Tabelle multilivello

G. Lettieri

15 Maggio 2021

1 MMU_1: tabella su più livelli

Proviamo a calcolare quanto sono grandi le tabelle di corrispondenza usate dalla Super-MMU. La memoria virtuale è almeno di 2^{48} byte[1] e ogni pagina è grande 2^{12} byte, quindi la memoria virtuale contiene

$$\frac{2^{48}}{2^{12}} = 2^{36} = 64\,\text{Gi pagine.}$$

La tabella di corrispondenza di ogni processo deve avere una entrata per ognuna di queste pagine. Ogni entrata deve contenere almeno i bit P, R/W, U/S, PCD, PWT, A, D e il numero di frame che fornisce i bit da 12 a 51 dell'indirizzo fisico, per un totale di 43 bit, arrotondati in 6 byte. Se poi vogliamo che la dimensione di ogni entrata sia una potenza di 2 dovremo usare almeno 8 byte. In conclusione, la tabella di corrispondenza dovrà essere di

$$64\,\text{Gi} \times 8\,\text{B} = 512\,\text{GiB}.$$

Difficilmente, quindi, possiamo pensare di avere un dispositivo di memoria che possa contenere anche una sola di queste tabelle.

Per affrontare il problema notiamo che la stragrande maggioranza dei programmi ha bisogno soltanto di una piccola frazione dei 2^{48} byte disponibili di memoria virtuale. Vorremmo avere una struttura dati che contenga le sole entrate effettivamente utilizzate.

Introduciamo quindi MMU_1, una MMU del tutto identica alla Super-MMU, tranne che per il formato della tabella di corrispondenza. Come la Super-MMU, MMU_1 possiede una memoria interna in cui salvare le tabelle di corrispondenza e un registro, **`cr3`**, che serve ad individuare la tabella di corrispondenza attiva ad ogni istante. La speranza è di poter usare una memoria interna molto più piccola rispetto a quella richiesta dalla Super-MMU.

[1] Come detto precedentemente, omettiamo di considerare il caso di memoria virtuale grande 2^{57} byte.

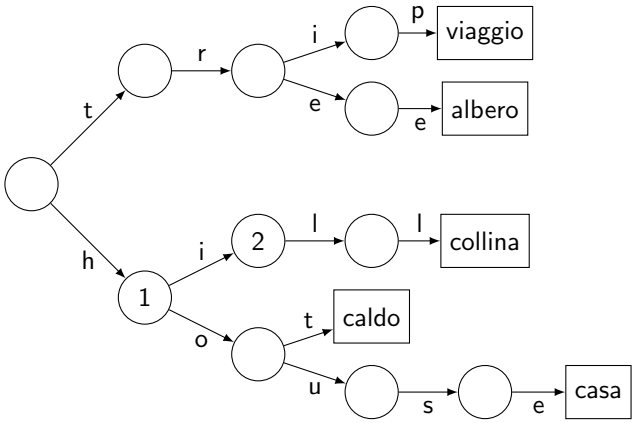

Figura 1: Un esempio di trie con chiavi e valori di tipo stringa.

1.1 La struttura dati *trie*

La struttura dati utilizzata da MMU_1 è un *bitwise trie*, che è una variante di trie. I trie sono strutture dati ad albero che permettono di mappare chiavi di tipo stringa in valori, in modo che i caratteri successivi della chiave guidino la ricerca all'interno dell'albero. Consideriamo, per esempio, il trie mostrato in Figura 1. L'albero memorizza le associazioni trip $\mapsto$ viaggio, tree $\mapsto$ albero, hill $\mapsto$ collina, hot $\mapsto$ caldo e house $\mapsto$ casa. Si noti come gli archi dell'albero siamo marcati con i caratteri delle chiavi e il valore associato ad ogni chiave si trovi nella foglia che si raggiunge partendo dalla radice e seguendo il percorso indicato dalla chiave.

Un modo di implementare un trie è di avere, in ogni nodo dell'albero, un array di 128 entrate, ciascuna delle quali contenga il puntatore al prossimo nodo da visitare in base al codice ASCII del prossimo carattere della chiave.

