Esercizi su matrici in C

Stefano Cherubin*

06/11/2015

[**Informatica A**] Esercitazione #8

corso per Ing. Gestionale a.a. 2015/16

*<nome.cognome>@polimi.it

Indice

1 Simmetria rispetto alla diagonale secondaria

Leggere una matrice quadrata di numeri reali dallo standard input, determinare se si tratta di una matrice "simmetrica rispetto alla diagonale secondaria" e in caso contrario annullare le diagonali principali della matrice inserita e stampare a video la matrice risultato.

1.1 Soluzione C

Listato 1: Simmetria diagonale secondaria

```
1  #include <stdio.h>
2
3  #define MAXLEN 30
4  int main( ) {
5    int main( ) {
6    float mat[MAXLEN][MAXLEN];
7    int i, j, n;
8    int flag;
9    /* acquisizione dati*/
10   do {
11     printf("\nOrdine della matrice 0 < n <=
          MAXLEN : ");
12     scanf("%d", &n);
13   } while ( n <= 0 || n > MAXLEN);
14   printf("\nInserire coeff. matrice di ordine %
       d \n", n);
15   /* Lettura della matrice */
16   for (i = 0; i < n; i++)
17     for (j = 0; j < n; j++) {
18       printf("\n mat[%d][%d] := ", i, j);
19       scanf("%f", &mat[i][j]);
20     }
21   /* Verifica simmetria rispetto alla diagonale
        secondaria */
22   flag = 1;
23   for (i = n - 1; i > 0 && flag == 1 ; --i)
24     for (j = n - 1; j > n - 1 - i && flag == 1;
          --j)
25       if (mat[i][j] != mat[n-1-j][n-1-i])
26         flag = 0;
27   if (flag == 0) {
28     printf("\nLa matrice inserita non e'
         simmetrica!");
```

```
29      printf("\nAnnullo le diagonali principali
           .....\n");
30      for (i = 0; i < n; i++) {
31        mat[i][i] = 0;
32        mat[i][n-i-1] = 0;
33      }
34      printf("\n i valori nella matrice sono:\n\n
           ");
35      for (i = 0; i < n ; i++) {
36        for (j = 0; j < n ; j++)
37          printf("%f", mat[i][j]);
38      }
39      printf("\n");
40    } else {
41      printf("\n la matrice inserita e'
           simmetrica!");
42    }
43    return 0;
44  }
```

2 Quadrato magico

> i progetti e codifichi una funzione C che avendo come parametri d'ingresso una matrice di interi e l'ordine di tale matrice, riempia le celle della matrice con i valori corrispondenti di un quadrato magico di dimensione n, con n dispari.
>
> Un quadrato magico di ordine n contiene i primi n numeri naturali $\left(1, 2, 3, \ldots, n^2\right)$ disposti in modo tale che la somma dei numeri su ogni riga, su ogni colonna e sulle due diagonali principali sia sempre la stessa.

Esiste una regola molto semplice per percorrere la matrice disponendo i numeri interi in ordine crescente.

Partendo col posizionare un 1 nella posizione centrale sull'ultima riga, si percorre la matrice incrementando di una unità il numero di riga e il numero di colonna dell'elemento attuale, avendo cura di considerare i bordi opposti della matrice come adiacenti.

- Se durante questa operazione si individua una cella vuota si scrive il numero con valore successivo a quello della cella di partenza;
- altrimenti, il numero successivo, viene posizionato nella cella avente riga immediatamente precedente a quella della cella di partenza.

Es: n = 3

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{matrix} \quad \begin{matrix} 0 & 0 & 2 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{matrix} \quad \begin{matrix} 0 & 0 & 2 \\ 3 & 0 & 0 \\ 0 & 1 & 0 \end{matrix} \quad \begin{matrix} 4 & 0 & 2 \\ 3 & 0 & 0 \\ 0 & 1 & 0 \end{matrix}$$

$$\begin{matrix} 4 & 0 & 2 \\ 3 & 5 & 0 \\ 0 & 1 & 0 \end{matrix} \quad \begin{matrix} 4 & 0 & 2 \\ 3 & 5 & 0 \\ 0 & 1 & 6 \end{matrix} \quad \begin{matrix} 4 & 0 & 2 \\ 3 & 5 & 7 \\ 0 & 1 & 6 \end{matrix} \quad \begin{matrix} 4 & 0 & 2 \\ 3 & 5 & 7 \\ 8 & 1 & 6 \end{matrix}$$

