# Esercizi d'esame

Raccolta di esercizi degli esami del corso di Informatica Generale 1

Marco Liverani

Corso di Laurea in Matematica
Facoltà di Scienze M.F.N.
Università degli studi di Roma Tre

Febbraio 2003

Ultimo aggiornamento: 19 febbraio 2003

# Esonero del 27 gennaio 1999

## Esercizio n.1

Leggere una lista $L$ di numeri interi $\{x_1, x_2, ..., x_n\}$. Letto in input un intero $X$ ($X \geq x_i,\ \forall i = 1, ..., n$) ripartire la lista $L$ in $k$ sotto-liste $L_1, L_2, ..., L_k$ tali che $\sum_{x_i \in L_j} x_i \leq X,\ \forall j = 1, ..., k$. Stampare le sotto-liste generate.

**Esempio.** Sia $L = \{3, 7, 1, 4, 2, 8, 4, 3, 2\}$ e sia $X = 10$. Allora $L_1 = \{3, 7\}$, $L_2 = \{1, 4, 2\}$, $L_3 = \{8\}$, $L_4 = \{4, 3, 2\}$.

### Soluzione

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

#define MAX 50

struct nodo {
  int info;
  struct nodo *next;
};

struct nodo *leggi_lista(void) {
  struct nodo *primo, *p;
  int a;

  printf("Inserisci gli elementi della lista (-1 per terminare):");
  primo = NULL;
  scanf("%d", &a);
  while (a != -1) {
    p = malloc(sizeof(struct nodo));
    p->info = a;
    p->next = primo;
    primo = p;
    scanf("%d", &a);
  }

  return(primo);
}

void stampa_lista(struct nodo *p) {
  while (p != NULL) {
```

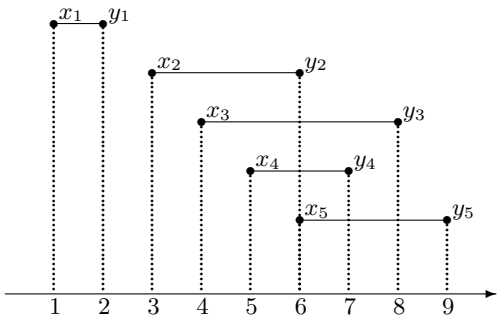
```
    printf("%d -> ", p->info);
    p = p->next;
  }
  printf("NULL\n");
  return;
}

struct nodo *suddividi(struct nodo **primo, int x) {
  struct nodo *p, *primo1, *prec;
  int somma;

  p = *primo;
  primo1 = *primo;
  somma = 0;
  prec = NULL;
  do {
    somma = somma + p->info;
    prec = p;
    p = p->next;
  } while ((p != NULL) && (somma + p->info <= x));

  prec->next = NULL;
  *primo = p;
  return(primo1);
}

int main(void) {
  struct nodo *primo, *lista[MAX];
  int i, j, x;

  primo = leggi_lista();
  scanf("%d", &x);

  i = 0;
  while (primo != NULL) {
    lista[i] = suddividi(&primo, x);
    i++;
  }

  for (j=0; j<i; j++) {
    stampa_lista(lista[j]);
  }

  return(1);
}
```

## Esercizio n.2

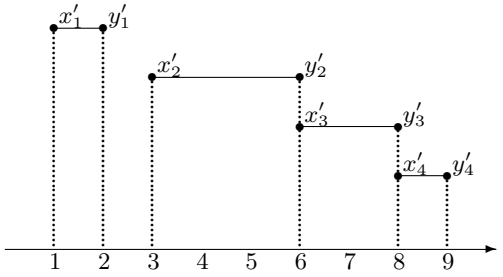
Leggere una lista di coppie di numeri interi $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ tali che $x_i < y_i, \forall i = 1, ..., n$; ogni coppia $(x_i, y_i)$ rappresenta un intervallo sulla retta. Riordinare gli elementi della lista in modo tale che $x_1 \leq x_2 \leq ... \leq x_n$. Costruire una nuova lista che rappresenti gli intervalli della prima lista, ma privi di intersezioni

(fatta eccezione per gli estremi degli intervalli); gli eventuali intervalli nulli (tali che $x_i \geq y_i$) prodotti dalla rielaborazione degli intervalli originali devono essere eliminati.

**Esempio.** Siano $(x_1, y_1) = (1, 2)$, $(x_2, y_2) = (3, 6)$, $(x_3, y_3) = (4, 8)$, $(x_4, y_4) = (5, 7)$, $(x_5, y_5) = (6, 9)$ gli intervalli originali riordinati in modo tale che $x_i \leq x_{i+1} \forall i = 1, ..., n-1$. La figura seguente rappresenta gli intervalli:




Dopo l'elaborazione della lista viene prodotta una nuova lista composta di soli quattro nodi: $(x'_1, y'_1) = (1, 2)$, $(x'_2, y'_2) = (3, 6)$, $(x'_3, y'_3) = (6, 8)$, $(x'_4, y'_4) = (8, 9)$. Come risulta dalla figura seguente i nuovi intervalli sono privi di intersezione e l'intervallo $(x_4, y_4)$ della lista originale è stato eliminato perché durante l'elaborazione si era ridotto solo ad un punto.




## Soluzione

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

struct nodo {
  int x;
  int y;
  struct nodo *next;
};

struct nodo *leggi_lista(void) {
  struct nodo *primo, *p;
  int x, y, i, n;

  printf("Numero di elementi: ");
  scanf("%d", &n);
```

```
  primo = NULL;
  for (i=0; i<n; i++) {
    printf("inserisci x%d e y%d: ", i, i);
    scanf("%d %d", &x, &y);
    p = malloc(sizeof(struct nodo));
    p->x = x;
    p->y = y;
    p->next = primo;
    primo = p;
  }
  return(primo);
}

void stampa_lista(struct nodo *p) {
  while (p != NULL) {
    printf("(%d,%d) ", p->x, p->y);
    p = p->next;
  }
  printf("\n");
  return;
}

void ordina(struct nodo *primo) {
  int appx, appy, flag;
  struct nodo *p;

  flag = 1;
  while (flag == 1) {
    flag = 0;
    p = primo;
    while (p->next != NULL) {
      if (p->x > (p->next)->x) {
        appx = p->x;
        appy = p->y;
        p->x = (p->next)->x;
        p->y = (p->next)->y;
        (p->next)->x = appx;
        (p->next)->y = appy;
        flag = 1;
      }
      p = p->next;
    }
  }
  return;
}

struct nodo *elabora(struct nodo *primo) {
  struct nodo *p, *p2, *primo2;
  p = primo;
  primo2 = NULL;
  while (p != NULL) {

    if (p->x < p->y) {
      p2 = malloc(sizeof(struct nodo));
```

```
      p2->x = p->x;
      p2->y = p->y;
      p2->next = primo2;
      primo2 = p2;
    }

    if ((p->next != NULL) && ((p->next)->x < p->y)) {
      (p->next)->x = p->y;
      if ((p->next)->y < (p->next)->x) {
        (p->next)->y = (p->next)->x;
      }
    }

    p = p->next;
  }
  return(primo2);
}

int main(void) {
  struct nodo *primo, *primo2;

  primo = leggi_lista();
  ordina(primo);
  primo2 = elabora(primo);
  stampa_lista(primo2);
  return(1);
}
```

# Esame del 5 febbraio 1999

## Esercizio n.1

Generare una matrice quadrata di dimensione $n$ di numeri interi casuali. Scrivere una funzione che restituisca 1 se la matrice è un quadrato magico e zero altrimenti. Una matrice $n \times n$ è un quadrato magico se la somma degli elementi su ogni riga, su ogni colonna e sulle due diagonali principali è costante.

**Esempio.** La seguente matrice $3 \times 3$ è un quadrato magico:

$$\begin{pmatrix} 6 & 7 & 2 \\ 1 & 5 & 9 \\ 8 & 3 & 4 \end{pmatrix}$$

Infatti la somma degli elementi di ogni riga, di ogni colonna e delle due diagonali principali è 15. La seguente matrice invece non è un quadrato magico:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

### Soluzione

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

#define MAX_N 10

int genera(int *m) {
  int n, i, j;

  srand((unsigned) time(NULL));
  n = rand() % (MAX_N-2) + 2;

  for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
      *(m+i*MAX_N+j) = rand() % 100;
    }
  }
  return(n);
}
```

```
void stampa_matrice(int *m, int n) {
  int i, j;
  for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
      printf("%3d", *(m+i*MAX_N+j));
    }
    printf("\n");
  }
  return;
}

int verifica(int *m, int n) {
  int i, j, r, s, somma;

  r = 1;
  somma = 0;
  for (j=0; j<n; j++) {
    somma = somma + *(m+j);
  }

  /* controllo le righe */
  i = 1;
  while (i<n && r==1) {
    s = 0;
    for (j=0; j<n; j++) {
      s = s + *(m+i*MAX_N+j);
    }
    if (s != somma) {
      r = 0;
    }
    i++;
  }

  /* controllo le colonne */
  j = 0;
  while (j<n && r==1) {
    s = 0;
    for (i=0; i<n; i++) {
      s = s + *(m+i*MAX_N+j);
    }
    if (s != somma) {
      r = 0;
    }
    j++;
  }

  /* controllo la prima diagonale */
  s = 0;
  i = 0;
  while (i<n && r==1) {
    s = s + *(m+i*MAX_N+i);
    i++;
  }
  if (s != somma) {
```

```
    r = 0;
  }

  /* controllo la seconda diagonale */
  s = 0;
  i = 0;
  while (i<n && r==1) {
    s = s + *(m + i*MAX_N + (n-i-1));
    i++;
  }
  if (s != somma) {
    r = 0;
  }

  return(r);
}

int main(void) {
  int M[MAX_N][MAX_N], n;

  n = genera(&M[0][0]);
  if (verifica(&M[0][0],n) == 1) {
    printf("La matrice\n");
    stampa_matrice(&M[0][0], n);
    printf("e' un quadrato magico.\n");
  } else {
    printf("La matrice\n");
    stampa_matrice(&M[0][0], n);
    printf("non e' un quadrato magico.\n");
  }
  return(1);
}
```

# Esercizio n.2

Riceviamo in input una sequenza di schede che indicano l'importo di una certa spesa ed il periodo dell'anno in cui è stata compiuta. Dopo aver memorizzato tutte le informazioni lette in input in una struttura dati dinamica, calcolare l'importo delle spese per ogni mese dell'anno, senza fare uso di array o matrici. Stampare i mesi e l'importo della spesa relativa in ordine decrescente di spesa.

**Esempio.** Supponiamo di ricevere in input le informazioni riportate nella seguente tabella:

| Inizio | Fine | Importo |
|---|---|---|
| 1 | 3 | 90 |
| 2 | 6 | 50 |
| 1 | 1 | 20 |
| 5 | 7 | 30 |

La prima riga indica che da gennaio a marzo sono stati spesi 90 milioni per una determinata voce di spesa (quindi 30 milioni al mese), da febbraio a giugno 50

milioni per un'altra voce di spesa (quindi 10 milioni al mese), ecc. L'output del programma dovrà quindi essere il seguente:

| Mese | Importo |
|---|---|
| 1 | 50 (= 30 + 20) |
| 2 | 40 (= 30 + 10) |
| 3 | 40 (= 30 + 10) |
| 5 | 20 (= 10 + 10) |
| 6 | 20 (= 10 + 10) |
| 4 | 10 (= 10) |
| 7 | 10 (= 10) |

## Soluzione

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

struct nodo {
  int m1, m2, importo;
  struct nodo *next;
};

struct nodo2 {
  int m, importo;
  struct nodo2 *next;
};

struct nodo *input(int n) {
  int i, m1, m2, importo;
  struct nodo *p, *primo;

  primo = NULL;
  for (i=0; i<n; i++) {
    scanf("%d %d %d", &m1, &m2, &importo);
    p = malloc(sizeof(struct nodo));
    p->m1 = m1;
    p->m2 = m2;
    p->importo = importo;
    p->next = primo;
    primo = p;
  }
  return(primo);
}

struct nodo2 *trovamese(struct nodo2 *primo, int m) {
  struct nodo2 *p;

  p = primo;
  while (p != NULL && p->m != m) {
    p = p->next;
  }
  return(p);
}
```

```
struct nodo2 *calcola(struct nodo *primo) {
  struct nodo *p;
  struct nodo2 *p2, *primo2;
  int i;

  p = primo;
  primo2 = NULL;

  while (p != NULL) {
    for (i=p->m1; i<=p->m2; i++) {
      p2 = trovamese(primo2, i);
      if (p2 == NULL) {
        p2 = malloc(sizeof(struct nodo2));
        p2->importo = 0;
        p2->m = i;
        p2->next = primo2;
        primo2 = p2;
      }
      p2->importo = p2->importo + p->importo/(p->m2 - p->m1 + 1);
    }
    p = p->next;
  }

  return(primo2);
}

void scambia(struct nodo2 *p1, struct nodo2 *p2) {
  int m, importo;

  m = p1->m;
  importo = p1->importo;
  p1->m = p2->m;
  p1->importo = p2->importo;
  p2->m = m;
  p2->importo = importo;
  return;
}

void bubble_sort(struct nodo2 *primo) {
  struct nodo2 *p, *ultimo;
  int flag;

  flag = 1;
  ultimo = NULL;
  while (flag == 1) {
    flag = 0;
    p = primo;
    while (p->next != ultimo) {
      if (p->importo < (p->next)->importo) {
        scambia(p, p->next);
        flag = 1;
      }
      p = p->next;
```

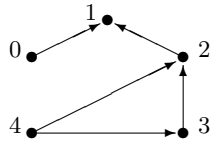
```
    }
    ultimo = p;
  }
  return;
}

void stampa(struct nodo2 *primo2) {
  printf("mese\timporto\n");
  while (primo2 != NULL) {
    printf("%d\t%d\n", primo2->m, primo2->importo);
    primo2 = primo2->next;
  }
  return;
}

int main(void) {
  struct nodo *primo;
  struct nodo2 *primo2;
  int n;

  scanf("%d", &n);
  primo = input(n);
  primo2 = calcola(primo);
  bubble_sort(primo2);
  stampa(primo2);
  return(1);
}
```

# Esame del 23 febbraio 1999

## Esercizio n.1

Leggere in input un grafo orientato $G = (V, E)$ e rappresentarlo mediante liste di adiacenza. Implementare una funzione che stampi tutti i vertici "sorgente" e tutti i vertici "pozzo" di $G$. Un vertice $v \in V$ è una sorgente se ha solo spigoli uscenti e nessuno spigolo entrante; è un pozzo se ha solo spigoli entranti e nessuno spigolo uscente.

**Esempio.** Si consideri il seguente grafo $G$ orientato:




I vertici 0 e 4 sono sorgenti, mentre il vertice 1 è un pozzo.

### Soluzione

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

#define MAX 100

struct nodo {
  int info;
  struct nodo *next;
};

struct nodo *leggi_lista(void) {
  int a;
  struct nodo *p, *primo;

  primo = NULL;
  scanf("%d", &a);
  while (a != -1) {
    p = malloc(sizeof(struct nodo));
    p->info = a;
    p->next = primo;
    primo = p;
```

```
    scanf("%d", &a);
  }
  return(primo);
}

int leggi_grafo(struct nodo **l) {
  int i, n;

  printf("Numero di vertici: ");
  scanf("%d", &n);
  for (i=0; i<n; i++) {
    printf("%d: ", i);
    *(l+i) = leggi_lista();
  }
  return(n);
}

int *calcola_grado(struct nodo **l, int n) {
  int *m, i;
  struct nodo *p;

  /* alloco dinamicamente una matrice m di n*2 interi */
  m = malloc(sizeof(int)*2*n);
  if (m == NULL) {
    printf("Errore: memoria insufficiente.\n");
    exit(-1);
  }

  /* azzero la matrice */
  for (i=0; i<n; i++) {
    *(m+i*2+0) = 0;
    *(m+i*2+1) = 0;
  }

  /* scorro le liste di adiacenza di ogni vertice di G ed eseguo
     il calcolo. */
  for (i=0; i<n; i++) {
    p = *(l+i);
    while (p != NULL) {

      /* incremento il numero di spigoli uscenti da i */

      *(m+i*2) += 1;

      /* p->info e' adiacente ad i: incremento il numero di spigoli
         entranti in p->info. */

      *(m + (p->info)*2 + 1) += 1;

      p = p->next;
    }
  }
  return(m);
}
```

```
void pozzi_e_sorgenti(int *m, int n) {
  int i;

  printf("Sorgenti di G: ");
  for (i=0; i<n; i++) {
    if (*(m+2*i+1) == 0) {
      printf("%d ", i);
    }
  }

  printf("\nPozzi di G: ");
  for (i=0; i<n; i++) {
    if (*(m+2*i) == 0) {
      printf("%d ", i);
    }
  }
  printf("\n");
  return;
}

int main(void) {
  struct nodo *liste[MAX];
  int n, *matrice;

  n = leggi_grafo(&liste[0]);
  matrice = calcola_grado(&liste[0], n);
  pozzi_e_sorgenti(matrice, n);
  return(1);
}
```

# Esercizio n.2

Leggere in input un grafo orientato $G = (V, E)$ rappresentarlo mediante una matrice di adiacenza. Dato un vertice $v \in V$ verificare che $v$ abbia un solo predecessore ($\exists!\ u \in V$ t.c. $(u, v) \in E$). In tal caso costruire il grafo $G'$ (mediante liste di adiacenza) ottenuto da $G$ collegando il predecessore di $v$ a tutti i successori di $v$ stesso e sconnettendo dal grafo il vertice $v$.

**Esempio.** Si consideri il seguente grafo orientato in fig. 1; se $v = 2$ il grafo $G'$ ottenuto da $G$ è quello riportato in fig. 2:

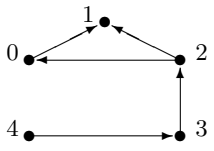



**Fig. 1**

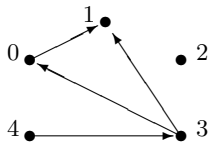



**Fig. 2**

## Soluzione

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

