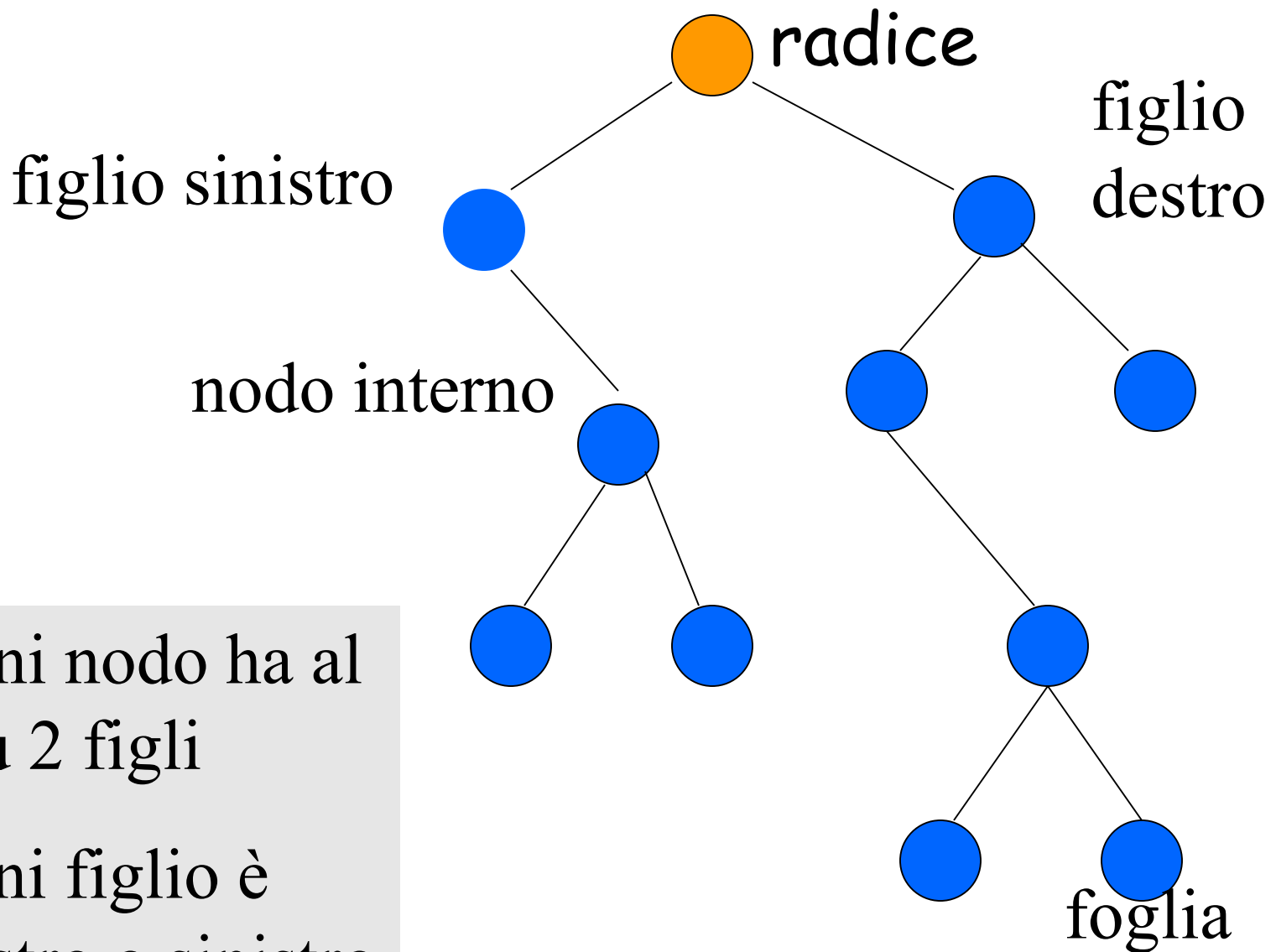


alberi binari e ricorsione

cap. 12 del testo

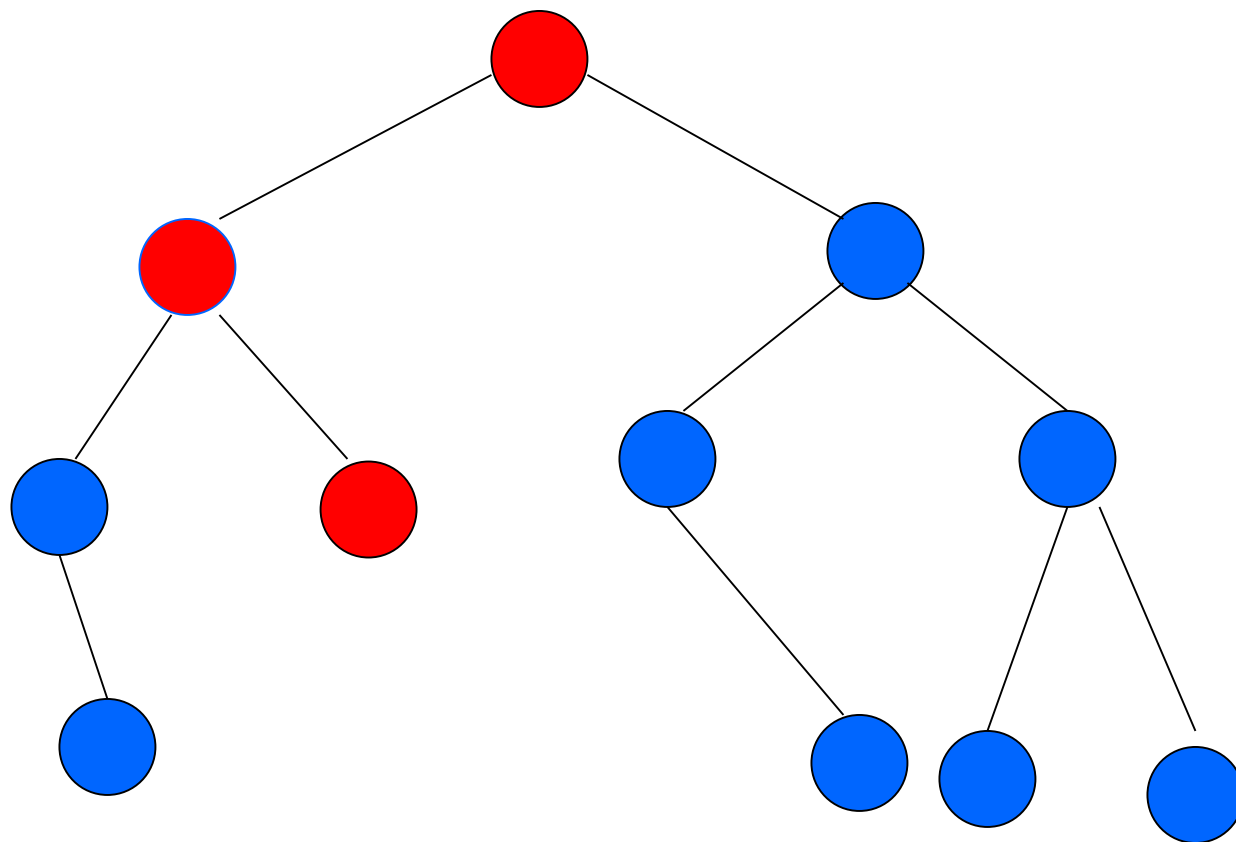
alberi binari



