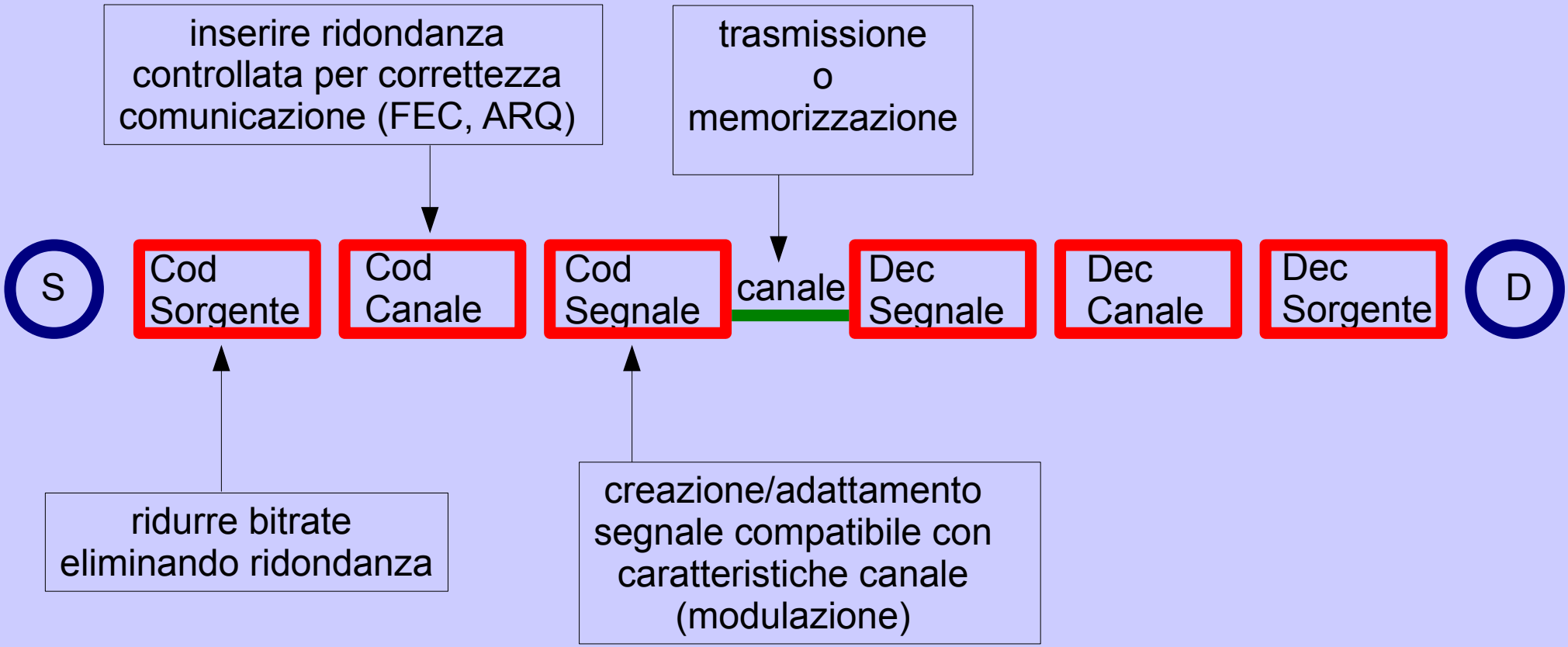
Capitolo 3
Compressione sorgenti statiche

- **Elementi di codifica di sorgente**
  - Entropia
  - Classificazione degli algoritmi
- **Testo**
  - RLE
  - Huffman statico
  - Huffman dinamico
  - Codifica aritmetica
  - LZ
- **Immagini**
  - Prestazioni
  - GIF
  - JPEG

Elementi di codifica



Entropia

- Codifica sorgente → occorre una misura relativa alla quantità di informazione presente
- **Entropia, H(X): MISURA DELLA QUANTITÀ DI INFORMAZIONE PRESENTE ALLA SORGENTE X**, indice dell'incertezza di erogazione di un simbolo
- X sorgente originale, C informazione compressa → |C| < |X|
- Realizzazione j-esima, xj e cj
- **Shannon → Informazione mutua** $I(cj,xj)=\log_2 P(xj|cj)/P(xj) = \log_2 [P(xj,cj)]/[P(xj)P(c_j)]$
  - xj e cj indipendenti → I(cj,xj)=0
  - xj e cj completamenti dipendenti (conoscere cj è come conoscere xj) → $I(cj,xj)=I(xj)=\log_2 [1/P(xj)] = \log_2 [(P(xj))^{-1}] = - \log_2 P(xj)$ autoinformazione del simbolo xj
- Media di tutte le possibili rappresentazioni → I(X)=H(X) informazione media di un generico simbolo emesso dalla sorgente

$$I(X) = \sum_j P(x_j) I(x_j) = -\sum_j P(x_j) \log_2 P(x_j) = H(X)$$

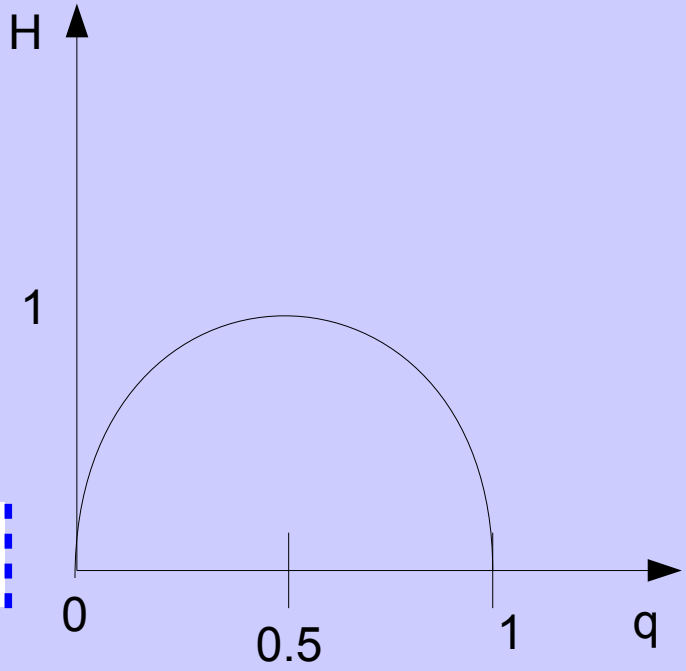
Entropia – *casi particolari*

- **Simboli equiprobabili → ENTROPIA MASSIMA**
- **Sorgente che emette sempre lo stesso simbolo → entropia nulla**
  - Sorgente con n simboli equiprobabili → P(xj)=1/n, per ogni j
  - $H(x) = -\sum_j \frac{1}{n} \log_2 \frac{1}{n} = \log_2 n$

$$0 <= H(X) \leq \log_2 n$$

- **Sorgente binaria**
  - Probabilità di simbolo 0 → q
  - Probabilità di simbolo 1 → 1-q

$$H(q) = -q \log_2 q - (1-q) \log_2 (1-q)$$



Classificazione degli algoritmi di compressione

- **LOSSLESS (senza perdita)**
  - Reversibili
  - Al ricevitore è possibile, sulla base della conoscenza di quanto ricevuto, reintrodurre la ridondanza eliminata e ricostruire l'esatta informazione originale
- **LOSSY (con perdita)**
  - Non reversibili
  - Al ricevitore non si vuole ricostruire l'esatta informazione originale ma una versione che venga percepita come tale dall'essere umano
  - Parametrici: possibilità di definire livello di compressione e di degrado in funzione di un parametro
- **CODICI A LUNGHEZZA FISSA**
  - Facile implementazione ma basso livello di compressione
- **CODICI A LUNGHEZZA VARIABILE**
  - Implementazione complessa ma alto livello di compressione

