Formato dei file

Il formato immagine pbm

Il formato immagine pbm consente di rappresentare immagini in bianco e nero. Ogni file rappresenta una immagine come un file testo.

Definizione del formato immagine pbm

Definizione del formato immagine pbm

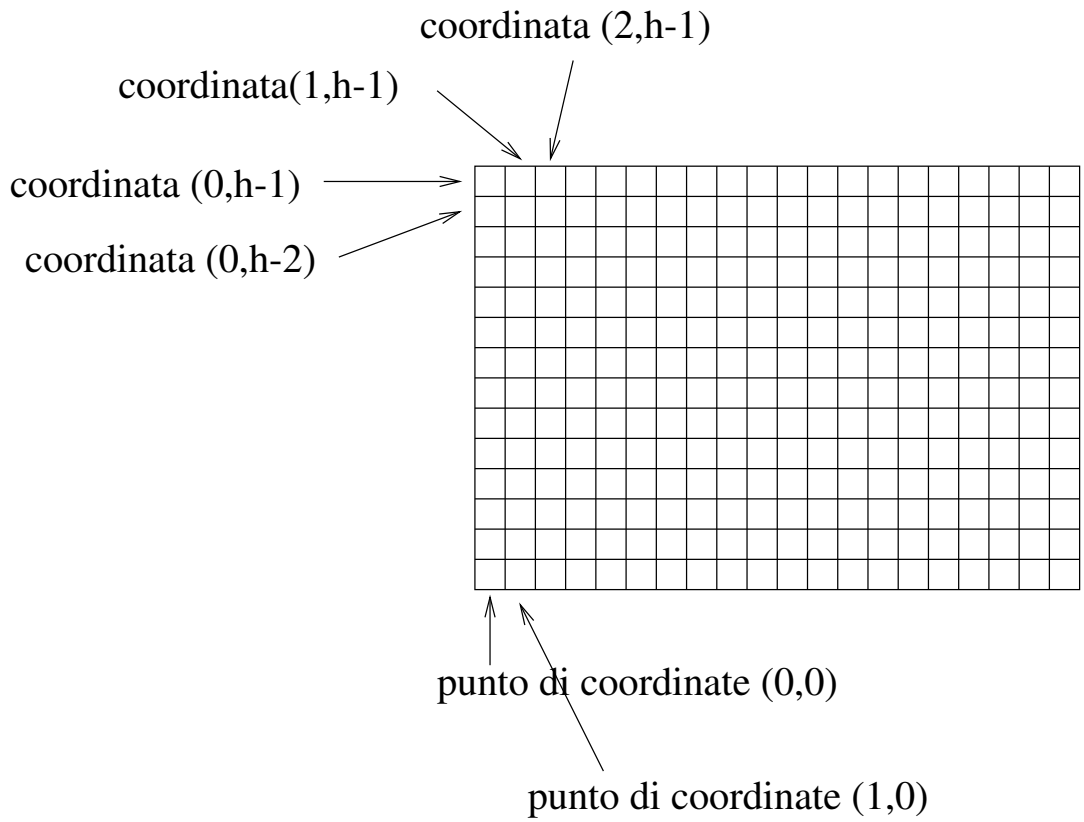
In questa pagina vediamo come è possibile, con un programma C, scrivere un file che contiene una immagine. Per prima cosa, i file immagini sono file come tutti gli altri, per cui possono venire aperti (in lettura oppure in scrittura) come un qualsiasi altro file. Quello che li rende diversi è solo il modo in cui sono organizzati, ossia il tipo di informazioni che contengono. Esistono numerosi modi di rappresentare una immagine su file, e ognuno di questi dà luogo a un differente formato. Formati esistenti sono per esempio: JPEG, GIF, BMP, PCX, PNG, TIFF, TGA, RAS/SUN, ICO, PBM, PPM, ecc.

