ANALISI E STUDIO DEL GRAFO WIKIPEDIA

Advanced Algorithms and Graph Mining

Salvatore Baglieri

INTRODUZIONE

Questo progetto si concentra sull'analisi del grafo di Wikipedia utilizzando due dataset distinti:

- Wikipedia IT 2013: un'istantanea della versione italiana di Wikipedia, acquisita alla fine di febbraio 2013, analizzato utilizzando la libreria NetworkX
- Wikipedia EN 2023: un grafo più recente della versione inglese di Wikipedia (2023), analizzato utilizzando la libreria igraph

Il grafo rappresenta le pagine di Wikipedia come nodi e i collegamenti ipertestuali tra di esse come archi.

OBIETTIVI

1. Analisi delle distribuzioni dei gradi:
   - *1.1. Trovare le top 10 pagine con il maggior numero di in-degree in G*
   - *1.2. Trovare le top 10 pagine con il maggior numero di out-degree in G*
   - *1.3. Trovare le top 10 pagine con il maggior numero di gradi totali in U(G)*
2. Calcolo del diametro:
   - *2.1. Calcolare il diametro della componente connessa più grande in U(G)*
   - *2.2. Ripetere l'analisi rimuovendo i nodi con "disambigua" nel nome*
3. Ricerca di cliques massimali:
   - *3.1. Trovare una clique massimale con almeno 3 nodi*
   - *3.2. Trovare due cliques massimali nel grafo*
4. Analisi con enwiki-2023 e igraph:
   - *4.1. Ripetere le analisi su enwiki-2023 utilizzando igraph*
   - *4.2. Trovate tutte le cliques massimali su un sottografo casuale di 100 nodi*
   - *4.3. Trovare la clique massimale con il maggior numero di nodi*

CREAZIONE DEL GRAFO WIKIPEDIA CON NETWORKX

```python
def create_graph_from_files(ids_file_path, arcs_file_path, max_lines=None, max_nodes=None, max_edges=None):
    """
    Crea il grafo diretto G utilizzando solo le prime max_lines righe del file .arcs e max_nodes nodi.
    Può anche limitare il numero di nodi e archi per migliorare le performance.
    Ritorna G (grafo diretto) e U_G (versione non diretta di G).

    - max_lines: Numero massimo di righe da processare (opzionale).
    - max_nodes: Numero massimo di nodi da aggiungere (opzionale).
    - max_edges: Numero massimo di archi da aggiungere (opzionale).
    """
    G = nx.DiGraph()

    id_to_name = {}
    name_to_id = {}

    with open(ids_file_path, 'r', encoding='utf-8') as ids_file:
        for i, line in enumerate(ids_file):
            if max_nodes and i >= max_nodes:
                break
            name = line.strip()
            id_to_name[i] = name
            name_to_id[name] = i

    edges = []
    edge_count = 0
    with open(arcs_file_path, 'r') as arcs_file:
        for i, line in enumerate(arcs_file):
            if max_lines and i >= max_lines:
                break
            if max_edges and edge_count >= max_edges:
                break
            u, v = map(int, line.strip().split())
            if max_nodes is None or (u < max_nodes and v < max_nodes):
                edges.append((u, v))
                edge_count += 1

    G.add_edges_from(edges)
    U_G = G.to_undirected()
    return G, U_G, id_to_name, name_to_id
```

Per motivi di performance, limitiamo la creazione del grafo settando il paramentro *max_lines* a 10.000.000 di righe

Output

```
Tempo di creazione dei grafi G e U(G): 78.70 secondi

Numero di nodi nel grafo: 753032
Numero di archi nel grafo: 10000000
```

1. ANALISI DELLE DISTRIBUZIONI DEI GRADI

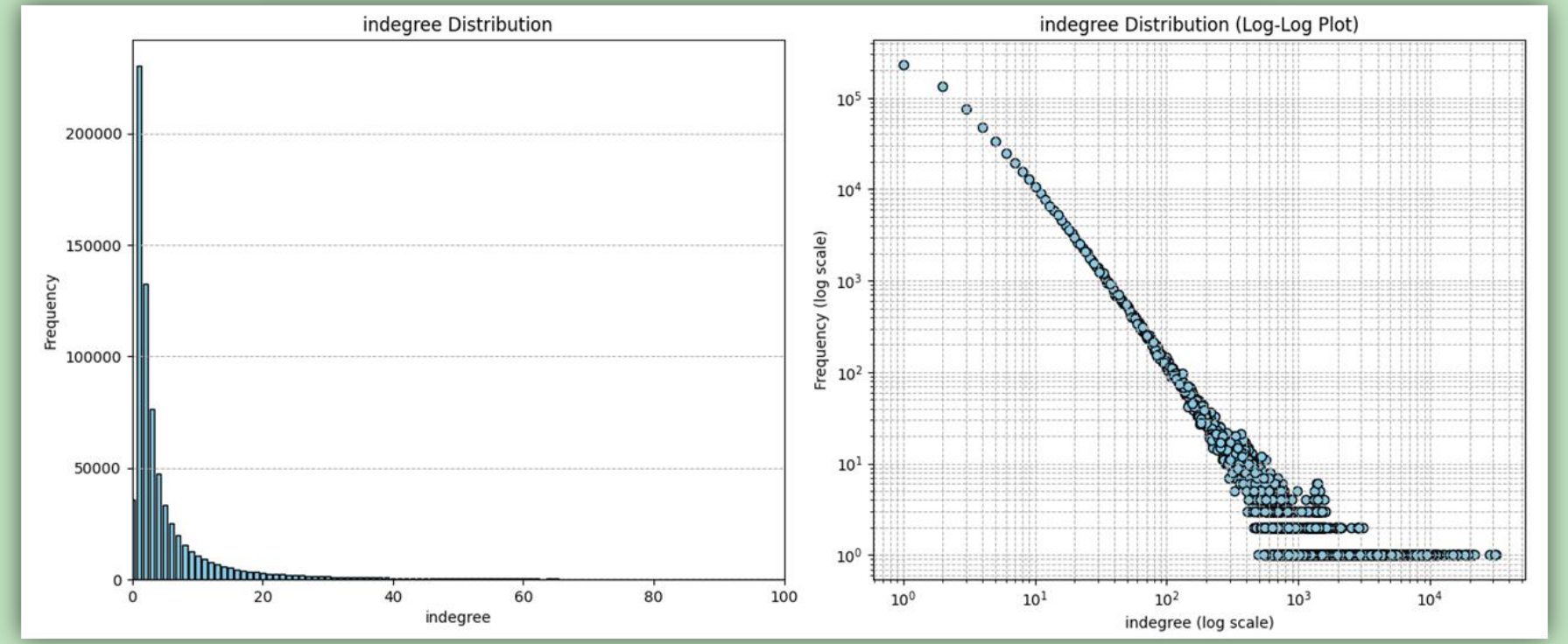
```python
def calculate_degree(graph, degree_type='total'):
    """
    Calcola l'indegree, l'outdegree o il grado totale per ciascun nodo nel grafo e restituisce un dizionario con i valori.

    Parametri:
    - graph: Il grafo su cui calcolare i gradi.
    - degree_type: Specifica se calcolare 'in', 'out' o 'total' degree (default è 'total' per grafi non diretti).

    Ritorna:
    - Dizionario con i gradi (indegree, outdegree o grado totale) per ciascun nodo.
    """
    degree_dict = {}

    if degree_type == 'out':
        for u, v in graph.edges():
            if u in degree_dict:
                degree_dict[u] += 1
            else:
                degree_dict[u] = 1
    elif degree_type == 'in':
        for u, v in graph.edges():
            if v in degree_dict:
                degree_dict[v] += 1
            else:
                degree_dict[v] = 1
    elif degree_type == 'total':
        for u, v in graph.edges():
            if u in degree_dict:
                degree_dict[u] += 1
            else:
                degree_dict[u] = 1
            if v in degree_dict:
                degree_dict[v] += 1
            else:
                degree_dict[v] = 1

    for node in graph.nodes():
        if node not in degree_dict:
            degree_dict[node] = 0

    return degree_dict
```

```python
def plot_degree_distribution(degree_dict, degree_type='out'):
    """
    Visualizza la distribuzione dei gradi (indegree, outdegree o totale) e la loro rappresentazione log-log
    fianco a fianco in due grafici.

    Parametri:
    - degree_dict: Dizionario con i valori dei gradi (indegree/outdegree/total).
    - degree_type: Specifica se stampare 'in', 'out' o 'total' degree (default 'out').
    """
    degree_count = {}
    for degree in degree_dict.values():
        degree_count[degree] = degree_count.get(degree, 0) + 1

    degrees = list(degree_count.keys())  # Gradi unici
    frequencies = [degree_count[degree] for degree in degrees]

    if degree_type == 'in':
        degree_name = "indegree"
    elif degree_type == 'out':
        degree_name = "outdegree"
    else:
        degree_name = "total degree"

    title = f'{degree_name} Distribution'

    fig, axs = plt.subplots(1, 2, figsize=(14, 6))

    # Grafico a barre della distribuzione dei gradi
    axs[0].bar(degrees, frequencies, color='skyblue', edgecolor='black')
    axs[0].set_xlabel(degree_name)
    axs[0].set_ylabel('Frequency')
    axs[0].set_title(title)
    axs[0].set_xlim([0, 100])
    axs[0].grid(True, which='both', axis='y', linestyle='--', linewidth=0.7)

    # Grafico log-log per la distribuzione dei gradi
    axs[1].scatter(degrees, frequencies, color='skyblue', edgecolor='black')
    axs[1].set_xscale('log')  # Scala logaritmica per l'asse x
    axs[1].set_yscale('log')  # Scala logaritmica per l'asse y
    axs[1].set_xlabel(f'{degree_name} (log scale)')
    axs[1].set_ylabel('Frequency (log scale)')
    axs[1].set_title(f'{degree_name} Distribution (Log-Log Plot)')
    axs[1].grid(True, which='both', linestyle='--', linewidth=0.7)

    plt.tight_layout()
    plt.show()
```

