Karageorgiadis-HW2

April 14, 2021

1 Network Analysis and Web Knownledge Mining

1.1 HW2

2 Import packages needed

[230]:
```
import community as community_louvain
from collections import defaultdict
import matplotlib.cm as cm
import networkx as nx
from sklearn.cluster import SpectralClustering
from sklearn import metrics
import matplotlib.pyplot as plt
import numpy as np
import itertools as it
import scipy.cluster.hierarchy as hierarchy
%matplotlib inline
```

[74]:
```
# change defaults to be less ugly
mpl.rc('xtick', labelsize=14, color="#222222")
mpl.rc('ytick', labelsize=14, color="#222222")
mpl.rc('font', **{'family':'sans-serif','sans-serif':['Arial']})
mpl.rc('font', size=16)
mpl.rc('xtick.major', size=6, width=1)
mpl.rc('xtick.minor', size=3, width=1)
mpl.rc('ytick.major', size=6, width=1)
mpl.rc('ytick.minor', size=3, width=1)
mpl.rc('axes', linewidth=1, edgecolor="#222222", labelcolor="#222222")
mpl.rc('text', usetex=False, color="#222222")
```

[26]:
```
dir(nx)
```

[26]:
```
['AmbiguousSolution',
 'DiGraph',
 'ExceededMaxIterations',
 'Graph',
 'GraphMLReader',
 'GraphMLWriter',
 'HasACycle',
 'LCF_graph',
 'LFR_benchmark_graph',
 'MultiDiGraph',
 'MultiGraph',
 'NetworkXAlgorithmError',
 'NetworkXError',
 'NetworkXException',
 'NetworkXNoCycle',
 'NetworkXNoPath',
 'NetworkXNotImplemented',
 'NetworkXPointlessConcept',
 'NetworkXTreewidthBoundExceeded',
 'NetworkXUnbounded',
 'NetworkXUnfeasible',
 'NodeNotFound',
 'NotATree',
 'OrderedDiGraph',
 'OrderedGraph',
 'OrderedMultiDiGraph',
 'OrderedMultiGraph',
 'PlanarEmbedding',
 'PowerIterationFailedConvergence',
 '__author__',
 '__bibtex__',
 '__builtins__',
 '__cached__',
 '__date__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__path__',
 '__spec__',
 '__version__',
 'adamic_adar_index',
 'add_cycle',
 'add_path',
 'add_star',
 'adj_matrix',
 'adjacency',
 'adjacency_data',
 'adjacency_graph',
 'adjacency_matrix',
 'adjacency_spectrum',
 'adjlist',
 'algebraic_connectivity',
 'algebraicconnectivity',
 'algorithms',
 'all',
 'all_neighbors',
 'all_node_cuts',
 'all_pairs_bellman_ford_path',
 'all_pairs_bellman_ford_path_length',
 'all_pairs_dijkstra',
 'all_pairs_dijkstra_path',
 'all_pairs_dijkstra_path_length',
 'all_pairs_lowest_common_ancestor',
 'all_pairs_node_connectivity',
 'all_pairs_shortest_path',
 'all_pairs_shortest_path_length',
 'all_shortest_paths',
 'all_simple_edge_paths',
 'all_simple_paths',
 'all_topological_sorts',
 'all_triads',
 'all_triplets',
 'ancestors',
 'antichains',
 'approximate_current_flow_betweenness_centrality',
 'articulation_points',
 'assortativity',
 'astar',
 'astar_path',
 'astar_path_length',
 'asteroidal',
 'atlas',
 'attr_matrix',
 'attr_sparse_matrix',
 'attracting',
 'attracting_components',
 'attribute_assortativity_coefficient',
 'attribute_mixing_dict',
 'attribute_mixing_matrix',
 'attrmatrix',
 'authority_matrix',
 'average_clustering',
 'average_degree_connectivity',
 'average_neighbor_degree',
 'average_node_connectivity',
 'average_shortest_path_length',
 'balanced_tree',
 'barabasi_albert_graph',
 'barbell_graph',
 'barycenter',
 'beamsearch',
 'bellman_ford_path',
 'bellman_ford_path_length',
 'bellman_ford_predecessor_and_distance',
 'bethe_hessian_matrix',
 'bethe_hessian_spectrum',
 'bethehessianmatrix',
 'betweenness',
 'betweenness_centrality',
 'betweenness_centrality_source',
 'betweenness_centrality_subset',
 'betweenness_subset',
 'bfs_beam_edges',
 'bfs_edges',
 'bfs_predecessors',
 'bfs_successors',
 'bfs_tree',
 'biconnected',
 'biconnected_component_edges',
 'biconnected_components',
 'bidirectional_dijkstra',
 'bidirectional_shortest_path',
 'binary',
 'binomial_graph',
 'binomial_tree',
 'bipartite',
 'bipartite_layout',
 'boundary',
 'boundary_expansion',
 'breadth_first_search',
 'bridges',
 'bull_graph',
 'capacity_scaling',
 'cartesian_product',
 'caveman_graph',
 'center',
 'centrality',
 'chain_decomposition',
 'chains',
 'check_planarity',
 'chordal',
 'chordal_cycle_graph',
 'chordal_graph_cliques',
 'chordal_graph_treewidth',
 'chvatal_graph',
 'circulant_graph',
 'circular_ladder_graph',
 'circular_layout',
 'classes',
 'classic',
 'clique',
 'cliques_containing_node',
 'closeness',
 'closeness_centrality',
 'closeness_vitality',
 'cluster',
 'clustering',
 'cn_soundarajan_hopcroft',
 'cographs',
 'coloring',
 'combinatorial_embedding_to_pos',
 'common_neighbor_centrality',
 'common_neighbors',
 'communicability',
 'communicability_alg',
 'communicability_betweenness_centrality',
 'communicability_exp',
 'community',
 'complement',
 'complete_bipartite_graph',
 'complete_graph',
 'complete_multipartite_graph',
 'complete_to_chordal_graph',
 'components',
 'compose',
 'compose_all',
 'condensation',
 'conductance',
 'configuration_model',
 'connected',
 'connected_caveman_graph',
 'connected_components',
 'connected_double_edge_swap',
 'connected_watts_strogatz_graph',
 'connectivity',
