POLITECNICO
DI TORINO
Dipartimento
di Automatica e Informatica

Capitolo 5: Problem-solving su problemi complessi

DAL PROBLEMA AL PROGRAMMA: LA SCOMPOSIZIONE IN SOTTOPROBLEMI

Il paradigma “divide et impera”

- Un problema complesso può essere affrontato (e risolto) con successo mediante una strategia “divide et impera”:
  - scomposizione in sotto-problemi più semplici
  - soluzione dei problemi elementari
  - ricombinazione delle soluzioni elementari nella soluzione del problema di partenza.
- I sotto-problemi non sono necessariamente di ugual natura
- La scomposizione non sempre è univoca, esistono cioè vari modi di scomporre un problema (e la relativa soluzione)
- La scomposizione in sotto-problemi si attua in termini di:
  - struttura dati
  - algoritmo

Divisione-Soluzione-Ricombinazione

- Divisione in sotto-problemi:
  - può essere elementare e immediata: basta individuare i sotto-problemi
  - può essere complessa e richiedere la generazione di strutture dati “ad hoc” per i sotto-problemi
- Soluzione dei sotto-problemi:
  - possono essere indipendenti (li si può eseguire in qualunque ordine)
  - oppure richiedono una determinata successione, in quanto i risultati di un sotto-problema sono dati in ingresso per altri sotto-problemi
- Ricombinazione delle soluzioni:
  - può essere elementare e immediata: la soluzione globale è già disponibile una volta risolti i sotto-problemi
  - occorre elaborare i risultati (parziali) dei sotto-problemi, per ottenere il risultato finale del problema complessivo

Scomposizione di strutture dati

- La struttura dati può essere scomposta in vari modi e/o livelli:
  - nessuna scomposizione: la scomposizione è unicamente di tipo algoritmico. La struttura dati (unica) del problema, viene manipolata in tutti i sotto-problemi
  - partizionamento dei dati: la struttura dati viene suddivisa in parti (omogenee) associate a singoli sotto-problemi (es. vettore o matrice scomposti in sezioni)
  - strutture dati “ad hoc” per singoli sotto-problemi (ed eventuali partizionamenti): è lo schema più generale.

Scomposizione di algoritmi

- Un algoritmo può essere considerato scomposto in sezioni, corrispondenti a:
  - costrutti condizionali
  - (singole iterazioni di) costrutti iterativi
  - (chiamate a) funzioni
- Le singole sezioni possono:
  - manipolare dati globali
  - ricevere dati in ingresso e restituire risultati

Indipendenza e/o correlazione

- Le sezioni/parti possono essere:
  - indipendenti, se operano su dati diversi
    - può essere arbitrario l'ordine di risoluzione dei sottoproblemi
    - non sono necessari risultati intermedi tra i sottoproblemi

      Esempio: calcolo del valor medio, per ogni riga di una matrice)
  - correlate, se ogni sottoproblema dipende dagli altri:
    - non è arbitrario l'ordine di risoluzione dei sottoproblemi
    - è necessario prevedere (a livello di struttura dati) la memorizzazione di risultati intermedi

Import-export

- Formulazione: sono dati due file di testo, contenenti le informazioni sulle operazioni di una ditta di import/export, che ha relazioni commerciali con società appartenenti a vari stati
  - un primo file (detto nel seguito F0) contiene l'elenco degli stati e delle società con cui esistono relazioni commerciali
  - un secondo file (detto nel seguito F1) contiene le informazioni su un certo numero di transazioni commerciali (con una delle società elencate in F0)

Dati in ingresso

- Il file F0 contiene l'elenco degli stati e delle società:
  - nella prima riga due interi <nst> <nsoc> (separati da spazio): numero totale di stati e di società
  - nelle <nst> righe successive sono elencati gli stati, uno per riga, secondo il formato:
    <codice_stato> <nome_stato>
    - <codice_stato> è un codice intero compreso tra 0 e 99 (ogni stato ha un codice unico, non si garantisce che gli stati siano elencati per ordine crescente di codice)
    - <nome_stato> è il nome dello stato, privo di spazi
  - Nelle <nsoc> righe finali del file sono elencate le società. Per ognuna, su una riga, viene riportato:
    <codice_stato> <nome_società>:
    - <codice_stato> è il codice dello stato cui appartiene la società
    - <nome_società> è il nome della società

- F1 contiene le informazioni su un insieme di transazioni commerciali, ognuna delle quali viene rappresentata, su una riga del file, secondo il formato:
  <nome_società> <importo> <data>

  <nome_società> è il nome della società
  - <importo> (intero con segno) rappresenta l'importo della transazione (negativo per importazione, positivo per esportazione)
  - <data> è la data della transazione (formato gg/mm/aaaa)

