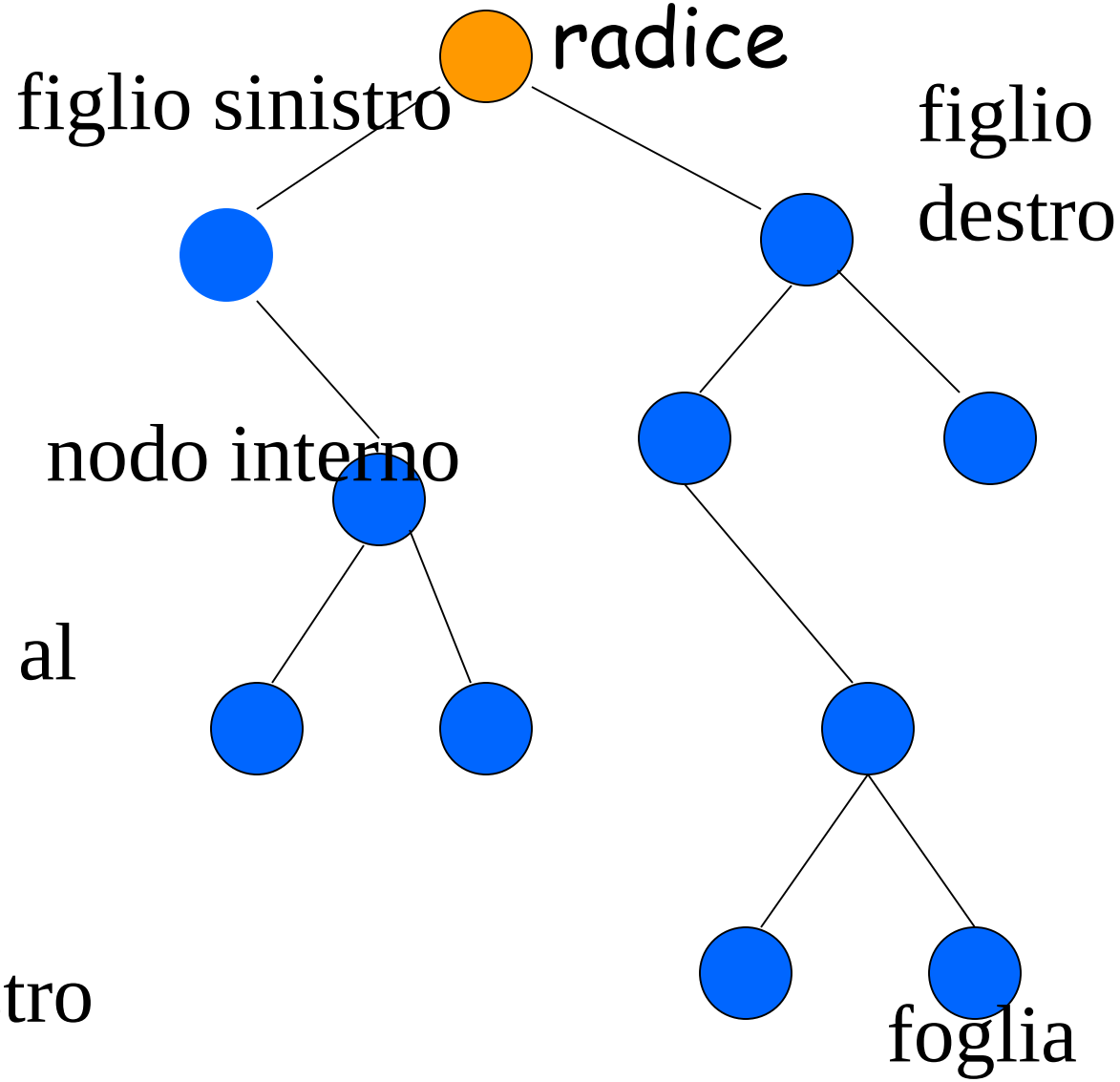
alberi binari e ricorsione

