Tabelle hash

Look-up table, caso generale

Anche *memoria associativa*, *associative array*, *map*, *symbol table*, *dictionary*: collezione di elementi nella forma coppia attributo / valore (key, value), quindi dati generici, caratterizzati da un campo chiave, con associate le operazioni:

- Inserimento di una coppia nella collezione
- Cancallazione di una coppia dalla collezione
- Ricerca di una chiave
- Modifica di un valore corrispondente ad una chiave

Memorizzazione dati strutturati

Ogni elemento ha una chiave K_i tale che $1 \le K_i \le n$

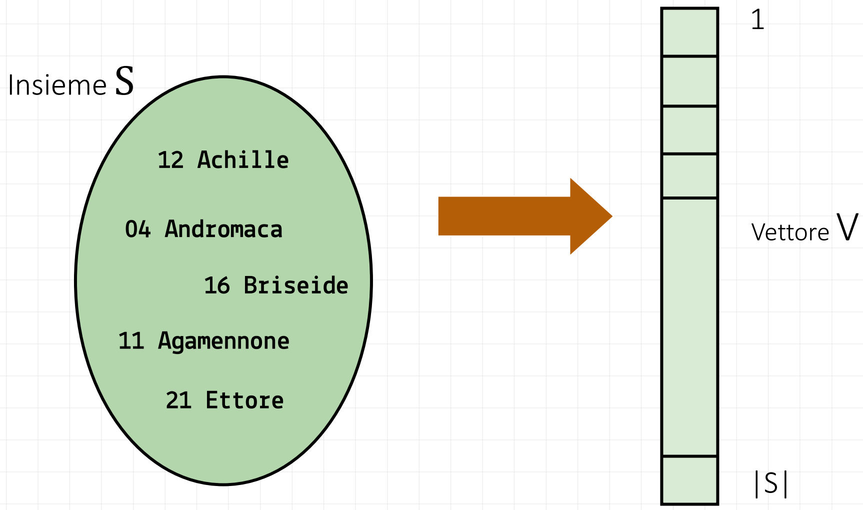


Tabella a indirizzamento diretto

Posizione nel vettore = valore della chiave

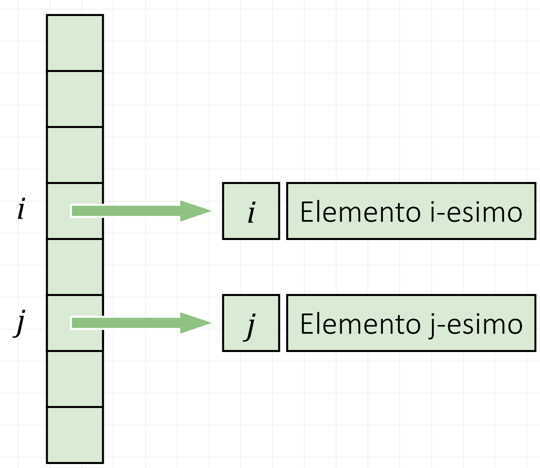


Tabella a indirizzamento diretto

```
Search(V,k)
   return V[k]

Insert(V,x)
   V[key[x]] = x

Delete(V,x)
   V[key[x]] = nil
```

Costo computazionale = $\Theta(1)$

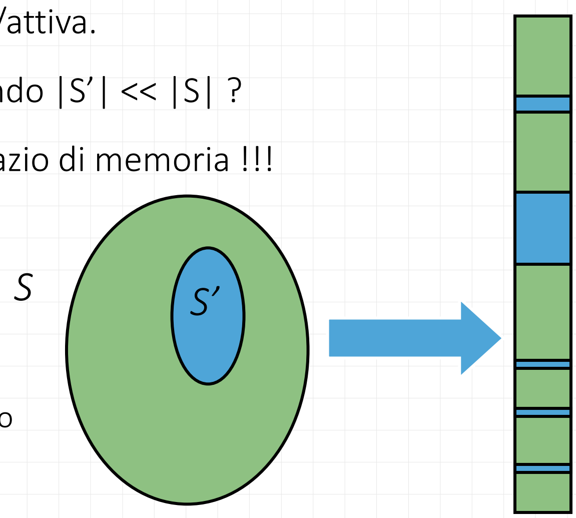
Problemi...

Supponiamo che solo una parte *S'* dello spazio *S* delle chiavi sia utilizzata/attiva.

Cosa succede quando |S'| << |S| ?

Si spreca molto spazio di memoria !!!

