LEZIONE 2018/12/16 - TABELLE HASH

Gabor Galazzo 20024195 A.A. 2018/2019

Il programma *hash* permette di effettuare un test di valutazione delle performance effettive di operazioni di inserimento e ricerca all'interno di un dizionario implementato tramite **tabella hash** con gestione delle collisione tramite **lista di adiacenza**, **Indirizzamento aperto + scansione lineare** e **Indirizzamento aperto + doppio hash**.

> Uso: ./hash `in_file` `out_file` `table_dim (potenza di 2 || numero primo)`

L'applicativo procede con un for da i = 0.5 a 1.0 con un passo di 0.1; crea una **tabella hash** di dimenzione `table_dim` per ogni tipo di *gestione di collisioni*, nel nostro caso 3. Per ogni tabella legge dal file `in_file` `table_dim`**i* elementi e li inserisce, poi effettua le ricerche salvando il numero medio di **hits** per **fattore di carico (α = table_dim*i)** in `out_file`.

Ai fini della consegna prenderò in considerazione solo i risultati derivanti l'utilizzo di una tabella hash con lista di adiacenza

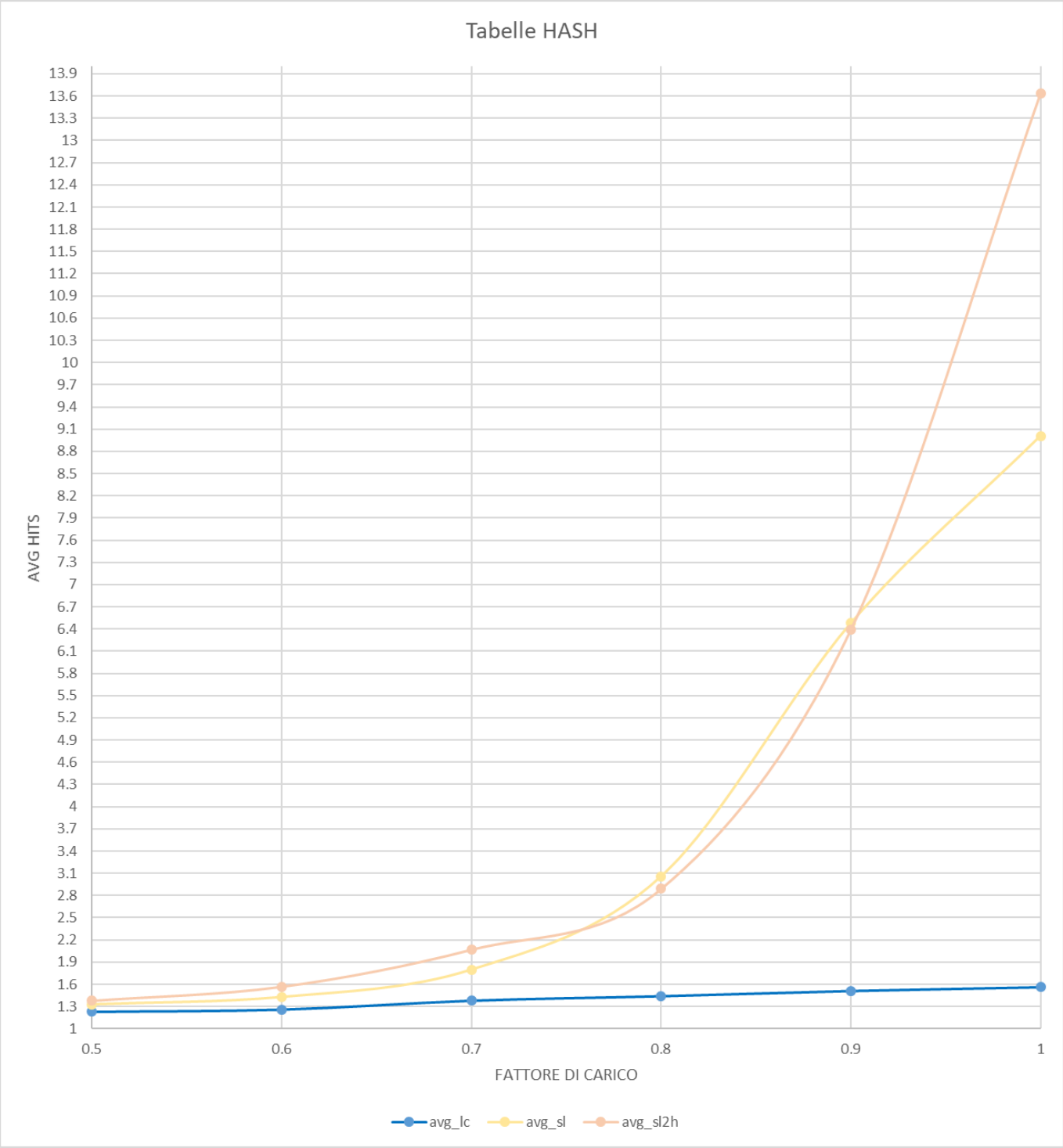
Vantaggi e svantaggi

Il vantagio principale di questo tipo di tabella si può constatare nel caso medio della ricerca dato che la funzione *hash(key)* ci da esattamente la lista in cui si trova l'elemento, la lista è piccola e quindi il numero di hits è solitamente una frazione 'piccola' (nel caso migliore 1/n) di **n**

