

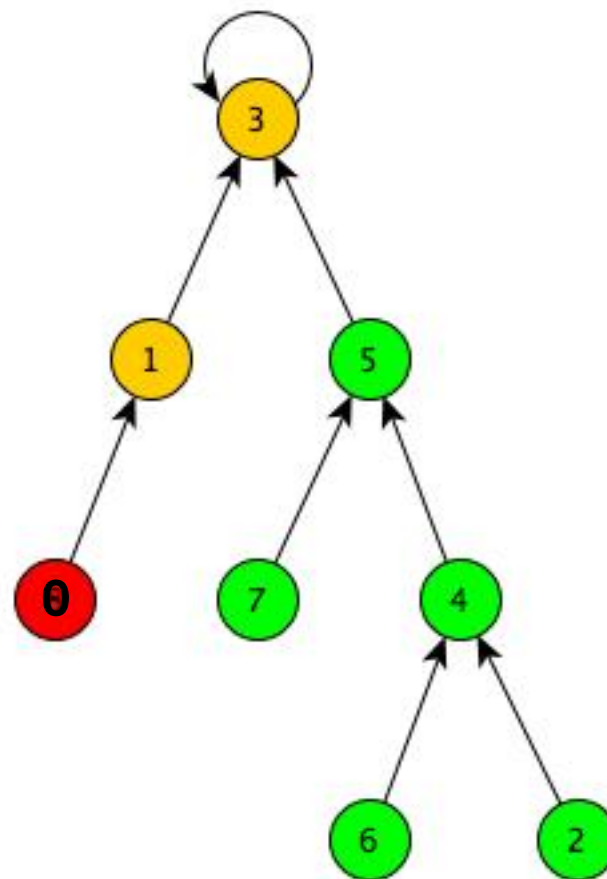
ESERCIZIO:

Dato un albero T di n nodi rappresentato tramite il vettore dei padri P (per convenzione il padre del nodo radice è il nodo stesso) e un suo nodo x , dare lo pseudocodice di un algoritmo che in tempo $O(n)$ produce la lista dei nodi di T presenti nel sottoalbero radicato in x .

Ad esempio per

$$P = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 3 & 4 & 3 & 5 & 3 & 4 & 5 \\ \hline \end{array}$$

e $x = 5$ la funzione deve restituire l'insieme $\{2,4,5,6,7\}$ mentre per $x = 0$ deve restituire $\{0\}$



Ricerca esaustiva: per ogni nodo verifica se appartiene o meno al sottoalbero radicato in x .

- La funzione di test per ogni nodo dell'albero: passa in rassegna i suoi antenati, se tra questi è presente il nodo x allora aggiungi il nodo all'insieme soluzione

```
def es(x,P):  
    return {u for u in range(len(P)) if test(u,x,P)}  
  
def test(u,x,P):  
    while u !=x and P[u] !=u: u=P[u]  
    if u==x:  
        return True  
    return False
```

L'algoritmo ha complessità $O(n^2)$

L'algoritmo ha complessità $O(n^2)$

Basta considerare il caso in cui l'albero è una catena di n nodi:

con $x=n-1$

$$P = \boxed{1 \ 2 \ 3 \ . \ . \ . \ n-2 \ n-1}$$



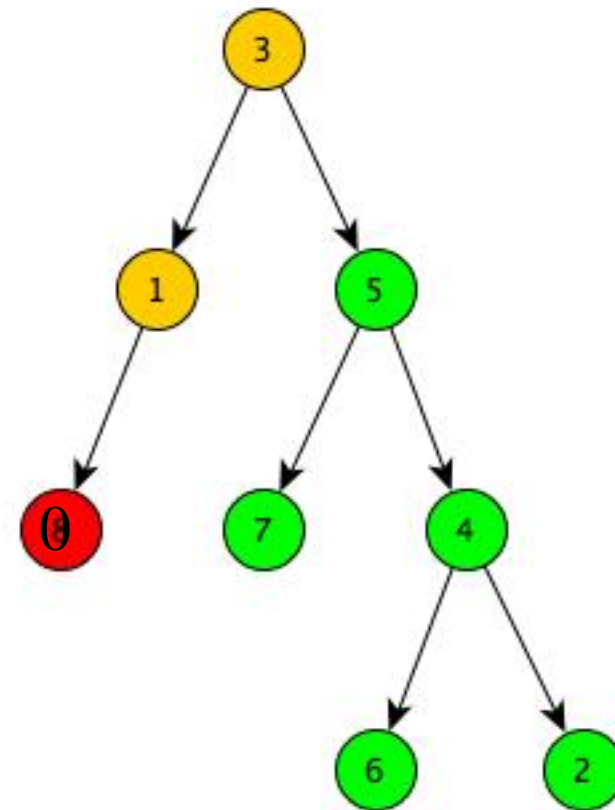
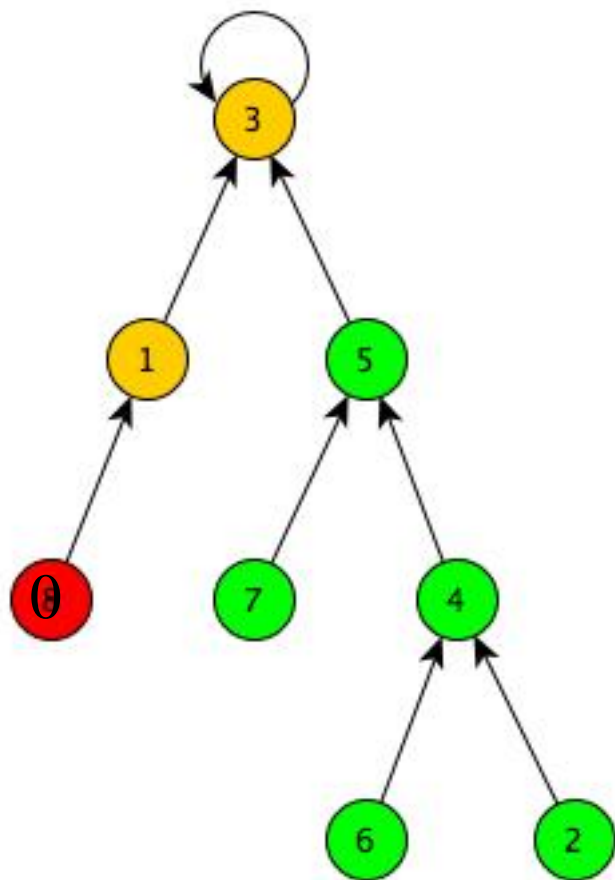
IDEA: il problema sarebbe facilmente risolvibile se il grafo avesse gli archi diretti dal padre ai figli (basterebbe una visita a partire da x)

