Esercitazione 7
Hashing

Corso di Fondamenti di Informatica II

BIAR2 (Ing. Informatica e Automatica) e BSIR2 (Ing. dei Sistemi)

A.A. 2010/2011

10 Dicembre 2010

Sommario

Scopo della esercitazione è quello di realizzare una struttura dati per gestire una mappa basata su bucket array da utilizzare per confrontare le varie funzioni di hashing disponibili e paragonare le differenti tecniche di gestione delle collisioni.

1 Mappe

1.1 Bucket Array

Un **bucket array** per una tabella hash e' un array A di lunghezza n. Ogni cella dell'array e' considerata come un bucket (secchio) e contiene una coppia chiave-valore sottoforma di entry. Il valore n definisce la capacita' dell'array. Le celle occupate conterranno una Entry e con chiave k e valore v mentre quelle libere conterranno null.

1.2 La funzione di hashing

La **funzione di hashing** e' una funzione che trasforma ogni chiave k della mappa di origine in un numero intero compreso tra 0 e N-1, in cui N e' la capacita' della mappa. L'idea di questo approccio e' quella di memorizzare la chiave k nel bucket array in posizione *h(k)*. Naturalmente puo' capitare

che due chiavi differenti possano avere lo stesso valore di hashing ed in tal caso si genera una **collisione** (descritta di seguito). La funzione di hashing opera in due fasi.

- Nella prima fase (**generazione dell'hashcode**) prende una qualunque chiave k e la trasforma in un numero intero, non necessariamente compreso nell'intervallo $[0, N-1]$.
- Nella seconda fase invece (**compressione**) l'hashcode ottenuto dalla fase 1 viene normalizzato in modo da rientrare nel range $[0, N-1]$

1.3 Gestione delle collisioni

Quando per due chiavi distinte *k1*, *k2* la funzione di hashing genera uno stesso valore numerico si ha una collisione. In questo caso solo una delle due chiavi potra' essere memorizzata nella posizione dell'array identificata dal suo hashcode mentre l'altra dovra' essere memorizzata in una posizione differente, in modo pero' di poterla ritrovare. Per ottenere questo risultato vengono utilizzate varie tecniche come il *linear probing*, il *quadratic probing* e il *double hashing*.

2 Classi di supporto

Le classi di supporto fornite per l'esercitazione sono le seguenti:

- `BucketEntry<V>`: rappresenta una entry chiave-valore utilizzata per memorizzare le informazioni in un bucket array. In particolare la chiave e' di tipo String mentre il valore e' di tipo generico `<V>`. La classe e' una classe concreta puo' essere utilizzata direttamente senza bisogno di essere estesa.
- `HashingFunction` si tratta di una classe che definisce la funzione di hashing (non normalizzato). Contiene il seguente metodo da implementare:
  - `int hashing(String key)` che data una chiave di tipo stringa restituisce l'hashcode numerico (non normalizzato)

- `CompressionFunction` si tratta di una classe che definisce la funzione di compressione per la normalizzazione del codice hash. Contiene il seguente metodo da implementare:
  - `int compression(int completeHashCode, int n)` che normalizza l'hashcode ottenuto dalla funzione `hashing(...)` in modo da rientrare nel range $0, N-1$
- `BucketArray<V>` che rappresenta il bucket array. Nel costruttore viene specificata la capacita' ed inoltre la funzione di hashing (attraverso il parametro `HashingFunction hashing` e `CompressionFunction compression`). Contiene inoltre alcuni metodi astratti che devono essere implementati e che permettono di definire la politica di gestione delle collisioni:
  - `V get(String key)` che a partire da una chiave restituisce il valore contenuto nella mappa o null se non presente.
  - `void put(String key, V value)` che inserisce un valore nella mappa, eventualmente eliminando la precedente entry con la stessa chiave se presente.
  - `V remove(String key)` rimove l'elemento identificato dalla chiave *key* e restituisce il valore.
  - `int size()` che restituisce il numero di elementi utilizzati all'interno del bucket array.
- `BucketFrame` che permette di visualizzare in forma grafica il contenuto di un bucket Array evidenziando i bucket vuoti, normali e quelli con collisione.

