AIM OF THE PROJECT: Create a recommendation system for TV series using the PageRank algorithm involves several key steps.

Use of PageRank Algorithm: The goal is to use the PageRank algorithm to identify the most influential genres and then leverage those scores to recommend series based on genre relevance.

STEPS INVOLVED IN THIS PROJECT:

Step 1: Loading the Dataset

Loaded the TV series dataset in the format:

	Series Title	Release Year	Runtime	Genre	Rating	Cast	Synopsis
0	Wednesday	(2022–)	45 min	Comedy, Crime, Fantasy	8.2	Jenna Ortega, Hunter Doohan, Percy Hynes White...	Follows Wednesday Addams' years as a student, ...

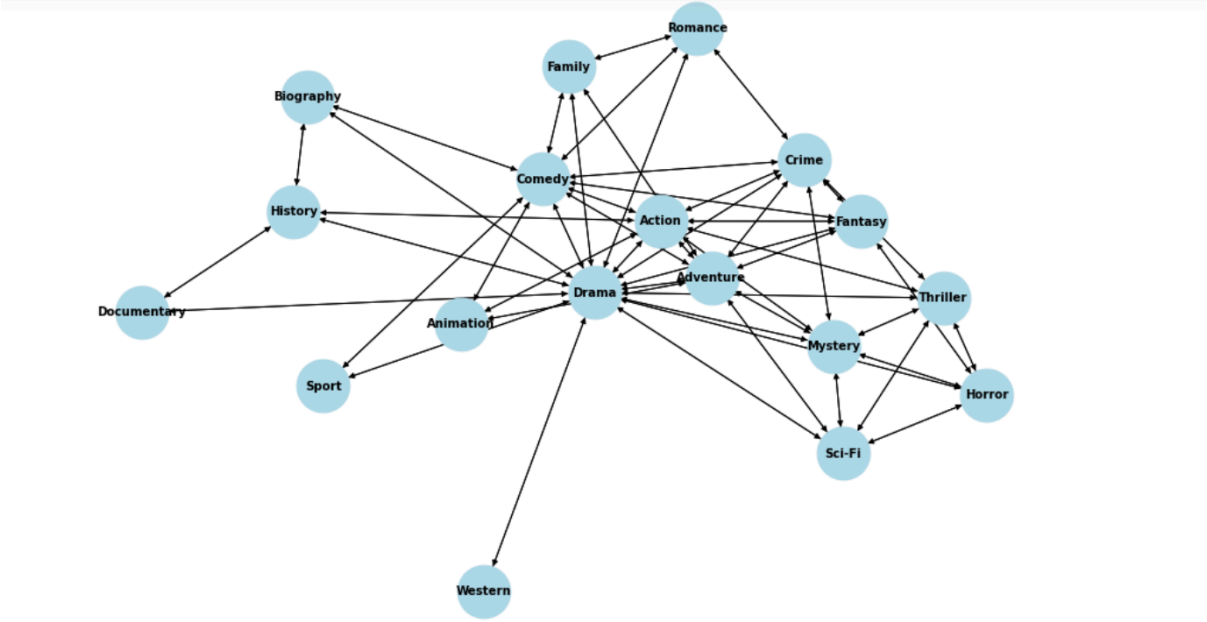
Step 2: Created a Graph Structure

- **Built a graph** where:
  - **Nodes** are genres.
  - **Edges** represent connections between genres that co-occur in the same series.
- **Added edges** by iterating over the dataset:
  - For each series, split the genre string into a list of genres.
  - Connect each genre to others that appear in the same series, forming a web of co-occurrences.

In [97]:
```
import itertools
genre_graph = {}

for genres in df['Genre']:
    genre_list = genres.split(", ")
    # preprocessing

    # EXPLANATION
    # Create edges between genres in the same series
    for genre1, genre2 in itertools.combinations(genre_list, 2):
        if genre1 not in genre_graph:
            genre_graph[genre1] = []
        if genre2 not in genre_graph[genre1]:
            genre_graph[genre1].append(genre2)

        if genre2 not in genre_graph:
            genre_graph[genre2] = []
        if genre1 not in genre_graph[genre2]:
            genre_graph[genre2].append(genre1)
```

```
In [98]: genre_graph
```

```
Out[98]: {'Comedy': ['Crime',
           'Fantasy',
           'Drama',
           'Romance',
           'Action',
           'Sport',
           'Animation',
           'Adventure',
           'Biography',
           'Family'],
          'Crime': ['Comedy',
           'Fantasy',
           'Drama',
           'Action',
           'Adventure',
           'Thriller',
           'Mystery',
           'Romance'],
          'Fantasy': ['Comedy', 'Crime', 'Horror', 'Drama', 'Action', 'Adventure'],
```