cap. 12

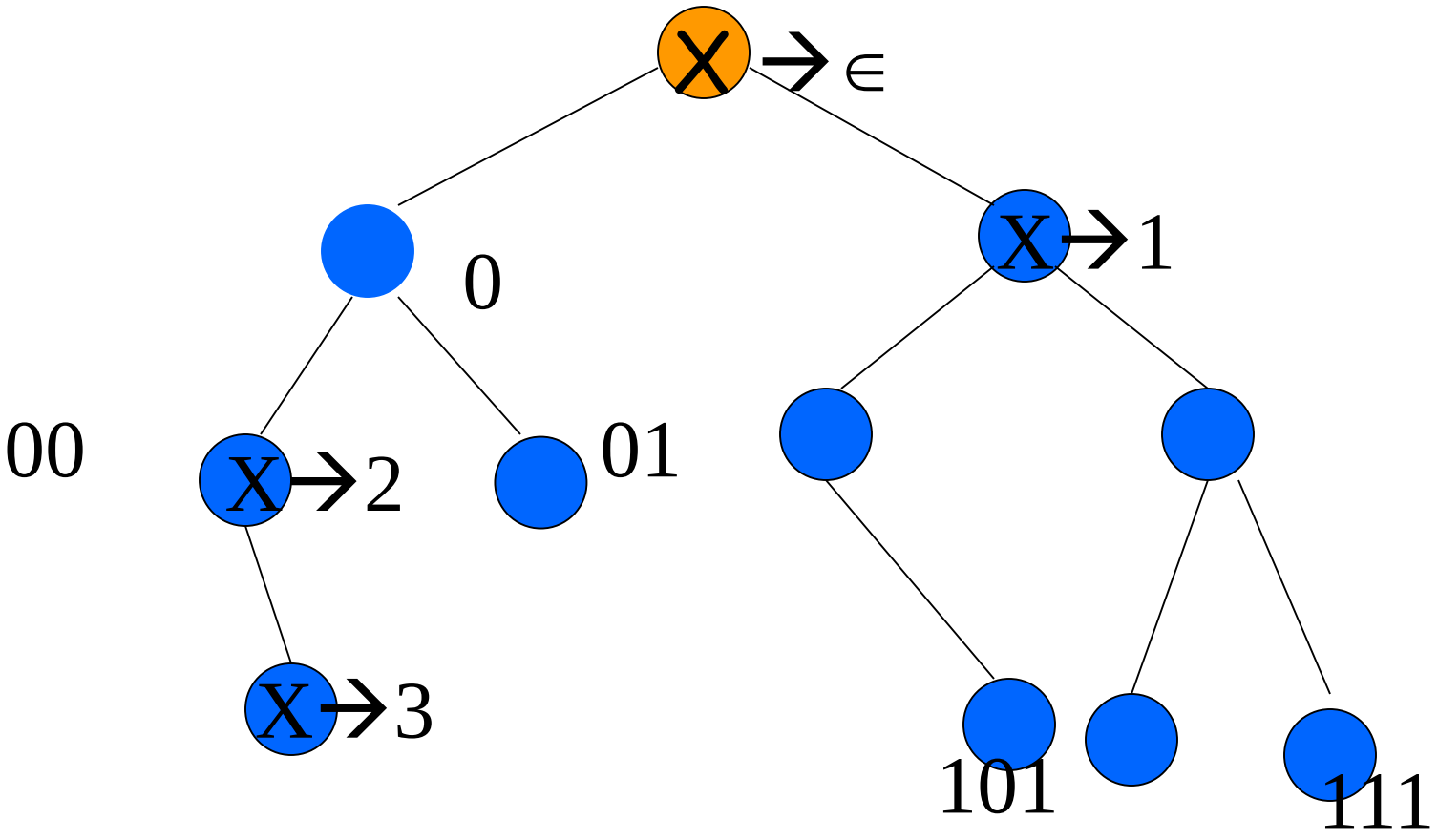
un albero binario:



ogni nodo ha al più 2 figli

ogni figlio è destro o sinistro

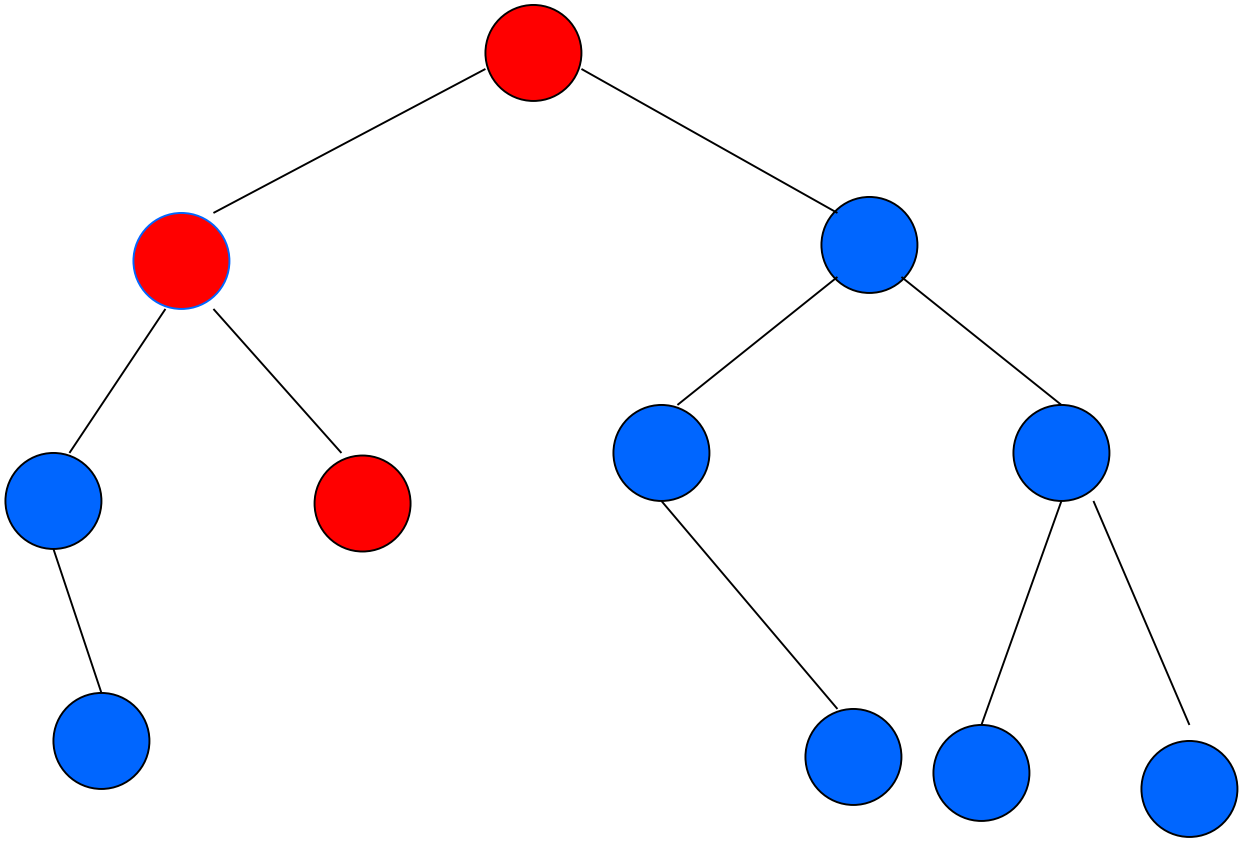
cammini = sequenze di nodi = sequenze di 0 e 1