Soluzioni?



= Spazio sprecato

Una soluzione …

Possiamo ridurre l'occupazione di spazio da $\Theta(|S|)$ a $\Theta(|S'|)$ usando liste.

PROBLEMA (non finiscono mai):
Inserimento, Cancellazione e Ricerca costano $\Theta(|S'|)$ invece di $\Theta(1)$.

Cala la memoria e cresce il tempo …

Compromesso TEMPO / SPAZIO

Si possono ottenere tutti e due i risultati assieme!

Usando tabelle Hash possiamo raggiungere:

- Tempo di accesso: $\Theta(1)$
- Spazio di memoria: $\Theta(|S'|)$

Ma ... in media e non nel caso pessimo.

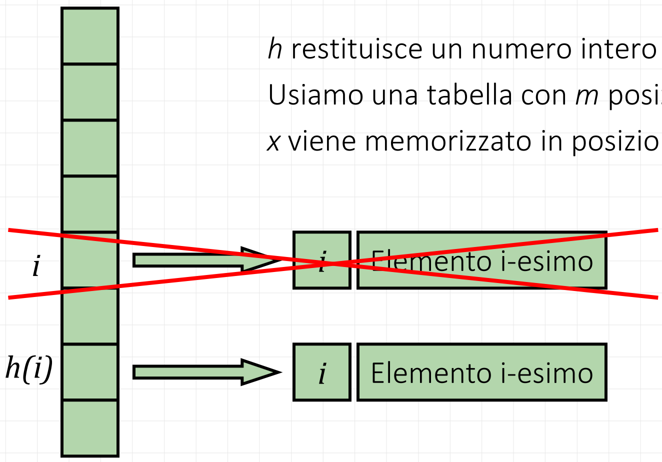
h, funzione di hash

h = funzione hash

h restituisce un numero intero da *1* a *m*.

Usiamo una tabella con *m* posizioni

x viene memorizzato in posizione *h(key[x])*



Proprietá per una "buona" h

Deterministica ma deve sembrare "random" in modo da minimizzare le collisioni.

x e y generano una collisione se

$$x \neq y \quad \text{e} \quad h(x) = h(y)$$

h deve minimizzare il numero di collisioni

Frequenza delle collisioni

Tabella di dimensione m e p elementi da inserire:

Funzione hash, m^p diverse possibilità di allocazione dei p elementi

(*Esempio*: con p = 8 e m = 10 ⟶ 10^8 possibili allocazioni diverse)

Ci sono $\frac{m!}{(m-p)!}$ possibilità di un hashing senza collisioni

(*Esempio*: se p = 8 e m = 10 ci sono 3 · 4 · . . . · 10 possibilità di hashing senza collisioni)

Paradosso del compleanno

Con 23 persone, la probabilità che tutti abbiano il compleanno in un giorno diverso è < 1/2

Cioè: con p = 23 e m = 365, la probabilità di collisione è ≥ ½

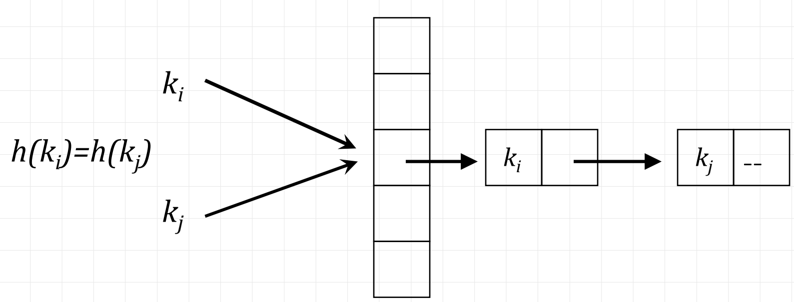
(prob. cond. ogni compl. diverso = $\left(\frac{1}{365}\right)^{23} \cdot (365 \cdot 364 \cdot \ldots \cdot 343)$ ≈ 0.49)

La prob. Di collisione cresce in modo sorprendente!

Chaining

Risoluzione di collisioni con “chaining”

- Si crea una lista per ogni cella della tabella (valore assumibile dalla h)
- Si collegano i record afferenti a una stessa cella nella sua lista
- Le celle della tabella sono le teste delle liste (*nil* se liste vuote)



Chaining

```
Chained-hash-insert(T,x)
      // inserisci x in testa alla lista
      T[h(key[x])]

Chained-hash-search(T,k)
      // ricerca l'elemento con chiave k nella lista
      T[h(k)]

Chained-hash-delete(T,x)
      // cancella x dalla lista
      T[h(key[x])]
```

Complessità di T()?