 'constraint',
 'contracted_edge',
 'contracted_nodes',
 'convert',
 'convert_matrix',
 'convert_node_labels_to_integers',
 'core',
 'core_number',
 'coreviews',
 'correlation',
 'cost_of_flow',
 'could_be_isomorphic',
 'covering',
 'create_empty_copy',
 'cubical_graph',
 'current_flow_betweenness',
 'current_flow_betweenness_centrality',
 'current_flow_betweenness_centrality_subset',
 'current_flow_betweenness_subset',
 'current_flow_closeness',
 'current_flow_closeness_centrality',
 'cut_size',
 'cuts',
 'cycle_basis',
 'cycle_graph',
 'cycles',
 'cytoscape',
 'cytoscape_data',
 'cytoscape_graph',
 'd_separated',
 'd_separation',
 'dag',
 'dag_longest_path',
 'dag_longest_path_length',
 'dag_to_branching',
 'davis_southern_women_graph',
 'degree',
 'degree_alg',
 'degree_assortativity_coefficient',
 'degree_centrality',
 'degree_histogram',
 'degree_mixing_dict',
 'degree_mixing_matrix',
 'degree_pearson_correlation_coefficient',
 'degree_seq',
 'degree_sequence_tree',
 'dense',
 'dense_gnm_random_graph',
 'density',
 'depth_first_search',
 'desargues_graph',
 'descendants',
 'descendants_at_distance',
 'dfs_edges',
 'dfs_labeled_edges',
 'dfs_postorder_nodes',
 'dfs_predecessors',
 'dfs_preorder_nodes',
 'dfs_successors',
 'dfs_tree',
 'diameter',
 'diamond_graph',
 'difference',
 'digraph',
 'dijkstra_path',
 'dijkstra_path_length',
 'dijkstra_predecessor_and_distance',
 'directed',
 'directed_combinatorial_laplacian_matrix',
 'directed_configuration_model',
 'directed_havel_hakimi_graph',
 'directed_joint_degree_graph',
 'directed_laplacian_matrix',
 'directed_modularity_matrix',
 'disjoint_union',
 'disjoint_union_all',
 'dispersion',
 'distance_measures',
 'distance_regular',
 'dodecahedral_graph',
 'dominance',
 'dominance_frontiers',
 'dominating',
 'dominating_set',
 'dorogovtsev_goltsev_mendes_graph',
 'double_edge_swap',
 'draw',
 'draw_circular',
 'draw_kamada_kawai',
 'draw_networkx',
 'draw_networkx_edge_labels',
 'draw_networkx_edges',
 'draw_networkx_labels',
 'draw_networkx_nodes',
 'draw_planar',
 'draw_random',
 'draw_shell',
 'draw_spectral',
 'draw_spring',
 'drawing',
 'dual_barabasi_albert_graph',
 'duplication',
 'duplication_divergence_graph',
 'eccentricity',
 'edge_betweenness',
 'edge_betweenness_centrality',
 'edge_betweenness_centrality_subset',
 'edge_bfs',
 'edge_boundary',
 'edge_connectivity',
 'edge_current_flow_betweenness_centrality',
 'edge_current_flow_betweenness_centrality_subset',
 'edge_dfs',
 'edge_disjoint_paths',
 'edge_expansion',
 'edge_load_centrality',
 'edge_subgraph',
 'edgebfs',
 'edgedfs',
 'edgelist',
 'edges',
 'effective_size',
 'efficiency',
 'efficiency_measures',
 'ego',
 'ego_graph',
 'eigenvector',
 'eigenvector_centrality',
 'eigenvector_centrality_numpy',
 'empty_graph',
 'enumerate_all_cliques',
 'equitable_color',
 'erdos_renyi_graph',
 'estrada_index',
 'euler',
 'eulerian_circuit',
 'eulerian_path',
 'eulerize',
 'exception',
 'expanders',
 'expected_degree_graph',
 'extended_barabasi_albert_graph',
 'extrema_bounding',
 'fast_could_be_isomorphic',
 'fast_gnp_random_graph',
 'faster_could_be_isomorphic',
 'fiedler_vector',
 'filters',
 'find_asteroidal_triple',
 'find_cliques',
 'find_cliques_recursive',
 'find_cores',
 'find_cycle',
 'find_induced_nodes',
 'florentine_families_graph',
 'flow',
 'flow_hierarchy',
 'flow_matrix',
 'floyd_warshall',
 'floyd_warshall_numpy',
 'floyd_warshall_predecessor_and_distance',
 'freeze',
 'from_dict_of_dicts',
 'from_dict_of_lists',
 'from_edgelist',
 'from_graph6_bytes',
 'from_nested_tuple',
 'from_numpy_array',
 'from_numpy_matrix',
 'from_pandas_adjacency',
 'from_pandas_edgelist',
 'from_prufer_sequence',
 'from_scipy_sparse_matrix',
 'from_sparse6_bytes',
 'frucht_graph',
 'fruchterman_reingold_layout',
 'full_join',
 'full_rary_tree',
 'function',
 'gaussian_random_partition_graph',
 'general_random_intersection_graph',
 'generalized_degree',
 'generate_adjlist',
 'generate_edgelist',
 'generate_gexf',
 'generate_gml',
 'generate_graphml',
 'generate_multiline_adjlist',
 'generate_pajek',
 'generators',
 'generic',
 'geographical_threshold_graph',
 'geometric',
 'get_edge_attributes',
 'get_node_attributes',
 'gexf',
 'global_efficiency',
 'global_parameters',
 'global_reaching_centrality',
 'gml',
 'gn_graph',
 'gnc_graph',
 'gnm_random_graph',
 'gnp_random_graph',
 'gnr_graph',
 'goldberg_radzik',
 'gomory_hu_tree',
 'google_matrix',
 'gpickle',
 'graph',
 'graph6',
 'graph_atlas',
 'graph_atlas_g',
 'graph_clique_number',
 'graph_edit_distance',
 'graph_hashing',
 'graph_number_of_cliques',
 'graphical',
 'graphmatrix',
 'graphml',
 'graphviews',
 'greedy_color',
 'grid_2d_graph',
 'grid_graph',
 'group',
 'group_betweenness_centrality',
 'group_closeness_centrality',
 'group_degree_centrality',
 'group_in_degree_centrality',
 'group_out_degree_centrality',
 'harmonic',
 'harmonic_centrality',
 'has_bridges',
 'has_eulerian_path',
