Corso di Laurea in INFORMATICA

Algoritmi e Strutture Dati

a.a. 2012-13

MODULO 11

Tabelle Hash e Dizionari

Generalità. Dizionari: Specifiche, rappresentazione e confronto tra realizzazioni alternative.

ESISTONO DELLE APPLICAZIONI CHE, PUR RICHIEDENDO UNA STRUTTURA DATI DEL TIPO INSIEME, NON RICHIEDONO TUTTE LE OPERAZIONI DEFINITE SULL'INSIEME.

AD ESEMPIO, NEL TRATTAMENTO DEI TESTI SONO PRESENTI SPESSO DEI DIZIONARI DELLA LINGUA, MESSI A DISPOSIZIONE PER CONTROLLARE LA CORRETTEZZA ORTOGRAFICA DELLO SCRITTO.

E' CERTAMENTE UN PROBLEMA ADATTO AL TIPO ASTRATTO DI DATI *INSIEME*: NON SEMBRANO NECESSARIE OPERAZIONI COME UNIONE, INTERSEZIONE E DIFFERENZA E SONO, INVECE, NECESSARIE OPERAZIONI ESPLICITE DI RITROVAMENTO ED ESTRAZIONE DI UN ELEMENTO DALL'INSIEME.

IN QUESTE STRUTTURE GLI ELEMENTI SONO GENERALMENTE TIPI STRUTTURATI AI QUALI SI ACCEDE PER MEZZO DI UN RIFERIMENTO A UN CAMPO CHIAVE TALORA LEGATO AL CONTENUTO STESSO DELL'ELEMENTO.

GLI ELEMENTI ASSUMONO LA FORMA DI UNA COPPIA COSTITUITA DA CHIAVE - ATTRIBUTO.

LA CARATTERISTICA DELLA CHIAVE E' LEGATA ALLA APPLICAZIONE: NEI DIZIONARI DEGLI ELABORATORI DI TESTI LA CHIAVE INDIVIDUA UNA PAROLA MENTRE IN UN DIZIONARIO DI MAGAZZINO LA CHIAVE PUO' ESSERE UN CODICE PEZZO.

L'ATTRIBUTO ASSOCIATO, INVECE, RAPPRESENTA L'INFORMAZIONE ASSOCIATA PER SCOPI DI GESTIONE O MANUTENZIONE.

ESEMPIO:

NELLE TABELLE DI SIMBOLI, UTILI AI COMPILATORI E AI LINKER, LA CHIAVE VIENE USATA PER L'IDENTIFICATORE MENTRE NELL'ATTRIBUTO VENGONO MEMORIZZATE LE INFORMAZIONI DI GESTIONE (INDIRIZZO DELLA LOCAZIONE, DIMENSIONE, POSIZIONI NELLE QUALI L'IDENTIFICATORE E' USATO NEL PROGRAMMA ETC.)

Dizionario = insieme dinamico di oggetti che consente di effettuare le operazioni di ricerca, inserimento, cancellazione.

DEFINIRE UN DATO ASTRATTO ADATTO NON RISULTA BANALE VISTO CHE NELL'INSIEME NON SI E' MAI FATTO ALCUN CENNO AL FATTO CHE SI POTESSE ACCEDERE AI SINGOLI ELEMENTI DELL'INSIEME SE NON PER L'OPERAZIONE DI VERIFICA DI APPARTENENZA E DI INSERIMENTO.

LE OPERAZIONI APPLICATE AD UN DIZIONARIO DEVONO CONSENTIRE LA VERIFICA DELL'ESISTENZA DI UNA DEFINITA CHIAVE E DEVE ESSERE POSSIBILE L'INSERIMENTO DI NUOVE COPPIE CHIAVE-ATTRIBUTO COME PURE LA CANCELLAZIONE.

PUO' ESSERE UTILE ANCHE IL RECUPERO DELLE INFORMAZIONI PRESENTI NELL'ATTRIBUTO OPPURE LA LORO EVENTUALE MODIFICA.

