BONUS -

Studio della struttura dati

Questo documento è stato creato per la mia soddisfazione personale e non fa parte della relazione ufficiale. Gentilmente, la prego di non sentirsi in obbligo a leggerlo e di non considerarlo nella valutazione del progetto, ringraziando anticipatamente per la comprensione.

Legenda

- N : numero di libri nella biblioteca
- M : dimensione media di un libro
- C : massimo numero di campi in una biblioteca
- T: tutti i nodi che in cui c'è un match con il primo campo:valore della richiesta
- cR: numero dei campi di una richiesta
- m: dimensione media di una stringa libro
- sC: dimensione media di una stringa campo
- sV: dimensione media di una stringa valore

Complessità

Creazione

Anche se è possibile che i libri siano scritti nel file record in modo ordinato per un campo, è impossibile che siano ordinati per più di un campo, perciò possiamo ignorare il caso peggiore in cui ci sia un inserimento in ordine nell'albero. Allora il numero medio di operazioni in aggiungi_libri() è

$$(C + log\, n) * C = C^2 + C\, log\, n$$

*(dove n è il numero di libri inseriti fino a quel momento)

La spiegazione è che per inserire una coppia valore-libro bisogna prima cercare il campo giusto e questo richiede massimo C operazioni. L'inserimento costa log n. L'operazione si ripete per il numero di campi da inserire.

Considerato che aggiungi_libri() viene chiamata per ogni libro nella biblioteca, avremo:

$$\sum_{n=1}^{N} C^2 + C\, log\, n = N * C^2 + C\,(N * log\, N - N)$$

*1_n è il Pochammer symbol (fonte Wolframalpha)

All’aumentare dei libri di una biblioteca il numero dei possibili campi ha una crescita così minore da poter considerare C una costante, per N molto grandi.
Questo vuol dire che per biblioteche abbastanza grandi l'efficienza diventa:*

$$O(N) + O(N \log N) = O(N \log N)$$

Ricerca

Il numero di passaggi (nel caso migliore) della funzione chiedi_libri() è:

log N + (T * cR)

con T=1 e cR=1

perché prima cerchiamo in uno degli alberi binari tutti i possibili libri che combaciano con la richiesta e, successivamente, per ogni libro trovato dobbiamo scorrere la richiesta e controllare che combaci effettivamente con il libro.

Nel caso peggiore invece abbiamo

N + (T*cR)

con T=N e cR = C

quindi N * (C+1)

Memoria

- **Libro**: Ogni struct libro ha un puntatore alla stringa del libro, due short e un puntatore ad una data contenente 8 short. Quindi: m + 28B.
- **Array Campi:** ogni elemento ha un puntatore alla stringa campo e un puntatore alla testa di un ABO. Quindi: sC + 8B.
- **ABO di info**: ogni nodo ha un puntatore alla stringa valore, un puntatore ad un libro e due puntatori ad altri nodi. Quindi: sV + 16B.

In totale quindi la memoria occupata è:
$N * (m + 28B) + C * (sC + 8B) + C * N * (sV + 16B)$

Possiamo definire $m \simeq C * sC + C * sV$ in quanto la dimensione di una stringa è la somma di tutte le coppie campo-valore. Alla fin fine, sC e sV sono più o meno uguali, per cui possiamo dire che $m \simeq 2 * C * sV$ ovvero che $C * sV \simeq \frac{m}{2}$

ciò vuol dire che possiamo riscrivere l’equazione come:

$N * (m + 28B) + \frac{m}{2} + C * 8B + N * (\frac{m}{2} + C * 16B)$

$(m + 28B + \frac{m}{2} + C * 16B) * N + \frac{m}{2} + C * 8B$

Proviamo quindi a calcolare m:
$m = (dim(bib1.txt) + dim(bib2.txt) + dim(bib3.txt) + dim(bib4.txt) + dim(bib5.txt))/totale\ libri$
$= (4,1kB + 2,8\,kB + 960\,bytes + 2,8\,kB + 1,3\,kB)/46$
$= 12224B/46 = 156B$ (arrotondato per eccesso)

e stabiliamo per semplicità che C non sia mai più di 20, per quanto grande la biblioteca.
Allora possiamo finalmente calcolare una stima della dimensione del file record dato il numero di libri:
$(156B + 28B + 156B/2 + 20 * 16B) * N + 156B/2 + 20 * 8B$
$= 582B * N + 238B$

Considerando che la dimensione di un libro è 156B, il rapporto tra la dimensione della struttura dati e il file record è 582/156=3,7

Confronto

Complessità e memoria di una struttura dati semplice

Considerando che i data test ricevuti sono molto piccoli, si poteva pensare di utilizzare un approccio al problema molto semplice: copiare i libri uno ad uno. In questo modo avremmo avuto una complessità di N per la costruzione della struttura. Per la ricerca lineare avremmo per forza di cose costruito un algoritmo che per ogni libro confronta tutte le coppie campo-valore della richiesta, ovvero N * cR. Questo vuol dire che al caso migliore sarebbe N e al caso peggiore sarebbe C*N.
Inoltre la memoria sarebbe rimasta uguale a quella del file record originale.

Complessità my_struttura vs struttura_semplice

Formule

My struttura dati

- Creazione: $N * C^2 + C\,(N\,log\,N - N)$
- Ricerca caso migliore: log N +1
- Ricerca nel caso peggiore: N * (C+1)
- Memoria: $582B * N + 238B$

Struttura dati semplice

- Creazione: N
- Ricerca caso migliore: N
- Ricerca nel caso peggiore: N * C
- Memoria: 156B* N

assumeremo per semplicità che C=15

Tabelle paragone

N=	Creazione		Memoria	
-	My	Simple	My	Simple
1	210	1	840B	156B
1K	360K	1K	582KB	156KB
1M	509M	1M	582MB	156MB
1G	658G	1G	582GB	156GB

-	Ricerca			
-	My		Simple	
N=	caso migliore	caso peggiore	caso migliore	caso peggiore
1	1	16	1	15
1K	10	16K	1K	15K
1M	20	16M	1M	15M
1G	30	16G	1G	15G

Conclusioni

Aspettative del tempo di ricerca

Nonostante il caso peggiore del tempo di ricerca della struttura dati scelta sia leggermente peggiore di quella di una semplice struttura dati per array, è anche vero che il caso migliore è invece migliore dell'ordine di grandezza di 10^9. È importante notare anche come col crescere del numero di libri in una biblioteca l'altezza dell'albero binario ordinato tenderà sempre di più a log N che a N. Inoltre con l'introduzione di nuovi libri diventa sempre più

probabile che il sottoinsieme T sia più piccolo in rapporto ad N, migliorando ancora di più il tempo di ricerca.

Pro e Contro finali

Per grandi dimensioni, questa struttura dati permette di avere un tempo di ricerca molto più rapido, a costo dello spazio di memoria utilizzato e della velocità di creazione. In particolare la velocità di creazione è più lenta di un fattore di log N, mentre la memoria è aumentata di ben 3,7 volte