Testo – Run Length Encoding (RLE)

- **Tecnica di compressione dove vengono processati dei RUN**
- **RUN → sequenza dove un ELEMENTO (SINGOLO CARATTERE o STRINGA) è presente più volte consecutivamente**
- **Ogni run viene sostituito dall'elemento e dal numero di volte in cui tale elemento si ripete (Lunghezza del run).**
- **Se ELEMENTO è singolo carattere è un codice a lunghezza fissa, se è una stringa è un codice a lunghezza variabile**
- Esempio con singolo carattere
  - AAAAABBBAABBBBBBBAAAA → 5A 3B 2A 6B 4A
  - Lunghezza run espressa in binario con 8 bit → lunghezza max 256
  - Codifica a lunghezza fissa, ogni codice è lungo 2 simboli da 8 bit ciascuno (lunghezza run e simbolo ASCII)
- Prestazioni
  - fi = lunghezza, in simboli, del run i-esimo
  - L = totale run presenti
  - Condizione di convenienza → $\sum_{i=1}^{L} f_i > 2L$
  - pj = P[fi = j] ----------→ $\frac{1}{L}\sum_{i=1}^{L} f_i = \sum_{j} jp_j > 2$
  - Con simboli fortemente casuali → $p_1=1$ → RLE non conviene

Lunghezza media di un run

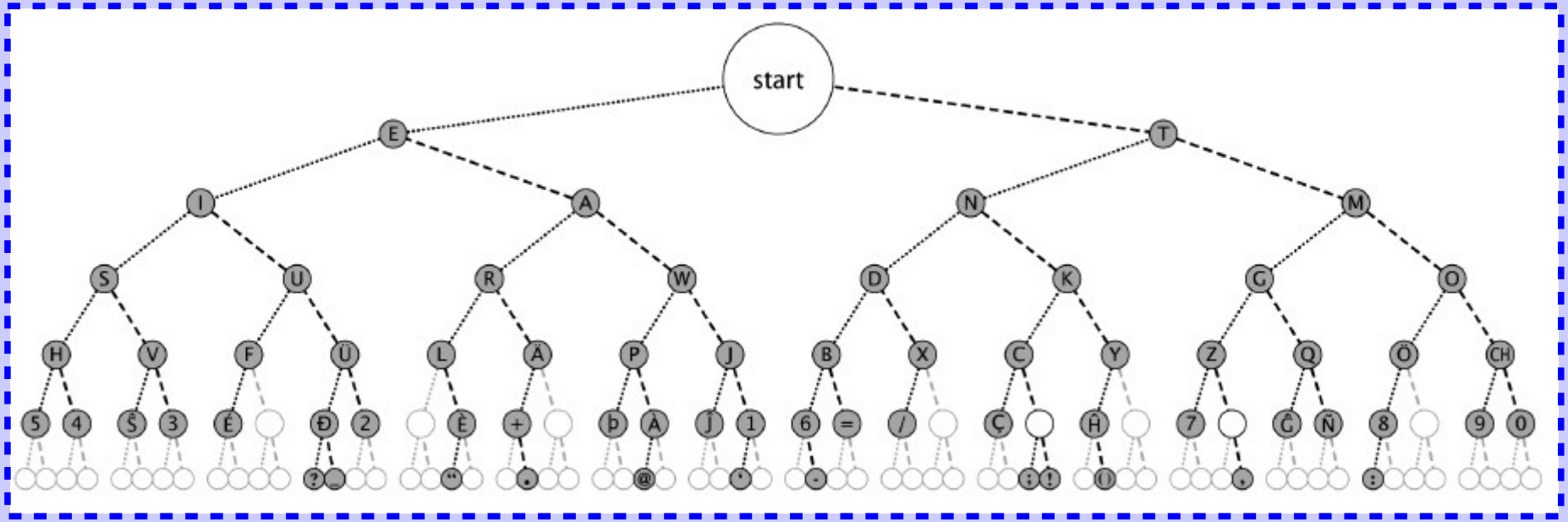
Valore atteso di lunghezza di un run

RLE – esempio tesina

- *Algoritmo di RLE: codifica e decodifica*
  - *Lunghezza fissa → singolo carattere*
  - *Lunghezza variabile → stringhe*
- *Implementazione di un programma di codifica e di un programma di decodifica*
- *Test di codifica e decodifica con diversi testi*
  - *Verifica coincidenza tra file originale e file decompresso*
  - *Calcolo prestazioni (rapporto compressione = dimensione file originale / dimensione file compresso, complessità computazionale, ...)*
  - *Tabella risultati*
  - *Tabella eventuali confronti tra diversi algoritmi*
- *Slides di presentazione del lavoro*

Testo – HUFFMAN STATICO

- **CODICE A LUNGHEZZA VARIABILE**
- **LOSSLESS**
- **Codificare i simboli più frequenti con codici di lunghezza minore**
- Decodifica semplice, a bassa complessità, tramite un **albero binario** con simboli associati alle **foglie**
  - codice con proprietà di **PREFISSO** → nessuna parola di codice costituisce l'inizio (prefisso) di altre parole di codice, capisco subito quando un simbolo è finito, non devo attendere quello che segue.
- **CODICE MORSE**: esempio di codice a lunghezza variabile senza proprietà di prefisso → simboli associati sia a foglie che a nodi



Testo – HUFFMAN STATICO
Bit per simbolo

- Per codici a lunghezza variabile
- R = Numero medio di bit necessario a rappresentare un simbolo

$$R = \sum_{j=1}^{n} m_j P(x_j)$$

- Codice ottimo → minimizza R
- Con codici che coincidono con le foglie di alberi binari, la condizione che consente di minimizzare R è la DISUGUAGLIANZA DI KRAFT
  - Nelle ipotesi m1 <= m2 <= m3 <= … <= mn la condizione NECESSARIA e SUFFICIENTE per costruire un albero binario decodificabile seguendo i prefissi è
  - $$\sum_{j=1}^{n} 2^{-m_j} \leq 1$$
  - DIMOSTRAZIONE → FARE
- Il limite inferiore per R è dato dall'Entropia, secondo il Teorema di SHANNON
  - $$H(X) \leq R < H(X) + 1$$