In questa pagina vediamo il solo formato ‘‘plain pbm’’, che è il più facile da realizzare usando un programma C. Cominciamo quindi a leggere la specifica del formato pbm.

Per prima cosa, si tratta di un formato *raster*, ossia rappresenta una immagine pixel per pixel. Inoltre, si tratta di un formato per rappresentare solo immagini in bianco e nero. È quindi chiaro che, per ogni pixel dell'immagine che si va a rappresentare, il pixel può essere solo bianco o nero. In particolare, un file pbm è fatto come segue:

1. Il formato specifica che il file deve cominciare con la stringa `P1`. Se quindi vogliamo realizzare un programma che scrive un file in formato pbm, occorre aprire un file in scrittura, e scrivere subito la stringa `P1`.
2. La seconda cosa che il file deve contenere è la sua dimensione. In altre parole, per scrivere un file pbm, dobbiamo scrvivere larghezza e altezza di seguito alla stringa `P1`, tutti separati da spazi.
3. La cosa che segue nel file è la sequenza dei pixel della immagine. Come si è detto, ogni pixel può essere colorato solo in nero oppure in bianco. In particolare, il nero si rappresenta come `1`, mentre il bianco è `0`. Una cosa importante da tenere conto è la disposizione di questi numeri su file. Infatti, si deve partire dalla posizione in alto a sinistra, spostandosi ogni volta a destra fino alla fine della riga e poi in basso.

I primi due punti sono facili da interpretare: si scrive su file la stringa `P1`, poi la larghezza dell'immagine e poi l'altezza. Il terzo punto risulta subito chiaro se si disegna la griglia dei pixel, con le corrispondenti coordinate:



Il terzo punto della specifica dice semplicemente che i pixel devono apparire sul file come se venissero letti normalmente da sinistra a destra e dall'alto in basso. In altre parole, il file si ottiene leggendo i valori dei pixel come se fossero parole di un testo.

È quindi chiaro che si deve partire dal pixel in alto a sinistra, leggendo una riga per volta. Si deve quindi stampare prima il pixel di coordinate `0,h-1` poi `1,h-1` poi `2.h-2`, ecc e solo quando la riga è finita si può passare alla coordinata `y=h-2`.

Supponiamo quindi di voler stampare una matrice in cui tutti i punti siano bianchi tranne quelli all'interno del rettangolo compreso fra i punti `(10,10)` e `(40,20)`. Dal momento che dobbiamo stampare una riga per volta, e che il valore della coordinata `y` che indica la riga deve andare da `h-1` a `0`, il ciclo più esterno sarà:

```
for(y=h-1; y>=0; y--)
```

Per ogni riga, vanno stampati tutti i suoi punti, quindi occorre fare un secondo ciclo sulla coordinata `x`:

```
  for(x=0; x<=w-1; x++)
```

Per ogni pixel, possiamo decidere se deve essere bianco o nero a seconda se si trova o meno all'interno del rettangolo. In particolare, il punto è nero solo se la sue coordinata `x` è compresa fra le `x` dei due punti estremi del rettangolo, e lo stesso per la `y`. Quindi, per ogni punto facciamo:

```
    if( x>=10 && x<=40 && y>=10 && y<=20 )
      fprintf(fd, "1\n");
    else
      fprintf(fd, "0\n");
```

Il programma completo di stampa del rettangolo rettangolo.c è riportato qui sotto.

```
/*
  Scrive un rettangolo su file.
*/

#include<stdlib.h>
#include<stdio.h>

int main() {
  int w=200, h=100;
  FILE *fd;
  int x, y;

                /* apre il file */
  fd=fopen("rettangolo.pbm", "w");
  if( fd==NULL ) {
    perror("Errore in apertura del file");
    exit(1);
  }


                /* scrive il rettangolo */
  fprintf(fd, "P1\n");
  fprintf(fd, "%d %d\n", w, h);

  for(y=h-1; y>=0; y--)
    for(x=0; x<=w-1; x++)
      if( x>=10 && x<=40 && y>=10 && y<=20 )
        fprintf(fd, "1\n");
      else
        fprintf(fd, "0\n");


                /* chiude il file */
  fclose(fd);

  return 0;
}
```

L'immagine che viene prodotta da questo programma è riportata qui sotto.

