Tripla orientata

Una volta riusciti a trovare l'insieme di appartenenza della key in tempo costante, ci siamo accorti che avremmo potuto distribuire gli elementi nell'array in modo omogeneo, lasciando il numero di elementi per cella basso. Se all'interno della cella del vettore c'è una lista, l'inserimento risulta essere più efficiente in quanto non dipende dagli elementi inseriti prima, però il search teoricamente è più lento poiché deve scorrere tutta la lista (nel peggiore dei casi), ma il problema non si pone perché essa può essere massimo di 5 elementi. Nell'albero avl, invece, ad ogni inserimento bisogna ribilanciare la struttura con una certa probabilità, ma il search è nettamente più veloce per le proprietà degli avl stessi. Questo ci permette di dire che lavorare con le liste senza fare il cambio in avl risulterebbe più conveniente ai fini della velocità dell'algoritmo, anche perché il passaggio da lista ad avl è una operazione che richiede il suo tempo.

Se noi sapessimo in anticipo le chiavi degli elementi che andremmo ad inserire (in realtà ci basterebbe il numero di essi e la distanza media tra una chiave e l'altra) potremmo progettare un algoritmo che calcoli una tripla adatta ad esse che renda le operazioni più efficienti. È proprio quello che fa la funzione tripleGeneratorOriented. Dato in input il vettore che contiene tutte le chiavi che si andranno ad inserire, la funzione calcola la distanza media tra una chiave e l'altra. Abbiamo detto in precedenza che usare le liste senza trasformarle in avl risulterebbe più efficiente. Tuttavia le condizioni del problema ($b > r = 6$) ci impediscono di usare solo le liste nel caso in cui la distanza media tra le key è 1. Questo perché b rappresenta il range di valori che può contenere una cella. È chiaro che nel caso in cui la distanza tra gli elementi sia > 1 per fare in modo che ogni cella contenga all'incirca sempre lo stesso numero di elementi, bisogna fare in modo che il range di valori che può contenere la cella sia proporzionale alla distanza media tra i singoli elementi. Per fare un esempio, nel caso di 10 elementi con distanza 2 uno dall'altro avremmo un vettore di questo tipo [0, 2, 4, …, 18]. Ora se volessimo mettere 5 elementi per ogni cella il range di valori sarà 5 * 2 = 10, dove 2 è la distanza tra gli elementi. Tornando al discorso di prima, se la distanza fosse 1 il range, che equivale a b, non potrà essere minore di 7, quindi dovrà per forza avere un avl per ogni cella del vettore.

Il numero di celle (d) del vettore è dato dal rapporto tra max-min e b. Per ottenere il d ideale per inserire 7 elementi per ogni cella, è chiaro che dovremmo dividere il numero di elementi totali per 7, e ottenere quindi n/7 celle del vettore. Dato che b = 7 * distanzaMedia, e d deve essere n/7 si può dedurre che il valore di max-min dovrà essere n * distanzaMedia (Ricavato dalla formula d = (max-min) / b).

Ora per trovare il minimo (e di conseguenza il massimo) bisogna fare un ragionamento molto semplice. Per definire gli insiemi di appartenenza delle key (ad esclusione degli ultimi due), il minimo gioca un ruolo fondamentale in quanto definisce l'inizio (la fine è determinata da b) di essi. Ovviamente la variabile i = {0, 1, …, d – 1} serve a spostare l'inizio di ogni insieme del valore della grandezza dello stesso (b). Quindi il minimo dovrà essere proprio il più piccolo elemento tra quelli che vogliamo inserire; in questo modo partiremo ad inserire gli elementi dal più piccolo nel primo insieme e il più grande dell'ultimo (insiemi d e d+1 esclusi).

Una volta ricavato il minimo possiamo ricavare di conseguenza il valore di max che ci permetterà di avere finalmente la tripla completa (min, max, b) adatta ai nostri scopi.