Ogni nodo si trova sul percorso di tutte le chiavi che iniziano con lo stesso prefisso. Per esempio, il nodo marcato con "2" si trova nel percorso di tutte le chiavi che iniziano con "hi". Un puntatore nullo nell'array di un nodo indica che il trie non contiene chiavi con il corrispondente prefisso. Per esempio, una ricerca della chiave "history" nel trie di Figura 1 seguirebbe il ramo "h" dalla radice per arrivare al nodo "1", quindi il ramo "i" per arrivare al nodo "2". Qui troverebbe un puntatore nullo associato al carattere "s" e la ricerca si concluderebbe con un fallimento. L'inserimento di una nuova associazione chiave/valore nel trie comporta una visita dell'albero come in una ricerca, ma creando eventuali nodi mancanti fino alla foglia che deve contenere il valore.

Nel nostro caso la chiave è il numero di pagina e il valore che vi vogliamo associare è il corrispondente numero di frame. Per questo scopo possiamo usare un *bitwise* trie, che funziona esattamente come un trie ma, al posto dei caratteri, usa gruppi di bit della chiave. In particolare, il numero di pagina è composto da 36 bit che possiamo raggruppare in 4 gruppi di 9 bit. Ogni nodo del bitwise trie

conterrà dunque una tabella di $2^9 = 512$ entrate con puntatori, eventualmente nulli, al nodo successivo. Le foglie stesse possono essere tabelle indicizzate dall'ultimo gruppo di 9 bit della chiave. In questo caso le entrate delle tabelle foglie conterranno il numero di frame associato al numero di pagina. Si arriva dunque alla struttura dati illustrata in Figura 2. Ciascun nodo dell'albero di Figura 2, foglie incluse, è una tabella di 512 entrate, ciascuna grande 8 byte. Ogni tabella è grande dunque 4096 byte. L'albero ha al massimo 4 livelli, che per convenzione vengono numerati da 4 a 1 (il livello delle foglie), in modo da poter parlare di tabelle di livello 4, di livello 3 e così via. È molto comodo rappresentare i numeri di pagina in base 8, in quanto ciascuna cifra in base 8 corrisponde a 3 bit del numero di pagina, e dunque ciascun gruppo di 9 bit può essere rappresentato da 3 cifre in base 8. Per esempio, supponiamo che la MMU_1 si trovi a dover tradurre l'indirizzo virtuale $v = (000\ 777\ 000\ 777\ 1234)_8$. Il numero di pagina è $(000\ 777\ 000\ 777)_8$. I primi 9 bit del numero di pagina sono $(000)_8$. La MMU_1 usa quindi l'entrata di indice 0 (vale a dire la prima entrata) della tabella di livello 4 per trovare la prossima tabella da consultare. La MMU_1 passerà dunque alla tabella di livello 3 in alto in Figura 2. Qui usa i successivi 9 bit, $(777)_8$ per sapere quale entrata di questa tabella deve consultare per raggiungere la prossima tabella, di livello 2. Dunque la MMU_1 userà l'ultima entrata della tabella e passerà alla seconda tabella di livello 2 dall'alto in Figura 2. Qui userà i bit $(000)_8$ del terzo gruppo e proseguirà verso la terza tabella di livello 1 dall'alto. Infine, troverà la traduzione che stava cercando nell'entrata $(777)_8$ di questa tabella.

Il formato delle entrate delle tabelle di livello 1 è mostrato in Fig. 3 e rispecchia quello dell'architettura Intel/AMD a 64 bit. Chiameremo queste entrate *descrittori di pagina virtuale* o anche *descrittori di livello 1*, essendo contenuti nelle tabelle di livello 1. Si noti che i descrittori contengono tutte le informazioni che abbiamo già introdotto parlando della Super-MMU. I bit P, R/W, U/S, PWT e PCD sono scritti dalla routine di sistema che attiva il processo (nel nostro caso, `activate_p()`) e soltanto letti dalla MMU_1. I bit A e D, invece, sono letti e scritti sia dal software (di sistema) che dalla MMU_1.

Ogni volta che il sistema carica le pagine di un processo in memoria (alla prima attivazione, oppure dopo uno *swap-in* in seguito ad uno *swap-out*), dovrebbe porre D=0 in tutte le entrate della tabella di corrispondenza. Al momento di eseguire uno swap-out del processo, il sistema può evitare di salvare tutte le pagine del processo nel dispositivo di swap ri-esaminando le entrate e notando in quali di esse il bit D è diventato 1: queste puntano a pagine che sono state modificate e vanno risalvate; per tutte le altre il salvataggio si può evitare, in quanto la copia già presente nello swap è ancora valida. Il bit A può essere usato per capire quali pagine/tabelle sono più usate o sono state usate più recentemente, ed è di ausilio soprattutto all'implementazione della paginazione su domanda.

