

FILE 5: HASH, HUFFMAN E LCS

PARTE 1: HASHING (TAVOLE HASH)

Concetti Base

- **Universo chiavi U:** insieme di tutte le possibili chiavi
- **Tavola hash T[0..m-1]:** array di dimensione m << |U|
- **Funzione hash h(k):** mappa chiave k in posizione $h(k) \bmod m$
- **Collisione:** quando $h(k_1) = h(k_2)$ con $k_1 \neq k_2$

Risoluzione Collisioni

1. CHAINING (Liste di trabocco)

Idea: ogni cella contiene lista concatenata

Esempio Completo: Chaining con $h(k) = k \bmod m$, $m = 8$

Inserire sequenza: 28, 19, 10, 35, 26

Soluzione:

- 1) $28 \bmod 8 = 4 \rightarrow$ Inserisco 28 in posizione 4
- 2) $19 \bmod 8 = 3 \rightarrow$ Inserisco 19 in posizione 3
- 3) $10 \bmod 8 = 2 \rightarrow$ Inserisco 10 in posizione 2
- 4) $35 \bmod 8 = 3 \rightarrow$ COLLISIONE \rightarrow Creo lista L = {35 → 19}
- 5) $26 \bmod 8 = 2 \rightarrow$ COLLISIONE \rightarrow Creo lista L = {26 → 10}

Tavola finale:

```
[0]
[1]
[2] → 26 → 10
[3] → 35 → 19
[4] → 28
[5]
[6]
[7]
```

Complessità con chaining:

- Inserimento: O(1)
- Ricerca: $O(1 + \alpha)$ medio, dove $\alpha = n/m$ (fattore di carico)
- Se $n \leq c \cdot m \rightarrow \alpha$ costante $\rightarrow O(1)$ medio

2. OPEN ADDRESSING (Indirizzamento aperto)

Idea: tutte le chiavi nell'array, ispezione per trovare posto libero

Formula generale: $h(k, i)$ dove $i = \text{numero tentativo}$

a) Ispezione Lineare: $h(k, i) = (h(k) + i) \bmod m$

- Pro: semplice
- Contro: addensamento primario

b) Ispezione Quadratica: $h(k, i) = (h(k) + c_1 \cdot i + c_2 \cdot i^2) \bmod m$

- Contro: addensamento secondario

c) Doppio Hashing (MIGLIORE):

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

dove:

- $h_1(k) = k \bmod m$
- $h_2(k) = 1 + k \bmod (m-2)$
- $\text{MCD}(h_2(k), m) = 1$ (per ispezionare tutte le celle)

Esempio Completo: Doppio Hashing con $m = 8$

Inserire sequenza: 28, 19, 10, 35, 26

$$\begin{aligned} h_1(k) &= k \bmod 8 \\ h_2(k) &= 1 + k \bmod 6 \\ h(k, i) &= (h_1(k) + i \cdot h_2(k)) \bmod 8 \end{aligned}$$

Soluzione:

1) $h(28, 0) = (28 \bmod 8 + 0 \cdot (1 + 28 \bmod 6)) \bmod 8 = 4$
Inserisco 28 in posizione 4

2) $h(19, 0) = (19 \bmod 8 + 0 \cdot (1 + 19 \bmod 6)) \bmod 8 = 3$
Inserisco 19 in posizione 3

3) $h(10, 0) = (10 \bmod 8 + 0 \cdot (1 + 10 \bmod 6)) \bmod 8 = 2$
Inserisco 10 in posizione 2

4) $h(35, 0) = (35 \bmod 8 + 0 \cdot (1 + 35 \bmod 6)) \bmod 8 = 3 \rightarrow \text{COLLISIONE}$
$$\begin{aligned} h(35, 1) &= (3 + 1 \cdot (1 + 35 \bmod 6)) \bmod 8 \\ &= (3 + 1 \cdot (1 + 5)) \bmod 8 \\ &= 9 \bmod 8 = 1 \end{aligned}$$

