Titolo unità didattica: Algoritmi ricorsivi [14]

Titolo modulo : Ricorsività in C [04-C]

Sviluppo di function ricorsive in C per vari algoritmi ricorsivi

Argomenti trattati:

- ✓ function ricorsive in C per la ricerca binaria
- ✓ function ricorsiva in C per la ricerca binaria in un array di stringhe
- ✓ function ricorsiva in C per la somma degli elementi di un array
- ✓ function ricorsiva in C per il massimo degli elementi di un array
- ✓ function ricorsiva elementare in C per il calcolo dell'n-simo numero di Fibonacci
- ✓ function ricorsiva avanzata in C per il calcolo dell'n-simo numero di Fibonacci (approccio di programmazione dinamica)

Prerequisiti richiesti: AP-05-03-C, AP-09-03-C, AP-14-02-T, AP-14-03-T

```
int ric_bin_ric(char chiave,char elenco[],
                int primo, int ultimo)
{
 int mediano;
 if(primo > ultimo)
    return -1;
 mediano = (primo + ultimo)/2;
 if(chiave == elenco[mediano])
    return mediano;
   else if(chiave < elenco[mediano])
          return ric_bin_ric(chiave,elenco,
                 primo,mediano-1);
        else
          return ric_bin_ric(chiave,elenco,
                 mediano+1,ultimo);
}
```

```c
#include <stdio.h>
#include <string.h>
int ric_bin_ric(char,char [],int,int);
void main()
{
 char elenco[100],cdr;
 int indice;
 printf("inserire l'elenco di caratteri\n");
 gets(elenco);
 printf("inserire il carattere da ricercare: ");
 scanf("%c",&cdr);
 indice = ric_bin_ric(cdr,elenco,0,strlen(elenco)-1);
 if(indice >= 0)
   printf("chiave trovata al posto %d\n",indice);
 else
   printf("chiave non trovata\n");
}
```

```c
#include <stdio.h>
#include <string.h>
typedef enum {False,True} Logical;
Logical ric_bin_ricTF(char,char [],int) ;
void main(){
 char elenco[100],cdr;
 Logical k;
 printf("inserire l'elenco di caratteri\n");
 gets(elenco);
 printf("inserire il carattere da ricercare: ");
 scanf("%c",&cdr);
 k = ric_bin_ricTF(cdr,elenco,strlen(elenco)) ;
 if(k == True)
   printf("chiave trovata\n");
 else
   printf("chiave non trovata\n");
}
```

```
Logical ric_bin_ricTF(char chiave,char elenco[],int n)
{
 int mediano;
 if(n == 0)
    return False;
 mediano = (n-1)/2;
 if(chiave == elenco[mediano])
   return True;
  else if(chiave < elenco[mediano])
        return ric_bin_ricTF(chiave,elenco,mediano);
      else
        return ric_bin_ricTF(chiave,elenco+mediano+1,
                                    n-mediano-1);
}
```

```
Logical ric_bin_SricTF(char chiave[],char *elenco[],int n)
{
 int mediano;
 if(n == 0)
    return False;
 mediano = (n-1)/2;
 if(strcmp(chiave,elenco[mediano]) == 0)
    return True;
   else
    if(strcmp(chiave,elenco[mediano]) < 0)
       return ric_bin_SricTF(chiave,elenco,mediano);
      else
       return ric_bin_SricTF(chiave,elenco+mediano+1,
                                          n-mediano-1);
}
```

Ricerca binaria su un array ordinato di puntatori a stringhe

```
int ric_bin_ric_ind (char chiave,char elenco[],int n){
 int mediano, ind_loc;
 if(n == 0)
    return -1;
 mediano = (n-1)/2;
 if(chiave == elenco[mediano])
   return mediano;
  else if(chiave < elenco[mediano])
        return ric_bin_ric_ind(chiave,elenco,mediano);
      else {
        ind_loc = ric_bin_ric_ind(chiave,elenco+mediano+1,
                                          n-mediano-1);
         if (ind_loc  == -1)
           return -1;
          else
          return ind_loc + mediano + 1;}
}
```

dato di output: il valore dell'indice (se la chiave appartiene all'array);
-1 (se la chiave non appartiene all'array)

lo spiazzamento dell'indice deve essere fatta **solo** nell'autoattivazione sulla porzione di destra dell'array

tecnica di programmazione ricorsiva per l'algoritmo incrementale di **somma** dei primi **n** numeri naturali

```
int somma_ric(int n)
{
   if (n == 1)
      /* soluzione del caso base */
      return 1;
    else
      /* autoattivazione */
      return n+somma_ric(n-1);
   /* endif */
}
```

tecnica di programmazione ricorsiva per l'algoritmo incrementale del **fattoriale** di **n**

```
int fatt_ric(int n)
{
  if (n <= 1)
     /* soluzione del caso base */
     return 1;
   else
     /* autoattivazione */
     return n*fatt_ric(n-1);
  /* endif */
}
```

tecnica di programmazione ricorsiva per l'algoritmo incrementale della **somma** degli elementi di un array

```
int somma_a_ricAI(int a[],int n)
{
 if (n == 1)
      /* soluzione del caso base */
       return a[0];
    else
      /* autoattivazione */
       return a[n-1] + somma_a_ricAI(a,n-1);
 /* endif */
}
```

tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **somma** degli elementi di un array

```
int somma_a_ricDI(int a[],int primo,int ultimo)
{
 int mediano;
/* soluzione del caso base */
  if (primo == ultimo)
       return a[primo];
   else
    /* autoattivazioni */
      {
       mediano = (primo+ultimo)/2;
       return somma_a_ricDI(a,primo,mediano) + ...
              somma_a_ricDI(a,mediano+1,ultimo);
       }
}
```

versione 1

tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **somma** degli elementi di un array

```
int somma_a_ricDI(int a[], int n)
{
 int mediano;
/* soluzione del caso base */
 if (n == 1)
       return a[0];
   else
    /* autoattivazioni */
      {
       mediano = (n-1)/2;
       return somma_a_ricDI(a,mediano+1) + ...
            somma_a_ricDI(a+mediano+1,n-mediano-1);
       }
}
```

versione 2

tecnica di programmazione ricorsiva per l'algoritmo incrementale di **massimo** degli elementi di un array

```
int massimo_a_ricAI(int a[],int n)
{
 if (n == 1)
/* soluzione del problema banale */
   return a[0];
  else
      /* autoattivazione */
   return max_I(a[n-1],massimo_a_ricAI(a,n-1));
}
```

tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **massimo** degli elementi di un array

```
int massimo_a_ricDI(int a[],int primo,int ultimo)
{
 int mediano;
  /* soluzione del caso base  */
 if(primo == ultimo)
    return a[primo];
   else
    {
    /* autoattivazioni */
    mediano = (primo+ultimo)/2;
    return max_I(massimo_a_ricDI(a,primo,mediano),
                 massimo_a_ricDI(a,mediano+1,ultimo));
    }
}
```

versione 1

tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **massimo** degli elementi di un array

```
int massimo_a_ricDI(int a[],int n)
{
 int mediano;
  /* soluzione del caso base  */
 if(n == 1)
    return a[0];
  else
    {
    /* autoattivazioni */
    mediano = (n-1)/2;
    return max_I(massimo_a_ricDI(a,mediano+1), …
        massimo_a_ricDI(a+mediano+1,n-mediano-1));
    }
}
```

versione 2

tecnica di programmazione ricorsiva per l'algoritmo di **fibonacci** (versione elementare)

```
int fib_ric_el(int n)
{
 if (n == 1 || n == 0)
 /* soluzione del caso base */
    return n;
  else
 /* autoattivazioni */
     return fib_ric_el(n-1) + fib_ric_el(n-2);
}
```

$T(n) = O(1.6^n)$

somme

tecnica di programmazione ricorsiva per l'algoritmo di **fibonacci** , versione avanzata che effettua solo n-1 addizioni (come nel caso iterativo)

```
#include <stdio.h>
double fib_ric_PD(int n)
double fibogia[100];
void main()
{
 int n,i;
 printf(" inserire il valore di n \n");
 scanf("%d",&n);
 for(i=0;i<100;i++)
      fibogia[i]=0.0;
 printf("%d-simo num Fib:%e\n",n,fib_ric_PD(n));
}
/* CONTINUA */
```

dichiarazione di un array globale

```
double fib_ric_PD(int n)
{
 if(n <= 1)
    return (double)n;
 if(fibogia[n] != 0)
    return fibogia[n];
   else
   {
    fibogia[n]=fib_ric_PD(n-1)+fib_ric_PD(n-2);
    return fibogia[n];
   }
}
```

$T(n) = n-1$
somme

$S(n) = n$

approccio di **Programmazione Dinamica**