Unità didattica: Esempi di algoritmi ricorsivi [3-AT]

Titolo: Altri esempi di programmi ricorsivi

Argomenti trattati:

- ✓ Algoritmo ricorsivo per la formula di Horner
- ✓ Algoritmo ricorsivo per la costruzione di una lista
- ✓ Algoritmi ricorsivi per la visita di un albero binario
- ✓ Algoritmo ricorsivo di ricerca binaria su un albero binario di ricerca
- ✓ Cenni al metodo di bisezione per risolvere un'equazione non lineare

Prerequisiti richiesti: ricorsione, algoritmi per la valutazione di polinomi, strutture dati dinamiche

algoritmo di Horner

1 Laboratorio:

scrivere 2 function C (risp. iterativa e ricorsiva) per valutare un polinomio mediante algoritmo di Horner.

Assegnato un polinomio $P_n(x)$, di grado n, mediante i suoi coefficienti nell'array $a[0..n]$

$$P_n(x) = a_n + a_{n-1}x + a_{n-2}x^2 + a_{n-3}x^3 + \ldots + a_1x^{n-1} + a_0x^n$$

l'algoritmo di Horner prevede che il polinomio si esprima come

$$P_n(x) = a_n + x(a_{n-1} + x(a_{n-2} + x(a_{n-3} + x(\ldots + x(a_1 + xa_0))\ldots)))$$

La versione iterativa dell'algoritmo calcola il valore del polinomio valutando la formula dalle parentesi più interne a quelle più esterne; mentre quella ricorsiva opera dalle parentesi più esterne a quelle più interne (ogni livello di parentesi è una chiamata alla funzione).

algoritmo di Horner: versione iterativa

La valutazione delle parentesi procede da quella più interna a quella più esterna

$$P_n(x) = a_n + x(a_{n-1} + x(a_{n-2} + x(a_{n-3} + x(\ldots + x(a_1 + xa_0))\ldots)))$$

```
function H(x,n)
P := a0
for k:=1 to n
    P := x*P+ ak
end
return P
```

$P_n(x) = a_0$

$P_n(x) = (a_1 + xa_0)$

$P_n(x) = a_2 + x(a_1 + xa_0)$

$P_n(x) = a_3 + x(a_2 + x(a_1 + xa_0))$

$P_n(x) = a_4 + x(a_3 + x(a_2 + x(a_1 + xa_0)))$

$\vdots$

$$P_n(x) = a_n + x(a_{n-1} + x(a_{n-2} + x(a_{n-3} + x(\ldots + x(a_1 + xa_0))\ldots)))$$

algoritmo di Horner: versione ricorsiva

La valutazione delle parentesi procede da quella più esterna a quella più interna

$$P_n(x) = a_n + x(a_{n-1} + x(a_{n-2} + x(a_{n-3} + x(\ldots + x(a_1 + xa_0))\ldots)))$$

```
function H(x,n)
if n=0
   P := a0
else
   P := an + x*H(x,n-1)
end
return P
```

STACK: svuotamento

n	H(x,n)
N	$H(x,N) = a_n + x*(a_{n-1} + x*(a_{n-2} + x*(a_{n-3} + x*(\ldots + xa_0))))$
N-1	⋮
N-2	$H(x,3) = a_3 + x*(a_2 + x*(a_1 + x*a_0))$
⋮	$H(x,2) = a_2 + x*(a_1 + x*a_0)$
1	$H(x,1) = a_1 + x*a_0$
0	$H(x,0) = a_0$

Altri esempi di programmi ricorsivi in C:
costruzione di una lista

A partire da un array contenente le informazioni da inserire nei nodi, si costruisce una lista lineare.
In particolare, il seguente programma crea ricorsivamente una lista dove ogni nodo contiene un carattere.

```
#include <stdio.h>
#include <stdlib.h>
typedef char DATA;
struct linked_list
     { DATA d;
       struct linked_list *next;
     };
typedef struct linked_list ELEMENT;

typedef ELEMENT *LINK;

LINK array_to_list(DATA []);  ...
```

definisce il tipo dei nodi della lista

prototipo

```
...   void main()
      { DATA c_arr[]="questa e` una prova!";
        LINK head_list,p_list;
        head_list = array_to_list(c_arr);
        p_list=head_list;
        while (p_list != NULL)
           { putchar(p_list->d);
             p_list=p_list->next;
           }
        puts("");
      }

      LINK array_to_list(DATA s[])
      { LINK head;
        if (s[0] == '\0') return NULL;
        else
           { head = malloc(sizeof(ELEMENT));
             head->d = s[0];
             head->next = array_to_list(s+1);
             return head;
           }
      }
```

