Programmazione I

Il Linguaggio C

Strutture Dati

Daniel Riccio

Università di Napoli, Federico II

22 ottobre 2021

Sommario

- Argomenti
  - Esercizi su vettori e matrici
    - Sottosequenza comune
    - Rilevazione dei duplicati
    - Perimetro di un poligono
    - Concorso di intelligenza

Verifica – vettori

```
int vett[10];
```

Di che tipo è **vett**[4]?

Di che tipo è **vett**?

```
int vett[]={0,0,0,0,0};
int vett[5]={0};
```

Sono equivalenti?

Dichiarare un vettore di interi che contenga gli elementi 1,3,2,4,6,5,8,7

```
int vett[]={1,3,2,4,6,5,8,7};
```

Verifica – vettori

Dichiarare una matrice di interi che contenga gli elementi
1,3,2,
4,6,5,
8,7,9

```
int mat[3][3]={ {1,3,2},
                {4,6,5},
                {8,7,9}
              };
```

Scrivere un ciclo for che scorra la terza riga della matrice
8,7,9

```
for(i=0; i<3; i++)
  mat[2][i];
```

Scrivere un ciclo for che scorra la seconda colonna della matrice
3,
6,
7

```
for(i=0; i<3; i++)
  mat[i][1];
```

Verifica – vettori

```
int mat[3][3]={ {1,3,2},
                {4,6,5},
                {8,7,9}
              };
```

Scrivere la parte di codice per calcolare il massimo della matrice

```
int massimo = mat[0][0];
for(int i=0; i<3; i++)
  for(int j=0; j<3; j++)
    if(massimo < mat[i][j])
      massimo = mat[i][j];
```

Verifica – vettori

```
int mat[3][3]={ {1,3,2},
                {4,6,5},
                {8,7,9}
              };
```

Scrivere la parte di codice per calcolare il massimo della matrice con un solo ciclo for

Impiega un numero di passi maggiore o minore del precedente?

```
int massimo = mat[0][0];
int riga = 0;
int col = 0;

for(int ind=0; ind<9; ind++){
   riga = ind/3;
   col = ind%3;
   if(massimo < mat[riga][col])
         massimo = mat [riga][col];
}
```

Verifica – vettori

vett

1	3	2	4	6	5	8	7	9

mat

1	3	2
4	6	5
8	7	9

Scrivere la parte di codice per mappare **vett** in **mat**

```
int mat[3][3];
int vett[9]={1,3,2,4,6,5,8,7,9};
```

```
int riga = 0;
int col = 0;

for(int ind=0; ind <9; ind++){
   riga = ind/3;
   col = ind%3;
   mat [riga][col] = vett[ind];
}
```

Verifica – vettori

mat

1	3	2
4	6	5
8	7	9

vett

1	3	2	4	6	5	8	7	9

Scrivere la parte di codice per mappare **mat** in **vet**

```
int mat[3][3]={ {1,3,2},
                {4,6,5},
                {8,7,9}
              };

int vett[9]={0};
```

```
int riga = 0;
int col = 0;
int ind = 0;

for(riga=0; riga <3; riga ++)
  for(col=0; col <3; col ++){
    ind = 3*riga + col;
    vett[ind] = mat [riga][col];
  }
```

Esercizio – sottosequenza comune

Due colleghi intendono fissare una riunione,

pertanto devono identificare dei giorni nei quali sono entrambi liberi da impegni.

A tale scopo, essi realizzano un programma C che permetta a ciascuno di immettere le proprie disponibilità, e che identifichi i giorni nei quali entrambi sono liberi

Esercizio – sottosequenza comune

In particolare, in una prima fase il programma acquisisce le disponibilità dei due colleghi

Per ciascun collega il programma acquisisce un elenco di numeri interi (supponiamo compresi tra 1 e 31), che indicano i giorni del mese in cui essi sono disponibili.

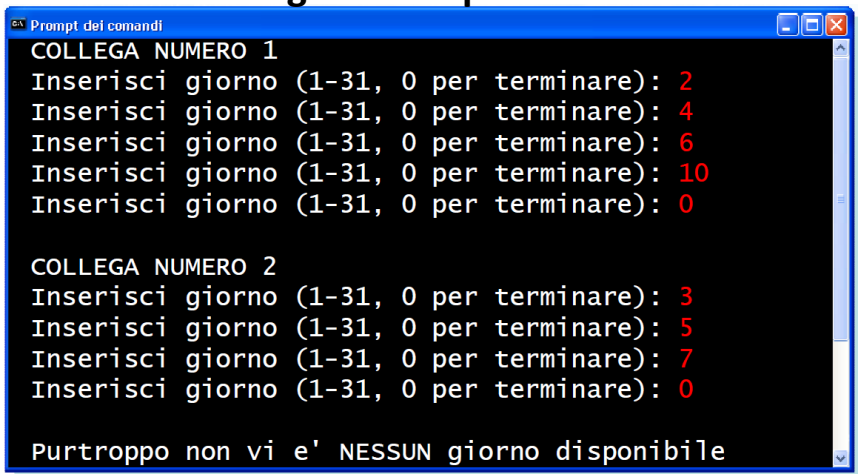
L'immissione dei dati termina inserendo 0.