Step 3: Implement the PageRank Algorithm

- **Initialize PageRank scores**: Assign an initial PageRank score to each genre
- **Iterate the PageRank update rule:**
    - Update each genre's score based on contributions from connected genres.
    - Apply a damping factor d (0.85) to ensure the algorithm accounts for "random jumps" between nodes.
- **Check for convergence**: Continue iterating until the difference between iterations is below a set threshold (1e−6).

In [102]:
```python
def compute_pagerank(graph, damping_factor=0.85, tol=1e-6):
    num_genres = len(graph)
    pagerank = {genre: 1.0 / num_genres for genre in graph}

    #STARTING THE ITERATION
    for _ in range(100):
        new_pagerank = {}
        for genre in graph:
            # Explanation
            # we are updating the pagerank values using the influence of the neighbor genres
            rank_sum = sum(pagerank[neighbor] / len(graph[neighbor]) for neighbor in graph[genre])
            new_pagerank[genre] = (1 - damping_factor) / num_genres + damping_factor * rank_sum

        # Convergence checking
        if sum(abs(new_pagerank[genre] - pagerank[genre]) for genre in pagerank) < tol:
            break
        pagerank = new_pagerank

    return pagerank

pagerank_scores = compute_pagerank(genre_graph)
print("PageRank Scores:", pagerank_scores)
```

```
PageRank Scores: {'Comedy': 0.09274257203228346, 'Crime': 0.07128385309983042, 'Fantasy': 0.055086316377643524, 'Drama': 0.14964192489613135, 'Western': 0.016283075163655826, 'Action': 0.08098742431553721, 'Thriller': 0.055345262719072544, 'Mystery': 0.06311623019106351, 'Adventure': 0.08041235329282934, 'Horror': 0.04766412815425006, 'Romance': 0.04030948203389856, 'Family': 0.040326470581268106, 'Biography': 0.03359683638901329, 'History': 0.044379221526268234, 'Sci-Fi': 0.04748516693492767, 'Documentary': 0.025713700187868244, 'Sport': 0.024166211364800877, 'Animation': 0.03145977073965768}
```

Step 4: Integrate PageRank Scores into the Recommendation System

- **Use PageRank scores** to rate how important each genre is.
- **Calculate a recommendation score** for each series based on the PageRank scores of its genres.
    - For a given series, sum the PageRank scores of all its genres.
- **Rank series** by their recommendation scores and return the top results.

Step 5: Interpret the Recommendations

- **Explanation**: The recommendation function works by finding series with genres that have high PageRank scores, indicating that these genres are influential based on their connections.
- **Output**: The system provides a list of series that share influential genres with the input series.

Recommendation System

In [109]:
```python
def recommend_series(series_title, n_recommendations):
    if series_title not in df['Series Title'].values:
        return f"Series '{series_title}' not found."

    # explanation
    # getting the genres of the input series
    genres = df[df['Series Title']==series_title]['Genre'].values[0].split(", ")
    recommendations = []

    # For each genre of the input series
    # We take the score of the series with the shared genre
    for title in df['Series Title']:
        if title != series_title:
            title_genres = df[df['Series Title'] == title]['Genre'].values[0].split(", ")
            score = sum(pagerank_scores.get(genre, 0) for genre in genres if genre in title_genres)
            recommendations.append((title, score))

    # EXPLANATION
    # Sorting the score in descending order to get the top series recommendations
    recommendations.sort(key=lambda x: x[1], reverse=True)
    return [title for title, score in recommendations[:n_recommendations]]

recommended_titles = recommend_series('Wednesday',5)
print("Recommended Series:", recommended_titles)
```

```
Recommended Series: ['The Boys', 'Bones', 'Death in Paradise', 'Dead to Me', 'Ghosts']
```

Benefits and Limitations

- **Benefits**: This method can highlight series with common influential genres and give nuanced recommendations.
- **Limitations**: It may not consider more complex factors like individual user preferences, ratings, or content metadata beyond genres.