LA SPECIFICA PER I DIZIONARI E' IDENTICA A QUELLA DEL TIPO DI DATO INSIEME. LE OPERAZIONI AMMESSE SONO:

CREA APPARTIENE INSERISCI CANCELLA

SONO NECESSARIE ANCHE OPERAZIONI COME

RECUPERA AGGIORNA

POSSIBILI SIGNIFICATI

RECUPERA(D,k) – dato un insieme D ed un valore chiave k restituisce un elemento in D tale che key[x] = k

INSERISCI (D,x) – inserisce in D l'elemento x

CANCELLA(D,k) - rimuove da D l'elemento con chiave k

AGGIORNA(D,k,x) – aggiorna in D con x il valore dell'elemento avente chiave k

COME PER LE REALIZZAZIONI VISTE PER L'INSIEME, ESISTONO REALIZZAZIONI CHE SI RIFANNO ALLA RAPPRESENTAZIONE CON TAVOLE AD ACCESSO DIRETTO E CON VETTORE BOOLEANO (VETTORE CARATTERISTICO).

ALTRE REALIZZAZIONI FANNO RIFERIMENTO ALLA RAPPRESENTAZIONE MEDIANTE UNA LISTA (I CUI ELEMENTI SONO QUELLI DELL'INSIEME).

CI SONO REALIZZAZIONI PIU' EFFICIENTI MEDIANTE VETTORI ORDINATI E TABELLE HASH.

Tavole ad accesso diretto

Sono dizionari basati sulla proprietà di accesso diretto alle celle di un array

Idea:

- dizionario memorizzato in un array *v* di m celle
- a ciascun elemento è associata una chiave intera nell'intervallo [0,m-1]
- elemento con chiave k contenuto in *v*[k]
- al più n≤m elementi nel dizionario

Complessità degli operatori

INSERISCI (D,x) - *inserisce in D l'elemento x*
O(1)

CANCELLA(D,k) - *rimuove da D l'elemento con chiave k*
O(1)

RECUPERA(D,k) - *dato un insieme D ed un valore chiave k restituisce un elemento in D tale che key[x] = k*
O(1)

Fattore di carico

Si può misurare il grado di riempimento di una tavola ad accesso diretto usando il fattore di carico

$$\alpha = \frac{n}{m}$$

Numero di elementi memorizzati

Dimensione della tabella

Esempio: tavola con i nomi di 100 studenti indicizzati da numeri di matricola a 6 cifre

n=100 $m=10^6$ $\alpha = 0{,}0001 = 0{,}01\%$

Grande spreco di memoria!

Pregi e difetti

Pregi:

– Tutte le operazioni richiedono tempo O(1)

Difetti:

– Le chiavi devono essere necessariamente interi in [0, m-1]
– Lo spazio utilizzato è proporzionale ad m, non al numero n di elementi: può esserci grande spreco di memoria!

RAPPRESENTAZIONE CON VETTORE ORDINATO

SI UTILIZZA UN VETTORE CON UN CURSORE ALL'ULTIMA POSIZIONE OCCUPATA.

AVENDO DEFINITO UNA RELAZIONE DI ORDINAMENTO TOTALE "<" SULLE CHIAVI, QUESTE SI MEMORIZZANO NEL VETTORE IN POSIZIONI CONTIGUE E IN ORDINE CRESCENTE A PARTIRE DALLA PRIMA POSIZIONE.

PER VERIFICARE L'APPARTENENZA DI UN ELEMENTO CON CHIAVE K, SI UTILIZZA LA RICERCA BINARIA (DICOTOMICA, LOGARITMICA), SI CONFRONTA CIOE' IL VALORE DA RICERCARE K CON IL VALORE V CHE OCCUPA LA POSIZIONE CENTRALE DEL VETTORE E SI STABILISCE IN QUALE META' CONTINUARE LA RICERCA. (DI FATTO POSSIAMO RAPPRESENTARE IL DIZIONARIO ANCHE CON UN ALBERO DI RICERCA BILANCIATO)

RAPPRESENTAZIONE CON VETTORE ORDINATO

IN QUESTO CASO LE OPERAZIONI PIU' RILEVANTI PER LA GESTIONE DELLA STRUTTURA HANNO COMPLESSITA'

INSERISCI - **O(n)**

Ho bisogno di:

O(log(n)) confronti → per trovare la giusta posizione in cui inserire l'elemento

O(n) trasferimenti → per mantenere l'array ordinato

(Ricorda che O(n) + O(log(n)) = O(n))

CANCELLA - (come per INSERISCI) **O(n)**

RECUPERA - **O(log(n))**

UNA POSSIBILE REALIZZAZIONE

```
Definizione dei tipi:
Dizionario: tipo strutturato con componenti
   - elementi: array di maxlung elementi di tipo
tipoelem
   - ultimo: intero in [0..maxlung]

APPARTIENE(k:tipoelem; D:dizionario): boolean
   return RICBIN(D.elementi, k, 1, D.ultimo)

RICBIN(V: vettore;
        k:tipoelem; i : integer; j: integer) → boolean

   if i>j then
     RICBIN ← false
   else
     m ← (i + j) div 2
     if k = V[m] then
       RICBIN ← true
     else
       if k < V[m] then
         RICBIN ← RICBIN (V, k, i, m-1)
       else
         RICBIN ← RICBIN (V, k, m+1, j)
```

E con le LISTE?

