



**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)
 - E' 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
- E' necessario prevedere (a livello di struttura dati) la memorizzazione di risultati intermedi

Esempio 1: 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:
 - 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

Dati in ingresso

- Il file 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 Chrysler
0 General_Motors
0 Chevrolet
```

F1

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

Elaborazioni richieste

- Leggere F0 e F1
 - Nomi 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
 - I codici sono compresi tra 0 e 99, quindi 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

Struttura dati

- 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
 - Possono essere lette e manipolate una alla volta
 - Non è opportuno salvare tutto il file!

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"

Codice dello stato

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:
 - Lettura di F0
 - Lettura di F1 e calcolo statistiche import export
 - Transazione è import o export?
 - Aggiornamento del bilancio dello stato
 - Ricerca dei massimi
 - Massimo surplus e massimo deficit
- Un ulteriore sottoproblema è costituito dalla conversione
nome società → codice stato
che può essere ricondotta a un problema di ricerca nella tabella delle società
 - Selezione su dati non numerici

Codice

```
#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[]
);
```

```
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 stati */
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[]
);
```

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

```
int main (int argc, char *argv[]) {
    FILE *fp;
    int nst, nsoc, codice, importo,
    int si se maxi maxe.

    /* lettura file F0 */
    fp = fopen("F0", "r");
    fscanf(fp, "%d", &nst, &nsoc);

    /* Inizializzazione nomi con stringa vuota */
    for (i=0; i<MAXSTATI; i++)
        strcpy(tab_st[i].nome, "");
```

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);
```

```
/* transazioni */
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);
```

```
/* transazioni */
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);
```

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

```
/* transazioni */
fp = fopen(argv[2], "r");

while (fscanf(fp,"%d%lf", &codice,&importo) != EOF) {
    if (codice == 1)
        se += importo;
    else
        si += importo;
    codice=cercaCodiceStato(tab_soc,nsoc,nome);
    tab_st[codice].bilancio += importo;
}

fclose(fp);
```

Transazioni

Non sono immagazzinate in vettore, ma sono gestite direttamente mentre vengono lette:

- sommatorie
- aggiornamento bilancio stato

```
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);
```

```
/* transazioni */  
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);
```

Filtro sui dati:

- se importo > 0 → export
- altrimenti → import

```
/* transazioni */  
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);  
}
```

Conversione nome-codice, fatta come
sottoproblema di ricerca

```
/* 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);
```

```
/* transazioni */  
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;
```

```
}
```

```
}
```

Cerca (indice dello) stato
con massimo export

```
/* 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);
```

```
}
```

Cerca (indice dello) stato
con massimo import

Codice

Stampa risultati finali

```
/* 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

```
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)