1) Scrivere un algoritmo che, dato un grafo G = (V, E) e un vertice s ∈ V , conta i vertici raggiungibili da s (escluso s stesso).

Noi sappiamo che, l'output uno degli output del BFS rappresenta il peso per far raggiungere un arco u ad un arco v.
Se questo valore è infinito, allora vuol dire che non si può trovare

Scriviamo il BFS:

```
BFS(G, s):
    // Inizializzo i dati
    For u ∈ V − {s} do
        Color[u] = W
        D[u] = ∞
        Pi[u] = NIL
    Color[s] = G
    D[s] = 0
    Pi[s] = NIl

    Q = {}
    ENQUEUE(Q, s)
    While Q ≠ {} do
        U = head(Q)
        For ogni v adiacenti a u do:
            If color[v] = W then
                Pi[v] = u
                D[v] = d[u] + 1
                Dolor[v] = G
                ENQUEUE(Q, v)

        Dequeue(Q)
        Color[u] = B
```

Scriviamo la funzione che conta:

```
BFS-CONTA(G, s)
    BFS(G, s)
    N = 0

    For ogni u ∈ V − {s} do
        If d[u] ≠ ∞ then
            N = n+1

    Return n
```

2) Scrivere un algoritmo che, dati un grafo G = (V, E) non orientato ed un vertice v ∈ V , stampa l'elenco dei vertici che hanno distanza pari da v (e quindi stampa anche v).

Facciamo la stessa cosa di prima
Scrivo la BFS(mi devo esercitare!)

```
BFS(G, s):
    For v ∈ V − {s} do
        D[v]=∞
        Pi[v]=NIL
        Color[v]=W
    Color[s]=G
    D[s]=0
    Pi[s]=NIL

    S={}
    ENQUEUE(S, u)
    While Q!={} do:
        U = HEAD(Q)
        For v intorno a u do:
            If color[v] = W then
                Color[v] = G
                Pi[v] = u
                D[v] = d[u] + 1
                ENQUEUE(Q, v)

        Color[u] = B
        DEQUEUE(Q, u)
```

Scrivo la funzione richiesta:

```
BFS-PARI(G, s):
    BFS(G, s)

    Foreach v ∈ V do
        If d[v] != ∞ and d[v] % 2 == 0 then
            Printina(v)
```

3) Scrivere un algoritmo che, dati un grafo G = (V, E), un vertice s ∈ V e un intero k > 0, effettua la visita in ampiezza di G sino a scoprire i vertici che distano k da s. Si richiede che al termine dell'esecuzione dell'algoritmo i vertici v con d[v] ≤ k abbiano colore nero, mentre gli altri vertici abbiano colore bianco.

Qui dobbiamo modificare il nostro BFS

```
BFS(G, s, k):
    Foreach v ∈ V − {s} do
        D[v] = ∞
        Pi[v] = NIL
        Color[v] = W
    D[s]=0
    Pi[s]=NIL
    Color[s] = G

    Q = {}
    ENQUEUE(Q, s)
    While Q ≠ {} do
        U = HEAD(Q)
        If d[v] < k then:
            Foreach v intorno a u do:
                If Color[v] = W then:
                    D[v] = d[u] + 1
                    Pi[v] = u
                    Color[v] = G
                    ENQUEUE(Q, v)


        Color[u] = B
        DEQUEUE(Q, u)
```

4) Scrivere un algoritmo che stabilisce se un grafo G = (V, E) non orientato `e connesso.
Possiamo usare l'algoritmo di prima BFS-CONTA

```
BFS-CONNESSO(G, s):
    N = BFS-CONTA(G, s)
    If n == |V|-1 then // Sarà vero solamente se non abbiamo mai avuto infiniti
        Return true
    Return false
```

5) Scrivere un algoritmo che stabilisce se un grafo G = (V, E) non orientato `e un albero.
Un albero è un grafo aciclico connesso
Abbiamo già la funzione per il connesso, dobbiamo comprendere se è aciclico

```
BFS-ALBERO(G):
    Connesso = BFS-CONNESSO(G, V[0])
    If connesso then
        // Per essere un albero, |E|=|V|-1
        If |E| = |V|-1 then
            Return true
    Return false
```

