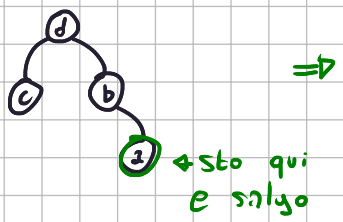
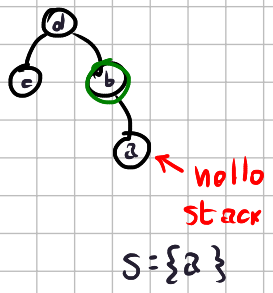
**Esercizio 1** (Stringhe binarie). *Data in input una stringa binaria di lunghezza $n$, progettare un algoritmo di complessità $O(n \log n)$ che restituisca il numero delle sue sottostringhe che cominciano con 0 e finiscono con 1.*

```
Sb(A: array){ //O(n)
    n2 = 0
    sol = 0
    For(i=0,1...,A.length()-1){
        if(A[i]==0){ n2 ++}
        else{sol+= n2}
    }
    return  Sol
}
```

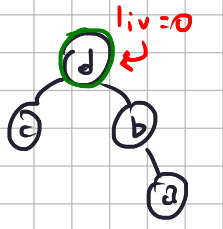
**Esercizio 2** (Nodi pari). *Sia $T$ un albero radicato nel nodo $x$ e rappresentato da un vettore dei padri $P$. Progettare un algoritmo di complessità $O(n)$ che dato in input il vettore $P$ restituisca il numero dei livelli di $T$ in cui compaiono un numero pari di nodi.*

```
LivPadri(P: array){
    n=P.length()
    S: Stack
    Liv[n] = {-1,-1,-1,... -1}
    Vis[n] = {0,0,0...,0}
    Vis[∞]=1
    Liv[∞]=0
    For(i=0...,n-1){ //O(n)
        cur = i
        level = 0
        while(P[cur]≠ cur){
            if(Vis[cur]≠1){
                Vis[cur]=1
                S.push(cur)
            }else{
                level = Liv[cur]
                break
            }
        }
        while(S≠∅){
            level ++
            Liv[S.pop()] = level
        }
    }
    m = max(Liv)
    D[m] = {0,0 ...,0}
    For(i=0,...n-1){
        D[i]++
    }
    livP = 0
    For(i=0,...m-1){
        if( D[i]≠0 ∧ D[i]%2 == 0){ livP++}
    }
    return  livP
}
```




S = {a}




S = {b, a}

Scorro lo stack incremen. tando il liv

| | | |
|---|---|---|
| liv = 0 | S = {b, a} | |
| liv = 1 | S = {a} | b |
| liv = 2 | S = { } | a |

**Esercizio 3** (Torneo). *Durante un girone di un torneo, Fabio si ritrova assieme ad $n$ sfidanti. Ciascuno sfidante $i$ richiede uno sforzo $s_i$ e fornisce $p_i$ punti. Per qualificarsi al girone successivo, Fabio deve ottenere un punteggio pari ad almeno $Q$. Inoltre, per raggiungere tale punteggio Fabio può scegliere un qualsiasi sottoinsieme degli $n$ sfidanti da affrontare.*

*Progettare un algoritmo che dati in input $Q, s_1, \ldots, s_n, p_1, \ldots, p_n$, restituisca un sottoinsieme di sfidanti che permetta a Fabio di qualificarsi con lo sforzo minimo possibile.*

num sFi → punti min da Fare

$T[k, \alpha]$ = costo minimo con valore $\geq \alpha$ con i primi $k$ sFidanti

$$T[1,\alpha] = \begin{cases} s_1 & \text{se } p_1 \geq \alpha \\ \infty & \text{altrimenti} \end{cases}$$

$T[k,1] = \min(\{s_i \mid i \leq k \mid p_i \geq 1\})$

```
Torneo(Q:intero, {s1...,sn}, {p1,...pn}) {
    T[n x Q]:matrice
    For(i=0...,Q){
        if(p1 ≥ i){ T[1,i]=p1 }
        else{ T[1,i]=∞ }
    }
    For(i=1...,n){
        For(J=0,1...,Q){
            THP={T[i-1,J]}
            if(pi ≥ J){ THP U {si} }
            if(pi<J){
                k= T[i-1,J-pi]+si
                if(k≥J){ THP U {k} }
            }
            T[i,J]=min(THP)
        }
    }
    return T[n,Q]
}
```

| punti | | 5 | 1 | 2 | 6 | 8 | 1 | |
|---|---|---|---|---|---|---|---|---|
| sforzo | | 3 | 5 | 1 | 4 | 7 | 2 | K |
| α | 1 | 3 | 3 | 1 | 1 | 1 | 1 | min |
| | 2 | 3 | 3 | 1 | 1 | 1 | 1 | |
| | 3 | 3 | 3 | 3 | 3 | 3 | 3 | |
| | 4 | 3 | 3 | 3 | 3 | 3 | 3 | |
| | 5 | 3 | 3 | 3 | 3 | 3 | 3 | |
| | 6 | ∞ | 8 | 4 | 4 | 4 | 4 | |
| | 7 | ∞ | ∞ | 4 | 4 | 4 | 4 | |
| | 8 | ∞ | ∞ | 9 | 5 | 5 | 5 | |

$T[k-1,\alpha]$ ←

se $p_i \geq \alpha$ $s_i$ ←

$T[k-1, \alpha - v_i] + s_i \geq \alpha$

$v_i < \alpha$

**Esercizio 4** (Fiaccola). *State percorrendo una strada da un punto $A$ ad un punto $B$ con una fiaccola, la quale resta accesa per $t$ minuti dopo essere stata ravvivata. Lungo il percorso, a partire dalla località $A$ sono presenti $n$ focolai in cui poter ravvivare completamente la fiaccola.*

*Siano $d_1, \ldots, d_n$ le distanze dei vari punti di sosta del tragitto, dove per $1 \leq i \leq n-1$ il valore $d_i$ rappresenta la distanza dal focolaio $i$ al focolaio $i+1$, mentre $d_n$ rappresenta la distanza dal focolaio $n$ alla località $B$. Siano inoltre $c_1, \ldots, c_n$ i costi da pagare per ravvivare la fiaccola in ognuno di tali focolai.*

*Assumendo che $d_1, \ldots, d_n \leq t$ e che alla località $A$ la fiaccola sia spenta (ricordiamo che il primo distributore è situato sulla località $A$), progettare un algoritmo di complessità $O(n)$ che restituisca il costo minimo per completare il viaggio senza che la fiaccola si spenga.*

```
Fiaccola({d1,d2...,dn}, {v1,v2...,vn}, t:reale){
    T[n] = {c1, c1, c1..., c1}
    For(i=2...,n){                          //Θ(n)
        tmp = t
        m = ∞
        For(J=i-1...,0){                    //O(t)
            tmp -= vJ
            m = min(m, cJ+T[J])
            if(tmp ≤ 0){
                T[i]=m
                break
            }
        }
    }
    return T[n-1]
}
```

**Funziona se le distanze sono ≥ 1, altrimenti è O(n^2)**