Programmazione

Giovanni Da San Martino

Dipartimento of Matematica, Università degli Studi di Padova
giovanni.dasanmartino@unipd.it
A.A. 2021-2022

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Compitini

- Solo chi passa il primo compitino può fare il secondo
- Chi passa entrambi i compitini ha superato l'esame, altrimenti si riparte da zero agli appelli (non si ridanno solo parti di un esame all'appello)
- L'esame sarà in aula (LUM250), probabilmente faremo due turni
  - nel caso metterò le istruzioni su Moodle prima possibile (ci sarà una lista a cui iscriversi)
  - portate penna; lapis e gomma per la brutta.
  - siate collaborativi (non presentatevi al turno sbagliato, ecc..)
  - Se vengono fatti due turni, potete uscire solamente al termine del turno
- Serve il green pass base per accedere alle strutture e mascherina chirurgica

Esercizi

https://www.wooclap.com/TJSSZS

Esercizi

Dato un array di 6 interi ed un intero x, stampare l'indice nell'array dell'ultima occorrenza di x, oppure "non trovato" se non c'è.

int x[6] = {2,6,1,5,1,5};

Esercizi

Definire una funzione ricorsiva che determini se esiste un percorso che permetta di attraversare un campo fiorito, dal basso verso l'alto, senza calpestare alcun fiore.

Il campo è rappresentato da una matrice, i cui valori rappresentano la presenza di un fiore (0) oppure la sua assenza (1). La posizione iniziale è fornita come parametro della funzione.

E' possibile muoversi una casella in alto oppure una casella verso destra.

{0,0,0,1,0},

{0,1,0,1,0},

{1,0,0,1,0},

{1,0,1,1,1},

{1,0,1,0,0}

Esercizio


```
int mossa(int dim_x, int dim_y, int campo[dim_x][dim_y], int pos_x, int pos_y) {

   if ( (pos_x<0) || (pos_y<0) || (pos_y>=dim_y) || (pos_x>=dim_x) ) {
         return 0; //mossa fuori dal percorso
   }
   if (campo[pos_x][pos_y]==0)
      return 0; // calpestato un fiore, percorso non valido
   if (pos_x==0)
      return 1; // sono arrivato nella prima riga, percorso valido
   return (                                                  //muovo di una casella
      (mossa(dim_x, dim_y, campo, pos_x-1, pos_y)) ||     // in alto
      (mossa(dim_x, dim_y, campo, pos_x, pos_y+1))       // a destra
   );
}
```

Esercizi

- L'esercizio su campo fiorito, sarebbe stato possibile risolverlo con una funzione ricorsiva se le mosse possibili fossero state 3: muovi di una casella a sinistra, muovi di una casella in alto, muovi di una casella a destra? Perché?

{0,0,0,1,0},

{0,1,0,1,0},

{1,0,0,1,0},

{1,0,1,1,1},

{1,0,1,0,0}

Esercizio


```
int mossa(int dim_x, int dim_y, int campo[dim_x][dim_y], int pos_x, int pos_y) {

   if ( (pos_x<0) || (pos_y<0) || (pos_y>=dim_y) || (pos_x>=dim_x) ) {
         return 0; //mossa fuori dal percorso
   }
   if (campo[pos_x][pos_y]==0)
      return 0; // calpestato un fiore, percorso non valido
   if (pos_x==0)
      return 1; // sono arrivato nella prima riga, percorso valido
   return (                                                  //muovo di una casella
      (mossa(dim_x, dim_y, campo, pos_x-1, pos_y)) ||     // in alto
      (mossa(dim_x, dim_y, campo, pos_x, pos_y-1)) ||     // a sinistra
      (mossa(dim_x, dim_y, campo, pos_x, pos_y+1))        // a destra
   );
}
```

Percorso:

{0,0,0,1,0},

{0,1,0,1,0},

{1,0,0,1,0},

{1,0,1,1,1},

{1,0,1,0,0}

mossa(3,3) ha successo, perché?