1.1 TROVARE LE TOP 10 PAGINE CON IL MAGGIOR NUMERO DI IN-DEGREE IN G

Utilizziamo le funzioni *calculate_degree* e *plot_degree_distribution* con parametro *degree_type = 'in'*

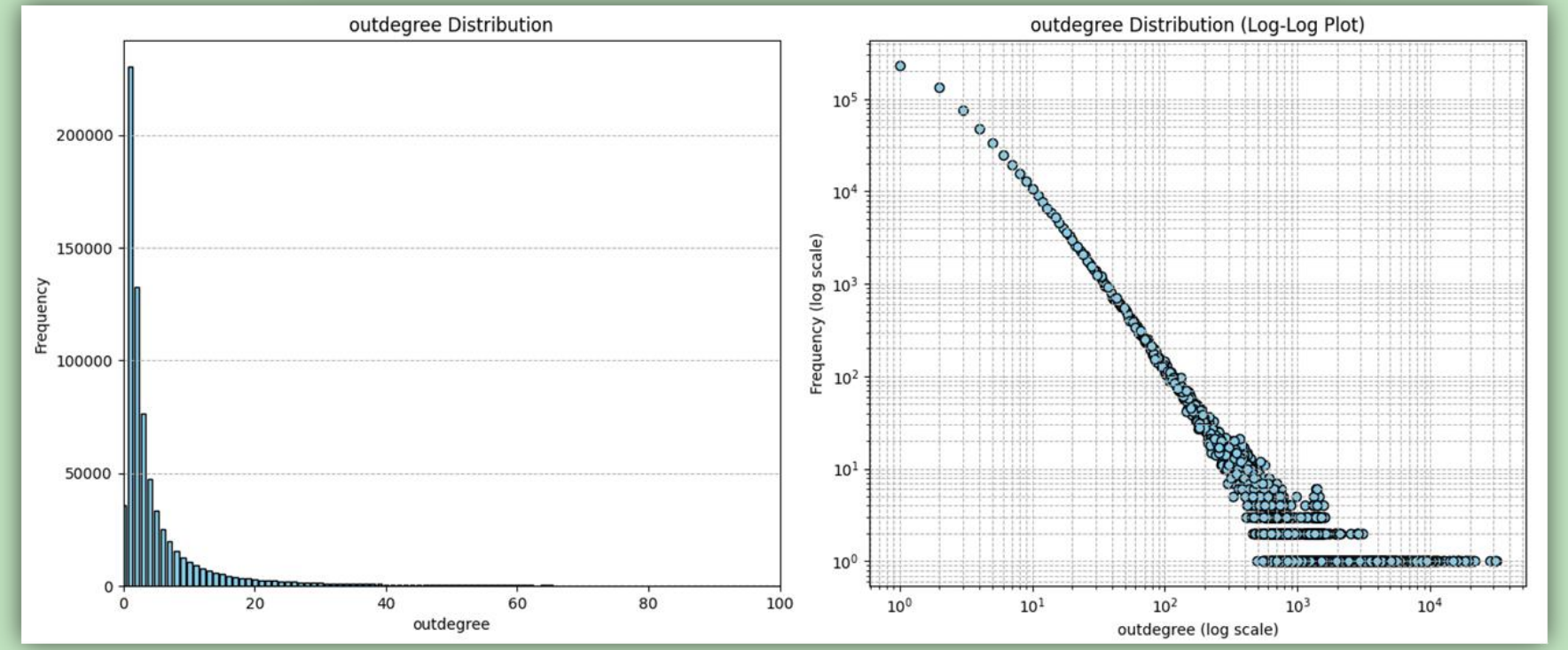
```
Tempo di esecuzione algoritmo per il calcolo della distribuzione degli indegree in G: 4.32 secondi

Prime 10 pagine con il maggior indegree:
1) Stati Uniti d'America, ID: 354122, Indegree: 31362
2) Comuni della Francia, ID: 2939, Indegree: 31205
3) Italia, ID: 396839, Indegree: 27957
4) 2007, ID: 396908, Indegree: 21686
5) 2006, ID: 396907, Indegree: 21075
6) 2008, ID: 396349, Indegree: 20507
7) 2005, ID: 396909, Indegree: 19955
8) 2004, ID: 396905, Indegree: 19767
9) 2009, ID: 396348, Indegree: 19476
10) Germania, ID: 353982, Indegree: 18575
```



1.2. TROVARE LE TOP 10 PAGINE CON IL MAGGIOR NUMERO DI OUT-DEGREE IN G

Utilizziamo le funzioni *calculate_degree* e *plot_degree_distribution* con parametro *degree_type = 'out'*