Esempio

F0

```
5 10
4 Germania
3 Francia
0 Stati_Uniti
2 Giappone
1 Corea
1 Kia
4 BMW
4 Mercedes
3 Peugeot
3 Citroen
2 Honda
2 Mitsubishi
0 Chrisler
0 General_Motors
0 Chevrolet
```

F1

```
Mitsubishi 101 21/01/2004
Chrisler -30 01/02/2004
General_Motors -2000 10/02/2004
Citroen -700 24/11/2004
Chrisler -367 12/12/2004
...
General Motors -1400 14/12/2004
Chevrolet -600 16/12/2004
```

Elaborazioni richieste

- Leggere F0 e F1, i cui nomi sono ricevuti come argomenti sulla linea di comando
- Calcolare, e stampare su video:
  - il numero totale delle transazioni (NT) e i bilanci complessivi delle transazioni di importazione e esportazione
    - SI: somma degli importi relativi a importazioni
    - SE: somma degli importi relativi a esportazioni
  - per ogni stato, il bilancio complessivo commerciale di import-export (sommatoria degli importi relativi a tutte le società appartenenti allo stato)
  - lo stato per cui è massimo il bilancio di importazione e quello per cui è massimo il bilancio di esportazione.

Struttura dati

- Due tabelle contengono i dati letti dai file:
  - tabella degli stati: contiene l'elenco degli stati, identificati da codice e nome. Siccome i codici sono compresi tra 0 e 99, è opportuno utilizzare il codice come indice in un vettore, per consentire la conversione da codice a nome con accesso diretto
  - tabella delle società: contiene l'elenco delle società, per ognuna delle quali viene riportato il nome e il codice dello stato di appartenenza. La tabella permette di ricavare il codice dello stato a partire dal nome di una società

- Variabili (scalari) per le statistiche richieste:
  - sommatoria delle importazioni
  - sommatoria delle esportazioni
- Un vettore per il bilancio di import-export dei singoli stati. Tale bilancio può essere aggiunto (come ulteriore campo) alla tabella degli stati
- NON è necessaria una struttura dati per l'elenco delle transazioni, che possono essere lette e manipolate una alla volta.

Rappresentazione della struttura dati

Tabella degli stati

0	0	"Stati Uniti"
1	0	"Corea"
2	0	"Giappone"
3	0	"Francia"
4	0	"Germania"

Tabella delle società

1	"Kia"
4	"BMW"
4	"Mercedes"
	…
	…
0	"Chevrolet"

Rappresentazione della struttura dati

Tabella degli stati

0	0	"Stati Uniti"
1	0	"Corea"
2	0	"Giappone"
3	0	"Francia"
4	0	"Germania"

Bilancio dello stato, inizialmente nullo

Tabella delle società

1	"Kia"
4	"BMW"
4	"Mercedes"
	…
	…
0	"Chevrolet"

Algoritmo

- Raccolta delle statistiche import-export: calcolo di sommatorie, dopo aver filtrato i dati:
  - transazioni negative (import) e positive (export)
  - transazioni per singoli stati
- Problema di ricerca del massimo, ripetuto per lo stato con massimo import e quello con massimo export

Scomposizione in sotto-problemi

- Un primo livello di scomposizione è connesso alla natura del problema:
  - calcolo statistiche import export
  - ricerca dei massimi
- Un ulteriore sottoproblema è costituito dalla conversione

  nome società → codice stato

  che può essere ricondotta a un problema di ricerca nella tabella delle società

Codice

```c
#include <stdio.h>
#define MAXC 101
#define MAXSTATI 100
#define MAXSOC 100
/* tipi struct per stati e societa' */
typedef struct {
  char nome[MAXC]; int bilancio;
} t_stato;
typedef struct {
  char nome[MAXC]; int codice_stato;
} t_societa;
/* prototipo di funzione */
int cercaCodiceStato(
  t_societa tab_soc[], int n, char nome[]
);
```

```c
int main (int argc, char *argv[]) {
  FILE *fp;
  int nst, nsoc, codice, importo,
      i, nt, si, se, maxi, maxe;
  char nome[MAXC];
  t_stato tab_st[MAXSTATI];
  t_societa tab_soc[MAXSOC];
  /* lettura intestazione file F0 */
  fp = fopen(argv[1],"r");
  fscanf(fp,“%d%d”,&nst,&nsoc);
  /* Inizializzazione nomi con stringa vuota */
  for (i=0; i<MAXSTATI; i++)
    strcpy(tab_st[i].nome,””);
```

