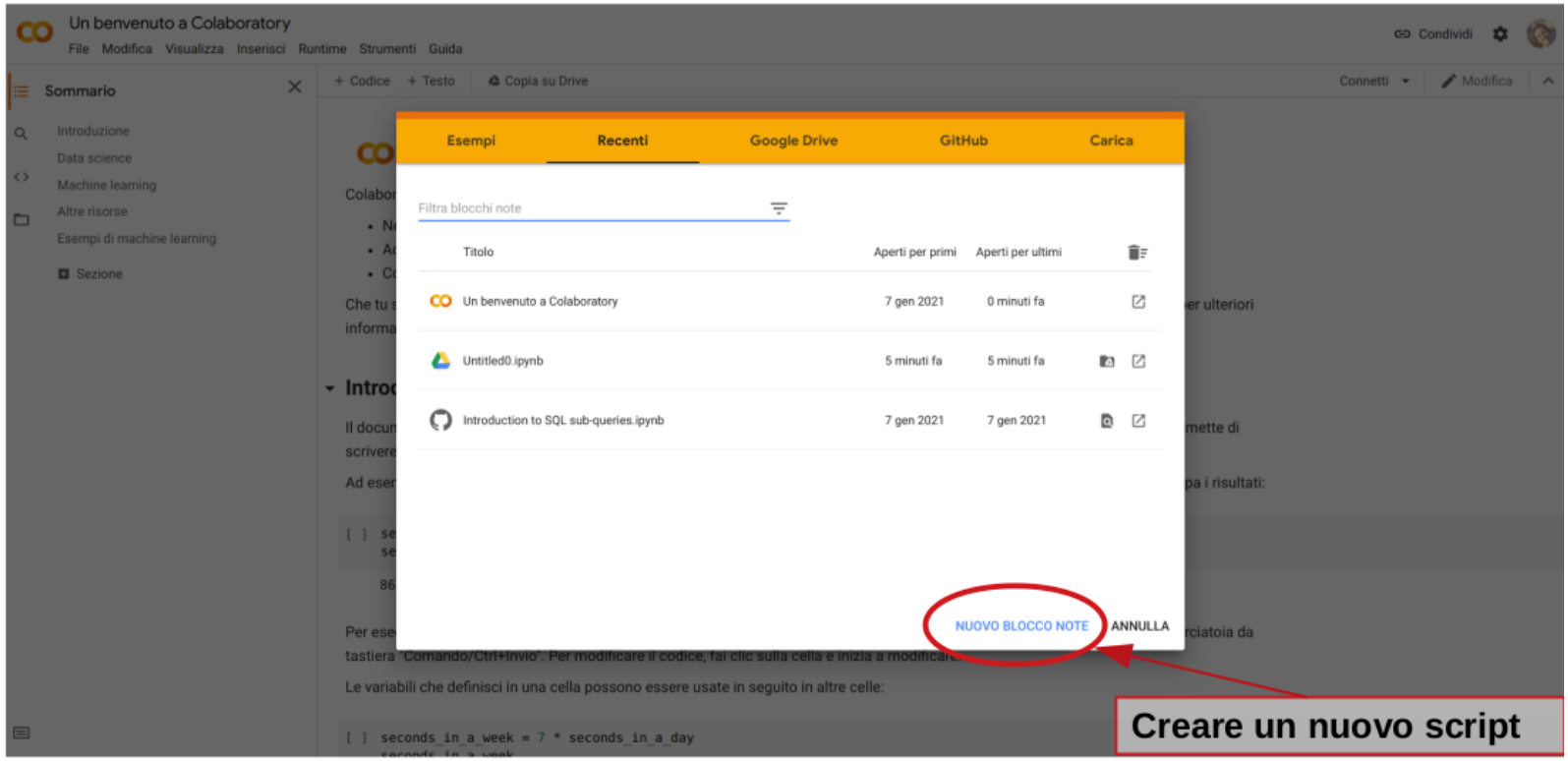
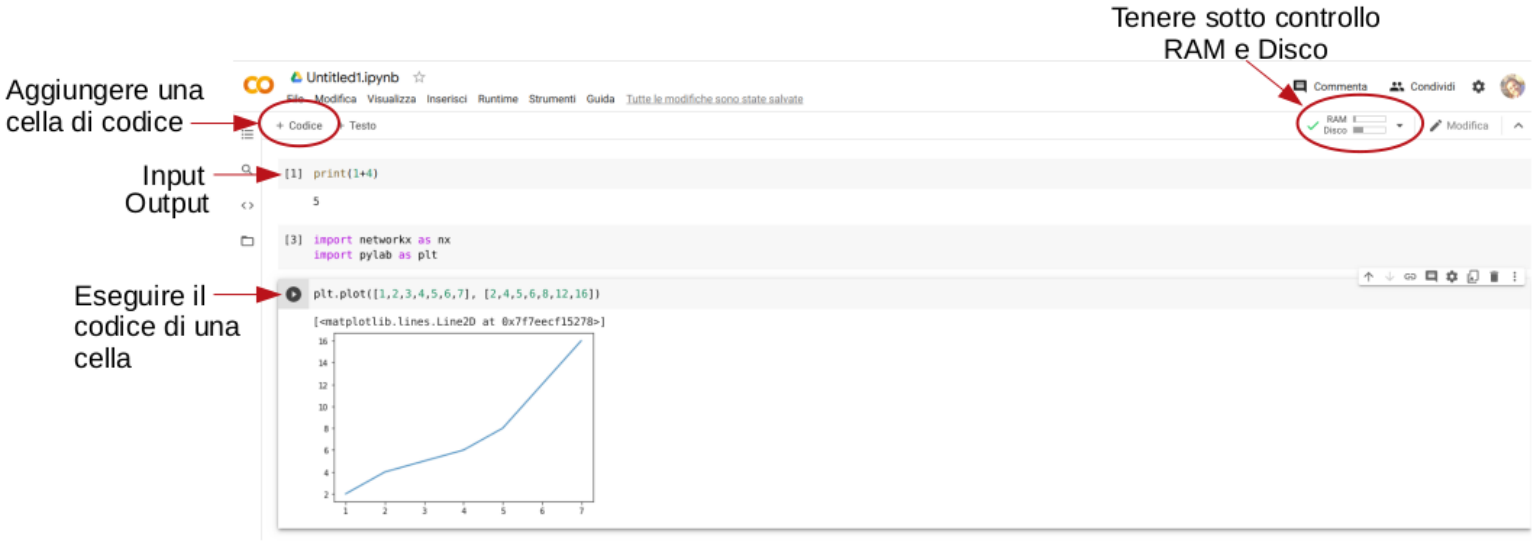
# Questionario di Ateneo

https://colab.research.google.com/




Tenere sotto controllo
RAM e Disco
Aggiungere una
cella di codice
Input
Output
Eseguire il
codice di una
cella
Untitled1.ipynb
File Modifica Visualizza Inserisci Runtime Strumenti Guida Tutte le modifiche sono state salvate
Commenta
Condividi
+ Codice
+ Testo
RAM
Disco
Modifica
[1] print(1+4)
5
[3] import networkx as nx
import pylab as plt
plt.plot([1,2,3,4,5,6,7], [2,4,5,6,8,12,16])
[<matplotlib.lines.Line2D at 0x7f7eecf15278>]

# Network Biologici

Oggi studiamo come analizzare alcuni network biologici.

Ne vedremo in particolare uno, chiamato **DISEASOMA**, che connette i **geni** e le **malattie**, e vedremo come lo possiamo usare per ricavare informazioni a proposito di alcuni geni e del modo in cui questi si sono evoluti.

Una delle più famose librerie per la gestione dei network in python è **networkx**, che rende molto semplice la gestione dei network.

**Networkx** non è computazionalmente molto potente, ma può lavorare su network molto complessi (come tipo di informazioni contenute).

In [2]:
```
import networkx as nx
import pylab as plt

G = nx.Graph()
```

Iniziamo definendo quali siano i nodi del nostro network.