 'has_path',
 'havel_hakimi_graph',
 'heawood_graph',
 'hexagonal_lattice_graph',
 'hierarchy',
 'hits',
 'hits_alg',
 'hits_numpy',
 'hits_scipy',
 'hoffman_singleton_graph',
 'house_graph',
 'house_x_graph',
 'hub_matrix',
 'hybrid',
 'hypercube_graph',
 'icosahedral_graph',
 'identified_nodes',
 'immediate_dominators',
 'in_degree_centrality',
 'incidence_matrix',
 'incremental_closeness_centrality',
 'induced_subgraph',
 'info',
 'information_centrality',
 'internet_as_graphs',
 'intersection',
 'intersection_all',
 'intersection_array',
 'interval_graph',
 'inverse_line_graph',
 'is_aperiodic',
 'is_arborescence',
 'is_at_free',
 'is_attracting_component',
 'is_biconnected',
 'is_bipartite',
 'is_branching',
 'is_chordal',
 'is_connected',
 'is_digraphical',
 'is_directed',
 'is_directed_acyclic_graph',
 'is_distance_regular',
 'is_dominating_set',
 'is_edge_cover',
 'is_empty',
 'is_eulerian',
 'is_forest',
 'is_frozen',
 'is_graphical',
 'is_isolate',
 'is_isomorphic',
 'is_k_edge_connected',
 'is_k_regular',
 'is_kl_connected',
 'is_matching',
 'is_maximal_matching',
 'is_multigraphical',
 'is_negatively_weighted',
 'is_path',
 'is_perfect_matching',
 'is_pseudographical',
 'is_regular',
 'is_semiconnected',
 'is_semieulerian',
 'is_simple_path',
 'is_strongly_connected',
 'is_strongly_regular',
 'is_tree',
 'is_triad',
 'is_valid_degree_sequence_erdos_gallai',
 'is_valid_degree_sequence_havel_hakimi',
 'is_valid_directed_joint_degree',
 'is_valid_joint_degree',
 'is_weakly_connected',
 'is_weighted',
 'isolate',
 'isolates',
 'isomorphism',
 'jaccard_coefficient',
 'jit',
 'jit_data',
 'jit_graph',
 'johnson',
 'join',
 'joint_degree_graph',
 'joint_degree_seq',
 'json_graph',
 'junction_tree',
 'k_components',
 'k_core',
 'k_corona',
 'k_crust',
 'k_edge_augmentation',
 'k_edge_components',
 'k_edge_subgraphs',
 'k_factor',
 'k_nearest_neighbors',
 'k_random_intersection_graph',
 'k_shell',
 'k_truss',
 'kamada_kawai_layout',
 'karate_club_graph',
 'katz',
 'katz_centrality',
 'katz_centrality_numpy',
 'kl_connected_subgraph',
 'kosaraju_strongly_connected_components',
 'krackhardt_kite_graph',
 'ladder_graph',
 'laplacian_matrix',
 'laplacian_spectrum',
 'laplacianmatrix',
 'lattice',
 'lattice_reference',
 'layout',
 'leda',
 'les_miserables_graph',
 'lexicographic_product',
 'lexicographical_topological_sort',
 'linalg',
 'line',
 'line_graph',
 'link_analysis',
 'link_prediction',
 'load',
 'load_centrality',
 'local_bridges',
 'local_constraint',
 'local_efficiency',
 'local_reaching_centrality',
 'lollipop_graph',
 'lowest_common_ancestor',
 'lowest_common_ancestors',
 'make_clique_bipartite',
 'make_max_clique_graph',
 'make_small_graph',
 'margulis_gabber_galil_graph',
 'matching',
 'max_flow_min_cost',
 'max_weight_clique',
 'max_weight_matching',
 'maximal_independent_set',
 'maximal_matching',
 'maximum_branching',
 'maximum_flow',
 'maximum_flow_value',
 'maximum_spanning_arborescence',
 'maximum_spanning_edges',
 'maximum_spanning_tree',
 'min_cost_flow',
 'min_cost_flow_cost',
 'min_edge_cover',
 'minimum_branching',
 'minimum_cut',
 'minimum_cut_value',
 'minimum_cycle_basis',
 'minimum_edge_cut',
 'minimum_node_cut',
 'minimum_spanning_arborescence',
 'minimum_spanning_edges',
 'minimum_spanning_tree',
 'minors',
 'mis',
 'mixing',
 'mixing_dict',
 'mixing_expansion',
 'modularity_matrix',
 'modularity_spectrum',
 'modularitymatrix',
 'moebius_kantor_graph',
 'moral',
 'moral_graph',
 'multi_source_dijkstra',
 'multi_source_dijkstra_path',
 'multi_source_dijkstra_path_length',
 'multidigraph',
 'multigraph',
 'multiline_adjlist',
 'multipartite_layout',
 'mycielski',
 'mycielski_graph',
 'mycielskian',
 'navigable_small_world_graph',
 'negative_edge_cycle',
 'neighbor_degree',
 'neighbors',
 'network_simplex',
 'networkx',
 'newman_watts_strogatz_graph',
 'node_attribute_xy',
 'node_boundary',
 'node_classification',
 'node_clique_number',
 'node_connected_component',
 'node_connectivity',
 'node_degree_xy',
 'node_disjoint_paths',
 'node_expansion',
 'node_link',
 'node_link_data',
 'node_link_graph',
 'nodes',
 'nodes_with_selfloops',
 'non_edges',
 'non_neighbors',
 'non_randomness',
 'nonisomorphic_trees',
 'normalized_cut_size',
 'normalized_laplacian_matrix',
 'normalized_laplacian_spectrum',
 'not_implemented_for',
 'null_graph',
 'number_attracting_components',
 'number_connected_components',
 'number_of_cliques',
 'number_of_edges',
 'number_of_isolates',
 'number_of_nodes',
 'number_of_nonisomorphic_trees',
 'number_of_selfloops',
 'number_strongly_connected_components',
 'number_weakly_connected_components',
 'numeric_assortativity_coefficient',
 'numeric_mixing_matrix',
 'nx',
 'nx_agraph',
 'nx_pydot',
 'nx_pylab',
 'nx_shp',
 'nx_yaml',
 'octahedral_graph',
 'omega',
 'onion_layers',
 'operators',
 'optimal_edit_paths',
 'optimize_edit_paths',
 'optimize_graph_edit_distance',
 'ordered',