Lista non Ordinata

RECUPERA O(n)

INSERISCI O(1)

CANCELLA O(n)

Lista Ordinata

RECUPERA O(n) Costerebbe comunque n
(non posso fare dimezzamenti successivi)

INSERISCI O(n) Devo mantenere ordinata
la lista

CANCELLA O(n)

ESISTE UNA TECNICA DENOMINATA "HASH", CHE SI APPOGGIA SU DI UNA STRUTTURA DI DATI TABELLARE, CHE SI PRESTA AD ESSERE USATA PER REALIZZARE DIZIONARI.

CON QUESTA STRUTTURA, LE OPERAZIONI DI RICERCA E DI MODIFICA DI UN DIZIONARIO POSSONO OPERARE IN TEMPI COSTANTI E INDIPENDENTI SIA DALLA DIMENSIONE DEL DIZIONARIO CHE DALL'INSIEME DEI VALORI CHE VERRANNO GESTITI.

RAPPRESENTAZIONE CON TABELLA HASH

IDEA BASE: RICAVARE LA *POSIZIONE* CHE LA CHIAVE OCCUPA IN UN VETTORE *DAL VALORE DELLA CHIAVE.*

LA **RAPPRESENTAZIONE CON TABELLA HASH** E' UNA ALTERNATIVA EFFICACE ALL'INDIRIZZAMENTO DIRETTO IN UN VETTORE PERCHE' LA DIMENSIONE E' PROPORZIONALE AL NUMERO DI CHIAVI ATTESE.

K L'INSIEME DI TUTTE LE POSSIBILI CHIAVI DISTINTE

T IL VETTORE DI DIMENSIONE **m** IN CUI SI MEMORIZZA IL DIZIONARIO

LA SOLUZIONE IDEALE E' DISPORRE DI UNA FUNZIONE DI ACCESSO

$$H: K \rightarrow \{ 1,\ldots,m \}$$

CHE PERMETTA DI RICAVARE LA POSIZIONE H(k) DELLA CHIAVE k NEL VETTORE T COSI' CHE, SE k E v APPARTENGONO ALL'INSIEME DELLE CHIAVI E SONO DIVERSE, AVRANNO POSIZIONI DIVERSE NEL VETTORE.

Notazione

- *U* – Universo di tutte le possibili chiavi
- *K* – Insieme delle chiavi effettivamente memorizzate
- Possibili implementazioni
  - |*U*| corrisponde al range *[0..m-1]*, |*K*| ~ |*U*| →
    - tabelle ad indirizzamento diretto
  - *U* è un insieme generico, |*K*| << |*U*| →
    - tabelle hash

Idea base:

- Chiavi prese da un universo totalmente ordinato K (possono non essere numeri)
- Funzione hash: H: K $\rightarrow$ [0, m-1] (funzione che trasforma chiavi in indici)
- Elemento con chiave k in posizione v[H(k)]

ESISTONO DIVERSE VARIANTI CHE COMUNQUE SI POSSONO FAR RISALIRE AD UNA FORMA *STATICA* E AD UNA FORMA *DINAMICA* O *ESTENSIBILE*.

LA PRIMA FA USO DI STRUTTURE O TABELLE DI DIMENSIONE PREFISSATA, MENTRE L'HASH DINAMICO E' IN GRADO DI MODIFICARE DINAMICAMENTE LE DIMENSIONI DELLA TABELLA HASH SULLA BASE DEL NUMERO DI ELEMENTI CHE VENGONO VIA VIA INSERITI O ELIMINATI.

NEL SEGUITO SI FARA' RIFERIMENTO SOLO ALLA FORMA *STATICA*.