Gli svantaggi sono:

- Il caso peggiore: la tabella può degenerare in una lista nel caso in cui tutte le chiavi del'insieme **K** collidano in un unica posizione. Questa problematica è facilmente arginabile in due modi: aumetando il fattore di carico e secgliendo un'opportuna funzione *hash(key)* in grado di distribuire nel modo migliore il dominio **K**
- Lo spazio: oltre ad allocare un array di dimensione **m** vengono allocate delle liste per gestire le collisioni e si hanno molteplici spazi vuoti all'interno del vettore *tabella*. Questa problematica è difficilmente arginabile dato che l'unico modo per rendere più efficiente in spazio questo metodo è scegliere un'opportuna funzione *hash(key)* in grado di distribuire nel modo migliore il dominio **K**

Ulteriori constatazioni



Si può vedere dal grafico che avg_lc (l'implementazione con lista di collisioni) è mediamente la più veloce. Il fattore di carico ottimale sembrerebbe essere il 50% in quanto ha la media di hits più bassa.

Aggiunte

All'interno della cartella doc sono presenti i dupm delle tabelle hash per tipo e fattore di carico, nella sezione DATI, è presente il numero di hits per ricerca (si effettuano le ricerche degli studenti in studenti.txt nell'ordine inverso rispetto a quello in cui sono stati inseriti)

Formato: `file_sorce~alpha~tipo_tabella.dat` (lc = lista collisione, sl = scansione lineare, sl2h = hashing doppio)

`doc\studenti.txt\0.5\avg_lc.dat`:

```
[0] -> NULL
[1] -> NULL
[2] -> (k: 1196628910) -> (k: 1097519727) -> NULL
[3] -> NULL
[4] -> NULL
[5] -> (k: 1159783613) -> (k: 1159783908) -> NULL
[6] -> NULL
[7] -> NULL
[8] -> NULL
[9] -> NULL
[10] -> NULL
[11] -> (k: 1172062262) -> NULL
[12] -> NULL
[13] -> (k: 1207044849) -> NULL
[14] -> NULL
[15] -> NULL
[16] -> NULL
[17] -> NULL
[18] -> NULL
[19] -> NULL
[20] -> (k: 1259782626) -> (k: 1097846954) -> (k: 1098112499) -> NULL
[21] -> NULL
[22] -> NULL
[23] -> NULL
[24] -> (k: 1097694186) -> NULL
[25] -> (k: 1159782803) -> (k: 1218047034) -> NULL
[26] -> (k: 1097732790) -> NULL
[27] -> (k: 1068559228) -> NULL
[28] -> NULL
[29] -> NULL
[30] -> (k: 1210510625) -> NULL
[31] -> NULL
[32] -> (k: 1217017485) -> NULL
[33] -> NULL
[34] -> (k: 1191339364) -> NULL
[35] -> NULL
[36] -> NULL
[37] -> (k: 1204628410) -> NULL
[38] -> NULL
[39] -> NULL
[40] -> (k: 1157584091) -> NULL
[41] -> NULL
[42] -> (k: 1097753408) -> NULL
[43] -> NULL
[44] -> NULL
[45] -> (k: 1166110380) -> NULL
[46] -> (k: 1065687581) -> NULL
[47] -> NULL
[48] -> (k: 1202808359) -> NULL
[49] -> (k: 1159782626) -> NULL
[50] -> (k: 1159783602) -> (k: 1065688298) -> NULL
[51] -> NULL
```

```
[52] -> NULL
[53] -> NULL
[54] -> NULL
[55] -> (k: 1097775539) -> NULL
[56] -> NULL
[57] -> (k: 1097599654) -> NULL
[58] -> (k: 1209317610) -> (k: 1098447444) -> NULL
[59] -> (k: 1159783570) -> NULL
[60] -> NULL
[61] -> (k: 1233687521) -> NULL
[62] -> NULL
[63] -> (k: 1203544976) -> (k: 1223212404) -> NULL
[64] -> (k: 1128546803) -> NULL
[65] -> NULL
[66] -> NULL
[67] -> (k: 1097672360) -> NULL
[68] -> NULL
[69] -> (k: 1097775985) -> NULL
[70] -> (k: 1161044277) -> NULL
[71] -> NULL
[72] -> NULL
[73] -> NULL
[74] -> NULL
[75] -> NULL
[76] -> (k: 1097601437) -> NULL
[77] -> NULL
[78] -> NULL
DATA: ------------
1
1
1
1
1
1
1
2
1
1
1
1
1
1
2
1
2
1
1
1
1
1
1
1
1
2
```

```
1
1
1
1
2
3
1
1
1
1
1
2
2
```