Codice

```
#include <stdio.h>
#define MAXC 101
#define MAXSTATI 100
#define MAXSOC 100
/* tipi struct per stat
typedef struct {
  char nome[MAXC]; int
} t_stato;
typedef struct {
  char nome[MAXC]; int codice_stato;
} t_societa;
/* prototipo di funzione */
int cercaCodiceStato(
  t_societa tab_soc[], int n, char nome[]
);
```

```
int main (int argc, char *argv[]) {
  FILE *fp;
  int nst, nsoc, codice, importo,
      i, nt, si, se, maxi, maxe;
  /* lett                    file FO */
  fp = fopen                  );
  fscanf(fp,"%d               nst,&nsoc);
  /* Inizializzazione nomi con stringa vuota */
  for (i=0; i<MAXSTATI; i++)
    strcpy(tab_st[i].nome,"");
```

I nomi di stati sono inizializzati a stringa vuota.
I codici con nome vuoto dopo la lettura sono non assegnati

Codice

```
/* lettura tabella stati */
for (i=0; i<nst; i++) {
  fscanf(fp,"%d%s", &codice,nome);
  strcpy(tab_st[codice].nome,nome);
  tab_st[codice].bilancio = 0;
}
/* lettura tabella societa' */
for (i=0; i<nsoc; i++) {
  fscanf(fp,"%d%s", &codice, nome);
  strcpy(tab_soc[i].nome,nome);
  tab_soc[i].codice_stato = codice;
}
fclose(fp);
```

```
/* transazipni */
fp = fopen(argv[2],"r");
nt = si = se = 0;
while (fscanf(fp,"%s%d%*s",nome,&importo) != EOF) {
  nt++;
  if (importo > 0)
    se += importo;
  else
    si += importo;
  codice=cercaCodiceStato(tab_soc,nsoc,nome);
  tab_st[codice].bilancio += importo;
}
fclose(fp);
```

Codice

```
/* lettura tabella stati */
for (i=0; i<nst; i++) {
  fscanf(fp,"%d%s", &codice,nome);
  strcpy(tab_st[codice].nome,nome);
  tab_st[codice].bilancio = 0;
}
/* lettura tabella societa' */
for (i=0; i<nsoc; i++) {
  fscanf(fp,"%d%s", &codice, nome);
  strcpy(tab_soc[i].nome,nome);
  tab_soc[i].codice_stato = codice;
}
fclose(fp);
```

```
/* transazipni */
fp = fopen(argv[2],"r");
nt = si = se = 0;
while (fscanf(fp,"%s%d%*s",nome,&importo) != EOF) {
  nt++;
  if (importo > 0)
    se += importo;
}
fclose(fp);
```

Corrispondenza indice-dato
Le informazioni sugli stati sono immagazzinate nella casella avente come indice il codice. Il bilancio è inizializzato a 0

Codice

```
/* lettura tabella stati */
for (i=0; i<nst; i++) {
  fscanf(fp,"%d%s", &codice,nome);
  strcpy(tab_st[codice].nome,nome);
  tab_st[codice].bilancio = 0;
}
/* lettura tabella societa' */
for (i=0; i<nsoc; i++) {
  fscanf(fp,"%d%s", &codice, nome);
  strcpy(tab_soc[i].nome,nome);
  tab_soc[i].codice_stato = codice;
}
fclose(fp);
```

```
/* transazipni */
fp = fopen(argv[2],"r");
                                    )F) {
    se += importo;
  else
    si += importo;
  codice=cercaCodiceStato(tab_soc,nsoc,nome);
  tab_st[codice].bilancio += importo;
}
fclose(fp);
```

Vettore come contenitore
Le informazioni sulle società sono immagazzinate progressivamente (con indici crescenti)

Codice

```
/* lettura tabella stati */
for (i=0; i<nst; i++) {
  fscanf(fp,"%d%s", &codice,nome);
  strcpy(tab_st[codice].nome,nome);
  tab_st[codice].bilancio = 0;
}
/* lettura tabella societa' */
for (i=0; i<nsoc; i++) {
```

Transazioni
Non sono immagazzinate in vettore, ma sono gestite direttamente mentre vengono lette:
- sommatorie
- aggiornamento bilancio stato

```
/* transazipni */
fp = fopen(argv[2],"r");
nt = si = se = 0;
while (fscanf(fp,"%s%d%*s",nome,&importo) != EOF) {
  nt++;
  if (importo > 0)
    se += importo;
  else
    si += importo;
  codice=cercaCodiceStato(tab_soc,nsoc,nome);
  tab_st[codice].bilancio += importo;
}
fclose(fp);
```

Codice

```
/* lettura tabella stati */
for (i=0; i<nst; i++) {
  fscanf(fp,"%d%s", &codice,nome);

}
/* lettura tabella società */
for (i=0; i<nsoc; i++) {
  fscanf(fp,"%d%s", &codice, nome);
  strcpy(tab_soc[i].nome,nome);
  tab_soc[i].codice_stato = codice;
}
fclose(fp);
```