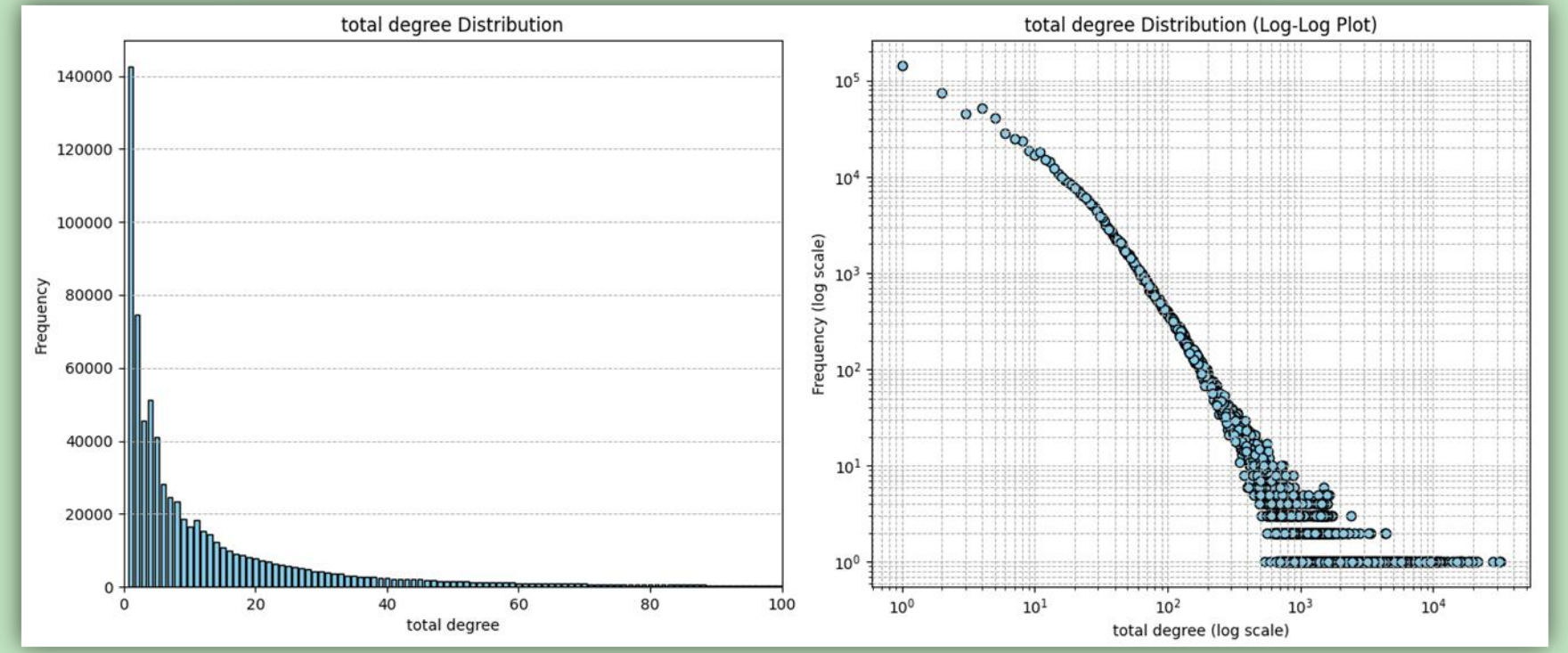
```
Tempo di esecuzione algoritmo per il calcolo della distribuzione degli outdegree in G: 2.34 secondi

Prime 10 pagine con il maggior outdegree:
1) Città dell'India, ID: 261576, Outdegree: 5212
2) Nati nel 1981, ID: 395575, Outdegree: 3256
3) Nati nel 1985, ID: 395611, Outdegree: 3250
4) Nati nel 1983, ID: 395769, Outdegree: 3235
5) Nati nel 1984, ID: 395650, Outdegree: 3231
6) Nati nel 1980, ID: 395546, Outdegree: 3120
7) Nati nel 1979, ID: 395354, Outdegree: 3007
8) Nati nel 1987, ID: 395324, Outdegree: 3004
9) Nati nel 1988, ID: 395593, Outdegree: 2909
10) Nati nel 1978, ID: 395685, Outdegree: 2821
```



1.3 ANALISI DELLE DISTRIBUZIONI DEI GRADI: TROVARE LE TOP 10 PAGINE CON IL MAGGIOR NUMERO DI GRADI TOTALI IN U(G)

Utilizziamo le funzioni *calculate_degree* e *plot_degree_distribution* con parametro *degree_type = 'total'*

```
Tempo di esecuzione algoritmo per il calcolo della distribuzione del degree in U(G): 8.88 secondi

Prime 10 pagine con il maggior total degree:
1) Stati Uniti d'America, ID: 354122, Total degree: 31800
2) Comuni della Francia, ID: 2939, Total degree: 31496
3) Italia, ID: 396839, Total degree: 27957
4) 2007, ID: 396908, Total degree: 21686
5) 2006, ID: 396907, Total degree: 21075
6) 2008, ID: 396349, Total degree: 20507
7) 2005, ID: 396909, Total degree: 19955
8) 2004, ID: 396905, Total degree: 19767
9) 2009, ID: 396348, Total degree: 19476
10) Germania, ID: 353982, Total degree: 18837
```



2. CALCOLO DEL DIAMETRO: CALCOLARE IL DIAMETRO DELLA COMPONENTE CONNESSA PIÙ GRANDE IN U(G) E RIPETERE L'ANALISI RIMUOVENDO I NODI CON "DISAMBIGUA" NEL NOME

- Trovare la componente connessa più grande (LCC): usiamo una BFS per trovare la LCC in un grafo
- Calcolo del diametro del grafo: l'algoritmo iFub calcola il diametro di un grafo attraverso BFS ed espansioni bidirezionali per ridurre significativamente il numero di nodi da visitare prima di trovare un percorso ottimale
- Rimozione dei nodi 'disambigua': filtra i nodi con il termine 'disambigua' e ricalcola il diametro
- Pipeline: combina queste operazioni per calcolare il diametro sia con che senza nodi 'disambigua', misurando il tempo di esecuzione

```python
def find_largest_connected_component(graph):
    """
    Trova la componente connessa più grande in un grafo non diretto usando BFS.
    Restituisce il sottografo della LCC.
    """
    def bfs_component(start_node, visited):
        queue = deque([start_node])
        component = set([start_node])

        while queue:
            node = queue.popleft()
            for neighbor in graph.neighbors(node):
                if neighbor not in visited:
                    visited.add(neighbor)
                    component.add(neighbor)
                    queue.append(neighbor)
        return component

    visited = set()
    largest_component = set()

    for node in graph.nodes():
        if node not in visited:
            visited.add(node)
            component = bfs_component(node, visited)
            if len(component) > len(largest_component):
                largest_component = component

    return graph.subgraph(largest_component).copy()
```

```python
def ifub_diameter(graph):
    """
    Calcola il diametro del grafo usando l'algoritmo iFub ottimizzato con BFS bidirezionale.
    """
    start_node = list(graph.nodes())[0]

    _, farthest_node = bfs_eccentricity(graph, start_node)

    lb, farthest_node = bfs_eccentricity(graph, farthest_node)
    ub = 2 * lb
    i = lb

    while ub > lb:
        fringe_nodes = bfs_fringe(graph, farthest_node, i)

        max_eccentricity_in_fringe = 0
        for node in fringe_nodes:
            eccentricity = bidirectional_bfs(graph, farthest_node, node)
            max_eccentricity_in_fringe = max(max_eccentricity_in_fringe, eccentricity)

        lb = max(lb, max_eccentricity_in_fringe)
        ub = 2 * (i - 1)
        i -= 1

    return lb
```

```python
def pipeline(graph, id_to_name):
    """
    Calcola il diametro del grafo con e senza nodi 'disambigua'.
    """
    start_time = time.time()
    print("Calcolo con nodi 'disambigua'...")
    largest_cc = find_largest_connected_component(graph)
    diameter = ifub_diameter(largest_cc)
    print(f"Diametro: {diameter}")
    end_time = time.time()
    print(f"Tempo di esecuzione: {end_time - start_time:.2f} secondi\n")

    start_time = time.time()
    print("Rimozione dei nodi 'disambigua' e ricalcolo...")
    graph_cleaned = remove_disambigua_nodes(graph.copy(), id_to_name)
    largest_cc = find_largest_connected_component(graph_cleaned)
    diameter = ifub_diameter(largest_cc)
    print(f'Diametro (senza nodi disambigua): {diameter}')
    end_time = time.time()
    print(f"Tempo di esecuzione: {end_time - start_time:.2f} secondi\n")
```