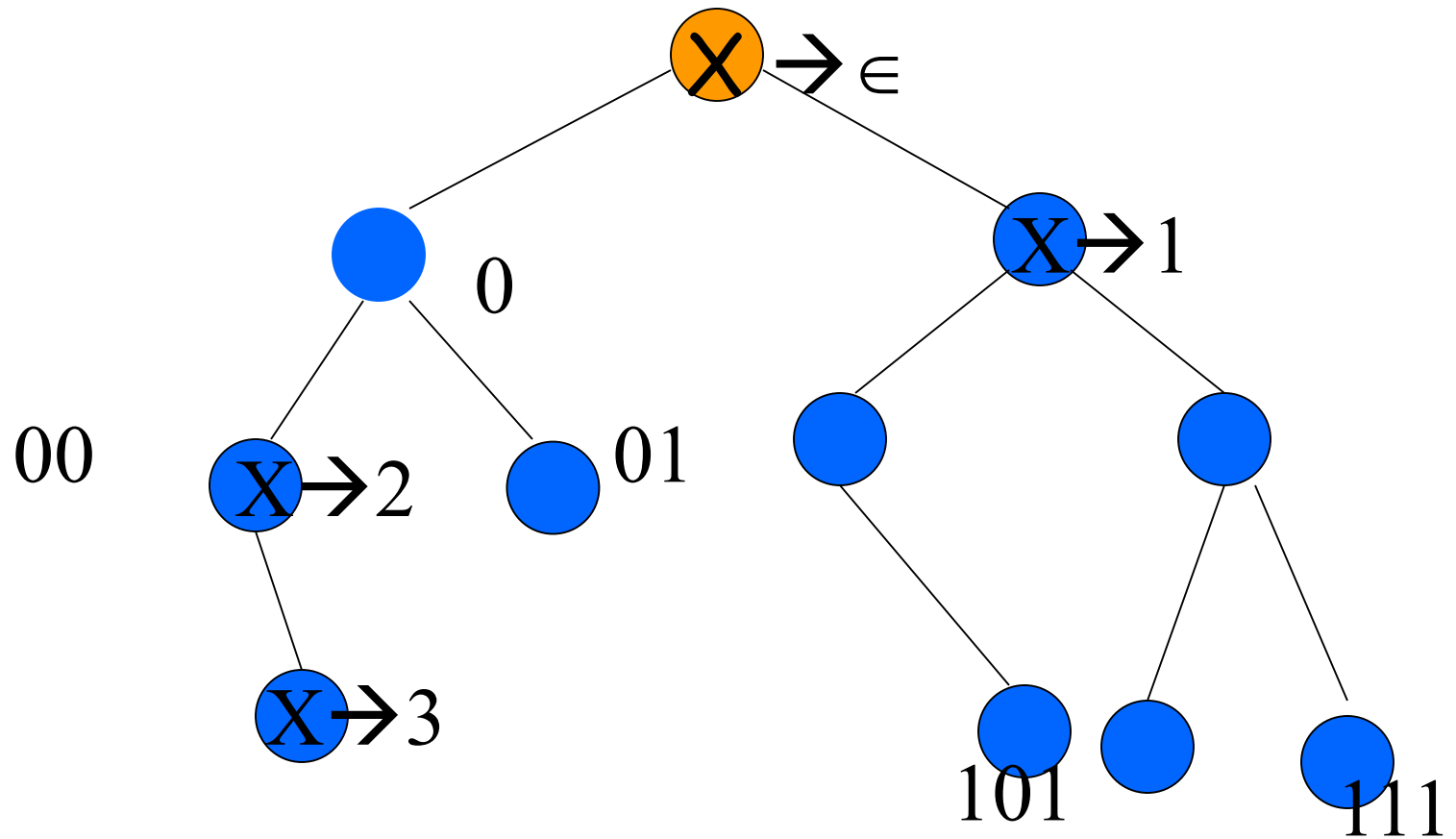
ogni nodo ha al
più 2 figli

ogni figlio è
destro o sinistro

un cammino da un nodo fino ad una foglia
assomiglia molto ad una lista concatenata