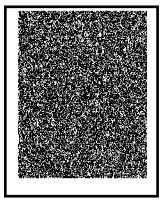
Una immagine random

Una immagine random

Il seguente programma random.c scrive una immagine random su un file. La istruzione `srandom(time(NULL));` è necessaria all'inizio per inizializzare il generatore di numeri random. Dopo aver aperto il file, si determinano altezza e larghezza dell'immagine sempre come numeri random. Per ogni punto dell'immagine, si decide se deve essere nero o bianco semplicemente usando il generatore di numeri casuali. Si ricorda che il risultato della espressione `random()%n`, in cui `n` è un numero intero, ritorna un numero casuale compreso fra `0` ed `n-1`.

```
/*
  Scrive una bitmap random.
*/

#include<stdlib.h>
#include<stdio.h>
#include<time.h>

int main() {
  FILE *fd;

  int x, y;
  int w, h;


                /* inizializza il generatore casuale */
  srandom(time(NULL));

                /* apre il file in scrittura */
  fd=fopen("test.pbm", "w");
  if(fd==NULL) {
    perror("Errore in apertura del file");
    exit(1);
  }

                /* determina larghezza e altezza */
  w=random()%300;
  h=random()%300;

                /* intestazione */
  fprintf(fd, "P1 %d %d\n", w, h);

                /* scrive la bitmap su file */
  for(y=h-1; y>=0; y--)
    for(x=0; x<=w-1; x++)
      if(random()%2)
        fprintf(fd, "1\n");
      else
        fprintf(fd, "0\n");

                /* chiude il file */
  fclose(fd);

  return 0;
}
```

Un esempio di immagine prodotta è qui sotto.



Usare una struttura per rappresentare una immagine

Usare una struttura per rappresentare una immagine

La strategia seguita nelle pagine precedenti per scrivere un file immagine andava bene per immagini semplici. Infatti, quello che si faceva era un doppio ciclo, in cui per ogni punto, si controllava se doveva essere bianco o nero in base alle specifiche di come doveva essere fatta l'immagine. Questo va bene per disegnare un singolo rettangolo, ma si provi a pensare come va modificato il codice se i rettangoli da disegnare sono quattro: l'istruzione di test se un punto è all'interno di essi diventa molto lunga. Si tenga anche presente che l'immagine con quattro rettangoli è molto semplice, e che può servire scrivere immagini molto più complicate.

A meno che l'immagine non sia realmente molto semplice, si usa di solito un'altra strategia: si memorizza l'immagine in una matrice, e ogni volta che si vuole disegnare qualcosa si cambiano gli elementi di questa matrice. Ogni volta che si vuole il file immagine, si stampa la matrice su file.

Per essere più precisi, per rappresentare l'immagine servono due interi che identificano larghezza e altezza, più una matrice grande abbastanza per contenere tutti i pixel. Dal momento che queste tre variabili descrivono complessivamente la stessa cosa, usiamo una struttura. Per la matrice, ipotizziamo che l'immagine non sia più larga di 400×400.

```
struct ImmagineBW {
  int larghezza;
  int altezza;
  int mat[400][400];
};
```

Con questa struttura possiamo memorizzare immagini di diverse dimensioni, a condizioni che entrambe le dimensioni siano comprese fra 0 e 400.

Il primo passo da fare è quello di definire una variabile di tipo immagine per rappresentare l'immagine che stiamo costruendo. La seconda cosa da fare è quella di dire quanto è grande l'immagine, ossia mettere i valori giusti di larghezza e altezza nei campi della variabile immagine.