Output

```
Calcolo con nodi 'disambigua'...
Diametro: 12
Tempo di esecuzione: 447.84 secondi

Rimozione dei nodi 'disambigua' e ricalcolo...
Diametro (senza nodi disambigua): 12
Tempo di esecuzione: 722.30 secondi
```

3. RICERCA DI CLIQUES MASSIMALI: TROVARE DUE CLIQUES MASSIMALI CON ALMENO 3 NODI

- Algoritmo Bron-Kerbosch per trovare le cliques massimali all'interno di un grafo. Una clique è un sottoinsieme di nodi in cui ogni nodo è collegato a tutti gli altri.
- L'algoritmo implementato ha dei vincoli che ci consentono di: limitare il numero di cliques trovate (2 di default); considerare soltanto le cliques non banali (con almeno 3 nodi). Inoltre, seleziona un nodo pivot per ridurre il numero di ricorsioni, ottimizzando la ricerca delle cliques

```python
def bron_kerbosch(R, P, X, graph, maximal_cliques, max_cliques_needed=2):
    """
    Algoritmo Bron-Kerbosch modificato per trovare cliques massimali.

    Parametri:
    - R: Insieme dei nodi che sono già inclusi nella clique corrente.
    - P: Insieme dei nodi candidati che possono ancora essere aggiunti alla clique.
    - X: Insieme dei nodi che non possono più essere inclusi nella clique (già processati).
    - graph: Il grafo su cui viene eseguito l'algoritmo.
    - maximal_cliques: Lista dove vengono salvate le cliques massimali trovate.
    - max_cliques_needed: Numero massimo di cliques da trovare (valore di default 2).

    Logica:
    - L'algoritmo continua finché non ci sono più candidati in P o nodi in X.
    - Se la clique formata (R) ha almeno 3 nodi ed è massimale, la aggiunge a `maximal_cliques`.
    - Il pivot viene scelto da P U X e i nodi di P che non sono vicini del pivot vengono processati.
    - Ricorsivamente, vengono trovate tutte le cliques massimali.
    """

    if not P and not X:
        if len(R) >= 3:
            maximal_cliques.append(R)
        return

    if len(maximal_cliques) >= max_cliques_needed:
        return

    pivot = next(iter(P.union(X)), None)  # Prende un nodo qualsiasi da P U X
    if pivot is None:
        return

    for node in P - set(graph.neighbors(pivot)):
        bron_kerbosch(
            R.union([node]),
            P.intersection(graph.neighbors(node)),
            X.intersection(graph.neighbors(node)),
            graph, maximal_cliques, max_cliques_needed
        )
        P.remove(node)
        X.add(node)
```

```python
def find_maximal_cliques_bron_kerbosch(graph, id_to_name, max_cliques_needed=2):
    """
    Funzione che avvia l'algoritmo Bron-Kerbosch per trovare più cliques massimali non banali.

    Parametri:
    - graph: Il grafo in cui si vogliono trovare le cliques massimali.
    - id_to_name: Dizionario che mappa gli ID dei nodi ai nomi per la stampa.
    - max_cliques_needed: Numero massimo di cliques massimali da trovare.

    Ritorna:
    - Una lista con le cliques massimali trovate.
    """
    R = set()
    P = set(graph.nodes())
    X = set()

    maximal_cliques = []

    bron_kerbosch(R, P, X, graph, maximal_cliques, max_cliques_needed)

    for idx, clique in enumerate(maximal_cliques, start=1):
        clique_names = [id_to_name[node] for node in clique]
        print(f"Clique massimale {idx}: {clique_names}")

    return maximal_cliques
```

Output

```
Clique massimale 1: ['Cabra de Mora', 'Spagna', 'Aragona', 'Formiche Alto', 'Comunità autonome della Spagna', 'El Castellar']
Clique massimale 2: ['Cabra de Mora', 'El Castellar', 'Alcalá de la Selva']

Tempo di esecuzione: 10.98 secondi
```

CREAZIONE DEL GRAFO WIKIPEDIA CON IGRAPH

```python
def create_graph_from_files_igraph(ids_file_path, arcs_file_path, max_lines=None, max_nodes=None, max_edges=None):
    """
    Crea il grafo diretto G utilizzando un limite opzionale per max_lines righe del file .arcs.
    Può anche limitare il numero di nodi e archi nel grafo per migliorare le performance.
    Ritorna G (grafo diretto) e U_G (versione non diretta di G).

    - max_lines: Numero massimo di righe da processare (opzionale).
    - max_nodes: Numero massimo di nodi da aggiungere (opzionale).
    - max_edges: Numero massimo di archi da aggiungere (opzionale).
    """
    G = ig.Graph(directed=True)

    id_to_name = {}
    name_to_id = {}

    with open(ids_file_path, 'r', encoding='utf-8') as ids_file:
        for i, line in enumerate(ids_file):
            if max_nodes and i >= max_nodes:
                break
            name = line.strip()
            id_to_name[i] = name
            name_to_id[name] = i
            G.add_vertex(name=name)

    edges = []
    edge_count = 0
    with open(arcs_file_path, 'r') as arcs_file:
        for i, line in enumerate(arcs_file):
            if max_lines and i >= max_lines:
                break
            if max_edges and edge_count >= max_edges:
                break
            u, v = map(int, line.strip().split())
            if u < len(G.vs) and v < len(G.vs):
                edges.append((u, v))
                edge_count += 1

    G.add_edges(edges)
    U_G = G.as_undirected()

    return G, U_G, id_to_name, name_to_id
```

Per motivi di performance, limitiamo la creazione del grafo settando il paramentro *max_lines* a 10.000.000 di righe e *max_nodes* a 75.000

Output

```
Tempo di creazione dei grafi G e U(G): 18.11 secondi

Numero di nodi nel grafo: 75000
Numero di archi nel grafo: 444147
```