Altri esempi di programmi ricorsivi in C:
visita di un albero binario

organizzazione dei dati

```
#include    ...

typedef char DATA;
struct b_tree
     { DATA d;
       struct b_tree *figlio_sx;
       struct b_tree *figlio_dx;
     };
typedef struct b_tree ELEMENT;
typedef ELEMENT *LINK;

        ...
```

attraversamento preorder (ordine anticipato)

visita nell'ordine:

1. la radice,
2. il sottoalbero sinistro,
3. il sottoalbero destro.

```
...

      void preorder_visit(LINK ptr)
      {
        if (prt!=NULL)
           { visita_nodo(ptr); /* visita la radice */
             preorder_visit(ptr->figlio_sx);
             preorder_visit(ptr->figlio_dx);
           }
      }
```

attraversamento inorder (ordine simmetrico)

visita nell'ordine:

1. il sottoalbero sinistro,
2. la radice,
3. il sottoalbero destro.

```
...

      void inorder_visit(LINK ptr)
      {
        if (ptr!=NULL)
           { inorder_visit(ptr->figlio_sx);
             visita_nodo(ptr); /* visita la radice */
             inorder_visit(ptr->figlio_dx);
           }
      }
```

attraversamento postorder (ordine differito)

visita nell'ordine:

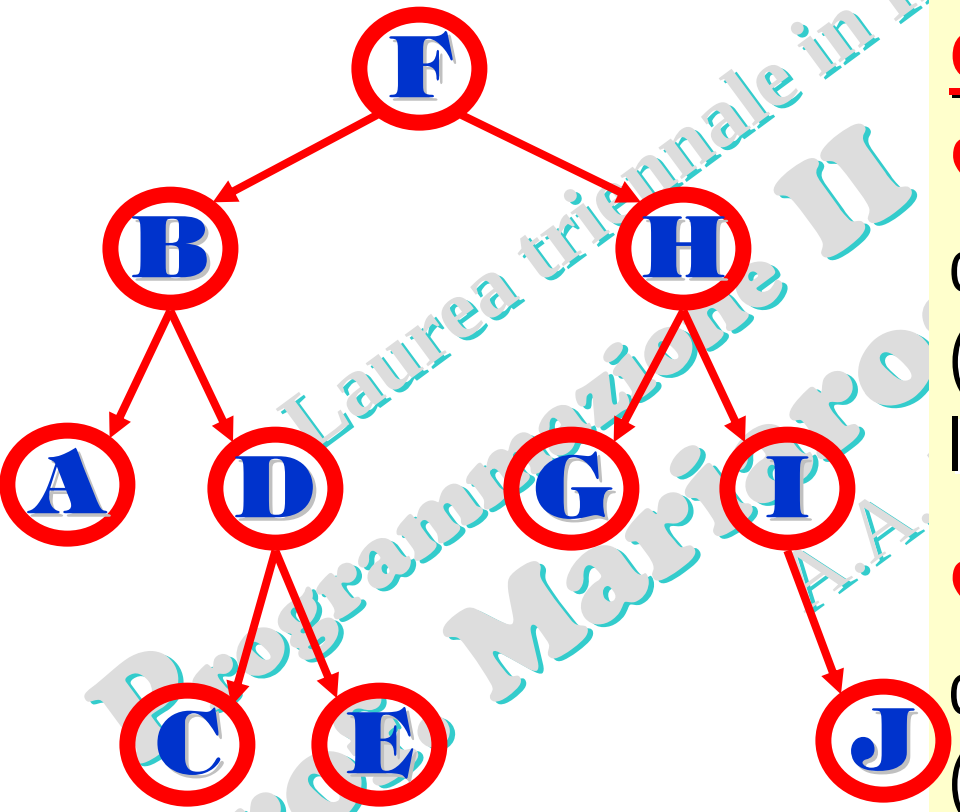
1. il sottoalbero sinistro,
2. il sottoalbero destro,
3. la radice.

```
...

      void postorder_visit(LINK ptr)
      {
        if (prt!=NULL)
           { postorder_visit(ptr->figlio_sx);
             postorder_visit(ptr->figlio_dx);
             visita_nodo(ptr); /* visita la radice */
           }
      }
```

Altri esempi di algoritmi ricorsivi in C: ricerca binaria in un albero binario ordinato

L'albero dev'essere costruito in modo tale che l'ordine naturale dei nodi si ritrovi mediante la visita inorder.