Nella seconda fase, il programma identificherà i giorni in cui entrambi i colleghi sono disponibili, e li stamperà a video

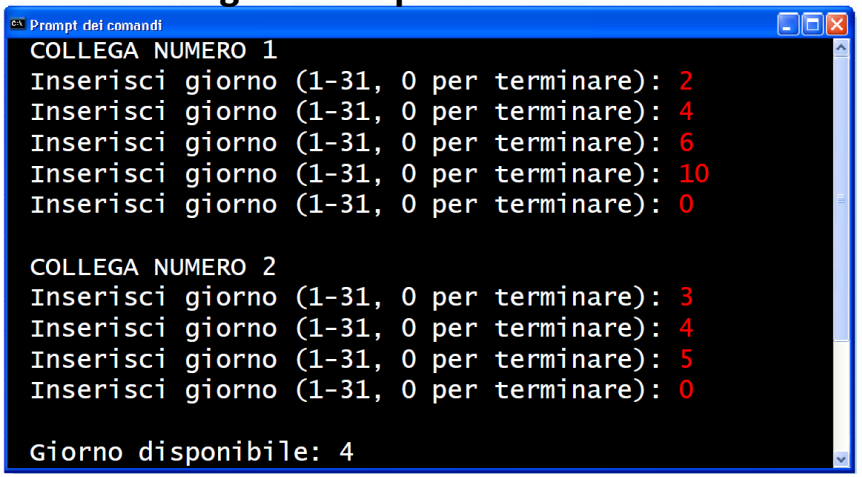
Esercizio – sottosequenza comune

Esempio:

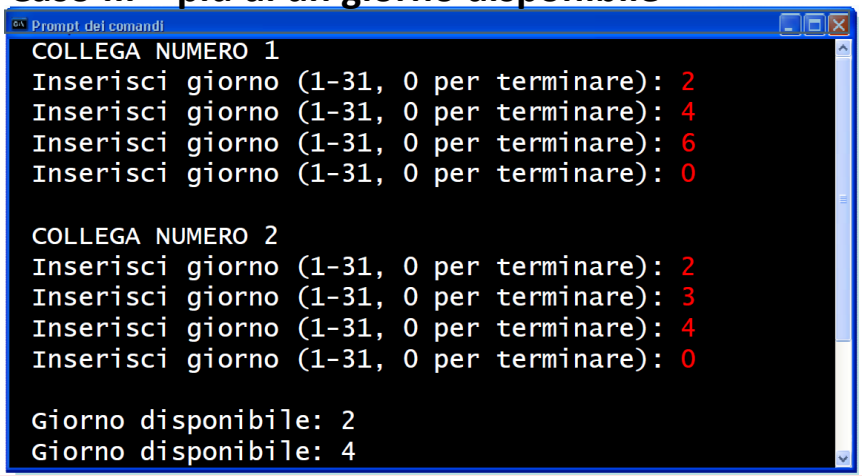
Caso I – nessun giorno disponibile



Caso II – un giorno disponibile



Caso III – più di un giorno disponibile



Esercizio – sottosequenza comune

Acquisisci le disponibilità del collega 1

- Vettore giorni1[] di N1 elementi

Acquisisci le disponibilità del collega 2

- Vettore giorni2[] di N2 elementi

Verifica se vi sono elementi di giorni1[] che siano anche elementi di giorni2[]