RAPPRESENTAZIONE CON TABELLA HASH

L'*HASH STATICO* PUO' ASSUMERE A SUA VOLTA DUE FORME DIVERSE DENOMINATE RISPETTIVAMENTE

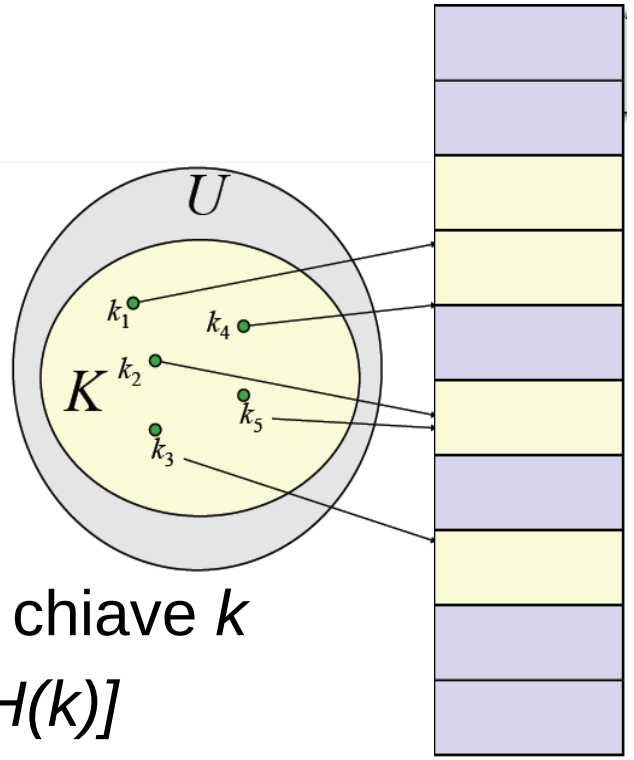
- HASH CHIUSO: CONSENTE DI INSERIRE UN INSIEME LIMITATO DI VALORI IN UNO SPAZIO DI DIMENSIONE FISSA

- HASH APERTO: CONSENTE DI MEMORIZZARE UN INSIEME DI VALORI DI DIMENSIONE QUALSIASI IN UNO SPAZIO POTENZIALMENTE ILLIMITATO

AMBEDUE QUESTE VARIANTI PERO' UTILIZZANO UNA SOTTOSTANTE TABELLA HASH A *DIMENSIONE FISSA* COSTITUITA DA UNA STRUTTURA ALLOCATA SEQUENZIALMENTE IN MEMORIA E CHE ASSUME LA FORMA DI UN ARRAY.

Tabelle hash

- Tabelle hash:
  - Un vettore *A[0..m-1]*
  - Una *funzione hash*
    H: U →{0,..,m-1}



- Indirizzamento hash:
  - Diciamo che *H(k)* è il valore hash della chiave *k*
  - Chiave *k* viene "mappata" nello slot *A[H(k)]*
  - Quando due o più chiavi nel dizionario hanno lo stesso valore hash, diciamo che è avvenuta una *collisione*
- Idealmente: vogliamo funzioni hash senza collisioni

Collisioni

Si ha una **collisione** quando si deve inserire nella tavola hash un elemento con chiave k, e nella tavola esiste già un elemento con chiave v tale che H(k)=H(v): il nuovo elemento andrebbe a sovrascrivere il vecchio!

SEQUENZA DI ARRIVO	INDIRIZZI HASH
ALBINONI	H(ALBINONI) = 1
OFFENBACH	H(OFFENBACH) = 15
PALESTRINA	H(PALESTRINA) = 16
PUCCINI	H(PUCCINI) = 16
PROKOFEV	H(PROKOFEV) = 16
ROSSINI	H(ROSSINI) = 18

MEMORIZZAZIONE

ALBINONI	1
:	
OFFENBACH	15
PALESTRINA	16
PUCCINI	17
PROKOFEV	18
ROSSINI	19
:	
	26

SIA K = { *Cognomi di musicisti* } E SI ASSUMA m= 26. UNA POSSIBILE FUNZIONE E' H(K)=h, $1 \leq h \leq 26$, SE IL CARATTERE INIZIALE DI K E' L'h-ESIMA LETTERA DELL'ALFABETO (INGLESE).

H NON E' BIUNIVOCA!

Funzioni hash perfette

Per evitare il fenomeno delle collisioni, dunque, si devono usare funzioni hash perfette.

Una **funzione hash** si dice **perfetta** se è iniettiva, cioè per ogni $k,v \in K$:

$$k \neq v \Rightarrow H(k) \neq H(v)$$

UTILIZZANDO $m = |K|$ SI HA GARANZIA DI BIUNIVOCITA' E DI ACCESSO DIRETTO ALLA POSIZIONE CONTENENTE LA CHIAVE. ……. SE $|K|$ E' GRANDE, SI HA SPRECO ENORME DI MEMORIA!

