Esercizio 1. (max 6) G un grafo non-diretto e conesso. Dati due vertici v_1 e v_2, la distanza da v_1 a v_2, scritto dist(v_1, v_2), e' la lunghezza minima di un cammino da v_1 a v_2. Invece, dati due sottoinsiemi di vertici X e Y, la distanza da X a Y e

$$\min_{\{x \in X,\, y \in Y\}} \text{dist}(x, y)$$

Si osservi che nel caso in cui $X \cap Y \neq \emptyset$, diciamo che dist(X,Y) = 0.

Dare lo pseudocodice di un algoritmo O(|V| + |E|) per trovare dist(X,Y) dati G, X,Y in input.

```
BFS_set (G:grafo, X:nodi, Y:nodi){
    Q:queue
    Dist[n]={-1,-1...,-1}
    for each x∈X{
        Q.push(x)
        Dist[x]=0
    }
    while(Q≠∅){
        u=Q.pop()
        For each v∈u{
            iF(Dist[v]==-1){
                Dist[v]=Dist[u]+1
            }
        }
    }
    M = ∞
    for each y∈Y{
        M=min(M,Dist[y])
    }
    return M
}
```

Esercizio 2. (max 9) Dare lo pseudocodice per risolve il seguente problema: dati n oggetti $x_1, x_2, \ldots, x_n$ ognuno di un costo c_i e un valore v_i, $1 \le i \le n$, e un vincolo A, trovare il sottoinsieme di $x_1, x_2, \ldots, x_n$ di costo minimo con valore totale al minimo A.

Definisco $T[k, \alpha]$ = somma dei costi minima dei primi k oggetti che sia superiore o uguale ad α.

$$T[1,\alpha] = \begin{cases} c_1 & \text{se } v_1 \ge \alpha \\ \infty & \text{altrimenti} \end{cases}$$

$$T[k,\alpha] = \min\left(T[k-1,\alpha],\ T[k-1,\alpha - v_k] + c_k,\ c_k\right) \quad \uparrow \text{se } v_k \ge \alpha$$

$$T[k,1] = \min\left(\bigcup_{i=1}^{k} c_i\right) \text{ se } v_i \ge 1$$

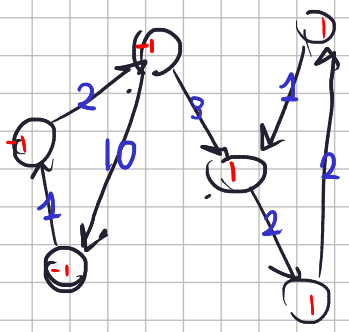
```
Es2(A:intero, O:{(x_i,c_i,v_i)}){
    n=O.length()
    T[n+1 x A+1]: matrice inizializzata a ∞
    For(i=1,2...,A){
        T[1,i]=∞
        iF(v_1 ≥ i){ T[1,i]=c_1 }
    }
    For(i=2,3...,n){
        For(J=1,2...,A){
            m=T[i-1,J]
            iF(v_i<J){ m=min(m,T[i-1,J-v_i]+c_i) }
            iF(v_i≥J){ m=min(m,c_i) }
            T[i,J]=m
        }
    }
    Sol={} // a questo punto, ho la matrice, devo estrapolare gli oggetti x_i
    J=A
    For(i=n...,1){
        iF(T[i,J]≠T[i-1,J]){
            J-=v_i
            Sol.add(x_i)
        }
    }
    return Sol
}
```

v	0	5	1	3	4
c	0	5	2	1	3
0	0	0	0	0	0
1	∞	5	2	2	2
2	∞	5	5	2	2
3	∞	5	5	2	2
4	∞	5	5	4	3

Esercizio 3. (max 7) Dato un grafo diretto G con pesi p sugli archi, il peso di un ciclo e' la somma dei pesi degli archi. Dare lo pseudocodice di un algoritmo che prenda in input un grafo diretto con pesi e trovi il ciclo di peso minimo. Se il grafo non ha cicli, l'algoritmo dovrebbe restituire "INFINITY". L'algoritmo dovrebbe avere complessita' O(m(n+m)logn).

```
DFS_cerca_cicli_min(
    S:Stack
    C={ }
    mC = ∞
    Vis[n] = {0,0...,0}
    x = V(G)[0] // un nodo a caso del grafo
    Vis[x] = -1
    S.push(x)
    while(S≠∅){ O(n)
        v = S.top()
        if(∃ w | w∈v ∧ Vis[v] ≠ 1){ O(m)
            if(Vis[w]==0){
                Vis[w] = -1 //visitato ma è nello stack (soggetto a cicli)
                S.push(w)
            }
            if(Vis[w]== -1) { //ciclo
                tmpC, pC = PesoCiclo(G, S, w)  O(n)
                if( pC < mC){
                    mc = pC
                    C = tmpC
                }
                E(G).remove( (v, w))  O(m)
            }
        }else{
            S.pop()
            Vis[v] = 1 // rimosso dallo stack
        }
    }
    return C
}

PesoCiclo(G:grafo, S:Stack, w:nodo){
    SS = S.reverse()
    prec = w
    p = 0
    C = { }
    for each u∈S {
        C = C ∪ {(u, prec)}
        p += w(u, prec)
        if(u == w){ break }
        prec = u
    }
    return C, p
}
```



n

5

Esercizio 4. (max 8) A un array di interi di lunghezza n. Un *inversione* in A e' un paio di indici (i, j) dove i < j perro A[i] > A[j]. Per esempio, con A = [6, 5, 4, 8], (2,3) e' un inversione perche A[2] = 5 > A[3] = 4. Dare lo pseudocodice per un algoritmo per contare il numero di inversioni in un array dato in input. Nel essempio, l'algoritmo restituisce 3 (gli inversioni sono (1,2), (1,3), e (2,3)). L'algoritmo dovrebbe avere complessita' O(n (log n)^2).

```
Inv_non_ottimizzato( A: array){  // O(n^2)
    n = A.length()
    I = { }
    for(i=0..., n-1){
        For(j=i+1,... n-1){
            if(A[j]<A[i]){ I.add(i,j)}
        }
    }
    return  I
}
```