Filtro sui dati:
se importo > 0 -> export
altrimenti -> import

```
/* transazipni */
fp = fopen(argv[2],"r");
nt = si = se = 0;
while (fscanf(fp,"%s%d%*s",nome,&importo) != EOF) {
  nt++;
  if (importo > 0)
    se += importo;
  else
    si += importo;
  codice=cercaCodiceStato(tab_soc,nsoc,nome);
  tab_st[codice].bilancio += importo;
}
fclose(fp);
```

Codice

```
/* lettura tabella stati */
for (i=0; i<nst; i++) {
  fscanf(fp,"%d%s", &codice,nome);
  strcpy(tab_st[codice].nome,nome);
}
/* lettura tabella società */
for (i=0; i<nsoc; i++) {
  fscanf(fp,"%d%s", &codice, nome);
  strcpy(tab_soc[i].nome,nome);
  tab_soc[i].codice_stato = codice;
}
fclose(fp);
```

Conversione nome-codice, fatta come sottoproblema di ricerca

```
/* transazipni */
fp = fopen(argv[2],"r");
nt = si = se = 0;
while (fscanf(fp,"%s%d%*s",nome,&importo) != EOF) {
  nt++;
  if (importo > 0)
    se += importo;
  else
    si += importo;
  codice=cercaCodiceStato(tab_soc,nsoc,nome);
  tab_st[codice].bilancio += importo;
}
fclose(fp);
```

Codice

```
/* calcola stati con max. imp.-exp. */
maxi = maxe = -1;
for (i=0; i<nst; i++)
  if (tab_st[i].nome[0] != '\0') {
    if (tab_st[i].bilancio > 0) {
      if (maxe<0 || tab_st[i].bilancio
                  > tab_st[maxe].bilancio)
        maxe = i;
      } else if (tab_st[i].bilancio < 0) {
        if (maxi<0 || tab_st[i].bilancio
                    < tab_st[maxi].bilancio)
          maxi = i;
      }
    }
```

```
  /* stampa statistiche globali */
  printf("Num. transazioni: %d\n", nt);
  printf("Totali imp.-exp.: %d %d\n\n", si, se);
  /* stampa statistiche per stato */
  for (i=0; i<nst; i++)
    if (tab_st[i].nome[0] != '\0')
      printf("STATO: %-30s BILANCIO: %8d\n",
       tab_st[i].nome,tab_st[i].bilancio);
  /* stampa max. import-export */
  if (maxe>=0) printf("Max. exp -> %s: %d\n",
    tab_st[maxe].nome,tab_st[maxe].bilancio);
  if (maxi>=0) printf("Max. imp -> %s: %d\n",
    tab_st[maxi].nome,tab_st[maxi].bilancio);
}
```

Codice

```
/* calcola stati con max. imp.-exp. */
maxi = maxe = -1;
for (i=0; i<nst; i++)
  if (tab_st[i].nome[0] != '\0') {
    if (tab_st[i].bilancio > 0) {
      if (maxe<0 || tab_st[i].bilancio
                  > tab_st[maxe].bilancio)
        maxe = i;
      } else if (tab_st[i].bilancio < 0) {
        if (maxi<0 || tab_st[i].bilancio
                    < tab_st[maxi].bilancio)
          maxi = i;
      }
    }
```

```
  /* stampa statistiche globali */
  printf("Num. transazioni: %d\n", nt);
  printf("Totali imp.-exp.: %d %d\n\n", si, se);
  /* stampa statistiche per stato */
  for (i=0; i<nst; i++)
        tab_st[i].nome,tab_st[i].bilancio);
  /* stampa max. import-export */
  if (maxe>=0) printf("Max. exp -> %s: %d\n",
    tab_st[maxe].nome,tab_st[maxe].bilancio);
  if (maxi>=0) printf("Max. imp -> %s: %d\n",
    tab_st[maxi].nome,tab_st[maxi].bilancio);
}
```

Cerca (indice dello) stato con massimo export

Codice

```
/* calcola stati con max. imp.-exp. */
maxi = maxe = -1;
for (i=0; i<nst; i++)
  if (tab_st[i].nome[0] != '\0') {
    if (tab_st[i].bilancio > 0) {
      if (maxe<0 || tab_st[i].bilancio
                  > tab_st[maxe].bilancio)
        maxe = i;
      } else if (tab_st[i].bilancio < 0) {
        if (maxi<0 || tab_st[i].bilancio
                    < tab_st[maxi].bilancio)
          maxi = i;
      }
    }
```

```
  /* stampa statistiche globali */
  printf("Num. transazioni: %d\n", nt);
  printf("Totali imp.-exp.: %d %d\n\n", si, se);
  /* stampa statistiche per stato */
  for (i=0; i<nst; i++)
    if (tab_st[i].nome[0] != '\0')
      printf("STATO: %-30s BILANCIO: %8d\n",
       tab_st[i].nome,tab_st[i].bilancio);
  /* stampa max. import-export */
  if (ma
    tab_
    tab_st[maxi].nome,tab_st[maxi].bilancio);
}
```