LA DIMENSIONE m DEL VETTORE VA SCELTA IN BASE AL NUMERO DI CHIAVI ATTESE. LA SOLUZIONE DI COMPROMESSO E' SCEGLIERE UN m MAGGIORE DI 1 MA MOLTO MINORE DI $|K|$.

Esempio

Tavola hash con i nomi di 100 studenti aventi come chiavi numeri di matricola nell'insieme U=[234717, 235717]

Funzione hash perfetta: h(k) = k - 234717

n=100 m=1001 α = 0,1 = 10%

L'assunzione |U| ≤ m necessaria per avere una funzione hash perfetta è raramente conveniente (o possibile)…

Esempio

Tavola hash con elementi aventi come chiavi lettere dell'alfabeto U={A,B,C,…}

Funzione hash non perfetta (ma buona in pratica per m primo): h(k) = ascii(k) mod m

Ad esempio, per m=11:

h('C') = 67 mod 11=1

h('N')= 78 mod 11=1 $\Rightarrow$ h('C') = h('N')

$\Rightarrow$ se volessimo inserire sia 'C' che 'N' nel dizionario avremmo una collisione!

Uniformità delle funzioni hash

Per ridurre la probabilità di collisioni, una buona funzione hash dovrebbe essere in grado di distribuire in modo uniforme le chiavi nello spazio degli indici della tavola
Questo si ha, ad esempio, se la funzione hash gode della proprietà di uniformità semplice

Uniformità semplice

Sia *P(k)* la probabilità che la chiave k sia presente nel dizionario e sia:

$$Q(i) = \sum_{k:h(k)=i} \mathcal{P}(k)$$

la probabilità che la cella i sia occupata.

Una funzione hash h gode dell'uniformità semplice se, per ogni i=1..m:

$$Q(i) = \frac{1}{m}$$

Riassumendo

- Utilizzo di funzioni hash perfette
  - Una funzione hash *H* si dice *perfetta* se è iniettiva, ovvero:

$$\forall u, v \in U : u \neq v \Rightarrow H(u) \neq H(v)$$

Si noti che questo richiede che $m \geq |U|$

- Problema: spazio delle chiavi spesso grande, sparso, non conosciuto
  - E' spesso impraticabile ottenere una funzione hash perfetta

Risoluzione delle collisioni

Esistono funzioni hash più o meno buone ma le collisioni non si potranno mai eliminare del tutto. Nel caso in cui non si possano evitare, va trovato un modo per risolverle. Due metodi tradizionalmente usati sono i seguenti:

1. **Indirizzamento libero o aperto ($n \leq m$, $\alpha \leq 1$).** Tutti gli elementi sono contenuti nella tabella: se una cella è occupata, se ne cerca un'altra libera
2. **Liste di collisione ($n \geq m$, $\alpha \geq 1$).** Gli elementi sono contenuti in liste esterne alla tabella: v[i] punta alla lista degli elementi tali che h(k)=i

NELL'ESEMPIO DELLA TABELLA DI MUSICISTI SI E' ADOTTATA UNA SEMPLICE STRATEGIA LINEARE PER LA RISOLUZIONE DELLE COLLISIONI:

SE H(k) PER QUALUNQUE CHIAVE k INDICA UNA POSIZIONE GIA' OCCUPATA, SI ISPEZIONA LA POSIZIONE SUCCESSIVA NEL VETTORE.

SE LA POSIZIONE E' PIENA SI PROVA CON LA SEGUENTE E COSI' VIA FINO A TROVARE UNA POSIZIONE LIBERA O TROVARE CHE LA TABELLA E' COMPLETAMENTE PIENA.

UNA POSIZIONE LIBERA PUO' VENIRE FACILMENTE SEGNALATA IN FASE DI REALIZZAZIONE DA UNA CHIAVE FITTIZIA *LIBERO*.

PER LA CANCELLAZIONE E' PIU' SEMPLICE SOSTITUIRE L'OGGETTO CANCELLATO CON UNA CHIAVE FITTIZIA *CANCELLATO* CHE DOVREBBE ESSERE FACILMENTE DISTINGUIBILE DALLE ALTRE CHIAVI REALI E DALL'ALTRA CHIAVE FITTIZIA *LIBERO*.

LA STRATEGIA LINEARE PUO' PRODURRE NEL TEMPO IL CASUALE ADDENSAMENTO DI INFORMAZIONI IN CERTI TRATTI DELLA TABELLA (AGGLOMERATI), PIUTTOSTO CHE UNA LORO DISPERSIONE.