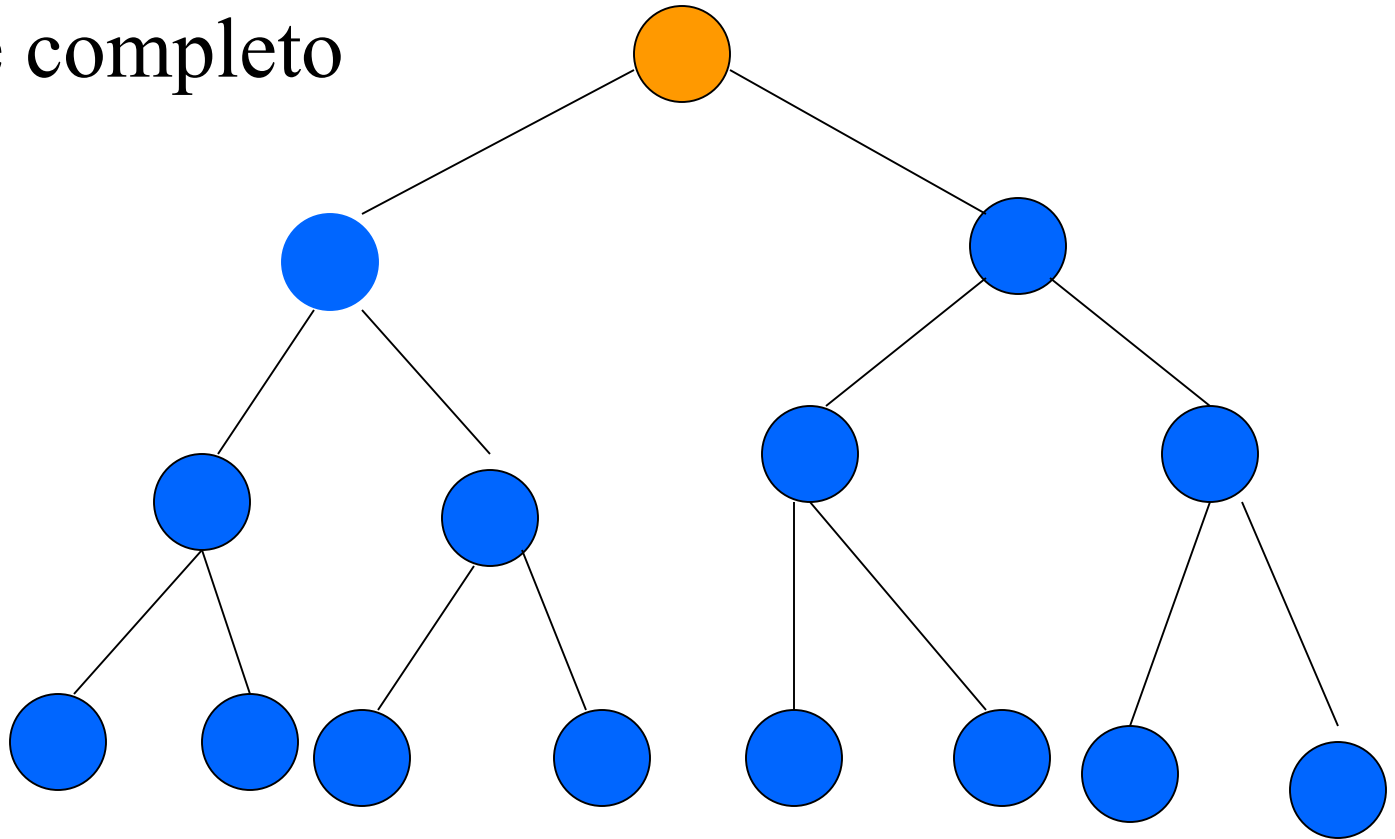
cammini = sequenze di nodi = sequenze di 0 e 1



profondità di un nodo

altezza dell'albero=prof. max delle foglie

albero binario completo, se ogni
livello è completo



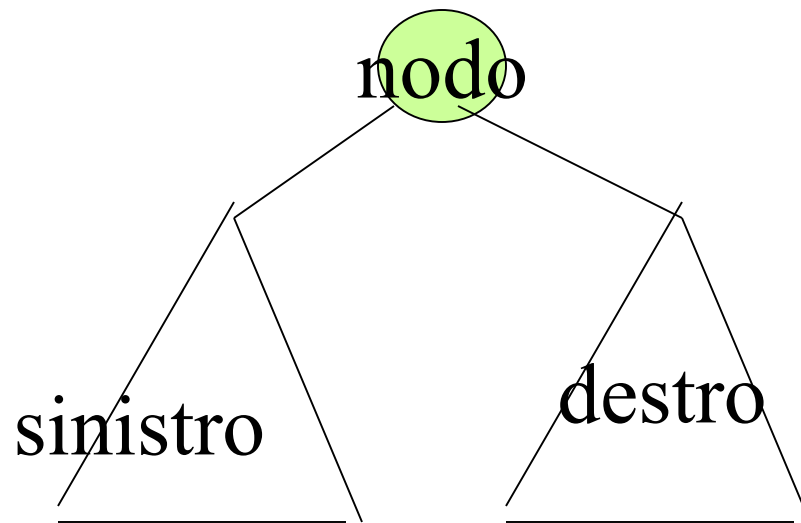
h = altezza

l'albero contiene $2^{h+1} - 1$ nodi

definizione ricorsiva degli alberi:

albero binario è:

- un albero vuoto
- `nodo(albero sinistro, albero destro)`



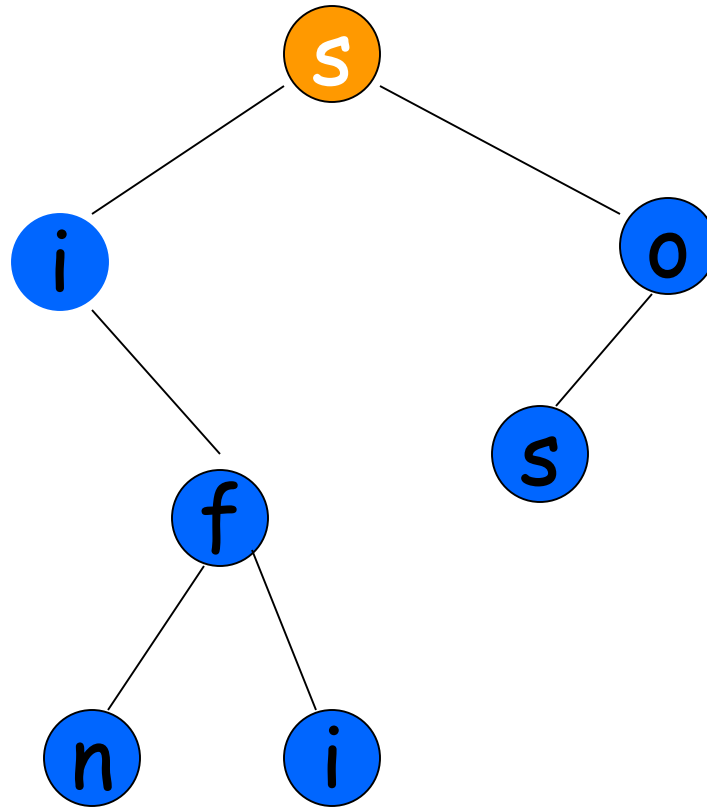
attraversamento di un albero è un modo di visitare tutti i suoi nodi

in profondità = depth-first

ma anche in larghezza = breath-first

percorso in profondità infisso:

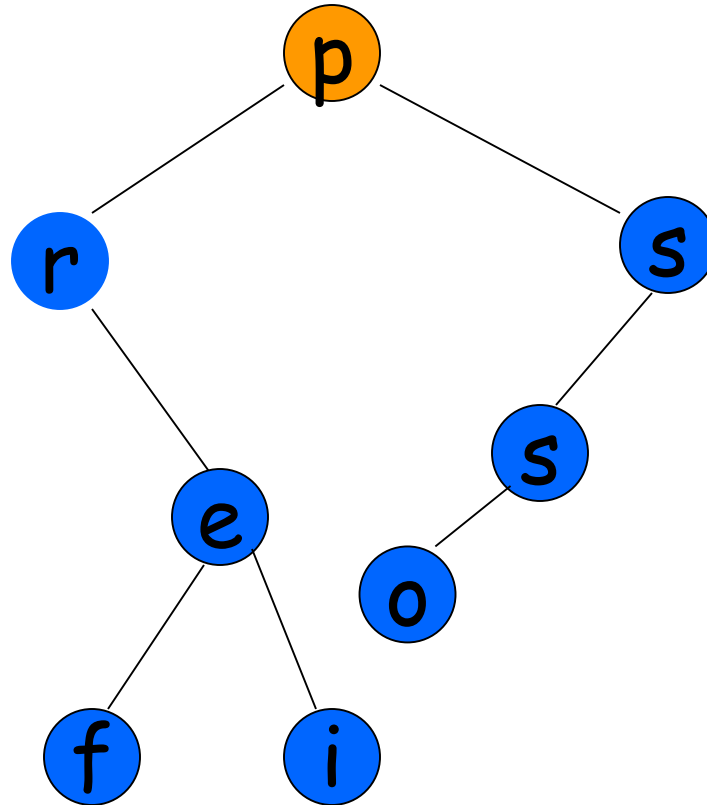
1. a sinistra
2. radice
3. a destra



in profondità da sinistra a destra

percorso in profondità prefisso:

1. radice
2. a sinistra
3. a destra

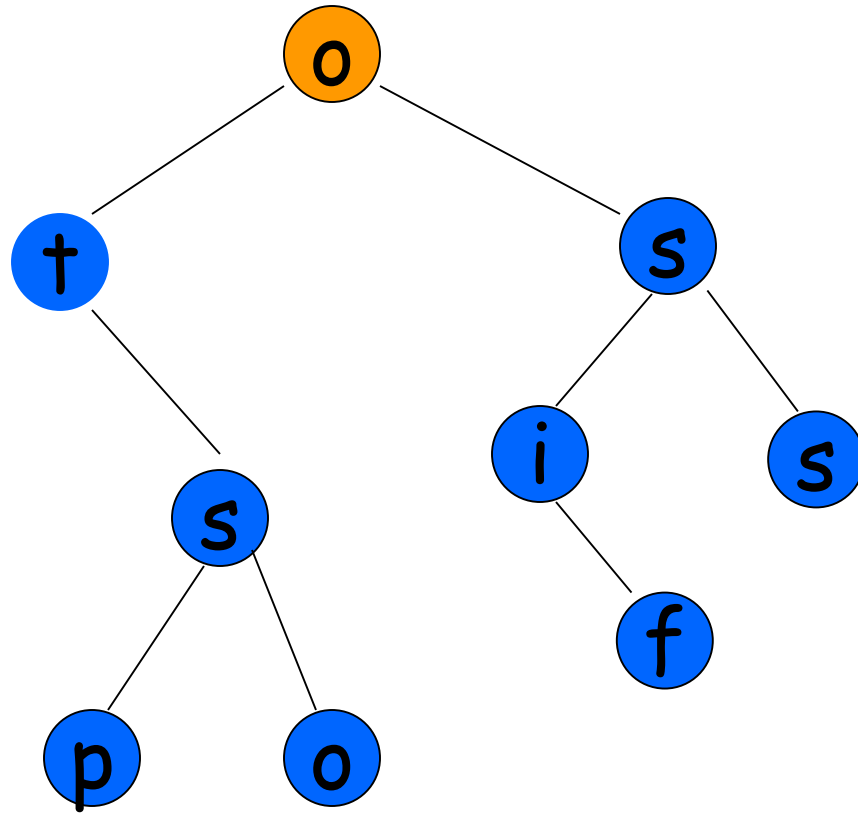


percorso in profondità postfisso:

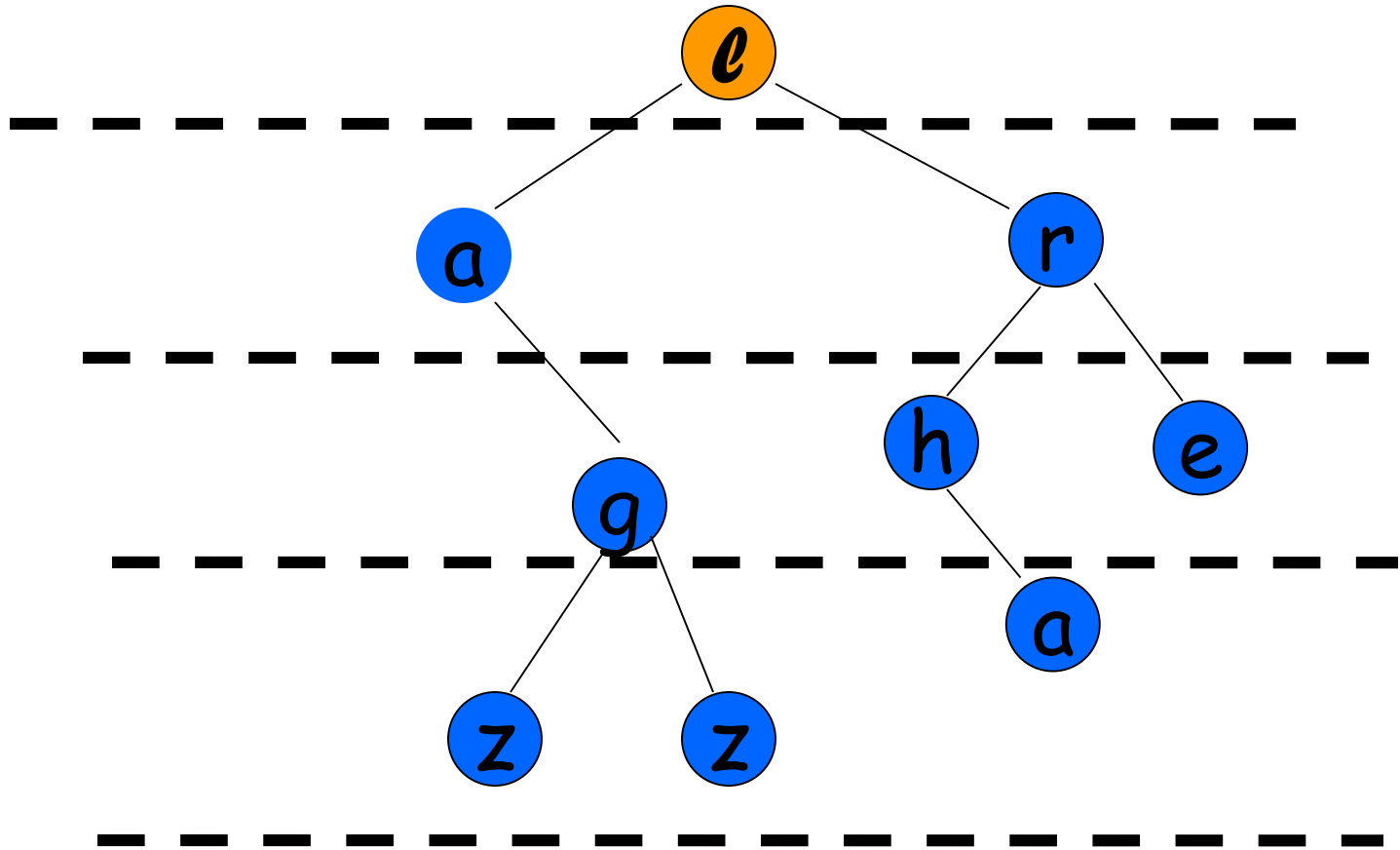
1. a sinistra

2. a destra

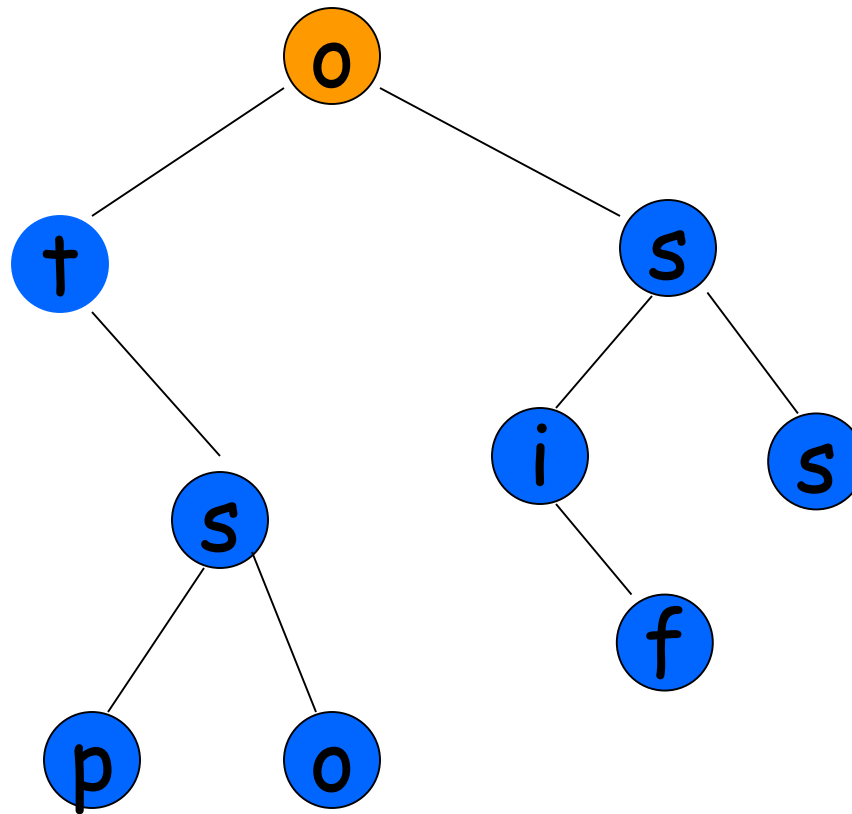
3. radice



in larghezza



ogni percorso stabilisce un ordine totale tra i nodi



infisso: t p s o o i f s s

prefisso: o t s p o s i f s

postfisso: p o s t f i s s o

come realizzare un nodo di un albero binario in C++:

```
struct nodo {
```

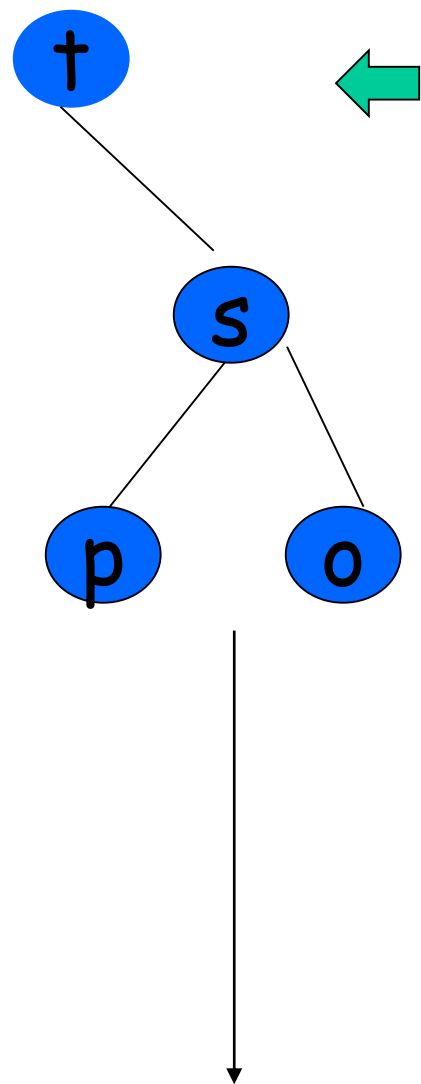
```
char info;
```

```
nodo* left, *right;
```

```
nodo(char a='\0', nodo*b=0, nodo* c=0)
```

```
{info=a; left=b; right=c;}
```

```
};
```



costruiamo questo albero:

```
nodo * root=new nodo('t');
```

```
root→right=new nodo('s');
```

```
root→right→left=new nodo('p');