Cerca (indice dello) stato con massimo import

Codice

```
/* calcola stati con max. imp.-exp. */
maxi = maxe = -1;
for (i
  if (
    if (tab_st[i].bilancio > 0) {
      if (maxe<0 || tab_st[i].bilancio
                  > tab_st[maxe].bilancio)
        maxe = i;
      } else if (tab_st[i].bilancio < 0) {
        if (maxi<0 || tab_st[i].bilancio
                    < tab_st[maxi].bilancio)
          maxi = i;
      }
    }
```

Stampa risultati finali

```
  /* stampa statistiche globali */
  printf("Num. transazioni: %d\n", nt);
  printf("Totali imp.-exp.: %d %d\n\n", si, se);
  /* stampa statistiche per stato */
  for (i=0; i<nst; i++)
    if (tab_st[i].nome[0] != '\0')
      printf("STATO: %-30s BILANCIO: %8d\n",
       tab_st[i].nome,tab_st[i].bilancio);
  /* stampa max. import-export */
  if (maxe>=0) printf("Max. exp -> %s: %d\n",
    tab_st[maxe].nome,tab_st[maxe].bilancio);
  if (maxi>=0) printf("Max. imp -> %s: %d\n",
    tab_st[maxi].nome,tab_st[maxi].bilancio);
}
```

Codice

```
int cercaCodiceStato (t_societa tab_soc[],
                      int n, char nome[]) {
  int i;
  for (i=0; i<n; i++) {
    if (strcmp(tab_soc[i].nome,nome)==0)
      return (tab_soc[i].codice_stato);
  }
  return -1;
}
```

Codice

```
int cercaCodiceStato (t_societa tab_soc[],
                      int n, char nome[]) {
  int i;
  for (i=0; i<n; i++) {
    if (strcmp(tab_soc[i].nome,nome)==0)
      return (tab_soc[i].codice_stato);
  }
  return -1;
}
```

Sotto problema di ricerca in tabella (corrispondenza stringa numero)

Codifica di immagine

- Formulazione: è data un'immagine di tipo "bitmap", rappresentata come matrice di 800x600 pixel (800 colonne, 600 righe)
- I pixel sono codificati con profondità 16 bit, ad ogni pixel è cioè associato un colore, con valore numerico compreso tra 0 e 2^{16}-1
- Ogni pixel può essere univocamente identificato mediante due coordinate **(r,c)**, indici di riga e di colonna **($0 \leq r < 600$, $0 \leq c < 800$)**

Codifica a punti e rettangoli

- L’immagine, originariamente memorizzata in un file binario, è stata successivamente convertita, in un file testo, secondo il formato seguente
  - L’immagine viene considerata come costituita dalla sovrapposizione di rettangoli e punti:
  - un punto corrisponde a un pixel
  - un rettangolo è l’insieme di tutti e soli i punti aventi coordinate di riga **r** e colonna **c,** tali che:
    - **r0 ≤ r ≤ r1**, **c0 ≤ c ≤ c1**

    con **r0, r1, c0 e c1** valori costanti

- Ad ogni rettangolo e/o punto vengono associati:
  - un colore (intero compreso tra 0 e 2^{16}-1)
  - un livello (intero compreso tra 0 e 255)
- Ogni punto o rettangolo di livello superiore va considerato sovrapposto ai livelli inferiori. Il colore di un pixel di coordinate **(r,c)** viene determinato dal punto o rettangolo di livello più alto che copra **(r,c)**

- Ogni punto è rappresentato, nel file, da una riga

  p <r> <c> <colore> <livello>
  - **<r>** e **<c>** sono le coordinate del punto
  - **<colore>** e **<livello>** (interi) rappresentano il colore e il livello del punto
- Ogni rettangolo è rappresentato da una riga

  r <r0> <r1> <c0> <c1> <colore> <livello>
  - **<r0>, <r1>, <c0>** e **<c1>** sono i limiti per le coordinate dei punti
  - **<colore>** e **<livello>** (interi) sono il colore e il livello di tutti i punti appartenenti al rettangolo

Elaborazioni richieste

- Leggere il file, il cui nome è ricevuto come primo argomento sulla linea di comando (argv[1])
- Eliminare punti e rettangoli invisibili (totalmente coperti da punti e/o rettangoli di livello più alto) e riscrivere l'immagine (stesso formato) su un secondo file, con nome ricevuto in argv[2]
- Calcolare il numero di pixel di colore uguale a un pixel di riferimento **P0**, le cui coordinate sono ricevute in argv[3] e argv[4]
- Calcolare quali sono i rettangoli e/o punti (visibili o invisibili) che coprono **P0**

Struttura dati

- Una tabella, per l'elenco di rettangoli (e punti):
  - vettore di struct, che rappresentano ognuna un rettangolo, con coordinate, colore e livello. I punti possono essere assimilati a rettangoli di dimensione minima
- Matrice di pixel:
  - per ogni pixel si possono memorizzare
    - livello e colore (determinati in base al rettangolo di livello più alto che copre il pixel)
    - rettangolo di livello più alto che copre il pixel (da cui si possono ricavare livello e colore)

  Si sceglie la seconda opzione.

