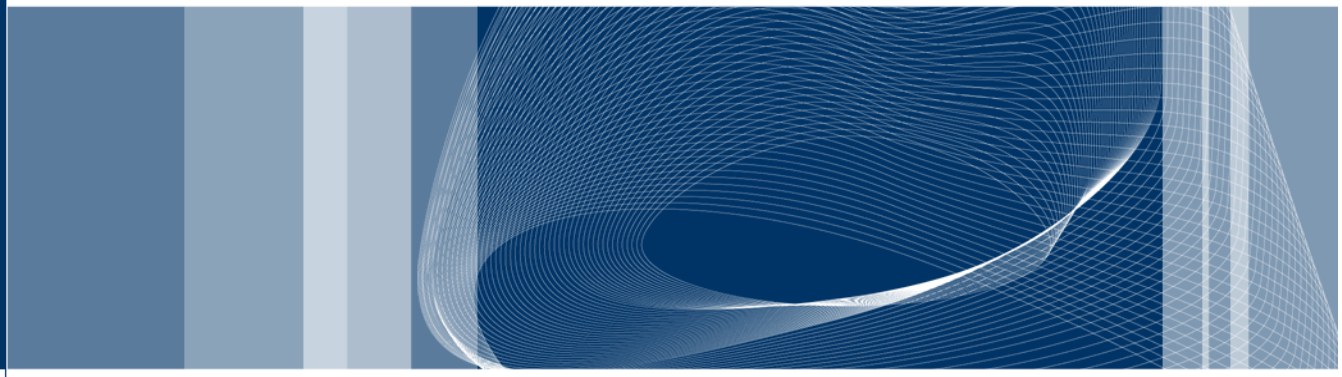
POLITECNICO DI MILANO



Algoritmi e Strutture Dati

Laboratorio 15/12/2008

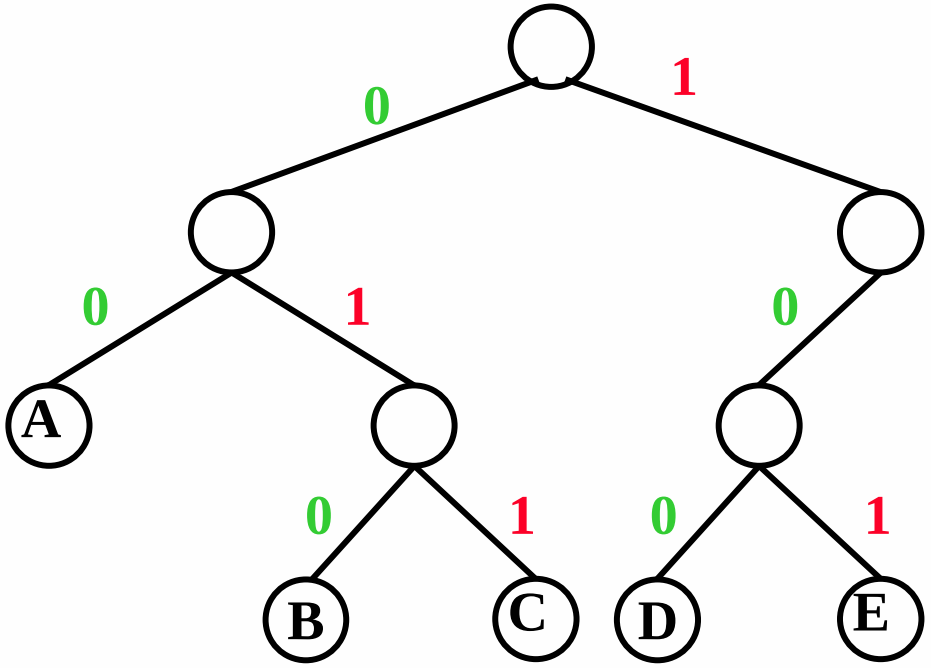
Daniele Loiacono

Problema della compressione

- Rappresentare i dati in modo efficiente
  - Impiegare il numero minore di bit per la rappresentazione
  - Goal: risparmio spazio su disco e tempo di trasferimento
- Una possibile tecnica di compressione: *codifica di caratteri*
  - Tramite *funzione di codifica f: f(c) = x*
    - *c* è un possibile carattere preso da un alfabeto Σ
    - *x* è una rappresentazione binaria
    - "*c* è rappresentato da *x*"