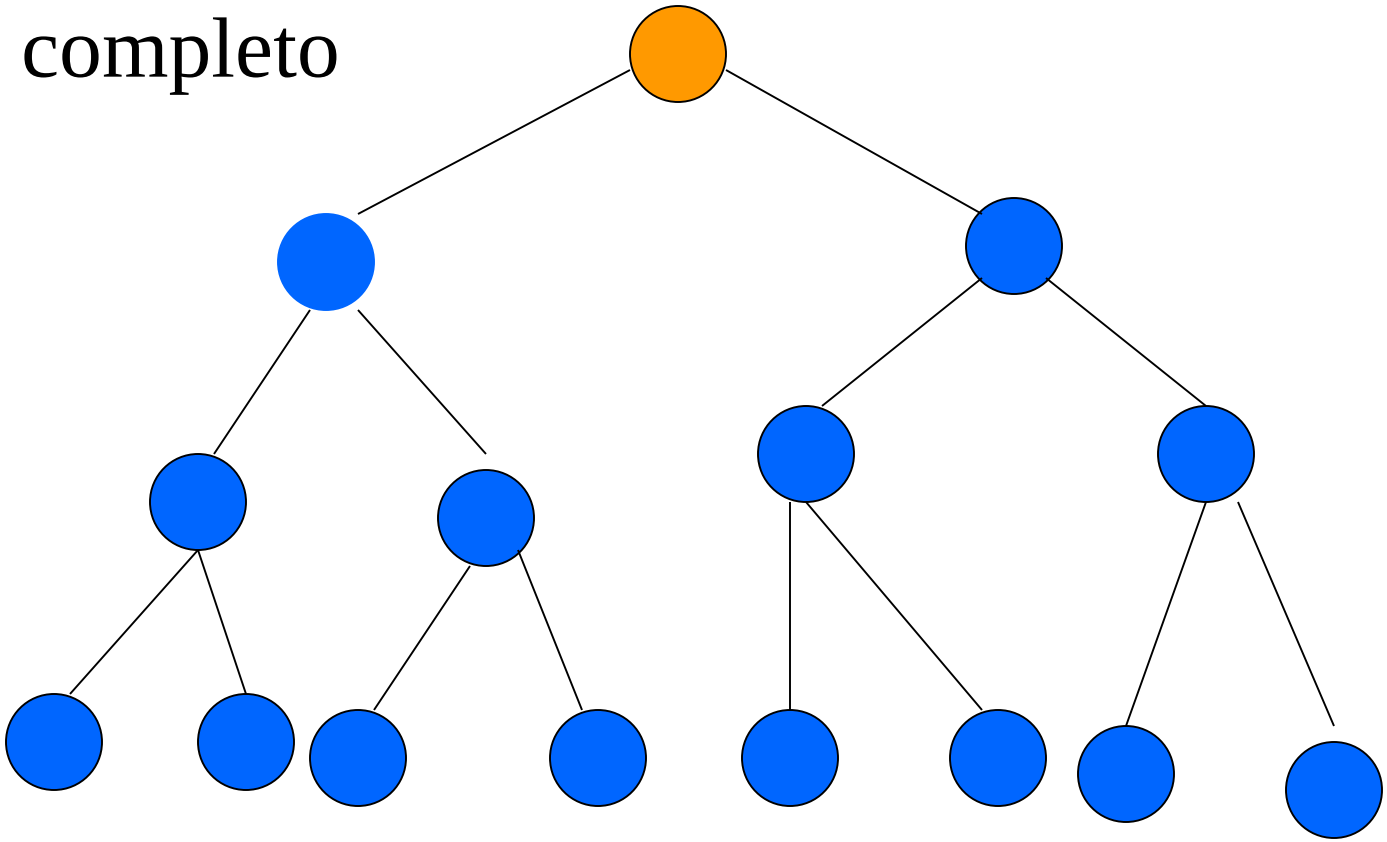
profondità di un nodo

altezza dell'albero=prof. max delle foglie

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



albero binario completo, se ogni livello è completo



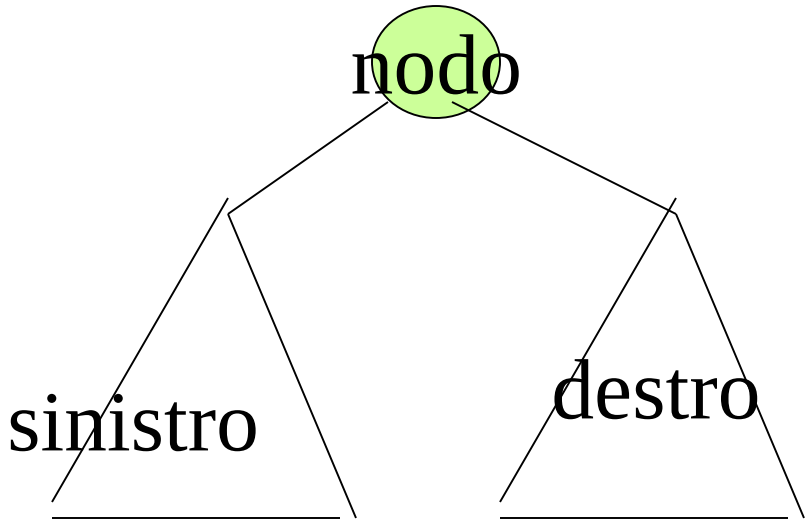
h= altezza

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

definizione ricorsiva degli alberi:

albero binario è:

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



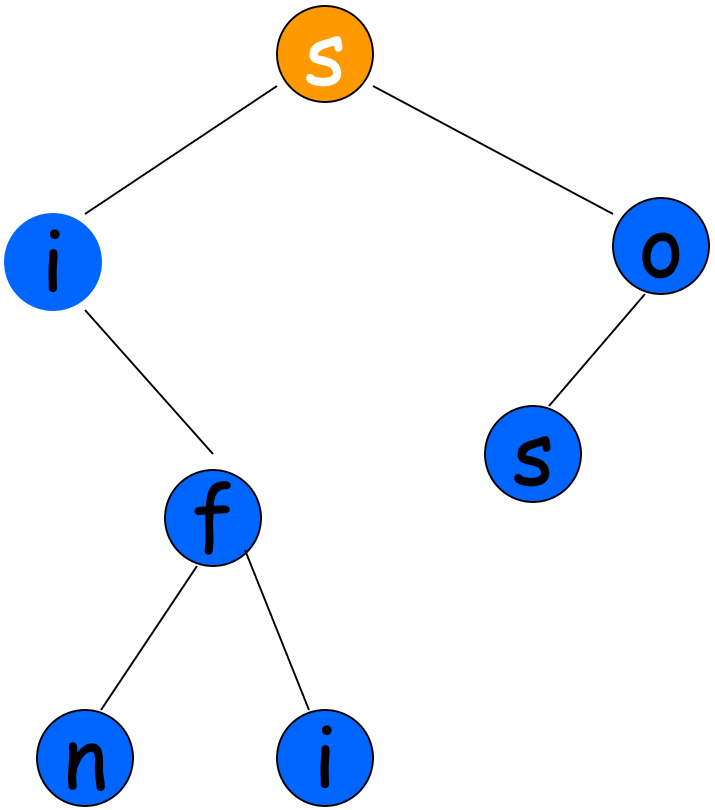
attraversamento di un albero è un modo di visitare tutti i loro nodi

in profondità = depth-first

ma anche in larghezza = breath-first

percorso in profondità infisso:

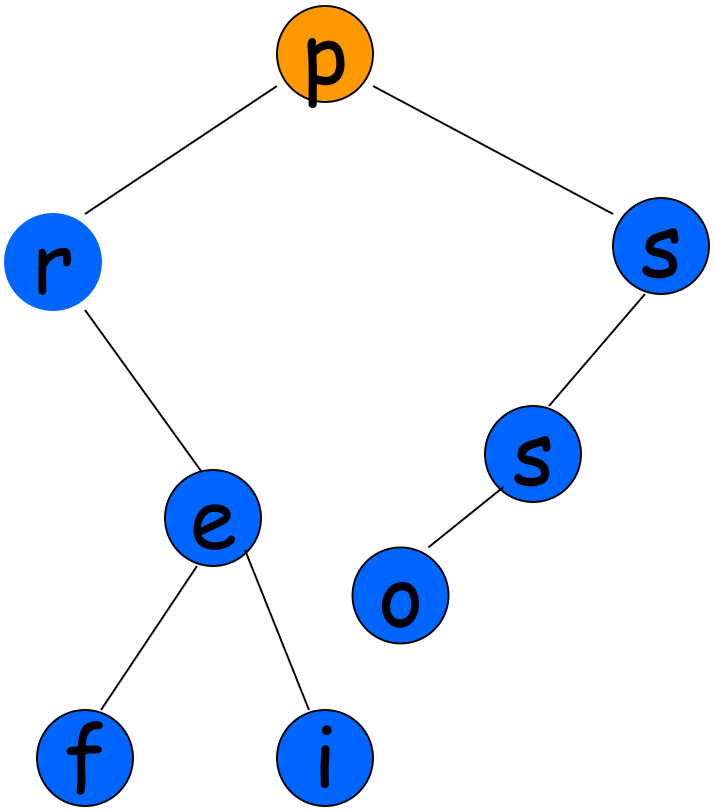
1. a sinistra
2. nodo
3. a destra



in profondità da sinistra a destra

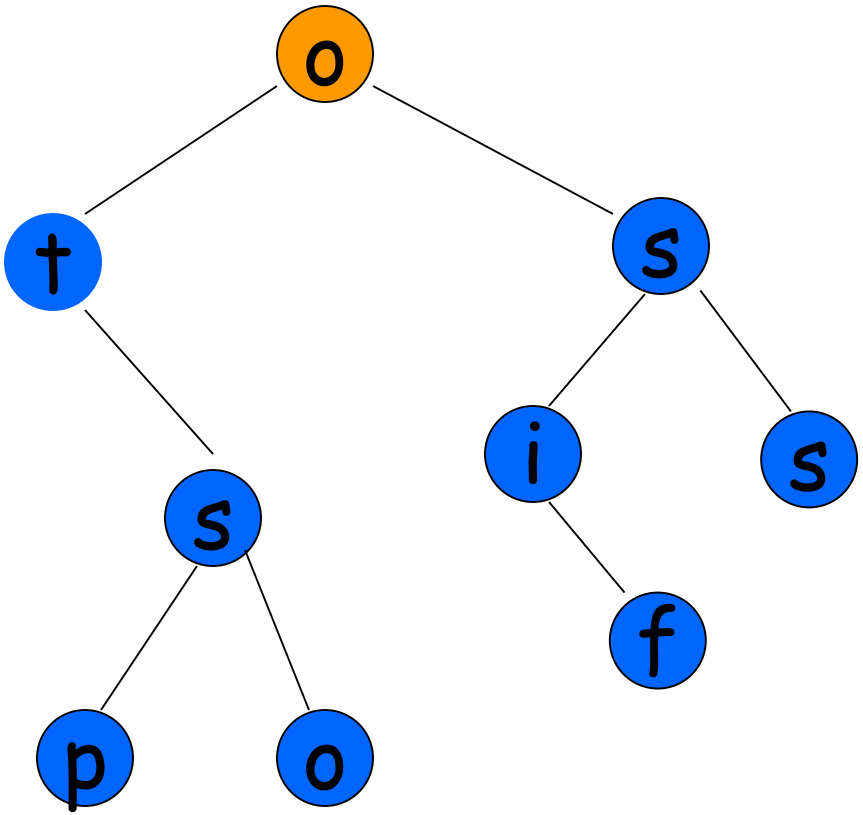
percorso in profondità prefisso:

1. nodo
2. a sinistra
3. a destra

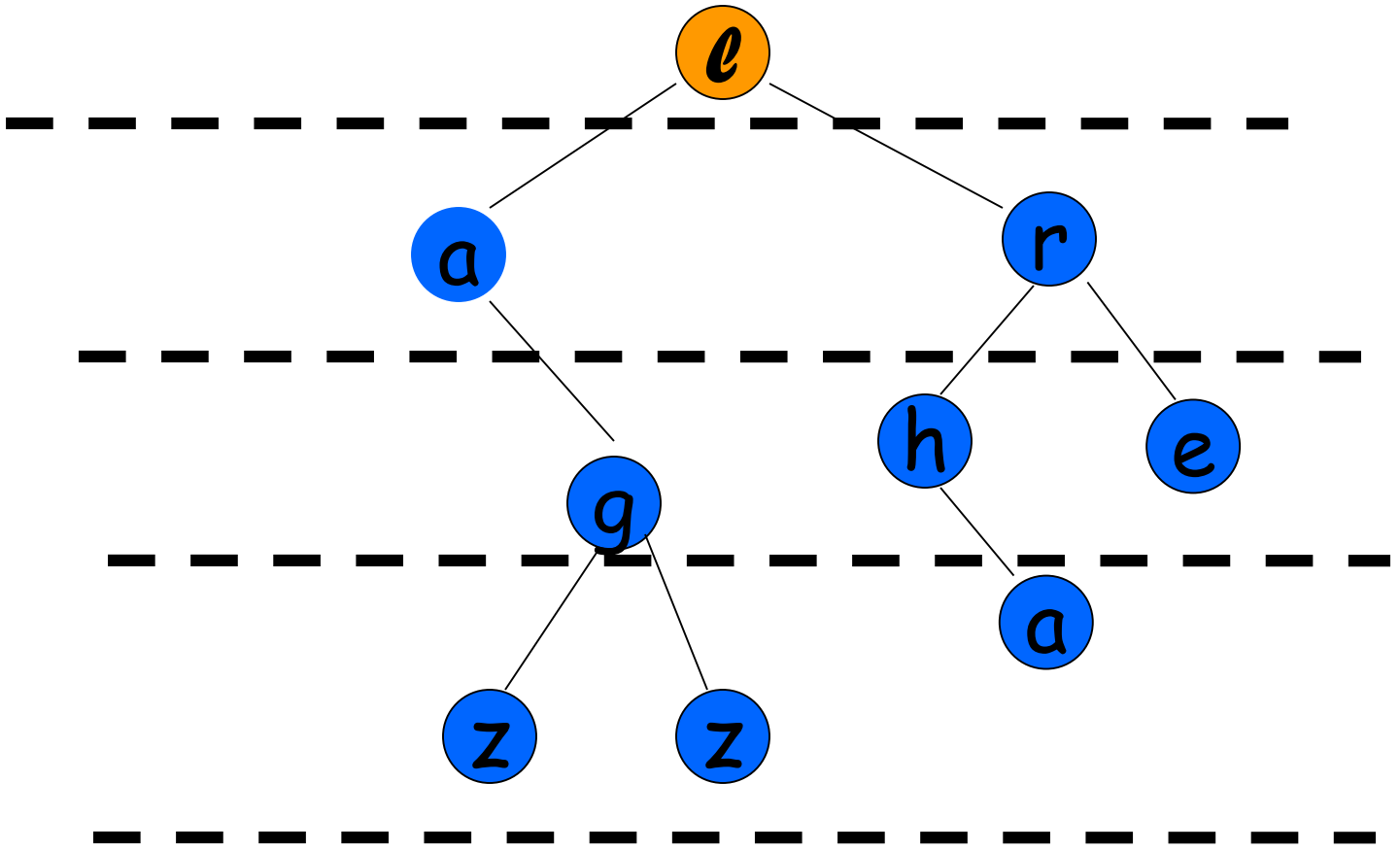


percorso in profondità postfisso:

1. a sinistra
2. a destra
3. nodo

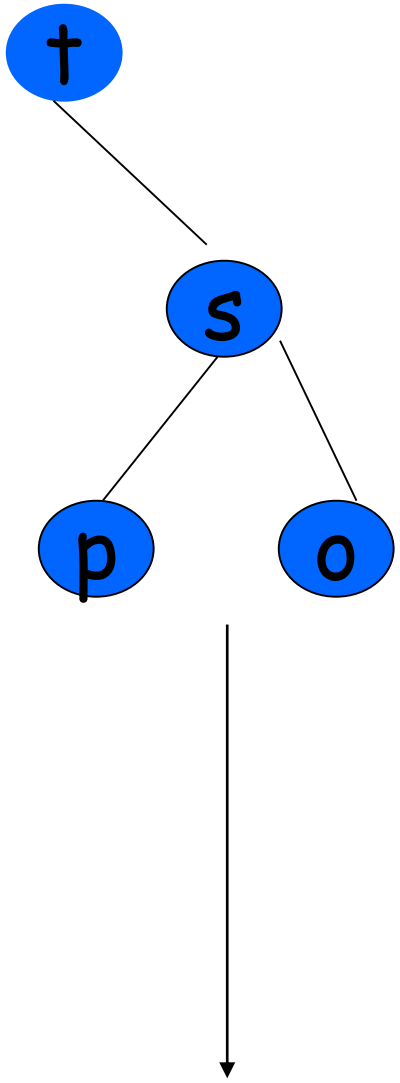


in larghezza



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’,0,0);

