Relazione Codifica Huffman

AUTORE: Filippo Manini
MATRICOLA: 4798004

TESTO:
Ogni carattere ASCII occupa in memoria un byte. Fissa un file in input che consiste di almeno 10^5 caratteri (spazi bianchi inclusi). Supponiamo che il file contenga M caratteri diversi. Poni la frequenza empirica del carattere x_i del file uguale alla probabilità p_i e calcola l'entropia di Shannon H (X) associata con X = {x_1 , . . . , x_M }. Implementa una codifica di Huffman C per l'alfabeto X e confronta la lunghezza attesa L(C, X) con H(X). Comprimi il testo usando la codifica e valuta la compressione assumendo che, nella codifica di Huffman, ogni 0 o 1 sia immagazzinato in un bit.

ESECUZIONE:
g++ -std=c++17 huffmanCoding.cpp

FUNZIONAMENTO:
Complessità temporale: O (n log n) dove n è il numero di caratteri univoci.
Per prima cosa viene chiesto il nome del file di cui si farà la codifica di Huffman e viene letto tramite la funzione: *letturaFile(…)*

```
Digita il nome del file: pg.txt
```

ENTROPIA DI SHANNON:
definita come, valore atteso dell'informazione di Shannon

$$H(X) = \sum_{i=1}^{N} p_X(x_i) \log_2 \frac{1}{p_X(x_i)},$$

questo dato viene calcolato tramite la funzione: *entropiaShannon(…)*

```
entropia di Shannon H(X) = 4.45612
```

Vediamo ora i passaggi per la costruzione dell'Albero di ***Huffman:***

1. Creazione di un nodo foglia per ogni carattere univoco e costruzione di un heap minimo di tutti i nodi foglia (Min Heap è usato come coda di priorità. Il valore del campo frequenza è usato per confrontare due nodi in un heap minimo. Inizialmente, il carattere meno frequente è a radice)
2. Estrai due nodi con la frequenza minima dall'heap min.
3. Creare un nuovo nodo interno con una frequenza uguale alla somma delle frequenze dei due nodi.
4. Imposta il primo nodo estratto come figlio sinistro e l'altro nodo estratto come figlio destro.
5. Aggiungi questo nodo all'heap min.
6. Si ripetono i passaggi precedenti finché l'heap non contiene un solo nodo.
7. Il nodo rimanente è il nodo radice e l'albero è completo.

questo processo viene eseguito tramite la funzione *huffmanCod(…)*

LUNGHEZZA ATTESA:
La ***lunghezza attesa*** L(**C**, **X**) di una codifica C è
(somma della probabilità del carattere per la lunghezza della sua codifica)
Dove **C**: codifica **X**: insieme di caratteri

$$L(C, \mathcal{X}) = \sum_{x \in \mathcal{X}} p(x) L_C(x) = \sum_{i=1,\ldots,|\mathcal{X}|} p_i L_i.$$

```
lunghezza attesa = 4.4927
```

osserviamo che la codifica risulta interessante in quanto la lunghezza attesa si avvicina al valore dell'entropia

Il programma visualizza:

Stampa dell'albero:

t	5014
l	5117
r	5191
v	2398
├	613
▓	161
E	170
?	172
Q	44
U	22
F	23
¼	90
	760
f	812
a	10822
n	5789
d	2929
,	1608
g	1628
╣	92
V	47
O	47
N	94
M	95
ö	446
Ô	907
Ç	907
h	909
c	3557
o	7889
i	7943
Ö	449
q	469
.	949
á	105

D	120
Z	5
H	7
ê	13
T	27
B	33
ô	3
j	2
'	1
x	1
ª	9
R	20
A	124
-	483
b	1036
m	2078
p	2170
	17120
s	4299
z	531
I	125
L	137
;	274
P	138
*	69
G	70
:	140
C	142
!	285
S	150
¿	152
u	2329
e	9296

potrebbe succedere che alcuni caratteri non vengano visualizzati correttamente quello è dovuto dal modo con cui li visualizziamo (ricordiamo però che uno è il carattere spazio e l'altro il carattere tab)

le prossime stampe permettono di visualizzare in ordine i caratteri e le loro informazioni grazie all'utilizzo di mappe C++ (classe che permette l'implementazione del tipo [chiave, valore])

frequenza dei caratteri:

CARATTERE	FREQUENZA
Ç	907
ê	13
ô	3
ö	446
Ö	449
á	105
ª	9
¿	152
¼	90
▓	161
╣	92
├	613
Ô	907
	760
	17120
!	285
'	1
*	69
,	1608
-	483
.	949
:	140
;	274
?	172
A	124
B	33
C	142
D	120
E	170
F	23
G	70
H	7
I	125
L	137
M	95

N	94
O	47
P	138
Q	44
R	20
S	150
T	27
U	22
V	47
Z	5
a	10822
b	1036
c	3557
d	2929
e	9296
f	812
g	1628
h	909
i	7943
j	2
l	5117
m	2078
n	5789
o	7889
p	2170
q	469
r	5191
s	4299
t	5014
u	2329
v	2398
x	1
z	531

codifica dei caratteri:

CARATTERE	CODIFICA
Ç	1000010
ê	1011010010001
ô	101101001011000
ö	10000001
Ö	10110000
á	1011010000
ª	10110100101101
¿	1110101111
¼	0011101111
▓	001110100
╣	1000000000
├	0011100
Ô	1000001
	0011110
	110
!	111010110
'	101101001011001l0
*	11101010010
,	011110
-	10110101
.	1011001
:	1110101010
;	111010011
?	001110110
A	1011010011
B	101101001010
C	1110101011
D	1011010001
E	001110101
F	001110111011
G	11101010011
H	10110100100001
I	1110100100
L	1110100101
M	1000000011

N	1000000010
O	10000000011
P	1110101000
Q	00111011100
R	1011010010111
S	1110101110
T	101101001001
U	001110111010
V	10000000010
Z	10110100100000
a	010
b	1011011
c	10001
d	01110
e	1111
f	0011111
g	011111
h	1000011
i	1010
j	10110100101100 10
l	0001
m	101110
n	0110
o	1001
p	101111
q	10110001
r	0010
s	11100
t	0000
u	111011
v	00110
x	101101001011001 11
z	11101000

Proprio per come funziona l’algoritmo possiamo notare che valori con frequenza più alta hanno una conversione in binario di lunghezza inferiore a caratteri con frequenza minore

Vediamo ora i passaggi per la **decodifica**:

1. Iniziamo dalla radice e continuo fino a trovare una foglia.
2. Se il bit corrente è 0, ci spostiamo sul nodo sinistro dell'albero.
3. Se il bit è 1, ci spostiamo sul nodo destro dell'albero.
4. Se durante l'attraversamento incontriamo un nodo foglia, stampiamo il carattere di quel particolare nodo foglia e poi continuiamo di nuovo l'iterazione dei dati codificati a partire dal passaggio 1.

questo processo viene eseguito tramite la funzione *decodifica(…)*

```
dimensione in bit del testo in input: 877256 bit
dimensione in bit dell'output (codifica): 492589 bit
```

come si evince dalle stampe sopra riportate otteniamo un grande risparmio di spazio dell’output codificato