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

Luglio 2000

Questo fascicolo è stato impaginato utilizzando il software LaTeX 2.09 in ambiente Linux.

Ultimo aggiornamento: 16 luglio 2000

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) {
```

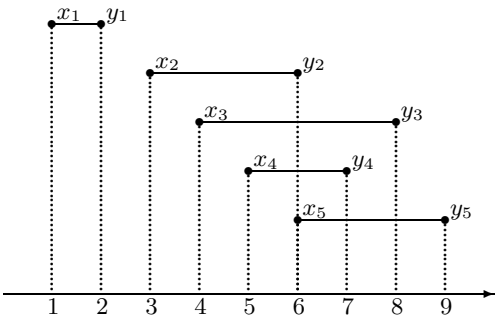
```
    printf("%d -> ", p->info);
    p = p->next;
  }
  printf("NULL\n");
  return;
}

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

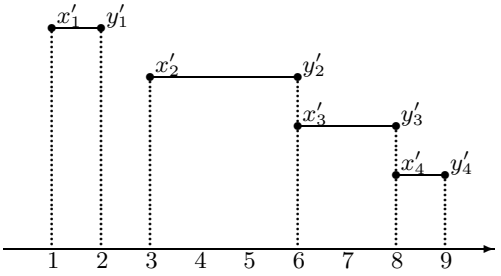
Leggere una lista di coppie di numeri interi $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ tali che $x_i < y_i, \forall i = 1, ..., n$; ogni coppia (x_i, y_i) rappresenta un intervallo sulla retta. Riordinare gli elementi della lista in modo tale che $x_1 \leq x_2 \leq ... \leq x_n$. Costruire una nuova 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 3×3 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;
```

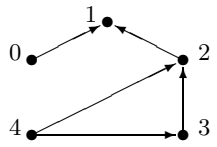
```
    }
    ultimo = p;
  }
  return;
}

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:

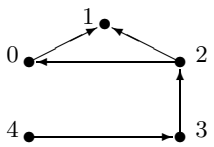


Fig. 1

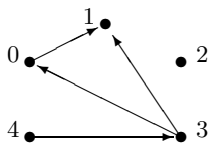


Fig. 2

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:

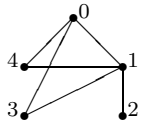
1. $V = \{0, ..., n-1\}$;
2. siano L_i ed L_j $(i \neq j)$ due delle n liste; se $L_i \cap L_j \neq \emptyset$ allora $(i, j) \in E$.

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);
```

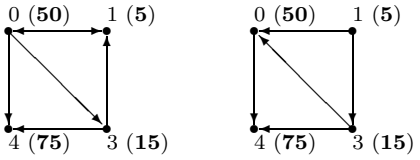
```
  for (i=0; i<n; i++)
    L[i] = leggi_lista();
  costruisci(L,G,n);
  stampa_grafo(G,n);
  return(1);
}
```

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'articolo

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 20×40 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));
```

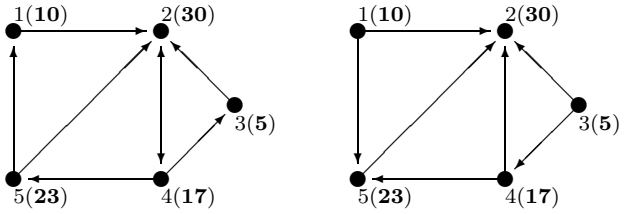
```
  for (i=0; i<RIGHE; i++)
    for (j=0; j<COLONNE; j++)
      m[i][j] = rand() % 2;
  return;
}

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

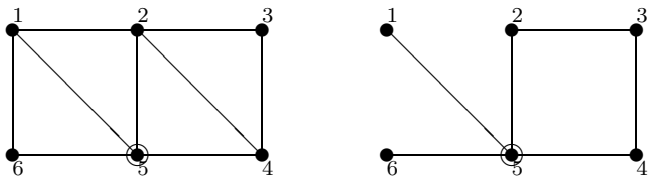
```
#include <stdlib.h>
#include <stdio.h>

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);
}
```

Indice