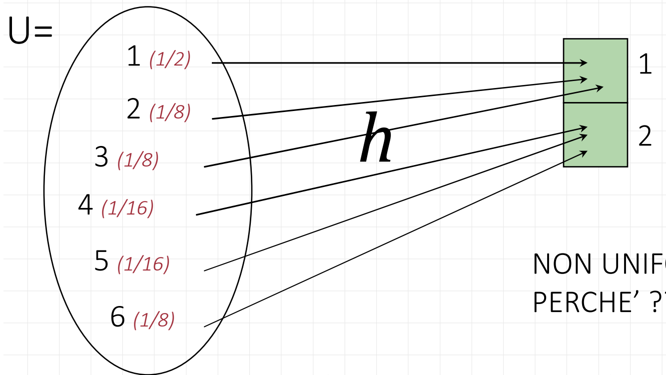
Chaining, analisi

Load factor $\alpha = n/m$,
n = numero di elementi memorizzati in tabella
m = dimensione della tabella

Caso pessimo: tutte le n chiavi finiscono nella stessa posizione. Ricerca = $\Theta(n)$

Caso medio:
Simple uniform hashing: $Pr(\ h(k)=i\ ) = Pr(\ h(k)=j\ )$

Simple uniform hashing, esempio

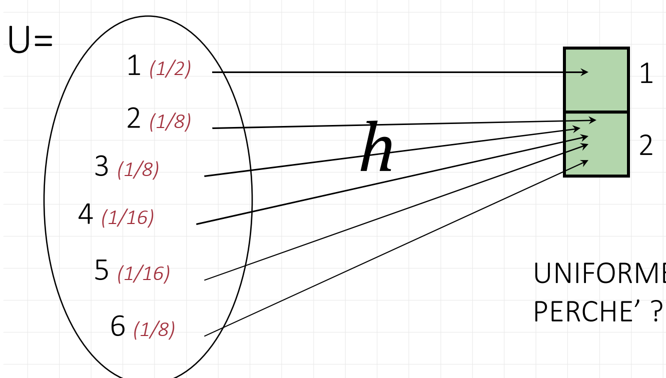


m = 2

NON UNIFORME !!!
PERCHE' ???

in rosso è indicata la probabilità che una certa chiave debba essere inserita nella tabella

Simple uniform hashing, esempio



m = 2

UNIFORME !!!
PERCHE' ???

in rosso è indicata la probabilità che una certa chiave debba essere inserita nella tabella

Simple uniform hashing

Una funzione hash si dice semplice uniforme quando rende uniforme il riempimento della tabella.

Non quando la distribuzione delle chiavi è uniforme.

Complessitá SUH

Teorema:

Ipotesi:

- collisioni gestite con chaining
- simple uniform hashing
- caso medio

Tesi:

una ricerca ha costo computazionale $\Theta(1+\alpha)$

Dimostrazione: prima caso ricerca senza successo, poi con successo

Complessitá SUH

Dimostrazione: caso di ricerca senza successo.

Il load factor α è la lunghezza media di una catena.

In una ricerca senza successo il numero di elementi esaminati è uguale alla lunghezza media delle catene, cioè $\frac{n}{m} = \alpha$.

Calcolare $h()$ costa $\Theta(1)$.

La ricerca costerà $\Theta(1) + \Theta(\alpha) = \Theta(1 + \alpha)$

Complessitá SUH

Dimostrazione: caso di ricerca con successo.

Assumiamo di inserire elementi *in coda* alla catena.

Simple uniform hashing $\rightarrow$ numero medio di elementi in una catena dopo i inserimenti = i/m

L'elemento j verrà inserito mediamente nella posizione $1+(j-1)/m$ all'interno di una catena.

Un elemento generico finirà in media nella posizione data dalla formula:

$$1/n \sum(1+ (i-1)/m) = 1/n\ (n + [n(n+1)]/[2m] - n/m) =$$
$$= 1 + \alpha/2 - 1/(2m) =$$
$$= \Theta(1+\alpha)$$

Complessitá SUH

Supponiamo che $n=O(m)$. Ovvero che il numero di elementi inseriti nella tabella sia proporzionale alla dimensione della tabella. Avremo:

$$\alpha = n/m = O(m)/m = O(1)$$

In questo caso la ricerca impiega tempo costante !!!

