Gabriele Palazzolo (Mat: 883434) - Michela Rivautella (Mat: 881496)

Laboratorio di Algoritmi e Strutture Dati - Corso B

MACCHINA UTILIZZATA

Modello: MacBook Pro 13-inch, 2017
Processore: 2,3 GHz Intel core i5 Dual-core
Memoria: 8 GB 2133 MHz LPDDR3

INSERIMENTO ASSOCIAZIONI

	Caso peggiore temporalmente	**Caso medio temporalmente**	**Caso ottimo temporalmente**
Hash table	$\Theta(n)$	$O(1)$	$O(1)$
Quick sort	$\Theta(n^2)$	$\Theta(n\log_2 n)$	$\Theta(n\log_2 n)$

N.B.: *n* del caso peggiore della *Hash table* si riferisce al numero di associazioni già inserite nella tabella che compongono la lista in una specifica posizione corrispondente al valore trovato tramite un metodo. Quindi per effettuare l'inserimento è necessario scorrere *n* associazioni (vedi sotto paragrafo *Hash table scelta*).

RICERCA DELLA CHIAVE

	Caso peggiore temporalmente	**Caso medio temporalmente**	**Caso ottimo temporalmente**
Hash table (con concatenamento)	$O(n)$	$\Theta(1+\alpha)$	$O(1)$
Binary search	$O(\log_2 n)$	$O(\log_2 n)$	$O(1)$

N.B.: α corrisponde al numero medio di elementi da percorrere nella lista ($\Theta(\alpha)$).

ANALISI TEMPI DI INSERIMENTO E RIORDINAMENTO

I test vengono effettuati con *n* associazioni di tipo <*chiave, valore*>, prese dal file *hashes.csv*:

	1 000 associazioni	**2 000 associazioni**	**5 000 associazioni**	**6 321 078 associazioni**
Hash table	0.73 msec.	1.33 msec.	3.4 msec.	3.49 sec.
Quick sort	0.53 msec.	1.3 msec.	3 msec.	3.31 sec.

Per effettuare i test si è così strutturata la lettura:

- *hash table*: vengono lette le *n* associazioni da file e inserite in un array statico (con dimensione 6 321 078) appositamente realizzato. Ogni associazione presente nel vettore viene inserita in un oggetto di tipo Hash Map;
- *quick sort*: vengono lette le *n* associazioni da file e inserite in un array statico (con dimensione 6 321 078) appositamente realizzato e soltanto al termine di questa fase viene applicato l'algoritmo di sort.

Dalla tabella si evince che l'algoritmo di *Quick sort* impiega minor tempo nell'ordinamento delle associazioni, tuttavia risulta migliore l'algoritmo della *Hash table,* in quanto esso prevede inserimento e ordinamento ad ogni sua chiamata all'interno di un oggetto di tipo *Hash table*. Mentre il *Quick sort* si occupa di effettuare un ordinamento su tutto l'insieme di associazioni una sola volta.

Questi risultati sono conseguenza delle modalità di gestione della lettura dei dati e dell'approccio adottato per effettuarne l'inserimento e dunque l'ordinamento delle associazioni.
Si osserva che la differenza dei tempi dei due algoritmi è minima. Quindi questo porta a conclude che la *Hash table* nonostante venga chiamata n associazioni volte è estremamente efficiente.

Hash table scelta

Si è scelto di utilizzare *tavole hash con concatenamento,* adottando la strategia delle *liste di trabocco* (semplici). In questo modo si è andati a concatenare gli elementi in collisione. Il metodo scelto per effettuare l'inserimento è stato "il metodo della divisione".
Quindi per ogni associazione viene calcolato un hash secondo:

$$h(k) = k \bmod(m)$$

dove *k* = key dell'associazione da inserire e *m* = nel nostro caso corrisponde al numero di associazioni lette in input.

La scelta di effettuare l'inserimento da tastiera del modulo con il quale effettuare il calcolo è stata fatta per dimostrare che qualunque questo valore sia, il metodo e l'inserimento vengono effettuati correttamente. In ogni caso questo non determina la dimensione dell'oggetto *Hash table,* in quanto sfruttando le liste, risulta essere una struttura dinamica.

La struttura dell'oggetto *Hash table* realizzata, è in realtà ottimizzabile, in quanto l'inserimento di una nuova associazione nel caso in cui nella posizione di *hash* calcolata è già presente una lista, prevede che essa venga percorsa tutta e venga effettuate un'allocazione al fondo di essa. Questa strategia comporta che nel caso peggiore la complessità di inserimento di un'associazione sarà $\Theta(n)$.
La si potrebbe quindi migliorare andando ad aggiungere ogni nuova associazione in testa alla lista della posizione *hash* corrispondente (pila realizzata con la lista). Con questo approccio la complessità di ridurrebbe a O(1) (ottimale).

Osservazioni

La *Hash table* risulta migliore anche per la dinamicità, infatti se in qualunque momento si decidesse di effettuare l'inserimento di una nuova associazione, nel caso della *Hash table* sarà sufficiente calcolare la *hash* ed effettuare l'inserimento, con precedente allocazione di memoria, che se nella posizione calcolata non vi è già una lista, è diretto, altrimenti si percorrerà l'intera lista. Nel caso del *Quick sort* sarà, invece, necessario effettuare non solo nuovamente l'ordinamento di tutte le associazioni (controlli con la nuova associazione), ma anche allocare un array statico sufficientemente capiente per poter contenere tutte le coppie di valori.

ANALISI TEMPI DI RICERCA DI UNA CHIAVE CON SUCCESSO E SENZA SUCCESSO

Vengono generati 10 000 000 valori e il recupero dei valori associati alle chiavi viene effettuato sul file *hashes.csv*:

Hash table	**Successi**	**Insuccessi**	**Tempo**
1 000 associazioni	981	9999019	0.79 sec.
2 000 associazioni	2053	9997947	0.81 sec.
5 000 associazioni	4951	9995049	0.89 sec.
6 321 078 associazioni	6320745	3679255	3.03 sec.

Birary search	**Successi**	**Insuccessi**	**Tempo**
1 000 associazioni	981	9999019	1.67 sec.
2 000 associazioni	2053	9997947	1.87 sec.
5 000 associazioni	4951	9995049	2.14 sec.
6 321 078 associazioni	6320745	3679255	5.98 sec.

Dai tempi nella tabella possiamo notare che nella ricerca le *Hash table* sono più efficienti rispetto alla *Binary search*, questo deriva dalla complessità, nel caso medio abbiamo per la ricerca binaria $O(\log_2 n)$ mentre per la tabella *hash*:

1. elemento assente: corrisponde al tempo per individuare la lista unito al tempo per percorrerla $\Theta(1) + \Theta(\alpha) = \Theta(1 + \alpha)$;
2. elemento presente: rispetto al caso precedente viene introdotto il concetto di uniformità semplice ottenendo $\Theta(1) + \Theta(1 + \frac{\alpha}{2} + \frac{\alpha}{2n}) = \Theta(1 + \alpha)$.

In ogni caso la ricerca di un elemento con la chiave k richiede un tempo proporzionale alla lunghezza della lista $T[h(k)]$.

Per il *Binary search* l'array utilizzato è quello ordinato precedentemente con il *Quick sort*.