```
#define MAX 50

struct nodo {
  int info;
  struct nodo *next;
};

int leggi_grafo(int *m) {
  int i, j, n;

  printf("Numero di vertici: ");
  scanf("%d", &n);

  for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
      *(m+i*MAX+j) = 0;
    }
  }

  for (i=0; i<n; i++) {
    printf("Vertici adiacenti a %d (-1 per terminare): ", i);
    scanf("%d", &j);
    while (j != -1) {
      *(m+i*MAX+j) = 1;
      scanf("%d", &j);
    }
  }
  return(n);
}

void stampa_grafo(struct nodo **l, int n) {
  int i;
  struct nodo *p;

  for (i=0; i<n; i++) {
    p = *(l+i);
    printf("%d: ", i);
    while (p != NULL) {
      printf("%d -> ", p->info);
      p = p->next;
    }
    printf("NULL\n");
  }
  return;
}

int verifica(int *m, int n, int v) {
  int i, predecessore, npred;

  npred = 0;
  predecessore = -1;

  for (i=0; i<n; i++) {
```

```
    if (i != v && *(m+i*MAX+v) == 1) {
      predecessore = i;
      npred++;
    }
  }

  if (npred == 1)
    return(predecessore);
  else
    return(-1);
}

void elimina_vertice(int predecessore, int v, int *m, int n) {
  int i;

  for (i=0; i<n; i++) {
    if (*(m + v*MAX + i) == 1) {
      *(m + predecessore*MAX + i) = 1;
      *(m + v*MAX + i) = 0;
    }
  }
  *(m + predecessore*MAX + v) = 0;
  return;
}

void nuovo_grafo(int *m, struct nodo **l, int n) {
  int i, j;
  struct nodo *p;

  for (i=0; i<n; i++) {
    *(l+i) = NULL;
    for (j=0; j<n; j++) {
      if (*(m + i*MAX + j) == 1) {
        p = malloc(sizeof(struct nodo));
        p->info = j;
        p->next = *(l+i);
        *(l+i) = p;
      }
    }
  }
  return;
}

int main(void) {
  int m[MAX][MAX], n, v, pred;
  struct nodo *lista[MAX];

  n = leggi_grafo(&m[0][0]);
  printf("Inserisci il vertice v: ");
  scanf("%d", &v);

  pred = verifica(&m[0][0], n, v);

  if (pred >= 0) {
```

```
    elimina_vertice(pred, v, &m[0][0], n);
    nuovo_grafo(&m[0][0], &lista[0], n);
    stampa_grafo(&lista[0], n);
  } else {
    printf("Il predecessore di %d non e' unico.\n");
  }
  return(1);
}
```

# Esame del 18 giugno 1999

## Esercizio n.1

Nel database di una società telefonica le informazioni sono contenute su tre tabelle principali:

***Clienti:*** è una tabella con due colonne: nella prima c'è un numero identificativo del cliente e nella seconda il suo numero telefonico.

***Traffico:*** ha cinque colonne che servono ad archiviare i dati relativi alle telefonate effettuate da tutti i clienti. La prima colonna contiene la durata della telefonata in secondi, la seconda l'ora di inizio della telefonata, la terza i minuti dell'ora di inizio, la quarta il numero chiamato e la quinta il numero identificativo del cliente che ha compiuto la telefonata.

***Tariffe:*** è una tabella con cinque colonne e serve ad archiviare le informazioni relative alle tariffe vigenti per ogni fascia oraria. La prima e la seconda colonna rappresentano l'ora ed i minuti di inizio della fascia oraria, la terza e la quarta l'ora ed i minuti di termine della fascia oraria e la quinta colonna il costo in lire per ogni minuto di telefonata.

Per semplicità supponiamo che la normativa in vigore dica che se la telefonata viene iniziata in una determinata fascia oraria, allora si teve applicare la tariffa relativa a tale fascia all'intera telefonata (anche se ha "sconfinato" in una fascia oraria adiacente).

Lette in input le tre tabelle (matrici), si calcoli per ogni cliente il costo complessivo (la somma) delle telefonate effettuate.

### Soluzione

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

#define MAX 1000

int leggi_mat(int *m, int colonne) {
  int i, j, righe;

  printf("n. di righe: ");
  scanf("%d", &righe);
  printf("inserisci %d valori per %d righe:\n", colonne, righe);
```

```
  for (i=0; i<righe; i++)
    for (j=0; j<colonne; j++)
      scanf("%d", m+i*colonne+j);
  return(righe);
}

int main(void) {
  int clienti[MAX][2], traffico[MAX][5], tariffe[24][5];
  int spesa[MAX][2], costo, durata, min_tel, min_start, min_stop;
  int i, j, k, n_clienti, n_chiamate, n_tariffe;

  n_clienti  = leggi_mat(&clienti[0][0], 2);
  n_chiamate = leggi_mat(&traffico[0][0], 5);
  n_tariffe  = leggi_mat(&tariffe[0][0], 5);

  for (i=0; i<n_clienti; i++) {
    spesa[i][0] = clienti[i][0];
    spesa[i][1] = 0;
    for (j=0; j<n_chiamate; j++) {
      if (traffico[j][4] == clienti[i][0]) {
        costo = 0;
        durata = (traffico[j][0] + 59) / 60;
        min_tel = traffico[j][1]*60+traffico[j][2];
        for (k=0; k<n_tariffe; k++) {
          min_start = tariffe[k][0]*60+tariffe[k][1];
          min_stop  = tariffe[k][2]*60+tariffe[k][3];
          if ((min_start <= min_tel) && (min_tel <= min_stop))
            costo = tariffe[k][4] * durata;
        }
        spesa[i][1] += costo;
      }
    }
  }

  printf("Cliente\tSpesa\n-------\t-------\n");
  for (i=0; i<n_clienti; i++)
    printf("%7d\t%7d\n", spesa[i][0], spesa[i][1]);

  return(1);
}
```

# Esercizio n.2

Letti in input due grafi $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$, costruire il grafo unione $G = (V_1 \cup V_2, E_1 \cup E_2)$. Stampare le liste di adiacenza del grafo $G$. Si noti che $V_1$ e $V_2$ sono due insiemi finiti qualsiasi di numeri interi. Quindi ad esempio è possibile che $V_1 = \{1, 2, 3, 4\}$ e $V_2 = \{2, 4, 6, 8, 10\}$.

## Soluzione

```
#include <stdlib.h>
```

```
#include <stdio.h>

#define MAX 100

struct nodo {
   int info;
   struct nodo *next;
};

struct nodo *leggi_lista(void) {
   int a;
   struct nodo *primo, *p;

   printf("Lista dei vertici adiacenti (-1 per terminare): ");
   scanf("%d", &a);
   primo = NULL;
   while (a != -1) {
      p = malloc(sizeof(struct nodo));
      p->info = a;
      p->next = primo;
      primo = p;
      scanf("%d", &a);
   }
   return(primo);
}

int leggi_grafo(struct nodo **l, int *v) {
   int n, i;

   printf("Numero di vertici: ");
   scanf("%d", &n);
   for (i=0; i<n; i++) {
      printf("Etichetta del vertice %d: ", i);
      scanf("%d", v+i);
      *(l+i) = leggi_lista();
   }
   return(n);
}

void stampa(struct nodo **l, int *v, int n) {
   int i;
   struct nodo *p;

   for (i=0; i<n; i++) {
      printf("%d: ", *(v+i));
      p = *(l+i);
      while (p != NULL) {
         printf("%d --> ", p->info);
         p = p->next;
      }
      printf("NULL\n");
   }
   printf("\n");
   return;
```

```
}

struct nodo *copia_lista(struct nodo *p) {
   struct nodo *q, *primo;

   primo = NULL;
   while (p != NULL) {
      q = malloc(sizeof(struct nodo));
      q->info = p->info;
      q->next = primo;
      primo = q;
      p = p->next;
   }
   return(primo);
}

struct nodo *fondi_liste(struct nodo *la, struct nodo *lb) {
   struct nodo *p, *q;

   while (lb != NULL) {
      p = la;
      while (p != NULL && p->info != lb->info)
         p = p->next;
      if (p == NULL) {
         q = malloc(sizeof(struct nodo));
         q->info = lb->info;
         q->next = la;
         la = q;
      }
      lb = lb->next;
   }

   return(la);
}

int unione(struct nodo **l1, int *v1, int n1,
           struct nodo **l2, int *v2, int n2,
           struct nodo **l3, int *v3) {
   int i, j, n3;

   n3 = n1;
   for (i=0; i<n1; i++) {
      *(v3+i) = *(v1+i);
      *(l3+i) = copia_lista(*(l1+i));
   }

   for (i=0; i<n2; i++) {
      j = 0;
      while (j < n1 && *(v2+i) != *(v1+j))
         j++;
      if (j == n1) {
         *(v3+n3) = *(v2+i);
         *(l3+n3) = copia_lista(*(l2+i));
         n3++;
```

```
      } else {
         *(l3+j) = fondi_liste(*(l1+j), *(l2+i));
      }
   }
   return(n3);
}

int main(void) {
   int n1, n2, n3, v1[MAX], v2[MAX], v3[MAX];
   struct nodo *l1[MAX], *l2[MAX], *l3[MAX];

   n1 = leggi_grafo(&l1[0], &v1[0]);
   n2 = leggi_grafo(&l2[0], &v2[0]);
   n3 = unione(&l1[0], &v1[0], n1, &l2[0], &v2[0], n2, &l3[0],
        &v3[0]);
   stampa(&l1[0], &v1[0], n1);
   stampa(&l2[0], &v2[0], n2);
   stampa(&l3[0], &v3[0], n3);
   return(1);
}
```

# Esame del 9 luglio 1999

## Esercizio n.1

Data in input una sequenza di $n$ numeri in virgola mobile, memorizzarla in un array. Scorrendo l'array costruire un albero binario completo utilizzando delle strutture costituite da tre campi: il numero, un puntatore al figlio sinistro ed un puntatore al figlio destro. L'albero deve essere costruito assumendo che gli elementi dell'array sono tali che l'elemento $i$-esimo ha come figlio sinistro l'elemento di indice $2i$ e come figlio destro l'elemento $2i + 1$.

### Soluzione

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

#define MAX 100

struct nodo {
  float info;
  struct nodo *fs;
  struct nodo *fd;
};

int leggi_array(float *x) {
  int n, i;

  printf("Numero di elementi: ");
  scanf("%d", &n);
  for (i=1; i<=n; i++) {
    scanf("%f", x+i);
  }
  return(n);
}

struct nodo *crea_nodo(int i, float *x, int n) {
  struct nodo *r;

  r = malloc(sizeof(struct nodo));
  r->info = *(x+i);
  if (2*i <= n) {
    r->fs = crea_nodo(2*i, x, n);
```

```
  } else {
    r->fs = NULL;
  }
  if (2*i+1 <= n) {
    r->fd = crea_nodo(2*i+1, x, n);
  } else {
    r->fd = NULL;
  }
  return(r);
}

void stampa_nodo(struct nodo *p) {
  if (p != NULL) {
    printf("Nodo: %f\n", p->info);
    if (p->fs) {
      printf("Figlio sinistro: %f\n", (p->fs)->info);
    }
    if (p->fd) {
      printf("Figlio destro: %f\n", (p->fd)->info);
    }
    stampa_nodo(p->fs);
    stampa_nodo(p->fd);
  }
  return;
}

int main(void) {
  float v[MAX], n;
  struct nodo *radice;

  n = leggi_array(&v[0]);
  radice = crea_nodo(1, &v[0], n);
  stampa_nodo(radice);
  return(1);
}
```

# Esercizio n.2

Leggere in input due numeri naturali in base 2 e memorizzarne le cifre (0 o 1) in due array distinti. Calcolare e stampare la somma binaria dei due numeri.

## Soluzione

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

#define MAX 32

int leggi_array(int *x) {
  int n, i;
```

```
  printf("numero di cifre: ");
  scanf("%d", &n);
  for (i=0; i<n; i++)
    scanf("%d", x+n-i-1);
  for (i=n; i<MAX; i++)
    *(x+i) = 0;
  return(n);
}

int somma(int *u, int n, int *v, int m, int *w) {
  int i, riporto = 0, max;

  if (n>m)
    max = n;
  else
    max = m;
  for (i=0; i<max; i++) {
    *(w+i) = *(u+i) + *(v+i) + riporto;
    if (*(w+i) > 1) {
      *(w+i) = 0;
      riporto = 1;
    } else {
      riporto = 0;
    }
  }
  if (riporto == 1) {
    *(w+i) = riporto;
    max++;
  }
  return(max);
}

void stampa_array(int *x, int n) {
  int i;

  for (i=n-1; i>=0; i--)
    printf("%d", *(x+i));
  printf("\n");
  return;
}

int main(void) {
  int n, m, k, u[MAX], v[MAX], w[MAX];

  n = leggi_array(&u[0]);
  m = leggi_array(&v[0]);
  k = somma(&u[0], n, &v[0], m, &w[0]);
  stampa_array(&u[0], k);
  stampa_array(&v[0], k);
  stampa_array(&w[0], k);
  return(1);
}
```

# Esame del 17 settembre 1999

## Esercizio n.1

Generare una matrice $M$ di $n \times n$ numeri casuali. Leggere in input una sequenza $X$ di $2k$ numeri interi tali che $X_i \in \{-1, 0, 1\} \; \forall \; i = 0, 1, ..., 2k-1$. Partendo dall'elemento $M_{0,0}$ della matrice, ogni coppia di elementi di $X$, $(x_i, x_{i+1}), i = 0, 2, 4, ..., 2k-2$, rappresenta uno spostamento sulla matrice $M$.

Stampare l'elemento su cui si ferma la sequenza di spostamenti $X$ dopo le $k$ "mosse". Se una delle mosse è "proibita" perché porterebbe ad uscire dalla matrice $M$, allora deve essere stampato un messaggio di errore ed il programma deve interrompere la sua esecuzione.

### Soluzione

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

#define MAX_N 20
#define MAX_X 30

int genera(int *M) {
  int n, i, j;

  srand((unsigned)time(NULL));
  n = rand() % MAX_N;
  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
      *(M+i*MAX_N+j) = rand()%100;
  return(n);
}

int leggi_array(int *X) {
  int n, i;

  printf("numero di mosse: ");
  scanf("%d", &n);
  n *= 2;
  for (i=0; i<n; i++)
    scanf("%d", X+i);
```

```
  return(n);
}

int main(void) {
  int M[MAX_N][MAX_N],X[MAX_X];
  int k, n, m, i, x=0, y=0;

  n = genera(&M[0][0]);
  printf("E' stata generata una matrice %dx%d.\n", n, n);
  m = leggi_array(&X[0]);

  for (i=0; i<m; i+=2) {
    if (x+X[i]<0 || x+X[i]>=n || y+X[i+1]<0 || y+X[i+1]>=n) {
      printf("Errore: sei uscito dalla matrice!\n\n");
      exit(-1);
    } else {
      x += X[i];
      y += X[i+1];
    }
  }
  printf("Il cammino sulla matrice e' terminato sull'elemento ");
  printf("M[%d][%d] = %d.\n", x, y, M[x][y]);
  return(1);
}
```

# Esercizio n.2

Letta in input una sequenza di $n$ numeri interi, memorizzarla in una lista. Generare una sequenza di $m$ numeri casuali memorizzandoli in una seconda lista.

Per ogni nodo della seconda lista memorizzare in uno dei suoi campi il numero di occorrenze dello stesso elemento nella prima lista. Se un elemento non è presente nella prima lista, allora deve essere eliminato dalla seconda. Stampare in output la seconda lista al termine dell'elaborazione.

## Soluzione

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

struct nodo {
  int info;
  struct nodo *next;
};

struct nodo1 {
  int info, n;
  struct nodo1 *next;
};

struct nodo *leggi_lista(void) {
```

```
  int n, x, i;
  struct nodo *p, *primo;

  printf("Numero di elementi: ");
  scanf("%d", &n);
  primo = NULL;
  printf("Inserisci %d numeri minori di 100:\n", n);
  for (i=0; i<n; i++) {
    scanf("%d", &x);
    p = malloc(sizeof(struct nodo));
    p->info = x;
    p->next = primo;
    primo = p;
  }
  return(primo);
}

struct nodo1 *genera_lista(void) {
  int m, x, i;
  struct nodo1 *p, *primo = NULL;

  srand((unsigned)time(NULL));
  m = rand() % 100;
  for (i=0; i<m; i++) {
    x = rand() %  100;
    p = malloc(sizeof(struct nodo1));
    p->info = x;
    p->n = 0;
    p->next = primo;
    primo = p;
  }
  return(primo);
}

void stampa_lista(struct nodo1 *primo1) {
  while (primo1 != NULL) {
    printf("(%d, %d) -->", primo1->info, primo1->n);
    primo1 = primo1->next;
  }
  printf("NULL\n\n");
  return;
}

int main(void) {
  struct nodo *primo, *p;
  struct nodo1 *primo1, *p0, *p1;

  primo = leggi_lista();
  primo1 = genera_lista();
  stampa_lista(primo1);

  p0 = NULL;
  p1 = primo1;
  while (p1 != NULL) {
```

```
    p = primo;
    while (p != NULL) {
      if (p1->info == p->info)
        p1->n += 1;
      p = p->next;
    }
    if (p1->n == 0) {
      if (p0 == NULL) {
        primo1 = p1->next;
        free(p1);
        p1 = primo1;
      } else {
        p0->next = p1->next;
        free(p1);
        p1 = p0->next;
      }
    } else {
      p0 = p1;
      p1 = p1->next;
    }
  }
  stampa_lista(primo1);
  return(1);
}
```

# Esonero del 20 novembre 1999

## Esercizio n.1

Letta in input una sequenza di $2n$ numeri interi, memorizzarla in un array. I primi due elementi della sequenza rappresentano rispettivamente il numero di righe e di colonne di una matrice $M$. I restanti elementi rappresentano, in formato compresso, una sequenza di numeri che deve essere inserita nella matrice $M$. Per decodificare la sequenza i numeri dell'array devono essere considerati a coppie $(x, y)$: il primo elemento rappresenta il numero di volte in cui il secondo elemento deve essere inserito in celle adiacenti sulla stessa riga della matrice $M$. Costruire la matrice e, al termine della costruzione, stamparne il contenuto.