HASH STATICO CHIUSO

HASH CHIUSO

LA STRUTTURA E' COMPOSTA DA UN CERTO NUMERO (MAXBUCKET) DI CONTENITORI DI UGUALE DIMENSIONE DENOMINATI BUCKET.
(ORIGINARIAMENTE ERA L'INSIEME DI PAGINE CONTENENTI LE ETICHETTE DEI RECORD DEI DATI. SE UNA PAGINA BUCKET PRIMARIA E' PIENA VIENE CREATA UNA PAGINA DI OVERFLOW.)

OGNI BUCKET PUO' MANTENERE AL PROPRIO INTERNO AL MASSIMO UN NUMERO $nb \geq 1$ DI ELEMENTI CHE COMPRENDERANNO LA CHIAVE E IL CORRISPONDENTE ATTRIBUTO (NEL CASO nb=1 OGNI BUCKET AVRA' UNA SOLA COPPIA CHIAVE-ATTRIBUTO)

HASH STATICO APERTO

- **HASH APERTO**

LA STRUTTURA SARA' COMPOSTA DA UN CERTO NUMERO INDETERMINATO DI CONTENITORI BUCKET.

IN AMBEDUE I CASI VIENE USATA UNA FUNZIONE ARITMETICA ALLO SCOPO DI CALCOLARE, PARTENDO DALLA CHIAVE, LA POSIZIONE IN TABELLA DELLE INFORMAZIONI CONTENUTE NELL'ATTRIBUTO COLLEGATO ALLA CHIAVE.

LISTE DI TRABOCCO

UNA TECNICA CHE EVITA LA FORMAZIONE DI AGGLOMERATI E' QUELLA DELL'HASH APERTO CHE RICHIEDE CHE LA TABELLA HASH MANTENGA LA LISTA DEGLI ELEMENTI LE CUI CHIAVI PRODUCONO LO STESSO VALORE DI FUNZIONE (TRASFORMATA).

LA TABELLA HASH VIENE REALIZZATA DEFINENDO UN ARRAY DI LISTE DI BUCKET (LISTE DI TRABOCCO).

LA FUNZIONE HASH VIENE UTILIZZATA PER DETERMINARE QUALE LISTA POTREBBE CONTENERE L'ELEMENTO CHE POSSIEDE UNA DETERMINATA CHIAVE IN MODO DA POTER ATTIVARE UNA SUCCESSIVA OPERAZIONE DI RICERCA NELLA LISTA CORRISPONDENTE E DA RESTITUIRE LA POSIZIONE DEL BUCKET CHE CONTIENE LA CHIAVE.

LISTE DI TRABOCCO

T CONTIENE, IN OGNI POSIZIONE, UN PUNTATORE AD UNA LISTA

APPARTIENE, INSERISCI E CANCELLA SI RIDUCONO A RICERCA INSERZIONE E CANCELLAZIONE DI LISTA

IN DEFINITIVA:

QUALE CHE SIA LA FUNZIONE HASH ADOTTATA, DEVE ESSERE PREVISTA UNA STRATEGIA PER GESTIRE IL PROBLEMA DEGLI AGGLOMERATI E DELLE COLLISIONI.

- **OCCORRE UNA *FUNZIONE HASH*, CALCOLABILE VELOCEMENTE E CHE DISTRIBUISCA LE CHIAVI UNIFORMEMENTE IN T, IN MODO DA RIDURRE LE COLLISIONI;**
- **OCCORRE UNA STRATEGIA PER LA SOLUZIONE DELLE COLLISIONI (METODO DI SCANSIONE) UTILE A REPERIRE CHIAVI CHE HANNO TROVATO LA POSIZIONE OCCUPATA E CHE NON PROVOCHI LA FORMAZIONE DI AGGLOMERATI DI CHIAVI;**

INOLTRE

- **LA DIMENSIONE m DEL VETTORE T DEVE ESSERE UNA SOVRASTIMA DEL NUMERO DELLE CHIAVI ATTESE, PER EVITARE DI RIEMPIRE T COMPLETAMENTE.**

PER DEFINIRE FUNZIONI HASH, E' CONVENIENTE CONSIDERARE LA RAPPRESENTAZIONE BINARIA bin(k) DELLA CHIAVE k . NEL CASO DI CHIAVI NON NUMERICHE SI PUO' CONSIDERARE LA RAPPRESENTAZIONE BINARIA DEI CARATTERI E bin(K) E' DATA DALLA LORO CONCATENAZIONE.

