## *Esercizio*

Il formato di immagini GIF (Graphics Interchange Format) è un vecchio formato per memorizzare immagini raster. Consideriamo la sua prima versione chiamata “GIF 87a” che non ammetteva ancora animazioni. Tutti i valori a più byte sono memorizzati in Little Endian.

Il formato comincia con una *GIF Signature* di 6 byte contenente i caratteri “GIF87a”.

Segue un header riferito a tutto il file chiamato *Screen Descriptor* contenente i seguenti campi:

| Offset# | Size | Purpose |
|---|---|---|
| 0 | 2 bytes | Screen width (larghezza della superifice) |
| 2 | 2 bytes | Screen height (altezza della superifice) |
| 4 | 1 byte | SD_Flags |
| 5 | 1 byte | L’indice, relativo alla global color map, del background color |
| 6 | 1 byte | Un byte contenente il valore zero |

Il byte SD_Flags è così suddiviso (il 7° bit indica il bit più significativo):

| Bit# | Purpose |
|---|---|
| 7 | M: se 1 indica che una color map globale segue lo *Screen Descriptor* |
| 6,5,4 | CR: il valore di CR+1 indica il numero di bit di color resolution. |
| 3 | Non utilizzato |
| 2,1,0 | Pixel: il valore di pixel+1 indica il numero di bit per pixel nell’immagine |

Lo *Screen Descriptor* descrive la superificie su cui verranno disegnate le immagini descritte successivamente nel file (*Image Descriptors*).

Se il flag M è attivo segue la *Global Color Map*. Il numero totale di colori è dato da $2^{\text{Pixel+1}}$. Per ogni colore sono presenti 3 byte, rispettivamente R, G e B:

| Byte# | Content | Purpose |
|---|---|---|
| 0 | R | Valore di rosso per l’indice 0 |
| 1 | G | Valore di verde per l’indice 0 |
| 2 | B | Valore di blu per l’indice 0 |
| 3 | R | Valore di rosso per l’indice 1 |
| 4 | G | Valore di verde per l’indice 1 |
| 5 | B | Valore di blu per l’indice 1 |
| … | … | … |

Dopo la *Global Color Map* sono presenti degli *Image Descriptor* che contengono informazioni su una parte dell’immagine (eventualmente tutta):

| Offset# | Size | Purpose |
|---|---|---|
| 0 | 1 byte | Image separator, ossia il carattere ‘,’ (virgola) |
| 2 | 2 bytes | Image Left (posizione orizzontale in pixel all’interno dello screen in cui va posizionata l’immagine) |
| 4 | 2 bytes | Image Top (posizione verticale in pixel all’interno dello screen in cui va posizionata l’immagine) |
| 6 | 2 bytes | Image Width (larghezza dell’immagine) |
| 8 | 2 bytes | Image Height (altezza dell’immagine) |
| 10 | 1 byte | ID_Flags |

Il byte ID_Flags è così suddiviso:

| Bit# | Purpose |
|---|---|
| 7 | M: se 1 indica che seguirà una *Local Color Map* da usare al posto di quella Global, se a 0 indica che verrà utilizzata la *Global Color Map* |
| 6 | I: se 1 indica che le righe dell’immagine saranno ordinate in modalità *interlaced* altrimenti saranno ordinate normalmente, dall’alto in basso e da sinistra a destra. |
| 5,4,3 | Non utilizzati |
| 2,1,0 | Pixel: il valore di pixel+1 indica il numero di bit per pixel nell’immagine. È da ignorare se M = 0 perché viene usato il numero di bit per pixel della Global Color Map. |