- Se si, stampa tali elementi
- Se no, stampa un messaggio

Esercizio – sottosequenza comune

giorni1

2	8	4	12	7	21	18	22	9	10	25	30	3	17	29

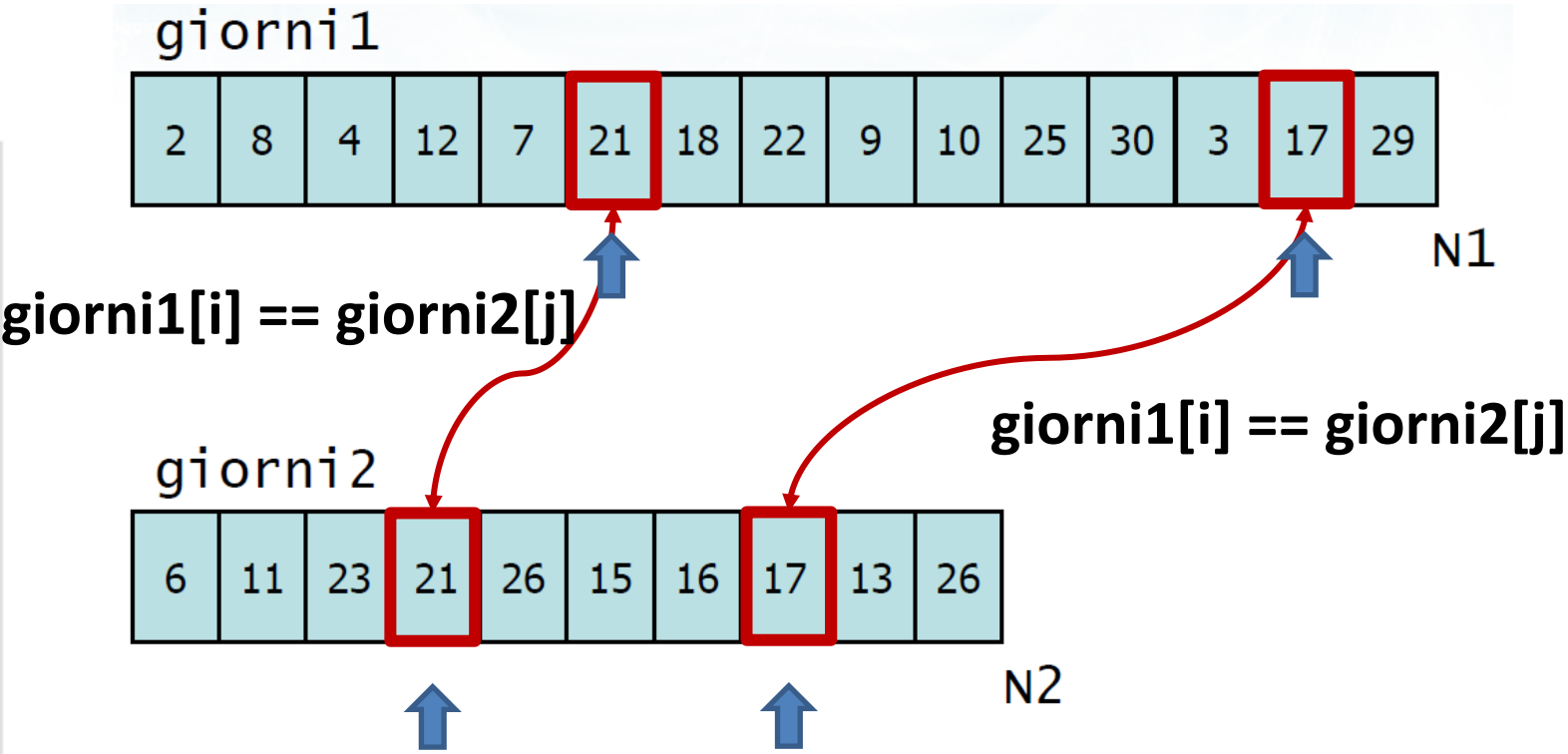
N1

giorni2

6	11	23	21	26	15	16	17	13	26

N2

Esercizio – sottosequenza comune



Esercizio – sottosequenza comune

```
#define MAXN 100          /* dimensione massima del vettore */

int N1, N2;

int giorni1[MAXN];        /* giorni collega 1 */

int giorni2[MAXN];        /* giorni collega 2 */

int giorno;

int i, j;

int trovato;              /* flag: giorni1[i] in giorni2[]? */

int fallito;              /* flag: trovato almeno un giorno? */
```

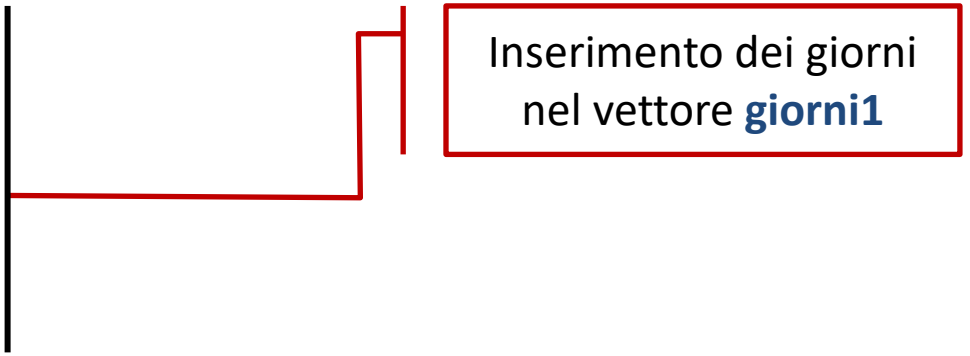
Esercizio – sottosequenza comune

```
/* DISPONIBILITA' COLLEGA 1 */
printf("COLLEGA NUMERO 1\n");

N1 = 0 ;
printf("Inserisci giorno (1-31): ");
scanf("%d", &giorno) ;

while( giorno != 0 )
{
        giorni1[N1] = giorno ;
        N1++ ;
        printf("Inserisci giorno (1-31): ");
        scanf("%d", &giorno) ;
}

printf("Collega 1 ha inserito %d giorni\n", N1);
```