**Esempio.** Sia $V = (3, 5, 2, 1, 2, 17, 1, 3, 4, 8, 1, 6, 3, 7, 2, 5)$. Allora la matrice $M$ sarà costituita da 3 righe e 5 colonne:

$$M = \begin{pmatrix} 1 & 1 & 17 & 17 & 3 \\ 8 & 8 & 8 & 8 & 6 \\ 7 & 7 & 7 & 5 & 5 \end{pmatrix}$$

### Soluzione

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

#define MAX 100

int leggi_array(int x[]) {
  int i, n;

  scanf("%d", &n);
  for (i=0; i<n; i++)
    scanf("%d", &x[i]);
  return(n);
}

void stampa_matrice(int m[MAX][MAX], int nrighe, int ncolonne) {
  int i, j;

  for (i=0; i<nrighe; i++) {
    for (j=0; j<ncolonne; j++)
```

```
      printf("%3d ", m[i][j]);
    printf("\n");
  }
  return;
}

void costruisci_matrice(int m[MAX][MAX], int v[], int n) {
  int i=0, j=0, h, k;

  for (k=2; k<n; k+=2) {
    for (h=1; h<=v[k]; h++) {
      m[i][j] = v[k+1];
      j++;
      if (j>=v[1]) {
        j = 0;
        i++;
      }
    }
  }
  return;
}

int main(void) {
  int v[MAX], m[MAX][MAX], i=0, j=0, h, k, n;

  n = leggi_array(v);
  costruisci_matrice(m, v, n);
  stampa_matrice(m, v[0], v[1]);
  return(1);
}
```

# Esercizio n.2

Leggere in input due sequenze di $n$ ed $m$ numeri *floating point* e memorizzarle in due array $A$ e $B$. Senza utilizzare un terzo array di appoggio, inserire gli elementi di $B$ al centro dell'array $A$ e stampare $A$.

**Esempio.** Siano $A = (3, 2, 4, 7, 6, 5)$ e $B = (13, 1, 9)$. Allora al termine dell'elaborazione si avrà $A = (3, 2, 4, 13, 1, 9, 7, 6, 5)$.

## Soluzione

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

#define MAX 100

int leggi_array(float x[]) {
  int n, i;

  scanf("%d", &n);
  for (i=0; i<n; i++)
```

```
    scanf("%f", &x[i]);
  return(n);
}

void stampa_array(float x[], int n) {
  int i;

  for (i=0; i<n; i++)
    printf("%f ", x[i]);
  printf("\n");
  return;
}

void inserisci(float x[], float y[], int n, int m) {
  int i, j=0;

  for (i=n-1; i>=n/2; i--)
    x[i+m] = x[i];
  for (i=n/2; i<n/2+m; i++)
    x[i] = y[i - n/2];
  return;
}

int main(void) {
  float A[MAX], B[MAX];
  int n, m;

  n = leggi_array(A);
  m = leggi_array(B);
  inserisci(A, B, n, m);
  stampa_array(A, n+m);
  return(1);
}
```

# Esonero del 13 gennaio 2000

## Esercizio n.1

Leggere in input una lista di numeri interi ordinati in ordine crescente. Dopo aver letto la sequenza, inserire nella posizione corretta all'interno della lista, tutti i numeri mancanti. Stampare in output la lista. Non devono essere usate altre liste o array di appoggio.

**Esempio.** Supponiamo che sia fornita in input la sequenza $4, 7, 8, 9, 15, 17, 21$. Dopo aver memorizzato gli elementi nella lista, vengono inseriti i numeri mancanti, ottenendo la lista composta dagli elementi $4, 5, 6, 7, 8, \ldots, 18, 19, 20, 21$.

### Soluzione

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

struct nodo {
  int info;
  struct nodo *next;
};

struct nodo *leggi_lista(void) {
  struct nodo *p, *primo = NULL;
  int i, n, x;

  printf("Numero di elementi: ");
  scanf("%d", &n);
  for (i=0; i<n; i++) {
    scanf("%d", &x);
    p = malloc(sizeof(struct nodo));
    p->info = x;
    p->next = primo;
    primo = p;
  }
  return(primo);
}

void stampa_lista(struct nodo *p) {
  while (p != NULL) {
    printf("%d --> ", p->info);
```

```
    p = p->next;
  }
  printf("Null\n");
  return;
}

void completa_lista(struct nodo *p) {
  struct nodo *q;

  while (p->next != NULL) {
    if (p->info > p->next->info + 1) {
      q = malloc(sizeof(struct nodo));
      q->info = p->next->info + 1;
      q->next = p->next;
      p->next = q;
    } else {
      p = p->next;
    }
  }
  return;
}

int main(void) {
  struct nodo *primo;

  primo = leggi_lista();
  completa_lista(primo);
  stampa_lista(primo);
  return(1);
}
```

## Esercizio n.2

Leggere in input $n$ sequenze di numeri *floating point* e memorizzarle come liste. Costruire, utilizzando delle liste di adiacenza, il "grafo intersezione" $G = (V, E)$ secondo le seguenti regole:

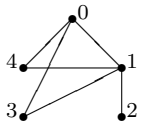
1. $V = \{0, ..., n-1\}$;
2. siano $L_i$ ed $L_j$ $(i \neq j)$ due delle $n$ liste; se $L_i \cap L_j \neq \emptyset$ allora $(i, j) \in E$.

È bene osservare che naturalmente, per la simmetria dell'intersezione, il grafo $G$ è non orientato.

**Esempio.** Siano $n = 5$; consideriamo le seguenti sequenze:

$$\begin{aligned} L_0 &= (1.1, 2.7, 3.14) \\ L_1 &= (71, 100, 2.7, 24.42, 67.9) \\ L_2 &= (5.8, 6.9, 71) \\ L_3 &= (1, 2, 3.14, 2.7, 1513.5) \\ L_4 &= (0.3, 24.42, 67.9, 1.1, 1.2, 1.3) \end{aligned}$$

Il grafo $G$ è costituito dall'insieme dei vertici $V = \{0, 1, 2, 3, 4\}$ e dall'insieme degli spigoli $E = \{\{0, 1\}, \{1, 2\}, \{0, 3\}, \{1, 3\}, \{0, 4\}, \{1, 4\}\}$, come rappresentato in figura.




## Soluzione

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

#define MAX 100

struct nodo1 {
  float info;
  struct nodo1 *next;
};

struct nodo2 {
  int info;
  struct nodo2 *next;
};

struct nodo1 *leggi_lista(void) {
  struct nodo1 *p, *primo = NULL;
  int i, n;
  float x;

  printf("Numero di elementi: ");
  scanf("%d", &n);
  for (i=0; i<n; i++) {
    scanf("%f", &x);
    p = malloc(sizeof(struct nodo1));
    p->info = x;
    p->next = primo;
    primo = p;
  }
  return(primo);
}

void stampa_lista(struct nodo2 *p) {
  while (p != NULL) {
    printf("%d --> ", p->info);
    p = p->next;
  }
  printf("Null\n");
  return;
}

void stampa_grafo(struct nodo2 *l[], int n) {
```

```
  int i;

  for (i=0; i<n; i++) {
    printf("%d: ");
    stampa_lista(l[i]);
  }
  return;
}

int intersezione(struct nodo1 *p, struct nodo1 *q) {
  struct nodo1 *r;
  int risposta = 0;
  while (p != NULL && risposta == 0) {
    r = q;
    while (r != NULL && p->info != r->info)
      r = r->next;
    if (r != NULL)
      risposta = 1;
    p = p->next;
  }
  return(risposta);
}

void aggiungi_vertice(struct nodo2 **p, int x) {
  struct nodo2 *q;

  q = malloc(sizeof(struct nodo2));
  q->info = x;
  q->next = *p;
  *p = q;
  return;
}

void costruisci(struct nodo1 *l[], struct nodo2 *g[], int n) {
  int i, j;

  for (i=0; i<n; i++)
    g[i] = NULL;
  for (i=0; i<n-1; i++)
    for (j=i+1; j<n; j++)
      if (intersezione(l[i], l[j])) {
        aggiungi_vertice(&g[i], j);
        aggiungi_vertice(&g[j], i);
      }
  return;
}

int main(void) {
  int i, n;
  struct nodo1 *L[MAX];
  struct nodo2 *G[MAX];

  printf("Numero di liste: ");
  scanf("%d", &n);
```

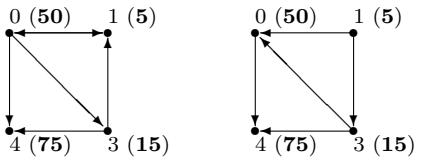
```
  for (i=0; i<n; i++)
    L[i] = leggi_lista();
  costruisci(L,G,n);
  stampa_grafo(G,n);
  return(1);
}
```

# Esame del 21 gennaio 2000

## Esercizio n.1

Leggere in input un grafo orientato $G = (V, E)$ e rappresentarlo mediante liste di adiacenza. Leggere in input un insieme di pesi $W = \{w_i\}_{i=1,\ldots,n}$ associati ai vertici del grafo. Verificare che per ogni spigolo $(i, j) \in E$ risulti $w_i \leq w_j$. Se tale disuguaglianza non è rispettata, allora invertire lo spigolo, eliminando lo spigolo $(i, j)$ ed introducendo, se non esiste, lo spigolo $(j, i)$. Nel far questo devono essere modificate le liste di adiacenza del grafo originale, che al termine dell'elaborazione devono essere stampate.

**Esempio.** Consideriamo il grafo $G = (V, E)$ con quattro vertici $V = \{1, 2, 3, 4\}$ ai quali sono associati i pesi $w_1 = 50, w_2 = 5, w_3 = 15, w_4 = 75$. Supponiamo che l'insieme degli spigoli sia $E = \{(1, 2), (1, 3), (1, 4), (2, 1), (3, 2), (3, 4)\}$. Allora il grafo finale sarà costituito dagli spigoli $\{(1, 4), (2, 1), (2, 3), (3, 1), (3, 4)\}$, eliminando quindi lo spigolo $(1, 2)$ (visto che già esiste lo spigolo opposto $(2, 1)$) ed invertendo il verso degli spigoli $(3, 2)$ e $(1, 3)$.




### Soluzione

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

#define MAX 100

struct nodo {
  int info;
  struct nodo *next;
};

struct nodo *leggi_lista(void) {
  struct nodo *primo, *p;
  int i, n, x;

  primo = NULL;
```

```
  printf("Numero di elementi: ");
  scanf("%d", &n);
  for (i=0; i<n; i++) {
    scanf("%d", &x);
    p = malloc(sizeof(struct nodo));
    p->info = x;
    p->next = primo;
    primo = p;
  }
  return(primo);
}

int leggi_grafo(struct nodo *l[]) {
  int i, n;

  printf("Numero di vertici: ");
  scanf("%d", &n);
  for (i=0; i<n; i++)
    l[i] = leggi_lista();
  return(n);
}

void leggi_pesi(int w[], int n) {
  int i;

  printf("Inserisci i pesi associati ai vertici del grafo: ");
  for (i=0; i<n; i++)
    scanf("%d", &w[i]);
  return;
}

void elabora(struct nodo *l[], int n, int w[]) {
  struct nodo *p, *q, *prec;
  int i, j;

  for (i=0; i<n; i++) {
    p = l[i];
    prec = NULL;
    while (p != NULL) {
      if (w[i] > w[p->info]) {
        j = p->info;
        if (prec == NULL) {
          l[i] = p->next;
          free(p);
          p = l[i];
        } else {
          prec->next = p->next;
          free(p);
          p = prec->next;
        }
        q = l[j];
        while (q != NULL && q->info != i)
          q = q->next;
        if (q == NULL) {
```

```
          q = malloc(sizeof(struct nodo));
          q->info = i;
          q->next = l[j];
          l[j] = q;
        }
      } else {
        prec = p;
        p = p->next;
      }
    }
  }
  return;
}

void stampa_lista(struct nodo *p) {
  while (p != NULL) {
    printf("%d -> ", p->info);
    p = p->next;
  }
  printf("NULL\n");
  return;
}

void stampa_grafo(struct nodo *l[], int n) {
  int i;

  for (i=0; i<n; i++) {
    printf("%d: ", i);
    stampa_lista(l[i]);
  }
  return;
}

int main(void) {
  struct nodo *lista[MAX];
  int n, w[MAX];

  n = leggi_grafo(lista);
  leggi_pesi(w,n);
  stampa_grafo(lista,n);
  elabora(lista,n,w);
  stampa_grafo(lista,n);
  return(1);
}
```

## Esercizio n.2

Leggere in input tre tabelle che rappresentano i dati contabili di un grande magazzino. Le tre tabelle sono le seguenti:

***Articoli:*** contiene due colonne con numeri interi, la prima con il codice dell'artico-

lo e la seconda con il suo prezzo. I codici degli articoli potrebbero non essere progressivi.

***Clienti:*** contiene due colonne con numeri interi, la prima con il codice cliente e la seconda con il tasso di sconto applicato sui prezzi per quel particolare cliente. I codici cliente potrebbero non essere progressivi.

***Vendite:*** contiene l'elenco degli oggetti venduti e dei clienti a cui sono stati venduti; è una tabella con due colonne costituite da numeri interi: la prima presenta il codice del prodotto venduto e la seconda il codice del cliente a cui tale prodotto è stato venduto.

Calcolare il totale della spesa per ogni cliente (applicando quindi differenti tassi percentuali di sconto).

**Esempio.** Supponiamo di ricevere in input le tabelle riportate di seguito. Il cliente 10, ad esempio, ha speso $1.000 \cdot 3 + 1.500 \cdot 2 = 6.000$ lire; su questo totale deve però essere applicato il tasso di sconto di cui gode il cliente, ossia il 30%, ottenendo quindi 4.200 lire.

| **Articoli** | |
|---|---|
| Codice | Prezzo |
| 100 | 1.000 |
| 200 | 1.500 |
| 300 | 750 |

| **Clienti** | |
|---|---|
| Codice | Sconto |
| 10 | 30 |
| 13 | 20 |
| 17 | 15 |

| **Vendite** | |
|---|---|
| Articolo | Cliente |
| 100 | 10 |
| 200 | 10 |
| 300 | 13 |
| 100 | 10 |
| 200 | 17 |
| 200 | 13 |
| 200 | 10 |
| 100 | 10 |
| 100 | 17 |
| 100 | 13 |
| 300 | 17 |

## Soluzione

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

#define MAX 100

int leggi_matrice(int mat[MAX][2]) {
  int i, j, n;

  printf("Numero di righe: ");
  scanf("%d", &n);
  for (i=0; i<n; i++)
    for (j=0; j<2; j++)
      scanf("%d", &mat[i][j]);
  return(n);
}
```

```
float spesa(int art[MAX][2], int cli[MAX][2], int vend[MAX][2],
  int riga, int nart, int nven) {
  int cliente, sconto, i, j;
  float totale = 0.0;

  cliente = cli[riga][0];
  sconto  = cli[riga][1];
  for (i=0; i<nven; i++)
    if (vend[i][1] == cliente)
      for (j=0; j<nart; j++)
        if (vend[i][0] == art[j][0]) {
          totale += art[j][1] - (float)(art[j][1] * sconto / 100.0);
          j = nart;
        }
  return(totale);
}

int main(void) {
  int articoli[MAX][2], clienti[MAX][2], vendite[MAX][2];
  int nart, ncli, nven, i;

  nart = leggi_matrice(articoli);
  ncli = leggi_matrice(clienti);
  nven = leggi_matrice(vendite);

  for (i=0; i<ncli; i++)
    printf("Il cliente %d ha speso %f lire\n", clienti[i][0],
      spesa(articoli, clienti, vendite, i, nart, nven));
  return(1);
}
```

# Esame dell'11 febbraio 2000

## Esercizio n.1

Generare una matrice $20 \times 40$ in modo casuale, con elementi appartenenti all'insieme $\{0, 1\}$. La matrice rappresenta un labirinto: gli elementi che valgono 0 sono un passaggio, mentre quelli con valore 1 rappresentano una barriera (un muro). Leggere in input una sequenza di $n$ mosse orizzontali e verticali: le prime sono rappresentate da coppie di elementi $(\pm k, 0)$, le altre da coppie di elementi $(0, \pm h)$, con $h < 20$ e $k < 40$. Verificare se, partendo dalla cella di coordinate $(0, 0)$, la sequenza di mosse applicate al "labirinto" generato in modo casuale, produce un percorso effettivamente realizzabile (privo di impedimenti) ed in tal caso stampare le coordinate della cella su cui il cammino termina.

**Esempio.** Si consideri la seguente matrice in formato ridotto:

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

La sequenza di mosse $(0, 1)$, $(1, 0)$, $(0, 1)$, $(3, 0)$, $(0, -1)$ è una sequenza lecita che ci porta nell'elemento posto sulla quinta colonna della seconda riga. Viceversa la sequenza $(0, 1)$, $(2, 0)$, $(0, 2)$ è una sequenza di mosse che non produce un percorso realizzabile, visto che la seconda mossa ci porterebbe ad invadere la cella posta sulla terza colonna della seconda riga, che è occupata da un ostacolo.