Cosa succede se gli elementi vengono inseriti all'inizio delle liste?

Riepiloghiamo…

Se usiamo liste doppiamente linkate per le catene e se inseriamo i nuovi elementi in testa alle liste abbiamo

Ricerca
Cancellazione
Inserimento

$O(1)$ operazioni in media

Funzioni hash: progettazione

$Pr(k)$ = probabilità della chiave k

$S_j = \{ k \in U \text{ tali che } h(k) = j \}$

Vogliamo uniform hashing ovvero

$\sum_{k \in S_j} \Pr(k) = 1/m$ (m = dimensione della tabella)

Funzioni hash: progettazione

Se Pr(•) è sconosciuta $\Rightarrow$ Usiamo euristiche

Supponiamo per semplicità che le chiavi siano numeri naturali.

IDEA:

- h deve dipendere da tutti i bit di k
- deve essere indipendente da eventuali pattern che possono essere presenti nelle chiavi

Metodo della divisione

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

Esempio:

m=12, k=100,

$h(100) = 100 \bmod 12 = 4$

Per controllare se uno ha scelto un buon m è consigliabile usare un “benchmark” reale.

Metodo della divisione

Metodo della divisione, possibili cattive scelte di m

Se m e tutte le chiavi sono pari → si usano solo la metà degli slot

Es., m = 100 e i dati sono 50, 100, 150, 200, 250

50 → 50
100 → 100
150 → 50
200 → 100
250 → 50

Es., $m = 2^4 = 16$ e i dati sono 0, 32, 64, 96

0000000 = 0 → 16
0010000 = 32 → 16
0100000 = 64 → 16
0110000 = 96 → 16

Euristica: usare m primo e non troppo vicino a una potenza di 2 o di 10 (le basi più comuni)

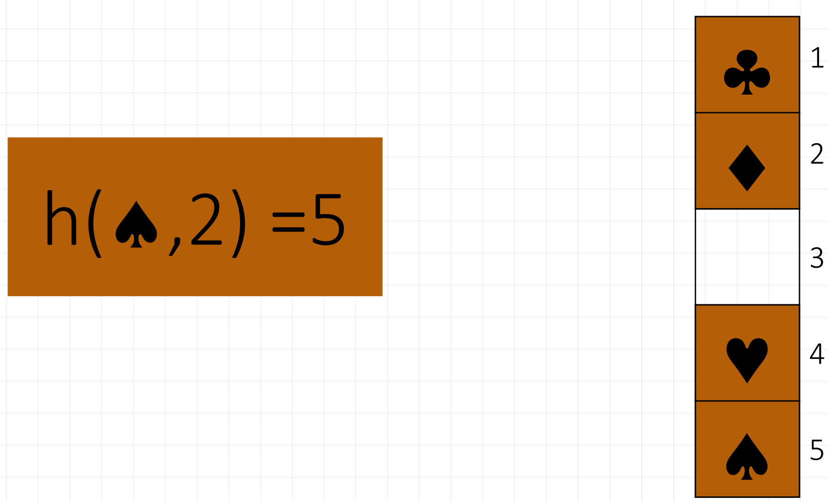
Metodo della moltiplicazione

$$h(k) = \lfloor m(kA \bmod 1) \rfloor$$

- Scegli una costante A con 0 < A < 1
- Calcola k · A
- Prendi la parte frazionaria: $k{\cdot}A - \lfloor k{\cdot}A \rfloor$
- Moltiplica il risultato per m
- Prendi la parte intera
- La chiave k è mappata in $\lfloor (m \cdot (k{\cdot}A - \lfloor k{\cdot}A \rfloor)) \rfloor$

Esempio:
$A = (5^{1/2}-1)/2 = 0.618...$,
k = 123456,
m = 10000
h(123456) = conti conti conti = 41

Risoluzione collisioni: open addressing



Open addressing

- Nessun puntatore: spazio risparmiato!
- $\alpha \leq 1$ sempre. Nessuna lista per gestire le collisioni
- Hash function più complessa: $<h(k,0), h(k,1), \ldots, h(k,m\text{-}1)>$ deve essere una permutazione di $<1, \ldots, m>$

$h(k,i)$ = posizione della tabella in cui inserire la chiave k quando tutte le posizioni $h(k,0), \ldots, h(k,i\text{-}1)$ sono già occupate.

Open addressing, uniform hashing

Se gestiamo le collisioni con il metodo open addressing, la funzione hash restituisce una permutazione degli indici <1, ... ,m>.