Per iniziare con una cosa comprensibile, consideriamo un network di persone ed interessi.

Dobbiamo per prima cosa definire quali siano le persone presenti e gli argomenti disponibili come interessi.

In [3]:
```
G.add_node('Enrico', tipo='persona')
G.add_node('Daniel', tipo='persona')
G.add_node('Alessandra', tipo='persona')
G.add_node('Claudia', tipo='persona')
G.add_node('Tommaso', tipo='persona')
G.add_node('Gastone', tipo='persona')
```

In [4]:

```
G.add_node('Python', tipo='linguaggio')
G.add_node('R', tipo='linguaggio')
G.add_node('Matlab', tipo='linguaggio')
G.add_node('Mathematica', tipo='linguaggio')
```

Possiamo ora inserire la lista delle connessioni fra le persone ed i loro interessi.

In [5]:
```
G.add_edge('Enrico', 'Python')
G.add_edge('Claudia', 'Python')
G.add_edge('Claudia', 'R')
G.add_edge('Daniel', 'Matlab')
G.add_edge('Alessandra', 'Matlab')
G.add_edge('Alessandra', 'Python')
G.add_edge('Tommaso', 'Python')
G.add_edge('Tommaso', 'R')
G.add_edge('Gastone', 'Mathematica')
```

Networkx permette una semplice visualizzazione.

Non è molto piacevole (conviene usare programmi dedicati, come Gephi, per fare visualizzazioni più raffinate), ma è molto semplice ed è abbastanza per i nostri scopi.

In [6]:
```
nx.draw_networkx(G, node_color='white', linewidths=0)
plt.axis('off');
```

Possiamo usare la funzione `nodes()` per vedere la lista dei nodi a disposizione o le proprietà di ogni singolo nodo

In [7]:
```
print(G.nodes())
```

```
['Enrico', 'Daniel', 'Alessandra', 'Claudia', 'Tommaso', 'Gastone', 'Python',
'R', 'Matlab', 'Mathematica']
```

In [8]:
```
G.nodes["Enrico"]
```

Out[8]:
```
{'tipo': 'persona'}
```

Possiamo selezionare tutti i nodi di un tipo in base alle proprietà dei nodi in modo abbastanza semplice

In [9]:
```
[n for n in G if G.nodes[n]['tipo']=='linguaggio']
```

Out[9]:
```
['Python', 'R', 'Matlab', 'Mathematica']
```

In [10]:
```
[n for n in G if G.nodes[n]['tipo']=='persona']
```

Out[10]:
```
['Enrico', 'Daniel', 'Alessandra', 'Claudia', 'Tommaso', 'Gastone']
```

Networkx include anche alcune funzioni utili per descrivere le proprietà dei nodi.

Una che useremo molto è la funzione `degree()`, che restituisce il numero dei vicini che ciascun nodo ha.

In [11]:
```
print(nx.degree(G))
```

```
[('Enrico', 1), ('Daniel', 1), ('Alessandra', 2), ('Claudia', 2), ('Tommaso',
2), ('Gastone', 1), ('Python', 4), ('R', 2), ('Matlab', 2), ('Mathematica',
1)]
```

La funzione `shortest_path()` ci dice qual è il cammino più corto per andare da un nodo ad un altro.

In [12]:
```
nx.shortest_path(G, source='Enrico', target='Daniel')
```

Out[12]:
```
['Enrico', 'Python', 'Alessandra', 'Matlab', 'Daniel']
```

La funzione che ci interessa di più per oggi è la possibilità di accorpare i nodi del network per creare un nuovo network.

Questo nuovo network conterrà un sottoinsieme dei nodi e avrà un link tra due nodi se questi avevano un vicino in comune nel network iniziale.

In [13]:

```
from networkx.algorithms.bipartite import  projected_graph

nodi = [n for n in G if G.nodes[n]['tipo']=='persona']

B = projected_graph(G, nodi)

print(B.nodes())

print(B.edges())
```

```
['Enrico', 'Daniel', 'Alessandra', 'Claudia', 'Tommaso', 'Gastone']
[('Enrico', 'Tommaso'), ('Enrico', 'Claudia'), ('Enrico', 'Alessandra'), ('Dan
iel', 'Alessandra'), ('Alessandra', 'Tommaso'), ('Alessandra', 'Claudia'), ('C
laudia', 'Tommaso')]
```

In [14]:
```
nx.draw_networkx(B, node_color='w')
plt.axis('off');
```

Questo network è chiaramente diviso in due pezzi separati, visto che un nodo non è connesso a nessun altro.

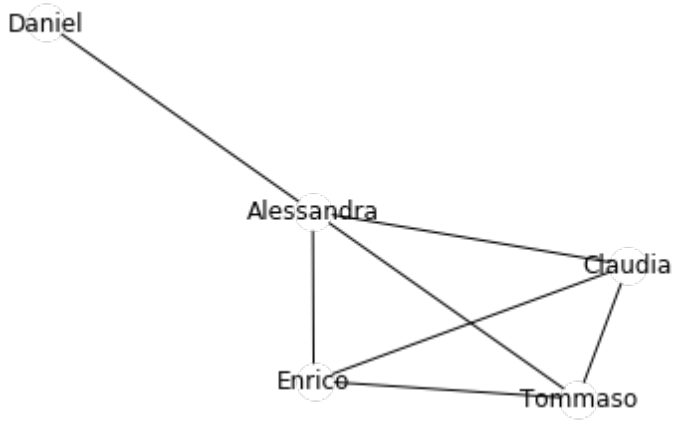
Posso quindi dividerlo in pezzi e studiarne ciascuno indipendentemente.

In [15]:
```
componenti = list(nx.connected_components(B)) # componenti connesse
print(componenti)
print(len(componenti))
print(len(componenti[0]))
print(len(componenti[1]))
```

```
[{'Daniel', 'Enrico', 'Tommaso', 'Alessandra', 'Claudia'}, {'Gastone'}]
2
5
1
```

In [16]:
```
B_sub_0 = B.subgraph(componenti[0]) # grafo contenente la prima componente
```

In [17]:
```
nx.draw_networkx(B_sub_0, node_color='w')

plt.axis('off');
```



In [18]:
```
B_sub_1 = B.subgraph(componenti[1])  # grafo contenente la seconda componente

nx.draw_networkx(B_sub_1, node_color='w')

plt.axis('off');
```

# KEGG (Kyoto Encyclopedia of Genes and Genomes)

https://www.genome.jp/kegg/ (https://www.genome.jp/kegg/)

KEGG è un database libero, curato a mano e costantemente aggiornato che contiene databases riguardanti:

- i genomi (organismi)
- i pathways biologici (insieme delle reazioni chimiche coinvolte nel metabolismo cellulare)
- le malattie
- i farmaci
- le sostanze chimiche

I dati del diseasome si trovano in **KEGG DISEASE** ( https://www.kegg.jp/kegg/disease/ (https://www.kegg.jp/kegg/disease/) ).

Per scaricarli basterà connettersi ad una pagina web.

Da lì potremo scaricare un file di testo contenente la **lista di relazioni fra geni e malattie**.

Una volta che avremo queste relazioni, potremo unirle in un unico enorme network, ed ottenere la rete dei geni facendo lo stesso collasso che abbiamo fatto prima.