IN MOLTI LINGUAGGI ESISTE UNA FUNZIONE PRIMITIVA CHE, APPLICATA A UN CARATTERE, NE RESTITUISCE IL CODICE INTERO (IN PASCAL è ord)

4 BUONI METODI DI GENERAZIONE HASH

DENOTIAMO CON int(b) IL NUMERO INTERO RAPPRESENTATO DA UNA STRINGA BINARIA b. INDICHIAMO DA 0 A m-1 GLI ELEMENTI DI T.

a) H(k)=int(b), DOVE b E' UN SOTTOINSIEME DI p BIT DI bin(k), SOLITAMENTE ESTRATTI NELLE POSIZIONI CENTRALI

b) H(k)=int(b), DOVE b E' DATO DALLA SOMMA MODULO 2, EFFETTUATA BIT A BIT, DI DIVERSI SOTTOINSIEMI DI p BIT DI bin(k).

c) H(k)=int(b), DOVE b E' UN SOTTOINSIEME DI p BIT ESTRATTI DALLE POSIZIONI CENTRALI DI $bin(int(bin(K))^2)$.

d) H(k) E' UGUALE AL RESTO DELLA DIVISIONE int(bin(K)) /m (m E' DISPARI; SE FOSSE UGUALE A 2^P, DUE NUMERI CON GLI STESSI p BIT FINALI DAREBBERO SEMPRE LUOGO A UNA COLLISIONE).

L'ULTIMA FUNZIONE HASH DEFINITA E' LA MIGLIORE DAL PUNTO DI VISTA PROBABILISTICO E FORNISCE UN'ECCELLENTE DISTRIBUZIONE DEGLI INDIRIZZI H(k) NELL'INTERVALLO [0, m-1].

ESEMPIO

(FUNZIONE HASH): SI SUPPONGA CHE LE CHIAVI SIANO DI 6 CARATTERI ALFANUMERICI (CHIAVI PIU' LUNGHE SONO TRONCATE, MENTRE CHIAVI PIU' CORTE SONO ESPANSE A DESTRA CON SPAZI). SI ASSUMA ord(A)=1, ord(B)=2, … , ord(Z)=26 E ord(b)=32, DOVE b INDICA LO SPAZIO, CON RAPPRESENTAZIONE DI OGNI ORDINALE SU 6 BIT.

PER LE CHIAVI WEBER E WEBERN SI OTTIENE:

bin(WEBERb) = 010111 000101 000010 000101 010010 100000 0000
bin(WEBERN) = 010111 000101 000010 000101 010010 001110 0000

DOVE SI SONO EVIDENZIATI PER CHIAREZZA I GRUPPI DI 6 BIT CHE RAPPRESENTANO CIASCUN CARATTERE.

ESEMPIO/1

CALCOLIAMO GLI INDIRIZZI HASH OTTENUTI CON LE FUNZIONI DEFINITE DI SEGUITO IN (a), (b) E (d)

bin(WEBERb) = 010111 000101 000010 000101 010010 100000 0000
bin(WEBERN) = 010111 000101 000010 000101 010010 001110 0000

(a) SIA m = 256 = 2^8. ESTRAENDO GLI 8 BIT DALLA POSIZIONE 15 ALLA 22 DI bin(k),

H(WEBER) = H(WEBERN) = Int(00100001) = 33.

LE DUE CHIAVI DANNO PERTANTO LUOGO AD UNA COLLISIONE

ESEMPIO/2

(b) SIA ANCORA $m = 256 = 2^8$ E SI CALCOLI b RAGGRUPPANDO bin(k) IN CINQUE GRUPPI DI 8 BIT (DOPO AVER ESPANSO A DESTRA bin(k) CON QUATTRO ZERI). H(WEBER) = int(11000011) = 195, POICHE'

11000011 = 01011100 ⊕ 01010000 ⊕ 10000101 ⊕ 01001010 ⊕ 00000000

DOVE ⊕ INDICA LA SOMMA BIT A BIT MODULO 2.

ANALOGAMENTE, H(WEBERN) = int(00100001) = 33, POICHE'

00100001 = 01011100 ⊕ 01010000 ⊕ 10000101 ⊕ 01001000 ⊕ 11100000

BENCHE' H(WEBERN) SIA UGUALE A 33, COME NEL CASO (a), LA COLLISIONE E' STATA ELIMINATA.

ESEMPIO/3

(d) SIA IL NUMERO DISPARI m = 383.