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

2.1 Soluzione C

Listato 2: Quadrato magico

```
1  #include <stdio.h>
2  #define MAX_DIM 51
3  void quadratoMagico(int mat[][MAX_DIM], int n);
4  int main( ) {
5    int matrix[MAX_DIM][MAX_DIM];
6    int i, j, lim, sum;
```

```
7     do {
8       printf("\n dim. quadrato (dispari <= %d):",
            MAX_DIM);
9       scanf("%d", &lim);
10    } while ( lim > MAX_DIM || lim % 2 == 0 );
11    quadratoMagico(matrix, lim);
12    sum = 0;
13    for (j = 0; j < lim; j++)
14      sum += matrix[0][j];
15    printf("\nIl quadrato magico di ordine %d è:\
         n", lim);
16    printf("\nLa somma su ogni linea è %d.\n",
         sum);
17    for (i = 0; i < lim; i++) {
18      printf("\n");
19      for (j = 0; j < lim; j++)
20        printf("%4d", matrix[i][j]);
21    }
22    return 0;
23  } // end main
24
25  void quadratoMagico(int mat[][MAX_DIM], int n){
26    int i, j, k ;
27    for (i = 0; i < n; i++)
28      for (j = 0; j < n; j++)
29        mat[i][j] = 0;
30    i = n - 1;
31    j = n / 2;
32    for (k = 0; k < n * n; k++) {
33    /* passa alla riga immediatamente precedente
         */
34      if (mat[i][j] != 0)
35        i = (i - 1) % n;
36      mat[i][j] = k + 1;
37  /* indice + 1 % n consente di restare sempre
       nei limiti, passando alla riga o colonna
       successiva considerando adiacenti la l'indice
        n-1 e 0. Infatti n - 1 + 1 % n = 0 */
38      i = (i + 1) % n;
39      j = (j + 1) % n;
40    }
41  } // end quadratoMagico
```

3 Tappeto elastico

```
#define DIM 10
typedef struct Intv { unsigned int r, c;
    } Casella;
typedef Casella TappetoElastico[DIM][DIM
   ];
```

Pare che tra le discipline degli ottocenteschi "circhi delle pulci" non ci fosse il tappeto elastico (non in quello del Prof. Heckler, almeno). Avrebbe potuto funzionare così:

> la pulce salta sulla prima cella (0,0) del tappeto elastico (quadrato), atterrandovi legge le coordinate (riga e colonna) della prossima cella su cui saltare, e da lì continua a saltare, ogni volta leggendo le coordinate della cella successiva verso cui saltare. Se/quando le coordinate lette indicano un punto esterno al tappeto, la pulce scende (tra gli applausi del pubblico).
>
> Si implementino le seguenti funzioni in C e si spieghi brevemente come funzionano gli algoritmi usati.
>
> che riceve in input un tappeto elastico e restituisce 1 se esso obbliga a saltare indefinitamente, 0 se invece a un certo punto la pulce potrà scendere
>
> che riceve un tappeto elastico e misura il numero di salti che la pulce compie prima di scendere (se il tappeto non è ciclico), oppure 1
>
> che controlla se un tappeto è adatto all'esibizione di coppia: una seconda pulce inizia a saltare dalla casella (DIM1, DIM1) contemporaneamente alla prima pulce, ed esse continuano a saltare e atterrare in perfetta sincronia fino a uscire assieme dal tappeto, senza mai "scontrarsi", cioè atterrare contemporaneamente sulla stessa casella