In [19]:

```
import requests

url = "http://rest.kegg.jp/link/disease/hsa"
gene_disease = requests.get(url).text
```

In [20]:
```
print(repr(gene_disease[:40]))
```

```
'hsa:7428\tds:H00021\nhsa:4233\tds:H00021\nhs'
```

In [21]:
```
print(gene_disease[:37])
# hsa è il codice per Homo Sapiens
# hsa:7428 è il codice del gene 7428 di Homo Sapiens
# ds:H00021 è il codice della malattia
```

```
hsa:7428        ds:H00021
hsa:4233        ds:H00021
```

Posso cercare una descrizione completa del gene e della malattia in KEGG:

https://www.kegg.jp/dbget-bin/www_bget?hsa:7428 (https://www.kegg.jp/dbget-bin/www_bget?hsa:7428) → VHL: Von Hippel-Lindau tumor suppressor

https://www.kegg.jp/dbget-bin/www_bget?ds:H00021 (https://www.kegg.jp/dbget-bin/www_bget?ds:H00021) → Renal cell carcinoma

Oppure direttamente da python:

In [22]:

```
info = requests.get("http://rest.kegg.jp/get/hsa:7428").text

# print(info)
print("\n".join(info.splitlines()[1:10]))
```

```
NAME        VHL, HRCA1, RCA1, VHL1, pVHL
DEFINITION  (RefSeq) von Hippel-Lindau tumor suppressor
ORTHOLOGY   K03871  von Hippel-Lindau disease tumor supressor
ORGANISM    hsa  Homo sapiens (human)
PATHWAY     hsa04066  HIF-1 signaling pathway
            hsa04120  Ubiquitin mediated proteolysis
            hsa05200  Pathways in cancer
            hsa05211  Renal cell carcinoma
NETWORK     nt06225  HIF-1 signaling
```

In [24]:
```
# Possiamo salvare il file e rileggerlo:

file = open("hsa.txt", "w")

file.write(gene_disease)

file.close()
```

In [25]:
```
import itertools as it

with open("hsa.txt", 'r') as infile:
    for line in it.islice(infile,10):
        gene, disease = line.strip().split('\t') # strip() rimuove gli spazi bi
anchi a sx e dx della stringa
        print(gene, disease)
```

```
hsa:7428 ds:H00021
hsa:4233 ds:H00021
hsa:2271 ds:H00021
hsa:201163 ds:H00021
hsa:7030 ds:H00021
hsa:894 ds:H00023
hsa:411 ds:H00131
hsa:1075 ds:H00274
hsa:2720 ds:H00281
hsa:2588 ds:H00123
```

In [29]:

```
KEGG = nx.Graph()

with open("hsa.txt", 'r') as file:

    for line in file:
    # for line in it.islice(file, 500):  # per velocizzare posso usare solo le
prime 500 righe:
        gene, disease = line.strip().split('\t')

        KEGG.add_node(disease, tipo='disease')

        KEGG.add_node(gene, tipo='gene')

        KEGG.add_edge(gene, disease)
```

In [30]:
```
nx.draw_networkx(KEGG, node_size=5, with_labels=False)

plt.axis('off')
```

Out[30]:
```
(-1.1021840200873292,
 1.0723974333258544,
 -1.098590861722799,
 1.0918201317041833)
```

In [31]:
```
nodi_geni = [n for n in KEGG if KEGG.nodes[n]['tipo']=='gene']
nodi_malattie = [n for n in KEGG if KEGG.nodes[n]['tipo']=='disease']
```

In [32]:
```
# Grafo dei geni:
gene_net = projected_graph(KEGG, nodi_geni)
len(gene_net)
```

Out[32]: 4071

In [33]:
```
# Grafo delle malattie:
disease_net = projected_graph(KEGG, nodi_malattie)
len(disease_net)
```

Out[33]: 1966

In [34]:
```
componenti_geni = list(nx.connected_components(gene_net))
len(componenti_geni)
```

Out[34]:
```
665
```

In [35]:
```
# Quanti geni ci sono in ogni componente?
componenti_geni = sorted(componenti_geni, key=len, reverse=True)
list(map(len, componenti_geni[:10]))
```

Out[35]:
```
[3058, 21, 19, 17, 13, 12, 11, 10, 9, 8]
```

In [36]:
```
componenti_disease = list(nx.connected_components(disease_net))
len(componenti_disease)
```

Out[36]:
```
665
```

In [37]:
```
# Quante malattie ci sono in ogni componente?
componenti_disease = sorted(componenti_disease, key=len, reverse=True)
list(map(len, componenti_disease[:10]))
```

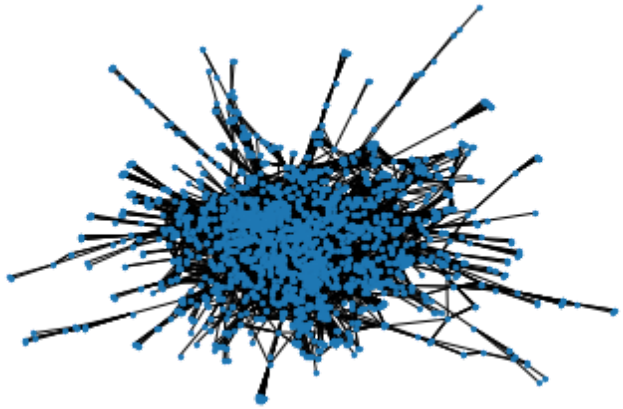
Out[37]:
```
[1183, 8, 8, 6, 5, 5, 5, 4, 4, 4]
```

In [38]:
```
sottografo_geni = gene_net.subgraph(componenti_geni[0])  # grafo della prima co
mponente
list(sottografo_geni)[:5]
```

Out[38]:
```
['hsa:7428', 'hsa:4233', 'hsa:2271', 'hsa:201163', 'hsa:7030']
```

In [39]:
```
nx.draw_networkx(sottografo_geni, node_size=5, with_labels=False)

plt.axis('off')
```

Out[39]:
```
(-0.9497891501637374,
 1.0056190923901476,
 -0.7632485528439005,
 1.0893508692234475)
```



In [40]:
```
import operator as op

sorted(sottografo_geni.degree(),
       key = op.itemgetter(1), # ordina in base al secondo elemento (il contegg
io)
       reverse = True)[:5]
```

Out[40]:
```
[('hsa:57465', 187),
 ('hsa:7157', 154),
 ('hsa:2904', 140),
 ('hsa:1917', 140),
 ('hsa:11284', 135)]
```

In [41]:
```
info = requests.get("http://rest.kegg.jp/get/hsa:57465").text
print("\n".join(info.splitlines()[1:20]))
```

```
NAME        TBC1D24, DEE16, DFNA65, DFNB86, DOORS, EIEE16, EPRPDC, FIME, TLDC6
DEFINITION  (RefSeq) TBC1 domain family member 24
ORTHOLOGY   K21841  TBC1 domain family member 24
ORGANISM    hsa  Homo sapiens (human)
DISEASE     H00604  Deafness, autosomal dominant
            H00605  Deafness, autosomal recessive
            H00606  Early infantile epileptic encephalopathy
            H01815  Malignant migrating partial seizures in infancy
            H02212  Familial infantile myoclonic epilepsy
            H02218  DOORS syndrome
BRITE       KEGG Orthology (KO) [BR:hsa00001]
             09180 Brite Hierarchies
              09182 Protein families: genetic information processing
               04131 Membrane trafficking [BR:hsa04131]
                57465 (TBC1D24)
            Membrane trafficking [BR:hsa04131]
             Endocytosis
              Arf GTPases and associated proteins
               Arf associated proteins
```

In [42]:
```
info = requests.get("http://rest.kegg.jp/get/hsa:7157").text
print("\n".join(info.splitlines()[1:20])) # detta "il guardiano del genoma"
```

```
NAME        TP53, BCC7, BMFS5, LFS1, P53, TRP53
DEFINITION  (RefSeq) tumor protein p53
ORTHOLOGY   K04451  tumor protein p53
ORGANISM    hsa  Homo sapiens (human)
PATHWAY     hsa01522  Endocrine resistance
            hsa01524  Platinum drug resistance
            hsa04010  MAPK signaling pathway
            hsa04071  Sphingolipid signaling pathway
            hsa04110  Cell cycle
            hsa04115  p53 signaling pathway
            hsa04137  Mitophagy - animal
            hsa04151  PI3K-Akt signaling pathway
            hsa04210  Apoptosis
            hsa04211  Longevity regulating pathway
            hsa04216  Ferroptosis
            hsa04218  Cellular senescence
            hsa04310  Wnt signaling pathway
            hsa04722  Neurotrophin signaling pathway
            hsa04919  Thyroid hormone signaling pathway
```

# Modellizzare il network

Vorremmo descrivere il nostro network, cercando di intuirne delle proprietà.