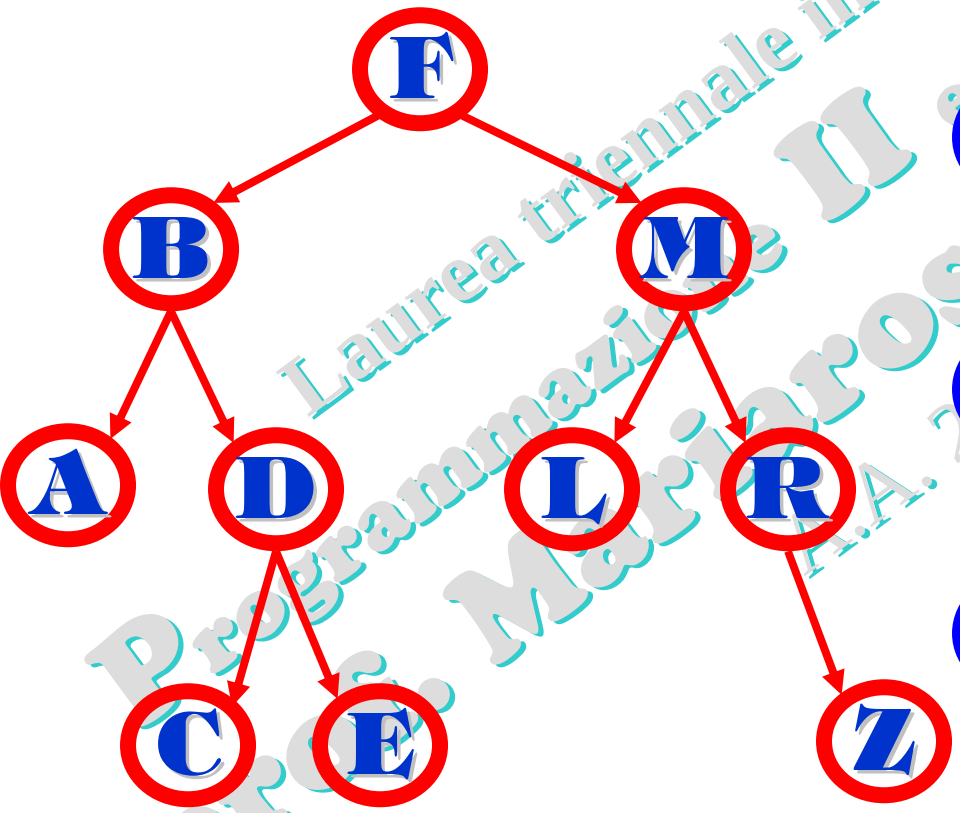


Criterio: per ciascun nodo

- il suo sottoalbero sinistro contiene dati che precedono (secondo l'ordinamento naturale) la radice del sottoalbero;
- il suo sottoalbero destro contiene dati che seguono (secondo l'ordinamento naturale) la radice del sottoalbero;

ricerca binaria su binary search tree

Idea algoritmo: se l'albero non è vuoto, confronta **elem** con il nodo **radice** scendendo per il sottoalbero sinistro se **elem<radice** oppure scendendo per il sottoalbero destro se **elem>radice**. Ripete il confronto.



Esempio: elem= E

E < F → sub alb. sx

E > B → sub alb. dx

E > D → sub alb. dx

E = E

```
typedef char DATA;
struct b_tree
     { DATA d;
       struct b_tree *figlio_sx;
       struct b_tree *figlio_dx;
     };
typedef struct b_tree ELEMENT;
typedef ELEMENT *LINK;
```

organizzazione dei dati

```
LINK btree_search(DATA elem, LINK radice)
/* restituisce NULL se non ha trovato elem,
 * altrimenti restituisce il puntatore al nodo con elem
 */
{ LINK ptr;
  if (radice==NULL) return NULL;
  else
     { if (elem == radice->d) return radice;
       else if (elem < radice->d)
                ptr=btree_search(elem,radice->figlio_sx);
       else     ptr=btree_search(elem,radice->figlio_dx);
     }
}
```

Esercizi: scrivere 2 *function C* (risp. iterativa e ricorsiva) per …

1 Valutare un polinomio mediante algoritmo di Horner [liv.3]

2 Visitare una lista lineare, stampando le informazioni [liv.3]

3 Visitare un albero binario (risp. in ordine anticipato, simmetrico e differito) stampando le informazioni [liv.2]

4 Approssimare lo zero di una funzione mediante algoritmo di bisezione ricorsivo. [liv.3]

help

Cenni al metodo di bisezione per risolvere un'equazione $f(x)=0$

Teor. Se la $f(x)$ è monotona in $[a, b]$ e cambia segno agli estremi dell'intervallo, cioè $f(a)f(b)<0$, allora il suo **unico zero** cade in $[a,b]$ ed in ogni suo sottointervallo in cui la f cambi segno agli estremi.