4. ANALISI DELLE DISTRIBUZIONI DEI GRADI

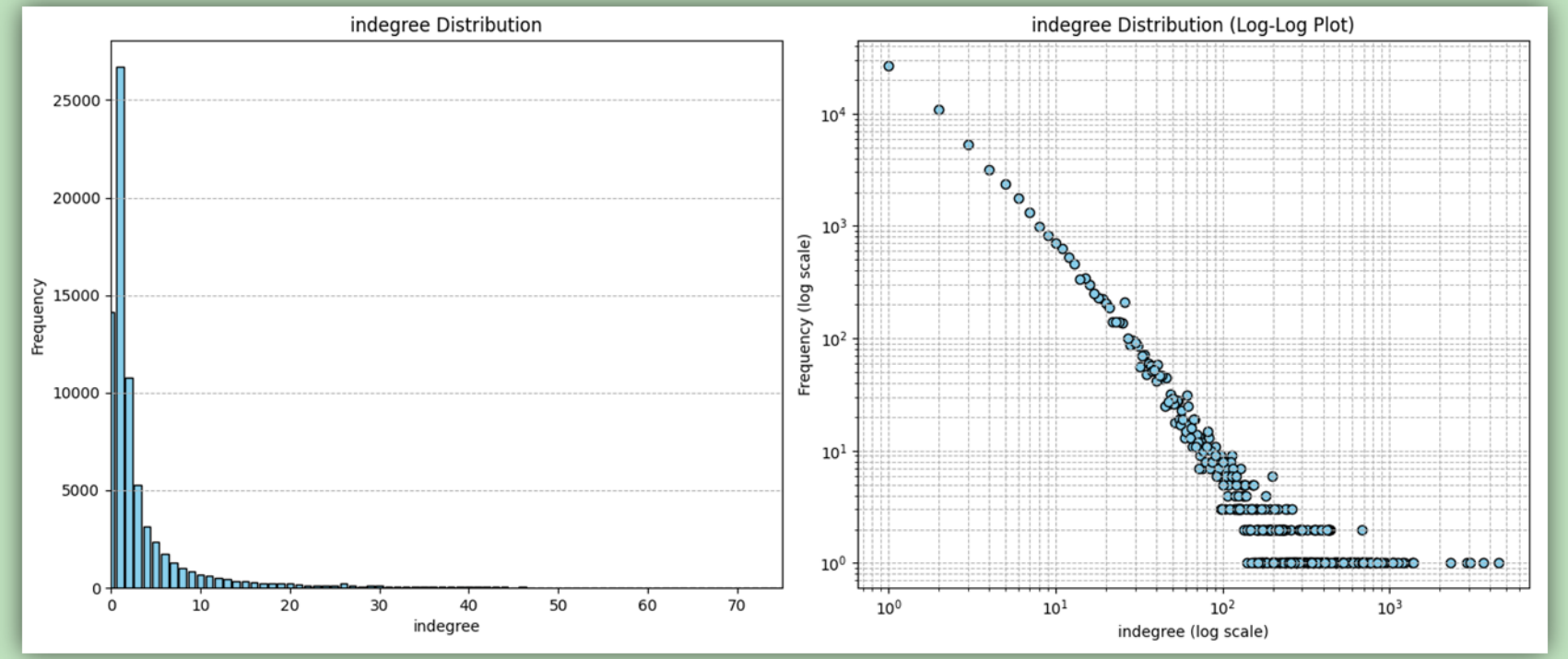
```python
def calculate_degree_igraph(graph, degree_type='total'):
    """
    Calcola l'indegree, l'outdegree o il grado totale per ciascun nodo nel grafo e restituisce un dizionario con i valori.

    Parametri:
    - graph: Il grafo su cui calcolare i gradi (usando igraph).
    - degree_type: Specifica se calcolare 'in', 'out' o 'total' degree (default è 'total' per grafi non diretti).

    Ritorna:
    - Dizionario con i gradi (indegree, outdegree o grado totale) per ciascun nodo.
    """
    if degree_type == 'out':
        degrees = graph.outdegree()
    elif degree_type == 'in':
        degrees = graph.indegree()
    elif degree_type == 'total':
        degrees = graph.degree()

    degree_dict = {v.index: degree for v, degree in zip(graph.vs, degrees)}

    return degree_dict
```

```python
def plot_degree_distribution_igraph(degree_dict, degree_type='out'):
    """
    Visualizza la distribuzione dei gradi (indegree o outdegree) e la loro rappresentazione log-log
    fianco a fianco in due grafici, utilizzando igraph.

    Parametri:
    - degree_dict: Dizionario con i valori dei gradi (indegree/outdegree).
    - degree_type: Specifica se stampare 'in' o 'out' degree (default 'out').
    """
    degree_count = {}
    for degree in degree_dict.values():
        degree_count[degree] = degree_count.get(degree, 0) + 1

    degrees = list(degree_count.keys())
    frequencies = [degree_count[degree] for degree in degrees]

    if degree_type == 'in':
        degree_name = "indegree"
    elif degree_type == 'out':
        degree_name = "outdegree"
    else:
        degree_name = "total degree"
    title = f'{degree_name} Distribution'

    fig, axs = plt.subplots(1, 2, figsize=(14, 6))

    # Grafico a barre della distribuzione dei gradi
    axs[0].bar(degrees, frequencies, color='skyblue', edgecolor='black')
    axs[0].set_xlabel(degree_name)
    axs[0].set_ylabel('Frequency')
    axs[0].set_title(title)
    axs[0].set_xlim([0, 75])
    axs[0].grid(True, which='both', axis='y', linestyle='--', linewidth=0.7)

    # Grafico log-log della distribuzione dei gradi
    axs[1].scatter(degrees, frequencies, color='skyblue', edgecolor='black')
    axs[1].set_xscale('log')
    axs[1].set_yscale('log')
    axs[1].set_xlabel(f'{degree_name} (log scale)')
    axs[1].set_ylabel('Frequency (log scale)')
    axs[1].set_title(f'{degree_name} Distribution (Log-Log Plot)')
    axs[1].grid(True, which='both', linestyle='--', linewidth=0.7)

    plt.tight_layout()
    plt.show()
```

4.1 ANALISI DELLE DISTRIBUZIONI DEI GRADI: TROVARE LE TOP 10 PAGINE CON IL MAGGIOR NUMERO DI IN-DEGREE IN G

Utilizziamo le funzioni
calculate_degree_igraph e
plot_degree_distribution_igraph
con parametro *degree_type = 'in'*

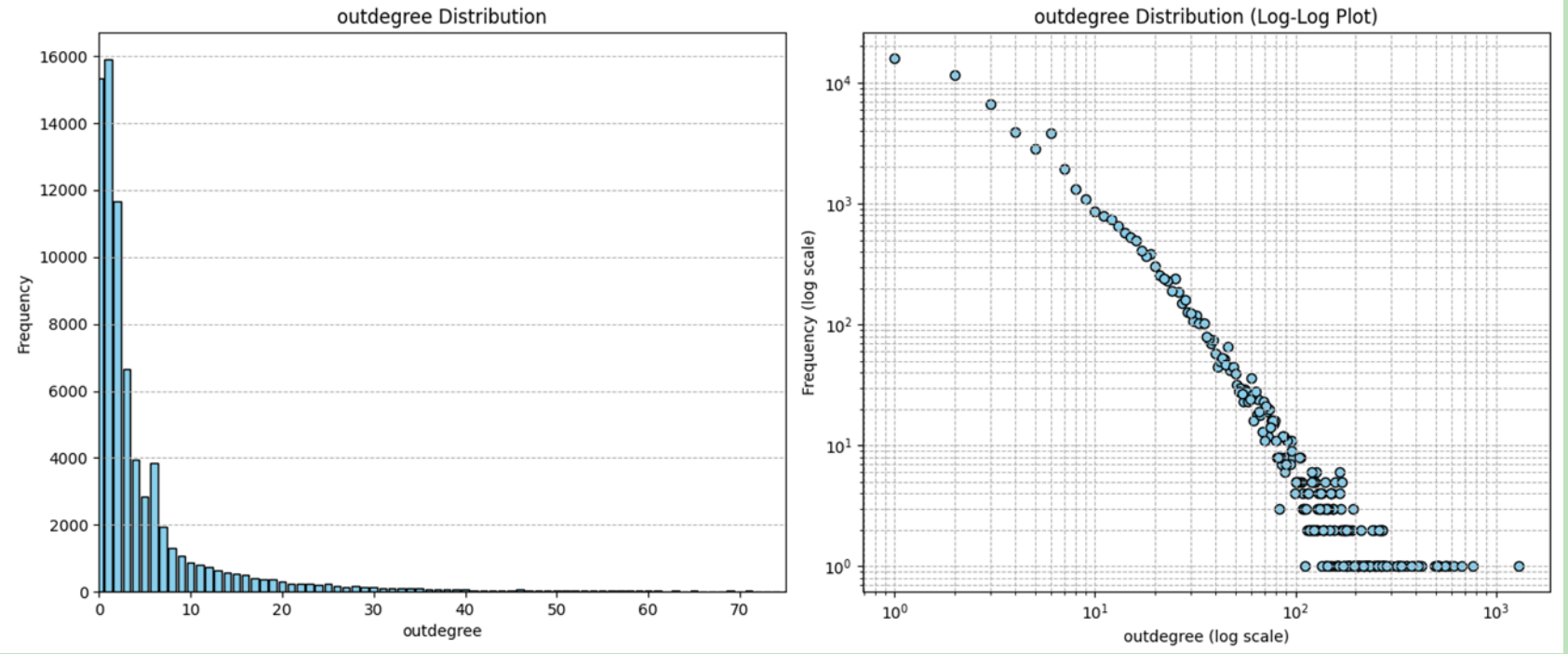
```
Tempo di esecuzione algoritmo per il calcolo della distribuzione degli indegree in G: 0.17 secondi

Prime 10 pagine con il maggior indegree:
1) Mexico, ID: 53706, Indegree: 4508
2) Iran, ID: 5100, Indegree: 3665
3) Mexico City, ID: 53412, Indegree: 3034
4) Romanization, ID: 5090, Indegree: 2912
5) Ukraine, ID: 74307, Indegree: 2307
6) Institutional Revolutionary Party, ID: 54489, Indegree: 1388
7) Kohgiluyeh and Boyer-Ahmad province, ID: 4877, Indegree: 1366
8) Veracruz, ID: 53068, Indegree: 1229
9) Nicaragua, ID: 59551, Indegree: 1194
10) Verkhovna Rada, ID: 72544, Indegree: 1121
```



