamo nella posizione j

$$D^h_i[j] = (D^h_{i-1}[j-1] \text{ AND } B_{T[i]}[j]) \text{ OR } D^{h-1}_{i-1}[j-1] \text{ OR } D^{h-1}_{i-1}[j] \text{ OR } D^{h-1}_i[j-1]$$

$$D^h_i[1] = (D^h_{i-1}[0] \text{ AND } B_{T[i]}[1]) \text{ OR } D^{h-1}_{i-1}[0] \text{ OR } D^{h-1}_{i-1}[1] \text{ OR } D^{h-1}_i[0]$$

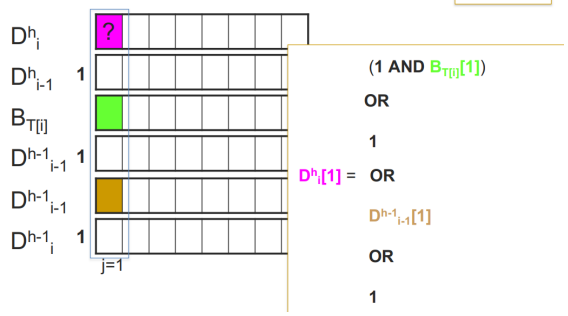
L'ultimo caso (posizione 0) non esiste. Quindi sostituisco brutalmente quella posizione 0 con 1.

$D^h_i[1] = 1$ per $h > 0, i > 0$

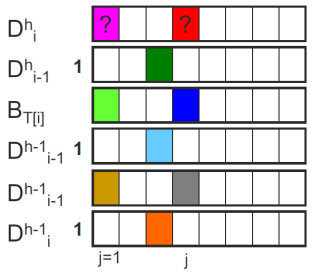
$$D^h_i[1] = \begin{array}{l} (1 \textbf{ AND } B_{T[i]}[1]) \\ \textbf{OR} \\ 1 \\ \textbf{OR} \\ D^{h-1}_{i-1}[1] \\ \textbf{OR} \\ 1 \end{array}$$

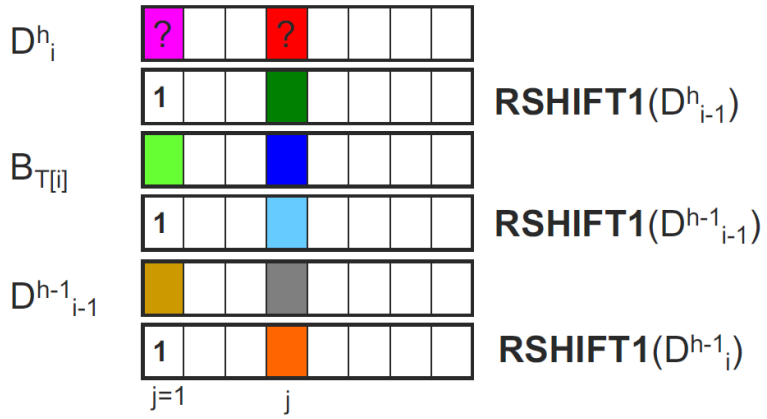
Sostituisco 1 al posto di $D^h_{i-1}[0]$, $D^{h-1}_{i-1}[0]$ e $D^{h-1}_i[0]$

Calcolo di $D^h_i[1]$ ($h > 0, i > 0$) — $j = 1$



Calcolo di $D^h_i[j]$ ($h > 0, i > 0$) — $j = 1$, $j > 1$





Quindi per $j>1$

E per $j=1$

$$D^h_i[j] = \begin{array}{l}(\textbf{RSHIFT1}(D^h_{i-1})[j] \textbf{ AND } B_{T[i]}[j]) \\ \quad\quad \textbf{OR} \\ \textbf{RSHIFT1}(D^{h-1}_{i-1})[j] \\ \quad\quad \textbf{OR} \\ D^{h-1}_{i-1}[j] \\ \quad\quad \textbf{OR} \\ \textbf{RSHIFT1}(D^{h-1}_{i})[j]\end{array} \qquad D^h_i[1] = \begin{array}{l}(\textbf{RSHIFT1}(D^h_{i-1})[1] \textbf{ AND } B_{T[i]}[1]) \\ \quad\quad \textbf{OR} \\ \textbf{RSHIFT1}(D^{h-1}_{i-1})[1] \\ \quad\quad \textbf{OR} \\ D^{h-1}_{i-1}[1] \\ \quad\quad \textbf{OR} \\ \textbf{RSHIFT1}(D^{h-1}_{i})[1]\end{array}$$