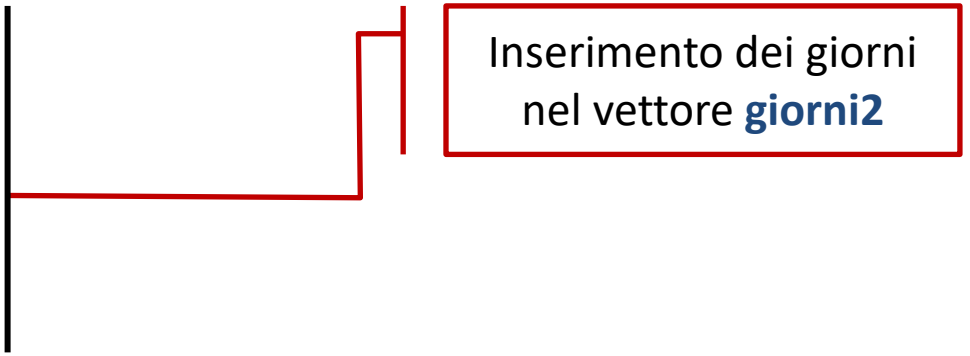
Esercizio – sottosequenza comune

```
/* DISPONIBILITA' COLLEGA 2 */
printf("COLLEGA NUMERO 2\n");

N2 = 0 ;
printf("Inserisci giorno (1-31): ");
scanf("%d", &giorno) ;

while( giorno != 0 )
{
        giorni2[N2] = giorno ;
        N2++ ;
        printf("Inserisci giorno (1-31): ");
        scanf("%d", &giorno) ;
}

printf("Collega 2 ha inserito %d giorni\n", N2);
```



Esercizio – sottosequenza comune

```
/* RICERCA DEGLI ELEMENTI COMUNI */
fallito = 1 ;
/* Per ogni giorno del collega 1... */
for( i=0 ; i<N1; i++ )
{
      trovato = 0 ;
      for( j=0; j<N2; j++ ){
         if( giorni2[j] == giorni1[i] )
            trovato = 1 ;
      }
      /* ...in caso affermativo stampalo */
      if( trovato == 1 ){
         printf("Giorno disponibile: %d\n", giorni1[i]) ;
         fallito = 0 ;
      }
}
/* Se non ne ho trovato nessuno, stampo un messaggio */
if(fallito==1)
   printf("NESSUN giorno disponibile\n");
```

Esercizio – sottosequenza comune

Esecuzione

```
COLLEGA NUMERO 1
Inserisci giorno (1-31): 1
Inserisci giorno (1-31): 4
Inserisci giorno (1-31): 8
Inserisci giorno (1-31): 10
Inserisci giorno (1-31): 12
Inserisci giorno (1-31): 25
Inserisci giorno (1-31): 0
Collega 1 ha inserito 6 giorni
COLLEGA NUMERO 2
Inserisci giorno (1-31): 4
Inserisci giorno (1-31): 9
Inserisci giorno (1-31): 10
Inserisci giorno (1-31): 11
Inserisci giorno (1-31): 25
Inserisci giorno (1-31): 29
Inserisci giorno (1-31): 0
Collega 2 ha inserito 6 giorni
Giorno disponibile: 4
Giorno disponibile: 10
Giorno disponibile: 25
```

Quanti passi sono necessari al programma per trovare i giorni comuni?

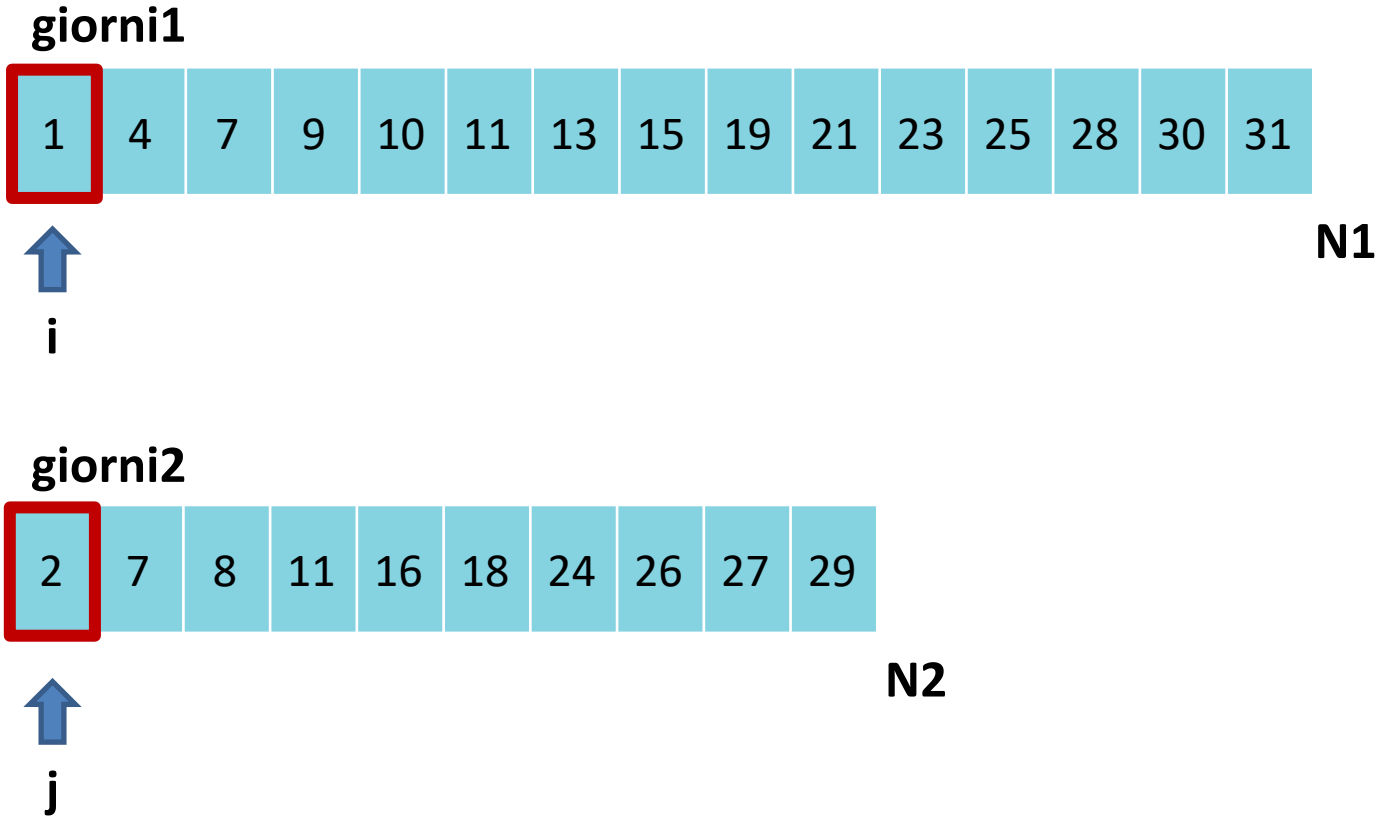
Ciclo esterno **N1** passi
  Ciclo interno **N2** passi