4.1 ANALISI DELLE DISTRIBUZIONI DEI GRADI: TROVARE LE TOP 10 PAGINE CON IL MAGGIOR NUMERO DI OUT-DEGREE IN G

Utilizziamo le funzioni *calculate_degree_igraph* e *plot_degree_distribution_igraph* con parametro *degree_type = 'out'*

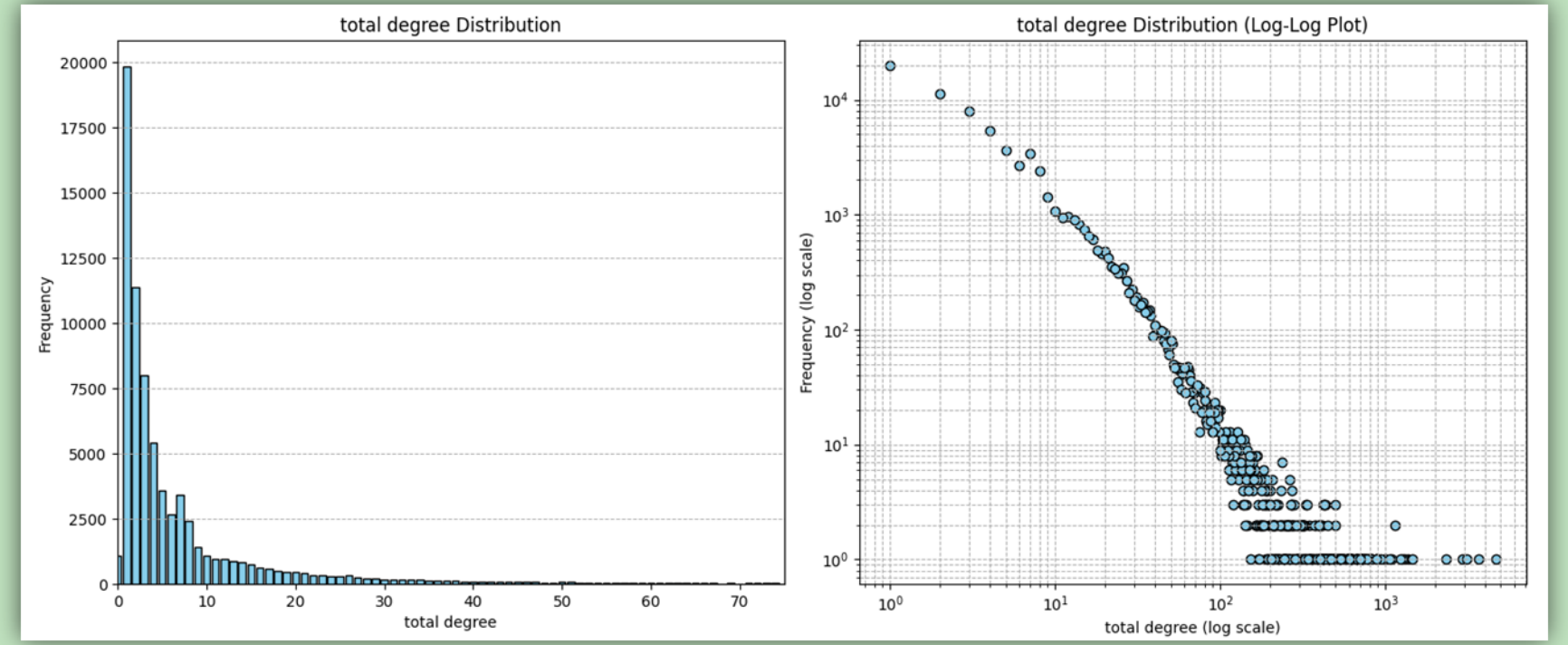
```
Tempo di esecuzione algoritmo per il calcolo della distribuzione degli outdegree in G: 0.17 secondi

Prime 10 pagine con il maggior outdegree:
1) List of cities, towns and villages in Kohgiluyeh and Boyer-Ahmad Province, ID: 4878, Outdegree: 1301
2) List of cities, towns and villages in Chaharmahal and Bakhtiari Province, ID: 3275, Outdegree: 771
3) List of places in Mexico named after people, ID: 53177, Outdegree: 676
4) List of Streptomyces species, ID: 42897, Outdegree: 617
5) Municipalities of Armenia, ID: 35257, Outdegree: 596
6) Communes of the Doubs department, ID: 24565, Outdegree: 567
7) List of populated places in Elazığ Province, ID: 18799, Outdegree: 560
8) Index of Nicaragua-related articles, ID: 59596, Outdegree: 556
9) Index of Mexico-related articles, ID: 53099, Outdegree: 523
10) Communes of the Calvados department, ID: 15243, Outdegree: 517
```



4.1 ANALISI DELLE DISTRIBUZIONI DEI GRADI: TROVARE LE TOP 10 PAGINE CON IL MAGGIOR NUMERO DI GRADI TOTALI IN U(G)

Utilizziamo le funzioni
calculate_degree_igraph e
plot_degree_distribution_igraph
con parametro *degree_type = 'total'*

```
Tempo di esecuzione algoritmo per il calcolo della distribuzione del degree in U(G): 0.18 secondi

Prime 10 pagine con il maggior total degree:
1) Mexico, ID: 53706, Total degree: 4680
2) Iran, ID: 5100, Total degree: 3666
3) Mexico City, ID: 53412, Total degree: 3089
4) Romanization, ID: 5090, Total degree: 2912
5) Ukraine, ID: 74307, Total degree: 2343
6) Institutional Revolutionary Party, ID: 54489, Total degree: 1450
7) Kohgiluyeh and Boyer-Ahmad province, ID: 4877, Total degree: 1368
8) List of cities, towns and villages in Kohgiluyeh and Boyer-Ahmad Province, ID: 4878, Total degree: 1301
9) Veracruz, ID: 53068, Total degree: 1278
10) Nicaragua, ID: 59551, Total degree: 1224
```



