# Tabelle Hash

Concetti generali

Implementazione in Java

Risoluzione delle collisioni

# Tabelle Hash

- Inserimento e Ricerca in tempo O(1) in un certo insieme A
- Per raggiungere questa efficienza in tempo si usa di più lo spazio
- Sia U = {el_1, el_2, …, e_N} l'insieme universo di tutti i possibili elementi che si possono (in teoria) inserire nell'insieme A
- Ogni elemento el_i di U ha una chiave **unica** key(el_i) che è un intero in un certo intervallo [p, q], sottoinsieme degli Interi, dove q − p = N

# Tabelle hash

- Uso un array di dimensione m per mettere gli elementi
- Se m = N ho completa efficienza O(1), ma ho bisogno di uno spazio esageratamente grande se l'insieme A che effettivamente viene inserito è solo una piccola parte di U
- Nella maggioranza dei casi vale che m << N e allora devo definire una **funzione di hash**

$$h: [p, q] \rightarrow [0, m-1]$$

# Tabelle hash

- L’elemento el_i dovrà essere inserito in posizione h(key(el_i)) appartienente a [0, m-1]
- h(key(el_i)) è l’indice in cui inserisco l’elemento el_i nell’array che supporta la mia tabella hash
- Una buona funzione di hash è la seguente:

h(r) = r mod m  dove m è un numero primo vicino a una potenza di 2

# Implementazione in Java

- La chiave di ogni oggetto di una classe generica E è data dal suo metodo hashcode()
- Ricordiamo che hashcode() va ridefinito insieme ad equals()
- Sia equals() che hashcode() sono basati su alcuni campi (generalmente immutabili) dell'oggetto, cioè su una **chiave** che identifica univocamente l'oggetto
- Quindi per noi l'intervallo [p, q] sarà

  [Integer.MIN_VALUE, Integer.MAX_VALUE]

# Collisioni

- Poiché m << N inevitabilmente ad oggetti diversi verrà assegnata la stessa posizione nell'array dalla funzione di hash
- Per risolvere questo problema ci sono due approcci:

1) Uso di Liste Concatenate

2) Indirizzamento Aperto

# Collisioni con Liste Concatenate

- Le celle dell'array non contengono elementi, ma liste concatenate (inizialmente vuote)
- Gli elementi che vengono posizionati nello stesso posto vengono messi nella lista concatenata associata a quella posizione
- Caso pessimo: tutti gli elementi sono posizionati nello stesso posto, quindi ho una lista concatenata di lunghezza n, con tempo O(n) per ricerca e inserimento
- Nel caso medio le tabelle hash si comportano "bene" cioè le operazioni hanno complessità circa O(1)

# Collisioni con indirizzamento aperto

- Gli elementi si trovano nelle posizioni dell'array
- La funzione di hash iniziale h(k) viene sostituita con la funzione

  $h'$: [p, q] x [0,m-1] → [0,m-1]
- La sequenza <h'(k,0), h'(k,1), …, h'(k,m-1)> deve essere una permutazione di <0, 1, …, m-1>
- Per inserire un elemento el si procede come segue

# Collisioni con indirizzamento aperto

- i = 0
- Se la cella in posizione p = h'(key(el),i) è vuota si  inserisce l'elemento in posizione p
- Altrimenti si prova a posizionare l'elemento in p = h'(key(el), i + 1)
- Si continua fino a quando non si trova una posizione libera oppure i = m, in questo caso si lancia una eccezione di overflow della tabella hash

# Scansione Lineare

- E’ una possible tecnica per la funzione h’
- Si definisce

$$h'(k,i) = (h(k) + i) \bmod m$$

- Dove h(k) è una funzione hash data non con indirizzamento aperto

# Tabelle Hash

- T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduzione agli Algoritmi e Strutture Dati, McGraw-Hill, 2005

- Pagg. 207 - 228