Numero di passi ≈ **N1** × **N2**

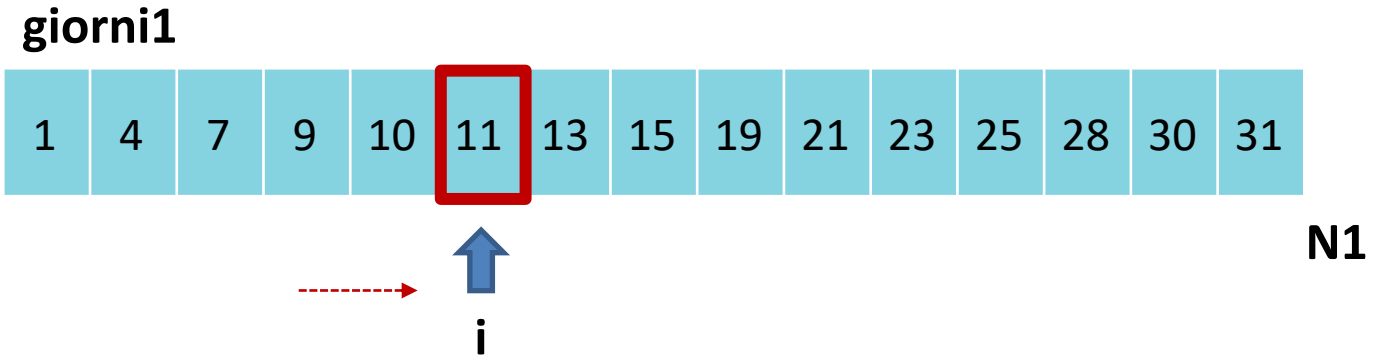
Si può fare di meglio?

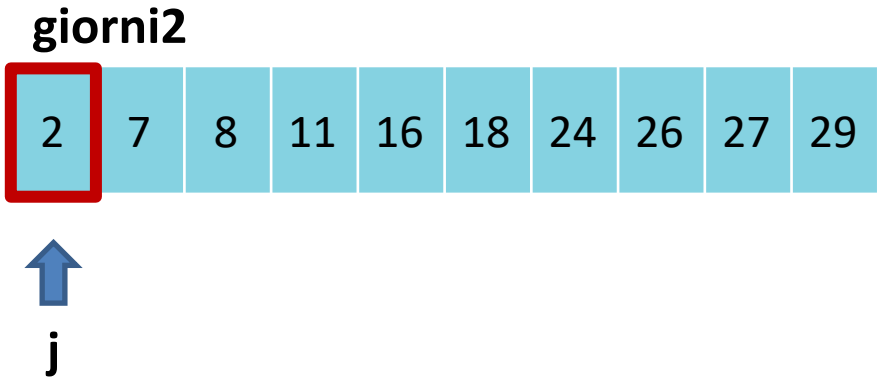
Si osservi che, generalmente, i giorni vengono inseriti in modo ordinato