Usiamo un approccio simile a quello che abbiamo fatto con la divina commedia: partendo da dei modelli probabilistici, vediamo se riusciamo a generare qualcosa che assomigli al nostro network.

**Partiamo con il modello più semplice possibile, quello random.**

In teoria dei network questo modello viene di solito chiamato modello di Erdős–Rényi (https://en.wikipedia.org/wiki/Erd%C5%91s%E2%80%93R%C3%A9nyi_model) (dai matematici che ne hanno studiato le proprietà per primi).

- Ho N nodi.
- Per ogni coppia di nodi ho una probabilità $p$ di avere un link.

Posso stimare questa probabilità $p$ in base al numero di link esistenti nel network reale rispetto a tutti quelli che potrebbero esistere fra i vari nodi.

In [44]:
```
N = len(sottografo_geni) # numero di nodi

num_of_possible_links = (N*(N-1)/2) # numero possibile di link

L = len(sottografo_geni.edges()) # numero di link

p =  L / num_of_possible_links # probabilità di avere un link tra due nodi

print(p)
```

```
0.008303322548491674
```

In [45]:

```
B_hat = nx.erdos_renyi_graph(50, p)

nx.draw_networkx(B_hat, node_size=20, with_labels=False)

plt.axis('off');
```

In [46]:

```
B_hat = nx.erdos_renyi_graph(len(sottografo_geni), p)
```

In [47]:
```
gradi_B = [val for key, val in sottografo_geni.degree()]

gradi_B_hat = [val for key, val in B_hat.degree()]
```

In [48]:
```
from statistics import mean

print(mean(gradi_B))
print(mean(gradi_B_hat))
```

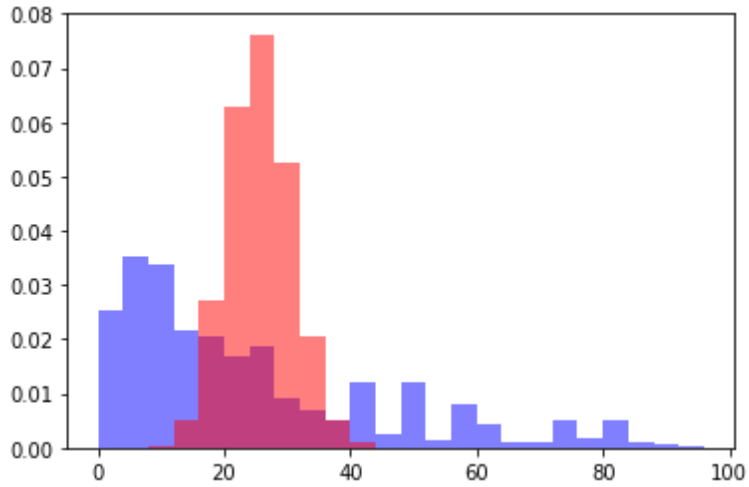
```
25.383257030739045
25.17331589274035
```

In [49]:
```
print(max(gradi_B))
print(max(gradi_B_hat))
```

```
187
43
```

In [50]:

```
import pylab as plt

bins = range(0, 100, 4)
props = dict(density=True, alpha=0.5, bins=bins)

plt.hist(gradi_B, **props, color='b')
plt.hist(gradi_B_hat, **props, color='r');
```



Il prossimo modello è un modello di accrescimento, chiamato **Barabasi-Albert**.

In questo modello parto da un nodo più antico e aggiungo progressivamente nuovi nodi.

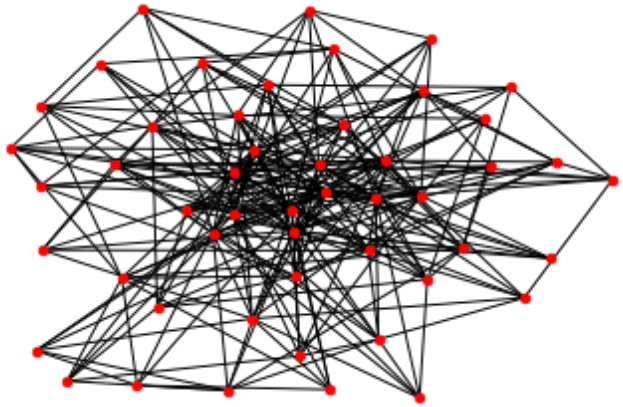
Questi nodi hanno a disposizione un certo numero di link, e si connettono con i nodi già presenti.

La probabilità di connettersi ad un nodo è proporzionale alla frazione di nodi già connessi con lui.

Viene definito un modello **"Rich get Richer"**.

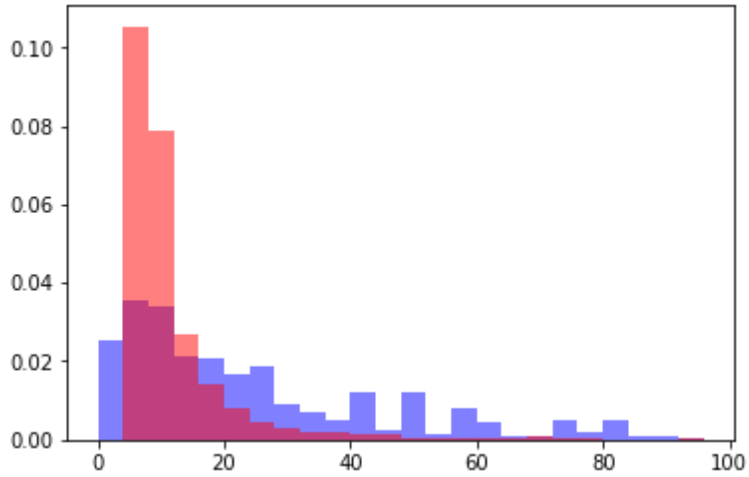
In [51]:

```
BBA = nx.barabasi_albert_graph(50, 6) # 50 nodi, 6 link per nodo

nx.draw_networkx(BBA, node_size=20, with_labels=False,node_color="red")

plt.axis('off');
```



In [52]:

```
BBA = nx.barabasi_albert_graph(len(sottografo_geni), 6)

gradi_bba = [val for key, val in BBA.degree()]

plt.hist(gradi_B, **props, color='b')
plt.hist(gradi_bba, **props, color='r');
```



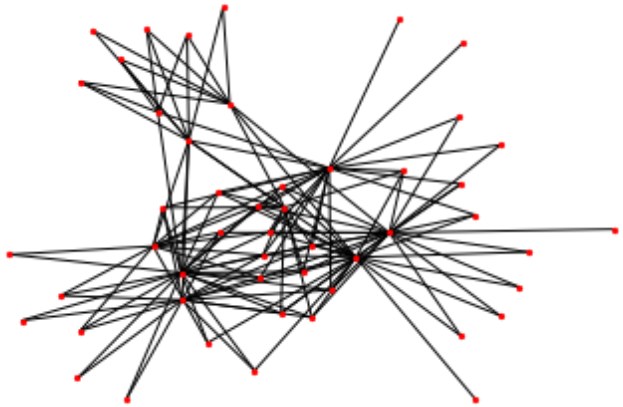
Un modello di ispirazione più biologica è il **duplication divergence**.

- Parto da un piccolo set di nodi iniziali.
- Ad ogni passo scelgo a caso uno di questi nodi e lo duplico, duplicando anche tutte le sue connessioni.
- Dopo la duplicazione, rimuovo a caso alcuni dei link che ha, e ne aggiungo di nuovi in modo casuale.

Questo processo assomiglia molto al processo di evoluzione naturale dei geni, in quello che è chiamato un processo di "evoluzione neutrale".