Rappresentazione della struttura dati

Matrice di pixel

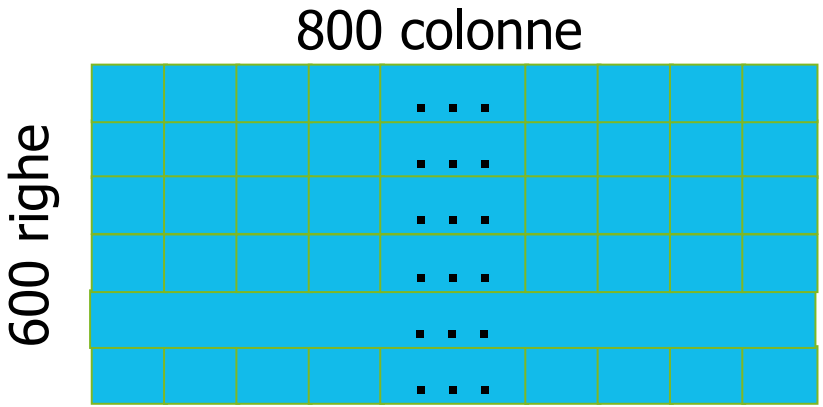
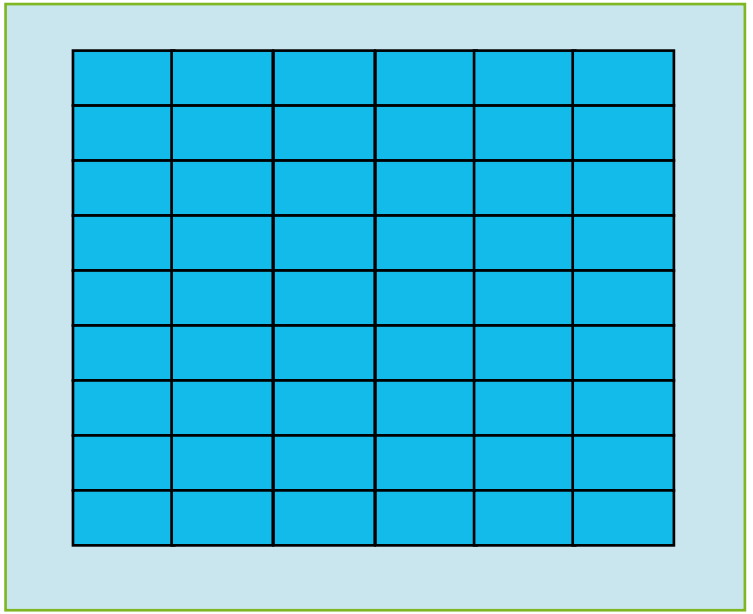
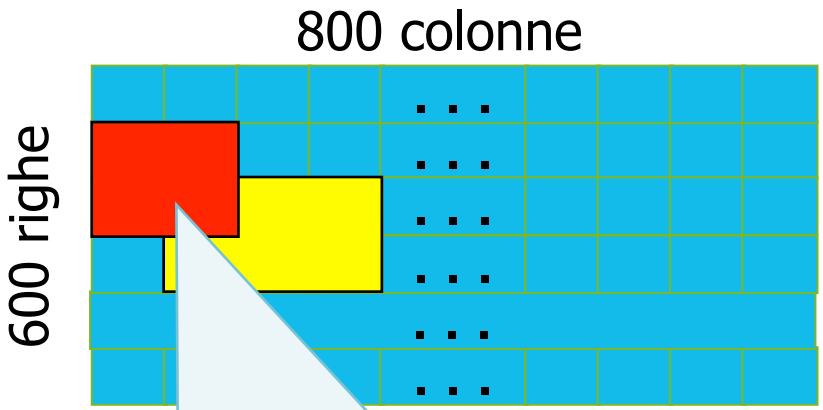