Esercizio – sottosequenza comune



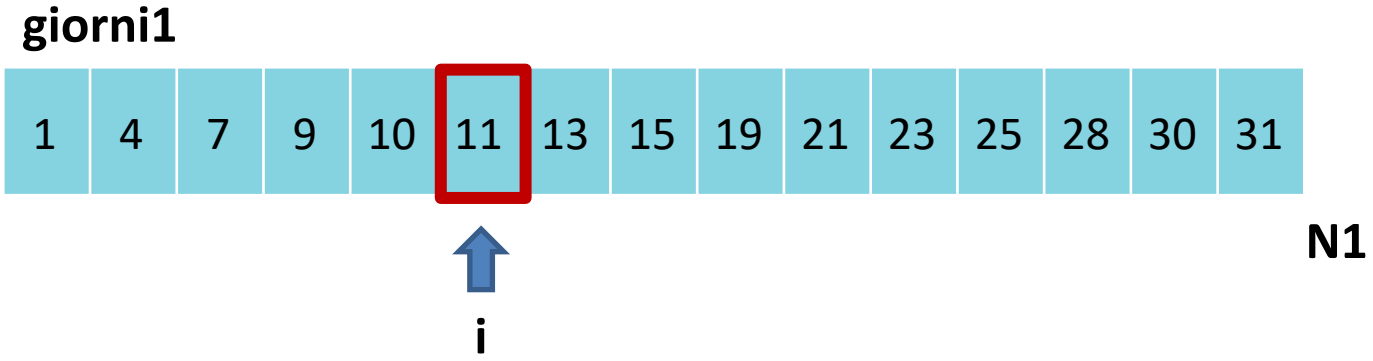
Esercizio – sottosequenza comune





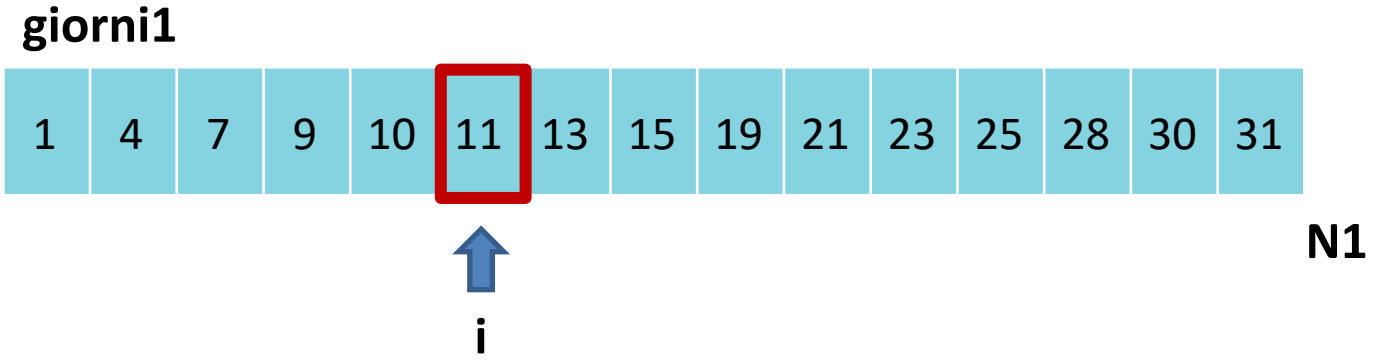
giorni1[i] <= giorni2[j] ➡ ++i

Esercizio – sottosequenza comune



giorni1[i] > giorni2[j] ➡ ++j

Esercizio – sottosequenza comune



giorni1[i] == giorni2[j] ➡ Giorno comune

Esercizio – sottosequenza comune

```
/* RICERCA DEGLI ELEMENTI COMUNI */
fallito = 1 ;
/* Per ogni giorno del collega 1... */
for( i=0 ; i<N1; i++ )
{
      trovato = 0 ;
      for( j=0; j<N2; j++ ){
         if( giorni2[j] == giorni1[i] )
            trovato = 1 ;
      }
      /* ...in caso affermativo stampalo */
      if( trovato == 1 ){
         printf("Giorno disponibile: %d\n", giorni1[i]) ;
         fallito = 0 ;
      }
}
/* Se non ne ho trovato nessuno, stampo un
messaggio */
if(fallito==1)
   printf("NESSUN giorno disponibile\n");
```

```
/* RICERCA DEGLI ELEMENTI COMUNI */
fallito = 1 ;

/* Per ogni giorno del collega 1... */
i=0;
j=0;

while(i<N1 && j<N2){
   if(giorni2[j] == giorni1[i]){
      printf("Giorno disponibile: %d\n", giorni1[i]);
      ++i;
      ++j;
      fallito = 0;
   }

   if(giorni1[i]<giorni2[j] && i<N1)
      ++i;

   if(giorni1[i]>giorni2[j]  && j<N2)
      ++j;
}

/* Se non ne ho trovato nessuno, stampo un messaggio */
if(fallito==1)
   printf("NESSUN giorno disponibile\n");
```

Esercizio – rilevazione dei duplicati

La società organizzatrice di un concerto vuole verificare che non ci siano biglietti falsi

A tale scopo, realizza un programma in linguaggio C che acquisisce i numeri di serie dei biglietti e verifica che non vi siano numeri duplicati

Esercizio – rilevazione dei duplicati

Il programma acquisisce innanzitutto il numero di biglietti venduti, N, ed in seguito acquisisce i numeri di serie degli N biglietti

Al termine dell'acquisizione, il programma stampa "Tutto regolare" se non ci sono riscontrati duplicati, altrimenti stampa il numero di serie dei biglietti duplicati

Caso I – tutto regolare

Caso I – biglietti duplicati

Esercizio – rilevazione dei duplicati

Azioni:

1) Acquisizione del valore di N

2) Lettura dei numeri di serie

3) Utilizzo di un vettore di interi **serie**[**MAXN**]