Testo – HUFFMAN STATICO
Bit per simbolo

$$R = \sum_{j=1}^{n} m_j P(x_j)$$

Teorema di Shannon - *dimostrazione*

$$
\begin{aligned}
H(X) - R &= \sum_j P(x_j)[-\log_2 P(x_j) - m_j] = \sum_j P(x_j) \log_2 \frac{2^{-m_j}}{P(x_j)} \\
&\leq (\log_2 e) \sum_j P(x_j) \left[\frac{2^{-m_j}}{P(x_j)} - 1 \right] = (\log_2 e) \left\{ \left[\sum_j 2^{-m_j} \right] - 1 \right\} \\
&\leq 0
\end{aligned}
$$

ln x <= x-1

Cambio base

$\log_2 x / \log_2 e <= x-1$

Disuguaglianza di Kraft

L'uguaglianza vale solo in caso di $P(x_j) = 2^{-m_j}$

Teorema di Shannon - *dimostrazione*

$$
\begin{aligned}
H(X) - R &= \sum_j P(x_j)[-\log_2 P(x_j) - m_j] = \sum_j P(x_j) \log_2 \frac{2^{-m_j}}{P(x_j)} \\
&\leq (\log_2 e) \sum_j P(x_j) \left[\frac{2^{-m_j}}{P(x_j)} - 1 \right] = (\log_2 e) \left\{ \left[\sum_j 2^{-m_j} \right] - 1 \right\} \\
&\leq 0
\end{aligned}
$$

ln x <= x-1

Cambio base

$\log_2 x / \log_2 e <= x-1$

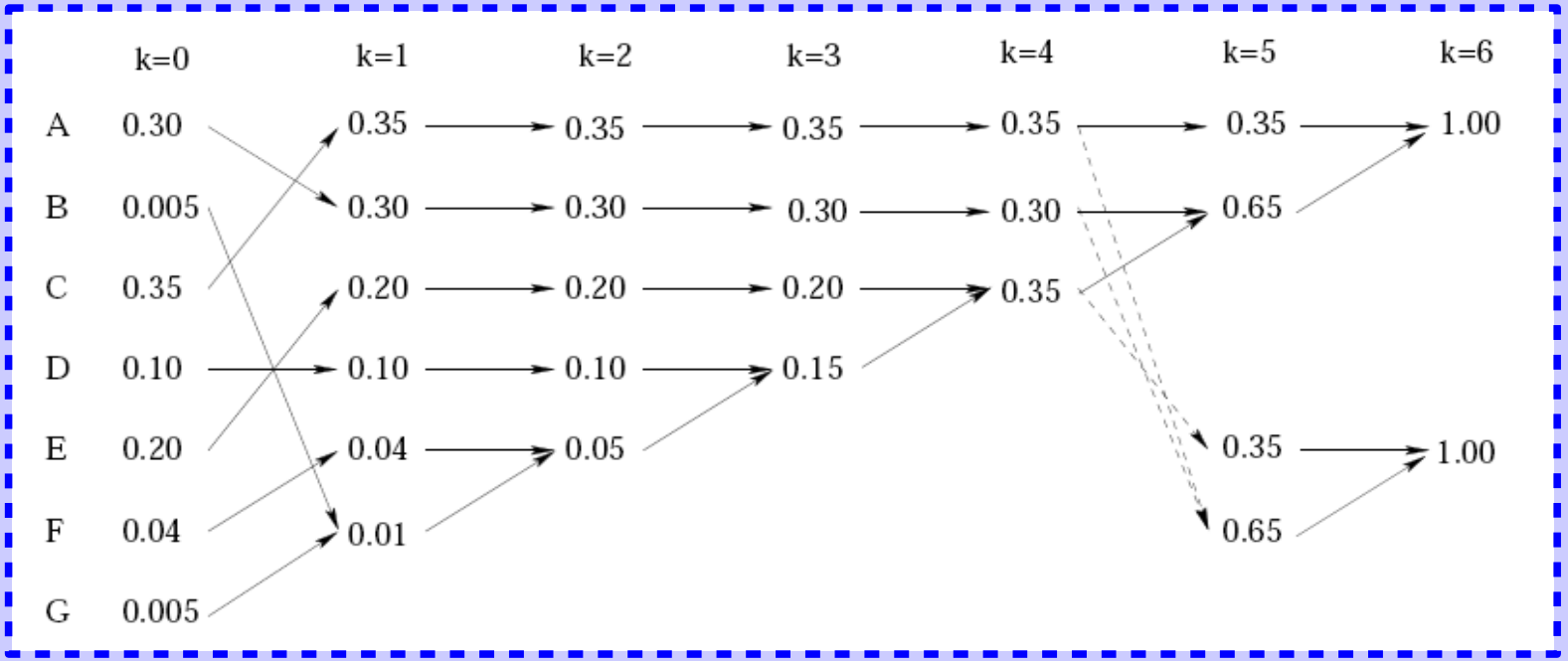
Disuguaglianza di Kraft

L'uguaglianza vale solo in caso di $P(x_j) = 2^{-m_j}$

Testo – HUFFMAN STATICO
Algoritmo

- Conoscere **frequenze** dei simboli (probabilità emissione dei simboli)
  - **Offline** con distribuzioni generiche (es: lettere lingua italiana, inglese...)
    - Veloce ma potenziale discordanza tra statistica assunta e statistica effettiva
  - **Ad hoc** sul testo in esame
    - Efficacia della statistica effettiva ma necessità di operazione preliminare di parsing e di disponibilità dell'intero testo
- **Somma progressiva** dei **due elementi a minore probabilità**, fino ad arrivare alla radice dell'albero (probabilità 1)
- **Associazione dei codici** **0 e 1** a partire dalla radice, andando verso le foglie.
  - A ogni biforcazione si associa 1 al ramo con probabilità maggiore e 0 all altro o viceversa.
    - Due codifiche diverse ma a stesso R
- **Memorizzazione codici**: manipolazione stream di bit con codici a lunghezza variabile, indipendenti da granularità di memorizzazione (byte = 8 bit)
- **Memorizzazione albero**: overhead