 'out_degree_centrality',
 'overall_reciprocity',
 'pagerank',
 'pagerank_alg',
 'pagerank_numpy',
 'pagerank_scipy',
 'pairs',
 'pajek',
 'paley_graph',
 'pappus_graph',
 'parse_adjlist',
 'parse_edgelist',
 'parse_gml',
 'parse_graphml',
 'parse_leda',
 'parse_multiline_adjlist',
 'parse_pajek',
 'partial_duplication_graph',
 'path_graph',
 'path_weight',
 'percolation',
 'percolation_centrality',
 'periphery',
 'petersen_graph',
 'planar_drawing',
 'planar_layout',
 'planarity',
 'planted_partition_graph',
 'power',
 'powerlaw_cluster_graph',
 'predecessor',
 'preferential_attachment',
 'prefix_tree',
 'product',
 'project',
 'projected_graph',
 'quotient_graph',
 'ra_index_soundarajan_hopcroft',
 'radius',
 'random_clustered',
 'random_clustered_graph',
 'random_cograph',
 'random_degree_sequence_graph',
 'random_geometric_graph',
 'random_graphs',
 'random_internet_as_graph',
 'random_k_out_graph',
 'random_kernel_graph',
 'random_layout',
 'random_lobster',
 'random_partition_graph',
 'random_powerlaw_tree',
 'random_powerlaw_tree_sequence',
 'random_reference',
 'random_regular_graph',
 'random_shell_graph',
 'random_tree',
 'random_triad',
 'reaching',
 'read_adjlist',
 'read_edgelist',
 'read_gexf',
 'read_gml',
 'read_gpickle',
 'read_graph6',
 'read_graphml',
 'read_leda',
 'read_multiline_adjlist',
 'read_pajek',
 'read_shp',
 'read_sparse6',
 'read_weighted_edgelist',
 'read_yaml',
 'readwrite',
 'reciprocity',
 'reconstruct_path',
 'recursive_simple_cycles',
 'regular',
 'relabel',
 'relabel_gexf_graph',
 'relabel_nodes',
 'relaxed_caveman_graph',
 'release',
 'reportviews',
 'rescale_layout',
 'rescale_layout_dict',
 'resistance_distance',
 'resource_allocation_index',
 'restricted_view',
 'reverse',
 'reverse_view',
 'rich_club_coefficient',
 'richclub',
 'ring_of_cliques',
 'rooted_product',
 's_metric',
 'scale_free_graph',
 'second_order',
 'second_order_centrality',
 'sedgewick_maze_graph',
 'selfloop_edges',
 'semiconnected',
 'set_edge_attributes',
 'set_node_attributes',
 'shell_layout',
 'shortest_path',
 'shortest_path_length',
 'shortest_paths',
 'shortest_simple_paths',
 'sigma',
 'similarity',
 'simple_cycles',
 'simple_paths',
 'simrank_similarity',
 'simrank_similarity_numpy',
 'single_source_bellman_ford',
 'single_source_bellman_ford_path',
 'single_source_bellman_ford_path_length',
 'single_source_dijkstra',
 'single_source_dijkstra_path',
 'single_source_dijkstra_path_length',
 'single_source_shortest_path',
 'single_source_shortest_path_length',
 'single_target_shortest_path',
 'single_target_shortest_path_length',
 'small',
 'smallworld',
 'smetric',
 'social',
 'soft_random_geometric_graph',
 'spanner',
 'sparse6',
 'sparsifiers',
 'spectral_graph_forge',
 'spectral_layout',
 'spectral_ordering',
 'spectrum',
 'spiral_layout',
 'spring_layout',
 'square_clustering',
 'star_graph',
 'stochastic',
 'stochastic_block_model',
 'stochastic_graph',
 'stoer_wagner',
 'strong_product',
 'strongly_connected',
 'strongly_connected_components',
 'strongly_connected_components_recursive',
 'structuralholes',
 'subgraph',
 'subgraph_alg',
 'subgraph_centrality',
 'subgraph_centrality_exp',
 'subgraph_view',
 'sudoku',
 'sudoku_graph',
 'swap',
 'symmetric_difference',
 'tensor_product',
 'test',
 'testing',
 'tetrahedral_graph',
 'thresholded_random_geometric_graph',
 'to_dict_of_dicts',
 'to_dict_of_lists',
 'to_directed',
 'to_edgelist',
 'to_graph6_bytes',
 'to_nested_tuple',
 'to_networkx_graph',
 'to_numpy_array',
 'to_numpy_matrix',
 'to_numpy_recarray',
 'to_pandas_adjacency',
 'to_pandas_edgelist',
 'to_prufer_sequence',
 'to_scipy_sparse_matrix',
 'to_sparse6_bytes',
 'to_undirected',
 'topological_sort',
 'tournament',
 'transitive_closure',
 'transitive_closure_dag',
 'transitive_reduction',
 'transitivity',
 'traversal',
 'tree',
 'tree_all_pairs_lowest_common_ancestor',
 'tree_data',
 'tree_graph',
 'trees',
 'triad_graph',
 'triad_type',
 'triadic_census',
 'triads',
 'triads_by_type',
 'triangles',
 'triangular_lattice_graph',
 'trivial_graph',
 'trophic',
 'trophic_differences',
 'trophic_incoherence_parameter',
 'trophic_levels',
 'truncated_cube_graph',
 'truncated_tetrahedron_graph',
 'turan_graph',
 'tutte_graph',
 'unary',
 'uniform_random_intersection_graph',
 'union',
 'union_all',
 'unweighted',
 'utils',
 'vitality',
 'volume',
 'voronoi',
 'voronoi_cells',
 'voterank',
 'voterank_alg',
 'watts_strogatz_graph',
 'waxman_graph',
 'weakly_connected',
 'weakly_connected_components',
 'weighted',
 'weisfeiler_lehman_graph_hash',
 'wheel_graph',
 'wiener',
 'wiener_index',
 'windmill_graph',
 'within_inter_cluster',
 'write_adjlist',
 'write_edgelist',
 'write_gexf',
 'write_gml',
 'write_gpickle',
 'write_graph6',
 'write_graphml',
 'write_graphml_lxml',
 'write_graphml_xml',
 'write_multiline_adjlist',
 'write_pajek',
 'write_shp',
 'write_sparse6',
 'write_weighted_edgelist',
 'write_yaml']
```