Si genera la seguente successione:

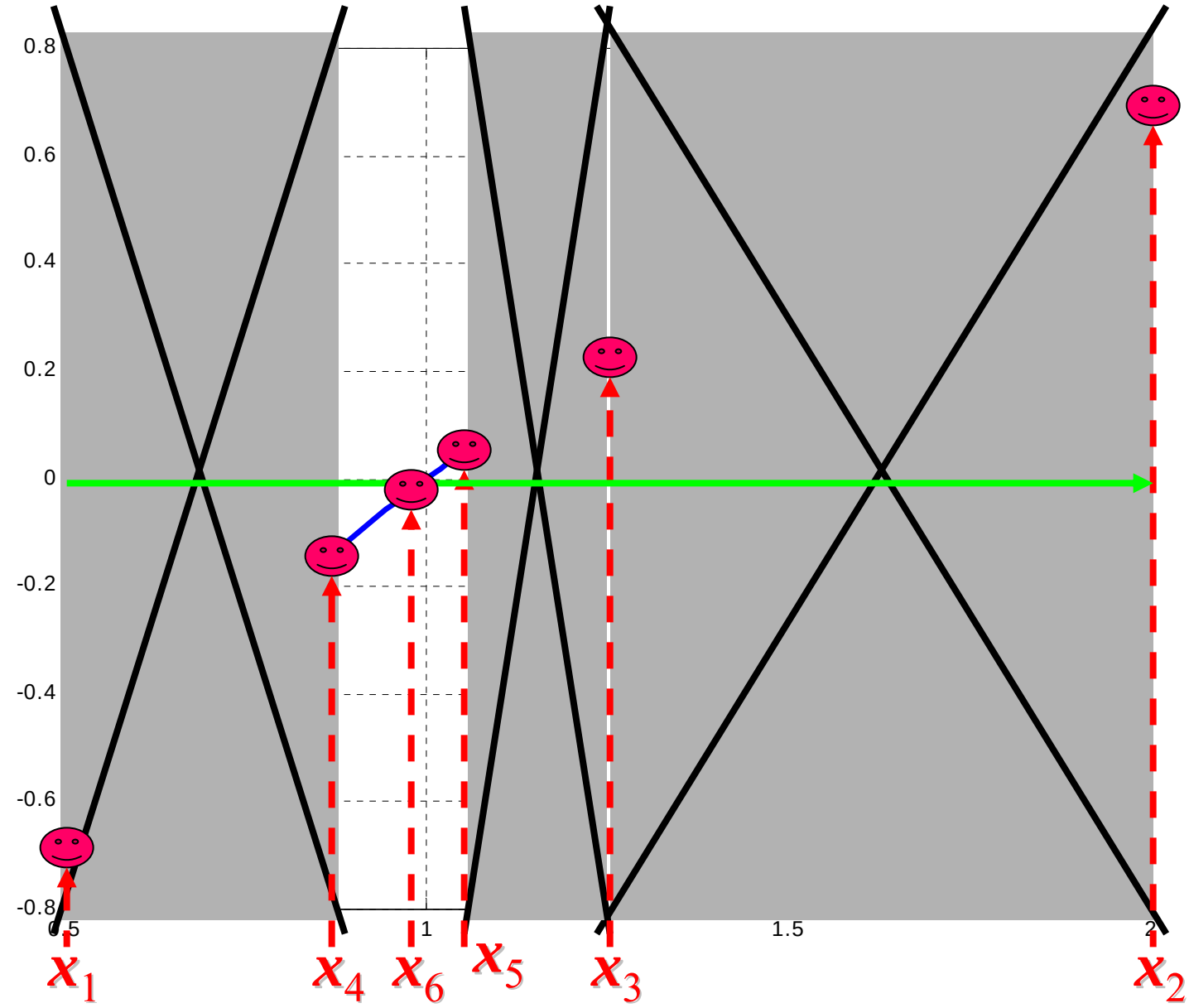
intervallo $[a,b]$ $\quad x_1=a,\ x_2=b,\ x_3=(x_1+x_2)/2$

nuovo intervallo $=\begin{cases} [x_1, x_3] & \text{se } f(x_1)f(x_3)<0 \\ \text{oppure} & \\ [x_3, x_2] & \text{altrimenti} \end{cases}$

Si ripete lo stesso ragionamento nel nuovo intervallo.

L'iterazione **termina** quando l'ampiezza dell'intervallo contenente lo zero è diventata sufficientemente piccola.

Esempio: $f(x)=\log x, \quad x\in[0.5, 2]$



$f(x_{inf})$	$f(x_m)$	$f(x_{sup})$
<0	>0	>0
<0	<0	>0
<0	>0	>0
<0	<0	>0

...