Tutte le tabelle di livello 2, 3 e 4 hanno lo stesso formato. Ciascuna di esse contiene 512 entrate con il formato illustrato in Fig. 4. Il formato è simile a quello dei descrittori di livello 1 mostrati in Fig. 3: ci sono ancora i bit P, R/W e U/S e A, nelle stesse posizioni dei bit omonimi di Fig. 3; il campo "Indirizzo

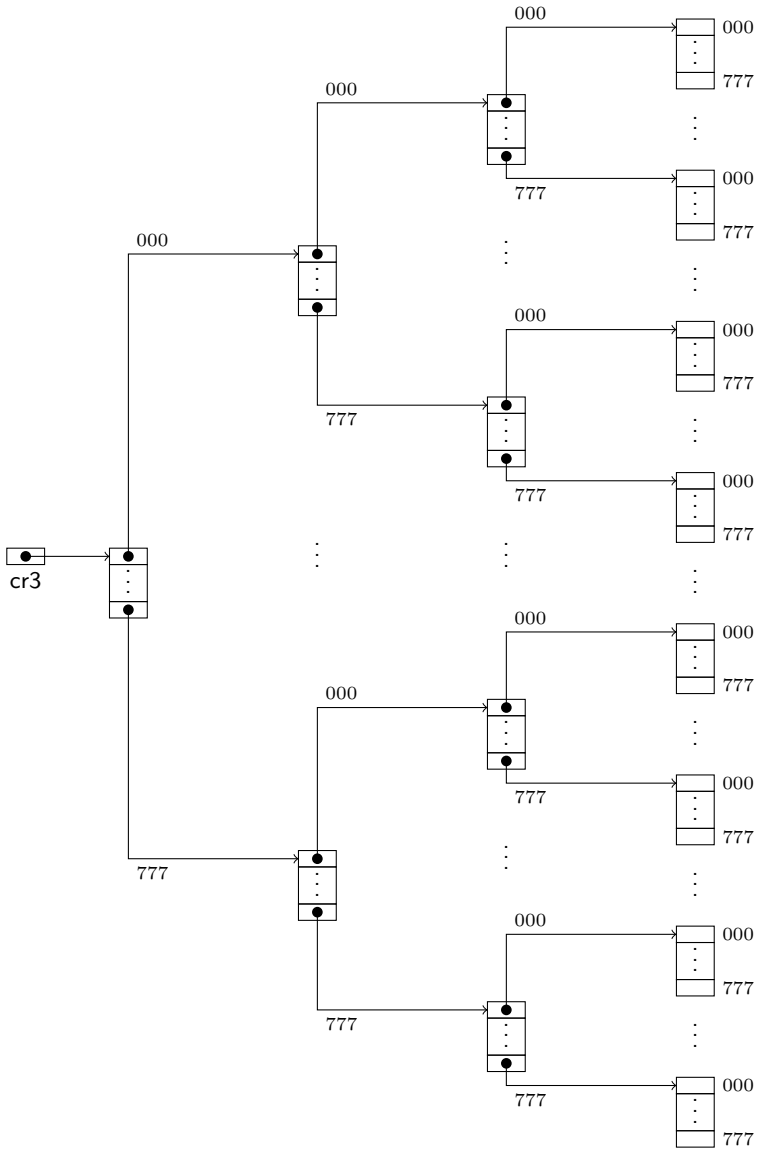

Figura 2: Tabella di corrispondenza su 4 livelli, implementata con un bitwise trie.