[172]:
```
def modularity(G, partition):
    m = G.number_of_edges()

    degree = G.degree()
    norm = 1.0/(2.0*m)

    Q = 0.0
    for c in partition:
        for u, v in it.product(c, repeat=2):
            w = 1 if G.has_edge(u, v) else 0
            #  double count self loop
            if u == v:
                w *= 2.0
            Q += w - degree[u] * degree[v] * norm
    return norm*Q
```

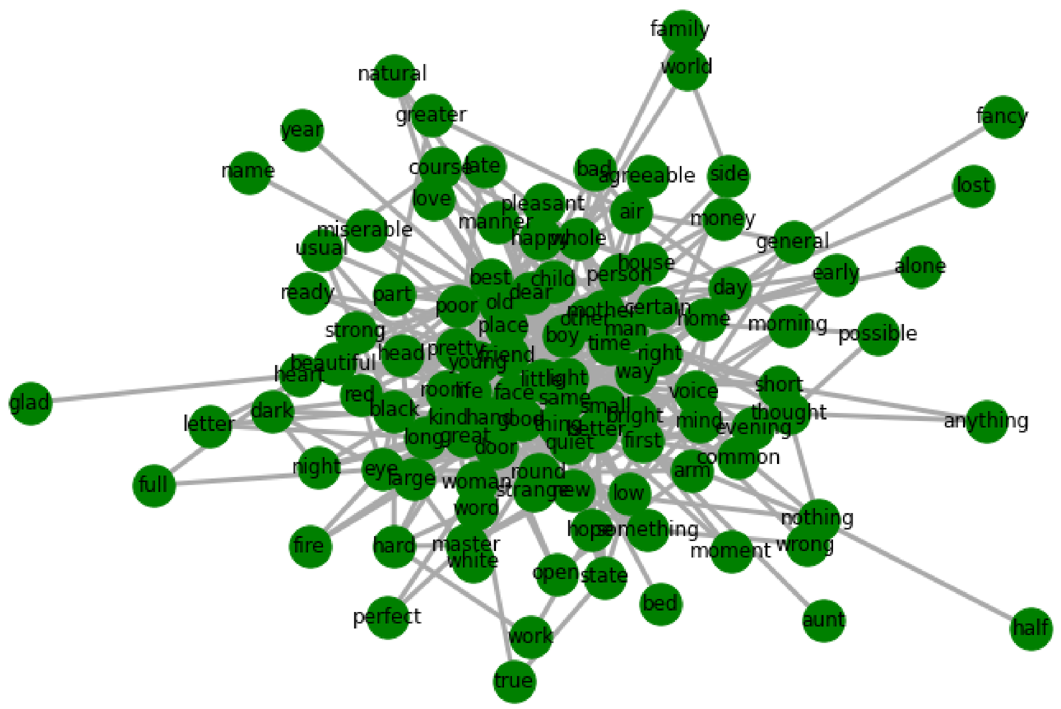
[178]:
```
def girvan_newman(G):
    """ run the algorithm of Girvan + Newman up to the first separation
        and return list of components of G, list of edges removed
    """

    # we're going to remove edges, so do it on a copy of the original graph
    G = G.copy()

    def find_best_edge(G0):
        """ get the edge from G0 with highest betweenness centrality"""
        eb = nx.edge_betweenness_centrality(G0)
        edges = eb.keys()
        return max(edges, key=lambda e: eb[e])

    removed_edges = []
    # Proceed until we separate the graph
    while nx.number_connected_components(G) == 1:
        u, v = find_best_edge(G)
        G.remove_edge(u, v)
        removed_edges.append((u, v))

    return list(nx.connected_components(G)), removed_edges
```

2.1 Read Data from files

[5]:
```
H1 = nx.read_gml("adjnoun/adjnoun.gml")
H2 = nx.read_gml("polbooks/polbooks.gml")

# these lines reads a graph from Geographic Markdown Lang
```

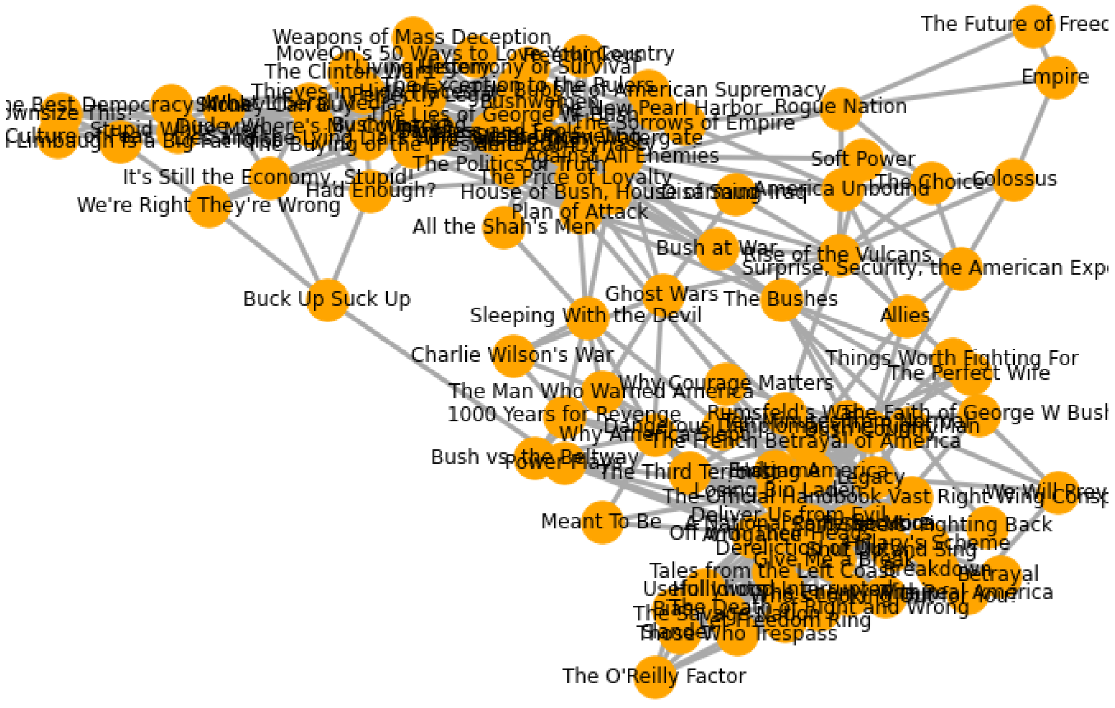
2.1.1 Plot Graph H1 words clustering

[122]:
```
pos1 = nx.spring_layout(H1)

plt.figure(figsize=(12,8))
nx.draw_networkx(H1, pos=pos1, node_color='green', edge_color='darkgray',␣
 ↪width=3, node_size=640)
limits = plt.axis('off')
```