4) Ricerca dei duplicati

5) Stampa dei messaggi finali

Esercizio – rilevazione dei duplicati

Prendi un elemento per volta

elem = **serie**[i] ;

Cerca se altri elementi del vettore sono uguali a tale elemento

uguali ⇒ (**elem** == **serie**[j])

altri ⇒ (i != j)

Si tratta di una ricerca di esistenza per ogni elemento considerato

Esercizio – rilevazione dei duplicati

```
#include <stdio.h>
#define MAXN  100

int main(){
   int N ; /* num tot biglietti */
   int serie[MAXN] ; /* numeri serie */
   int elem ;
   int i, j ;

   int dupl ; /* flag: trovato almeno un duplic.? */
   int trovato; /* flag per ricerca di esistenza */
```

Esercizio – rilevazione dei duplicati

```
printf("RICERCA DUPLICATI\n") ;
printf("\n");

/* ACQUISIZIONE VALORE DI N */
do{
   printf("Num tot di biglietti: ") ;
   scanf("%d", &N) ;

   if (N<2 || N>MAXN)
      printf("N=%d non valido\n", N) ;

} while(N<2 || N>MAXN) ;
```

Qual è lo scopo di questo ciclo?

Ripete la richiesta del valore N finché non viene inserito un valore valido, ovvero N>2 e N<MAXN

Esercizio – rilevazione dei duplicati

```
/* LETTURA DEI NUMERI DI SERIE */
for( i=0 ; i<N ; i++ ){
   printf("Numero serie biglietto %d: ", i+1) ;
   scanf("%d", &serie[i]) ;
}

/* RICERCA DEI DUPLICATI */
dupl = 0 ;
for( i=0 ; i<N ; i++ ){
   /* verifica se serie[i] e' duplicato */
   elem = serie[i] ;
   trovato = 0 ;
   for( j=0 ; j<N ; j++ ){
      if( (i!=j) && (elem == serie[j]) )
         trovato = 1 ;
   }
   if(trovato == 1){
      printf("ATTENZIONE: %d duplicato\n", elem) ;
      dupl = 1 ;
   }
}
```

Esercizio – rilevazione dei duplicati

```
   /* STAMPA DEI MESSAGGI FINALI */
   if(dupl==0)
      printf("Tutto regolare\n");

   return 0;
}
```

Esecuzione

```
RICERCA DUPLICATI

Num tot di biglietti: 5
Numero serie biglietto 1: 1243
Numero serie biglietto 2: 5464
Numero serie biglietto 3: 2365
Numero serie biglietto 4: 5464
Numero serie biglietto 5: 3476
ATTENZIONE: 5464 duplicato
ATTENZIONE: 5464 duplicato
```

Perché stampa il biglietto due volte?

Quanti passi impiega il programma?

Ciclo esterno **N** passi
  Ciclo interno **N** passi

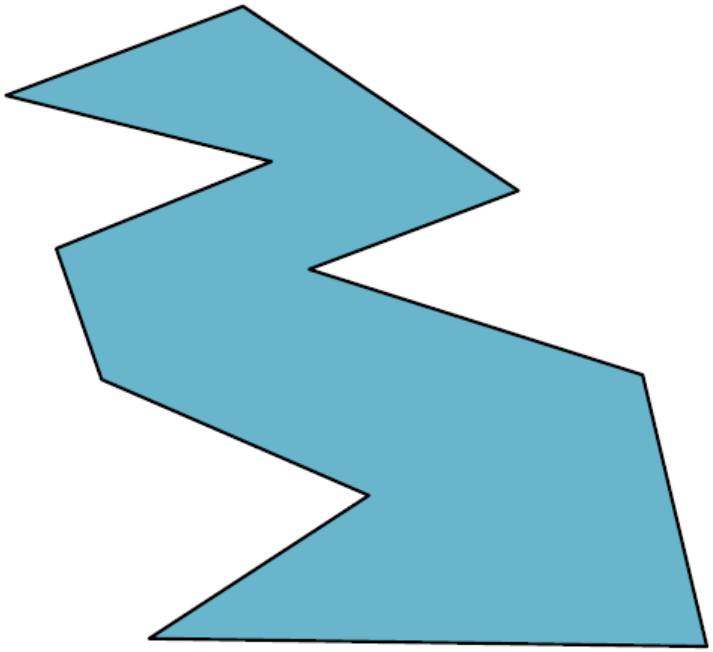
Numero di passi ≈ **N** × **N** = $\mathbf{N^2}$

Si può fare di meglio?