Ad esempio per

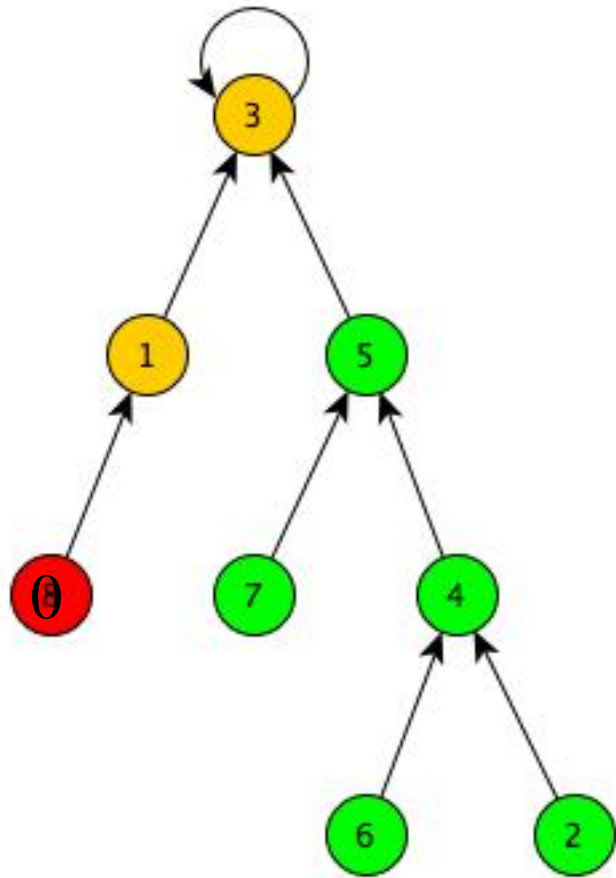
$P =$

1	3	4	3	5	3	4	5
---	---	---	---	---	---	---	---

e $x = 5$ la funzione deve restituire l'insieme $\{2,4,5,6,7\}$ mentre per $x = 0$ deve restituire $\{0\}$

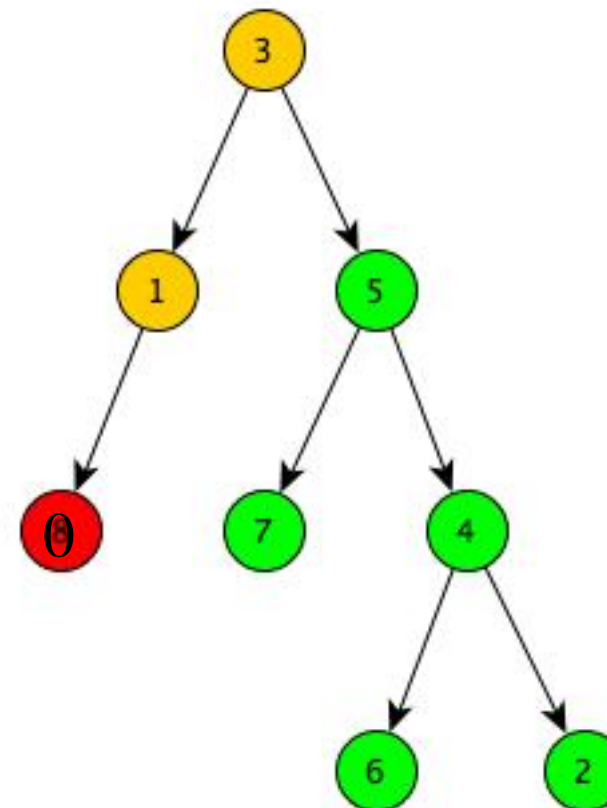


Dal vettore dei padri costruisco le liste di adiacenza



```
def inverti(P):  
    G={x:[] for x in range(len(P))}  
    for x in range(len(P)):  
        if P[x]!=x:  
            G[P[x]].append(x)  
    return G
```

0		→
1		→ 0
2		→
3		→ 1 5
4		→ 6 2
5		→ 7 4
6		→
7		→



```
def visita(x,G,sol):  
    sol.add(x)  
    for y in G[x]:  
        visita(y,G,sol)
```

```
def es(x,P):  
    G=inverti(P)  
    sol=set()  
    visita(x,G,sol)  
    return sol
```

```
def inverti(P):  
    G={x:[] for x in range(len(P))}  
    for x in range(len(P)):  
        if P[x]!=x:  
            G[P[x]].append(x)  
    return G
```

Complessità della procedura: $O(n)$