### Soluzione

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

#define RIGHE   20
#define COLONNE 40
#define MAX     50

void genera_matrice(int m[RIGHE][COLONNE]) {
  int i, j;

  srand((unsigned) time(NULL));
```

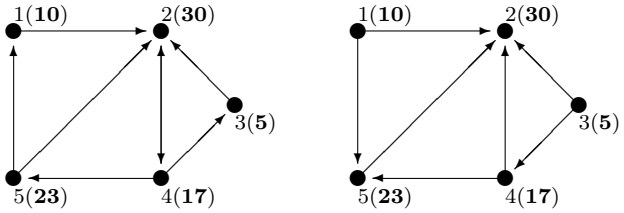
```
  for (i=0; i<RIGHE; i++)
    for (j=0; j<COLONNE; j++)
      m[i][j] = rand() % 2;
  return;
}

int leggi_mosse(int mosse[MAX][2]) {
  int i, n;

  printf("Numero di mosse: ");
  scanf("%d", &n);
  for (i=0; i<n; i++)
    scanf("%d %d", &mosse[i][0], &mosse[i][1]);
  return(n);
}

void stampa_matrice(int m[RIGHE][COLONNE]) {
  int i, j;

  for (i=0; i<RIGHE; i++) {
    for (j=0; j<COLONNE; j++)
      printf("%d ", m[i][j]);
    printf("\n");
  }
  return;
}

int main(void) {
  int m[RIGHE][COLONNE], mosse[MAX][2], r=0, c=0, i, j, n, ok;

  genera_matrice(m);
  stampa_matrice(m);
  n = leggi_mosse(mosse);
  i = 0;
  if (m[r][c] == 0) {
    ok = 1;
    for (i=0; i<n && ok; i++) {
      for (j=0; j<abs(mosse[i][0]) && ok; j++) {
        if (mosse[i][0] > 0)
          c++;
        else
          c--;
        if (c >= 0 && c < COLONNE && m[r][c] == 1)
          ok = 0;
      }
      for (j=0; j<abs(mosse[i][1]) && ok; j++) {
        if (mosse[i][1] > 0)
          r++;
        else
          r--;
        if (r >= 0 && r < RIGHE && m[r][c] == 1)
          ok = 0;
      }
    }
```

```
  } else {
    ok = 0;
  }
  if (ok)
    printf("Il percorso e' terminato in (%d,%d).\n", r, c);
  else
    printf("Il percorso non e' valido!\n");
  return(1);
}
```

# Esercizio n.2

Leggere in input un grafo orientato $G = (V, E)$ e rappresentarlo mediante liste di adiacenza. Leggere in input un insieme di pesi (interi) associati ai vertici del grafo: $\{w_1, ..., w_n\}$. Modificando le liste di adiacenza con cui è stato rappresentato il grafo $G$, variare l'orientamento degli spigoli in modo tale che per ogni spigolo $(u, v)$ risulti $w_u \leq w_v$.

**Esempio.** Sia $G = (V, E)$ il grafo letto in input, con $V = \{1, 2, 3, 4, 5\}$ ed $E = \{(1, 2), (2, 4), (3, 2), (4, 2), (4, 3), (4, 5), (5, 1), (5, 2)\}$. Sia $W$ l'insieme dei pesi associati ai vertici del grafo: $W = \{10, 30, 5, 17, 23\}$. Sulla sinistra è rappresentato il grafo letto in input e sulla destra il grafo prodotto dalla rielaborazione richiesta dall'esercizio.




## Soluzione

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

#define MAX 50

struct nodo {
  int info;
  struct nodo *next;
};

struct nodo *leggi_lista(void) {
  int i, n, x;
  struct nodo *p, *primo = NULL;

  printf("Numero di elementi: ");
  scanf("%d", &n);
  for (i=0; i<n; i++) {
```

```
    scanf("%d", &x);
    p = malloc(sizeof(struct nodo));
    p->info = x;
    p->next = primo;
    primo = p;
  }
  return(primo);
}

int leggi_grafo(struct nodo *v[]) {
  int i, n;

  printf("Numero di vertici: ");
  scanf("%d", &n);
  for (i=0; i<n; i++)
    v[i] = leggi_lista();
  return(n);
}

void stampa_lista(struct nodo *p) {
  while (p != NULL) {
    printf("%d --> ", p->info);
    p = p->next;
  }
  printf("NULL\n");
  return;
}

void stampa_grafo(struct nodo *v[], int n) {
  int i;

  for (i=0; i<n; i++) {
    printf("%d: ", i);
    stampa_lista(v[i]);
  }
  return;
}

void leggi_pesi(int w[], int n) {
  int i;

  for (i=0; i<n; i++)
    scanf("%d", &w[i]);
  return;
}

void aggiungi(struct nodo *v[], int i, int j) {
  struct nodo *p;

  p = v[i];
  while (p != NULL && p->info != j)
    p = p->next;
  if (p == NULL) {
    p = malloc(sizeof(struct nodo));
```

```
    p->info = j;
    p->next = v[i];
    v[i] = p;
  }
  return;
}

int main(void) {
  int i, n, w[MAX];
  struct nodo *v[MAX], *p, *prec;

  n = leggi_grafo(v);
  leggi_pesi(w, n);
  for (i=0; i<n; i++) {
    p = v[i];
    prec = NULL;
    while (p != NULL) {
      if (w[i] > w[p->info]) {
        aggiungi(v, p->info, i);
        if (prec != NULL) {
          prec->next = p->next;
          free(p);
          p = prec->next;
        } else {
          v[i] = p->next;
          free(p);
          p = v[i];
        }
      } else {
        prec = p;
        p = p->next;
      }
    }
  }
  stampa_grafo(v, n);
  return(1);
}
```

# Esame del 14 luglio 2000

## Esercizio n.1

Leggere in input una sequenza di numeri interi e memorizzarla in una lista in ordine crescente, operando sui puntatori agli elementi della lista. Non è consentito l'uso di array di appoggio.

### Soluzione

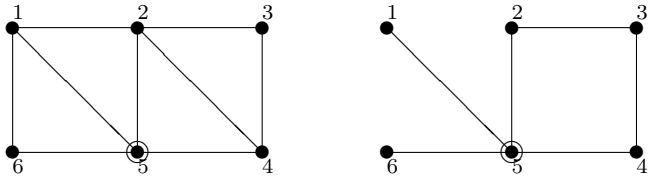
```
#include <stdlib.h>
#include <stdio.h>

struct nodo {
  int info;
  struct nodo *next;
};

void stampa_sequenza(struct nodo *p) {
  while (p != NULL) {
    printf("%d --> ", p->info);
    p = p->next;
  }
  printf("NULL\n");
  return;
}

struct nodo *inserisci(struct nodo *primo, int x) {
  struct nodo *nuovo, *prec, *p;

  nuovo = malloc(sizeof(struct nodo));
  nuovo->info = x;
  prec = NULL;
  p = primo;
  while (p != NULL && p->info < nuovo->info) {
    prec = p;
    p = p->next;
  }
  nuovo->next = p;
  if (prec == NULL)
    primo = nuovo;
  else
```

```
    prec->next = nuovo;
  return(primo);
}

struct nodo *leggi_sequenza(void) {
  int n, i, x;
  struct nodo *primo = NULL;

  printf("Numero di elementi: ");
  scanf("%d", &n);
  for (i=0; i<n; i++) {
    scanf("%d", &x);
    primo = inserisci(primo, x);
  }
  return(primo);
}

int main(void) {
  struct nodo *primo;

  primo = leggi_sequenza();
  stampa_sequenza(primo);
  return(1);
}
```

# Esercizio n.2

Leggere in input un grafo $G = (V, E)$ non orientato e memorizzarlo mediante una matrice di adiacenza. Scelto arbitrariamente uno dei vertici $v \in V$ di grado massimo, eliminare dal grafo tutti gli spigoli $(u, w) \in E$ per ogni $u$ e $w$ adiacenti a $v$.

**Esempio.** Sia $G = (V, E)$ il grafo letto in input, con $V = \{1, 2, 3, 4, 5, 6\}$ ed $E = \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1), (1, 5), (2, 5), (2, 4)\}$. I vertici di grado massimo sono 2 e 5 (entrambi di grado 4). Scegliendo il vertice 5, devono essere eliminati gli spigoli $(1, 2)$ (perché $(1, 5), (2, 5) \in E$), $(1, 6)$ (perché $(1, 5), (6, 5) \in E$) e $(2, 4)$ (perché $(5, 4), (5, 2) \in E$).




## Soluzione

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

#define MAX 20
```

```
int leggi_grafo(int M[MAX][MAX]) {
  int n, x, i, j;

  printf("numero di vertici: ");
  scanf("%d", &n);
  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
      M[i][j] = 0;
  for (i=0; i<n; i++) {
    printf("Vertici adiacenti a %d (-1 per terminare): ", i);
    scanf("%d", &x);
    while (x >= 0) {
      M[i][x] = 1;
      scanf("%d", &x);
    }
  }
  return(n);
}

int gradomassimo(int M[MAX][MAX], int n){
  int i, j, rigamax, gradomax, grado;

  gradomax = -1;
  for (i=0; i<n; i++) {
    grado = 0;
    for (j=0; j<n; j++)
      grado += M[i][j];
    if (grado > gradomax) {
      gradomax = grado;
      rigamax = i;
    }
  }
  return(rigamax);
}

void stampa_matrice(int M[MAX][MAX], int n) {
  int i, j;

  for (i=0; i<n; i++) {
    for (j=0; j<n; j++)
      printf("%3d ", M[i][j]);
    printf("\n");
  }
  return;
}

int main(void) {
  int n, max, i, j, M[MAX][MAX];

  n = leggi_grafo(M);
  stampa_matrice(M, n);
  max = gradomassimo(M, n);
  printf("Il vertice di grado massimo scelto e' %d.\n", max);
  for (i=0; i<n; i++)
```

```
    if (i != max && M[max][i] == 1)
      for (j=0; j<n; j++)
        if (M[max][j] == 1)
          M[i][j] = 0;
  stampa_matrice(M, n);
  return(1);
}
```

# Esonero del 4 aprile 2001

## Esercizio n.1

Leggere in input una sequenza $A$ di $n$ numeri interi. Stampare in output la successione $B$ calcolata secondo la seguente regola:

$$\begin{aligned} b_0 &= a_0 \\ b_i &= b_{i-1} + a_i \text{ se } a_i \text{ è pari} \\ b_i &= b_{i-1} - a_i \text{ se } a_i \text{ è dispari} \end{aligned}$$

**Esempio.** Supponiamo di leggere in input la sequenza $16, 7, -6, -9, 16, 14, 17, 36, 2$. Allora il programma stampa in output la sequenza $b_0 = 16$, $b_1 = 16 - 7 = 9$, $b_2 = 9 + (-6) = 3$, $b_3 = 3 - (-9) = 12$, $b_4 = 12 + 16 = 28$, ...

### Soluzione

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

#define MAX 100

int leggi_array(int V[]) {
  int i, n;

  printf("Numero di elementi: ");
  scanf("%d", &n);
  for (i=0; i<n; i++)
    scanf("%d", &V[i]);
  return(n);
}

int main(void) {
  int i, n, x, A[MAX];
  n = leggi_array(A);
  x = A[0];
  printf("b[0] = %d\n", x);
  for (i=1; i<n; i++) {
    if (A[i] % 2 == 0)
```

```
      x = x + A[i];
    else
      x = x - A[i];
    printf("b[%d] = %d\n", i, x);
  }
  return(1);
}
```

# Esercizio n.2

Leggere in input una sequenza di $n$ numeri interi e memorizzarla in un array $A$. Si supponga che la sequenza letta in input sia già ordinata in ordine crescente. Generare in modo casuale una seconda sequenza di $m$ numeri interi ed inserire gli elementi generati nella posizione corretta nell'array $A$ in modo che $A$ continui ad essere ordinato, eventualmente spostando in avanti gli elementi già presenti per fare posto ai nuovi elementi da inserire. Stampare in output l'array $A$.

**Esempio.** Sia $n = 8$ e sia $A = (3, 4, 7, 8, 17, 35)$ la sequenza letta in input. Sia $B = (15, 8, 6, 47)$ la sequenza di $m$ interi generata in modo casuale. Allora al termine dell'elaborazione l'array $A$ conterrà i seguenti elementi: $A = (3, 4, 6, 7, 8, 8, 15, 17, 35, 47)$.

## Soluzione

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

#define MAX 100

int leggi_array(int V[]) {
  int i, n;

  printf("Numero di elementi: ");
  scanf("%d", &n);
  for (i=0; i<n; i++)
    scanf("%d", &V[i]);
  return(n);
}

int genera_array(int V[]) {
  int i, n;

  printf("Numero di elementi: ");
  scanf("%d", &n);
  srand((unsigned)time(NULL));
  for (i=0; i<n; i++)
    V[i] = rand()%100;
  return(n);
}

void stampa_array(int V[], int n) {
  int i;
```

```
  for (i=0; i<n; i++)
    printf("%d ", V[i]);
  printf("\n");
  return;
}

void inserisci(int A[], int n, int B[], int m) {
  int i, j;

  for (j=0; j<m; j++) {
    i = n+j-1;
    while (i>=0 && A[i]>B[j]) {
      A[i+1] = A[i];
      i--;
    }
    A[i+1] = B[j];
  }
  return;
}

int main(void) {
  int i, n, m, x, A[MAX], B[MAX];
  n = leggi_array(A);
  m = genera_array(B);
  printf("La sequenza generata in modo casuale e':\n");
  stampa_array(B, m);
  inserisci(A, n, B, m);
  stampa_array(A, n+m);
  return(1);
}
```

# Esonero del 5 giugno 2001

## Esercizio n.1

Leggere in input un polinomio a coefficienti reali e memorizzarlo in una lista costituita da nodi con tre campi (grado, coefficiente, puntatore al seguente). Scrivere un programma che dopo aver costruito una seconda lista che rappresenta il polinomio derivato di quello fornito in input, ne stampi l'espressione sullo schermo.

Quale vantaggio si ottiene nel rappresentare i coefficienti del polinomio mediante una lista costruita in questo modo, anziché con un array? (max due righe di risposta).

**Esempio.** Digitando i dati corrispondenti al polinomio $7.5x^3+x+3.5$, il programma stamperà in output una frase del tipo:

```
Derivando il polinomio 7.5x^3 + 1.0x^1 + 3.5x^0, si ottiene
il polinomio 22.5x^2 + 1.0.
```

### Soluzione

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

struct nodo {
  int grado;
  float coefficiente;
  struct nodo *next;
};

struct nodo *leggi_lista(void) {
  struct nodo *p, *primo;
  int x, n, i;
  printf("Grado del polinomio: ");
  scanf("%d", &n);
  primo = NULL;
  for (i=0; i<=n; i++) {
    printf("Coefficiente di grado %d: ",i);
    scanf("%d", &x);
    if (x != 0) {
      p = (struct nodo *) malloc(sizeof(struct nodo));
      p->grado = i;
      p->coefficiente=x;
      p->next = primo;
```

```
      primo = p;
    }
  }
  return(primo);
}

void stampapolinomio(struct nodo *l) {
  while(l != NULL) {
    if (l->coefficiente != 0) {
      if(l->next != NULL) {
        printf("%4.1fx^%d", l->coefficiente, l->grado);
        if (l->next->coefficiente > 0)
          printf(" +");
      } else {
        printf("%4.1f", l->coefficiente);
      }
    }
    l = l->next;
  }
  printf("\n");
  return;
}

struct nodo *invertilista(struct nodo *p) {
  struct nodo *q, *primo=NULL;
  while(p != NULL) {
    q = (struct nodo *) malloc(sizeof(struct nodo));
    q->grado = p->grado;
    q->coefficiente = p->coefficiente;
    q->next = primo;
    primo = q;
    p = p->next;
  }
  return(primo);
}

struct nodo *derivapolinomio(struct nodo *p) {
  struct nodo *q, *primo=NULL;
  while(p != NULL) {
    if(p->grado != 0) {
      q = (struct nodo *) malloc(sizeof(struct nodo));
      q->grado = p->grado-1;
      q->coefficiente = (p->grado) * (p->coefficiente);
      q->next = primo;
      primo = q;
    }
    p = p->next;
  }
  return(primo);
}

int main(void){
  struct nodo *l, *l2;
  l = leggi_lista();
```

```
  printf("Derivando il polinomio:\n");
  stampapolinomio(l);
  printf("si ottiene il polinomio:\n");
  l = invertilista(l);
  l2 = derivapolinomio(l);
  stampapolinomio(l2);
  return(1);
}
```

# Esercizio n.2

Si consideri un linguaggio nel quale si dispone di tre paia di parentesi, utilizzate per delimitare parti di testo (come ad esempio le parentesi graffe e tonde nel linguaggio C). Si supponga che queste tre paia di parentesi siano le parentesi tonde "`(`" e "`)`", le parentesi quadre "`[`" e "`]`", e le parentesi graffe "`{`" e "`}`".

Si scriva un programma che chiede in input una stringa, ne memorizza i singoli caratteri in una lista e verifica se la stringa digitata inizialmente è "ben parentesizzata".

Diremo che una stringa è ben parentesizzata quando sono soddisfatte le due condizioni seguenti:

1. ad ogni parentesi aperta di uno dei tre tipi possibili, corrisponde, nel seguito dell'espressione, una parentesi chiusa dello stesso tipo;
2. ogni parentesi chiusa deve essere stata aperta in precedenza con una parentesi dello stesso tipo.

**Esempio.** La stringa "`]aso(o)[`" non è ben parentesizzata perché non soddisfa nessuna delle due condizioni. Anche "`]aso(a)[aa]`" non è ben parentesizzata perché pur soddisfacendo la condizione 1, non soddisfa la 2. La stringa "`(as[d]`" non soddisfa la prima condizione, mentre la stringa "`x+(y-z)+1}`" non soddisfa la seconda condizione.

Sono ben parentesizzate le seguenti stringhe: "`{x-lo[d(po])}sa`", "`([{o]})`", "`asdlt`". Sono valide quindi sia stringhe che non contengono parentesi, che quelle che non rispettano la "nidificazione" corretta, come nella stringa "`([{o]})`".