Dopo un Image Descriptor iniziano i dati Raster ad esso relativi. Questi dati sono memorizzati secondo il seguente schema:

| Field name | Purpose | Note |
|---|---|---|
| Code size | 1 byte: numero di bit iniziale da usare per inizializzare il dizionario. Normalmente il numero di bit per pixel. | |
| Block Byte Count | 1 byte: contiene il numero di byte presenti nel blocco successivo. | Ripetuti finchè necessario |
| Data bytes | <Block Byte Count> bytes: contiene una porzione dei dati dell’immagine. | |
| … | … | |
| 0 | 1 byte dal valore zero per indicare la fine dei dati raster (è un blocco da zero byte) | |

I dati raster delle immagini sono compressi utilizzando una variante dell’algoritmo LZW. Nello stream dei data bytes i bit vanno letti a partire dal *meno significativo di ogni byte.* Consideriamo ad esempio i seguenti 3 codici da 9 bit: 256 (0x100), 249 (0xF9), 170 (0xAA). Questi vengono impacchettati in 4 byte così:

```
             Code 3            Code 2            Code 1
          ================= ================= =================
0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 1 1 1 1 1 0 0 1 1 0 0 0 0 0 0 0 0
============== ============== ============== ==============
    Byte 4          Byte 3          Byte 2         Byte 1
```

Nel file quindi si troverebbero i 4 byte (in esadecimale): 00 F3 A9 02. Attenzione, perché questo è diverso dal modo classico che abbiamo visto a lezione nel bitreader.

L'algoritmo LZW usato nel GIF è diverso da quello normale per le seguenti caratteristiche:

- Un *clear code* speciale utilizzato per resettare il dizionario è costituito dal valore $2^{code\_size}$. Ad esempio se code_size è 4, i valori da 0 a 15 del dizionario sono gli indici 0-15 della color table, il valore $2^4 = 16$ è il clear code. Il clear code può comparire in qualsiasi punto dello stream e **compare sempre all'inizio dei dati di ogni Image Descriptor**.
- Un *end of information* è il clearcode + 1 e indica la fine dei dati dell'immagine. Sempre nell'esempio di prima dopo il reset, il dizionario contiene 18 valori: gli indici precaricati 0-15, il clear code (16) e l'end of information (17).
- Il primo valore che viene aggiunto al dizionario è quindi quello alla posizione clearcode + 2 e appena il numero di elementi nel dizionario supera il massimo valore rappresentabile con il numero di bit corrente, questo numero di bit viene incrementato di 1. Nel caso appena considerato, di comincia leggendo 5 bit (codesize=4, ma nel dizionario abbiamo anche già il clearcode e l'end of information), poi si inserirà la prima sequenza alla posizione 18. Dopo un po' si inseriscono altri valori nel dizionario fino alla posizione 31. Qui si leggono ancora 5 bit, ma poi viene aggiunto l'elemento alla posizione 32 e adesso il numero di bit da leggere diventa 6. Si procede in questo modo al massimo fino a 12 bit inclusi. Arrivati ad inserire il valore $2^{12} = 4096$, bisognerebbe leggere 13 bit, ma lo standard lo vieta. Qui si continua con 12 bit e necessariamente il prossimo codice sarà un clearcode (se non lo fosse lo stream sarebbe sbagliato).

Nel caso specifico di questo esercizio non sarà necessario tenere conto dei casi in cui le righe sono memorizzate in modalità interlaced e sarà sempre presente un solo *Image Descriptor* che coprirà tutta l'area a disposizione indicata nello *Screen Descriptor*.

Si scriva un programma a linea di comando che accetti le seguenti opzioni:

```
gif2ppm <input file .GIF> <output file .PPM>
```

Il programma deve aprire un file GIF, estrarne gli header, la color table, estrarre i dati dell'immagine, decomprimerli (ottenendo una mappa di indici della color table), creare una immagine a colori inserendo in ogni pixel il colore corrispondente all'indice del pixel e infine salvarla in PPM.

**Note:**
**- Consegnate solo codice che compila**
**- Se non gestite la linea di comando correttamente (devono esserci 2 argomenti oltre al nome dell'eseguibile dove il primo è il nome del file di input e il secondo il nome del file di output), perderete 3 punti.**