UTILIZZANDO 6 BIT, int(bin(k)) E' UN NUMERO IN BASE 2^6 = 64.

Int(bin(WEBERN))= $23 * 64^5 + 5 * 64^4 + 2 * 64^3 + 5 * 64^2 + 18 * 64^1 + 14 * 64^0$=
64(64(64(64(23*64)+5)+2)+5)+18)+14

H(WEBERN) E H(WEBER) SONO OTTENUTI PRENDENDO IL RESTO DELLA DIVISIONE

Int(bin(…...))/383

METODI DI SCANSIONE:

UNA SISTEMATIZZAZIONE

NELLA REALIZZAZIONE CON HASH APERTO E LISTE DI TRABOCCO SI E' USATO UN METODO DI SCANSIONE ESTERNO CONTRAPPOSTO ALLA SCANSIONE LINEARE CHE E' UN METODO DI SCANSIONE INTERNO.

ALTRI METODI INTERNI (O CHIUSI) SONO

SCANSIONE QUADRATICA
SCANSIONE PSEUDOCASUALE
HASHING DOPPIO

SCANSIONE INTERNA

CHIAMIAMO f_i LA FUNZIONE CHE VIENE UTILIZZATA L'i-ESIMA VOLTA CHE SI TROVA OCCUPATA UNA POSIZIONE DEL VETTORE T, $i \geq 0$, (PER $i=0$, $f_0 = H$).

f_i VA SCELTA IN MODO DA TOCCARE TUTTE LE POSIZIONI DI T (UNA SOLA VOLTA).

SCANSIONE LINEARE:

$$f_i = (H(k) + h*i) \text{ MOD } m$$

h E' UN INTERO POSITIVO PRIMO CON m

h RAPPRESENTA LA DISTANZA TRA DUE POSIZIONI SUCCESSIVE ESAMINATE NELLA SCANSIONE (SE h = 1, *SCANSIONE A PASSO UNITARIO*).

ESSENDO h E m PRIMI TRA LORO, VENGONO ESAMINATE TUTTE LE POSIZIONI DI T PRIMA DI RICONSIDERARE LE POSIZIONI GIA' ESAMINATE.

SVANTAGGIO: NON RIDUCE LA FORMAZIONE DI AGGLOMERATI

SCANSIONE QUADRATICA

$f_i = (H(k) + h*i + i(i-1)/2) \text{ MOD } m$

m E' PRIMO.

LA DISTANZA TRA DUE POSIZIONI SUCCESSIVE NELLA SEQUENZA E' VARIABILE, QUINDI LA POSSIBILITA' DI AGGLOMERATI E' RIDOTTA.

SVANTAGGIO: LA SEQUENZA DI SCANSIONE NON INCLUDE TUTTE LE POSIZIONI DI T (SVANTAGGIO TRASCURABILE PER m NON TROPPO PICCOLO).

SCANSIONE PSEUDOCASUALE

$f_i = (H(k) + p_i) \text{ MOD } m$

p_i E' L'i-ESIMO NUMERO GENERATO DA UN GENERATORE DI NUMERI PSEUDOCASUALI, CHE GENERA GLI INTERI TRA 1 E m UNA SOLA VOLTA IN UN ORDINE QUALUNQUE.

HASHING DOPPIO

$f_i = (H(k) + i * F(k)) \text{ MOD } m$

F E' UN'ALTRA FUNZIONE HASH DIVERSA DA H.

PROBLEMI PER LA CANCELLAZIONE DI CHIAVI

USANDO METODI DI SCANSIONE INTERNA E POTENDO CANCELLARE CHIAVI, NON SI E' MAI SICURI CHE, RAGGIUNTA UNA POSIZIONE VUOTA NELLA RICERCA DI k, TALE CHIAVE NON SI TROVI IN UN'ALTRA POSIZIONE DI T, POICHE' LA POSIZIONE ORA VUOTA ERA OCCUPATA QUANDO k E' STATA INSERITA.

BISOGNA DUNQUE SCANDIRE ANCHE LE POSIZIONI IN CUI SI E' CANCELLATO E FERMARSI O SULLA POSIZIONE MAI RIEMPITA O DOPO ESSERE TORNATI SU UNA POSIZIONE GIA' SCANDITA.

CIO' DETERMINA UN AUMENTO DEL TEMPO DI RICERCA.

UTILIZZARE IL METODO DI SCANSIONE ESTERNA SE SONO PREVISTE MOLTE CANCELLAZIONI.