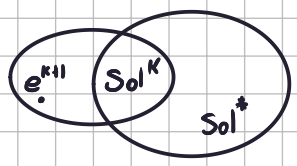
Esercizio 1. (max 7) Dare il pseudocodice di un algoritmo che accetta in input un grafo G non diretto con pesi sugli archi e restituisce un albero di copertura di peso massima. Discutere la correttezza dell'algoritmo e la complessita'.

```
Max_spanning_tree(G=(V,E): grafo){
    Sol:set
    Es: sort(E(G), ω)  //Es contiene gli archi ordinati decr in base al peso Θ(m log(m))
    Vis[n]={0,0,...0}
    For each (u,v) ∈ Es { Θ(m)
        if(Vis[u]·Vis[v]==0){ // collega un nodo non ancora connesso
            Sol.add((u,v))
            Vis[u]=1
            Vis[v]=1
        }
    }
    return Sol
}
```

il costo dell'algoritmo e' $O(m \cdot \log(m))$

Dimostrazione correttezza: Sia Sol^K il valore di Sol al K-esimo passo. Sia Sol^* una qualunque soluzione ottimale, dimostro che $\forall K \; Sol^K \subseteq Sol^*$. Procedo per induzione:

- caso base: $Sol^0 = \emptyset \subseteq Sol^*$
- ipotesi induttiva: $Sol^K \subseteq Sol^*$
- passo induttivo: si ha $Sol^{K+1} = Sol^K \cup \{e_{K+1}\}$, sappiamo che $e_{K+1} \notin Sol^*$



Se aggiungessimo e_{K+1} a Sol^*, avremmo un ciclo $C=\{w_1, w_2 \dots, w_i, e_{K+1}\}$, in tale ciclo vi e' un nodo NON considerato in Sol^K, sia questo w, per costruzione $\omega(w) \leq \omega(e_{K+1})$, quindi, $T = \{Sol^* \setminus \{w\}\} \cup \{e_{K-1}\}$ NON ha cicli ed e' un albero di copertura, dato !, si ha che $\omega(T) \geq \omega(Sol^*)$, ma Sol^* era massimale $\Rightarrow$ T e' massimale e contiene Sol^{K+1}. ■

Dato un array A ordinato di numeri reali (quindi $A[1] \leq A[2] \leq \cdots \leq A[n]$),
dare lo pseudocodice per un algoritmo per trovare un insieme di cardinalità minima di intervalli di lunghezza uno (quindi intervalli di tipo $[x, x+1]$) che contengono tutti gli elementi $A[i]$.
Per esempio, dato $A = [1.1, 2, 2.05, 3, 4]$, una soluzione sarebbe $\{\,[1.1, 2.1]\,,\ [3, 4]\,\}$.
La complessità dell'algoritmo dovrebbe essere $O(n)$.

```
Intervalli_unitari (A : array) {
    Sol : list
    Sol.add([A[0], A[0]+1])
    For (i=1,2..., n-1) { // O(n)
        if (A[i] > Sol.last()[1]) { Sol.add([A[i], A[i]+1]) }
    }
    return Sol
}
```

Exercise 4 (max 8) Un cavallo bianco è posizionata sulla casella (x_1, y_1) di una scacchiera n×n e mediante una sequenza di mosse deve raggiungere il re nero sulla casella (x_2, y_2). Un cavallo posizionata sulla generica casella (i, j) ha al più otto mosse possibili: ad (i+1, j+2), (i+1, j-2), (i-1, j+2), (i-1, j-2) oppure ad (i+2, j+1), (i+2, j-1), (i-2, j+1), (i-2, j-1). Descrivere un algoritmo per determinare se esiste una seria di mosse da posizione (x_1, y_1) a (x_2, y_2) e se esiste un tale seria, trova una con il numero minimo di mosse.

```
cavallo( n:intero, (x1,y1): Interi , (x2,y2):Interi){
    Q: coda
    M[n,n] : matrice  inizializzata  a -1
    M[x2][y2] = 0
    Q.push( (x2,y2))
    do {
        (u,v) = Q.top()
        For each (i,j) ∈ mosse( (u,v), n) {
            if(M[i][j] == -1){
                M[i][j] = M[u][v]+1
                Q.push( (i,j))
            }
        }
        Q.pop()
    }
    Sol = M[x1][y1]
    if (Sol == -1){return ∞}
    return Sol
}
```

```
mosse( (u,v):Interi , n:intero){
    moves = {(1,2), (1,-2), (-1,2), (-1,-2),
             (2,1), (2,-1), (-2,1), (-2,-1)}
    S = { }
    For (i,j) ∈ moves {
        pos = (u+i, v+j)
        if (pos[0] < n ∧ pos[1] < n){
            S.add(pos)
        }
    }
    return pos
}
```

Dare lo pseudocodice di un algoritmo che prenda in input un grafo diretto G, due vertici u,v, e un intero k e restituisca il numero di passeggiate distinte di lunghezza al massimo k da u a v.
L'algoritmo dovrebbe avere complessità $O(n \cdot m \cdot k)$

deFinisco $T[\alpha, x]$ = num di passeggiate lungo α da u a x.

$$T[\alpha, x] = \sum_{(y,x) \in E(G)} T[\alpha-1, y]$$

e' chiavo che $T[1, x] = \begin{cases} 1 & \text{se } \exists\ (u,x) \\ 0 & \text{altrimenti} \end{cases}$

```
Passeggiate(k:int, G:grafo, u:nodo, v:nodo){
    n=|V(G)|
    M[k+1][n]:matrice
    For(i=0...,n-1){
        if(∃ (u,i)∈E(G)){M[1][i]:1}
    }
    For(i=2...k){
        For(j=0...,n-1){
            sum=0
            For each x | ∃ (x,j)∈E(G){
                sum+=M[i-1,x]
            }
            M[i,j]= sum
        }
    }
    return Σ_{i=1}^{k} M[i][v]
}
```