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.

```
Es2({x1,x2...xn}, A){
    O:{ }
    sumV=0
    sumC=0
    For(i=1,2...n){
        if(sum<A){
            O.add(xi)
            sumV+=vi
            sumC+=ci
        }else{
            toSub=-1
            minC=sumC
            For(j=1.., i-1){
                if(sumV-vj+vi≥A){
                    if(minC-cj+ci< minC){
                        minC=minC-cj+ci
                        toSub=j
                    }
                }
            }
            if(toSub≠-1){
                SumV+=(vi-vtoSub)
                SumC+=(ci-vtoSub)
                O.remove(xtoSub)
                O.add(xi)
            }
        }
    }
}
```

È un problema di programmazione dinamica, definisco la matrice $T: n \times A$ tale che:

$T[K, \alpha]$ = sottoinsieme di $\{x_1, x_2 \dots, x_K\}$ di costo minimo e valore $\sum_{i=1}^{K} v_i \geq \alpha$.

Casi banali: $T[0, \alpha]$ = NON DEFINITO $T[K, 0] = \{\}$ // insieme vuoto

$T[1, \alpha] = \{x_1\}$ se $v_1 \geq \alpha$. Ovviamente $0 \leq K \leq n \wedge 0 \leq \alpha \leq A$

! Si è nell'ipotesi che i costi c_i, i valori v_i, ed il valore A siano numeri naturali.

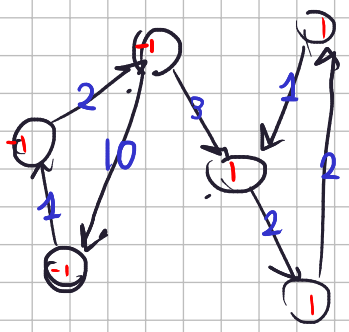
	0	1	2	...	v_1	...	A
0	$\emptyset$			...			
1	$\emptyset$	$\{x_1\}$	$\{x_1\}$	...	$\{x_1\}$	...	$\emptyset$
2	$\emptyset$						
⋮							
n	$\emptyset$						

Supponiamo di conoscere $T[K', \alpha']$ $\forall K' < K$ $\forall \alpha' < \alpha$ e vogliamo calcolare $T[K, \alpha]$. Se la somma dei valori di $T[K, \alpha-1]$ è maggiore di α, allora $T[K, \alpha] = T[K, \alpha-1]$. Inoltre $\forall K' > K$ $T[K', \alpha] = T[K, \alpha]$. Se la somma dei valori in $T[K, \alpha-1]$ è $\alpha-1$, allora sarà necessario aggiungere un oggetto in $T[K, \alpha]$.

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
}
```