Argomenti da linea di Comando

- E’ possibile passare argomenti ad un programma C direttamente da linea di comando al momento dell’esecuzione (invece di leggerli da tastiera)

gcc –o palindroma palindroma.c

palindroma abba

stampa “la stringa abba è palindroma”

- i parametri sono separati da spazi: “palindromo abba 1221“ indica due parametri stringa
- gli argomenti da linea di comando sono i parametri della funzione main:

int main (int argc, char *argv[])

- i nomi delle variabili sono arbitrari, ma si usa argc e argv per tradizione

Argomenti da linea di Comando

int main (int argc, char *argv[])

- una volta definiti i parametri della funzione main, ovvero

  int main (void) -> int main (int argc, char *argv[])

- si avrà sempre un almeno argomento passato da linea di comando

argc>=1

argv[0] è il nome del programma

- argc: numero di parametri passati dalla linea di comando (incluso il nome del programma)
- argv[i]: l'i-esimo argomento

Esercizio

- Scrivere un frammento di codice che stampi la lista dei parametri passati da linea di comando, escluso il nome del programma

```
int main (int argc, char *argv[]) {
        ...
}
```

Esercizio

- Completare la seguente funzione ricorsiva e mostrarne la correttezza

// date n persone in una stanza che stanno salutandosi, calcolare il numero totale di strette di mano

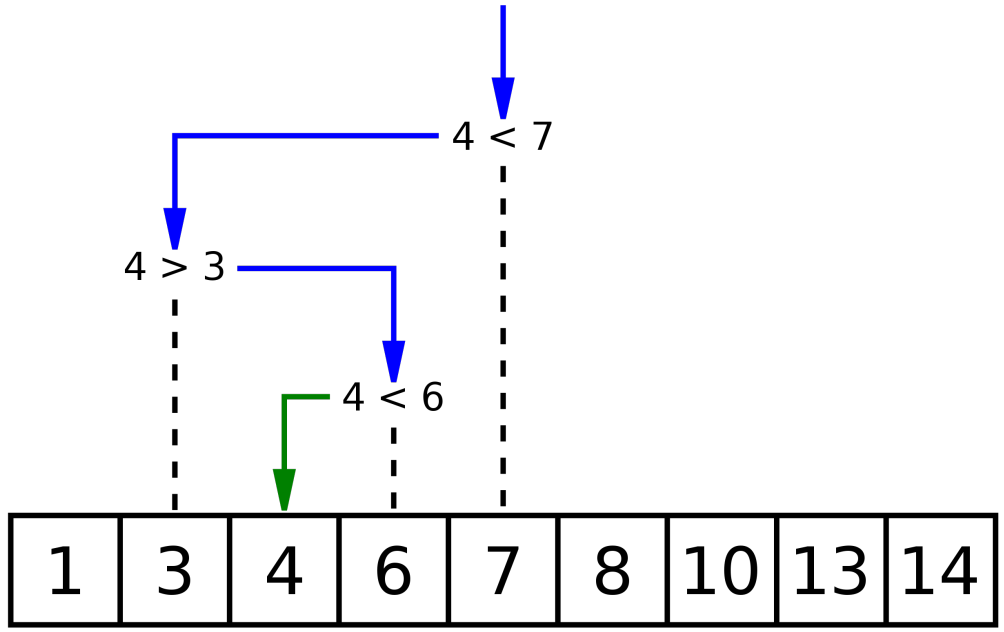
```
unsigned int handshake(unsigned int n) {
        //POST Restituisce il numero totale di strette di mano tra n persone
}
```

Divide and Conquer

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Ricerca Binaria

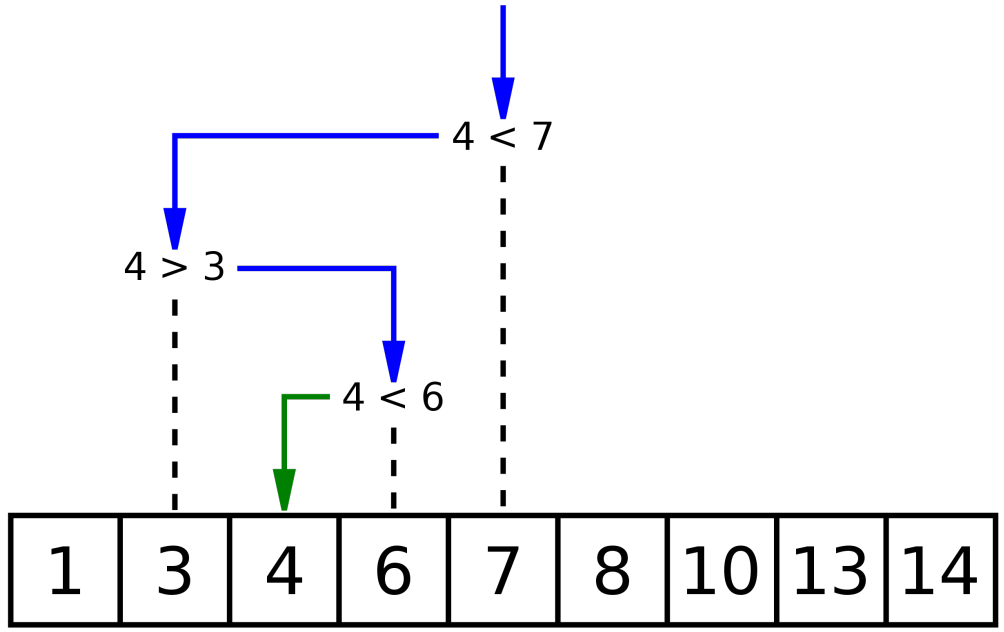

```
int ricerca_binaria(int *X, int dim, int elem) {
/*
    PRE: X ordinato in modo crescente
    POST: restituisce 1 se elem è in X, 0 altrimenti
*/
/*
confronta elem con l'elemento centrale dell'array; se è minore elem può solo essere nella metà di sinistra (X è ordinato), altrimenti può solo essere a destra. Ripeti l'algoritmo con la metà dell'array prescelta.
*/
}
```



Ricerca Binaria


```
int ricerca_binaria(int *X, int dim, int elem) {
   /* PRE: X ordinato in modo crescente
        POST: restituisce 1 se elem è in X, 0 altrimenti */
   int n = dim/2;
   if (dim<=0)
      return 0;
   if (elem==X[n])
      return 1;

   if (elem<X[n])
      return ricerca_binaria(X, n, elem);
   else
      return ricerca_binaria(X+n+1, dim-n-1, elem);
}
```



Ricerca Binaria


```
int ricerca_binaria(int *X, int dim, int elem) {
   /* PRE: X ordinato in modo crescente
        POST: restituisce 1 se elem è in X, 0 altrimenti */
   int n = dim/2;
   if (dim<=0)    // *1
      return 0;
   if (elem==X[n])
      return 1;   // *2

   if (elem<X[n])
      return ricerca_binaria(X, n, elem);  //*3
   else
      return ricerca_binaria(X+n+1, dim-n-1, elem); //*4
}
```

*1 l'array vuoto non contiene elem; *2 elem trovato

*3 ricerca_binaria()==1 => elem trovato; se ricerca_binaria()==0 elem non è in X[0],...,X[n-1], ma elem<X[n]<=X[n+1]<=...<=X[dim-1], quindi possiamo concludere che elem non sia in X e restituire 0 (*4 stesso ragionamento di *3)

Notate che stiamo usando l'ipotesi P(n/2)=>P(n) (possiamo usare qualsiasi n'<n)

Algoritmi Divide and Conquer

- Divide and Conquer consiste in

1. Dividere il problema in sottoproblemi più piccoli.
2. Risolvere ricorsivamente i sottoproblemi
3. Combinare le soluzioni dei sottoproblemi per ottenere la soluzione del problema originale

- Notate che la ricorsione è parte integrante della strategia di risoluzione del problema

Mergesort

- Mergesort è un algoritmo che utilizza la strategia divide and conquer per ordinare un array di valori
- Dato un array X di dimensione n,

1. richiama mergesort sulla prima metà dell'array ottenendo X_1
2. richiama mergesort sulla seconda metà dell'array ottenendo X_2
3. combina gli array ordinati X_1, X_2

```
MERGE-SORT(A, p, r) {
        if (p<r) {
                q = (p + r)/2;
                MERGE-SORT(A, p, q);
                MERGE-SORT(A, q + 1, r);
                combina_array(A, p, q, r);
        }
}
```