3.1 Soluzione C

Listato 3: ciclico

```
1 int ciclico(TappetoElastico te) {
2 /* tappeto aus. per tenere traccia delle
     caselle attraversate */
3   int tec[DIM][DIM] = { 0 };
4 /* flag per controllare se la destinazione del
     salto è fuori dal tappeto: */
5   int jmpOff;
6   jmpOff == 1
7 /* esiste un unico punto d'ingresso al tappeto:
      (0,0) */
```

```
 8   int r = 0, c = 0;
 9   do {
10     tec[r][c] = 1;
11     r = te[r][c].r;
12     c = te[r][c].c;
13     jmpOff = !(r >= 0 && r < DIM && c >= 0 && c
           < DIM);
14   } while (jmpOff == 0 && tec[r][c] == 0);
15 /* return 1 se il percorso della pulce è
     ciclico: cioè le coordinate di arrivo del
     prossimo salto (r,c) indicano una posizione
     già attraversata: tec[r][c] == 0  in questo
     caso il valore di jmpOff e': jmpOff == 0
16 return 0, se il percorso della pulce termina:
     jmpOff == 1 */
17   return !JmpOff;
18 }
19
20 /* Contasalti Versione 1 */
21 int contasalti(TappetoElastico te) {
22   int jmpOff, jmpCount = 0, r = 0, c = 0;
23   do {
24     jmpCount++;
25     r = te[r][c].r;
26     c = te[r][c].c;
27   } while(r >= 0 && r < DIM && c >= 0 && c <
       DIM && jmpCount <= DIM * DIM + 1);
28   /* troppi salti, era un ciclo infinito */
29   if (jmpCount > DIM * DIM + 1)
30     return  1;
31   return 1 + jmpCount;
32 }
33
34 /* Contasalti Versione 2 */
35 int contasalti(TappetoElastico te) {
36   int jmpCount = 0, r = 0, c = 0;
37   if (ciclico(te) == 1)
38     return  1;
39   do {
40     jmpCount++;
41     r = te[r][c].r;
42     c = te[r][c].c;
43   } while (r >= 0 && r < DIM && c >= 0 && c <
       DIM);
44   return 1 + jmpCount;
45 }
```

```
46
47 /* Contasalti Versione 3   */
48 int contasalti(TappetoElastico te)  {
49   int tec[DIM][DIM] = { 0 };
50   int jmpOff, jmpCount = 0, r = 0, c = 0;
51   do {
52     tec[r][c] = 1;
53     jmpCount++;
54     r = te[r][c].r;
55     c = te[r][c].c;
56     jmpOff = !(r >= 0 && r < DIM && c >= 0 && c
           < DIM);
57   } while (jmpOff == 0 && tec[r][c] == 0);
58   if (jmpOff == 1)
59     return 1 + jmpCount;
60   return  1;
61 }
62
63 int dicoppia(TappetoElastico te) {
64   int tec1[DIM][DIM] = { 0 }, tec2[DIM][DIM] =
       { 0 };
65   int jmpOff1, jmpOff2, r1 = 0, c1 = 0, r9 =
       DIM1, c9 = DIM1;
66   do {
67     tec1[r1][c1] = 1;
68     tec2[r9][c9] = 1;
69     r1 = te[r1][c1].r;
70     c1 = te[r1][c1].c;
71     r9 = te[r9][c9].r;
72     c9 = te[r9][c9].c;
73     jmpOff1 = !(r1 >= 0 && r1 < DIM && c1 >= 0
         && c1 < DIM);
74     jmpOff2 = !(r9 >= 0 && r9 < DIM && c9 >= 0
         && c9 < DIM);
75   } while ( jmpOff1 == 0 && jmpOff2 == 0 && !(
       r1 == r9 && c1 == c9) && tec1[r1][c1] == 0
       && tec2[r9][c9] == 0 );
76   if (jmpOff1 == 1 && jmpOff2 == 1 && !(r1 ==
       r9 && c1 == c9))
77     return 1;
78   return 0;
79 }
```

Licenza e crediti

Crediti

Quest'opera contiene elementi tratti da materiale di Gerardo Pelosi redatto per il corso di Fondamenti di Informatica per Ingegneria dell'Automazione a.a. 2014/15.

Licenza beerware[1]

Quest'opera è stata redatta da Stefano Cherubin. Mantenendo questa nota, puoi fare quello che vuoi con quest'opera. Se ci dovessimo incontrare e tu ritenessi che quest'opera lo valga, in cambio puoi offrirmi una birra.

[1] http://people.freebsd.org/~phk/