```

```
root→right→right=new nodo('o');
```

l'ordine è essenziale !

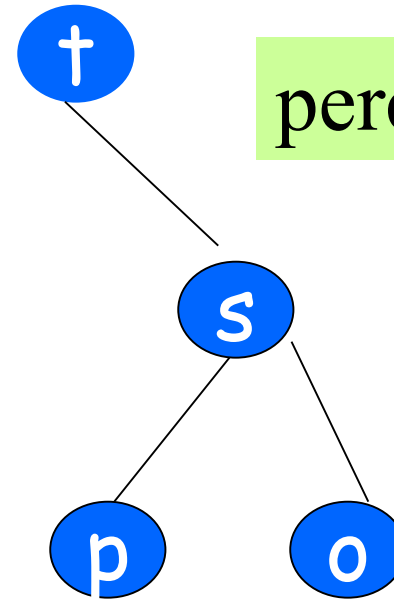
t(,s(p(,_,o(,_,_))) rappresentazione lineare

percorso prefisso

```

void stampaLin(nodo *r)
{
    if(r)
    {
        cout<<r->info<<'(';
        stampaLin(r->left);
        cout<<',';
        stampaLin(r->right);
        cout<<')';
    }
    else
        cout<< '_';
}

```

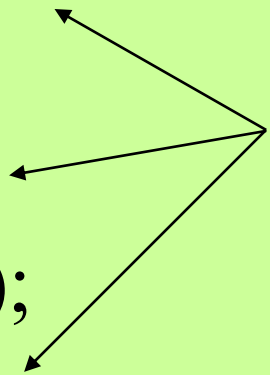


percorso prefisso

$t(_, s(p(_, _), o(_, _)))$

stampa secondo i 3 percorsi

```
void print(nodo *x) {  
    if(x)  
    {  
        print(x->left);  
        print(x->right);  
    }  
}
```



cout << x->info;

invocazione: infix(root);

