Esercizio 1 (Distanza massima). *Dato un grafo diretto G e un nodo $x \in V(G)$, progettare un algoritmo di complessità $O(n+m)$ che restituisca il numero dei nodi raggiungibili da x che si trovano alla massima distanza.*

```
BFS_count_max(G:grafo, x:nodo) {
    Dist[n] = {-1,-1...-1}
    Dist[x] = 0
    S: Queue
    S.push(x)
    while(S≠0) {
        y = S.top()
        For each  z ∈ y.adj {
            if( Dist[z]==-1) {
                Dist[z] = Dist[y]+1
                S.push(z)
            }
        }
        S.pop()
    }
    M = max(Dist)
    c = 0
    For(i=0...,n-1) {
        if(Dist[i]==M) { c++ }
    }
    return  c
}
```

Utilizzo un BFS per calcolare le distanze da x algi altri nodi, ciò ha costo O(n+m), per poi contare quanti nodi hanno distanza massima da x, ha costo O(n)

Esercizio 3 (Sottosequenze crescenti). *Data una sequenza di interi X, una sottosequenza Y di X corrisponde ad una sequenza ottenuta eliminando zero o più elementi da X. Inoltre, definiamo una sottosequenza come crescente se ogni elemento al suo interno è strettamente maggiore di tutti gli elementi della sottosequenza stessa ad esso precedenti.*

Progettare un algoritmo di complessità $O(n^2)$ che data in input una sequenza X restituisca una sottosequenza crescente di X di lunghezza massima. Ad esempio, per $X = [50, 2, 1002, 48, 3, 34, 30]$ una soluzione ottimale risulta essere $[\_, 2, \_, \_, 3, \_, 30]$

Bisogna considerare la programmazione dinamica

$T[i]$ = Sotto seq. massima di $X[0:i] \Rightarrow T[0] = X[0]$

$$T[1] = \begin{cases} X[0] & \text{se } X[0] \geq X[1] \\ X[0:1] & \text{altrimenti} \end{cases}$$

Supponiamo di conoscere $T[k]$, avremo che, se per qualche $i < k+1$, $X[i] < X[k+1] \Rightarrow T[k+1] = T[i] \cup \{X[k+1]\}$

```
max_subseq(X: array){
    T[n] = array lungo n di stringhe di interi
    T[0] = X[0]
    for(i=0,1...,n-1){
        mlen=0
        K:intero
        for(J=i-1,....0){
            if(X[J] < X[i]){
                if(T[J].length()>mlen){
                    K=J
                    mlen=T[J].length()
                }
            }
            T[i]=T[K] U{X[i]}
        }
    }
    return T[n-1]
}
```

Esercizio 4 (Minimi intervalli di lunghezza uno). *Dato un array A ordinato di numeri reali, progettare e dimostrare la correttezza di un algoritmo di complessità $O(n)$ che trovi un insieme di cardinalità minima di intervalli di lunghezza uno (dunque intervalli del tipo $[x, x+1]$) che contengano tutti gli elementi di A. Ad esempio, per $A = [1.1, 2, 2.05, 3, 4]$ una soluzione ottimale risulta essere $\{[1.1, 2.1], [3, 4]\}$.*

Ogni soluzione ottimale contiene l'intervallo $[A[0], A[0]+1]$

```
Min_unit_interval(A[0:n]: array di reali){
    SOL={ [A[0],A[0]+1] }
    for each i=1,2...,n-1 {
        if(A[i]> SOL.last()[1]) {
            SOL.add( [A[i],A[i]+1)
        }
    }
    return SOL
}
```

Prop: sia Sol^K il valore di Sol al K-esimo passo dell'algoritmo, $\exists\ Sol^*$ ottimale tale che $Sol^K \subseteq Sol^*$

Dim: si dimostra per induzione su K

- $K=0$: $Sol^K = \emptyset \subseteq Sol^*$
- si assume che $Sol^K \subseteq Sol^*$
- Consideriamo Sol^{K+1}

$Sol^{K+1} = Sol^K \cup \{[i, i+1]\}$. $[i, i+1]$ non sono in Sol^*, quindi