Inserisco 35 in posizione 1

```

5)  $h(26, 0) = (26 \bmod 8 + 0 \cdot (1 + 26 \bmod 6)) \bmod 8 = 2 \rightarrow \text{COLLISIONE}$ 
    $h(26, 1) = (2 + 1 \cdot (1 + 26 \bmod 6)) \bmod 8$ 
   =  $(2 + 1 \cdot (1 + 2)) \bmod 8$ 
   =  $5 \bmod 8 = 5$ 
Inserisco 26 in posizione 5

```

Tavola finale:

[0]
[1] → 35
[2] → 10
[3] → 19
[4] → 28
[5] → 26
[6]
[7]

Template Risoluzione Esercizi Hash

PASSO 1: Identificare tipo (chaining o open addressing)

PASSO 2: Calcolare $h(k)$ per ogni chiave

PASSO 3: In caso di collisione:

- Chaining: aggiungi in testa alla lista
 - Open addressing: incrementa i e ricalcola $h(k, i)$
-

PARTE 2: HUFFMAN CODING

Problema

Data frequenza di caratteri, costruisci codice prefisso che minimizza lunghezza media codifica.

Algoritmo Greedy

Scelta greedy: unisci sempre i due nodi con frequenza minima

Pseudocodice

```

HUFFMAN(frequencies)
1. Q = MinHeap(frequencies) // Heap di priorità minima
2. while |Q| > 1:
3.   x = ExtractMin(Q)
4.   y = ExtractMin(Q)
5.   z = CreateNode(x.freq + y.freq)
6.   z.left = x
7.   z.right = y

```

```
8. Insert(Q, z)
9. return ExtractMin(Q) // Radice dell'albero
```

Complessità: $O(n \log n)$

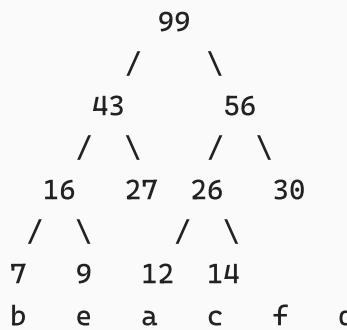
Esempio Completo: Alfabeto {a,b,c,d,e,f}

Frequenze: a=12, b=7, c=14, d=30, e=9, f=27

Step-by-step:

1. Iniziale: a:12, b:7, c:14, d:30, e:9, f:27
Minori: b:7, e:9
Merge → nodo1:16
2. Stato: a:12, nodo1:16, c:14, d:30, f:27
Minori: a:12, c:14
Merge → nodo2:26
3. Stato: nodo1:16, nodo2:26, d:30, f:27
Minori: nodo1:16, f:27
Merge → nodo3:43
4. Stato: nodo2:26, d:30, nodo3:43
Minori: nodo2:26, d:30
Merge → nodo4:56
5. Stato: nodo3:43, nodo4:56
Minori: nodo3:43, nodo4:56
Merge → root:99

Albero finale:



Codifiche (0=sinistra, 1=destra):

```
b = 000
e = 001
f = 01
a = 100
c = 101
d = 11
```

Proprietà Huffman

1. **Proprietà scelta greedy:** esiste soluzione ottima contenente scelta greedy
2. **Sottostruttura ottima:** dopo merge, sottoproblema ha soluzione ottima
3. **Dimostrazione:** exchange argument (cut & paste)

PARTE 3: LCS (LONGEST COMMON SUBSEQUENCE)

Problema

Date due stringhe $X[1..m]$ e $Y[1..n]$, trova la sottosequenza comune più lunga.

Nota: sottosequenza NON richiede caratteri consecutivi (differenza da substring)

Caratterizzazione Ricorsiva

```
LCS[i,j] = lunghezza LCS di X[1..i] e Y[1..j]
```

```
LCS[i,j] = {  
    0                      se i=0 o j=0  
    LCS[i-1,j-1] + 1      se X[i] = Y[j]  
    max(LCS[i-1,j], LCS[i,j-1]) se X[i] ≠ Y[j]  
}
```

Algoritmo Bottom-Up

```
LCS_LENGTH(X, Y, m, n)  
1. for i = 0 to m: L[i,0] = 0  
2. for j = 0 to n: L[0,j] = 0  
3. for i = 1 to m:  
4.   for j = 1 to n:  
5.     if X[i] = Y[j]:  
6.       L[i,j] = L[i-1,j-1] + 1  
7.     else:  
8.       L[i,j] = max(L[i-1,j], L[i,j-1])  
9. return L[m,n]
```