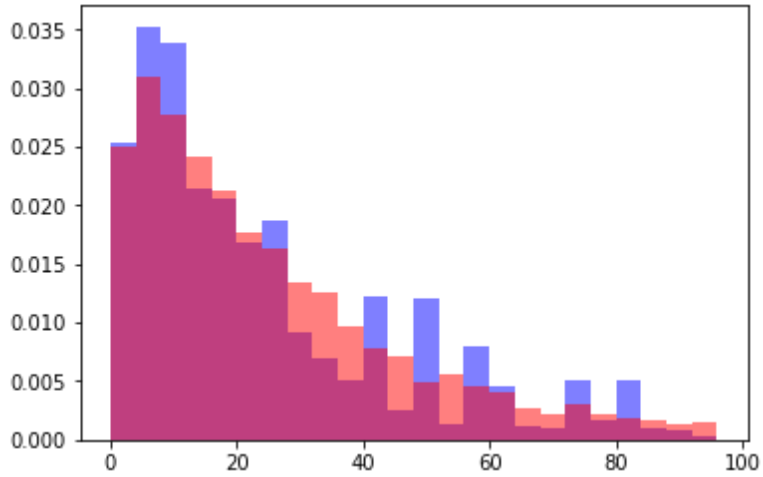
In [53]:

```
BDD = nx.duplication_divergence_graph(50, 0.7)

nx.draw_networkx(BDD, node_size=5, with_labels=False, node_color="red")

plt.axis('off');
```



In [54]:

```
BDD = nx.duplication_divergence_graph(len(sottografo_geni), 0.7)

gradi_bdd = [val for key, val in BDD.degree()]

plt.hist(gradi_B, **props, color='b')
plt.hist(gradi_bdd, **props, color='r');
```



# Reti di parole

Se volessimo, potremmo fare un network di parole, basato sulla distanza di edit.

La distanza di edit tra due stringhe A e B è il numero minimo di modifiche elementari che consentono di trasformare la A nella B. Per modifica elementare si intende

- la cancellazione di un carattere
- la sostituzione di un carattere con un altro
- l'inserimento di un carattere.

Possiamo copiare una implementazione da wikipedia:

https://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Levenshtein_distance#Python (https://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Levenshtein_distance#Python)

In [56]:

```
def lev(a, b):
    assert len(a)==len(b) # solo per stringhe lunghe uguali
    totale = 0
    for (ai, bi) in zip(a, b):
        if ai!=bi:
            totale += 1
    return totale

print(lev('canna', 'manna'))
print(lev('canna', 'manno'))
```

```
1
2
```

In [57]:

```
parole = ['gatto', 'patto', 'patio', 'matto', 'masso', 'lasso',
          'casto', 'casta', 'pasta', 'rasta', 'messa', 'massa',
          'cassa', 'bassa', 'pazzo', 'lessa', 'lesso', 'messo',
          'mezzo', 'mezza', 'rosso', 'rossa', 'mosso', 'mossa',
          'morso', 'morsa', 'corso', 'corsa', 'colza', 'cozza',
          'razza', 'razzo', 'rozza', 'rozzo', 'ressa', 'rissa',
          'ritta', 'ritto', 'risma', 'fitto', 'fitta', 'finto',
          'finta']

words = nx.Graph()
```

In [58]:

```
from itertools import combinations

# Per ogni coppia di parole aggiungo un link se la distanza è 1
for parola1, parola2 in combinations(parole, 2):

    if lev(parola1, parola2)==1:

        words.add_edge(parola1, parola2)
```

In [59]:

```
fig, ax = plt.subplots(figsize=(10, 5))
nx.draw_networkx(words, node_color='w')
ax.axis('off');
```

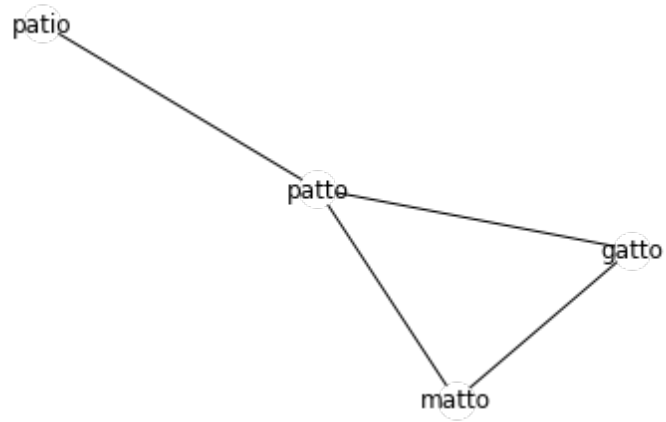
In [60]:
```
for componente in nx.connected_components(words):
    sottografo = words.subgraph(componente)
    print()
    print(sottografo.nodes())
```

```
['patto', 'patio', 'gatto', 'matto']

['masso', 'lasso', 'massa', 'messo', 'mosso', 'lesso', 'casto', 'casta', 'past
a', 'rasta', 'cassa', 'messa', 'lessa', 'mossa', 'ressa', 'bassa', 'rosso', 'r
ossa', 'rissa', 'morso', 'morsa', 'corso', 'corsa', 'risma']

['pazzo', 'razza', 'rozza', 'cozza', 'rozzo', 'razzo', 'colza']

['mezza', 'mezzo']

['fitta', 'finto', 'finta', 'fitto', 'ritto', 'ritta']
```

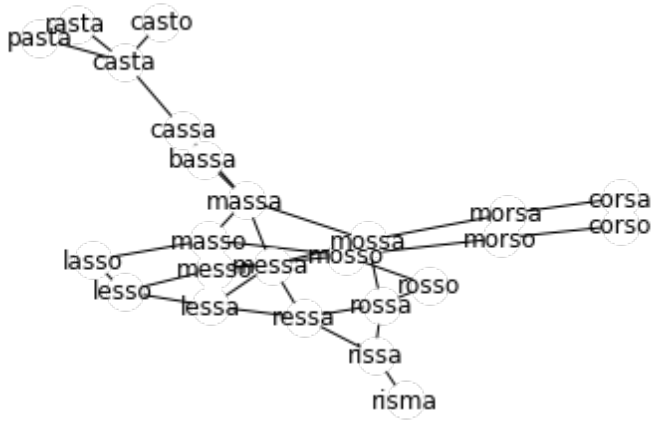
In [61]:
```
componenti = list(nx.connected_components(words))

print(len(componenti))
```

```
5
```

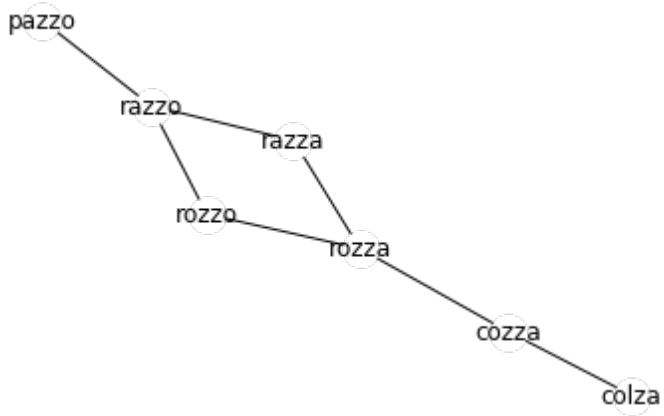
In [62]:
```
sottografo_0 = words.subgraph(componenti[0])
nx.draw_networkx(sottografo_0, node_color='w')
plt.axis('off');
```



In [63]:
```
sottografo_1 = words.subgraph(componenti[1])

nx.draw_networkx(sottografo_1, node_color='w')
plt.axis('off');
```