6) Modificare l'algoritmo DFS di visita di un grafo orientato G in maniera tale che stampi ogni arco di G specificandone il tipo

```
DFS(G):
    For v ∈ V do
        Color[v] = W
        Pi[v] = NIL
    Time = 0

    For v ∈ V do
        If color[v] = W then
            DFS-VISIT(G, u)

DFS-VISIT(G, u):
    Color[u] = G
    D[u] = time
    Time = time + 1

    For v intorno a u do
        If color[v] = W then
            (u, v) = "Arco T"
            Printina (u, v) "arco albero"
            Pi[v] = u
            DFS-VISIT(G, v)
        Else if color[v] = G then
            Printina (u, v) "arco indietro"
        Else if d[u] < d[v] // Quindi qui l'arco è nero
            Printina (u, v) "arco avanti"
        Else
            Printina (u, v) "arco trasversale"

    Time = time + 1
    F[u] = time
    Color[u] = B
```

7) Modificare l'algoritmo DFS di visita di un grafo non orientato G in maniera tale che stampi ogni arco di G specificandone il tipo.

E' estremamente simile a prima

```
DFS(G):
    For v ∈ V do
        Color[v] = W
        Pi[v] = NIL
    Time = 0

    For v ∈ V do
        If color[v] = W then
            DFS-VISIT(G, v)

DFS-VISIT(G, u):
    D[u] = time
    Time = time+1
    Color[u] = B

    For v intorno a u - {p[u]} do:
        If color[v] = W:
            Printina (u, v) "arco d'albero"
            Pi[v] = u
            DFS-VISIT(G, v)
        Else:
            Printina (u, v) "arco indietro"

    Time = time+1
    F[u] = time
```

8) Modificare l'algoritmo DFS di visita di un grafo orientato G in maniera tale che stabilisca se G `e acicilico, ossia se non contiene cicli.

Basta cotrollare che non ci siano archi all'indietro

```
DFS-ACICLICO(G)
    For v ∈ V do
        Color[u] = W
        Pi[u] = NIl
    Time = 0

    For v ∈ V do
        If color[v] = W then
            If not DFS-ACICLICO-VISIT(G, v) then
                Return false

    Return true

DFS_ACICLICO-VISIT(G, u):
    Color[u] = G
    D[u] = time
    Time = time + 1

    For ogni u intorno a v do:
        If color[v] = W then
            Pi[v] = u
            If not DFS-ACICLICO-VISIT(G, v) then
                Return false
        Else if color[v] = G then
            Return false


    F[u] = time
    Color[u] = B
    Time = time+1
    Return true
```

9) Modificare l'algoritmo DFS di visita di un grafo orientato G in maniera tale che stabilisca se G `e acicilico, ossia se non contiene cicli.

E' lo stesso di sopra.

10) Scrivere un algoritmo che determina il numero di componenti connesse di un grafo G = (V, E) non orientato.

Mi sto iniziando a stancare AHAHAHA

```
DFS-Componenti-Connesse(G)
    For ogni u ∈ V do
        Color[u] = W
        Pi[u]  = NIL
    Time = 0
    N = 0

    For u ∈ V do
        If color[u] = W then
            N = n+1
            DFS-VISIT(u)
```

11) -Non ho voglia **[Potrei volere farli domani o sta notte]**
11) Scrivere un algoritmo che, dato un grafo G = (V, E) non orientato, permetta di indivi- duare la componente connessa di cui ogni vertice u ∈ V fa parte.

```
DFS-Localizzazione(G):
    For ogni u ∈ V do
        Color[u] = W
        Pi[u] = NIL
    Time = 0
    N = 0

    For u ∈ V do
        If color[u] = W then
            DFS-VISIT(u, n)
            N = n+1

DFS-VISIT(u, n):
    D[u] = time
    Time = time+1
    Color[u] = G
    Cc[u] = n // Un array dove, a posizione u dico che quel vertice è nella n c.c.

    For ogni nodo v intorno a u
        If color[u] = W then
            Pi[v] = u
            DFS-VISIT(v, n)

    F[u] = time
    Time = time+1
    Color[u] = B
```

12) E' l'esercizio numero 10 fuso all'esercizio numero 9 **[Nel caso vedrete le soluzioni sotto]**
12) Scrivere un algoritmo che, dati un grafo G = (V,E) non orientato e un intero k > 0, stabilisca se G `e una foresta composta da k alberi.