alberi binari e ricorsione

esercizio: trovare e restituire un nodo con un campo
info = y

dobbiamo esplorare tutti i nodi di un albero

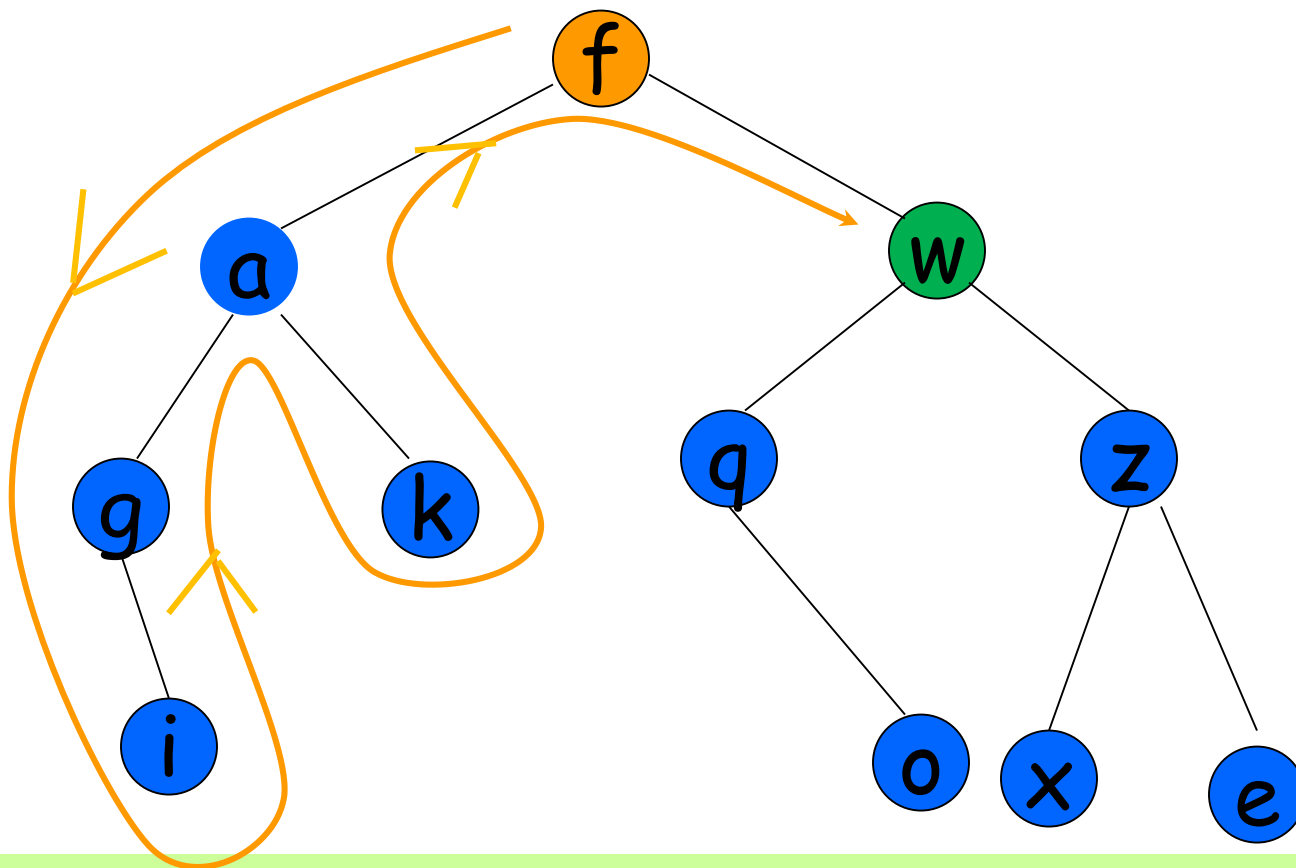
potremmo scegliere un percorso qualsiasi

quello prefisso sembra più ragionevole degli altri:
se arriviamo su un nodo guardiamo subito se il suo
info=x

PRE=(albero(x) ben formato)

```
nodo* trova(nodo *x, char y){  
  if(!x) return 0;    // fallimento  
  if(x->info==y) return x;  
  nodo * z= trova(x->left,y);  
  if(z) return z;  
  return trova(x->right,y);  
}
```

**POST=(se in albero(x) c'è un nodo con
info=y, restituisce il primo tale nodo secondo
l'ordine prefisso, altrimenti restituisce 0)**



cerchiamo w, la pila dei RA corrisponde ai
cammini percorsi

f -> fa -> fag -> fagi -> fag -> fa -> fak -> fa -> f
-> fw

altezza di un albero = profondità massima delle foglie

 altezza 0

 altezza 1

albero vuoto? per convenzione -1

PRE=(albero(x) ben formato)

int altezza(nodo *x)

{

if(!x) return -1; //albero vuoto

else

{

int a=altezza(x->left);

int b=altezza(x->right);

if(a>b) return a+1;

return b+1;

}

} **POST=(restituisce l'altezza di albero(x))**

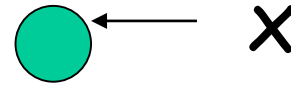
proviamo che è corretto:

base albero vuoto $\Rightarrow -1$

```
int altezza(nodo *x)
{
    if(!x) return -1;
    else {
        int a=altezza(x->left);
        int b=altezza(x->right);
        if(a>b) return a+1;
        return b+1;
    }
}
```

-1 OK

un solo nodo



```
int altezza(nodo *x)
{
    if(!x) return -1;
    else {
        int a=altezza(x->left);
        int b=altezza(x->right);
        if(a>b) return a+1;
        return b+1;
    }
}
```

a = -1
b = -1
return 0

OK

in generale:

```
int altezza(nodo *x)
```

```
{
```

```
    if(!x) return -1;
```

```
    else {
```

```
        int a=altezza(x->left);
```

```
        int b=altezza(x->right);
```

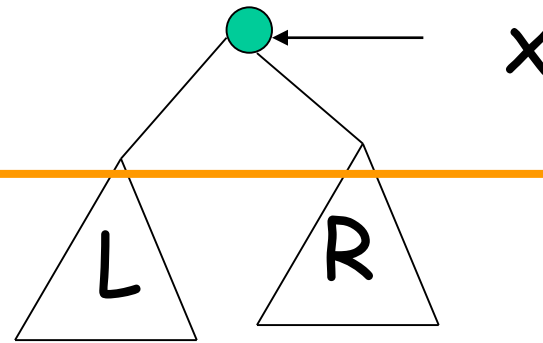
```
        if(a>b) return a+1;
```

```
        return b+1;
```

```
    }
```

```
}
```

```
}
```