## Soluzione

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

#define MAX 100

struct nodo {
  char info;
  struct nodo *next;
};
```

```
struct nodo *costruisci_lista(char s[]) {
  struct nodo *p, *primo=NULL;
  int i;

  for (i=0; i<strlen(s); i++) {
    p = (struct nodo *) malloc(sizeof(struct nodo));
    p->info = s[i];
    p->next = primo;
    primo = p;
  }
  return(primo);
}

int parentesi(struct nodo *p) {
  int cont1=0, cont2=0, cont3=0;

  while (p!=NULL && cont1>=0 && cont2>=0 && cont3>=0) {
    if (p->info == ')')
      cont1++;
    if (p->info == '(')
      cont1--;
    if (p->info == ']')
      cont2++;
    if (p->info == '[')
      cont2--;
    if (p->info == '}')
      cont3++;
    if (p->info == '{')
      cont3--;
    p=p->next;
  }
  if (cont1 != 0 || cont2 != 0 || cont3 != 0)
    return(0);
  else
    return(1);
}

int main(void){
  char s[MAX];
  int n;
  struct nodo *primo;

  printf("Digitare una stringa di caratteri:\n");
  scanf("%s", s);
  primo = costruisci_lista(s);
  if (parentesi(primo))
    printf("La parola %s e' ben parentesizzata.\n", s);
  else
    printf("La parola %s NON e' ben parentesizzata.\n", s);
  return(1);
}
```

# Esame del 15 giugno 2001

## Esercizio n.1

Generare in modo casuale una matrice $A$ di ordine $10 \times 15$ con valori interi 0 e 1. Sia $B$ la seguente matrice quadrata di ordine 3:

$$B = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Verificare se in $A$ è possibile collocare $B$ in modo tale che nessun elemento di valore 1 di $B$ si vada a sovrapporre ad un elemento di valore 1 in $A$. In caso affermativo stampare le coordinate di riga e colonna in $A$ in cui verrebbe collocato l'elemento $B_{0,0}$.

**Esempio.** Consideriamo la seguente matrice:

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & \dots \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & \dots \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & \dots \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & \dots \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & \dots \end{pmatrix}$$

La matrice $B$ può essere collocata in $A$ a partire dalla terza colonna della seconda riga.

### Soluzione

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

#define RIGHE 10
#define COLONNE 15

void genera(int A[RIGHE][COLONNE]) {
  int i, j;

  srand((unsigned)time(NULL));
  for (i=0; i<RIGHE; i++) {
    for (j=0; j<COLONNE; j++) {
      A[i][j] = rand()%2;
    }
```

```
  }
  return;
}

int verifica(int A[RIGHE][COLONNE], int B[3][3], int *x, int *y) {
  int i, j, h, k, flag;

  flag = 0;
  for (i=0; i<RIGHE-2 && flag==0; i++) {
    for (j=0; j<COLONNE-2 && flag==0; j++) {
      flag = 1;
      for (h=0; h<3 && flag==1; h++)
        for (k=0; k<3 && flag==1; k++)
          if (B[h][k] == 1 && A[i+h][j+k] == 1)
            flag = 0;
    }
  }
  if (flag) {
    *x = j;
    *y = i;
  }
  return(flag);
}

void stampa_matrice(int A[RIGHE][COLONNE]) {
  int i, j;

  for (i=0; i<RIGHE; i++) {
    for (j=0; j<COLONNE; j++)
      printf("%d ", A[i][j]);
    printf("\n");
  }
  return;
}

int main(void) {
  int A[RIGHE][COLONNE], B[3][3], x, y;

  B[0][0] = 0; B[0][1] = 1; B[0][2] = 0;
  B[1][0] = 1; B[1][1] = 1; B[1][2] = 1;
  B[2][0] = 0; B[2][1] = 1; B[2][2] = 0;
  genera(A);
  stampa_matrice(A);
  if (verifica(A, B, &x, &y)) {
    printf("La matrice B si incastra in A collocandola");
    printf("a partire dalla posizione (%d,%d).\n", y, x);
  } else {
    printf("La matrice B non si incastra nella matrice A.\n");
  }
  return(1);
}
```

# Esercizio n.2

I flussi di denaro tra le filiali di una banca sono rappresentati mediante un grafo orientato. I vertici del grafo rappresentano le diverse filiali e lo spigolo orientato $v_i \rightarrow v_j$ indica che c'è un flusso dalla filiale $v_i$ alla filiale $v_j$. La quantità di denaro trasferito è rappresentata mediante numeri interi positivi associati agli spigoli del grafo.

Dopo aver rappresentato il grafo con liste di adiacenza costituite da record con tre campi (il numero della filiale, il flusso trasferito verso tale filiale ed il puntatore all'elemento successivo), si stampi l'elenco delle filiali che al termine degli spostamenti di denaro hanno aumentato il proprio capitale.

**Esempio.** Supponiamo di acquisire in input un grafo rappresentato dalle seguenti liste di adiacenza:

$$\begin{array}{lcl} 0 & : & \longrightarrow (1,20) \longrightarrow (2,100) \\ 1 & : & \longrightarrow (0,10) \longrightarrow (2,10) \longrightarrow (3,30) \\ 2 & : & \longrightarrow (1,50) \\ 3 & : & \longrightarrow (0,50) \longrightarrow (2,80) \end{array}$$

Allora le filiali rappresentate dai vertici 1 e 2 guadagnano (rispettivamente 20 e 140) mentre le filiali 0 e 3 perdono.

## Soluzione

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

#define MAX 20

struct nodo {
  int info, importo;
  struct nodo *next;
};

struct nodo *leggi_lista(void) {
  struct nodo *p, *primo=NULL;
  int i, n, x, y;

  printf("Numero di elementi: ");
  scanf("%d", &n);
  for (i=0; i<n; i++) {
    printf("Numero della banca e importo: ");
    scanf("%d %d", &x, &y);
    p = malloc(sizeof(struct nodo));
    p->info = x;
    p->importo = y;
    p->next = primo;
    primo = p;
  }
  return(primo);
}
```

```
int leggi_grafo(struct nodo *l[]) {
  int i, n;

  printf("Numero di banche: ");
  scanf("%d", &n);
  for (i=0; i<n; i++)
    l[i] = leggi_lista();
  return(n);
}

int main(void) {
  struct nodo *lista[MAX], *p;
  int delta[MAX], i, n;

  n = leggi_grafo(lista);
  for (i=0; i<n; i++)
    delta[i] = 0;
  for (i=0; i<n; i++) {
    p = lista[i];
    while (p != NULL) {
      delta[i] = delta[i] - p->importo;
      delta[p->info] = delta[p->info] + p->importo;
      p = p->next;
    }
  }
  printf("Le banche che aumentano il proprio capitale sono: ");
  for (i=0; i<n; i++) {
    if (delta[i]>0)
      printf("%d ", i);
  }
  printf("\n");
  return(1);
}
```

# Esame del 7 settembre 2001

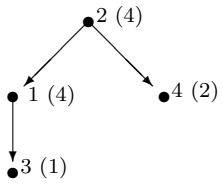
## Esercizio n.1

Letta in input una sequenza di numeri interi memorizzarla in una lista. Costruire un heap composto dagli elementi della lista, utilizzando come peso la frequenza di ognuno degli elementi. L'heap deve essere costruito facendo uso di un array o di una matrice. Stampare l'heap.

**Esempio.** Consideriamo la seguente lista letta in input:

$$1 \to 2 \to 1 \to 1 \to 2 \to 2 \to 2 \to 3 \to 1 \to 4 \to 4$$

Il numero 1 e il 2 compaiono quattro volte, il 3 una volta, il 4 due volte. Dunque l'heap costruito sulla base della lista è il seguente:




### Soluzione

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

#define MAX 100

struct nodo {
  int info;
  int flag;
  struct nodo *next;
};

struct nodo *leggi_lista(void) {
  int a, n, i;
  struct nodo *p, *primo=NULL;

  scanf("%d", &n);
  for (i=0; i<n; i++) {
    scanf("%d", &a);
```

```
    p = malloc(sizeof(struct nodo));
    p->info = a;
    p->flag = 0;
    p->next = primo;
    primo = p;
  }
  return(primo);
}

void stampa_array(int h[MAX][2]) {
  int i;

  for (i=1; i<=h[0][0]; i++)
    printf("%d (%d)  ", h[i][0], h[i][1]);
  printf("\n");
  return;
}

void scambia(int *a, int *b) {
  int c;

  c = *a;
  *a = *b;
  *b = c;
  return;
}

void inserisci(int h[MAX][2], int i, int f) {
  int last;

  last = h[0][0] + 1;
  h[last][0] = i;
  h[last][1] = f;
  while (last>1 && h[last/2][1]<h[last][1]) {
    scambia(&h[last/2][0], &h[last][0]);
    scambia(&h[last/2][1], &h[last][1]);
    last = last/2;
  }
  h[0][0]++;
  return;
}

int frequenza(struct nodo *p, struct nodo *q) {
  int f=0, flag=0;

  while (q != NULL && flag == 0) {
    if (q->info == p->info) {
      if (q->flag == 1) {
        flag = 1;
        f = 0;
      } else {
        f++;
        q->flag = 1;
      }
```

```
    }
    q = q->next;
  }
  return(f);
}

int main(void) {
  struct nodo *primo, *p;
  int H[MAX][2], f;

  H[0][0] = 0;
  primo = leggi_lista();
  p = primo;
  while (p != NULL) {
    f = frequenza(p, primo);
    if (f > 0)
      inserisci(H, p->info, f);
    p = p->next;
  }
  stampa_array(H);
  return(1);
}
```

# Esercizio n.2

Leggere in input una matrice rettangolare $A(100 \times 10)$, composta da numeri 0 e 1. Verificare se esiste una sequenza di elementi di valore 1 adiacenti tali che si possa individuare un percorso sulla matrice dalla prima all'ultima riga, spostandosi sempre da un elemento di valore 1 ad un altro adiacente, senza mai tornare su righe già percorse in precedenza.

**Esempio.** Consideriamo la seguente matrice (semplificata):

$$A = \begin{pmatrix} 0 & 1 & \mathbf{1} & 0 & 1 \\ 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 \\ 1 & 0 & 1 & \mathbf{1} & 0 \\ 0 & 1 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 1 \end{pmatrix}$$

Il percorso dalla prima all'ultima riga esiste ed è evidenziato dagli elementi in grassetto.

## Soluzione

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

#define RIGHE 100
#define COL 10
```

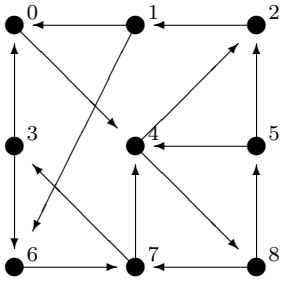
```
void leggi_matrice(int mat[RIGHE][COL]) {
  int i, j;

  for (i=0; i<RIGHE; i++)
    for (j=0; j<COL; j++)
      scanf("%d", &mat[i][j]);
  return;
}

int percorso(int mat[RIGHE][COL], int i, int j) {
  int risp;

  if (mat[i][j] == 1) {
    if (i==RIGHE-1) {
      risp = 1;
    } else {
      if ( (j>0 && percorso(mat, i+1, j-1)) ||
           (percorso(mat, i+1, j)) ||
           (j<COL-1 && percorso(mat, i+1, j+1)) )
        risp = 1;
      else
        risp = 0;
    }
  } else {
    risp = 0;
  }
  return(risp);
}

int main(void) {
  int M[RIGHE][COL], i;

  leggi_matrice(M);
  i=0;
  while (i<COL && !percorso(M, 0, i))
    i++;
  if (i==COL)
    printf("Il percorso NON esiste.\n");
  else
    printf("Il percorso esiste.\n");
  return(1);
}
```

# Esame del 14 settembre 2001

## Esercizio n.1

Leggere in input un grafo orientato $G = (V, E)$ ed una sequenza $S$ di vertici di $G$. Verificare se $S$ è un ciclo hamiltoniano, ossia un ciclo semplice in $G$ che consente di raggiungere tutti i vertici di $G$. Il grafo $G$ deve essere rappresentato utilizzando la struttura dati delle liste di adiacenza, la sequenza $S$ deve essere rappresentata mediante una lista.

**Esempio.** Consideriamo il grafo $G$ con nove vertici: $V = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$ rappresentato in figura.




La sequenza $S = (4, 8, 5, 2, 1, 6, 7, 3, 0, 4)$ è un ciclo hamiltoniano su $G$: infatti è un ciclo, è semplice e consente di visitare tutti i vertici di $G$.

### Soluzione

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

#define MAX 50

struct nodo {
  int info;
  struct nodo *next;
};

struct nodo *leggi_lista(void) {
  struct nodo *p, *primo, *ultimo;
  int a, i, n;

  primo = NULL;
```

```
  ultimo = NULL;
  printf("Numero di elementi: ");
  scanf("%d", &n);
  for (i=0; i<n; i++) {
    scanf("%d", &a);
    p = malloc(sizeof(struct nodo));
    p->info = a;
    p->next = NULL;
    if (!primo)
      primo = p;
    if (ultimo)
      ultimo->next = p;
    ultimo = p;
  }
  return(primo);
}

int leggi_grafo(struct nodo *l[]) {
  int i, n;

  printf("Numero di vertici del grafo: ");
  scanf("%d", &n);
  for (i=0; i<n; i++) {
    printf("Lista di adiacenza del vertice %d\n", i);
    l[i] = leggi_lista();
  }
  return(n);
}

int adiacente(int u, int v, struct nodo *l[]) {
  struct nodo *p;

  p = l[u];
  while (p!=NULL && p->info != v)
    p = p->next;
  if (p == NULL)
    return(0);
  else
    return(1);
}

int hamiltoniano(struct nodo *primo, struct nodo *l[], int n) {
  int visitato[MAX], i, flag;
  struct nodo *p;

  for (i=0; i<n; i++)
    visitato[i] = 0;
  p = primo;
  visitato[p->info] = 1;
  while (p->next != NULL && !visitato[p->next->info] &&
         adiacente(p->info, p->next->info, l)) {
    visitato[p->next->info] = 1;
    p = p->next;
  }
```

```
  flag = 1;
  for (i=0; i<n && flag==1; i++)
    flag = visitato[i];
  return(flag);
}

int main(void) {
  struct nodo *lista[MAX], *primo;
  int n;

  n = leggi_grafo(lista);
  primo = leggi_lista();
  if (hamiltoniano(primo, lista, n))
    printf("La sequenza e' un ciclo hamiltoniano in G.\n");
  else
    printf("La sequenza NON e' un ciclo hamiltoniano.\n");
  return(1);
}
```

# Esercizio n.2

Costruire in modo casuale una matrice $M$ di numeri interi con $n$ righe ed $m$ colonne ($n$ ed $m$ forniti in input dall'utente). Stampare tutte le matrici quadrate contenute in $M$.

**Esempio.** Consideriamo la seguente matrice di $n = 4$ righe e $m = 3$ colonne:

$$M = \begin{pmatrix} 0 & 5 & 4 \\ 1 & 4 & 9 \\ 2 & 5 & 6 \\ 7 & 8 & 1 \end{pmatrix}$$

La matrice $M$ contiene 6 matrici quadrate di ordine 2:

$$\begin{pmatrix} 0 & 5 \\ 1 & 4 \end{pmatrix} \quad \begin{pmatrix} 5 & 4 \\ 4 & 9 \end{pmatrix} \quad \begin{pmatrix} 1 & 4 \\ 2 & 5 \end{pmatrix} \quad \begin{pmatrix} 4 & 9 \\ 5 & 6 \end{pmatrix} \quad \begin{pmatrix} 2 & 5 \\ 7 & 8 \end{pmatrix} \quad \begin{pmatrix} 5 & 6 \\ 8 & 1 \end{pmatrix}$$

e 2 matrici quadrate di ordine 3:

$$\begin{pmatrix} 0 & 5 & 4 \\ 1 & 4 & 9 \\ 2 & 5 & 6 \end{pmatrix} \quad \begin{pmatrix} 1 & 4 & 9 \\ 2 & 5 & 6 \\ 7 & 8 & 1 \end{pmatrix}$$

## Soluzione

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

#define MAX 100

void genera_matrice(int M[MAX][MAX], int n, int m) {
  int i, j;
```

```
  for (i=0; i<n; i++)
    for (j=0; j<m; j++)
      M[i][j] = rand() % 10;
  return;
}

void stampa_matrice(int M[MAX][MAX], int n, int m) {
  int i, j;

  for (i=0; i<n; i++) {
    for (j=0; j<m; j++)
      printf("%2d ", M[i][j]);
    printf("\n");
  }
  return;
}

void stampa_quadrate(int M[MAX][MAX], int n, int m) {
  int i, j, h, k, dim, massimo, cont=0;

  if (n<m)
    massimo = n;
  else
    massimo = m;

  for (dim = 2; dim<=massimo; dim++) {
    printf("\nMatrici di ordine %d:\n\n", dim);
    for (h=0; h<=n-dim; h++) {
      for (k=0; k<=m-dim; k++) {
        cont++;
        printf("Matrice n.%d:\n", cont);
        for (i=h; i<h+dim; i++) {
          for (j=k; j<k+dim; j++)
            printf("%2d ", M[i][j]);
          printf("\n");
        }
      }
    }
  }
  return;
}

int main(void) {
  int n, m, mat[MAX][MAX];
  printf("Numero di righe: ");
  scanf("%d", &n);
  printf("Numero di colonne: ");
  scanf("%d", &m);
  genera_matrice(mat, n, m);
  stampa_matrice(mat, n, m);
  stampa_quadrate(mat, n, m);
  return(1);
}
```

# Esonero del 9 novembre 2001

## Esercizio n.1

Leggere in input una sequenza $A$ di $n$ numeri floating point ed un numero intero $k < n$. Stampare in output la sequenza di $k$ elementi contigui la cui somma sia massima.

**Esempio.** Supponiamo di leggere in input il numero $k = 4$ e la sequenza $A = (2.3, 4.7, 9.14, 6.54, 1.2, 3.5, 4.01, 14.0, -5.2, 2.1)$ di $n = 10$ elementi. La sequenza di $k = 4$ elementi di somma massima è $(1.2, 3.5, 4.01, 14.0)$ la cui somma degli elementi è 22.71.

### Soluzione