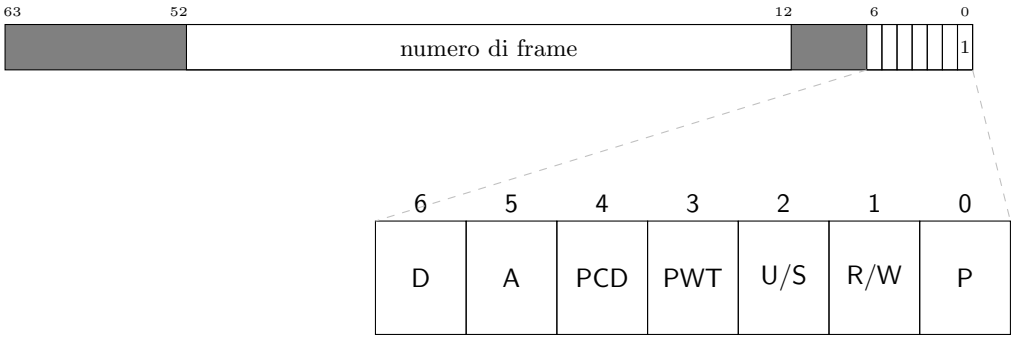

Figura 3: Descrittore di pagina virtuale (tabelle di livello 1).

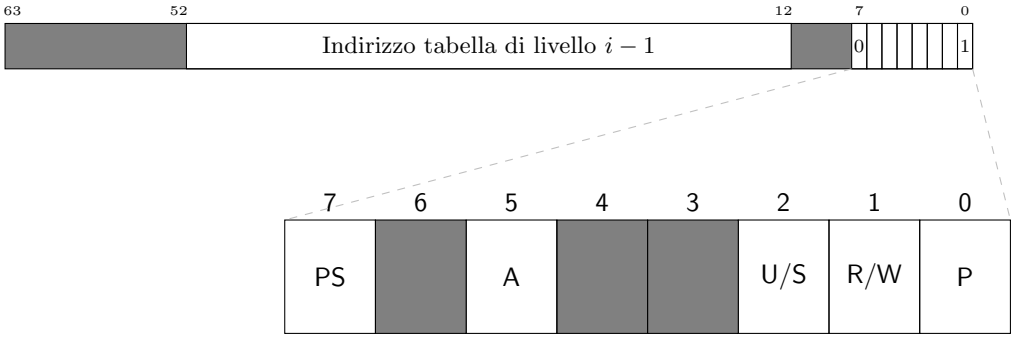

Figura 4: Descrittore di livello i, con $i = 2, 3, 4$.

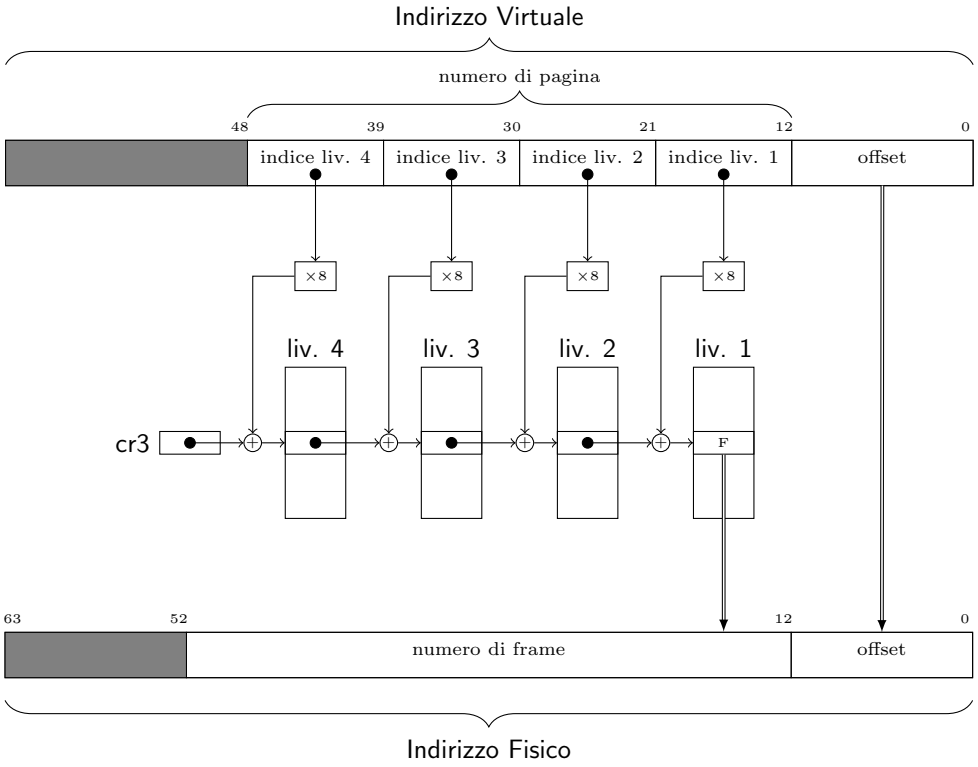

Figura 5: Traduzione da indirizzo virtuale a fisico (pagine di 4 KiB).