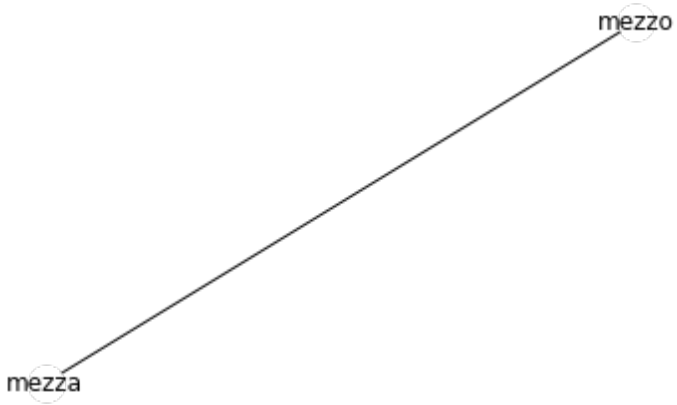



In [64]:
```
sottografo_2 = words.subgraph(componenti[2])
nx.draw_networkx(sottografo_2, node_color='w')
plt.axis('off');
```

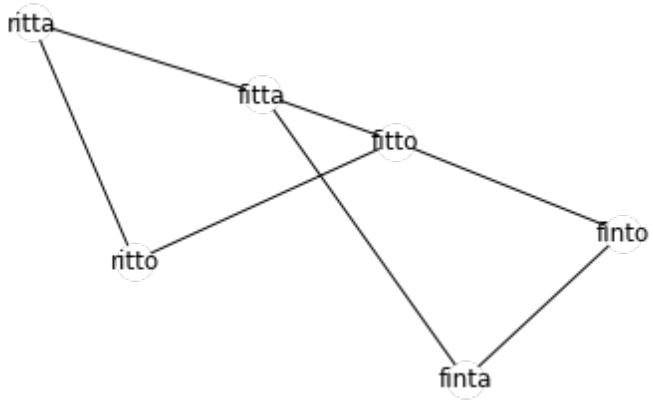


In [65]:

```
sottografo_3 = words.subgraph(componenti[3])

nx.draw_networkx(sottografo_3, node_color='w')
plt.axis('off');
```



In [66]:
```
sottografo_4 = words.subgraph(componenti[4])
nx.draw_networkx(sottografo_4, node_color='w')
plt.axis('off');
```



In [67]:
```
words.degree()
```

Out[67]:
```
DegreeView({'gatto': 2, 'patto': 3, 'matto': 2, 'patio': 1, 'masso': 4, 'lasso
': 2, 'massa': 5, 'messo': 4, 'mosso': 5, 'lesso': 3, 'casto': 1, 'casta': 4,
'pasta': 2, 'rasta': 2, 'cassa': 3, 'messa': 5, 'lessa': 3, 'mossa': 5, 'ressa
': 4, 'bassa': 2, 'pazzo': 1, 'razzo': 3, 'mezzo': 1, 'mezza': 1, 'rosso': 2,
'rossa': 4, 'rissa': 3, 'morso': 3, 'morsa': 3, 'corso': 2, 'corsa': 2, 'colza
': 1, 'cozza': 2, 'rozza': 3, 'razza': 2, 'rozzo': 2, 'risma': 1, 'ritta': 2,
'ritto': 2, 'fitta': 3, 'fitto': 3, 'finto': 2, 'finta': 2})
```

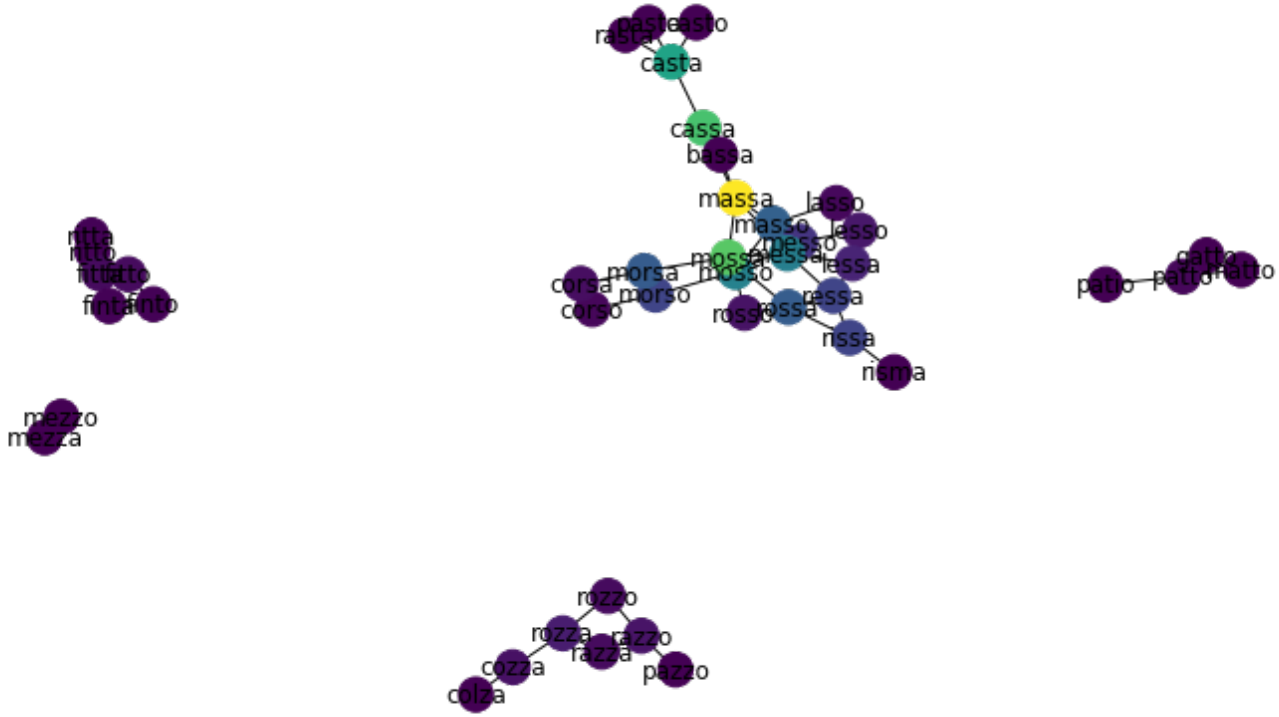
In [68]:
```
nx.betweenness_centrality(words) # somma della frazione di shortest paths che passano per un nodo
```

Out[68]:
```
{'gatto': 0.0,
 'patto': 0.0023228803716608595,
 'matto': 0.0,
 'patio': 0.0,
 'masso': 0.037940379403794036,
 'lasso': 0.0029036004645760743,
 'massa': 0.12433217189314752,
 'messo': 0.019231513743708867,
 'mosso': 0.05449090205187767,
 'lesso': 0.008168795973674023,
 'casto': 0.0,
 'casta': 0.07200929152148665,
 'pasta': 0.0,
 'rasta': 0.0,
 'cassa': 0.08826945412311266,
 'messa': 0.05245838172667441,
 'lessa': 0.012233836624080527,
 'mossa': 0.09223770809136661,
 'ressa': 0.025609756097560978,
 'bassa': 0.0,
 'pazzo': 0.0,
 'razzo': 0.006387921022067363,
 'mezzo': 0.0,
 'mezza': 0.0,
 'rosso': 0.0058362369337979095,
 'rossa': 0.036140147115756874,
 'rissa': 0.025551684088269452,
 'morso': 0.025687185443283005,
 'morsa': 0.03575300038714673,
 'corso': 0.002361595044521874,
 'corsa': 0.004684475416182733,
 'colza': 0.0,
 'cozza': 0.005807200929152149,
 'rozza': 0.009872241579558653,
 'razza': 0.003484320557491289,
 'rozzo': 0.003484320557491289,
```

```
'risma': 0.0,
'ritta': 0.000967866821525358,
'ritto': 0.000967866821525358,
'fitta': 0.003871467286101432,
'fitto': 0.0038714672861014324,
'finto': 0.000967866821525358,
'finta': 0.000967866821525358}
```