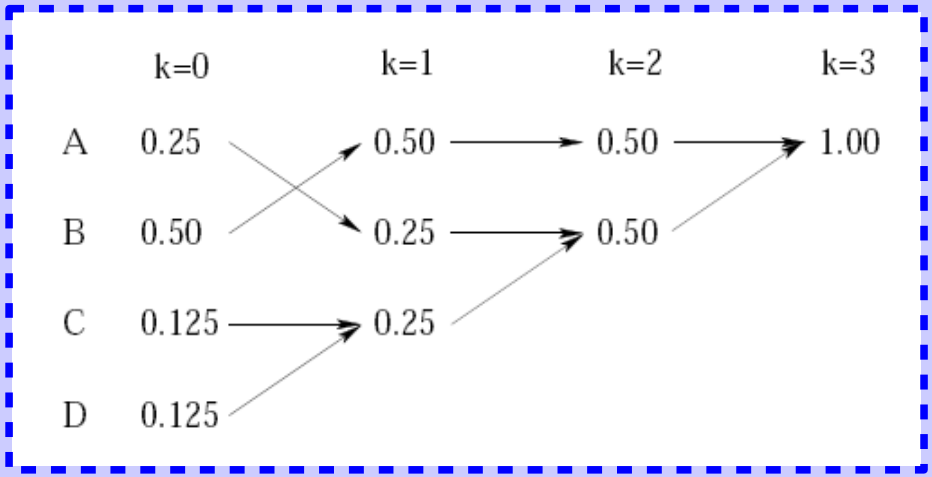
Testo – HUFFMAN STATICO
Esempi



H(X) = 2.11

R = 2.21

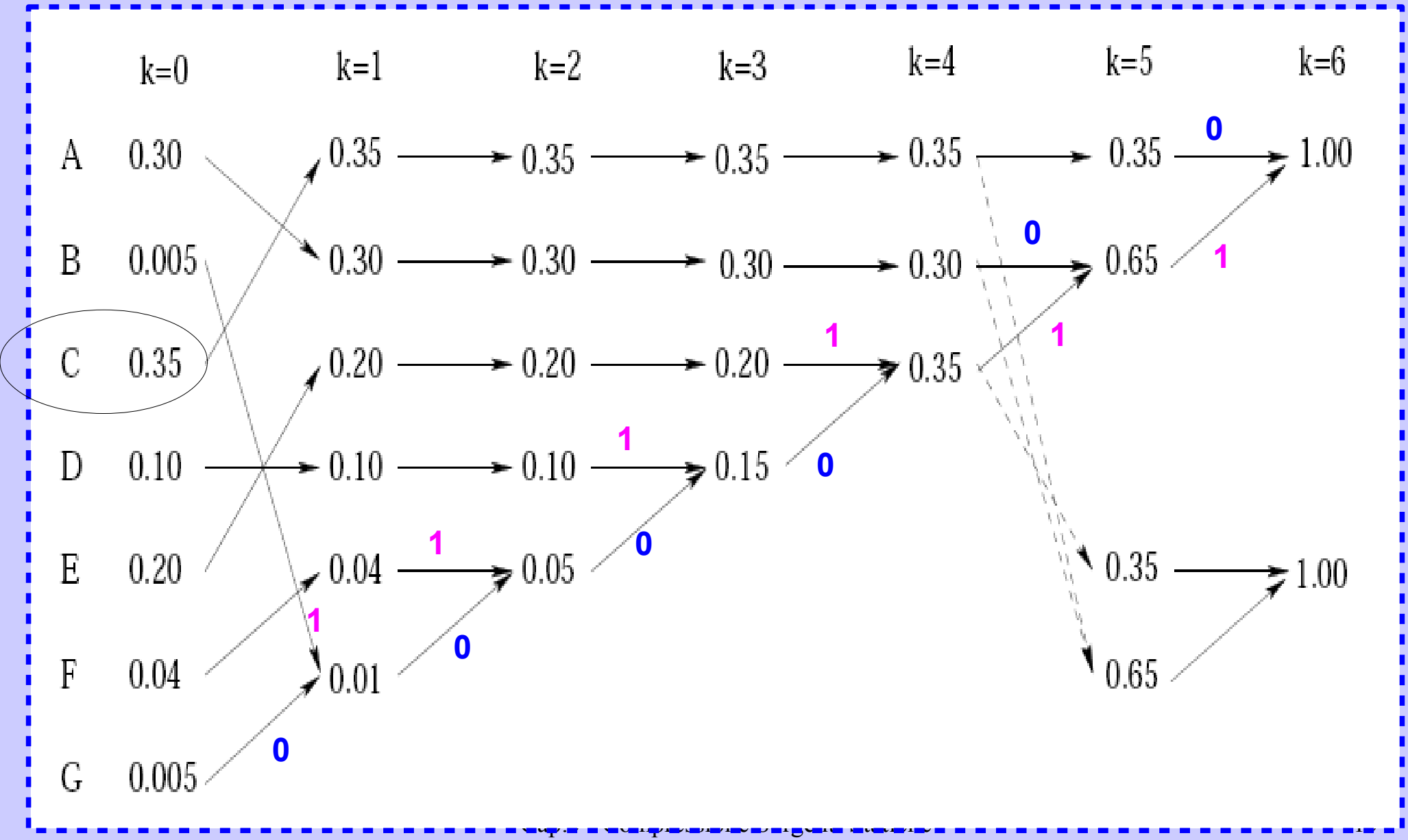
H(X) <= R < H(X)+1
2.11 <= 2.21 < 3.11



Probabilità di emissione dei simboli espresse come potenze di 2

H(X) = R = 1.75

Testo – HUFFMAN STATICO
Esempi



C=0, A=10, E=111, D=1101, F=11001, B=110001, G=110000

Testo – HUFFMAN STATICO
Esempi

C=0, A=10, E=111, D=1101, F=11001, B=110001, G=110000

Versione compressa
0010111011100011001.....

Decompressore
0 → C
0 → C
10 → A
111 → E
0 → C
111 → E
0 → C
0 → C
11001 → F
...

Testo – HUFFMAN STATICO
Esempio tesine

- *Algoritmo di Huffman statico: codificatore e decodificatore*
- *Implementazione codificatore e decodificatore*
- *Valutazione prestazioni con tabelle di confronto*
  - *Rapporto compressione (dimensione testo originale / dimensione testo compresso)*
  - *Carico computazionale*
    - *Testo compresso compreso od escluso overhead*
    - *Overhead albero o lista simboli e loro frequenza*
    - *Frequenze calcolate ad hoc o statistiche offline*
    - *Diversi testi di diverse lunghezze e caratteristiche*
  - *Confronti tra diversi algoritmi o diverse implementazioni*
- *Slides di presentazione lavoro*

Testo – HUFFMAN DINAMICO

- Serve quando **non si ha conoscenza dell'intero testo o della statistica** dei suoi simboli ma si hanno **tempi stringenti** di consegna dell'informazione per cui non si può attendere di collezionare l'intero testo
- **Si procede creando dinamicamente l'albero di codifica, contestualmente alla codifica:**
  - Ogni simbolo già presente nell'albero viene inviato con la sua attuale **codifica**
  - Ogni simbolo nuovo, non ancora presente nell'albero, viene inviato in **chiaro**
  - All'emissione di ogni simbolo cambia la statistica dei simboli emessi dalla sorgente e si procede costruendo un nuovo **albero**, sia lato codificatore che lato decodificatore

Testo – HUFFMAN DINAMICO
Algoritmo

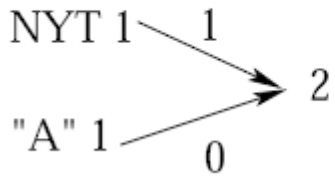
- Si accumulano le occorrenze dei simboli, fino alla generazione **p-esima**
- Per ogni simbolo generato si costruisce un **nuovo albero**, sia al ***codificatore*** che al ***decodificatore***, seguendo l'**algoritmo di Huffman statico** sulla base delle frequenze di emissione ottenute fino a quel momento
- Simbolo speciale, **Not Yet Transmitted (NYT)**
  - Occorrenza = 1 statica
  - Codice di escape per trasmettere un carattere emesso per la prima volta dalla sorgente: indica che gli 8 bit successivi sono un carattere espresso con codifica ASCII, in chiaro
- **Operazioni in ordine cronologico:**
  - Un simbolo già presente nell'albero viene emesso con la sua ATTUALE codifica.
  - Un simbolo nuovo viene emesso con NYT e codifica ASCII.
  - Vengono aggiornate le occorrenze dei simboli.
  - Viene aggiornato l'albero da entrambi i lati codificatore e decodificatore

Testo – HUFFMAN DINAMICO
terminologia

- Sorgente può emettere n possibili simboli → xj, j=1,...,n
  - Es: n=3: j=1 → x_1=A, j=2 → x_2=B, j=3 → x_3=C
- La sorgente emette una sequenza di S simboli, indice di generazione p, p=1, …, S
  - Es: S=7 → AABACCB → p=1, …, 7
- Indice del simbolo che la sorgente genera al passo p-esimo → gp
  - Es: g_1=1, g_2=1, g_3=2, g_4=1, g_5=3, g_6=3, g_7=2
- Vettore delle occorrenze al passo p-esimo → $o_{j,p}$, numero di volte che il simbolo j-esimo è stato emesso dalla sorgente dopo p passi, ossia dopo che la sorgente ha emesso p simboli. Si aggiorna come $o_{j,p} = o_{j,p-1} + 1$
  - Es: p=0 → o_1=0, o_2=0, o_3=0 → [0,0,0]
  - Al passo 1, p=1 → x_{g1}=A → [1,0,0]
  - Al passo 2, p=2 → x_{g2}=A → [2,0,0]
  - Al passo 3, p=3 → x_{g3}=B → [2,1,0]
  - Al passo 4, p=4 → x_{g4}=A → [3,1,0]
  - Al passo 5, p=5 → x_{g5}=C → [3,1,1]

