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'.

```
MaxSpTree(G:grafo){
    sort(E(G), ω) // ordina in base al peso decrescente   O(m·log(m))
    T:tree
    V(T)=V(G)
    For each e∈E(G){ //O(m)
        iF(T∪{e} non ha cicli){ // controllo in O(n+m)
            T=T∪{e}
        }
    }
    return T
}
```

Il costo Finale e' $O(m(m+n))$.

Prop: sia Sol^K il valore di T al k-esimo passo e Sol^* una qualsiasi soluzione ottimale, si ha che $Sol^K \subseteq Sol^*$.

Dim: per induzione su K

- c.b. $K=0 \Rightarrow Sol^K = \emptyset \subseteq Sol^*$
- i.i. $Sol^K \subseteq Sol^*$
- p.i ho che $Sol^{K+1} = \begin{cases} Sol^K \cup \{e_K\} \\ Sol^K \text{ // banale} \end{cases} \Rightarrow Sol^{K+1} = Sol^K \cup \{e_K\}$, e_K non e' in $Sol^* \Rightarrow Sol^* \cup \{e_K\}$ ha un ciclo C, per costruzione $\exists u \in C \mid \omega(u) \leq \omega(e_K) \Rightarrow SN = \{Sol^* \cup \{e_K\}\} \setminus \{u\}$ non ha cicli e:

$$\omega(SN) \leq \omega(Sol^*) \Rightarrow SN \text{ e' ottimale e } Sol^{K+1} \subseteq SN. \quad \blacksquare$$

Esercizio 2. (max 7) Sia A[1,...,n] un array di n interi con n dispari. Sia A ordinato e ogni valore occore esattamente due volte trane uno. Dare il pseudocodice per un algoritmo O(log n) che prenda in input l'array A e trova il valore presenta solo una volta. Per esempio, dato il input: A = [2, 2, 5, 5, 7, 9, 9, 10, 10], l'algoritmo restuisce "7". Dato il input: A = [1, 1, 4, 4, 6, 6, 8, 8, 9], l'algoritmo retuisce "9".

```
es2(A: array){
    i = ⌊A.length()/2⌋
    iF(A[i] ≠ A[i+1] ∧ A[i] ≠ A[i-1]){ return A[i]}
    iF ((i%2 == 0 ∧ A[i] == A[i-1]) V (i%2 ≠ 0 ∧ A[i] ≠ A[i-1]) {
        return es2(A[0:i-1])
    }
    return es2(A[i+1,n-1])
}
```

Exercise 3 (max 8) Una pedina è posizionata sulla casella (1, 1) in alto a sinistra di una scacchiera n×n e mediante una sequenza di mosse tra caselle adiacenti deve raggiungere la casella (n, n) in basso a destra. Una pedina posizionata sulla generica casella (i, j) ha al più due mosse possibili: spostarsi verso il basso nella casella (i + 1, j), posto che i < n, o spostarsi verso destra nella casella (i, j + 1), posto che j < n. La sequenza di caselle toccate determina un cammino. Ogni casella della scacchiera ha un colore c(i, j) che è o rosso o verde. Descrivere un algoritmo che in tempo $O(n^2)$ calcola il numero di cammini che passano solamente per caselle rosse e vanno da (1, 1) a (n, n).

Definisco T[i,j] = numero di cammini che passano solo per il rosso da (1,1) a (i,j).

$$T[1,j] = \begin{cases} T[1,j-1] \text{ se } c(1,2) = r \\ 0 \text{ se } c(1,2) = g \end{cases} \qquad T[j,1] = \begin{cases} T[j-1,1] \text{ se } c(2,1) = r \\ 0 \text{ se } c(2,1) = g \end{cases} \qquad T[1,1] = 1$$

$$T[i,j] = T[i,j-1] + T[j,i-1]$$

```
RB_Path(c: Funzione colore, n: intero){
    T[n x n] = matrice n×n
    T[0,0] = 1
    For(i=1...,n-1){
        iF(c(0,i) == R){ T[0,i] = T[0,i-1]}
        else{ T[0,i] = 0}
    }
    For(i=1...,n-1){
        iF(c(i,0) == R){ T[i,0] = T[i-1,0]}
        else{ T[i,0] = 0}
    }
    For(i=2...,n-1){
        For(j=2...,n-1){
            T[i,j] = T[i-1,j] + T[i,j-1]
        }
    }
    return T[n-1,n-1]
}
```

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[x2][y2]
    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
}
```