4.1 CALCOLO DEL DIAMETRO: CALCOLARE IL DIAMETRO DELLA COMPONENTE CONNESSA PIÙ GRANDE IN U(G) E RIPETERE L'ANALISI RIMUOVENDO I NODI CON "DISAMBIGUA" NEL NOME

- Trovare la componente connessa più grande (LCC): usiamo il metodo *connected_components* fornito dalla libreria e prendiamo soltanto la più grande
- Calcolo del diametro del grafo: l'algoritmo iFub calcola il diametro del grafo, ottimizzando il processo usando BFS bidirezionale
- Rimozione dei nodi 'disambigua': filtra i nodi con il termine 'disambigua' e ricalcola il diametro
- Pipeline: combina queste operazioni per calcolare il diametro sia con che senza nodi 'disambigua', misurando il tempo di esecuzione

```python
def ifub_diameter_igraph(graph):
    """
    Calcola il diametro del grafo usando l'algoritmo iFub ottimizzato con BFS bidirezionale.
    """
    start_node = 0

    _, farthest_node = bfs_eccentricity_igraph(graph, start_node)

    lb, farthest_node = bfs_eccentricity_igraph(graph, farthest_node)
    ub = 2 * lb
    i = lb

    while ub > lb:
        fringe_nodes = bfs_fringe_igraph(graph, farthest_node, i)

        max_eccentricity_in_fringe = 0
        for node in fringe_nodes:
            eccentricity = bidirectional_bfs_igraph(graph, farthest_node, node)
            max_eccentricity_in_fringe = max(max_eccentricity_in_fringe, eccentricity)

        lb = max(lb, max_eccentricity_in_fringe)
        ub = 2 * (i - 1)
        i -= 1

    return lb
```

```python
def find_largest_connected_component_igraph(graph):
    """
    Trova la componente connessa più grande in un grafo non diretto usando igraph.
    Restituisce il sottografo della LCC.
    """
    components = graph.connected_components(mode="WEAK")
    largest_component = components.giant()
    return largest_component
```

```python
def pipeline_igraph(graph, id_to_name):
    """
    Calcola il diametro del grafo con e senza nodi 'disambigua' in igraph.
    """
    start_time = time.time()
    print("Calcolo con nodi 'disambigua'...")
    largest_cc = find_largest_connected_component_igraph(graph)
    diameter = ifub_diameter_igraph(largest_cc)
    print(f"Diametro: {diameter}")
    end_time = time.time()
    print(f"Tempo di esecuzione: {end_time - start_time:.2f} secondi\n")

    start_time = time.time()
    print("Rimozione dei nodi 'disambigua' e ricalcolo...")
    graph_cleaned = remove_disambigua_nodes_igraph(graph.copy(), id_to_name)
    largest_cc = find_largest_connected_component_igraph(graph_cleaned)
    diameter = ifub_diameter_igraph(largest_cc)
    print(f'Diametro (senza nodi disambigua): {diameter}')
    end_time = time.time()
    print(f"Tempo di esecuzione: {end_time - start_time:.2f} secondi
```

Output

```
Calcolo con nodi 'disambigua'...
Diametro: 34
Tempo di esecuzione: 557.55 secondi

Rimozione dei nodi 'disambigua' e ricalcolo...
Diametro (senza nodi disambigua): 37
Tempo di esecuzione: 1167.92 secondi
```

4.1 RICERCA DI CLIQUES MASSIMALI: TROVARE DUE CLIQUES MASSIMALI CON ALMENO 3 NODI

- Algoritmo Bron-Kerbosch per trovare le cliques massimali all'interno di un grafo.
- Una clique è un sottoinsieme di nodi in cui ogni nodo è collegato a tutti gli altri.
- L'algoritmo implementato ha dei vincoli che ci consentono di:
- limitare il numero di cliques trovate (2 di default)
- considerare soltanto le cliques non banali (con almeno 3 nodi)
- Inoltre, seleziona un nodo pivot per ridurre il numero di ricorsioni, ottimizzando la ricerca delle cliques.

```python
def bron_kerbosch_igraph(R, P, X, graph, maximal_cliques, max_cliques_needed=2):
    """
    Algoritmo Bron-Kerbosch modificato per trovare cliques massimali utilizzando igraph.

    Parametri:
    - R: Insieme dei nodi che sono già inclusi nella clique corrente.
    - P: Insieme dei nodi candidati che possono ancora essere aggiunti alla clique.
    - X: Insieme dei nodi che non possono più essere inclusi nella clique (già processati).
    - graph: Il grafo su cui viene eseguito l'algoritmo (di tipo igraph).
    - maximal_cliques: Lista dove vengono salvate le cliques massimali trovate.
    - max_cliques_needed: Numero massimo di cliques da trovare (valore di default 2).
    """
    if not P and not X:
        if len(R) >= 3:
            maximal_cliques.append(R)
        return

    if len(maximal_cliques) >= max_cliques_needed:
        return

    pivot = next(iter(P.union(X)), None)
    if pivot is None:
        return

    for node in P - set(graph.neighbors(pivot)):
        bron_kerbosch_igraph(
            R.union([node]),
            P.intersection(set(graph.neighbors(node))),
            X.intersection(set(graph.neighbors(node))),
            graph, maximal_cliques, max_cliques_needed
        )
        P.remove(node)
        X.add(node)
```

```python
def find_maximal_cliques_bron_kerbosch_igraph(graph, id_to_name, max_cliques_needed=2):
    """
    Funzione che avvia l'algoritmo Bron-Kerbosch per trovare più cliques massimali non banali.

    Parametri:
    - graph: Il grafo in cui si vogliono trovare le cliques massimali (di tipo igraph).
    - id_to_name: Dizionario che mappa gli ID dei nodi ai nomi per la stampa.
    - max_cliques_needed: Numero massimo di cliques massimali da trovare.

    Ritorna:
    - Una lista con le cliques massimali trovate.
    """
    R = set()
    P = set(graph.vs.indices)
    X = set()

    maximal_cliques = []

    bron_kerbosch_igraph(R, P, X, graph, maximal_cliques, max_cliques_needed)

    for idx, clique in enumerate(maximal_cliques, start=1):
        clique_names = [id_to_name[node] for node in clique]
        print(f"Clique massimale {idx}: {clique_names}")

    return maximal_cliques
```

Output

```
Clique massimale 1: ['La Cultura station', 'San Borja Sur station', 'Miguel Grau station', 'Atocongo station', 'List of Lima Metro stations', 'Lima Metro']
Clique massimale 2: ['La Cultura station', 'Miguel Grau station', 'Atocongo station', 'List of Lima Metro stations', 'Lima Metro', 'Arriola station']

Tempo di esecuzione: 0.54 secondi
```