tabella di livello $i-1$" occupa la stessa posizione del campo "numero di frame" di Fig. 3; per il momento non ci preoccupiamo del nuovo bit PS e assumiamo che valga 0. Si noti che all'indirizzo della tabella mancano i bit da 0 a 11: la MMU_1 assume che questi bit siano 0, cioè che tutte le tabelle partano da indirizzi che sono multipli di 4 KiB (allineamento naturale). Questo vale anche per la tabella di livello 4: i 12 bit meno significativi di **`cr3`** devono essere tutti a 0.

Chiameremo le entrate delle tabelle di livello 2 *descrittori di livello 2* o *descrittori delle tabelle di livello 1*. Si noti il decremento del livello: i descrittori di livello 2 sono contenuti in tabelle di livello 2 e descrivono tabelle di livello 1. Allo stesso modo chiameremo le entrate delle tabelle di livello 3 *descrittori di livello 3* o *descrittori delle tabelle di livello 2* e, infine, chiameremo le entrate della tabella di livello 4 *descrittori di livello 4* o *descrittori delle tabelle di livello 3*.

Anche se simili, i descrittori di livello 2, 3 e 4 non vanno confusi con i descrittori di livello 1: i primi descrivono tabelle, mentre quelli di livello 1 descrivono pagine. I descrittori di tabella servono a trovare le tabelle di livello inferiore, ma solo le tabelle foglia contengono la traduzione cercata.

In Fig. 5 abbiamo riassunto il processo di traduzione operato dalla MMU_1,

mostrando più in dettaglio le operazioni svolte da MMU_1, assumendo che tutti i bit P valgano 1. Al primo passo MMU_1 deve leggere il corretto descrittore di livello 4, che si trova nella sua memoria. Per farlo deve eseguire una operazione di lettura in memoria, ovviamente specificando l'indirizzo. La tabella di livello 4 è un vettore di descrittori, ciascuno grande 8 byte, e la MMU_2 vuole leggere il descrittore il cui indice è contenuto nei bit 39–47 di V (indice di livello 4), sia i_4. L'indirizzo a cui vuole leggere è dunque $\texttt{cr3} + i_4 \times 8$. Si noti che, per quanto detto sull'allineamento naturale delle tabelle, questa operazione non comporta una vera somma, ma solo una concatenazione di bit. Anche la moltiplicazione per 8 consiste, ovviamente, nella concatenazione con tre bit costanti pari a 0. Letto il descrittore di livello 4, MMU_2 ne estrae il campo indirizzo (bit 12–51), lo concatena con i bit 30–38 di V e con altri tre bit a 0, ottenendo così l'indirizzo del descrittore di livello 3. E così anche per i descrittori di livello 2 e 1. Arrivata al livello 1, la MMU_2 estrae il campo F (bit 12–51), lo concatena con l'offset di V e ottiene così la traduzione.

Durante la traduzione la MMU_1 esegue anche altri compiti, analoghi ai compiti aggiuntivi che svolgeva la Super-MMU, ma applicati alla struttura multi-livello:

- controlla tutti i bit R/W: una operazione di scrittura è permessa solo se tutti e 4 i bit lungo il percorso la permettono;
- controlla tutti i bit U/S: una operazione (di lettura o scrittura) è permessa solo se tutti e 4 i bit lungo il percorso la permettono;
- passa al controllore cache le informazioni contenute nei bit PWD e PCD nel descrittore di livello 1;
- pone a 1 tutti e 4 i bit A incontrati, se non lo erano già;
- in caso di scrittura, pone a 1 il bit D nella tabella di livello 1 (i descrittori di livello maggiore di 1 non hanno il bit D).

Se uno qualunque dei bit P incontrati durante la traduzione vale 0, la MMU_1 smette di tradurre e solleva una eccezione di page fault. La routine di sistema che gestisce il fault terminerà il processo con un errore.

