Esercitazione sull'uso di `networkx`

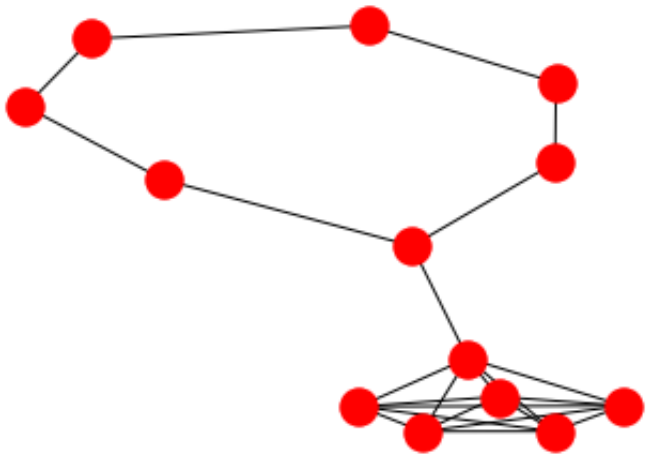
Questa esercitazione guidata riguarda l'uso della libreria Pyhton `networkx`. Per iniziare, suggeriamo di creare un ambiente isolato che contenga le librerie che servono, ad esempio con:

```
mkdir pippo      # La directory in cui si intende lavorare
cd pippo
pipenv install   # Crea l'ambiente virtuale (solo la prima volta)
pip shell        # Entra nell'ambiente virtuale
pip install jupyter networkx matplotlib   # Installa le librerie
jupyter notebook # Avvia jupyter
....
exit             # Esce dall'ambiente virtuale
```

Se necessario, consultate la documentazione di networkx.

1. Creare e analizzare un grafo non orientato

Dati `k>1` e `p>1`, considerate il grafo non orientato in figura:



Esso è costituito da

- un insieme di `k` vertici tutti collegati a due a due (una cosiddetta *clique* or *cricca*)
- un insieme di `p` vertici collegati a formare un ciclo (a volte questo grafo viene chiamato anche *anello*)
- uno (qualsiasi) dei vertici della clique è collegato inoltre a uno (qualsiasi) dei vertici dell'anello.

Chiamiamo `G(k,p)` il grafo appena descritto.

1. Scrivete una funzione Python che, dati `k` e `p` restituisca un oggetto `networkx` che rappresenti il grafo `G(k,p)`.

2. Disegnate un grafo `G(k,p)` (scegliete voi quali valori usare per i parametri).
3. Rispondete alle seguenti domande (usando carta e penna):

   3.1. Quanti vertici ha `G(k,p)` ?

   3.2. Quanti lati ha `G(k,p)` ?

   3.3. Il grafo è ciclico o aciclico?

   3.4. Oltre al ciclo di `p` vertici descritto, ci sono altri cicli semplici?

   3.5. Quanto è lungo e come è fatto il più lungo ciclo semplice nel grafo (la risposta potrebbe dipendere da `k` e `p`)? E il più lungo cammino semplice?

   3.6. Raggruppate i vertici del grafo a seconda del loro grado: quanti gruppi esistono? quali sono i gradi e quanti sono i vertici di ciascun grado?

   3.7. Esistono delle scelte di `(k,p)` per cui nel grafo ci sono solo due gradi? Quali e quante sono queste scelte?

4. Per ciascuna delle domande precedenti, tentate di rispondere usando le funzioni della libreria `networkx` . In qualche caso, ovviamente, dovrete realizzare dei brevi script per avere la risposta desiderata.