2.1.2 Plot Graph H2 books - 3clusters

```
[121]: pos2 = nx.spring_layout(H2)

       plt.figure(figsize=(12,8))
       nx.draw_networkx(H2, pos=pos2, node_color='orange', edge_color='darkgray',␣
        ↪width=3, node_size=640)
       limits = plt.axis('off')
```



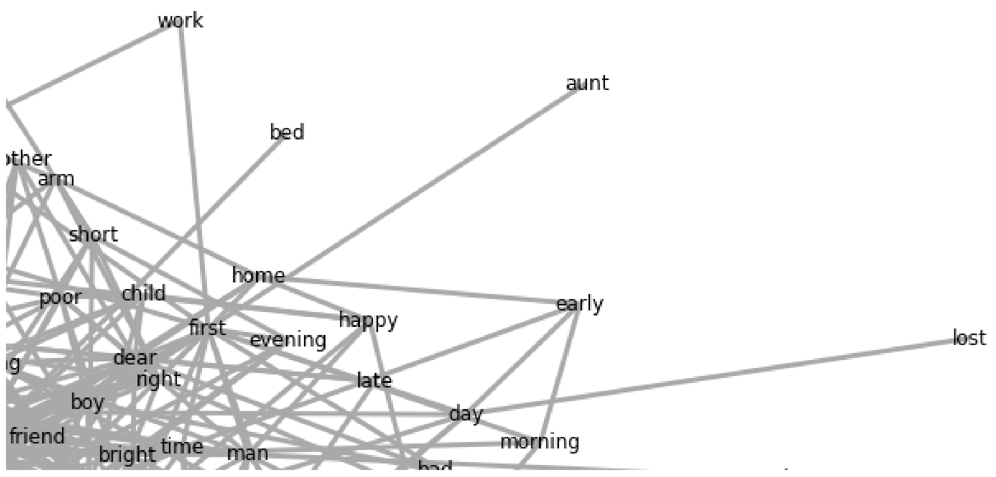
2.2 A.| Extract k-cliques from each graph

```
[129]: # get the 2-cliques for graph 1 adjnouns

       k=2
       c_h1 = nx.algorithms.community.k_clique_communities(H1, k)
       print("Graph H1 show {}-cliques\n\n {}".format(k,list(c_h1)) )

       cumminities_cl_H1 = [list(x) for x in c_h1]
```

```
Graph H1 show 2-cliques

 [frozenset({'thing', 'better', 'bright', 'pretty', 'work', 'glad', 'round',
'dear', 'room', 'long', 'new', 'strange', 'certain', 'family', 'life', 'full',
'morning', 'aunt', 'eye', 'common', 'home', 'state', 'quiet', 'kind', 'right',
'money', 'other', 'first', 'general', 'best', 'red', 'person', 'alone',
'friend', 'black', 'face', 'time', 'name', 'letter', 'half', 'young', 'light',
'large', 'house', 'true', 'miserable', 'wrong', 'greater', 'day', 'evening',
'hard', 'happy', 'low', 'open', 'heart', 'night', 'little', 'early', 'poor',
'bad', 'dark', 'strong', 'ready', 'bed', 'place', 'man', 'agreeable', 'part',
'white', 'usual', 'good', 'arm', 'boy', 'something', 'moment', 'mind',
'thought', 'old', 'manner', 'mother', 'lost', 'pleasant', 'love', 'door',
'same', 'small', 'voice', 'perfect', 'whole', 'world', 'course', 'head',
'beautiful', 'great', 'way', 'hope', 'word', 'fire', 'anything', 'late',
'woman', 'air', 'master', 'side', 'hand', 'short', 'fancy', 'possible', 'child',
'nothing', 'year', 'natural'})]
```

2.2.1 Plot 2-clique for adjnoun graph H1

[116]:
```
plt.figure(figsize=(12,8))
nx.draw_networkx_edges(H1, pos=pos, width=3, edge_color='darkgray')
colors = ['salmon', 'lightblue', 'green']

for community, color in zip(cumminities_cl_H1, colors):
    nx.draw_networkx_nodes(H1, pos=pos1, nodelist=cumminities_cl_H1,␣
 ↪node_color=color, node_size=640)

nx.draw_networkx_labels(H1, pos=pos)

_ = plt.axis('off')
```



