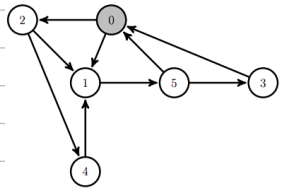
Esercizio 1. Dato un albero T di n nodi, rappresentato tramite il vettore dei padri P, dare lo pseudo-codice di un algortimo che produce in tempo $\mathcal{O}(n)$ la lista dei vertici di T_u, il sotto-albero radicado in un vertice u di T.

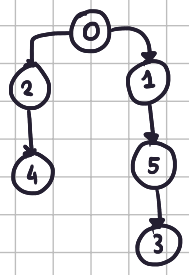
L'idea è quella di partire da un nodo e risalire verso la radice, appena trovo un nodo che fa parte del sotto-albero di u, tutti quelli dalla quale si è risaliti saranno anche essi nel sotto-albero. Se trovo un nodo che sicuramente non è nel sotto-albero, tutti quelli dalla quale si è risaliti non lo saranno. Alla fine tutti i nodi vengono visitati 2 volte. All'inizio, u è l'unico nel sottoalbero, e la radice l'unica che non vi è (a meno che la radice non sia proprio u, in tal caso la soluzione è costituita da tutti i nodi.)

```
SottoAlbero(P:array, u:nodo){
    n=P.length()
    Sol:list
    S:stack
    Sot[n]={0,0...,0}
    Sot[u]=1
    Sol.add(u)
    k=radice di P // si trova in O(n)
    if(k==u){return {0,1...n}}
    else{Sot[k]=-1}
    For(i=0,1...n){
        x=i
        while(Sot[x]==0){ // finche' trovo un elemento per cui so se e' o non e' in Tu
            S.push(x)
            x=P[x]
        }
        value=Sot[x] // Sot[x]==1 <=> nello stack ci sono elementi di Tu
        while(S≠∅){
            w=S.pop()
            Sot[w]=value
            if(value==1){Sol.add(w)}
        }
    }
    return Sol
}
```

Esercizio 2. Sia G il grafo in figura in cui le liste di adiacenza sono ordinate in senso crescente degli indici. Allora, determinare:

- l'albero di ricerca ottenuto in seguito ad una ricerca in profondità (DFS) di G con radice il vertice 0;
- specificare quali sono gli archi di attraversamento, in avanti e all'indietro in seguito a tale DFS.





Archi indietro:
(5,0), (3,0),

Archi attraversamento:
(2,1), (4,1)

Esercizio 4. Dato un intero n, sia c_n il numero di stringhe binarie lunghe n in cui non compaiono due zeri consecutivi. Fornire uno pseudo-codice che descrive un algoritmo, che dato $n \geq 1$, calcola il valore di c_n in tempo $\mathcal{O}(n)$. E se invece volessi calcolare d_n, cioè il numero di stringhe binarie lunghe n in cui non compaiono tre zeri consecutivi?

```
Es4Ric(A:array, n:int, Sol:int){ // versione non lineare
    K: A.length()
    iF(K ≥ n){ return 0 }
    B[K+1]: array
    For(i:0...K-1){ B[i]=A[i] }
    st:0
    iF(A[K-1]≠0){
        B[K]:0
        st: Es4Ric(B, n, Sol)+1
    }
    B[K]:1
    return st+ Es4Ric(B, n, Sol)+1
}
```

Esercizio 5. Dati due interi k e n, con $1 \leq k \leq n$, definiamo $P(k, n)$ come il numero di differenti partizioni dei numeri da 1 a n in k sotto-insiemi non vuoti. Fornire in pseudo-codice un algoritmo che calcola $P(k, n)$ in tempo $\mathcal{O}(k \cdot n)$.

Esercizio 6. Dato un intero $n \geq 2$, definiamo con x_n il minimo numero di operazioni con cui è possibile ottenere n partendo dal numero 2 e potendo effettuare le sole 3 operazioni di incremento di 1, prodotto per due e prodotto per tre; e con y_m il numero totale di modi (non per forza con un numero minimo di operazioni) per ottenere n con tali operazioni. Fornire un algoritmo che dato un intero n calcola sia x_n che y_n in tempo $\mathcal{O}(n)$.

```
Es6(n:intero) {
    Y[n+1]: array
    Y[2]=0
    X[n+1]: array
    X[2]=0
    For(i=3,4...,n) {
        TMP:Set
        TMP.add(X[i-1]+1)
        For(k=2,3){ if(i%k==0){ TMP.add(X[i/k]+1) }}
        X[i] = min(TMP)
        Y[i]= sum(TMP)
    }
    return   X[n], Y[n]
}
```