```
#include <stdlib.h>
#include <stdio.h>
#define MAX 100

int leggi_array(float V[]) {
  int i, n;

  printf("Numero di elementi: ");
  scanf("%d", &n);
  printf("Inserisci %d numeri floating point: ", n);
  for (i=0; i<n; i++)
    scanf("%f", &V[i]);
  return(n);
}

void stampa_array(float V[], int n) {
  int i;

  for(i=0; i<n; i++)
    printf("%f ", V[i]);
  printf("\n");
  return;
}

float somma(float V[], int k) {
  int i;
  float somma=0.0;
```

```
  for (i=0; i<k; i++)
    somma += V[i];
  return(somma);
}

int max_sequenza(float V[], int n, int k) {
  int i, i_max;
  float max, s;

  max = somma(&V[0], k);
  i_max = 0;
  for (i=1; i<n-k; i++) {
    s = somma(&V[i], k);
    if (s > max) {
      max = s;
      i_max = i;
    }
  }
  return(i_max);
}

int main(void) {
  int n, k, m;
  float A[MAX];

  n = leggi_array(A);
  printf("Lunghezza della sequenza: ");
  scanf("%d", &k);
  m = max_sequenza(A, n, k);
  stampa_array(&A[m], k);
  return(1);
}
```

# Esercizio n. 2

Leggere in input una matrice $A$ di $n \times m$ numeri interi compresi tra 0 e 9. Ogni riga della matrice rappresenta un numero intero positivo costituito da $m$ cifre, eventualmente riportando a sinistra degli zeri per numeri con meno di $m$ cifre. Calcolare la somma dei numeri e riportarne le cifre (numeri compresi tra 0 e 9) su un vettore $B$. Stampare $B$.

**Esempio.** Sia $A$ la seguente matrice di 3 righe e 5 colonne:

$$A = \begin{pmatrix} 0 & 2 & 9 & 4 & 3 \\ 8 & 7 & 7 & 0 & 2 \\ 9 & 6 & 4 & 0 & 0 \end{pmatrix}$$

Il risultato della somma dei tre numeri rappresentati sulle righe della matrice $(2943, 87702, 96400)$ è rappresentata nel seguente vettore $B$:

$$B = (1, 8, 7, 0, 4, 5)$$

## Soluzione

```
#include <stdlib.h>
#include <stdio.h>
#define MAX 15

void leggi_matrice(int A[MAX][MAX], int *n, int *m) {
  int i, j;

  printf("Numero di righe e di colonne: ");
  scanf("%d %d", n, m);
  for (i=0; i<*n; i++)
    for (j=0; j<*m; j++)
      scanf("%d", &A[i][j]);
  return;
}

void stampa_array(int V[], int n) {
  int i;

  for (i=0; i<n; i++)
    printf("%d ", V[i]);
  printf("\n");
  return;
}

void scambia(int *x, int *y) {
  int z;

  z = *x;
  *x = *y;
  *y = z;
  return;
}

void inverti_array(int V[], int n) {
  int i;

  for (i=0; i<n/2; i++)
    scambia(&V[i], &V[n-i-1]);
  return;
}

int main(void) {
  int A[MAX][MAX], B[MAX], n, m, k, i, j, somma, riporto;

  leggi_matrice(A, &n, &m);
  k = 0;
  riporto = 0;
  for (j=m-1; j>=0; j--) {
    somma = riporto;
    for (i=0; i<n; i++) {
      somma += A[i][j];
```

```
    }
    B[k] = somma - (somma/10)*10;
    riporto = somma/10;
    k++;
  }
  while (riporto>0) {
    B[k] = riporto - (riporto/10)*10;
    k++;
    riporto = riporto/10;
  }
  inverti_array(B, k);
  stampa_array(B, k);
  return(1);
}
```

# Esonero dell'11 gennaio 2002

## Esercizio n.1

Leggere in input una sequenza di caratteri alfanumerici e memorizzarli in una lista nell'ordine in cui sono stati letti. Dopo aver memorizzato l'intera sequenza, eliminare dalla lista le eventuali sottosequenze di elementi delimitati dalle parentesi tonde (aperta all'inizio della sequenza, chiusa alla fine della sequenza); sostituendoli con un solo elemento contenente un asterisco "*". Si tenga conto del fatto che la sequenza fornita in input non potrà contenere coppie di parentesi che si "intersecano" e che inoltre per ogni parentesi aperta esiste una parentesi chiusa.

**Esempio** Si consideri la sequenza "a, b, (, a, c, g, ), b, e, (, ), a, (, x, x, ), f". La lista di output sarà allora la seguente: a, b, (, *, ), b, e, (, *, ), a, (, *, ), f.

### Soluzione

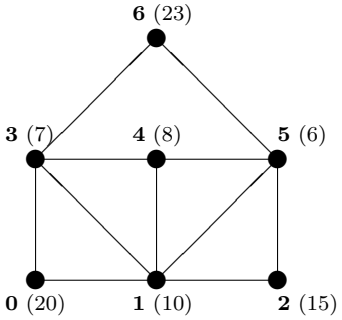
```
#include <stdlib.h>
#include <stdio.h>

#define MAX 20

struct nodo {
  char c;
  struct nodo *next;
};

struct nodo *leggi_lista(void) {
  struct nodo *p, *primo=NULL, *ultimo=NULL;
  int i, n;
  char a[10];

  printf("Numero di elementi: ");
  scanf("%d", &n);
  printf("Inserisci la sequenza di %d caratteri: ", n);
  for (i=0; i<n; i++) {
    p = malloc(sizeof(struct nodo));
    scanf("%s", a);
    p->c = a[0];
    p->next = NULL;
    if (primo == NULL) {
```

```
      primo = p;
      ultimo = p;
    } else {
      ultimo->next = p;
      ultimo = p;
    }
  }
  return(primo);
}

void eliminazione(struct nodo *primo) {
  struct nodo *p, *p1, *p2;

  p = primo;
  while (p != NULL) {
    if (p->c == '(') {
      p1 = p;
      p = p->next;
      while (p->c != ')') {
        p2 = p->next;
        free(p);
        p1->next = p2;
        p = p2;
      }
      p2 = p;
      p = malloc(sizeof(struct nodo));
      p->c = '*';
      p1->next = p;
      p->next = p2;
    }
    p = p->next;
  }
  return;
}

void stampa_lista(struct nodo *p) {
  while (p != NULL) {
    printf("%c ", p->c);
    p = p->next;
  }
  printf("\n");
  return;
}

int main(void) {
  struct nodo *primo;

  primo = leggi_lista();
  stampa_lista(primo);
  eliminazione(primo);
  stampa_lista(primo);
  return(1);
}
```

# Esercizio n. 2

Letto in input un grafo non orientato $G = (V, E)$ rappresentarlo mediante liste di adiacenza. Ad ogni verice del grafo è assegnato un peso intero non negativo. Stampare i vertici del triangolo di peso massimo. Un triangolo è un insieme di tre vertici di $V(G)$ tra loro adiacenti in $G$.

**Esempio** Si consideri il grafo $G$ rappresentato in figura, $V(G) = \{0, 1, 2, 3, 4, 5, 6\}$, ed i seguenti pesi assegnati ai vertici: $w(0) = 20$, $w(1) = 10$, $w(2) = 15$, $w(3) = 7$, $w(4) = 8$, $w(5) = 6$, $w(6) = 1$.




I triangoli contenuti in $G$ sono 4: $T_1 = \{0, 1, 3\}$, $T_2 = \{1, 3, 4\}$, $T_3 = \{1, 4, 5\}$, $T_4 = \{1, 2, 5\}$. Fra questi il triangolo di peso minimo è $T_3$: $w(T_1) = 20+10+7 = 37$.

## Soluzione

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

#define MAX 20

struct nodo {
  int info;
  struct nodo *next;
};

struct nodo *leggi_lista(void) {
  struct nodo *p, *primo=NULL;
  int i, n;

  printf("Numero di elementi: ");
  scanf("%d", &n);
  for (i=0; i<n; i++) {
    p = malloc(sizeof(struct nodo));
    scanf("%d", &p->info);
    p->next = primo;
    primo = p;
  }
  return(primo);
}

int leggi_grafo(struct nodo *L[]) {
```

```
  int i, n;

  printf("Numero di vertici del grafo: ");
  scanf("%d", &n);
  for (i=0; i<n; i++) {
    printf("Lista dei vertici adiacenti a %d.\n", i);
    L[i] = leggi_lista();
  }
  return(n);
}

void stampa_triangolo_max(struct nodo *L[], int n, int w[]) {
  int max = -1, i, v1, v2, v3;
  struct nodo *p, *q, *r;

  for (i=0; i<n; i++) {
    p = L[i];
    while (p != NULL) {
      q = p->next;
      while (q != NULL) {
        r = L[p->info];
        while (r != NULL && r->info != q->info) {
          r = r->next;
        }
        if (r != NULL && w[i] + w[p->info] + w[q->info] > max) {
          max = w[i] + w[p->info] + w[q->info];
          v1 = i;
          v2 = p->info;
          v3 = q->info;
        }
        q = q->next;
      }
      p = p->next;
    }
  }
  if (max >= 0) {
    printf("Il triangolo di peso massimo e' (%d, %d, %d)
      e pesa %d.\n", v1, v2, v3, max);
  } else {
    printf("Il grafo non contiene triangoli.\n");
  }
  return;
}

void leggi_pesi(int w[], int n) {
  int i;

  for (i=0; i<n; i++) {
    printf("w(%d): ", i);
    scanf("%d", &w[i]);
  }
  return;
}
```

```
int main(void) {
  struct nodo *L[MAX];
  int n, w[MAX];

  n = leggi_grafo(L);
  leggi_pesi(w, n);
  stampa_triangolo_max(L, n, w);
  return(1);
}
```

# Esame del 17 gennaio 2002

## Esercizio n.1

La grammatica di un certo linguaggio artificiale $L$, basato sull'alfabeto di simboli $A = \{a, b, c\}$, prevede che:

1. la lettera $b$ compaia in sequenze formate da un numero pari di elementi contigui;
2. la lettera $c$ compaia al massimo tre volte in ogni parola;
3. immediatamente dopo ogni occorrenza della lettera $c$ siano presenti almeno due simboli fra loro differenti.

Letta in input una sequenza di caratteri memorizzarla in una lista. Quindi verificare se la sequenza rappresenta una parola del linguaggio $L$.

**Esempio** Le seguenti stringhe sono parole del linguaggio $L$: $aaa \in L$, $abbbb \in L$, $cabbacbba \in L$, $ccabb \in L$. Viceversa le seguenti stringhe non appartengono al linguaggio: $cbb \notin L$ (la parola non soddisfa la regola 3), $cabbb \notin L$ (la parola non verifica la regola 1), $aaca \notin L$ (la parola non verifica la regola 3), $cabbcabbcabbcabb \notin L$ (la parola non soddisfa la regola 2).

### Soluzione

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

struct nodo {
  char info;
  struct nodo *next;
};

struct nodo *leggi_coda(void) {
  struct nodo *primo=NULL, *ultimo=NULL, *p;
  int i, n;
  char parola[100];

  printf("Parola: ");
  scanf("%s", parola);
```

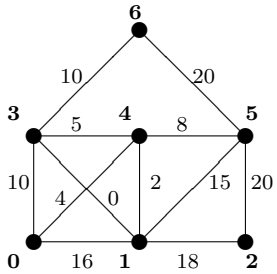
```
  for (i=0; i<strlen(parola); i++) {
    p = malloc(sizeof(struct nodo));
    p->info = parola[i];
    p->next = NULL;
    if (!primo)
      primo = p;
    if (ultimo)
      ultimo->next = p;
    ultimo = p;
  }
  return(primo);
}

void stampa_lista(struct nodo *p) {
  printf("Parola: '");
  while (p != NULL) {
    printf("%c", p->info);
    p = p->next;
  }
  printf("'\n");
  return;
}

int verifica(struct nodo *p) {
  int contab = 0, contac = 0, erab = 0, flag = 1, rc;
  while (p!=NULL && flag==1) {
    if (p->info == 'b') {
      contab++;
      erab=1;
    } else {
      if (erab == 1 && contab % 2 == 1) {
        flag = 0;
        contab = 0;
        erab = 0;
      }
      if (p->info == 'c') {
        contac++;
        if (p->next == NULL || p->next->next == NULL ||
        p->next->info == p->next->next->info)
          flag = 0;
      }
    }
    p = p->next;
  }
  if (flag == 0 || contab % 2 == 1)
    rc = 0;
  else
    rc = 1;
  return(rc);
}

int main(void) {
  struct nodo *p;
  p = leggi_coda();
```

```
  stampa_lista(p);
  if (verifica(p) == 1)
    printf("La parola appartiene al linguaggio.\n");
  else
    printf("La parola non appartiene al linguaggio.\n");
  return(1);
}
```

# Esercizio n. 2

Letto in input un grafo non orientato $G = (V, E)$, rappresentarlo mediante una matrice di adiacenza. Agli spigoli del grafo sono assegnati dei pesi interi non negativi $w(u, v) \geq 0$. Letta in input una sequenza $S$ di vertici di $G$, memorizzarla in un array. Verificare se il sottografo di $G$ indotto da $S$ è completo, ossia se per ogni $u, v \in S$ risulta $(u, v) \in E(G)$. In ognicaso stampare il peso complessivo del sottografo di $G$ indotto da $S$.

**Esempio** Si consideri il grafo $G$ rappresentato in figura, $V(G) = \{0, 1, 2, 3, 4, 5, 6\}$, ed i seguenti pesi assegnati agli spigoli: $w(0, 1) = 16$, $w(0, 3) = 10$, $w(0, 4) = 4$, $w(1, 3) = 0$, $w(1, 2) = 18$, $w(1, 4) = 2$, $w(1, 5) = 15$, $w(2, 5) = 20$, $w(3, 4) = 5$, $w(3, 6) = 10$, $w(4, 5) = 8$, $w(5, 6) = 20$.




L'insieme di vertici $S = \{0, 1, 3, 4\}$ induce su $G$ un sottografo completo (devo considerare come spigoli del sottografo tutti gli spigoli di $G$ che collegano vertici di $S$), il cui peso è $W = 16 + 10 + 4 + 2 + 0 + 5 = 37$. Viceversa l'insieme $S = \{3, 4, 5, 6\}$ non induce su $G$ un sottografo completo (mancano ad esempio gli spigoli $(4, 6)$ e $(3, 5)$); il peso del sottografo è $W = 5 + 10 + 8 + 20 = 43$.

## Soluzione

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

#define MAX 100

int leggi_grafo(int M[MAX][MAX]) {
  int n, i, j;

  printf("Numero di vertici: ");
  scanf("%d", &n);
  for (i=0; i<n; i++) {
    M[i][i] = -1;
```

```
    for (j=i+1; j<n; j++) {
      printf("Peso dello spigolo (%d,%d), -1 se non esiste: ", i, j);
      scanf("%d", &M[i][j]);
      M[j][i] = M[i][j];
    }
  }
  return(n);
}

int leggi_array(int a[]) {
  int n, i;

  printf("Numero di elementi: ");
  scanf("%d", &n);
  printf("Inserisci %d elementi: ", n);
  for (i=0; i<n; i++)
    scanf("%d", &a[i]);
  return(n);
}

int main(void) {
  int G[MAX][MAX], S[MAX], i, j, flag=1, somma=0, n, m;

  n = leggi_grafo(G);
  m = leggi_array(S);
  for (i=0; i<m; i++)
    for (j=i+1; j<m; j++)
      if (G[S[i]][S[j]] == -1)
        flag = 0;
      else
        somma += G[S[i]][S[j]];
  if (flag)
    printf("Il sottografo indotto da S su G e' completo.\n");
  else
    printf("Il sottografo indotto da S su G NON e' completo.\n");
  printf("Il sottografo indotto da S su G pesa %d.\n", somma);
  return(1);
}
```

# Esame del 12 aprile 2002

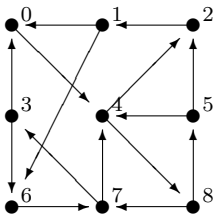
## Esercizio n.1

Un grafo orientato $G = (V, E)$ rappresenta i collegamenti (*link*) tra le pagine di un sito web (i vertici del grafo sono le pagine, gli spigoli sono i link tra le pagine). Si vuole calcolare un indice di rilevanza (*page rank*) associato ad ogni pagina del sito. Sia $u \in V$ una generica pagina del sito web e siano $v_1, v_2, \ldots, v_k \in V$ le pagine che contengono un link diretto alla pagina $u$ ($(v_1, u), (v_2, u), \ldots, (v_k, u) \in E$). Se indichiamo con $l_i$ il numero di link presenti nella pagina $v_i$, l'indice *page rank* associato alla pagina $u$ è dato dall'espressione:

$$r(u) = \sum_{i=1}^{k} \frac{1}{l_i}$$

Dopo aver letto in input il grafo $G$ rappresentandolo mediante liste di adiacenza, calcolare e stampare il *page rank* associato ad ogni vertice.

**Esempio** Consideriamo il grafo $G$ con nove vertici: $V = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$ rappresentato in figura.




Il *page rank* del vertice 4 è dato da: $r(4) = \frac{1}{l_0} + \frac{1}{l_5} + \frac{1}{l_7} = 1 + \frac{1}{2} + \frac{1}{2} = 2$.