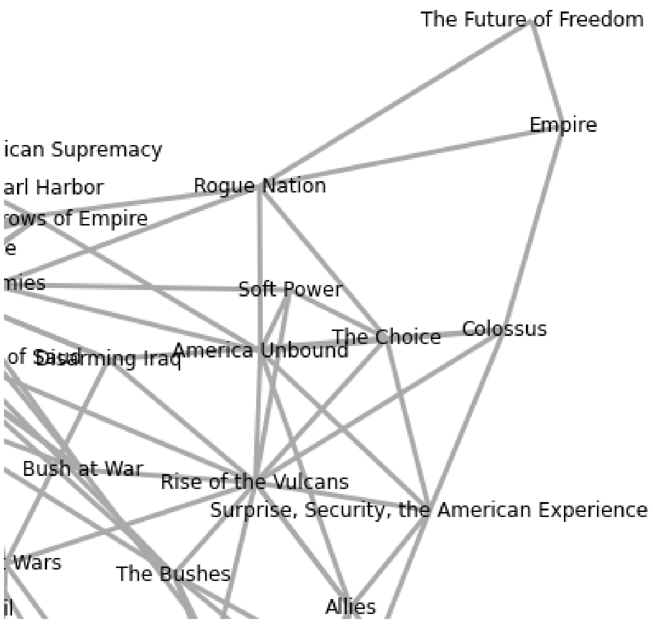
2.2.2 Get the cliques for books graph - H2

```
[120]: c_h2 = list(nx.algorithms.community.k_clique_communities(H2, 3))

       print("Graph H2 show {}-cliques\n\n {}".format(k,c_h2))


       cummunities_cl_H2 =  [list(x) for x in c_h2]  # 3-clique communities books␣
        ↪graph
```

```
Graph H2 show 2-cliques

 [frozenset({'Fighting Back', 'Persecution', 'Bush Country', 'Give Me a Break',
'The Right Man', 'Bias', 'Betrayal', 'Slander', 'Shut Up and Sing', 'The French
Betrayal of America', 'The Enemy Within', 'Arrogance', 'Why Courage Matters',
'Useful Idiots', 'Losing Bin Laden', "Rumsfeld's War", 'The Bushes', "The
O'Reilly Factor", 'Endgame', 'Dangerous Dimplomacy', 'The Perfect Wife',
'Dereliction of Duty', 'Hating America', 'Breakdown', 'A National Party No
More', 'Deliver Us from Evil', 'Power Plays', 'The Savage Nation', 'Hollywood
Interrupted', 'The Official Handbook Vast Right Wing Conspiracy', 'Why America
Slept', 'Ten Minutes from Normal', "Hillary's Scheme", 'Let Freedom Ring',
'Tales from the Left Coast', 'Things Worth Fighting For', 'The Death of Right
and Wrong', 'Those Who Trespass', 'The Faith of George W Bush', "Who's Looking
Out for You?", 'Legacy', 'Meant To Be', 'Off with Their Heads', 'The Real
America', 'Spin Sisters', 'The Third Terrorist'}), frozenset({'Shrub', 'The
Politics of Truth', 'Surprise, Security, the American Experience', 'Bush
Country', 'Thieves in High Places', 'The Price of Loyalty', 'Colossus', 'Against
All Enemies', 'Rush Limbaugh Is a Big Fat Idiot', 'Fanatics and Fools', 'Rogue
Nation', 'The Sorrows of Empire', 'Stupid White Men', 'Weapons of Mass
Deception', 'The Great Unraveling', "All the Shah's Men", 'Big Lies', 'The
Bushes', "Rumsfeld's War", 'The Clinton Wars', 'Living History', 'Bushwomen',
'Hegemony or Survival', "MoveOn's 50 Ways to Love Your Country", 'The New Pearl
Harbor', 'Allies', 'Downsize This!', 'Bushwhacked', "Dude, Where's My Country?",
'Sleeping With the Devil', 'Had Enough?', 'Bush at War', 'The Exception to the
Rulers', 'Soft Power', 'Plan of Attack', "We're Right They're Wrong", "It's
Still the Economy, Stupid!", 'America Unbound', 'Buck Up Suck Up', 'The Lies of
George W. Bush', 'Perfectly Legal', 'Rise of the Vulcans', 'The Choice', 'The
Culture of Fear', 'American Dynasty', 'Worse Than Watergate', 'House of Bush,
House of Saud', 'Ghost Wars', 'Disarming Iraq', 'What Liberal Media?', 'The
Buying of the President 2004', 'Freethinkers', 'The Best Democracy Money Can
Buy', 'Lies and the Lying Liars Who Tell Them', 'The Bubble of American
Supremacy'}), frozenset({'Empire', 'The Future of Freedom', 'Rogue Nation'}),
frozenset({'Losing Bin Laden', 'Sleeping With the Devil', 'Ghost Wars', 'Why
America Slept', 'Dangerous Dimplomacy', '1000 Years for Revenge', "Charlie
Wilson's War", 'Bush vs. the Beltway', 'The Man Who Warned America'})]
```

2.2.3 Plot 3-clique part of the graph for Books data -Graph H2

```
[127]: plt.figure(figsize=(12,8))
       nx.draw_networkx_edges(H2, pos=pos2, width=3, edge_color='darkgray')
       colors = ['salmon', 'lightblue', 'green']

       for community, color in zip(cummunities_cl_H2, colors):
           nx.draw_networkx_nodes(H2, pos=pos2, nodelist=cummunities_cl_H2,␣
        ↪node_color=color, node_size=640)

       nx.draw_networkx_labels(H2, pos=pos2)

       _ = plt.axis('off')
```



2.2.4 B.| Maximize Modularity

```
[196]: c_h1_m = list(nx.algorithms.community.greedy_modularity_communities(H1))

       print("Maximum Modularity on graph H1 \n\n",c_h1_m)
```

```


cummunities_gr_mod_h1 = [list(x) for x in c_h1_m]

print(cummunities_gr_mod_h1
     )
```

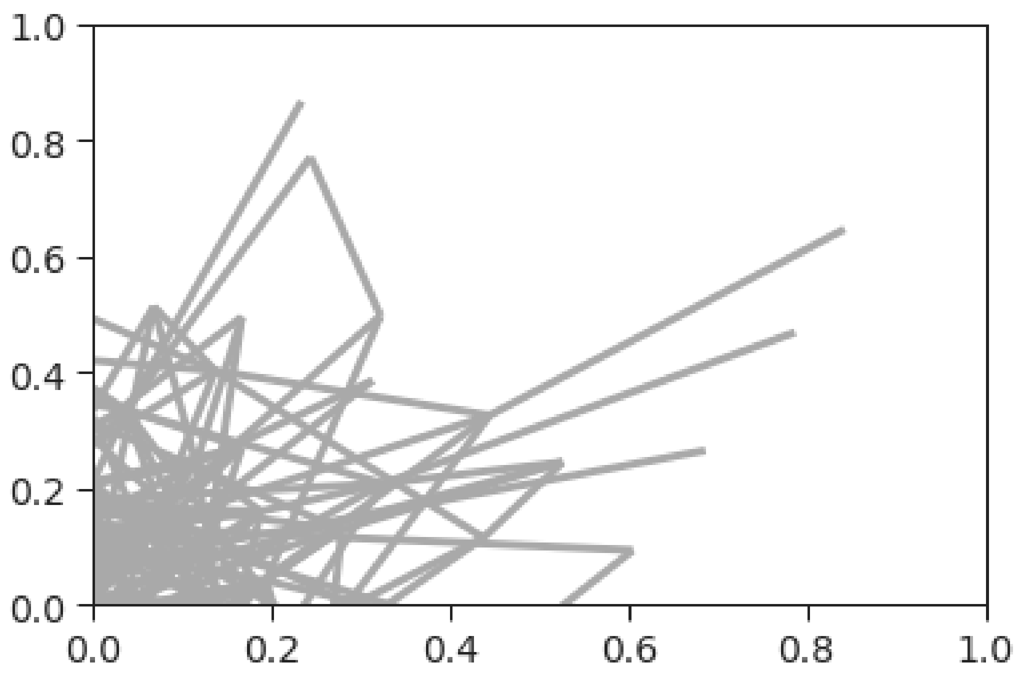
```
Maximum Modularity on graph H1

 [frozenset({'bright', 'pretty', 'round', 'room', 'long', 'part', 'usual',
'eye', 'good', 'arm', 'dark', 'red', 'black', 'door', 'small', 'letter',
'large', 'head', 'beautiful', 'great', 'hope', 'fire', 'open', 'hand', 'night',
'strong'}), frozenset({'thing', 'certain', 'man', 'agreeable', 'life', 'white',
'morning', 'aunt', 'common', 'money', 'moment', 'first', 'best', 'alone', 'old',
'lost', 'love', 'time', 'name', 'day', 'happy', 'early', 'year', 'person'}),
frozenset({'word', 'bed', 'better', 'state', 'woman', 'quiet', 'same', 'place',
'voice', 'low', 'right', 'mind', 'true', 'little', 'miserable', 'ready',
'home'}), frozenset({'pleasant', 'other', 'whole', 'world', 'general', 'light',
'course', 'house', 'short', 'natural', 'way', 'fancy', 'side', 'family',
'manner', 'evening', 'anything'}), frozenset({'friend', 'late', 'boy', 'air',
'kind', 'dear', 'young', 'thought', 'child', 'mother', 'possible', 'poor',
'greater', 'bad'}), frozenset({'face', 'hard', 'work', 'something', 'master',
'perfect', 'half', 'new', 'nothing', 'strange', 'wrong'}), frozenset({'glad',
'full', 'heart'})]
[['bright', 'pretty', 'round', 'room', 'long', 'part', 'usual', 'eye', 'good',
'arm', 'dark', 'red', 'black', 'door', 'small', 'letter', 'large', 'head',
'beautiful', 'great', 'hope', 'fire', 'open', 'hand', 'night', 'strong'],
['thing', 'certain', 'man', 'agreeable', 'life', 'white', 'morning', 'aunt',
'common', 'money', 'moment', 'first', 'best', 'alone', 'old', 'lost', 'love',
'time', 'name', 'day', 'happy', 'early', 'year', 'person'], ['word', 'bed',
'better', 'state', 'woman', 'quiet', 'same', 'place', 'voice', 'low', 'right',
'mind', 'true', 'little', 'miserable', 'ready', 'home'], ['pleasant', 'other',
'whole', 'world', 'general', 'light', 'course', 'house', 'short', 'natural',
'way', 'fancy', 'side', 'family', 'manner', 'evening', 'anything'], ['friend',
'late', 'boy', 'air', 'kind', 'dear', 'young', 'thought', 'child', 'mother',
'possible', 'poor', 'greater', 'bad'], ['face', 'hard', 'work', 'something',
'master', 'perfect', 'half', 'new', 'nothing', 'strange', 'wrong'], ['glad',
'full', 'heart']]
```