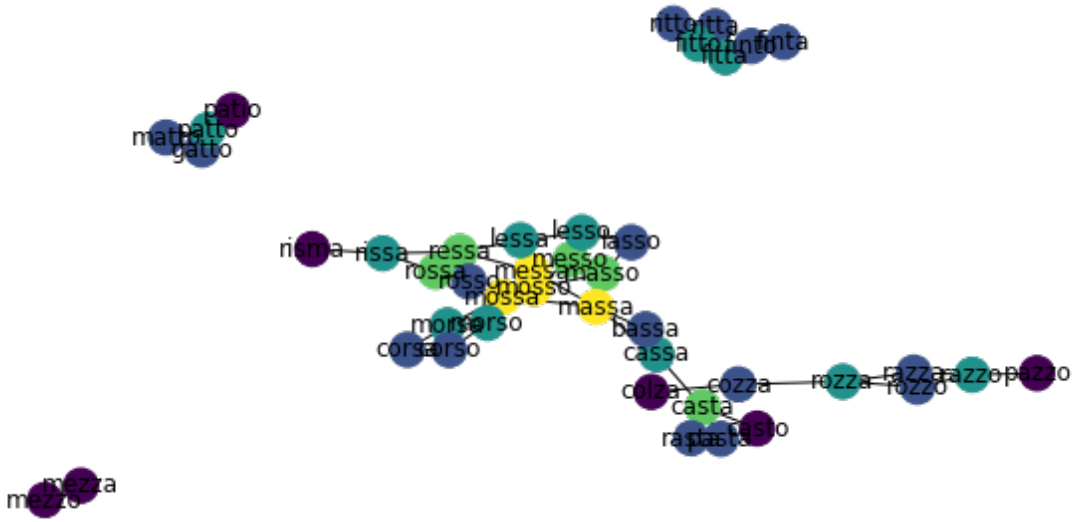
In [70]:

```
fig, ax = plt.subplots(figsize=(12, 7))
node_color = nx.betweenness_centrality(words).values()
loc = nx.spring_layout(words, k=1)
nx.draw_networkx(words,
                 loc=loc,
                 node_color=list(node_color),
                 cmap=plt.cm.viridis,
                 ax=ax)
ax.axis('off');
```



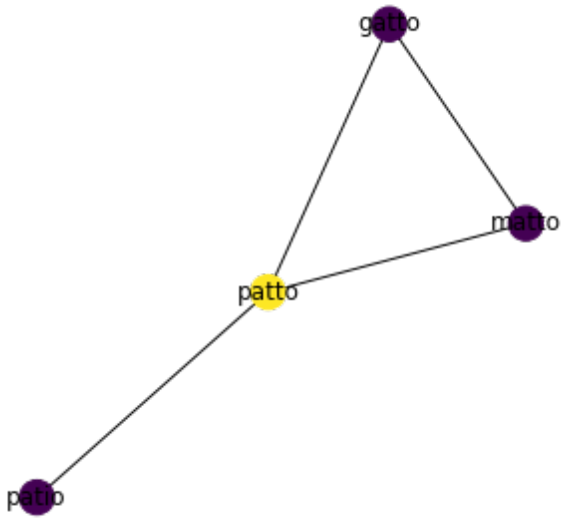
In [71]:

```
fig, ax = plt.subplots(figsize=(10, 5))
node_color = dict(words.degree()).values()
loc = nx.spring_layout(words, k=1)
nx.draw_networkx(words,
                 loc=loc,
                 node_color=list(node_color),
                 cmap=plt.cm.viridis,
                ax=ax)
ax.axis('off');
```



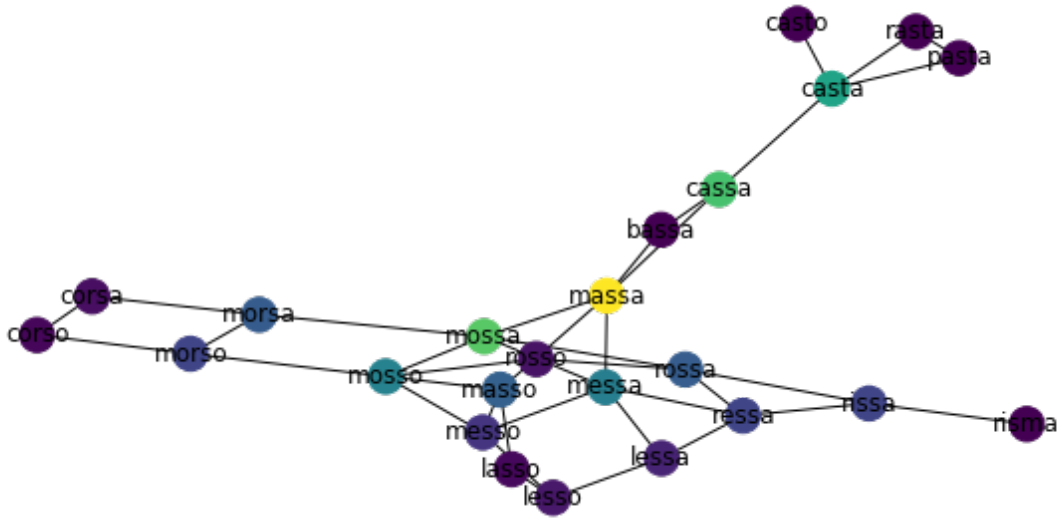
In [72]:

```
fig, ax = plt.subplots(figsize=(5, 5))
node_color = nx.betweenness_centrality(sottografo_0).values()
loc = nx.spring_layout(words, k=1)
r = nx.draw_networkx(sottografo_0,
    loc=loc,
    node_color=list(node_color),
    cmap=plt.cm.viridis,
    ax=ax)
ax.axis('off');
```



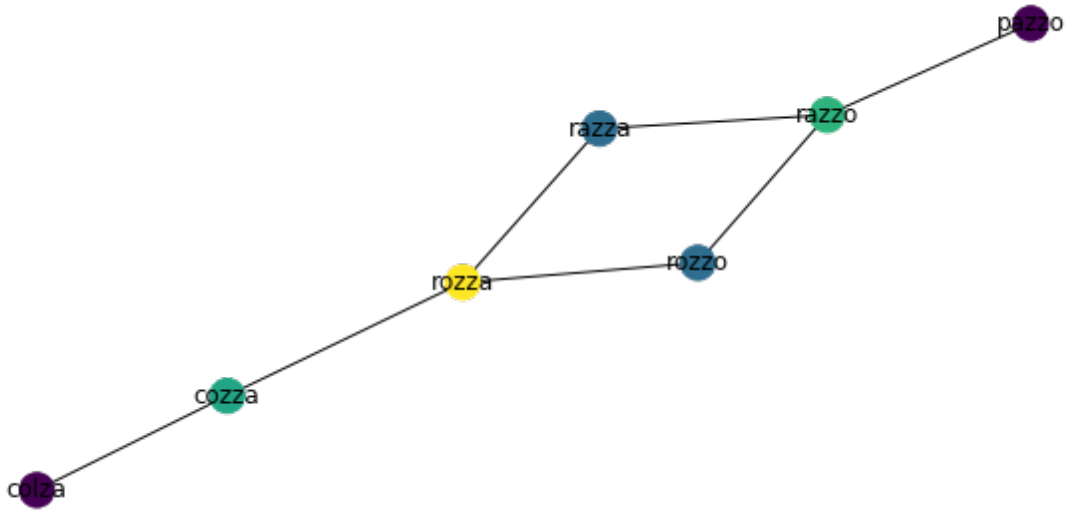
In [73]:

```
fig, ax = plt.subplots(figsize=(10, 5))
node_color = nx.betweenness_centrality(sottografo_1).values()
loc = nx.spring_layout(words, k=1)
r = nx.draw_networkx(sottografo_1,
    loc=loc,
    node_color=list(node_color),
    cmap=plt.cm.viridis,
    ax=ax)
ax.axis('off');
```