### Soluzione

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

#define MAX_N 20

struct nodo {
  int info;
  struct nodo *next;
```

```
};

struct nodo *leggi_lista(void) {
  struct nodo *p, *primo;
  int i, n;

  printf("Numero di elementi: ");
  scanf("%d", &n);

  primo = NULL;
  for (i=0; i<n; i++) {
    p = malloc(sizeof(struct nodo));
    scanf("%d", &p->info);
    p->next = primo;
    primo = p;
  }
  return(primo);
}

int leggi_grafo(struct nodo *L[]) {
  int i, n;

  printf("Numero di vertici del grafo: ");
  scanf("%d", &n);
  for (i=0; i<n; i++)
    L[i] = leggi_lista();
  return(n);
}

int main(void) {
  struct nodo *lista[MAX_N], *p;
  int i, n, l[MAX_N];
  float r[MAX_N];

  n = leggi_grafo(lista);
  for (i=0; i<n; i++) {
    r[i] = 0;
    l[i] = 0;
    p = lista[i];
    while (p != NULL) {
      l[i]++;
      p = p->next;
    }
  }
  for (i=0; i<n; i++) {
    p = lista[i];
    while (p != NULL) {
      r[p->info] = r[p->info] + 1.0/(float)l[i];
      p = p->next;
    }
  }
  for (i=0; i<n; i++)
    printf("r[%d] = %f\n", i, r[i]);
  return(1);
```

```
}
```

# Esercizio n. 2

Leggere in input una matrice $n \times m$ di caratteri. Letta in input una stringa, contare quante volte è contenuta nelle righe e nelle colonne della matrice.

**Esempio** Si consideri la seguente matrice $A$ di $n = 4$ righe e $m = 5$ colonne e la stringa $B$ di $k = 3$ caratteri:

$$A = \begin{pmatrix} x & c & e & c & a \\ w & a & e & c & q \\ d & t & p & a & z \\ p & c & a & t & f \end{pmatrix}$$

$$B = \begin{pmatrix} c & a & t \end{pmatrix}$$

La stringa $B$ è contenuta tre volte nella matrice $A$: nella seconda colonna, nella quarta colonna e nella quarta riga.

## Soluzione

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

#define MAX 20

void leggi_matrice(char A[MAX][MAX], int *n, int *m) {
  int i, j;

  printf("Numero di righe e di colonne: ");
  scanf("%d %d", n, m);
  for (i=0; i<*n; i++) {
    printf("Elementi riga %d: ", i);
    scanf("%s", &A[i]);
  }
  return;
}

int verifica_orizzontale(char A[MAX][MAX], char B[MAX], int i, int j) {
  int ok = 1, k;
  for (k=0; k<strlen(B) && ok==1; k++)
    if (A[i][j+k] != B[k])
      ok = 0;
  return(ok);
}

int verifica_verticale(char A[MAX][MAX], char B[MAX], int i, int j) {
  int ok = 1, k;
  for (k=0; k<strlen(B) && ok==1; k++)
    if (A[i+k][j] != B[k])
```

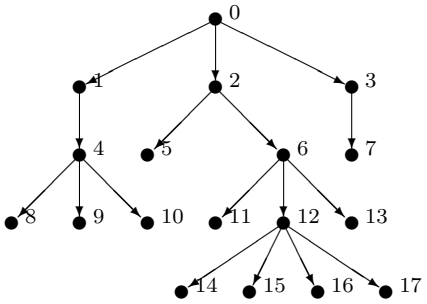
```
      ok = 0;
  return(ok);
}

int main(void) {
  int i, j, n, m, cont=0;
  char A[MAX][MAX], B[MAX];

  leggi_matrice(A, &n, &m);
  scanf("%s", B);

  for (i=0; i<n; i++)
    for (j=0; j<m-strlen(B)+1; j++)
      if (verifica_orizzontale(A, B, i, j))
        cont++;

  for (i=0; i<n-strlen(B)+1; i++)
    for (j=0; j<m; j++)
      if (verifica_verticale(A, B, i, j))
        cont++;

  printf("La stringa '%s' e' contenuta %d volte\n", B, cont);
  return(1);
}
```

# Esame del 21 giugno 2002

## Esercizio n.1

Leggere in input un albero $T = (V, E)$ con radice in 0, orientato dalla radice verso le foglie e rappresentarlo mediante liste di adiacenza. Stampare la profondità di $T$, ossia la lunghezza del cammino più lungo dalla radice ad una foglia di $T$.

**Esempio** Consideriamo l'albero $T$ riportato in figura. Il cammino più lungo in $T$ dalla radice 0 ad una foglia è ad esempio $\mathcal{P} = (0, 2, 6, 12, 15)$, quindi la profondità di $T$ è 4. Osserviamo che in realtà i cammini di lunghezza massima in $T$ sono quattro, ma per calcolare la profondità dell'albero sono equivalenti.




### Soluzione

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

#define MAX 50

struct nodo {
  int info;
  struct nodo *next;
};


struct nodo *leggi_lista(void) {
  struct nodo *p, *primo=NULL;
  int i, n;

  printf("Numero di elementi: ");
```

```
  scanf("%d", &n);
  for (i=0; i<n; i++) {
    p = malloc(sizeof(struct nodo));
    scanf("%d", &p->info);
    p->next = primo;
    primo = p;
  }
  return(primo);
}

int leggi_grafo(struct nodo *l[]) {
  int n, i;

  printf("Numero di vertici: ");
  scanf("%d", &n);
  for (i=0; i<n; i++) {
    printf("Vertice %d. ", i);
    l[i] = leggi_lista();
  }
  return(n);
}

int profondita(struct nodo *l[], int i) {
  int prof = 0, q;
  struct nodo *p;

  p = l[i];
  while (p != NULL) {
    q = profondita(l, p->info) + 1;
    if (q > prof)
      prof = q;
    p = p->next;
  }
  return(prof);
}

int main(void) {
  struct nodo *l[MAX];
  int p, n;

  n = leggi_grafo(l);
  p = profondita(l, 0);
  printf("Profondita': %d\n", p);
  return(1);
}
```

## Esercizio n. 2

Leggere in input quattro interi positivi $n, m, p, q$, tali che $0 < p \leq n$ e $0 < q \leq m$. Generare in modo casuale una matrice $M$ di $n \times m$ numeri interi tali che $-50 < M_{i,j} < 50$. Stampare la sottomatrice $M'$ di ordine $p \times q$ tale che la somma dei suoi elementi sia massima.

**Esempio** Si consideri la seguente matrice $M$ di $n = 4$ righe e $m = 5$ colonne:

$$M = \begin{pmatrix} 15 & 4 & -21 & 8 & 17 \\ -34 & -7 & -7 & 4 & 1 \\ 18 & 26 & 9 & 2 & -6 \\ -15 & -5 & 3 & 29 & 8 \end{pmatrix}$$

Siano $p = 2$, $q = 3$. La sottomatrice $M'$ di somma massima è la seguente:

$$M' = \begin{pmatrix} 26 & 9 & 2 \\ -5 & 3 & 29 \end{pmatrix}$$

## Soluzione

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

#define MAX 50

void crea_matrice(int A[MAX][MAX], int *r, int *c) {
  int i, j;

  srand((unsigned)time(NULL));
  printf("Numero di righe e di colonne: ");
  scanf("%d %d", r, c);
  for (i=0; i<*r; i++)
    for (j=0; j<*c; j++)
      A[i][j] = rand() % 100 - 50;
  return;
}

int somma(int A[MAX][MAX], int r, int c, int x, int y) {
  int s = 0, i, j;

  for (i=x; i<x+r; i++)
    for (j=y; j<y+c; j++)
      s = s + A[i][j];
  return(s);
}

void stampa_matrice(int A[MAX][MAX], int x, int y, int r, int c) {
  int i, j;

  for (i=x; i<x+r; i++) {
    for (j=y; j<y+c; j++)
      printf("%3d ", A[i][j]);
    printf("\n");
  }
  return;
}

int main(void) {
  int M[MAX][MAX], n, m, p, q, max, i, j, h=0, k=0, s;
```

```
  crea_matrice(M, &n, &m);
  stampa_matrice(M, 0, 0, n, m);
  printf("Righe e colonne della sotto matrice: ");
  scanf("%d %d", &p, &q);
  max = somma(M, p, q, 0, 0);
  for (i=0; i<n-p+1; i++) {
    for (j=0; j<m-q+1; j++) {
      s = somma(M, p, q, i, j);
      if (max < s) {
        max = s;
        h = i;
        k = j;
      }
    }
  }
  printf("La sottomatrice %dx%d di somma massima (%d) e':\n", p, q, max);
  stampa_matrice(M, h, k, p, q);
  return(1);
}
```

# Esame del 9 settembre 2002

## Esercizio n.1

Generare una matrice di $n$ righe ed $m$ colonne ($n$ e $m$ letti in input) di numeri interi casuali compresi tra 0 e 20. Stampare una colonna qualsiasi fra quelle in cui la massima differenza (in valore assoluto) tra due elementi consecutivi sia minima.

**Esempio** Consideriamo la seguente matrice costituita da $n = 4$ righe ed $m = 3$ colonne:

$$A = \begin{pmatrix} 15 & 13 & 7 \\ 6 & 18 & 4 \\ 11 & 4 & 12 \\ 13 & 9 & 5 \end{pmatrix}$$

La massima differenza tra gli elementi consecutivi della prima colonna è $9 = 15 - 6$; per la seconda colonna è $14 = 18 - 4$; per la terza è $8 = |4 - 12|$. Quindi il programma deve stampare la terza colonna la cui massima differenza è minima.

### Soluzione

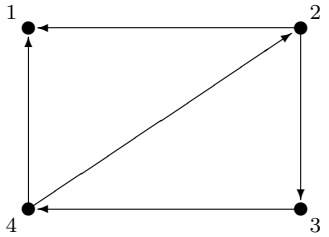
```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

#define MAX 50

void crea_matrice(int A[MAX][MAX], int *r, int *c) {
  int i, j;

  srand((unsigned)time(NULL));
  printf("Numero di righe e di colonne: ");
  scanf("%d %d", r, c);
  for (i=0; i<*r; i++)
    for (j=0; j<*c; j++)
      A[i][j] = rand() % 21;
  return;
}

void stampa_matrice(int A[MAX][MAX], int r, int c) {
  int i, j;
```

```
  for (i=0; i<r; i++) {
    for (j=0; j<c; j++)
      printf("%3d ", A[i][j]);
    printf("\n");
  }
  return;
}

int main(void) {
  int M[MAX][MAX], n, m, i, j, min_max_diff, max_diff, min_max_col;

  crea_matrice(M, &n, &m);
  stampa_matrice(M, n, m);
  min_max_diff = 21;
  min_max_col = -1;
  for (i=0; i<m; i++) {
    max_diff = -1;
    for (j=0; j<n-1; j++)
      if (max_diff < abs(M[j][i] - M[j+1][i]))
        max_diff = abs(M[j][i] - M[j+1][i]);
    if (min_max_diff > max_diff) {
      min_max_diff = max_diff;
      min_max_col = i;
    }
  }
  printf("La colonna con la min. max diff. e' la %d^: (", min_max_col+1);
  for (i=0; i<n; i++)
    printf("%d ", M[i][min_max_col]);
  printf("), massima differenza = %d\n", min_max_diff);
  return(1);
}
```

# Esercizio n. 2

Leggere in input un grafo orientato $G = (V, E)$ e rappresentarlo mediante l'uso di liste di adiacenza. Verificare se $G$ contiene almeno un ciclo $C$ di lunghezza 3 ed in caso affermativo stamparne uno.

**Esempio** Consideriamo il grafo $G = (V, E)$ rappresentato in figura, in cui $V = \{1, 2, 3, 4\}$ e $E = \{(2, 1), (2, 3), (3, 4), (4, 1), (4, 2)\}$:




Il grafo $G$ contiene un ciclo di lunghezza 3, costituito dalla sequenza di vertici

$(2, 3, 4, 2)$. La sequenza $(4, 2, 1, 4)$ invece non costituisce un ciclo, visto che lo spigolo $(1, 4)$ non esiste ($(1, 4) \notin E$).

## Soluzione

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

#define MAX 50

struct nodo {
  int info;
  struct nodo *next;
};

struct nodo *leggi_lista(void) {
  struct nodo *p, *primo=NULL;
  int i, n;

  printf("Numero di elementi: ");
  scanf("%d", &n);
  for (i=0; i<n; i++) {
    p = malloc(sizeof(struct nodo));
    scanf("%d", &p->info);
    p->next = primo;
    primo = p;
  }
  return(primo);
}

int leggi_grafo(struct nodo *l[]) {
  int n, i;

  printf("Numero di vertici: ");
  scanf("%d", &n);
  for (i=0; i<n; i++) {
    printf("Vertice %d. ", i);
    l[i] = leggi_lista();
  }
  return(n);
}

void calcola_ciclo(struct nodo *l[], int n) {
  struct nodo *p, *q, *r;
  int ok = 0, i;

  for (i=0; i<n && !ok; i++) {
    p = l[i];
    while (p != NULL && !ok) {
      q = l[p->info];
      while (q != NULL && !ok) {
        if (q->info != i) {
```

```
          r = l[q->info];
          while (r != NULL && !ok) {
            if (r->info == i)
              ok = 1;
            r = r->next;
          }
        }
        if (!ok)
          q = q->next;
      }
      if (!ok)
        p = p->next;
    }
    if (ok) {
      printf("\nEsiste un ciclo di lunghezza 3: ");
      printf("%d --> %d --> %d --> %d\n", i, p->info, q->info, i);
    }
  }
  if (!ok)
    printf("\nNon esiste un ciclo di ordine 3.\n");
  return;
}

int main(void) {
  struct nodo *l[MAX];
  int n;

  n = leggi_grafo(l);
  calcola_ciclo(l, n);
  return(1);
}
```

# Esonero del 4 novembre 2002

## Esercizio n.1

Leggere $n$ numeri interi positivi e memorizzarli in un array $V$. Costruire un array $W$ di $n$ interi i cui elementi $W_k$ $(0 \leq k < n)$ siano dati dalla somma degli elementi $V_0, \ldots, V_k$ che non siano primi. Stampare gli array $V$ e $W$.

**Esempio** Sia $V = (13, 5, 32, 16, 4, 7, 6)$ il vettore di $n = 7$ elementi acquisito in input. L'array $W$ sarà dato dai seguenti elementi: $W = (0, 0, 32, 48, 52, 52, 58)$, visto che 13, 5 e 7 sono numeri primi.

### Soluzione

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

#define MAX 100

int leggi_array(int V[]) {
  int i, n;

  printf("Numero di elementi: ");
  scanf("%d", &n);
  printf("Inserisci %d numeri interi: ", n);
  for (i=0; i<n; i++)
    scanf("%d", &V[i]);
  return(n);
}

void stampa_array(int V[], int n) {
  int i;

  for (i=0; i<n; i++)
    printf("%2d ", V[i]);
  printf("\n");
  return;
}

int primo(int x) {
  int i, flag;
```

```
  flag = 1;
  for (i=2; i<=x/2 && flag; i++)
    if (x%i == 0)
      flag = 0;
  return(flag);
}

void costruisci(int A[], int B[], int n) {
  int i, j, k;

  for (k=0; k<n; k++) {
    B[k] = 0;
    for (i=0; i<=k; i++)
      if (!primo(A[i]))
        B[k] += A[i];
  }
  return;
}

int main(void) {
  int V[MAX], W[MAX], n;

  n = leggi_array(V);
  costruisci(V, W, n);
  printf("V = ");
  stampa_array(V, n);
  printf("W = ");
  stampa_array(W, n);
  return(1);
}
```

# Esercizio n. 2

Costruire in modo casuale due matrici $A$ e $B$ di ordine $n \times m$, costituite da numeri interi tali che $0 \leq A_{i,j}, B_{i,j} \leq 47$ ($0 \leq i < n$, $0 \leq j < m$). Costruire una terza matrice $C$ di ordine $n \times m$ tale che la $j$-esima colonna di $C$ sia costituita dagli elementi della colonna $j$-esima di $A$ o di $B$ che abbia il massimo prodotto degli elementi.

**Esempio** Consideriamo le seguenti matrici di ordine $3 \times 4$

$$A = \begin{pmatrix} 7 & 25 & 13 & 31 \\ 0 & 35 & 11 & 6 \\ 38 & 46 & 43 & 32 \end{pmatrix} B = \begin{pmatrix} 15 & 34 & 32 & 28 \\ 6 & 17 & 41 & 36 \\ 18 & 31 & 42 & 40 \end{pmatrix}$$

La matrice $C$ costituita dalle colonne di prodotto massimo è la seguente:

$$C = \begin{pmatrix} 15 & 25 & 32 & 28 \\ 6 & 35 & 41 & 36 \\ 18 & 46 & 42 & 40 \end{pmatrix}$$

## Soluzione

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

#define MAX_N 100
#define MAX_M 100

void genera_matrice(int A[MAX_N][MAX_M], int n, int m) {
  int i, j;

  for (i=0; i<n; i++)
    for (j=0; j<m; j++)
      A[i][j] = rand() % 48;
  return;
}

void stampa_matrice(int A[MAX_N][MAX_M], int n, int m) {
  int i, j;

  for (i=0; i<n; i++) {
    for (j=0; j<m; j++)
      printf("%2d ", A[i][j]);
    printf("\n");
  }
  printf("\n");
  return;
}

int prodotto_colonna(int A[MAX_N][MAX_M], int n, int k) {
  int i, p;

  p = A[0][k];
  for (i=1; i<n; i++)
    p = p * A[i][k];
  return(p);
}

void calcola(int A[MAX_N][MAX_M], int B[MAX_N][MAX_M], int C[MAX_N][MAX_M],
int n, int m) {
  int i, k, p1, p2;

  for (k=0; k<m; k++) {
    p1 = prodotto_colonna(A, n, k);
    p2 = prodotto_colonna(B, n, k);
    if (p1>p2)
      for (i=0; i<n; i++)
        C[i][k] = A[i][k];
    else
      for (i=0; i<n; i++)
        C[i][k] = B[i][k];
  }
```

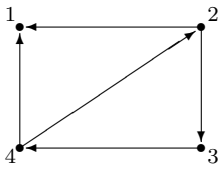
```
  return;
}


int main(void) {
  int A[MAX_N][MAX_M], B[MAX_N][MAX_M], C[MAX_N][MAX_M], n, m;

  printf("Numero di righe: ");
  scanf("%d", &n);
  printf("Numero di colonne: ");
  scanf("%d", &m);
  srand((unsigned)time(NULL));
  genera_matrice(A, n, m);
  genera_matrice(B, n, m);
  calcola(A, B, C, n, m);
  stampa_matrice(A, n, m);
  stampa_matrice(B, n, m);
  stampa_matrice(C, n, m);
  return(1);
}
```

# Esonero dell'8 gennaio 2003

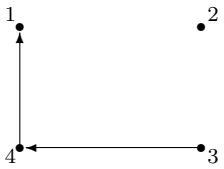
## Esercizio n.1

Letto un grafo orientato $G = (V, E)$, rappresentarlo con liste di adiacenza. Letto in input un vertice $u \in V$, senza modificare le liste di adiacenza di $G$, costruire le liste di adiacenza del grafo orientato $G' = (V, E')$ tale che $E' = \{(v, w) \in E(G) : v, w \neq u\}$. Stampare $G'$.

**Esempio** Consideriamo il grafo $G = (V, E)$ rappresentato in figura, in cui $V = \{1, 2, 3, 4\}$ e $E = \{(2, 1), (2, 3), (3, 4), (4, 1), (4, 2)\}$:




Se si sceglie $u = 2$ allora $G' = (V, E')$, $E' = \{(3, 4), (4, 1)\}$:




### Soluzione