Testo – HUFFMAN DINAMICO
Esempio → AA BBB C

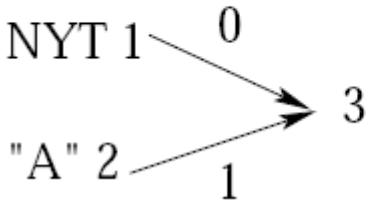
p=1 x_g_1="A" TX: ASCII("A")



NYT 1
"A" 0

Numero di bit trasmessi = 8

p=2 x_g_2="A" TX: 0

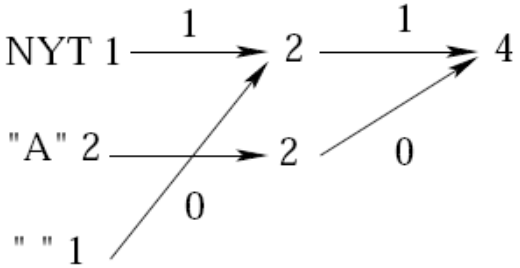


NYT 0
"A" 1

Numero di bit trasmessi = 9

Il codice 1 viene assegnato, per convenzione, al ramo con Numero di occorrenze MAGGIORE o, in caso di parità di numero di occorrenze, al ramo corrispondente al simbolo più VECCHIO

p=3 x_g_3=" " TX: 0 ASCII(" ")

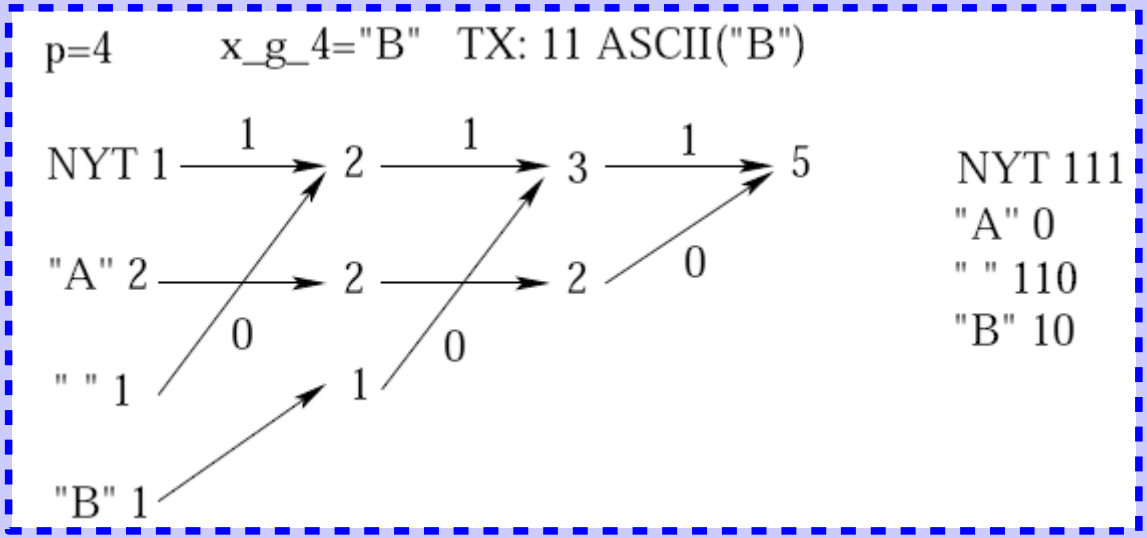


NYT 11
"A" 0
" " 10

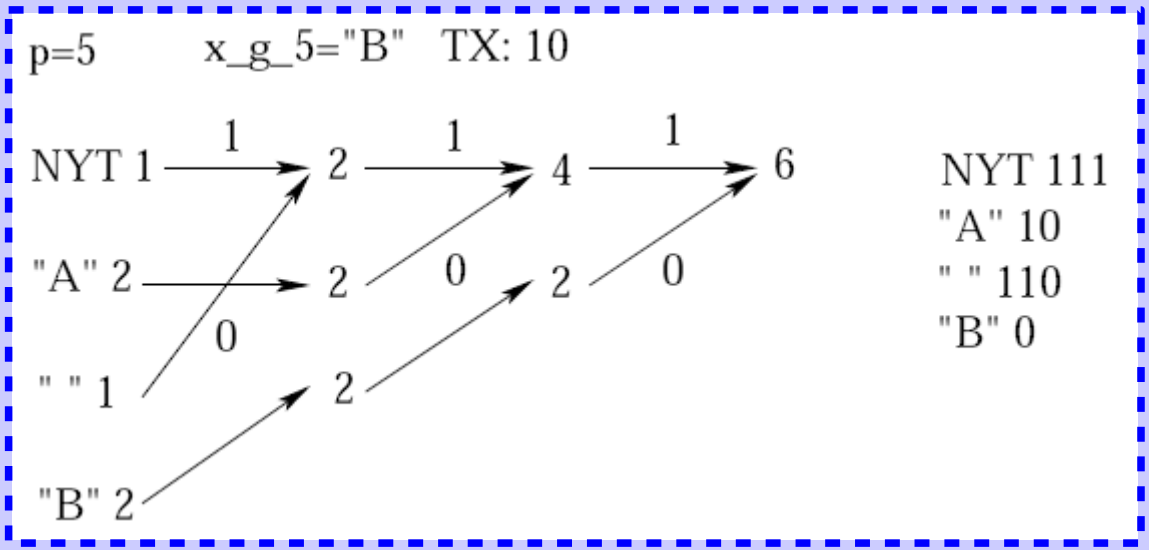
Numero di bit trasmessi = 18

Quando si devono fondere insieme rami con stesso numero di occorrenze si scelgono quelli corrispondenti a simboli più vecchi