4.2 RICERCA DI CLIQUES MASSIMALI: TROVARE TUTTE LE CLIQUES MASSIMALI SU UN SOTTOGRAFO CASUALE DI 100 NODI

- Algoritmo Bron-Kerbosch per trovare le cliques massimali all'interno di un grafo con un unico vincolo, quello di considerare soltanto le cliques non banali (con almeno 3 nodi).
- Creiamo un sottografo di 100 elementi utilizzando la funzione 'create_graph_from_files_igraph' vista in precedenza vincolando il parametro 'max_nodes'.

```python
def bron_kerbosch(R, P, X, graph, maximal_cliques):
    """
    Algoritmo Bron-Kerbosch per trovare cliques massimali nel grafo, utilizzando igraph.

    Parametri:
    - R: Insieme dei nodi già inclusi nella clique corrente.
    - P: Insieme dei nodi candidati che possono essere aggiunti alla clique.
    - X: Insieme dei nodi che sono stati processati e non possono essere aggiunti alla clique.
    - graph: Grafo igraph su cui si esegue l'algoritmo.
    - maximal_cliques: Lista per salvare tutte le cliques massimali trovate.

    Logica:
    - Se P e X sono vuoti, significa che R è una clique massimale.
    - La clique massimale viene aggiunta alla lista solo se contiene almeno 3 nodi.
    - Il nodo corrente viene aggiunto a R, mentre P e X sono aggiornati per includere solo i vicini del nodo corrente.
    """
    if not P and not X:
        if len(R) >= 3:
            maximal_cliques.append(R)
        return

    for node in list(P):
        bron_kerbosch(
            R.union([node]),
            P.intersection(set(graph.neighbors(node))),
            X.intersection(set(graph.neighbors(node))),
            graph, maximal_cliques
        )
        P.remove(node)
        X.add(node)
```

```python
def find_all_maximal_cliques_bron_kerbosch(graph, id_to_name):
    """
    Trova tutte le cliques massimali non banali nel grafo utilizzando l'algoritmo Bron-Kerbosch.

    Parametri:
    - graph: Il grafo igraph su cui si cerca.
    - id_to_name: Dizionario per mappare gli ID dei nodi ai nomi delle pagine.

    Ritorna:
    - Lista di tutte le cliques massimali trovate.
    """
    R = set()
    P = set(range(graph.vcount()))
    X = set()

    maximal_cliques = []

    bron_kerbosch(R, P, X, graph, maximal_cliques)

    for idx, clique in enumerate(maximal_cliques, start=1):
        clique_names = [id_to_name[node] for node in clique]
        print(f"Clique massimale {idx}: {clique_names}")

    return maximal_cliques
```

Output

```
Clique massimale 1: ['La Cultura station', 'San Borja Sur station', 'Miguel Grau station', 'Atocongo station', 'List of Lima Metro stations', 'Lima Metro']
Clique massimale 2: ['La Cultura station', 'Miguel Grau station', 'Atocongo station', 'List of Lima Metro stations', 'Lima Metro', 'Arriola station']
Clique massimale 3: ['San Borja Sur station', 'Angamos station', 'Atocongo station', 'Miguel Grau station', 'List of Lima Metro stations', 'Lima Metro']
Clique massimale 4: ['San Juan station', 'Atocongo station', 'Miguel Grau station', 'List of Lima Metro stations', 'Lima Metro', 'María Auxiliadora station', 'Villa El Salvador station']
Clique massimale 5: ['Miguel Grau station', 'Atocongo station', 'Villa María station', 'Pumacahua station', 'List of Lima Metro stations', 'Lima Metro', 'Villa El Salvador station']
Clique massimale 6: ['Miguel Grau station', 'Atocongo station', 'Villa María station', 'List of Lima Metro stations', 'Lima Metro', 'María Auxiliadora station', 'Villa El Salvador station']
Clique massimale 7: ['Miguel Grau station', 'Atocongo station', 'List of Lima Metro stations', 'Pumacahua station', 'Parque Industrial station', 'Lima Metro', 'Villa El Salvador station']
Clique massimale 8: ['Angamos station', 'Miguel Grau station', 'Atocongo station', 'List of Lima Metro stations', 'Cabitos station', 'Lima Metro']
Clique massimale 9: ['Miguel Grau station', 'Atocongo station', 'List of Lima Metro stations', 'Cabitos station', 'Lima Metro', 'Ayacucho station']
Clique massimale 10: ['Miguel Grau station', 'Atocongo station', 'List of Lima Metro stations', 'Lima Metro', 'Ayacucho station', 'Jorge Chávez station']
Clique massimale 11: ['Miguel Grau station', 'Atocongo station', 'List of Lima Metro stations', 'Lima Metro', 'Gamarra station', 'Arriola station']
Clique massimale 12: ['Miguel Grau station', 'List of Lima Metro stations', 'Lima Metro', 'Los Jardines station (Lima Metro)', 'Pirámide del Sol station', 'Bayóvar station']
Clique massimale 13: ['Los Postes station', 'Miguel Grau station', 'List of Lima Metro stations', 'Lima Metro', 'Los Jardines station (Lima Metro)', 'Bayóvar station']
Clique massimale 14: ['Miguel Grau station', 'List of Lima Metro stations', 'Lima Metro', 'Caja de Agua station', 'Pirámide del Sol station', 'Bayóvar station']
Clique massimale 15: ['Miguel Grau station', 'List of Lima Metro stations', 'Lima Metro', 'Presbítero Maestro station', 'Caja de Agua station', 'Bayóvar station']
Clique massimale 16: ['Miguel Grau station', 'List of Lima Metro stations', 'Lima Metro', 'Bayóvar station', 'Santa Rosa station (Lima Metro)', 'San Martín station']
Clique massimale 17: ['San Carlos station (Lima Metro)', 'Miguel Grau station', 'List of Lima Metro stations', 'Lima Metro', 'Bayóvar station', 'San Martín station']
Clique massimale 18: ['San Carlos station (Lima Metro)', 'Los Postes station', 'Miguel Grau station', 'List of Lima Metro stations', 'Lima Metro', 'Bayóvar station']
Clique massimale 19: ['Miguel Grau station', 'List of Lima Metro stations', 'Lima Metro', 'Presbítero Maestro station', 'El Ángel station', 'Bayóvar station']
Clique massimale 20: ['Saprosecans bialoviensis', 'Halolaelapidae', 'Saprosecans']
Clique massimale 21: ['Saprosecans baloghi', 'Saprosecans', 'Halolaelapidae']
Clique massimale 22: ['Halolaelapidae', 'Leitneria pugio', 'Leitneria (mite)']
Clique massimale 23: ['Halolaelapidae', 'Leitneria granulatus', 'Leitneria (mite)']
```

4.3 RICERCA DI CLIQUES MASSIMALI: TROVARE LA CLIQUE MASSIMALE CON IL MAGGIOR NUMERO DI NODI

- Algoritmo Bron-Kerbosch per trovare le cliques massimali all'interno di un grafo con un unico vincolo, quello di considerare soltanto le cliques non banali (con almeno 3 nodi).
- Creiamo un sottografo di 100 elementi utilizzando la funzione 'create_graph_from_files_igraph' vista in precedenza vincolando il parametro 'max_nodes'.
- Dopo aver trovato tutte le cliques massimali utilizziamo la funzione `max()` per trovare la clique con il numero massimo di nodi

```python
def find_largest_maximal_clique_bron_kerbosch(graph, id_to_name):
    """
    Trova la clique massimale più grande nel grafo utilizzando l'algoritmo Bron-Kerbosch.

    Parametri:
    - graph: Il grafo igraph su cui viene eseguito l'algoritmo per trovare la clique massimale più grande.
    - id_to_name: Dizionario che mappa gli ID dei nodi ai nomi per la stampa.

    Ritorna:
    - La clique massimale più grande trovata.
    """
    R = set()
    P = set(range(graph.vcount()))
    X = set()

    maximal_cliques = []

    bron_kerbosch(R, P, X, graph, maximal_cliques)

    largest_clique = max(maximal_cliques, key=len)

    clique_names = [id_to_name[node] for node in largest_clique]
    print(f"Clique massimale più grande: {clique_names}")

    return largest_clique
```

Output

```
Clique massimale più grande: ['San Juan station', 'Atocongo station', 'Miguel Grau station', 'List of Lima Metro stations', 'Lima Metro', 'María Auxiliadora station', 'Villa El Salvador station']

Tempo di esecuzione: 0.01 secondi
```