A questo punto, mettiamo 0 in tutti gli elementi significativi della matrice. Questo equivale a ‘‘cancellare’’ l'immagine, ossia a definire inizialmente tutti i punti come bianchi. Se non si compie questa operazione la matrice contiene valori non inizializzati, per cui i suoi valori sono indeterminati.

Abbiamo ora una lavagna su cui disegnare. Se per esempio vogliamo disegnare il punto di coordinate `x,y`, basta mettere a uno l'elemento della matrice di indici `x` e `y`. Per il rettangolo, facciamo un doppio ciclo, e in questo modo riusciamo a fare questa operazione per tutti i punti del rettangolo.

Il programma completo matrice.c è qui sotto.

```
/*
  Definisce il tipo immagine come una matrice.
*/

#include<stdlib.h>
#include<stdio.h>


/* definizione del tipo */

struct ImmagineBW {
  int larghezza;
  int altezza;
```

```
  int mat[400][400];
};


/*
  crea un file a partire da una matrice
*/

void CreaFileImmagine(char *nomefile, struct ImmagineBW img) {
  FILE *fd;
  int x, y;

                /* apre il file */
  fd=fopen(nomefile, "w");
  if( fd==NULL ) {
    perror("Errore in apertura del file");
    exit(1);
  }


                /* stampa la matrice */
  fprintf(fd, "P1\n");
  fprintf(fd, "%d %d\n", img.larghezza, img.altezza);

  for(y=img.altezza-1; y>=0; y--)
    for(x=0; x<=img.larghezza-1; x++)
      if( img.mat[x][y] )
        fprintf(fd, "1\n");
      else
        fprintf(fd, "0\n");


                /* chiude il file */
  fclose(fd);
}


/*
  main
*/

int main() {
  struct ImmagineBW q;
  int x, y;

                /* inizializza la matrice */
  q.larghezza=200;
  q.altezza=100;

  for(x=0; x<=q.larghezza-1; x++)
    for(y=0; y<=q.altezza-1; y++)
      q.mat[x][y]=0;


                /* disegna il rettangolo nella matrice */
  for(x=10; x<=40; x++)
    for(y=10; y<=20; y++)
      q.mat[x][y]=1;


                /* scrive l'immagine su file */
```

```
  CreaFileImmagine("matrice.pbm", q);

  return 0;
}
```

L'immagine che viene generata da questo programma è riportata qui sotto.

Grafico di una funzione

Grafico di una funzione

Il programma che segue realizza il grafico di una funzione in formato pbm. Il problema non presenta nessuna difficoltà: si dichiara una struttura per rappresentare l'immagine, e la si inizializza. Alla fine, va scritta su file.

Questo schema è fisso: quello che manca è la specifica di quello che si fa in mezzo, ossia il disegno vero e proprio che viene fatto sulla matrice. Nel nostro caso, per ogni valore di `x` determiniamo il valore di `y` (il risultato della funzione), e mettiamo a 1 il pixel di coordinate `x,y` della matrice. L'unica cosa da tenere in considerazione è il fatto che il valore di `y` non è necessariamente compreso negli indici ammissibili della matrice. Prima di effettuare l'operazione di mettere 1 nella matrice, occorre quindi controllare che il punto sia compreso all'intero delle coordinate dell'immagine.