Ciascuna delle tabelle di corrispondenza dalla Super-MMU deve essere sostituita da uno di questi alberi (in altre parole, ci serve un albero per ogni processo). Nella parte bassa di Figura 2 abbiamo riportato il numero massimo di tabelle per ogni livello del trie e quanto spazio occupa ogni livello. Si noti che le tabelle di livello 1 occupano complessivamente 512 GiB, esattamente come la tabella della Super MMU che stiamo cercando di sostituire. In effetti, se rimettessimo insieme tutte le tabelle di livello 1, in ordine, riotterremmo la tabella di corrispondenza della Super-MMU. Siccome a questo spazio dobbiamo aggiungere quello richiesto dai livelli superiori dell'albero, ci rendiamo conto che il trie completo occupa *più* spazio della tabella di corrispondenza originaria. Anche dal punto di vista del tempo richiesto per eseguire la traduzione ci stiamo perdendo: la tabella unica della Super-MMU richiede un unico accesso in memoria,

mentre il trie ne richiede 4. Qual è dunque il vantaggio di passare al trie? Un vantaggio è che possiamo evitare di allocare le parti di trie relative a indirizzi virtuali che il processo non usa. Per esempio, se un processo non usa nessun indirizzo il cui numero di pagina inizi con $(777)_8$, il trie di questo processo non ha bisogno di tutto il sottoalbero inferiore di Figura 2. L'omissione di questo sottoalbero si ottiene semplicemente ponendo a 0 il bit P dell'entrata $(777)_8$ della tabella di livello 4. Dal momento che la maggior parte dei processi userà soltanto una piccola parte dei possibili indirizzi virtuali, ci rendiamo conto che in questo modo il trie avrà quasi sempre una dimensione ragionevole.

1.2 Regioni e sottoregioni

Un altro modo per pensare alle operazioni svolte dalla MMU_1 è di ragionare in termini di regioni naturali (che, ricordiamo, sono intervalli di indirizzi con dimensione pari ad una potenza di 2 e allineate naturalmente). Possiamo identificare ciascuna tabella del trie specificando la sequenza di bit della chiave che porta dalla radice alla tabella in questione. Per esempio, la terza tabella di livello due dall'alto in Figura 2 è identificata dalla sequenza di 18 bit $(777\,000)_8$. La traduzione di tutti gli indirizzi virtuali che iniziano con questo prefisso deve passare da questa tabella. Questa tabella, dunque, è "responsabile" della traduzione dell'intera regione naturale, grande $2^{48-18} = 2^{30} = 1\,\mathrm{GiB}$, il cui numero di regione è appunto $(777\,000)_8$. Aggiungendo ulteriori 9 bit possiamo identificare anche ogni singola *entrata* della tabella. Per esempio, i 27 bit $(777\,000\,777)_8$ identificano sia la terza tabella di livello 1 dal basso di Figura 2, chiamiamola t, sia l'ultima entrata della seconda tabella di livello 2 dal basso, chiamiamola e. Di nuovo, la traduzione di tutti gli indirizzi virtuali che iniziano con $(777\,000\,777)_8$ deve passare dall'entrata e e poi da una dalle entrate della tabella t. Tutti questi indirizzi virtuali appartengono alla stessa regione naturale grande $2^{48-27} = 2^{21} = 2\,\mathrm{MiB}$, il cui numero di regione è $(777\,000\,777)_8$. Possiamo dire che l'entrata e, o l'intera tabella t, sono responsabili della traduzione in questa regione.

In generale, diremo che ogni entrata di una tabella di livello i, con $1 \leq 4$, sarà responsabile della traduzione di una regione naturale di livello $i-1$. Invece ogni tabella di livello i, nella sua interezza, sarà responsabile della traduzione di una regione naturale dello stesso livello i. Le regioni di livello 0 non sono altro che le pagine, grandi 2^{12} byte. In generale una regione di livello j, con $0 \leq j \leq 4$, è grande 2^{9j+12} byte. Quindi, ogni entrata di una tabella di livello 1 è responsabile della traduzione di una ben precisa pagina, mentre una intera tabella di livello 1, nel suo complesso, è responsabile della traduzione di una ben precisa regione di livello 1, grande $2^{9\times1+12}\,\mathrm{B} = 2\,\mathrm{MiB}$, la stessa regione di cui è responsabile l'entrata (in una tabella di livello 2) che punta alla tabella nel trie. E così via.