Tabella dei rettangoli



Matrice di pixel

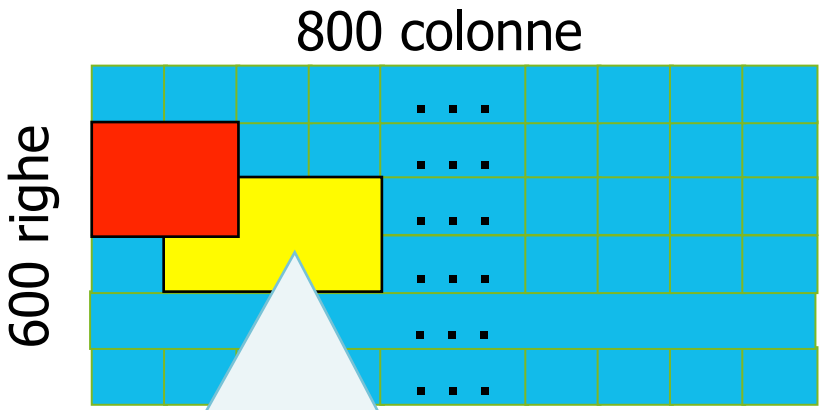


Rettangolo rosso, di livello 20, tra i punti (1,0) e (2,1)

Tabella dei rettangoli

1	2	0	1	20	
2	3	1	3	7	

Matrice di pixel



Rettangolo giallo, di livello 7, tra i punti (2,1) e (3,3)

Tabella dei rettangoli

1	2	0	1	20	
2	3	1	3	7	

Algoritmo

- Lettura dell’immagine e caricamento della prima tabella
- Caricamento della matrice di pixel:
  - può essere visto come un problema di massimo per ognuno dei pixel interessati
  - per ogni punto di ogni rettangolo si determina se il punto è visibile (ha il livello massimo per il pixel corrispondente) o meno
- Filtro sui rettangoli visibili:
  - si tratta di un problema di selezione, nel quale, per un dato rettangolo, vanno esaminati tutti i punti

- Conteggio dei pixel di colore uguale a **p0:**
  - una volta determinato il colore di p0, è un problema di selezione direttamente sulla matrice di pixel
- Calcolo dei rettangoli che coprono **p0:**
  - si tratta di un problema di ricerca/selezione, sull'elenco di rettangoli

Scomposizione in sottoproblemi

- I sottoproblemi sono ricavati in modo naturale dai singoli risultati richiesti:
  - filtro rettangoli:
    - selezione sul vettore dei rettangoli
    - come sottoproblema considera i punti appartenenti a un dato rettangolo
  - conteggio pixel di colore uguale a **p0:**
    - ricerca sulla matrice dei pixel
  - calcolo rettangoli che coprono **p0:**
    - selezione sul vettore di rettangoli

Codice

```c
#include <stdio.h>
#define NR 600
#define NC 800
#define MAXC 100
#define MAX_RETT 100

/* tipi struct per rettangoli */
typedef struct {
  int r0, r1, c0, c1;
  int livello;
  int colore;
} t_rettangolo;
/* variabili globali */
int nr, matr_pixel[NR][NC];
t_rettangolo rett[MAX_RETT];
```

```c
/* prototipi di funzioni (omessi) */
...

int main (int argc, char *argv[]) {
  FILE *fp;
  int i, colore, livello;
  int r, c, r0, r1, c0, c1;
  char riga[MAXC];

  fp = fopen(argv[1],"r");

  for (r=0; r<NR; r++)
    for (c=0; c<NC; c++)
      matr_pixel[r][c] = -1;
```

Codice

```
#include <stdio.h>
#define NR 600
#define NC 800
#define MAXC 100
#def

/* t
typedef struct {
  int r0, r1, c0, c1;
  int livello;
  int colore;
} t_rettangolo;
/* variabili globali */
int nr, matr_pixel[NR][NC];
t_rettangolo rett[MAX_RETT];
```

Inizializzazione matrice di pixel al valore -1

```
/* prototipi di funzioni (omessi) */
...

int main (int argc, char *argv[]) {
  FILE *fp;
  int i, colore, livello;
  int r, c, r0, r1, c0, c1;
  char riga[MAXC];

  fp = fopen(argv[1],"r");

  for (r=0; r<NR; r++)
    for (c=0; c<NC; c++)
      matr_pixel[r][c] = -1;
```

Codice

```
for (i=0; fgets(riga,MAXC,fp)!=NULL; i++);
  switch (riga[0]) {
    case 'p': case 'P':
      sscanf (riga,"%*c %d %d %d %d",
        &r0,&c0,&colore,&livello);
      r1 = r0; c1 = c0;
      break;
    case 'r': case 'R':
      sscanf (riga,"%*c %d %d %d %d %d %d",
        &r0,&r1,&c0,&c1,&colore,&livello);
      break;
  }
  rett[i].r0 = r0; rett[i].r1 = r1;
  rett[i].c0 = c0; rett[i].c1 = c1;
  rett[i].colore = colore;
  rett[i].livello = livello;
}
```

```
  nr = i;
  fclose(fp);
  coloraPixel();
  fp = fopen(argv[2],"w");
  scriviVisibili(fp);
  fclose(fp);
  r=atoi(argv[3]); c=atoi(argv[4]);
  colore = rett[matr_pixel[r][c]].colore;
  printf("%d pixel di colore %d\n",
    contaStessoColore(colore), colore);
  printf("%d rettangoli su pixel (%d,%d)\n",
    contaRettangoli(r,c), r, c);
  return 0;
}
```