Il programma grafico.c è qui sotto.

```
/*
  Grafico di una funzione in formato pbm
*/

#include<stdlib.h>
#include<stdio.h>


/* definizione del tipo */

struct ImmagineBW {
  int larghezza;
  int altezza;
  int mat[400][400];
};


/*
  crea un file a partire da una matrice
*/

void CreaFileImmagine(char *nomefile, struct ImmagineBW img) {
  FILE *fd;
  int x, y;

                /* apre il file */
  fd=fopen(nomefile, "w");
  if( fd==NULL ) {
    perror("Errore in apertura del file");
```

```
    exit(1);
  }


                /* stampa la matrice */
  fprintf(fd, "P1\n");
  fprintf(fd, "%d %d\n", img.larghezza, img.altezza);

  for(y=img.altezza-1; y>=0; y--)
    for(x=0; x<=img.larghezza-1; x++)
      if( img.mat[x][y] )
        fprintf(fd, "1\n");
      else
        fprintf(fd, "0\n");


                /* chiude il file */
  fclose(fd);
}


/*
  main
*/

int main() {
  struct ImmagineBW q;
  int x, y;

                /* inizializza la matrice */
  q.larghezza=200;
  q.altezza=100;

  for(x=0; x<=q.larghezza-1; x++)
    for(y=0; y<=q.altezza-1; y++)
      q.mat[x][y]=0;


                /* realizza il grafico della funzione */
  for(x=0; x<=q.larghezza-1; x++) {
    y=x/4+(x/8)%20-10;
    if(y>=0 && y<=q.altezza-1)
      q.mat[x][y]=1;
  }


                /* scrive l'immagine su file */
  CreaFileImmagine("funzione.pbm", q);

  return 0;
}
```

L'immagine che viene generata è qui sotto:

Funzioni di disegno

Funzioni di disegno

Nel programma precedente abbiamo visto una funzione che permette di scrivere una figura su file. È però possibile anche scrivere delle funzioni che realizzano disegni sulla struttura che rappresenta l'immagine. Una cosa sicuramente utile è scrivere una funzione che inizializza l'immagine, cioè mette i valori di altezza e larghezza nei corrispondenti campi, e riempie la matrice di zeri. Usiamo una funzione che restiuisce una immagine. È infatti possibile relizzare delle funzioni che hanno una struttura come valore di ritorno.

```
struct ImmagineBW ImmagineVuota(int l, int a) {
  struct ImmagineBW img;
  int x, y;

  img.larghezza=l;
  img.altezza=a;

  for(x=0; x<=l-1; x++)
    for(y=0; y<=a-1; y++)
      img.mat[x][y]=0;

  return img;
}
```

Questa funzione mette nei campi `larghezza` e `altezza` i parametri passati. Inoltre, mette a zero tutti gli elementi significativi della matrice.

Un'altra funzione che può essere utile è quella che disegna una linea orizzontale fra due punti `x1, y` e `x2, y`. Questa funzione deve ricevere come parametri le tre coordinate `x1, y2, x`, ed inoltre la struttura su cui fare il disegno. Si noti che la funzione modifica l'immagine, per cui questa va passata per riferimento. In altre parole, dobbiamo passare l'indirizzo della struttura e non la struttura stessa.

```
void DisegnaLineaOrizzontale(struct ImmagineBW *pi, int x1, int x2, int y) {
  int x;

  for(x=x1; x<=x2; x++)
    (*pi).mat[x][y]=1;
}
```

Se si fosse passata l'immagine per valore (ossia il parametro fosse stato `struct ImmagineBW img`) allora la funzione avrebbe operato con una copia locale della struttura, per cui le modifiche non avrebbero avuto effetto sulla struttura di partenza. In altre parole, chiamando `DisegnaLineaOrizzontale(q,...)` si sarebbe creata una nuova struttura immagine `img`, e le modifiche sarebbero state fatte su questa invece che su `q`. Passando l'indirizzo, siamo sicuri che ogni modifica fatta su `*pi` viene fatta sulla struttura della quale abbiamo passato l'indirizzo.

È anche possibile definire in modo analogo funzioni di disegno di linee verticali, rettangoli, linee oblique, ecc. L'immagine su cui si opera, dal momento che viene modificata dalla funzione, e vogliamo che le modifiche siano visibili al programma chiamante, va passata per riferimento.