Testo – HUFFMAN DINAMICO
Esempio → AA BBB C

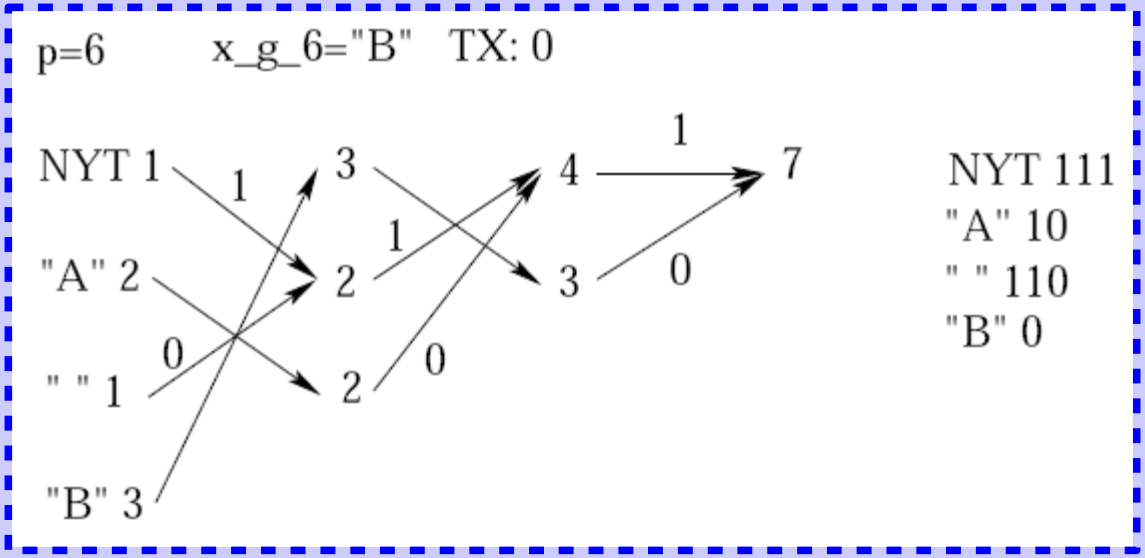


Numero di bit trasmessi = 28

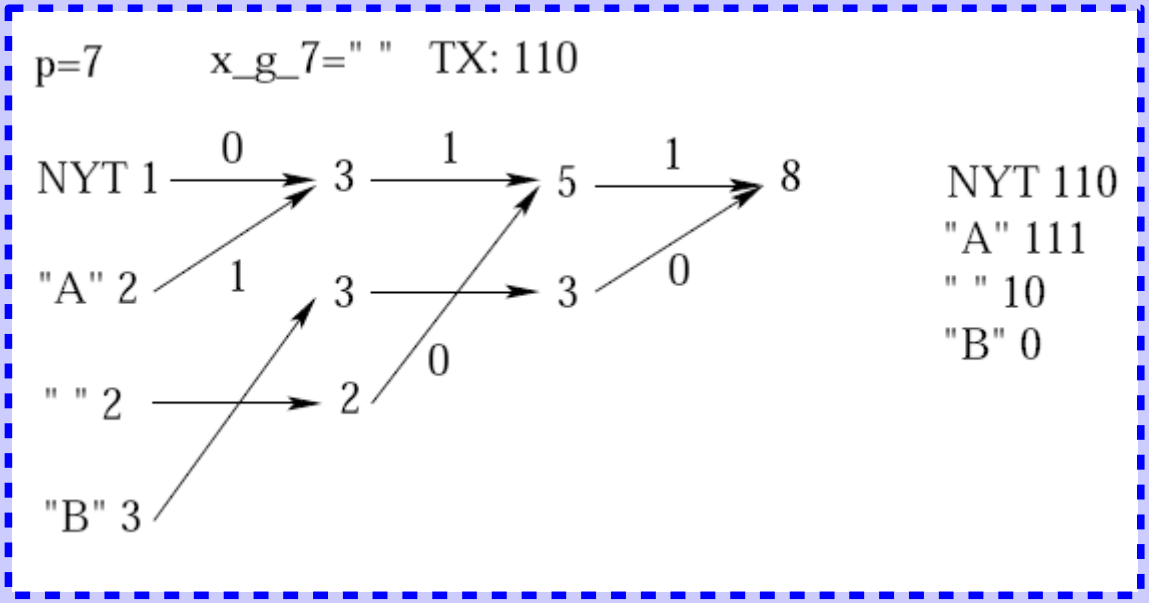


Numero di bit trasmessi = 30

Testo – HUFFMAN DINAMICO
Esempio → AA BBB C

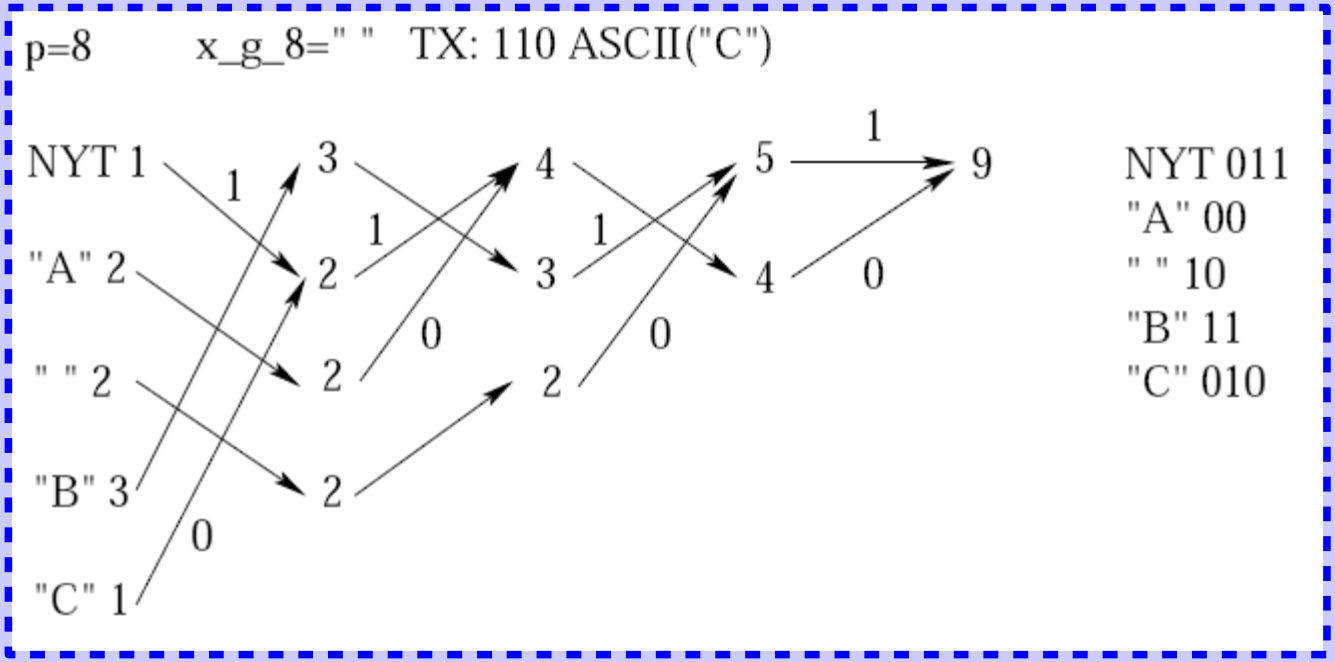


Numero di bit trasmessi = 31



Numero di bit trasmessi = 34

Testo – HUFFMAN DINAMICO
Esempio → AA BBB C



Numero di bit trasmessi = 45

R = 2.22 =

(1*3+2*2+2*2+3*2+1*3)/9

- **Confronto prestazioni**
  - **Non compresso**, ASCII → 64 bit
  - **Huffman statico** → 17 + 40(overhead albero, 4 caratteri in ascii, 4*8=32 + 2 bit per ogni carattere per rappresentare il numero di occorrenze, 4*2=8, 32+8=40) = 57 bit
  - **Huffman dinamico** → 45 bit

Testo – CODIFICA ARITMETICA

- LOSSLESS
- Messaggio codificato → numero reale nell'intervallo [0, 1[
- Rappresentare i simboli della sorgente con un insieme di sottointervalli, ognuno di dimensione proporzionale alla probabilità P(xj) legata al j-esimo simbolo dell'alfabeto.
- Ad ogni simbolo generato dalla sorgente,viene selezionato dall'intervallo attuale il sottointervallo relativo a quel particolare simbolo ed utilizzato come intervallo per la successiva divisione in sottointervalli proporzionali alle probabilità.
- La codifica del messaggio avviene restringendo sempre di più l'intervallo. Quanto più un intervallo è piccolo, tanti più sono i bit necessari a rappresentare un numero al suo interno.
- Simbolo terminatore TER, per specificare al decompressore il termine del messaggio.
- R = H(X)
  - R=#bit / #simboli
  - #bit per rappresentare
  - sottointervallo [0,1[
  - di dimensione s = -log2s
  - s = prodotto delle probabilità dei simboli usciti → log del prodotto = somma dei log

$$-\frac{1}{p}\log_2 s = -\frac{1}{p}\sum_{j=1}^{p}\log_2 P(x_{g_j}) = -\frac{1}{p}\sum_{j=1}^{p}\log_2 P(x_j)^{o_j}$$
$$= -\sum_{j=1}^{p} P(x_j)\log_2 P(x_j) = H(X)$$