Quindi li possiamo mettere insieme (j≥1):

$$D^h_i[j] = \begin{array}{l}(\textbf{RSHIFT1}(D^h_{i-1})[j] \textbf{ AND } B_{T[i]}[j]) \\ \quad\quad \textbf{OR} \\ \textbf{RSHIFT1}(D^{h-1}_{i-1})[j] \\ \quad\quad \textbf{OR} \\ D^{h-1}_{i-1}[j] \\ \quad\quad \textbf{OR} \\ \textbf{RSHIFT1}(D^{h-1}_{i})[j]\end{array}$$

1. Inizializzazione della parola D^0_0=00…0
2. Calcolo di D^0_i per i da 1 a n (algoritmo BYG)
1. Per h da 1 a k:
    1. Inizializzazione di D^h_0
    2. Per i da 1 a n
        Calcolo di D^h_i
        Se h=k **AND** $D^k_i[m] = 1$
            output i

Questo è quindi l'algoritmo per la scansione del testo.

Non chiederà mai la formula di calcolo per la parola, non chiede di fare un'esecuzione su un testo. All'esame può fare domande precise sulla parola singola, ora facciamo esercizi, ci interessa l'idea dell'algoritmo, come sono fatte le parole D con i e h.

Esercizi

Esercizio 1

La parola 0100 è la parola D^0_i dell'algoritmo di ricerca approssimata di Wu e Manber. Dire se 1011 può essere la parola D^1_i.

No. Diamo attenzione al secondo bit della parola, quello a 1.

$$D^0_i[2] = 1 \Rightarrow P[1,2] = suff_0(T[1,i]) \Rightarrow P[1,2] = suff_1(T[1,i])$$

Questo bit indica che il prefisso lungo 2 del pattern ha distanza di edit minore uguale a zero con una sottostringa del testo che finisce in posizione i.

Quindi questo è vero anche quando la distanza di edit deve essere minore uguale ad 1.

$$P[1,2] = suff_1(T[1,i]) \Rightarrow D^1_i[2] = 1$$

RISPOSTA: **NO**

In generale,
$D^0_i[j] = 1 \Rightarrow D^h_i[j] = 1,\ h > 0$
$D^{h'}_i[j] = 1 \Rightarrow D^h_i[j] = 1,\ h > h'$

Questa è la generalizzazione. Per qualsiasi i e j, se il bit è 1 allora se l'h è maggiore sarà per forza 1 anche dopo.

Esercizio 2

La parola D^0_i dell'algoritmo di ricerca approssimata di Wu e Manber è uguale a 10100. Indicare quali bit della parola D^1_i è possibile dedurre da D^0_i.

D^0_i = **1**0**1**00

In generale:

$D^0_i[j] = 1 \Rightarrow D^1_i[j] = 1$

$D^0_i[1] = 1 \Rightarrow D^1_i[1] = 1$
$D^0_i[2] = 1 \Rightarrow D^1_i[2] = 1$

$\Rightarrow D^1_i$ = **1**?**1**??

Innanzitutto abbiamo quanto detto nell'esercizio precedente, e abbiamo quindi questi.

Possiamo dire qualcosa sui bit vicini? (quindi quello in mezzo ai due 1, e quello subito a destra)

$$D^0_i[1] = 1 \Rightarrow P[1,1] = suff_0(T[1,i]) \Rightarrow P[1,2] = suff_1(T[1,i])$$
$$P[1,2] = suff_1(T[1,i]) \Rightarrow D^1_i[2] = 1$$

Se analizziamo il primo 1 abbiamo che il prefisso lungo 1 del pattern ha distanza di edit al più 0 (uguale) ad una sottostringa del testo che finisce in i. Questo implica che il prefisso lungo 2 del pattern ha distanza di edit che è ≤1 con una sottostringa che finisce in i. Quindi il bit di posizione 2 è 1.

$$D^0_i[3] = 1 \Rightarrow P[1,3] = suff_0(T[1,i]) \Rightarrow P[1,4] = suff_1(T[1,i])$$
$$P[1,4] = suff_1(T[1,i]) \Rightarrow D^1_i[4] = 1$$

Nello stesso modo anche il bit 4 sarà 1.

In generale:

$D^0_i[j] = 1 \Rightarrow D^1_i[j+1] = 1$

In generale:

$D^h_i[j] = 1 \Rightarrow D^{h+1}_i[j+1] = 1$