

Esercizio 1

Si consideri un processo industriale per il trattamento di materiali, composto da 10 fasi, in ognuna delle quali viene utilizzata una sostanza particolare. Per ogni fase sono disponibili un certo numero di sostanze, ognuna delle quali è caratterizzata da un determinato prezzo; ogni sostanza può inoltre essere incompatibile con altre sostanze utilizzabili nel processo, così che l'uso di una sostanza rende impossibile l'utilizzo di tutte le sostanze con essa incompatibili, in tutte le fasi del processo, precedenti e successive.

Si realizzi un programma per determinare per ogni fase la sostanza da utilizzare, in modo tale da minimizzare il costo del processo e senza che sostanze tra loro incompatibili siano utilizzate al suo interno. Il programma legge da file le informazioni relative alle sostanze disponibili, al loro prezzo (espresso tramite un numero intero inferiore a 1000), alla fase in cui sono utilizzabili, e alla lista delle eventuali incompatibilità. Il formato di tale file è il seguente:

- La prima riga contiene il numero di sostanze disponibili, sotto forma di un intero tra 1 e 100;
- Per ogni sostanza vi sono nel file:
 - Una riga contenente il nome della sostanza, il suo prezzo, la fase in cui può essere utilizzata, il numero di sostanze con cui è incompatibile, nel formato:

#nome prezzo fase numero_sostanze_incompatibili

- Tante righe quante sono le sostanze incompatibili, in ognuna delle quali è riportato il nome di una di essere. Si assume che la lista delle sostanze incompatibili contenga solo sostanze precedentemente lette da file.

SOLUZIONE:

```
/*  
*****  
SOSTANZE.C  
*****  
*/
```

Risolve il problema di trovare una serie di N elementi di costo minimo, non incompatibili tra loro.
L'elenco degli elementi, con il loro nome e le relative incompatibilità, viene letto da file.

```
*****  
*/
```

```
#include <stdio.h>  
  
#define N 10          /* numero fasi */  
#define MAX 100      /* max numero sostanze */  
  
typedef struct sostanza{  
    char    name[20];  
    int     phase,  
           price;  
}SOST;  
  
int     read_file( FILE *fin);  
void    solve( int phase);  
int     get_index( char *name);  
void    save_solution( void);  
  
SOST    vett[MAX];          /* elenco delle sostanze */  
  
char     incomp[MAX][MAX]; /* matrice di incompatibilità */  
  
int     solut[N+1],        /* soluzione corrente */  
        best_solut[N+1];  /* soluzione migliore */  
  
int     n_sost,  
        tot_price=0,  
        min_price=0x7FFF;
```

```

void main(void)
{
    char  name[20];
    FILE  *fin;
    int   i;

    printf( "Nome file di input: ");
    gets( name);
    if( (fin = fopen( name, "r")) == NULL)
    {
        printf( "Errore in apertura file di input \n");
        return;
    }

    if( read_file( fin))
    {
        printf( "Errore in lettura file di input \n");
        return;
    }

    solve( 1);
    if( min_price != 0)
    {
        for( i=1; i<N+1; i++)
            printf( "Fase %d - Sostanza %s \n", i, vett[best_solut[i]].name);
        printf( "Costo Totale %d\n", min_price);
    }
    else
        printf( "Non esiste soluzione \n");
}

```

```

/*****
                                READ_FILE
*****/

int read_file( FILE *fin)
{
    int   i,
          j,
          index,
          n_inc;
    char  name[20];

    fscanf( fin, "%d\n", &n_sost);

    for( i=0; i<n_sost; i++)
    {
        fscanf( fin, "%s", vett[i].name);
        fscanf( fin, "%d %d %d\n", &vett[i].phase, &vett[i].price, &n_inc);
        for( j=0; j<n_inc; j++)
        {
            fscanf( fin, "%s\n", name);
            index = get_index( name);
            incomp[i][index] = 1;
            incomp[index][i] = 1;
        }
    }

    return( 0);
}

```

```

/*****
                                SOLVE
*****/
void solve( int phase)
{
    int    i,
           j,
           is_inc;

    if( phase == N+1)
    {
        if( tot_price < min_price)
        {
            min_price = tot_price;
            save_solution();
        }
        return;
    }

    for( i=0; i<n_sost; i++)
    {
        if( vett[i].phase == phase)
        {
            is_inc = 0;
            for( j=1; j<phase; j++)
                if( incomp[solut[j]][i] == 1)
                    is_inc = 1;
            if( is_inc != 1)
            {
                solut[phase] = i;
                tot_price += vett[i].price;
                solve( phase+1);
                tot_price -= vett[i].price;
            }
        }
    }
}

```

```

/*****
                                GET_INDEX
*****/
int get_index( char *name)
{
    int    i;

    for( i=0; i<n_sost; i++)
    {
        if( strcmp( name, vett[i].name) == 0)
            return( i);
    }

    return( -1);
}

```

```
/******  
  
                SAVE_SOLUTION  
  
*****/  
void save_solution( void)  
{  
    int    i;  
  
    for( i=1; i<N+1; i++)  
        best_solut[i] = solut[i];  
}
```