Testo – CODIFICA ARITMETICA

- LOSSLESS
- Messaggio codificato → numero reale nell'intervallo [0, 1[
- Rappresentare i simboli della sorgente con un insieme di sottointervalli, ognuno di dimensione proporzionale alla probabilità P(xj) legata al j-esimo simbolo dell'alfabeto.
- Ad ogni simbolo generato dalla sorgente,viene selezionato dall'intervallo attuale il sottointervallo relativo a quel particolare simbolo ed utilizzato come intervallo per la successiva divisione in sottointervalli proporzionali alle probabilità.
- La codifica del messaggio avviene restringendo sempre di più l'intervallo. Quanto più un intervallo è piccolo, tanti più sono i bit necessari a rappresentare un numero al suo interno.
- Simbolo terminatore TER, per specificare al decompressore il termine del messaggio.
- R = H(X)
  - R=#bit / #simboli
  - #bit per rappresentare
  - sottointervallo [0,1[
  - di dimensione s = -log2s
  - s = prodotto delle probabilità dei simboli usciti → log del prodotto = somma dei log

Testo – CODIFICA ARITMETICA

$$s = \prod_{r=1}^{p} P(x_{g_r})$$

Logaritmo del prodotto = somma dei logaritmi

$$-\frac{1}{p}\log_2 s = -\frac{1}{p}\sum_{j=1}^{p}\log_2 P(x_{g_j}) = -\frac{1}{p}\sum_{j=1}^{p}\log_2 P(x_j)^{o_j}$$

$$= -\sum_{j=1}^{p} P(x_j)\log_2 P(x_j) = H(X)$$

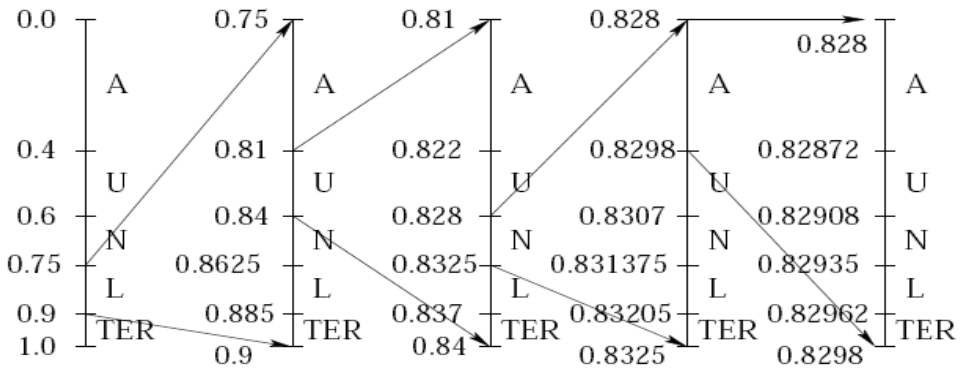
Testo – CODIFICA ARITMETICA

- ESEMPIO

  s=0.25 → 4 intervalli → 2=$\log_2 4$ bit per identificare un intervallo tra i 4 disponibili

- ESEMPIO “LUNA”
  - L → 0.15
  - U → 0.2
  - L*U → 0.15*0.2 = 0.03 = 0.84 – 0.81

A titolo di esempio si riporta la codifica aritmetica della stringa LUNA considerando una sorgente con $x_1 = A$, $x_2 = U$, $x_3 = N$, $x_4 = L$, $x_5 = TER$ e $P(x_1) = 0.4$, $P(x_2) = 0.2$, $P(x_3) = 0.15$, $P(x_4) = 0.15$, $P(x_5) = 0.1$.



Al quinto passo si raggiunge il terminatore e l’intervallo selezionato risulta $[0.82962, 0.8298[$, quindi la stringa di bit da utilizzare deve codificare un qualsiasi numero reale all’interno di questo intervallo.

Testo – CODIFICA ARITMETICA
Algoritmo

- Definizione delle probabilità cumulative

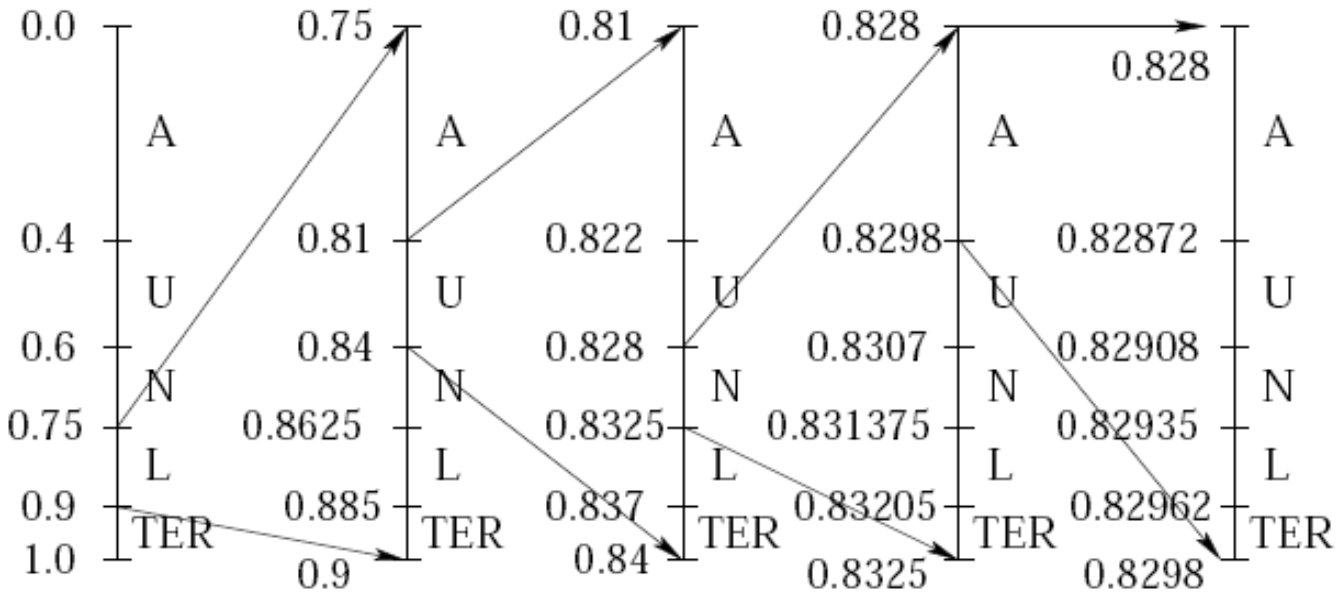
$$u_0 = 0,\ u_j = u_{j-1} + P(x_j)$$

- Definizione degli intervalli secondo un rescaling affine:
  - Passo p=0
    - $A_0=0$, $B_0=1$
    - Suddivisione intervalli $[x_{j-1}, x_j[$
  - Generico passo p (g_p simbolo emesso al passo p)
    - Selezionato intervallo $[x_{gp-1}, x_{gp}[$

$$\begin{aligned} A_p &= (B_{p-1} - A_{p-1})u_{g_p-1} + A_{p-1} \\ B_p &= (B_{p-1} - A_{p-1})u_{g_p} + A_{p-1} \\ C_p &\in [A_p, B_p] \end{aligned}$$

Testo – CODIFICA ARITMETICA
Esempio

A titolo di esempio si riporta la codifica aritmetica della stringa LUNA considerando una sorgente con $x_1 = A$, $x_2 = U$, $x_3 = N$, $x_4 = L$, $x_5 = TER$ e $P(x_1) = 0.4$, $P(x_2) = 0.2$, $P(x_3) = 0.15$, $P(x_4) = 0.15$, $P(x_5) = 0.1$.