Complessità: $\Theta(m \cdot n)$ tempo e spazio

Esempio Completo 1: $X = "armo"$, $Y = "oro"$

Tabella $L[i,j]$:

"	"	o	r	o
"	0	0	0	0

a	0	0	0	0
r	0	0	1	1
m	0	0	1	1
o	0	1	1	2

Spiegazione:

- $L[4,3] = 2 \rightarrow$ LCS ha lunghezza 2
- Sequenza: "ro" (r comune in pos 2, o comune in pos 4)

Esempio Completo 2: X = "armo", Y = "toro"

Tabella L[i,j]:

	"	t	o	r	o
"	0	0	0	0	0
a	0	0	0	0	0
r	0	0	0	1	1
m	0	0	0	1	1
o	0	0	1	1	2

Risultato: LCS = 2 (sequenza "ro" oppure "or")

Esempio Completo 3: X = "store", Y = "shoes"

Tabella L[i,j]:

	"	s	h	o	e	s
"	0	0	0	0	0	0
s	0	1	1	1	1	1
t	0	1	1	1	1	1
o	0	1	1	2	2	2
r	0	1	1	2	2	2
e	0	1	1	2	3	3

Risultato: LCS = 3 (sequenza "soe")

Ricostruzione Soluzione

```

PRINT_LCS(L, X, Y, i, j)
1. if i = 0 or j = 0:
2.   return
3. if X[i] = Y[j]:
4.   PRINT_LCS(L, X, Y, i-1, j-1)
5.   print X[i]
6. else if L[i-1, j] ≥ L[i, j-1]:
7.   PRINT_LCS(L, X, Y, i-1, j)

```

```

8. else:
9.     PRINT_LCS(L, X, Y, i, j-1)

```

Versione Memoizzata (Top-Down)

```

LCS_MEMO(X, Y, i, j, memo)
1. if i = 0 or j = 0:
2.     return 0
3. if memo[i,j] ≠ -1:
4.     return memo[i,j]
5. if X[i] = Y[j]:
6.     memo[i,j] = 1 + LCS_MEMO(X, Y, i-1, j-1, memo)
7. else:
8.     memo[i,j] = max(LCS_MEMO(X, Y, i-1, j, memo),
                      LCS_MEMO(X, Y, i, j-1, memo))
9. return memo[i,j]

```

RIEPILOGO COMPARATIVO

Problema	Tecnica	Complessità	Spazio
Hash Chaining	-	O(1) medio	O(n+m)
Hash Open Addr	-	O(1) medio	O(m)
Huffman	Greedy	O(n log n)	O(n)
LCS	PD	O(m·n)	O(m·n)

TEMPLATE RISOLUZIONE

Hash

1. Identifica m e funzione hash
2. Per ogni chiave k:
 - Calcola h(k)
 - Se collisione: applica strategia (chaining/open addressing)
3. Costruisci tavola finale

Huffman

1. Ordina caratteri per frequenza
2. Ripeti finché heap ha più di un elemento:

- Estrai i due minimi
 - Crea nodo padre con somma frequenze
 - Reinserisci nodo padre
3. Leggi codifiche dall'albero (0=sx, 1=dx)

LCS

1. Crea tabella $L[0..m, 0..n]$
 2. Inizializza riga 0 e colonna 0 a zero
 3. Riempì tabella con formula ricorsiva
 4. $L[m,n]$ contiene lunghezza soluzione
 5. Ricostruisci sequenza con backtracking
-

ERRORI COMUNI DA EVITARE

Hash

- Dimenticare di applicare mod m
- Confondere chaining con open addressing
- Non gestire correttamente le collisioni
- Mostrare tutti i passi dell'inserimento

Huffman

- Non ordinare correttamente le frequenze
- Unire nodi sbagliati (non i due minimi)
- Albero non binario completo
- Verificare che ogni nodo interno abbia 2 figli

LCS

- Dimenticare casi base (riga/colonna 0)
- Confondere sottosequenza con substring
- Errori nella formula ricorsiva
- Controllare direzione frecce per ricostruzione