2. Creare e analizzare un grafo orientato

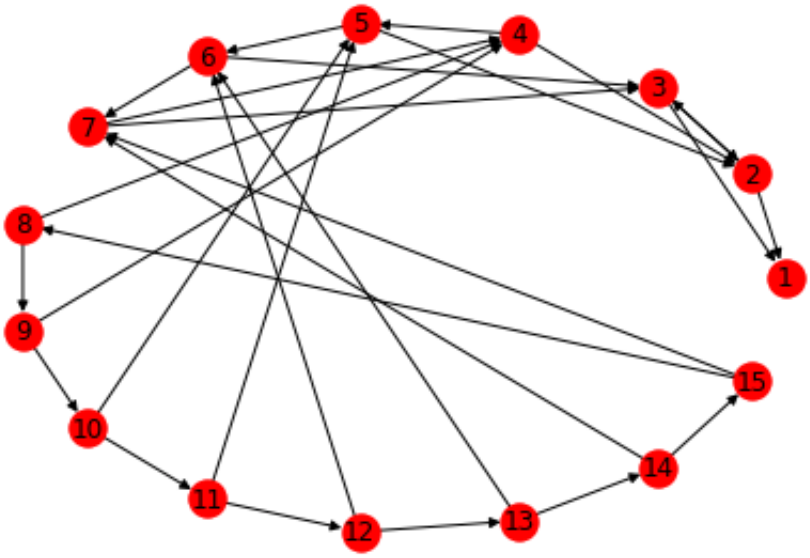
Dividete i numeri interi positivi in gruppi, che chiameremo *gironi*. Il primo girone è costituito solo dal numero 1. Ogni girone contiene il doppio di numeri del girone precedente. Quindi i gironi sono:

- 1
- 2, 3
- 4, 5, 6, 7
- 8, 9, 10, 11, 12, 13, 14, 15

Considerate il grafo `U(k)` i cui nodi sono i numeri dei primi `k` gironi, così collegati fra loro:

- i nodi di ciascun girone costituiscono un ciclo (ad es., i nodi del terzo girone sono collegati come 4 -> 5 -> 6 -> 7 -> 4)
- inoltre ogni nodo di ciascun girone è collegato ad un nodo del girone precedente, secondo la regola per cui il nodo `x` è collegato al nodo `int(x/2)` (ad esempio, 4 -> 2, 5 -> 2, 6 -> 3, 7 -> 3).

In questa figura mostriamo `U(4)` :



Osservate che i gironi non sono rappresentati (come vi aspettereste) da cerchi concentrici, ma sono comunque abbastanza facilmente individuabili. Provate manualmente a trovare un'altra rappresentazione più comprensibile dello stesso grafo.

1. Scrivete una funzione Python che, dato `k` restituisca un oggetto `networkx` che rappresenti il grafo *orientato* `U(k)` .
2. Disegnate il grafo `U(k)` , per una qualche scelta di `k` . Come vedrete, la rappresentazione non è ottimale. Per migliorarla potete cambiare l'algoritmo di layout. In generale, un algoritmo di layout è un algoritmo che prende un grafo e decide come posizionarne i nodi; in `networkx` esistono vari algoritmi di layout (quello usato per default si chiama `spring_layout`), ognuno dei quali può eventualmente prendere degli argomenti addizionali per migliorare l'output:

   2.1. Chiamate sul grafo `U(k)` l'algoritmo `shell_layout` e salvate la mappa che vi restituisce in un'apposita variabile (diciamo `pos`). Che cosa contiene `pos` ? Riuscite a interpretarne il significato? 2.2. Ora passate `pos` come secondo argomento a `nx.draw` e guardate l'esito.

3. Rispondete alle seguenti domande:

   3.1. Quanti nodi ha `U(k)` ?

   3.2. Quanti archi ha `U(k)` ?

   3.3. Quante e quali sono le componenti connesse forti di `U(k)` ?

   3.4. Quante e quali sono le componenti connesse deboli di `U(k)` ?

   3.5. Com'è fatto il DAG delle componenti connesse forti?

   3.6. Quanto è lungo e come è fatto il più lungo ciclo semplice nel grafo (la risposta potrebbe dipendere da `k`)? E il più lungo cammino semplice?

3.6. Raggruppate i vertici del grafo a seconda del loro grado positivo: quanti gruppi esistono? quali sono i gradi positivi e quanti sono i vertici di ciascun grado positivo?

3.6. Raggruppate i vertici del grafo a seconda del loro grado negativo: quanti gruppi esistono? quali sono i gradi negativi e quanti sono i vertici di ciascun grado negativo?

4. Per ciascuna delle domande precedenti, tentate di rispondere usando le funzioni della libreria `networkx`. In qualche caso, ovviamente, dovrete realizzare dei brevi script per avere la risposta desiderata. **Attenzione**: in `networkx` la funzione che calcola il DAG delle componenti connesse forti si chiama `nx.condensation`.