Al quinto passo si raggiunge il terminatore e l'intervallo selezionato risulta $[0.82962, 0.8298[$, quindi la stringa di bit da utilizzare deve codificare un qualsiasi numero reale all'interno di questo intervallo.

Testo – Lempel Ziv (LZ)

- **LOSSLESS**
- Vengono codificate **STRINGHE**, definite **dinamicamente** tramite parsing degli elementi emessi dalla sorgente
- Due categorie di codificatori
  - **Puntatore**
    - Se una stringa è già stata emessa, viene inviato il puntatore alla posizione di tale stringa e l'indicazione della sua lunghezza
    - **LZ77** e **LZSS** (Storer, Szymansky)
  - **Dizionario**
    - Creare dizionario delle stringhe emesse dalla sorgente ed inviare l'indice relativo alla entry di tale stringa nel dizionario
    - **LZ78** e **LZW** (Welch)

Testo – LZ77

1. Poni la posizione di codifica all'inizio dello stream di input

2. Cerca la stringa più lunga presente nei simboli da processare tale che sia presente nei simboli già processati.

3. Eroga in uscita la coppia (P, S), dove P è il puntatore relativo all'output dei simboli già processati corrispondente alla stringa e S è il simbolo immediatamente successivo alla stringa identificata.

4. Se vi sono ancora dei simboli da codificare, posizionati al carattere immediatamente successivo all'ultimo carattere considerato e procedi con il passo 2.

Numero Simbolo	Simbolo	Note	Output
#1	A	primo simbolo, nessun match	(0,0,A)
#2	A	match #1	(1,1,B)
#3	B	già trasmesso passo precedente	-
#4	C	prima occorrenza di C	(0,0,C)
#5	B	match #3	(2,1,B)
#6	B	già trasmesso passo precedente	-
#7 #8 #9	ABC	match #2 #3 #4	(5,3,-)

Esempio:
AABCBBABC

Output = (P,S) = (B,L,S)
Puntatore = (B,L)
B = #elementi di cui tornare indietro
L = #simboli della stringa

Testo – LZSS

- Il puntatore viene erogato in uscita solo se la lunghezza della stringa trovata è maggiore della lunghezza del puntatore → **L => L_P**
- Altrimenti viene inviata la stringa in chiaro
- **L_P = 2** perchè P=(B,L)
- FLAG per distinguere stringhe da puntatori

Numero Simbolo	Simbolo	Note	Output
#1	A	primo simbolo, nessun match	A
#2	A	match #1, conviene simbolo	A
#3	B	prima occorrenza di B	B
#4	C	prima occorrenza di C	C
#5	B	match #3, conviene simbolo	B
#6	B	match #3, conviene simbolo	B
#7 #8 #9	ABC	match #2 #3 #4, si usa puntatore	(5,3)

Esempio:
AABCBBABC

Output = P o S

Testo – LZ78

1. Considera un dizionario vuoto ed un prefisso P nullo.

2. Considera il prossimo simbolo C dello stream di ingresso.

3. Se P + C è presente nel dizionario procedi con P = P + C; se vi sono ancora simboli da processare procedi con il passo 2. Altrimenti genera la coppia (W,C) dove W è l'indice del prefisso P nel dizionario; aggiungi P + C al dizionario; considera un prefisso P nullo; se vi sono ancora dei simboli da processare procedi con il passo 2.

4. Se P non è nullo, genera in uscita il W corrispondente al P attuale.

Indice	Stringa
1	A
2	AB
3	C
4	B
5	BA
6	BC

Numero Simbolo	Simbolo	Output
#1	A	(0,A)
#2 #3	AB	(1,B)
#4	C	(0,C)
#5	B	(0,B)
#6 #7	BA	(4,A)
#8 #9	BC	(4,C)

Esempio:
AABCBBABC

Output = (W,C)

Testo – LZ78

Esempio:
AABCBBABC

Output = (W,C)v

Indice	Stringa
1	A
2	AB
3	C
4	B
5	BA
6	BC

Numero Simbolo	Simbolo	Output
#1	A	(0,A)
#2 #3	AB	(1,B)
#4	C	(0,C)
#5	B	(0,B)
#6 #7	BA	(4,A)
#8 #9	BC	(4,C)

P=0
C=A
P+C=A
Inserisco A in diz
Output (0,A)

P=0
C=A
P+C=A
P=A
P+C=AB
Inserisco AB in diz
Output (1,B)

P=0
C=C
P+C=C
Inserisco C in diz
Output (0,C)

P=0
C=B
P+C=B
Inserisco B in diz
Output (0,B)

P=0
C=B
P+C=B
P=B
P+C=BA
Inserisco BA in diz
Output (4,A)

P=0
C=B
P+C=B
P=B
P+C=BC
Inserisco BC in diz
Output (4,C)

Testo – LZW

1. Considera un dizionario contenente tutti i simboli singoli appartenenti alla sorgente ed un prefisso P nullo.

2. Considera il prossimo simbolo C dello stream di ingresso.

3. Se P + C é presente nel dizionario procedi con P = P + C; se vi sono ancora simboli da processare procedi con il passo 2. Altrimenti genera l'indice W del prefisso P; aggiungi P + C al dizionario; considera un prefisso P = C; se vi sono ancora dei simboli da processare procedi con il passo 2.

4. Genera in uscita il W corrispondente al P attuale.

Indice	Stringa
1	A
2	B
3	C
4	AA
5	AB
6	BC
7	CB
8	BB
9	BA
10	ABC

Numero Simbolo	Stringa	Output
#1 #2	AA	(1)
#2 #3	AB	(1)
#3 #4	BC	(2)
#4 #5	CB	(3)
#5 #6	BB	(2)
#6 #7	BA	(2)
#7 #8 #9	ABC	(5)
#9	C	(3)

Esempio:
AABCBBABC

Output = W

Testo – LZW

Esempio:
AABCBBABC

Output = W

Numero Simbolo	Stringa	Output
#1 #2	AA	(1)
#2 #3	AB	(1)
#3 #4	BC	(2)
#4 #5	CB	(3)
#5 #6	BB	(2)
#6 #7	BA	(2)
#7 #8 #9	ABC	(5)
#9	C	(3)

Indice	Stringa
1	A
2	B
3	C
4	AA
5	AB
6	BC
7	CB
8	BB
9	BA
10	ABC

P=0
C=A
P+C=A
P=A
P+C=AA
Inserisco AA in diz
Output (1)

P=A
P+C=AB
Inserisco AB in diz
Output (1)

P=B
C=C
P+C=BC
Inserisco BC in diz
Output (2)

P=C
C=B
P+C=CB
Inserisco CB in diz
Output (3)

P=B
C=B
P+C=BB
Inserisco BB in diz
Output (2)

P=B
C=A
P+C=BA
Inserisco BA in diz
Output (2)

P=A
C=B
P+C=AB
P=AB
P+C=ABC
Inserisco ABC in diz
Output (5)