Codice

```
for (i=0; fgets(riga,MAXC,fp)!=NULL; i++);
  switch (riga[0]) {
    case 'p': case 'P':
      sscanf (riga,"%*c %d %d %d %d",
        &r0,&c0,&colore,&livello);
      r1 = r0; c1 = c0;
      break;
    case 'r': case 'R':
      sscanf (riga,"%*c %d %d %d %d %d %d",
        &r0,&r1,&c0,&c1,&colore,&livello);
      break;
  }
  rett[i].r0 = r0; rett[i].r1 = r1;
  rett[i].c0 = c0; rett[i].c1 = c1;
  rett[i].colore = colore;
  rett[i].livello = livello;
}
```

```
  nr = i;
  fclose(fp);
  r=atoi(argv[3]); c=atoi(argv[4]);
  colore = rett[matr_pixel[r][c]].colore;
  printf("%d pixel di colore %d\n",
    contaStessoColore(colore), colore);
  printf("%d rettangoli su pixel (%d,%d)\n",
    contaRettangoli(r,c), r, c);
  return 0;
}
```

Lettura punti e rettangoli:
i punti sono convertiti in rettangoli di dimensione 1

Codice

```
for (i=0; fgets(riga,MAXC,fp)!=NULL; i++);
  switch (riga[0]) {
    case ‘p’: case ‘P’:
  }
  rett[i].r0 = r0; rett[i].r1 = r1;
  rett[i].c0 = c0; rett[i].c1 = c1;
  rett[i].colore = colore;
  rett[i].livello = livello;
}
```

Gli altri sotto-problemi sono risolti da funzioni:
- coloraPixel
- scriviVisibili
- contaStessoColore
- contaRettangoli

```
  nr = i;
  fclose(fp);
  coloraPixel();
  fp = fopen(argv[2],“w");
  scriviVisibili(fp);
  fclose(fp);
  r=atoi(argv[3]); c=atoi(argv[4]);
  colore = rett[matr_pixel[r][c]].colore;
  printf(“%d pixel di colore %d\n”,
    contaStessoColore(colore), colore);
  printf(“%d rettangoli su pixel (%d,%d)\n”,
    contaRettangoli(r,c), r, c);
  return 0;
}
```

Codice

```
void coloraPixel(void) {
  int i, livello;
  int p, r, c;

  for (i=0; i<nr; i++)
    for (r=rett[i].r0; r<rett[i].r1; r++)
      for (c=rett[i].c0; c<rett[i].c1; c++) {
        livello = rett[i].livello;
        p = matr_pixel[r][c];
        if (p<0 || livello > rett[p].livello)
          matr_pixel[r][c] = i;
      }
}
```

```
void scriviVisibili(FILE *fp) {
  int i, r, c, p, visibile;

  for (i=0; i<nr; i++) {
    visibile = 0;
    for (r=rett[i].r0; r<rett[i].r1; r++)
      for (c=rett[i].c0; c<rett[i].c1; c++) {
        p = matr_pixel[r][c];
        if (p == i)
          visibile = 1;
      }
    if (visibile) scriviRettangolo(fp,i);
  }
}
```

Codice

```
void scriviRettangolo(FILE *fp, int i) {
  int i, r0, r1, c0, c1;

  r0=rettangolo[i].r0; r1=rettangolo[i].r1;
  c0=rettangolo[i].c0; c1=rettangolo[i].c1;
  if ((r0 == r1) && (c0 == c1))
    fprintf (fp,"p %d %d",r0,c0);
  else
    fprintf (fp,"r %d %d %d %d",r0,r1,c0,c1);
  fprintf (fp," %d %d\n",rettangolo[i].livello,
    rettangolo[i].colore);
}
```

```
int contaStessoColore(int colore) {
  int num=0 ,r, c, p;

  for (r=0; r<NR; r++)
    for (c=0; c<NC; c++) {
      p = matr_pixel[r][c];
      if (colore == rett[p].colore)
        num++;
    }
  return num;
}
```

Codice

```
int contaRettangoli(int r, int c) {
  int num=0;
  for (i=0; i<nr; i++) {
    if ((rett[i].r0 <= r && ret[i].c0 <= c)
     && (rett[i].r1 >= r && ret[i].c1 >= c))
      num++;
  }
  return num;
}
```