Rappresentazione ad albero per la decodifica

- Rappresentazione come alberi binari
  - Figlio sinistro: 0 Figlio destro: 1
  - Caratteri dell'alfabeto sulle foglie



Algoritmo di decodifica:

1. parti dalla radice
2. leggi un bit alla volta percorrendo l'albero:
   0: sinistra
   1: destra
3. stampa il carattere della foglia
4. torna a 1

A: 00
B: 010
C: 011
D: 100
E: 101

Algoritmo di Huffman

- Principio del codice di Huffman
  - Minimizzare la lunghezza dei caratteri che compaiono più frequentemente
  - Assegnare ai caratteri con la frequenza minore i codici corrispondenti ai percorsi più lunghi all'interno dell'albero
- Un codice è progettato per un file specifico
  - Si ottiene la frequenza di tutti i caratteri
  - Si costruisce il codice
  - Si rappresenta il file tramite il codice

Algoritmo di Huffman

Passo 1: Costruire una lista di nodi foglia per ogni carattere, etichettato con la propria frequenza

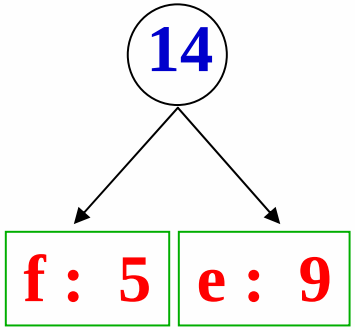
f : 5	e : 9	c : 12	b : 13	d : 16	a : 45

Costruzione del codice

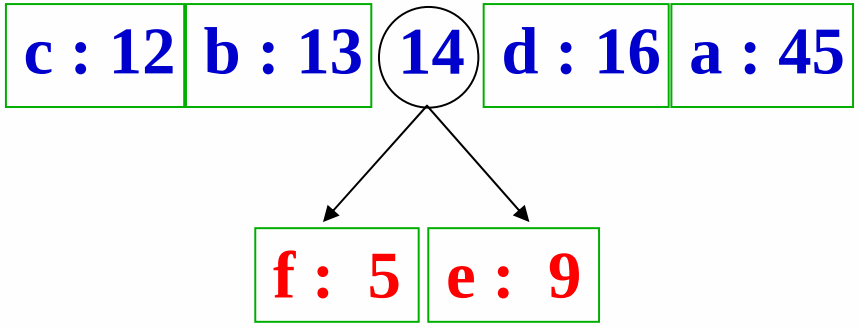
Passo 2: Rimuovere i due nodi "*più piccoli*" (con frequenze minori)

Passo 3: Collegarli ad un nodo padre etichettato con la frequenza combinata (sommata)

c : 12	b : 13	d : 16	a : 45

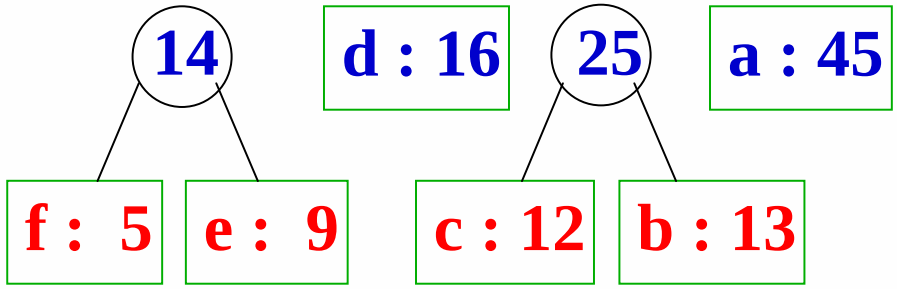


Passo 4: Aggiungere il nodo combinato alla lista.

