LZ78 encode

Submit Solution

Nel file `lz78encode.cpp` si implementi la funzione corrispondente alla seguente dichiarazione:

```
1. bool lz78encode(const std::string& input_filename, const std::string& output_filename, int maxbits);
```

La funzione deve aprire in modalità binaria il file il cui nome è passato nel parametro `input_filename` e salvarlo nel file binario il cui nome è passato nel parametro `output filename`, comprimendolo con l'algoritmo LZ78.

Il parametro `maxbits` deve essere un numero da 1 a 30 e indica il numero massimo di bit da utilizzare per salvare le posizioni e quindi anche la dimensione massima del dizionario. Se il dizionario raggiunge la dimensione massima, deve essere svuotato e si ricomincia a comprimere da dove si è arrivati, con un dizionario vuoto.

La funzione ritorna `false` in caso di errore, `true` altrimenti.

Il file di output è codificato con un magic number `LZ78` (4 byte), poi un valore intero a 5 bit senza segno che indica il massimo numero di bit utilizzati per il dizionario durante la codifica (`maxbits`) e poi il file codificato con LZ78. Le posizioni del dizionario vengono codificate con un numero binario codificato con tanti bit quanti quelli necessari a codificare il più grande valore presente nel dizionario in quel momento (ricordate che 0 significa stringa vuota e che le stringhe aggiunte partono da 1), mentre i byte aggiuntivi sono codificati con il loro valore a 8 bit.

Quando il dizionario è vuoto, non devo usare bit, dato che sicuramente la coppia sarà (0,...), poi:

- una volta inserito nel dizionario l'elemento di indice 1 dovrò usare 1 bit
- una volta inserito nel dizionario l'elemento di indice 2 dovrò usare 2 bit
- una volta inserito nel dizionario l'elemento di indice 4 dovrò usare 3 bit
- una volta inserito nel dizionario l'elemento di indice 8 dovrò usare 4 bit
- una volta inserito nel dizionario l'elemento di indice 16 dovrò usare 5 bit

e così via.

Ad esempio sia dato il seguente file:

```
1. aabaacabcabcbaa
```

Se lo codificassimo con `maxbits=4`, l'LZ78 rappresenterebbe questo file con le coppie

```
1. (0,a) (1,b) (1,a) (0,c) (2,c) (5,b) (1,a)
```

Visto che la dimensione massima del dizionario è 16, l'indice massimo sarà 15, che non viene mai raggiunto e quindi non è necessario svuotare il dizionario. Pertanto codificheremo il file come:

```
1. 4C 5A 37 38    "LZ78"
2. 00100          4 (max bits)
3.                0 (codificato con 0 bit) ← Notate che il primo zero non viene neppure trasmesso
4. 01100001       'a'
5. 1              1 (codificato con 1 bit)
6. 01100010       'b'
7. 01             1 (codificato con 2 bit)
8. 01100001       'a'
9. 00             0 (codificato con 2 bit)
10. 01100011      'c'
11. 010           2 (codificato con 3 bit)
12. 01100011      'c'
13. 101           5 (codificato con 3 bit)
14. 01100010      'b'
15. 001           1 (codificato con 3 bit)
16. 01100001      'a'
17. 00000         padding per completare l'ultimo byte
```

Scrivo i valori di seguito:

```
4C 5A 37 38 00100 01100001 1 01100010 01 01100001 00 01100011 010 01100011 101 01100010 001 01100001 00000
```

Li raggruppo in ottetti:

```
4C 5A 37 38 00100011 00001101 10001001 01100001 00011000 11010011 00011101 01100010 00101100 00100000
```

Tutto in esadecimale:

```
4C 5A 37 38 23 0D 89 61 18 D3 1D 62 2C 20
```

Se invece lo codificassimo con `maxbits=2`, l'LZ78 rappresenterebbe questo file con le coppie

```
(0,a) (1,b) (1,a) (0,c) [svuoto] (0,a) (0,b) (0,c) (1,b) [svuoto] (0,c) (0,b) (0,a) (0,a)
```

Visto che la dimensione massima del dizionario è 4, l'indice massimo sarà 3, e questo mi obbliga a svuotare il dizionario per ben due volte. Pertanto codificheremo il file come:

```
4C 5A 37 38    "LZ78"
00010          2 (max bits)
               0 (codificato con 0 bit)
01100001       'a'
1              1 (codificato con 1 bit)
01100010       'b'
01             1 (codificato con 2 bit)
01100001       'a'
00             0 (codificato con 2 bit)
01100011       'c' ← a questo punto il dizionario viene svuotato
                0 (codificato con 0 bit)
01100001       'a'
0              0 (codificato con 1 bit)
01100010       'b'
00             0 (codificato con 2 bit)
01100011       'c'
01             1 (codificato con 2 bit)
01100010       'b' ← a questo punto il dizionario viene svuotato
               0 (codificato con 0 bit)
01100011       'c'
0              0 (codificato con 1 bit)
01100010       'b'
00             0 (codificato con 2 bit)
01100001       'a'
00             0 (codificato con 2 bit)
01100001       'a'
0000           padding per completare l'ultimo byte
```

Scrivo i valori di seguito:

```
4C 5A 37 38 00010 01100001 1 01100010 01 01100001 00 01100011 01100001 0 01100010 00 01100011 01 01100010 01100011 0
01100010 00 01100001 00 01100001
```

Li raggruppo in ottetti:

```
4C 5A 37 38 00010011 00001101 10001001 01100001 00011000 11011000 01001100 01000011 00011010 11000100 11000110
01100010 00011000 01000110 00010000
```

Tutto in esadecimale:

```
4C 5A 37 38 13 0D 89 61 18 D8 4C 43 1A C4 C6 62 18 46 10
```