Una volta definite queste funzioni, possiamo realizzare un programma che scrive una immagine con due linee, in questo modo:

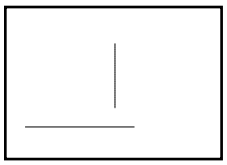
```
int main() {
  struct ImmagineBW q;

                /* inizializza l'immagine */
  q=ImmagineVuota(300, 200);


                /* disegna due linee */
  DisegnaLineaOrizzontale(&q, 10, 180, 20);
  DisegnaLineaVerticale(&q, 150, 50, 150);


                /* scrive l'immagine su file */
  CreaFileImmagine("duelinee.pbm", q);

  return 0;
}
```

Va notato come, in questo programma, non ci serve sapere né come sono rappresentate le immagini, e nemmeno in che modo funzionano le funzioni di inizializzazione, disegno e stampa su file. Tutto quello che ci serve sapere, di ognuna di queste funzioni, è cosa fa. In altre parole, se qualcun altro vuole scrivere un programma che fa un disegno differente, tutto quello che deve sapere è che prima va chiamata la funzione `ImmagineVuota` passando le dimensioni della figura, e poi che per disegnare una linea si usano le funzioni `DisegnaLineaOrizzontale` e `DisegnaLineaVerticale`, e che alla fine si chiama la funzione `CreaFileImmagine` per scrivere l'immagine su file. Non gli serve sapere né come è fatta la struttura `struct ImmagineBW` e nemmeno in che modo sono realizzate le varie funzioni.

Il programma completo confun.c è qui sotto.

```
/*
  Funzioni per disegnare in una struttura.
*/

#include<stdlib.h>
#include<stdio.h>


/* definizione del tipo */

struct ImmagineBW {
  int larghezza;
  int altezza;
  int mat[400][400];
};


/*
  crea un file a partire da una matrice
*/

void CreaFileImmagine(char *nomefile, struct ImmagineBW img) {
  FILE *fd;
  int x, y;

                /* apre il file */
  fd=fopen(nomefile, "w");
  if( fd==NULL ) {
    perror("Errore in apertura del file");
```

```
    exit(1);
  }


                /* stampa la matrice */
  fprintf(fd, "P1\n");
  fprintf(fd, "%d %d\n", img.larghezza, img.altezza);

  for(y=img.altezza-1; y>=0; y--)
    for(x=0; x<=img.larghezza-1; x++)
      if( img.mat[x][y] )
        fprintf(fd, "1\n");
      else
        fprintf(fd, "0\n");


                /* chiude il file */
  fclose(fd);
}


/*
  Crea una immagine vuota.
*/

struct ImmagineBW ImmagineVuota(int l, int a) {
  struct ImmagineBW img;
  int x, y;

  img.larghezza=l;
  img.altezza=a;

  for(x=0; x<=l-1; x++)
    for(y=0; y<=a-1; y++)
      img.mat[x][y]=0;

  return img;
}


/*
  Disegna una linea orizzontale.
*/

void DisegnaLineaOrizzontale(struct ImmagineBW *pi, int x1, int x2, int y) {
  int x;

  for(x=x1; x<=x2; x++)
    (*pi).mat[x][y]=1;
}


/*
  Disegna una linea verticale.
*/

void DisegnaLineaVerticale(struct ImmagineBW *pi, int x, int y1, int y2) {
  int y;

  for(y=y1; y<=y2; y++)
    (*pi).mat[x][y]=1;
}
```

```
/*
  main
*/

int main() {
  struct ImmagineBW q;

                /* inizializza l'immagine */
  q=ImmagineVuota(300, 200);


                /* disegna due linee */
  DisegnaLineaOrizzontale(&q, 10, 180, 20);
  DisegnaLineaVerticale(&q, 150, 50, 150);


                /* scrive l'immagine su file */
  CreaFileImmagine("duelinee.pbm", q);

  return 0;
}
```

Questo programma genera l'immagine riportata qui sotto.