maggiore delle 2
+ 1 OK

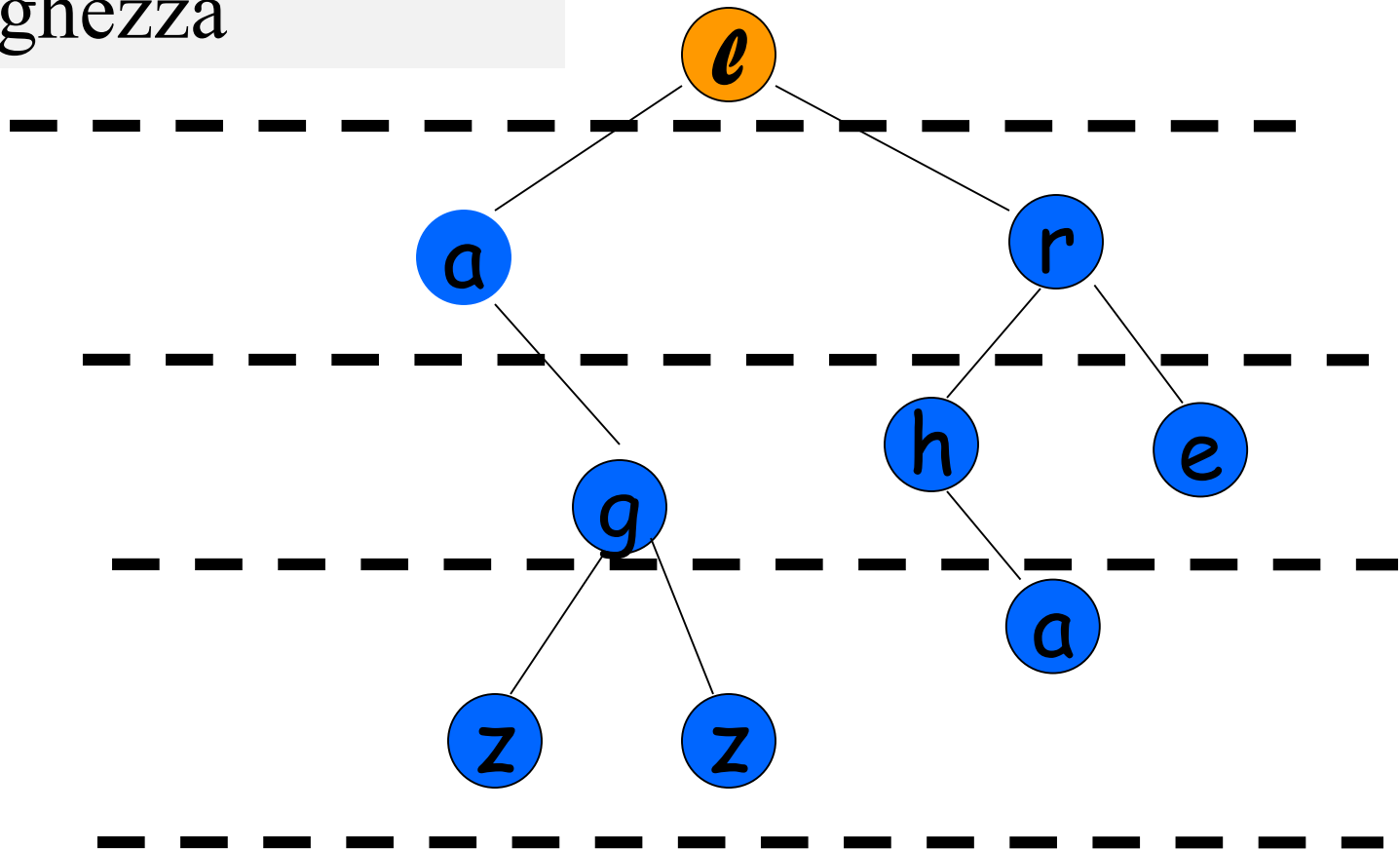
potremmo anche evitare di considerare l'albero vuoto per l'altezza.

PRE=(albero(x) ben formato e non vuoto)

esercizio: scrivere la funzione altezza che rispetta questa PRE e la POST=(restituisce l'altezza di albero(x))

fine parte 2

realizzare il percorso
in larghezza



i puntatori nell'albero vanno da padre a figli, invece vogliamo percorrere i fratelli, i cugini, i nipoti, ecc.

dobbiamo costruirci una lista con nodi che puntino a quelli dell'albero e che ci permetta di attraversarlo in larghezza

```
struct nodoF {nodo*info; nodoF*next;};
```

```
nodoF::nodoF(nodo*a=0, nodoF*b=0)  
{info=a; next=b;}
```

dobbiamo togliere nodi dall'inizio della lista e aggiungere nodi alla fine della lista
coda FIFO = FIRST IN-FIRST OUT

```
struct FIFO {nodoF*primo, *ultimo;};  
FIFO::FIFO(nodo*a=0)  
{  
    if(a)  
        primo=ultimo=new nodoF(a);  
    else  
        primo=ultimo=0;  
}
```

diremo che un valore FIFO gestisce sempre
una lista nodoF ben formata

PRE=(FIFO x gestisce lista ben formata e non vuota, vx=x)

```
nodo* pop(FIFO & x)
{
    nodoF*a=x.primo;
    x.primo=x.primo->next;
    if(!x.primo)
        x.ultimo=0;
    return a->info;
}
```

POST=(restituisce col return il nodo*del primo nodo di vx e x è vx senza il primo nodo)

PRE=(y è FIFO che gestisce lista nodoF ben formata, vy=y)

FIFO push(nodo*x, FIFO y)

{

if(y.primo)

{

y.ultimo->next=new nodoF(x);

y.ultimo=y.ultimo->next;

return y;

}

return FIFO(x);

} POST=(y è vy con nodoF(x) in fondo)

percorso in larghezza:

```
void breadthFirst(nodo*r) // PRE=(albero(r) ben for.)
{
    FIFO coda(r);
    while(coda.primo)
    {
        nodo*x=pop(coda);
        cout<<x->info<<' ';
        if(x->left)  coda=push(x->left,coda);
        if(x->right) coda=push(x->right,coda);
    }
} POST=(stampa i campi info secondo il percorso in
larghezza)
```

esercizio

definite una funzione ricorsiva che stampi i
nodi di un albero secondo il percorso in
larghezza