Sì. Se consideriamo i biglietti duplicati come le disponibilità dei giorni dell'esercizio precedente, si può risolvere con circa **N** passi

Esercizio – Perimetro di un poligono

Uno studente di geometria deve misurare il perimetro di una serie di poligoni irregolari, di cui conosce le coordinate cartesiane (x,y) dei vertici. Per far ciò realizza un programma in C



Esercizio – Perimetro di un poligono

Azioni:

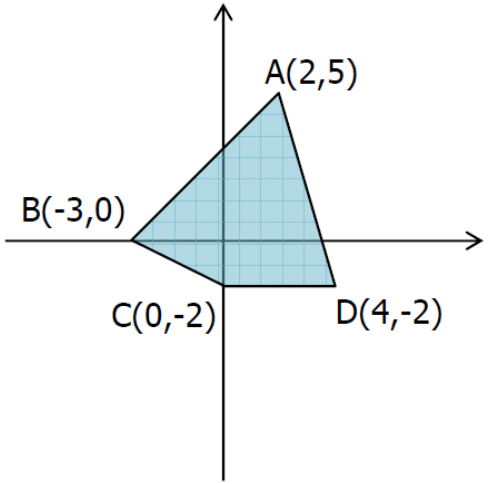
1) Il programma acquisisce il numero N di vertici di cui e composto il poligono

2) Il programma acquisisce le N coppie (x,y) corrispondenti agli N vertici

3) Il programma calcola la lunghezza di ciascun lato e somma i valori per calcolare il perimetro

4) Il programma stampa la lunghezza complessiva del perimetro del poligono irregolare

Esercizio – Perimetro di un poligono



$AB = \sqrt{(2-(-3))^2+(5-0)^2}$

$BC = \sqrt{(-3-0)^2+(0-(-2))^2}$

$CD = \sqrt{(0-4)^2+(-2-(-2))^2}$

$DA = \sqrt{(4-2)^2+(-2-5)^2}$

Perimetro =

- = AB + BC + CD + DA

```
Prompt dei comandi
CALCOLO DEL PERIMETRO

Numero di vertici: 4
Inserire le coordinate dei vertici
Vertice 1:  x = 2
            y = 5
Vertice 2:  x = -3
            y = 0
Vertice 3:  x = 0
            y = -2
Vertice 4:  x = 4
            y = -2

Lunghezza del perimetro: 21.956729
```

Esercizio – Perimetro di un poligono

In questo problema i dati da memorizzare non sono semplici numeri, ma **coppie di numeri**

Possiamo utilizzare due vettori

- **float** x[], contenente le ascisse dei punti
- **float** y[], contenente le ordinate dei punti

Esempio:

- Punto **A**(2,5) ⇒ x[0]=2 ; y[0]=5 ;

In tutte le elaborazioni, i due vettori verranno usati con uguale valore dell'indice

- Vettori "**paralleli**"

Esercizio – Perimetro di un poligono

```
#include <stdio.h>
#define MAXN  10

int main(){

   int N ; /* numero di vertici */
   float x[MAXN] ; /* vettori "paralleli" */
   float y[MAXN] ;
   int i ;
   float lato ;
   float perimetro ;

 /* ACQUISIZIONE VALORE DI N */
   do{
      printf("Numero di lati: ") ;
      scanf("%d", &N) ;

      if (N<3 || N>MAXN)
         printf("N=%d non valido\n", N) ;

   } while(N<3 || N>MAXN) ;
```

Esercizio – Perimetro di un poligono

```
/* ACQUISIZIONE COORDINATE VERTICI */
printf("Inserire coordinate\n");
for( i=0; i<N; i++ ){
   printf("Vertice %d: x = ", i+1) ;
   scanf("%f", &x[i]) ;
   printf(" y = ") ;
   scanf("%f", &y[i]) ;
}

perimetro = 0 ;
for( i=0; i<N-1; i++ ){
   /* (x[i],y[i])-(x[i+1],y[i+1]) */
   lato = sqrt( (x[i]-x[i+1])*(x[i]-x[i+1]) + (y[i]-y[i+1])*(y[i]-y[i+1]) ) ;
   perimetro = perimetro + lato ;
}

/* ultimo lato: (x[N-1],y[N-1])-(x[0],y[0]) */
lato = sqrt( (x[N-1]-x[0])*(x[N-1]-x[0]) + (y[N-1]-y[0])*(y[N-1]-y[0]) ) ;

perimetro = perimetro + lato ;
```

Esercizio – Perimetro di un poligono

```
  /* STAMPA DEL PERIMETRO */
  printf("La lunghezza del perimetro è: %f\n", perimetro);

  return 0;
}
```

Esecuzione

```
Numero di lati: 4
Inserire coordinate
Vertice 1: x = 2
 y = 5
Vertice 2: x = -3
 y = 0
Vertice 3: x = 0
 y = -2
Vertice 4: x = 4
 y = -2
La lunghezza del perimetro è: 21.956728
```

Esercizi

1. Vettori ad occupazione variabile

La principale limitazione dei vettori è la loro dimensione fissa, definita come costante al tempo di compilazione del programma

Molto spesso non si conosce l'effettivo numero di elementi necessari fino a quando il programma non andrà in esecuzione

Occorre identificare delle tecniche che ci permettano di lavorare con vettori di dimensione fissa, ma occupazione variabile

Dichiarare un vettore di dimensione sufficientemente ampia da contenere il massimo numero di elementi nel caso peggiore

Esempio: MAXN

La parte iniziale del vettore sarà occupata dagli elementi, la parte finale rimarrà inutilizzata

Dichiarare una variabile che tenga traccia dell'effettiva occupazione del vettore

Esempio: N