```
#include <stdlib.h>
#include <stdio.h>
#define MAX 20

struct nodo {
  int info;
  struct nodo *next;
};

struct nodo *leggi_lista(void) {
  int i, n;
  struct nodo *p, *primo=NULL;
```

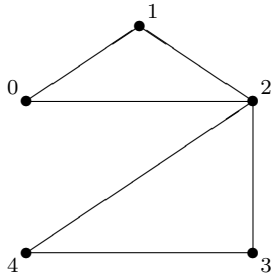
```
  printf("Numero di elementi: ");
  scanf("%d", &n);
  printf("Inserisci %d elementi: ", n);
  for (i=0; i<n; i++) {
    p = malloc(sizeof(struct nodo));
    scanf("%d", &p->info);
    p->next = primo;
    primo = p;
  }
  return(primo);
}

int leggi_grafo(struct nodo *v[]) {
  int i, n;

  printf("Numero di vertici del grafo: ");
  scanf("%d", &n);
  for (i=0; i<n; i++) {
    printf("\nLista di adiacenza del vertice %d\n", i);
    v[i] = leggi_lista();
  }
  return(n);
}

void stampa_lista(struct nodo *p) {
  while (p != NULL) {
    printf("%d --> ", p->info);
    p = p->next;
  }
  printf("NULL\n");
  return;
}

void stampa_grafo(struct nodo *v[], int n) {
  int i;

  printf("\nListe di adiacenza del grafo\n");
  for (i=0; i<n; i++) {
    printf("%d: ", i);
    stampa_lista(v[i]);
  }
  return;
}

void elimina(struct nodo *v[], struct nodo *v1[], int n, int u) {
  int i, j;
  struct nodo *p, *q;

  for (i=0; i<n; i++) {
    v1[i] = NULL;
    if (i != u) {
      p = v[i];
      while (p != NULL) {
```

```
        if (p->info != u) {
          q = malloc(sizeof(struct nodo));
          q->info = p->info;
          q->next = v1[i];
          v1[i] = q;
        }
        p = p->next;
      }
    }
  }
  return;
}

int main(void) {
  struct nodo *v[MAX], *v1[MAX];
  int n, u;

  n = leggi_grafo(v);
  printf("\nVertice da eliminare: ");
  scanf("%d", &u);
  elimina(v, v1, n, u);
  stampa_grafo(v1, n);
  return(1);
}
```

# Esercizio n. 2

Letto un grafo non orientato $G = (V, E)$ rappresentarlo con liste di adiacenza. Letta in input una sequenza di vertici di $V$, rappresentarla mediante una lista $L$. Verificare se $L$ è una copertura di $G$, ossia se per ogni $(u, v) \in E$ risulta $u \in L$ o $v \in L$ (o entrambi).

**Esempio** Consideriamo il grafo $G = (V, E)$ rappresentato in figura, in cui $V = \{0, 1, 2, 3, 4\}$ e $E = \{(0, 1), (0, 2), (1, 2), (2, 3), (2, 4), (3, 4)\}$:




La lista $L = \{1, 2, 3\}$ è una copertura di $G$, mentre $L' = \{0, 1, 3\}$ non è una copertura, infatti gli estremi dello spigolo $(4, 2)$ non appartengono a $L'$.

## Soluzione

```
#include <stdlib.h>
```

```
#include <stdio.h>
#define MAX 20

struct nodo {
  int info;
  struct nodo *next;
};

struct nodo *leggi_lista(void) {
  int i, n;
  struct nodo *p, *primo=NULL;

  printf("Numero di elementi: ");
  scanf("%d", &n);
  printf("Inserisci %d elementi: ", n);
  for (i=0; i<n; i++) {
    p = malloc(sizeof(struct nodo));
    scanf("%d", &p->info);
    p->next = primo;
    primo = p;
  }
  return(primo);
}

int leggi_grafo(struct nodo *v[]) {
  int i, n;

  printf("Numero di vertici del grafo: ");
  scanf("%d", &n);
  for (i=0; i<n; i++) {
    printf("Lista di adiacenza del vertice %d\n", i);
    v[i] = leggi_lista();
  }
  return(n);
}

void stampa_lista(struct nodo *p) {
  while (p != NULL) {
    printf("%d --> ", p->info);
    p = p->next;
  }
  printf("NULL\n");
  return;
}

void stampa_grafo(struct nodo *v[], int n) {
  int i;

  printf("Liste di adiacenza del grafo\n");
  for (i=0; i<n; i++) {
    printf("%d: ", i);
    stampa_lista(v[i]);
  }
  return;
```

```
}

int appartiene(int i, struct nodo *t) {
  int trovato = 0;

  while (t!=NULL && !trovato) {
    if (t->info == i)
      trovato = 1;
    t = t->next;
  }
  return(trovato);
}

int copertura(struct nodo *v[], int n, struct nodo *p) {
  struct nodo *q;
  int i, flag=1;

  for (i=0; i<n && flag; i++) {
    flag = 0;
    if (!appartiene(i, p)) {
      q = v[i];
      flag = 1;
      while (flag && q!=NULL) {
        if (!appartiene(q->info, p))
          flag = 0;
        q = q->next;
      }
    } else {
      flag = 1;
    }
  }
  return(flag);
}

int main(void) {
  struct nodo *v[MAX], *p;
  int n;

  n = leggi_grafo(v);
  p = leggi_lista();
  if (copertura(v, n, p))
    printf("La lista e' una copertura del grafo.\n");
  else
    printf("La lista non e' una copertura del grafo.\n");
  return(1);
}
```

# Esame del 17 gennaio 2003

## Esercizio n.1

Letti in input due interi $n, m > 0$, costruire una matrice $M$ di ordine $n \times m$ di numeri interi tali che $-20 \leq M_{i,j} \leq 20$ per $i = 0, \ldots, n-1$, $j = 0, \ldots, m-1$. Leggere un intero positivo $k$ ed un array $A$ di $2k$ numeri interi positivi. Gli elementi di $A$, considerati a coppie, rappresentano degli spostamenti (legittimi) di un "cursore" sulla matrice $M$, a partire dalla posizione $M_{0,0}$: $A_i$ rappresenta la direzione ($A_i = 1$ alto, $A_i = 2$ destra, $A_i = 3$ basso, $A_i = 4$ sinistra), mentre $A_{i+1}$ rappresenta lo spostamento nella direzione $A_i$. Ad esempio $A_i = 2, A_{i+1} = 3$ significa spostare il cursore a destra di 3 elementi. Calcolare e stampare la somma $s$ degli elementi di $M$ su cui passa il "cursore" nel suo percorso sulla matrice.

**Esempio** Consideriamo la seguente matrice di $n = 4$ righe e $m = 6$ colonne:

$$M = \begin{pmatrix} \mathbf{5} & \mathbf{2} & \mathbf{-3} & \mathbf{10} & -15 & 20 \\ 7 & -8 & -4 & \mathbf{-17} & 11 & 13 \\ 20 & \mathbf{-18} & \mathbf{2} & \mathbf{1} & 1 & 0 \\ 4 & 2 & 3 & -5 & 8 & 9 \end{pmatrix}$$

Sia $V = (2, 3; 3, 2; 4, 2)$; allora, partendo da $M_{0,0} = 5$ il cursore esegue il percorso evidenziato in grassetto sulla matrice: $2, -3, 10; -17, 1; 2, -18)$; quindi la somma è $s = 5 + 2 - 3 + 10 - 17 + 1 + 2 - 18 = -18$.

### Soluzione

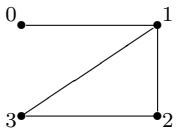
```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

#define MAX 20

void genera_matrice(int M[MAX][MAX], int *n, int *m) {
  int i, j;

  srand((unsigned)time(NULL));
  printf("Numero di righe e colonne: ");
  scanf("%d %d", n, m);
  for (i=0; i<*n; i++)
    for (j=0; j<*m; j++)
      M[i][j] = rand()%41 - 20;
```

```
  return;
}

int leggi_array(int A[]) {
  int i, n;

  printf("Numero di spostamenti sulla matrice: ");
  scanf("%d", &n);
  printf("Inserisci gli spostamenti (direzione, lunghezza): ");
  for (i=0; i<2*n; i++) {
    scanf("%d", &A[i]);
  }
  return(n);
}

int sommatoria(int M[MAX][MAX], int A[], int k) {
  int i, j, x, y, dx, dy, s;

  x = 0;
  y = 0;
  s = M[x][y];
  for (i=0; i<2*k; i+=2) {
    dx = 0;
    dy = 0;
    if (A[i] == 1)
      dx = -1;
    else if (A[i] == 2)
      dy = 1;
    else if (A[i] == 3)
      dx = 1;
    else
      dy = -1;
    for (j=0; j<A[i+1]; j++) {
      x = x+dx;
      y = y+dy;
      s = s + M[x][y];
    }
  }
  return(s);
}

int main(void) {
  int M[MAX][MAX], A[MAX], n, m, k, s;

  genera_matrice(M, &n, &m);
  k = leggi_array(A);
  s = sommatoria(M, A, k);
  printf("La somma e' %d", s);
  return(1);
}
```

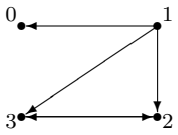
# Esercizio n. 2

Leggere in input un grafo non orientato $G = (V, E)$ e rappresentarlo utilizzando delle liste di adiacenza. Trasformare $G$ in un grafo orientato modificando le liste di adiacenza con cui è stato rappresentato, in modo tale che gli spigoli vengano tutti orientati dal vertice di grado maggiore o uguale a quello di grado minore o uguale.

**Esempio** Consideriamo il grafo $G = (V, E)$ rappresentato in figura, in cui $V = \{0, 1, 2, 3\}$:




Visto che il grado dei vertici è $g(0) = 1, g(1) = 3, g(2) = g(3) = 2$, allora il grafo viene modificato come segue




## Soluzione

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

#define MAX 20

struct nodo {
  int info;
  struct nodo *next;
};

struct nodo *leggi_lista(void) {
  int i, n;
  struct nodo *p, *primo=NULL;

  printf("Numero di elementi: ");
  scanf("%d", &n);
  printf("Inserisci %d elementi: ", n);
  for (i=0; i<n; i++) {
    p = malloc(sizeof(struct nodo));
    scanf("%d", &p->info);
    p->next = primo;
    primo = p;
  }
  return(primo);
}
```

```
int leggi_grafo(struct nodo *v[]) {
  int i, n;

  printf("Numero di vertici del grafo: ");
  scanf("%d", &n);
  for (i=0; i<n; i++) {
    printf("Lista di adiacenza del vertice %d\n", i);
    v[i] = leggi_lista();
  }
  return(n);
}

void stampa_lista(struct nodo *p) {
  while (p != NULL) {
    printf("%d --> ", p->info);
    p = p->next;
  }
  printf("NULL\n");
  return;
}

void stampa_grafo(struct nodo *v[], int n) {
  int i;

  printf("Liste di adiacenza del grafo\n");
  for (i=0; i<n; i++) {
    printf("%d: ", i);
    stampa_lista(v[i]);
  }
  return;
}

void calcola_grado(struct nodo *a[], int b[], int c) {
  int i;
  struct nodo *p;

  for (i=0; i<c; i++) {
    b[i] = 0;
    p = a[i];
    while (p != NULL) {
      b[i]++;
      p = p->next;
    }
  }
  return;
}

void modifica_grafo(struct nodo *a[], int b[], int c) {
  int i;
  struct nodo *p, *prec;

  for (i=0; i<c; i++) {
    p = a[i];
```

```
    prec = NULL;
    while (p != NULL) {
      if (b[i] < b[p->info]) {
        if (prec == NULL) {
          a[i] = p->next;
          free(p);
          p = a[i];
        } else {
          prec->next = p->next;
          free(p);
          p = prec->next;
        }
      } else {
        prec = p;
        p = p->next;
      }
    }
  }
  return;
}

int main(void) {
  int n, g[MAX];
  struct nodo *v[MAX];

  n = leggi_grafo(v);
  calcola_grado(v, g, n);
  modifica_grafo(v, g, n);
  stampa_grafo(v, n);
  return(1);
}
```

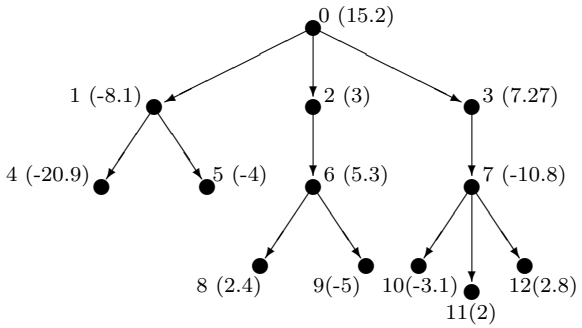
# Esame del 7 febbraio 2003

## Esercizio n.1

In un albero orientato $T = (V, E)$ con radice nel vertice $0 \in V$ ad ogni vertice $i \in V$ è assegnato un peso non intero $w_i$, negativo o non negativo. Il peso di un sottoalbero di $T$ con radice nel vertice $i$ è definito come la somma dei pesi di tutti i vertici discendenti di $i$ in $T$.

Letto in input l'albero rappresentarlo mediante liste di adiacenza; quindi leggere in input i pesi assegnati ai vertici di $T$ e stampare il vertice radice del sottoalbero di peso massimo.

**Esempio** Consideriamo l'albero $T = (V, E)$ rappresentato in figura, in cui i pesi assegnati ai vertici sono riportati tra parentesi accanto all'etichetta del vertice stesso.




Il sottoalbero di peso massimo è $T'$ con radice in 2, i cui vertici sono $V' = \{2, 6, 8, 9\}$ e il cui peso è quindi $w_2 + w_6 + w_8 + w_9 = 3 + 5.3 + 2.4 - 5 = 5.7$.

### Soluzione

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

#define MAX 20

struct nodo {
  int info;
  struct nodo *next;
};
```

```
struct nodo *leggi_lista(void) {
  struct nodo *p, *primo = NULL;
  int i, n;

  printf("Numero di elementi: ");
  scanf("%d", &n);
  printf("inserisci %d elementi: ", n);
  for (i=0; i<n; i++) {
    p = malloc(sizeof(struct nodo));
    scanf("%d", &(p->info));
    p->next = primo;
    primo = p;
  }
  return(primo);
}

int leggi_grafo(struct nodo *V[]) {
  int i, n;

  printf("Numero di vertici: ");
  scanf("%d", &n);
  for (i=0; i<n; i++)  {
    printf("\nLista di adiacenza del vertice %d\n", i);
    V[i] = leggi_lista();
  }
  return(n);
}

void leggi_pesi(float W[], int n) {
  int i;

  printf("Inserisci i pesi associati ai vertici dell'albero: ");
  for (i=0; i<n; i++)
    scanf("%f", &W[i]);
  return;
}

float peso_sottoalbero(struct nodo *V[], float W[], int i, float P[]) {
  struct nodo *p;
  float peso;

  peso = W[i];
  p = V[i];
  while (p != NULL) {
    peso = peso + peso_sottoalbero(V, W, p->info, P);
    p = p->next;
  }
  P[i] = peso;
  return(peso);
}

int main(void) {
  struct nodo *V[MAX];
```

```
  float W[MAX], P[MAX];
  int n, max, i;

  n = leggi_grafo(V);
  leggi_pesi(W, n);
  peso_sottoalbero(V, W, 0, P);
  max = 0;
  for (i=1; i<n; i++)
    if (P[max]<P[i])
      max = i;
  printf("Il sottoalbero di peso massimo ha radice in %d\n", max);
  return(1);
}
```

# Esercizio n. 2

C'era una volta una fabbrica di automobili in cui, per premiare l'impegno degli operai, venivano pagati gli straordinari a quanti alla fine del mese avevano lavorato mediamente più di 8 ore al giorno. Il calcolo degli straordinari veniva effettuato al termine di ogni mese sulla base dei dati registrati dalla macchina per la timbratura dei cartellini. La macchina produceva un output costituito da una tabella sulle cui righe venivano riportati il numero di matricola dell'operaio, l'ora e i minuti della registrazione ed un *flag* (0/1) che indicava se l'ora registrata corrispondeva all'inizio (1) o alla fine (0) di un turno. Dopo aver letto in input la tabella di numeri interi, per ogni operaio calcolare e stampare il numero di ore (e minuti) lavorate complessivamente.

**Esempio** Dai dati presenti nella seguente tabella risulta che l'operaio con matricola 10097 ha svolto complessivamente 9 ore e 51 minuti di lavoro. I dati della tabella di esempio sono parziali: nella tabella letta in input sono presenti i dati relativi all'intero mese.

| Matr. | Ora | Min. | In/Out |
|---|---|---|---|
| 10097 | 7 | 32 | 1 |
| 12406 | 7 | 35 | 1 |
| 20941 | 7 | 54 | 1 |
| 10097 | 12 | 45 | 0 |
| 20941 | 12 | 57 | 0 |
| 10097 | 13 | 4 | 1 |
| 12406 | 13 | 10 | 0 |
| 10097 | 17 | 42 | 0 |

## Soluzione

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

#define MAX 100

int leggi_tabella(int M[MAX][4]) {
  int i, n;
```

```
  printf("Numero di righe: ");
  scanf("%d", &n);
  for (i=0; i<n; i++)
    scanf("%d %d %d %d", &M[i][0], &M[i][1], &M[i][2], &M[i][3]);
  return(n);
}

int calcola_ore(int M[MAX][4], int S[MAX][2], int n) {
  int m, i, j;

  m = 0;
  for (i=0; i<n; i++) {
    for (j=0; j<m && M[i][0]!=S[j][0]; j++);
    if (j==m) {
      S[j][0] = M[i][0];
      S[j][1] = 0;
      m++;
    }
    if (M[i][3] == 1)
      S[j][1] = S[j][1] - (M[i][1]*60 + M[i][2]);
    else
      S[j][1] = S[j][1] + (M[i][1]*60 + M[i][2]);
  }
  return(m);
}

void stampa_ore(int S[MAX][2], int m) {
  int i;

  for (i=0; i<m; i++) {
    printf("Matricola:    %d\n", S[i][0]);
    printf("Ore lavorate: %dh%d'\n\n", S[i][1]/60, S[i][1]%60);
  }
  return;
}

int main(void) {
  int Tabella[MAX][4], Ore[MAX][2], n, m;

  n = leggi_tabella(Tabella);
  m = calcola_ore(Tabella, Ore, n);
  stampa_ore(Ore, m);
  return(1);
}
```

# Indice