Invece di simple uniform hashing parliamo di uniform hashing.

Uniform hashing: tutte le permutazioni devono apparire con la stessa probabilità

Open addressing, inserimento

```
Hash-insert(T,k)
    i=0
    repeat j=h(k,i)
            if T[j]=nil
            then T[j]=k
                 return j
           else i=i+1
    until i=m
    error "hash table overflow"
```

Vittorio Maniezzo - Universita di Bologna 31

31

Open addressing, ricerca

```
Hash-search(T,k)
    i=0
    repeat j=h(k,i)
            if T[j]=k
            then return j
           else i=i+1
    until (T[j]==nil) or (i==m)
    return nil
```

Vittorio Maniezzo - Universita di Bologna 32

32

Open addressing, cancellazione

```
Hash-delete(T,k)
      i=Hash-search(T,k)
      if i≠nil then T[i]=nil
```

NON FUNZIONA

Vittorio Maniezzo - Universita di Bologna

Open addressing, cancellazione

1	3
2	7
3	8
4	

Inseriamo 6 ⇒

h(6,0)=1
h(6,1)=2
h(6,2)=3
h(6,3)=4

1	3
2	7
3	8
4	6

Cancelliamo 8

1	3
2	7
3	
4	6

Ricerchiamo 6 ⇒

Risposta:
6 non c'è

Vittorio Maniezzo - Universita di Bologna

Open addressing, cancellazione

Esercizio:
Modificare Hash-search e Hash-delete
per risolvere il problema illustrato nel lucido precedente.

Si può usare un carattere con il quale contrassegnare gli elementi cancellati.

1	3
2	7
3	D
4	6

Open addressing, linear probing

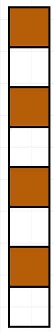
Sia h' una funzione hash "ordinaria".

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

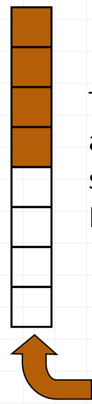
Esempio di linear probing: m=5, k=3, $h'(3)$=4

$$\left.\begin{array}{l} h(k,0) = 4 \\ h(k,1) = 0 \\ h(k,2) = 1 \\ h(k,3) = 2 \\ h(k,4) = 3 \end{array}\right\} = \text{probe}$$

Linear probing, primary clustering



Tempo medio di accesso per una ricerca senza successo: 1.5
Perche'?

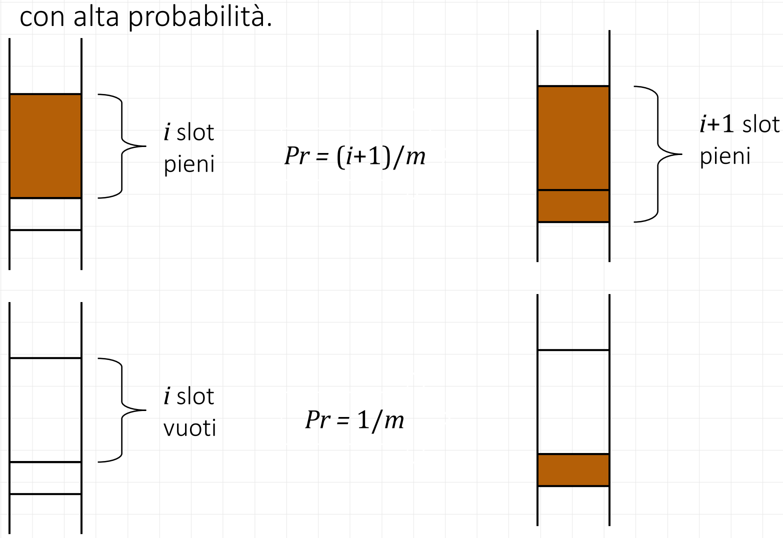


Tempo medio di accesso per una ricerca senza successo: 2.5
Perche'?

Clustering primario

Linear probing, primary clustering

Usando linear probing il clustering primario si forma con alta probabilità.



Quadratic probing

$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$

con $c_2 \neq 0$

Cosa si può dire sul clustering primario ?

Double hashing

$h(k,i) = (h_1(k) + i\, h_2(k)) \bmod m$

Cosa succede se $MCD(m,h_2(k)) = d > 1$???

Quante permutazioni distinte produce il double hashing ???