root→right=new nodo();

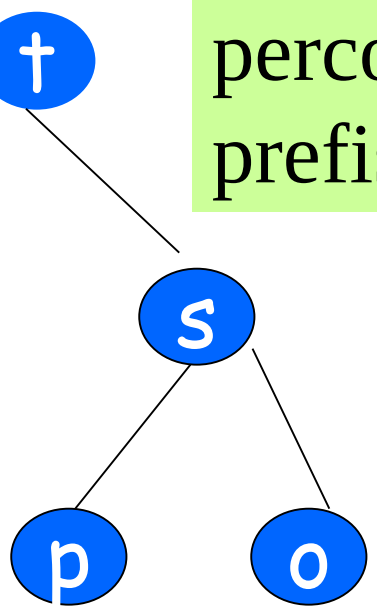
root→right→info=‘s’;

root→right→left=new nodo(‘p’,0,0);

root→right→right=new nodo(‘o’,0,0);

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

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



percorso prefisso

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

stampa in ordine infisso:

```
void infix(nodo *x){
 if(x) {
 infix(x->left);   // stampa albero sinistro
 cout<<x->info<<‘ ‘; // stampa nodo
 infix(x->right);  // stampa albero destro
}}
```

invocazione: infix(root);

trovare e restituire un nodo con un campo info =y

```
nodo* trova(nodo *x, char y){




}
```

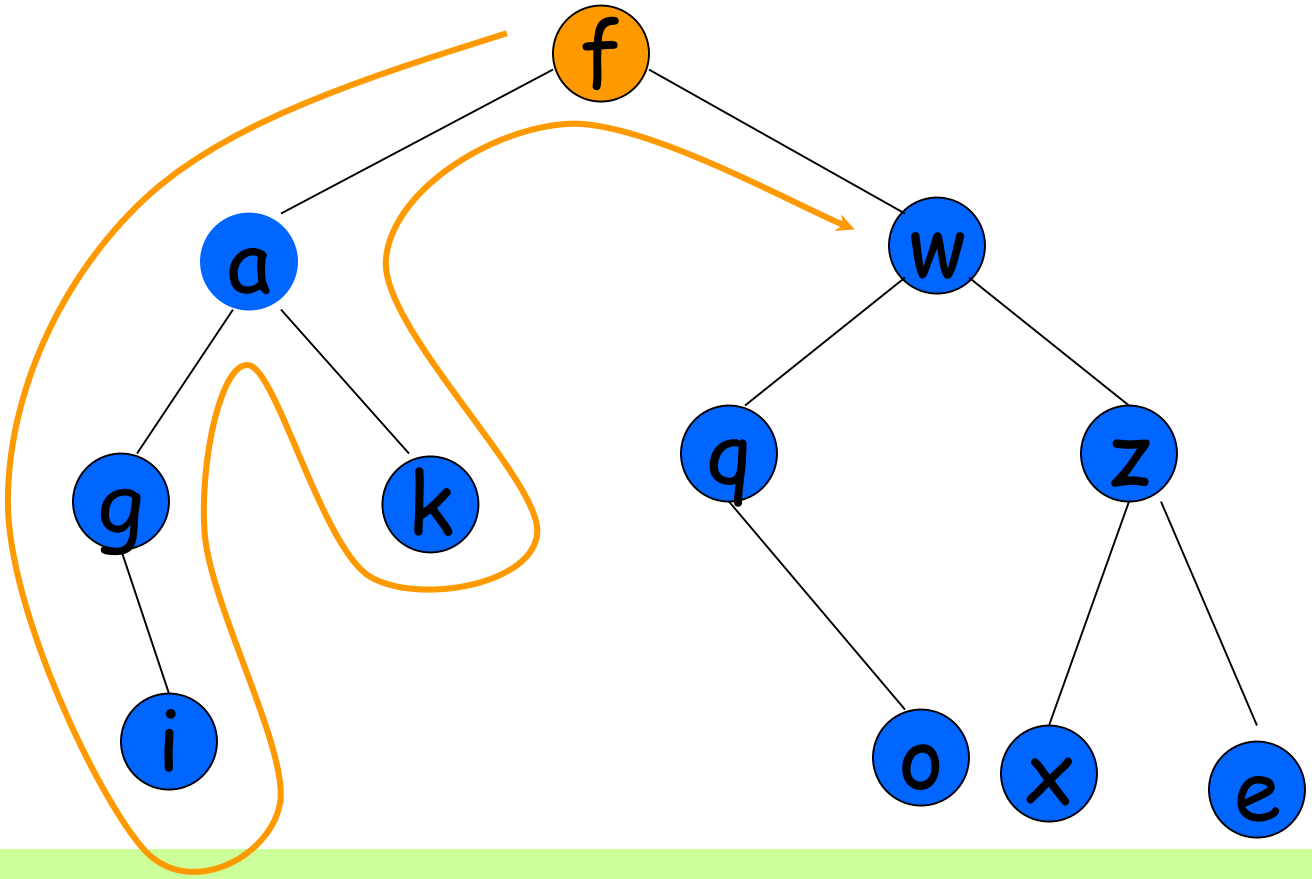
invocazione:

nodo *w=trova(root,y)

PRE=(albero(x) ben formato)

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

POST=(restituisce nodo* !=0 sse in albero(x) esiste un nodo con info=y) &&(se c'è restituisce il primo nodo nell'ordine prefisso)

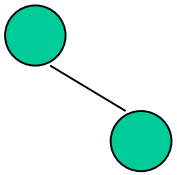


cerchiamo w, la ricorsione corrisponde ai cammini percorsi

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

altezza di un albero = profondità massima dei suoi nodi = distanza massima tra 2 nodi dell'albero

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 => -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;
           }
}
```

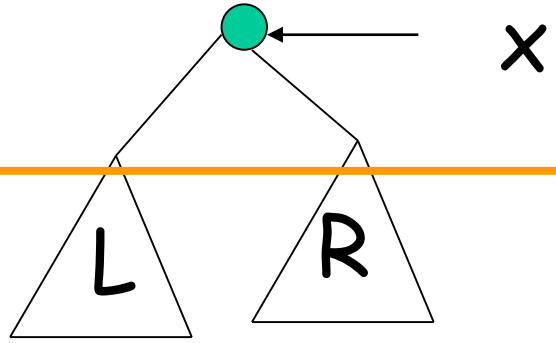
a = -1

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) corretto e non vuoto)

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

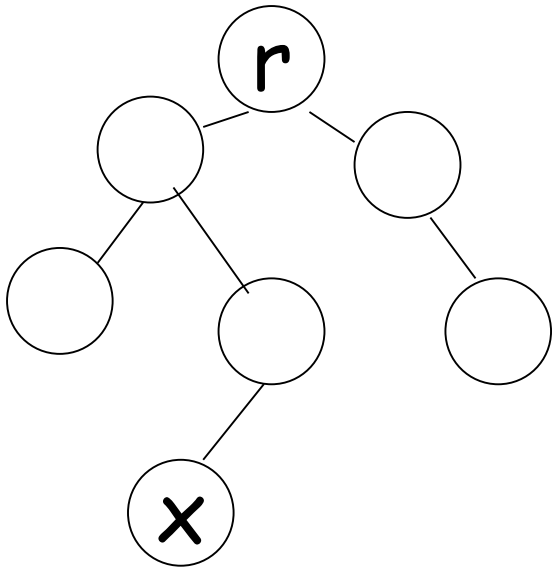
```
int altezza(nodo* x)
{
  if(!x->left && !x->right)
      return 0;
  int a=-1, b=-1;
  if(x->left) a=altezza(x->left);
  if(x->right) b=altezza(x->right);
  if(a>b) return a+1;
    else return b+1;
}
```

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

un cammino di un albero = sequenza di 0 e 1

0=sinistra 1= destra

array int C[] e il valore lung indica la lunghezza della sequenza:



cammino da r a x:

C=[010] lung=3

cammino da r a r C=[] e lung =0

Problema

dato un array C che contiene una sequenza di 0 e 1 e un albero ben formato, restituire il nodo corrispondente ...., se c'è

invocazione: nodo *z= trova(root, C, lung);

PRE=(albero(x) ben formato, lung >=0, C[0..lung-1] def e 0/1)

```
nodo * trova(nodo *x, int*  C, int lung)
{
   if(!x) return 0;  // fallito
   if(lung==0) return x;  //trovato
   if(*C==0) return trova(x->left, C+1, lung-1);
   else
        return trova(x->right,C+1, lung-1);
}
```

POST=(restituisce punt. a nodo alla fine del cammino C[0..lung-1], se c'è in albero(x), e altrimenti 0)