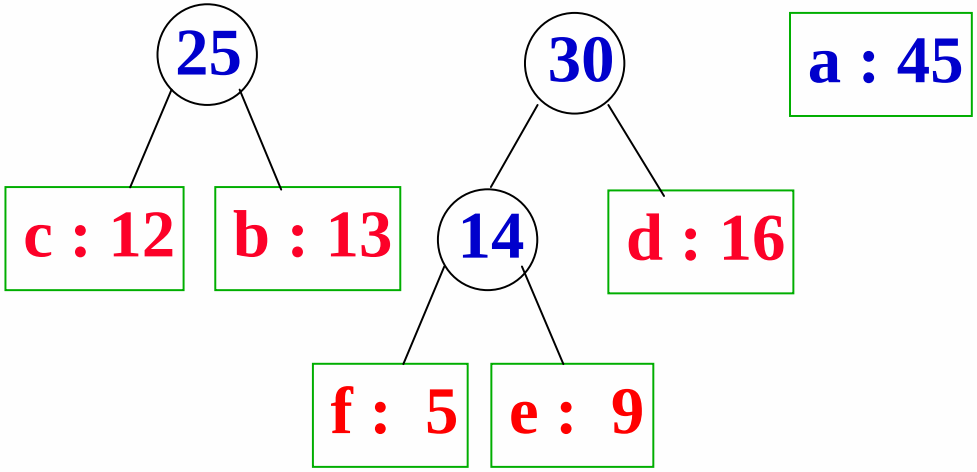


Costruzione del codice

Ripetere i passi 2-4 fino a quando non resta un solo nodo nella lista

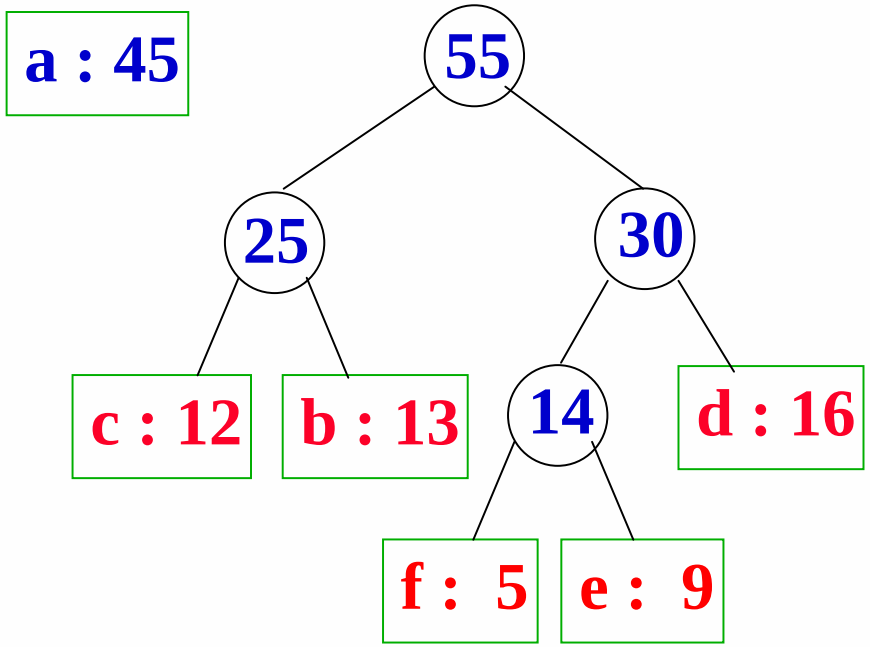


Ripetere i passi 2-4 fino a quando non resta un solo nodo nella lista



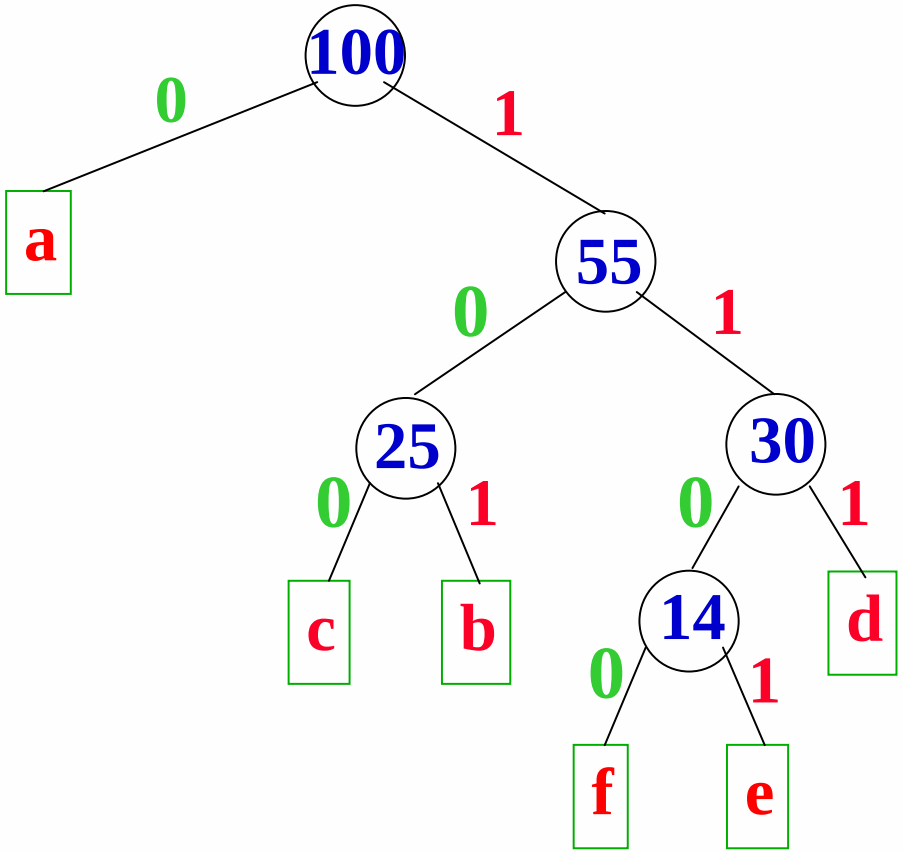
Costruzione del codice

Ripetere i passi 2-4 fino a quando non resta un solo nodo nella lista



Costruzione del codice

Al termine, si etichettano gli archi dell'albero con bit 0-1



Algoritmo in pseudo-codice

```
Huffman(f[1..n], c[1..n])
  Q := new PriorityQueue()
  for i := 1 to n
    z = new Tree(f[i], c[i])
    Q.enqueue(z)
  for i := 1 to n - 1
    z1 = Q.dequeue()
    z2 = Q.dequeue()
    z = new Tree(z1.f + z2.f,
  '')
    z.left = z1
    z.right = z2
    Q.enqueue(z)
  return Q.dequeue()
```

Tree:

f	// frequenza (key)
c	// carattere
left	// figlio sinistro
right	// figlio destro

Complessità
θ(n log n)

Esercizio 1

- Si scriva una funzione che implementa l'algoritmo descritto in precedenza per costruire un albero di decodifica
- Si supponga che la funziona riceva come parametri in ingresso un vettore contenente i caratteri ed un vettore contenente le relative frequenze
- Si supponga che i due vettori NON siano in alcun modo ordinati.
- La funzione deve restituire come output un albero di decodifica

Esercizio 2

- Come si può costruire una tabella di codifica a partire dall'albero di decodifica?
- Implementare una funzione che riceva come input un albero di decodifica e come output una tabella di codifica
- Si implementi la tabella di codifica come una tabella che associa a ciascun carattere nel codice ASCII (cioè a ciascuno dei caratteri con codice fra 0 e 255) una stringa che ne rappresenta la codifica o la stringa vuota laddove non è prevista alcuna codifica per il carattere

Esercizio 3

- Utilizzando le funzioni implementate in precedenza, implementare una funzione che data una stringa in ingresso la codifica utilizzando il codice con costo ottimo
- La funzione riceve in ingresso solo la stringa da codificare e deve perciò costruire una tabella di codifica opportuna
- L'output della funzione dovrà essere la stringa opportunamente codificata e un albero di decodifica