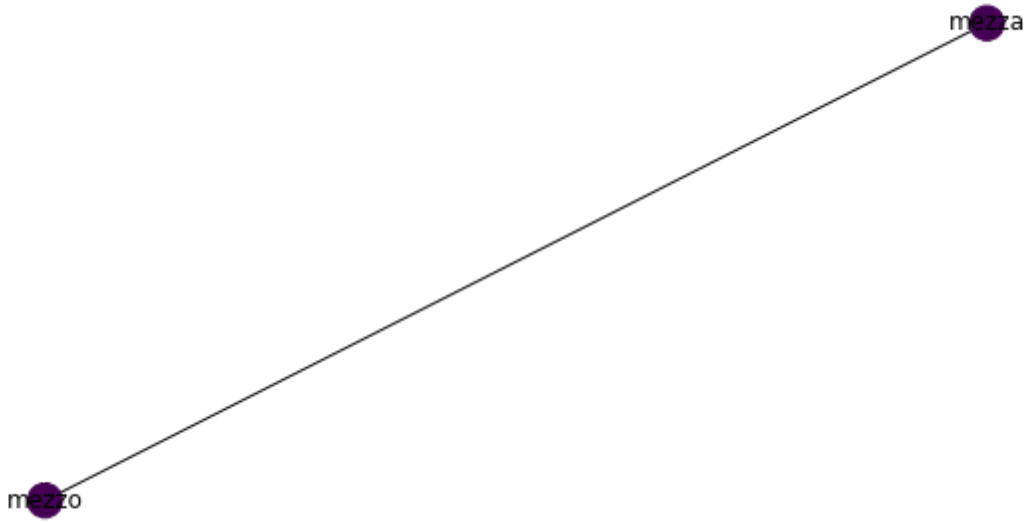
In [74]:

```
fig, ax = plt.subplots(figsize=(10, 5))
node_color = nx.betweenness_centrality(sottografo_2).values()
loc = nx.spring_layout(words, k=1)
r = nx.draw_networkx(sottografo_2,
    loc=loc,
    node_color=list(node_color),
    cmap=plt.cm.viridis,
    ax=ax)
ax.axis('off');
```



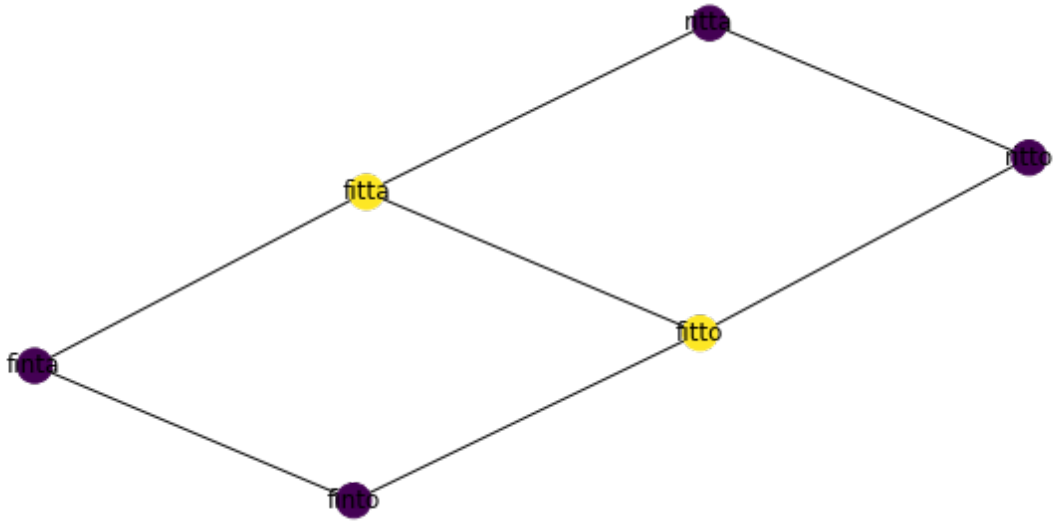
In [75]:

```
fig, ax = plt.subplots(figsize=(10, 5))
node_color = nx.betweenness_centrality(sottografo_3).values()
loc = nx.spring_layout(words, k=1)
r = nx.draw_networkx(sottografo_3,
    loc=loc,
    node_color=list(node_color),
    cmap=plt.cm.viridis,
    ax=ax)
ax.axis('off');
```



In [76]:

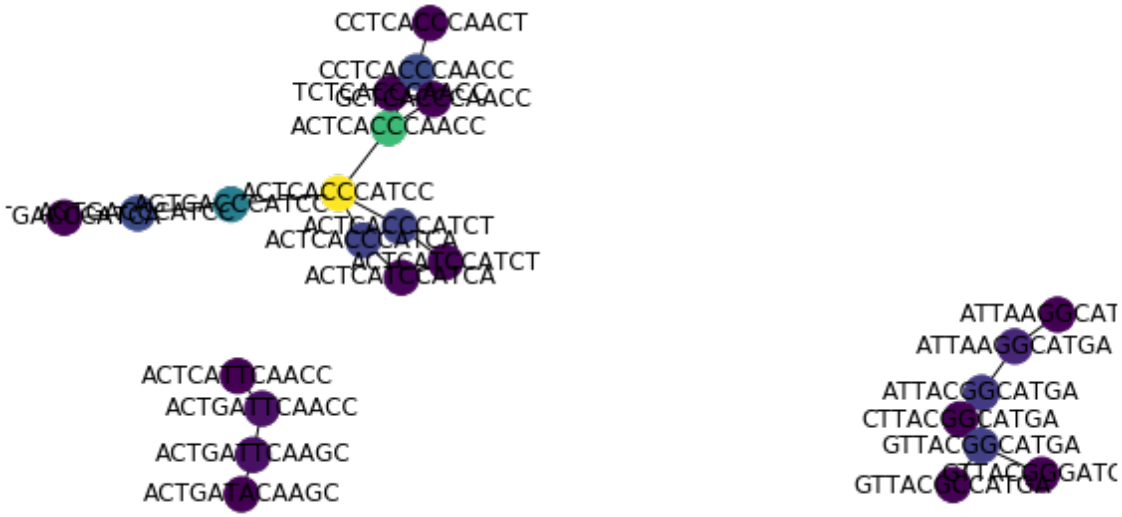
```
fig, ax = plt.subplots(figsize=(10, 5))
node_color = nx.betweenness_centrality(sottografo_4).values()
loc = nx.spring_layout(words, k=1)
r = nx.draw_networkx(sottografo_4,
    loc=loc,
    node_color=list(node_color),
    cmap=plt.cm.viridis,
    ax=ax)
ax.axis('off');
```



In [77]:

```
parole = ['ACTGATTCAAGC', 'ACTGATTCAACC', 'ACTGATACAAGC', 'ACTCATTCAACC', 'ACTC
ACCCAACC', 'TCTCACCCAACC',
          'CCTCACCCAACC', 'CCTCACCCAACT', 'GCTCACCCAACC', 'TCTCACCCAACC', 'AGTC
ACCCATCC', 'ACTCACCCATCC',
          'AGTGACCCATCC', 'AGTGACCCATCA', 'ACTCACCCATCA', 'ACTCATCCATCA', 'ACTC
ACCCATCT', 'ACTCATCCATCT',
          'ATTAAGGCATCA','ATTAAGGCATGA','ATTACGGCATGA','CTTACGGCATGA','GTTACGGC
ATGA','GTTACGCCATGA',
          'GTTACGGGATGA']

words = nx.Graph()
from itertools import combinations
for parola1, parola2 in combinations(parole, 2):
    if lev(parola1, parola2)==1:
        words.add_edge(parola1, parola2)
```

In [78]:
```
fig, ax = plt.subplots(figsize=(10, 5))
node_color = nx.betweenness_centrality(words).values()
loc = nx.spring_layout(words, k=1)
nx.draw_networkx(words,
                 loc=loc,
                 node_color=list(node_color),
                 cmap=plt.cm.viridis,
                 ax=ax)
ax.axis('off');
```




In [ ]: