Dizionari

Corso di Programmazione 3 - Ingegneria dell'Informazione e dell'Organizzazione
5 dicembre , 2001

Gino Perna

Uso ed utilizzo di tabelle di Hash

PREFAZIONE

I Dizionari sono strutture di dati che permettono di operare con funzioni di tipo: Inserimento, cancellazione e ricerca. Il tipo di dizionario maggiormente utilizzato nella pratica è la tabella di Hash; il suo funzionamento si basa sun una funzione di hash che serve a tradurre la chiave (indice) in un indirizzo. Esistono poi anche strutture tipo alberi binari ed alberi red-black. Ambedue i tipi di alberi cercano di minimizzare il numero di test durante l'inserimento e le operazioni di update; il loro funzionamento è simile agli algoritmi di ricerca binaria. Infine le liste con skip possono rappresentare un semplice approccio che utilizza numeri random per costuire un dizionario.

TABELLE DI HASH

Le tabelle di Hash sono un semplice ed efficiente metodo per implementare un dizionario. Il tempo medio per la ricerca di un elemento è O(1), mentre il tempo peggiore è O(n).

Teoria

Una tabella di hash è semplicemente un array indirizzato attraverso una funzione di hash. Per esempio nella figura seguente la tabella è costituita da un array di 8 elementi, a ciascuno dei quali è associata una lista linkata di dati numerici.

La funzione di Hash dell'esempio divide semplicemente il dato numerico per 8, prendendone il resto per indicizzare la tabella. In questo modo sono sicuro di avere sempre indici compresi tra 0 e 7, e quindi che l'indice nella tabella sia sempre valido.

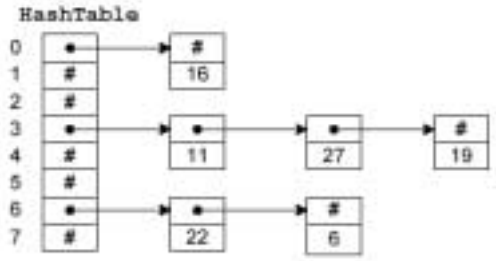

FIGURE 1. Tabella di Hash

Per inserire un nuovo elemento nella tabella, una volta applicata la funzione di hash alla chiave, si procede semplicemente con una lista linkata, inserendo l'elemento all'inizio della lista.

Esempio: per inserire il record con indice 11 dividiamo 11 per 8 ed otteniamo resto 3 (questa è la funzione di hash), perciò 11 deve andare nella lista puntata da HashTable di [3]. Per cercare un numero, si applica la funzione di hash al numero stesso, e si procede analogamente a prima, ricercando il numero nella lista linkata. La stessa cosa è possibile per cancellare un record.

Tutti i record nella tabella di Hash sono allocati dinamicamente in una lista linkata associata ad un elemento della tabella di Hash. Questo metodo è chiamato concatenamento. Una alternativa sarebbe memorizzare tutti i record direttamente nel vettore, ed è chiamato indirizzamento diretto della Hash.

Se la funzione di Hash è uniforme, ovvero suddivide in modo uniforme gli indici dalle chiavi di ricerca, allora si ha effettivamente una suddivisione efficiente nella ricerca. Il caso peggiore si ha quando tutti gli elementi sono nella stessa lista linkata, il caso migliore quando ciascun elemento è indicizzato in un nodo differente. E' perciò importante riuscire ad utilizzare una funzione di hash efficiente.

Si rimanda al testo "A compact guide to Sorting and Searching" T. Niemann per i dettagli su alcune funzionei di Hash (web site: epaperpress.com) disponibile in PDF.

OK! Ma a cosa serve?

L'utilizzo pratico nei programmi è di fonsdamentale importanza. Una Hash permette infatti di avere un buon compromesso tra efficienza (ovvero velocità di esecuzione) e memoria consumata. Sebbene ai giorni d'oggi si tenda a consumare molta RAM purtroppo in applicazioni dove il numero di dati coinvolto è notevole, oppure su sistemi embedded, oppure sui PDA è l'unico modo efficiente per ottenere un programma USABILE (ovvero che mantenga una interattività elevata).

Si prenda ad esempio il problema seguente: Aggiungere ad una lista di oggetti glio oggetti di un'altra lista che non sono già contenuti nella prima. (problema tipico nell'unire i risultati di due query diverse di un DB).

La maniera più efficiente per risolvere il problema è utilizzare una Hash Table alla quale aggiungere tutti gli elementi della prima lista e poi tutti quelli della seconda.

(Esempio alla lavagna)

Hash nei vari linguaggi

Le Hash sono talmente importanti che alcuni linguaggi le implementano NATIVAMENTE.

TABLE 1.

Linguaggio	Tipo Hash
PHP	Nativo, Multiindice
Perl	Nativo, (multiindice)
JAVA	Classe nella libreria standard, monoindice
C++	Classi siponibili in varie librerie PD, monoindice
C	Funzioni in varie librerie PD, monoindice

CLASSE HASH

vediamo ora i passi fondamentali per costruire una classe per maneggiare una Hash.

1. Abbiamo già implementato una classe per gestire liste linkate che utilizzeremo direttamente;
2. Dallo schema della fiura 1 si nota che la tabella di Hash è un insieme di liste linkate: la nostra implementazione si baserà quindi sul costruire una classe che gestisca un vettore di liste linkate
3. Dobbiamo decidere che funzione di Hash utilizzare;
4. Dobbiamo implementare i metodi base.

Prima di procedere dobbiamo aggiungere alla classe sviluppata per le liste linkate un metodo che permetta di stampare un solo elemento: questo può essere fatto prendendo direttamente il metodo print ed implementando un metodo simile con argomento il puntatore al nodo da stampare.

```
void LISTA::print(NODE *walker){
   if( walker != NULL){
      cout << "Matricola: "<<walker->matricola<<" Nome: "<<walker-
>nome<<" Voto "<<walker->voto<<"\n";
```

```
   } else {
      cout << "Not Found\n";
   }
}
```

DEfinizione della classe

Il file di include contenenete le definizioni delle DUE classi sarà:

```
struct NODE {
    int matricola;
    char nome[80];
    int voto;
    NODE * next;
};

class LISTA {
  private:
     NODE *head;
     NODE *tail;
  void insNode(char * nome, int voto, int id, NODE *, NODE *);

  public:

     LISTA();
     ~LISTA();
     void insertstart(char *, int, int );
     void insertend(char *, int ,int );
     void print();
     void printTail();
     void printHead();
     void print(NODE *);
        void clear();
     NODE * search(char *);
     int remove(char *); // rimuovi per nome
     int remove(int);    //rimuovi per matricola
     void salvafile(char *); //salva su file
     void readfile(char *);  // leggi da file
   void insertOrd(char * nome, int voto, int id);

};


class HASH {
  private:

  LISTA *dict;
  int hashvalue(char *key);
  int dictsize ;


  public:
    HASH();
```

```
   ~HASH();

    int get(char *key, NODE *);
    void insert(char *key, NODE *);
    void print(char *key);
    void printline(char *key);

};
```

Esso è costituito dalla classe LISTA vista fino ad oggi con aggiunta la classe HASH.

Implementazione

L'implementazione è quindi abbastanza agevole. Il costruttore dovrà allocare dinamicamente un vettore di oggetti LISTA (di lunghezza pari a dicttable, dipendente dal tipo di hash function utilizzata)

Il distruttore dovrà rilasciare la memoria e distruggere gli oggetti.

Funzione di hash

Si scelga la seguente funzione di Hash per l'esempio:

```
int HASH::hashvalue(char *key)
{
  int hv = 0;
  for(int i=0;i<strlen(key);i++){
    hv += key[i]%256;
  }
  return hv % 256;
}
```

Ovvero una funzione che restituisce la somma dei caratteri modulo 256.

Metodi

```
//
// DA IMPLEMENTARE
//
```

Programma test

```
#include "hlli.h"
#include <string.h>
#include <iostream.h>
#include <stdio.h>

int main(){

  HASH hh;
  NODE record;
```

```
  for(int i=0;i<5;i++){
     record.voto = i*20;
     record.matricola = 101+i;
     sprintf(record.nome,"Antonella%3.3d",i);
     hh.insert( record.nome, &record);
  }
  for(int i=0;i<7;i++){
     sprintf(record.nome,"Antonella%3.3d",i);
     hh.print( record.nome);
  }
   int i=27;
     record.voto = i*20;
     record.matricola = 101+i;
     sprintf(record.nome,"Antonella%3.3d",i);
     hh.insert( record.nome, &record);
     record.matricola ++;
     hh.insert( record.nome, &record);

     hh.print( record.nome);
     hh.printline(record.nome);

return 0;
}
```

Esercizi

Si implementino i metodi per cancellare un record e per restituire una lista di tutti i record con la stessa chiave.