Double hashing, esempio

$h_1(k) = k \bmod m$

$h_2(k) = 1 + (k \bmod (m - 2))$

$m = 13$, h_1(k) = k mod 13, h_2(k) = 1 + (k mod 11)

k	$h_1(k)$	$h_2(k)$	$h(k,i)$
18	5	8	5
41	2	9	2
22	9	1	9
44	5	1	5,6
59	7	5	7
32	6	11	6,4
31	5	10	5,2,7,11
73	8	8	8

Vittorio Maniezzo - Universita di Bologna 41

41

Open addressing, ricerca

Teorema:

Data una hash table con open addressing e load factor $\alpha = n/m < 1$, la lunghezza media di una "probe" in una ricerca senza successo è $1/(1-\alpha)$.

(Ipotesi: uniform hashing)

$\alpha = (m-1)/m$ ⇒ $1/(1-\alpha) = m$
(valore massimo di α)

$\alpha = 1/m$ ⇒ $1/(1-\alpha) = m/(m-1)$
(valore minimo di α)

$\alpha = 1/2$ ⇒ $1/(1-\alpha) = 2$

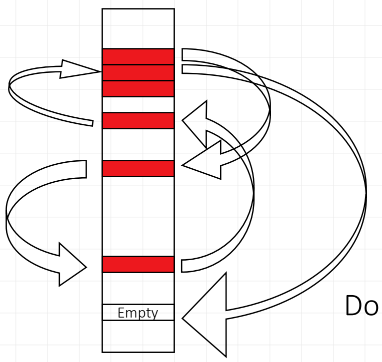
Vittorio Maniezzo - Universita di Bologna 42

42

Open addressing, ricerca

Dimostrazione:

IDEA: cosa succede quando facciamo una
ricerca senza successo ???



X = lunghezza probe =
quante volte devo calcolare
$h(k,i)$ prima di trovare uno slot
vuoto = elemento non trovato

Dobbiamo valutare $E[X]$ = *media di X*

Open addressing, ricerca

Lemma:

X variabile aleatoria discreta
(prob. di trovare i pos. occupate prima di una libera)

$$X = \begin{cases} 0 \dashrightarrow p_0 \\ 1 \dashrightarrow p_1 \\ \dots\dots\dots \\ i \dashrightarrow p_i \\ \dots\dots\dots \end{cases}$$

$$E[X] = \sum_{i=0}^{\infty} i p_i \qquad (p_i = 0 \ \forall i > n)$$

$$= \sum_{i=0}^{\infty} i\, p(X=i)$$

(nota: per i=0 gli addendi si annullano)

$$= \sum_{i=0}^{\infty} i \left(p(X \geq i) - p(X \geq i+1) \right) \qquad = \qquad \sum_{i=1}^{\infty} p(X \geq i)$$

Open addressing, ricerca

$$E[X] = \sum_{i=1}^{\infty} p(X \geq i)$$
$$\leq \sum_{i=1}^{\infty} (n/m)^i$$
$$\leq \sum_{i=1}^{\infty} \alpha^i$$

Costo per la ricerca:

costo pos. libera +
costo pos. occupate =

$1 + E[X] \leq$

$1 + \sum_{i=1}^{\infty} \alpha^i =$

$1 + \alpha + \alpha^2 + \alpha^3 + \ldots\ldots\ldots \quad (\alpha < 1)$

$1 / (1- \alpha)$

□

Assunte indipendenti

$q_i = p(X \geq i)$
$q_1 = n/m$
$q_2\,(1\ \text{occupata}) = n/m \cdot n\text{-}1/m\text{-}1 \;<\; (n/m)^2$
$q_i\,(i\text{-}1\ \text{occupate}) \;= n/m \cdot n\text{-}1/m\text{-}1 \cdot \ldots \cdot n\text{-}(i\text{-}1)/m\text{-}(i\text{-}1) \;<\; (n/m)^i$

Open addressing, inserimento

Teorema:

Data una hash table con open addressing e load factor $\alpha = n/m < 1$, la lunghezza media di una “probe” è $1/(1- \alpha)$.

(Ipotesi: uniform hashing)

Dimostrazione:

Nota: α deve essere < 1.

Per inserire un elemento abbiamo bisogno di determinare la posizione nella tabella dove inserirlo.

Ricerca: costo $1/(1-\alpha)$.

Per inserire nella tabella nella posizione appena determinata: $\Theta(1)$.

Se la tabella è piena al 50% , ci aspettiamo 2 probe
Se la tabella è piena al 90% , ci aspettiamo 10 probe