Esempio di utilizzo

```
BucketArray<Integer> a = ... //array
BucketFrame.show(a,"Bucket Array di prova");
```

3 Implementazione

3.1 Linear probing e simple hashing

Collisioni (Linear Probing) Si vuole implementare una classe Java di nome `LinearProbingBucketArray<V>` di tipo generico che implementa la

classe astratta `BucketArray<V>` e che utilizza il linear probing per la gestione delle collisioni. Implementare opportunamente tutti i metodi astratti.

Hashing (somma dei componenti) Si vuole implementare la classe Java di nome `SommaHashingFunction` che implementa la funzione di hashing *somma delle componenti*. Nel caso della stringa considerare ogni carattere (convertito in intero secondo il codice ASCII) come una componente da sommare.

Compression (metodo di divisione) Si vuole implementare la classe Java `DivCompressionFunction` che implementa la funzione di compressione hashing basata sul metodo di divisione.

Test Realizzare una classe MyTest con un metodo `main()` da utilizzare per il test delle classi prodotte. In particolare creare un array bucket di 30 elementi, inserire valori e stringhe casuali e poi visualizzare mediante `BucketFrame` la disposizione delle entry nella mappa.

3.2 Calcolo delle collisioni e sample random

Realizzare una classe `StatUtil` contenente il metodo statico `static <V> int contaCollisioni(BucketArray<V> a)` che conta il numero di collisioni all'interno dell'array, utilizzando i metodi `isEmpty(...)` e `isCollision(...)` gia' presenti nella classe. Realizzare poi un metodo statico `static List<BucketEntry<Integer>> generaRandom (int numValues)` che genera `numValues` entry casuali da utilizzare per effettuare test di riempimento sulla mappa.

Effettuare il test del metodo sull'array creato in precedenza per controllare la correttezza della implementazione.

3.3 Varianti

Gestione delle collisioni Implementare le varianti del bucket array in cui vengono implementate le varie tecniche di gestione delle collisioni ed in particolare:

- La classe `QuadraticProbingBucketArray<V>` che estende `BucketArray<V>` e che implementa la gestione delle collisioni mediante *quadratic probing*.

- La classe `DoubleHashingProbingBucketArray<V>` che estende `BucketArray<V>` e che implementa la gestione delle collisioni mediante *hashing doppio*. Utilizzera' in particolare una seconda funzione di hashing che puo' essere definita attraverso un oggetto di tipo `HashingFunction`

Funzioni di hashing Implementare le varianti di funzione di hashing ed in particolare:

- La classe `PHashingFunction` che estende `HashingFunction` e che implementa la funzione di hash polinomliale.
- La classe `DoubleHashingProbingBucketArray<V>` che estende `BucketArray<V>` e che implementa la gestione delle collisioni mediante *hashing doppio*. La funzione di hashing puo' essere implementata direttamente all'interno della classe.

Funzioni di compressione Implementare la seguente variante di funzione di compressione ed in particolare:

- La classe `MadCompressioniFunction` che estende `CompressionFunction` e che implementa la funzione di compressione MAD.

Effettuare il test di tutte le implementazioni, mano a mano che vengono completate.

4 Benchmark

Combinare tutte le implementazioni realizzate relative a tecniche di gestione delle collisioni, funzioni di hashing e compressioni in modo da effettuare un esperimento comparativo delle varie tecniche.

In particolare per ogni combinazione effettuare un test su un array bucket di 1000 elementi riempito al 25, 50, 75, 90 percento della capacita' (mediante la funzione di generazione random) e misurando le collisioni.

Stampare quindi una tabella riassuntiva in cui per ogni combinazione viene riportato il numero di collisioni.