Un grafo è una foresta se tutte le sue componenti connesse sono alberi
Ed un albero è un grafo connesso aciclico, e noi sappiamo se è aciclico se |e|=|v|-1

```
DFS(G, k):
    For u ∈ V do
        Pi[u] = NIL
        Color[u] = W
    Time = 0
    N = 0

    For u ∈ V do
        If color[u] = W then
            Nv = 0
            Na = 0
            DFS-VISIT(G, u)
            If Na = 0 then
                N = n+1
            Else if na = nv - 1 then
                N=n+1
            Else then
                Return false

    If n == k then
        Return true
    Else
        Return false

DFS-VISIT(G, u):
    Nv = nv+1
    D[u] = time
    Time = time+1
    Color[u] = g

    Foreach v intorno a u do
        Na = na+1
        If color[v] = W then
            Pi[v] = u
            DFS-VISIT(G, v)

    F[u] = time
    Time = time+1
    Color[u] = b
```

13) Questo è ancora più semplice degli altri
13) Scrivere un algoritmo che, dato un grafo G = (V, E) non orientato, calcola il numero di componenti connesse che contengono un numero pari di vertici.

E' praticamente come prima

```
DFS(G):
    For u ∈ V do
        Pi[u] = NIL
        Color[u] = W
    Time = 0
    N = 0

    For u ∈ V do
        If color[u] = W
            Nv = 0
            DFS-VISIT(G, u)
            If nv % 2 == 0 then
                N = n+1

    Return n
```

14) Lo abbiamo già fatto nell'esercizio 13
14) Scrivere un algoritmo che, dato un grafo G = (V, E) non orientato, calcola il numero di vertici di ogni componenti connessa.

Come prima

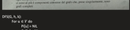
```
DFS(G):
    For u ∈ V do
        Pi[u] = NIL
        Color[u] = W
    Time = 0
    N = 0

    For u ∈ V do
        If color[u] = W
            Nv = 0
            DFS-VISIT(G, u)
            Cc[n] = nv // Io la salvo anche
```

15) Non ho voglia
15) Scrivere un algoritmo che, dato un grafo G = (V, E) non orientato, calcola il numero di archi di ogni componenti connessa.

Stessa cosa di prima ma anzichè nv abbiamo na e poi na = na/2

16)



Scrivere un algoritmo che, dati un grafo non orientato e due interi positivi h>0 e k>0, stabilisce se ENTRAMBE le seguenti condizioni sono verificate:

- ogni componente connessa del grafo ha al massimo h archi
- ci sono al più k componenti connesse del grafo che, prese singolarmente, sono grafi completi

```
DFS(G, h, k):
    For u ∈ V do
        Pi[u] = NIL
        Col[u] = W
    Time = 0
    J = 0 // Numero componenti connesse

    For u ∈ V do
        If col[u] = W then
            Na = 0
            Nv = 0
            DFS-VISIT(u)
            If na = 0 then
                J = 0
            Else if na = nv*(nv-1)/2 then
                J = j+1
                If j > k then
                    Return false
            Else if na > h then
                Return false

DFS-VISIT(u)
    D[u] = time
    Time = time+1
    Color[u] = G
    Nv = nv+1

    For ogni vertice v adiacente a u then
        Na = na+1
        If color[v] = W then
            Pi[v] = u
            DFS-VISIT(v)


    F[u] = time
    Time = time+1
    COlor[u] = B
```

17)

ESERCIZIO 2 Scrivere un algoritmo che, dati un grafo non orientato, due suoi vertici x e y e un intero positivi k>0, stabilisce se ENTRAMBE le seguenti condizioni sono verificate:

- il grafo è una foresta con esattamente k componenti connesse
- x e y appartengono a due componenti connesse distinte

```
DFS(G, x, y, h)
    For u ∈ V then
        Col[u] = W
        Pi[u] = NIl
    Time = 0
    J = 0
    For u in V then
        If col[u] = W then
            Na = 0
            Nv = 0
            Bol1 = false
            Bol2 = false
            J = j+1
            DFS-VISIT(u)
            Na = na/2
            If bol1 ^ bol2 then // x, y appartengono a 2 componenti connesse distinte
                Return false
            If na != 0 ^ nv != Na-1
                Return false
    If j == h then
        Return true
    Return false

DFS-VISIT(u)
    Col[u] = g
    D[u] = time
    Time = time+1
    Nv = nv+1

    For v adiacente a u then
        Na = na+1
        If col[v] = W then
            Pi[v] = u
            DFS-VISIT(v)
            If v == x then
                Bol1 = true
            If v == y then
                Bol2 = true


    F[u] = time
    Time = time+1
    Col[u] = n
```