2.2.5 Plot H1 graph with greedy modularity

```
[139]: nx.draw_networkx_edges(H1, pos=pos1, width=3, edge_color='darkgray')
colors = ['salmon', 'lightblue', 'green']

for community, color in zip(cummunities_gr_mod_h1, colors):
    nx.draw_networkx_nodes(H1, pos=pos1, nodelist=cummunities_gr_mod_h1,␣
 ↪node_color=color, node_size=640)
```

```
nx.draw_networkx_labels(H1, pos=pos1)

_ = plt.axis('off')
```

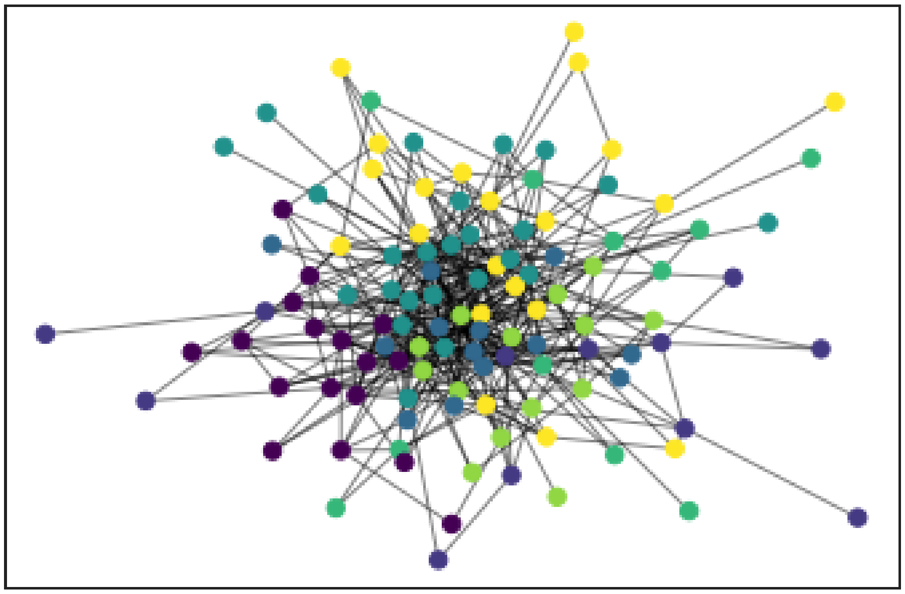
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-139-830116405d88> in <module>
      3
      4 for community, color in zip(cummunities_gr_mod_h1, colors):
----> 5     nx.draw_networkx_nodes(H1, pos=pos1, nodelist=cummunities_gr_mod_h1 ␣
 ↪node_color=color, node_size=640)
      6
      7 nx.draw_networkx_labels(H1, pos=pos1)

~/.local/lib/python3.9/site-packages/networkx/drawing/nx_pylab.py in␣
 ↪draw_networkx_nodes(G, pos, nodelist, node_size, node_color, node_shape,␣
 ↪alpha, cmap, vmin, vmax, ax, linewidths, edgecolors, label)
    454
    455     try:
--> 456         xy = np.asarray([pos[v] for v in nodelist])
    457     except KeyError as e:
    458         raise nx.NetworkXError(f"Node {e} has no position.") from e

~/.local/lib/python3.9/site-packages/networkx/drawing/nx_pylab.py in <listcomp> .
 ↪0)
    454
    455     try:
--> 456         xy = np.asarray([pos[v] for v in nodelist])
    457     except KeyError as e:
    458         raise nx.NetworkXError(f"Node {e} has no position.") from e

TypeError: unhashable type: 'list'
```



2.2.6 Graph H1 -adjnoouns modularity using Louvain

[221]:
```
### Use Louvain method for modularity

# compute the best partition
partition = community_louvain.best_partition(H1)
print(type(partition))

# cum_Louv_h1 = [ list(x) for x in partition]

l1,l2 = [],[]
for k,v in partition.items():
    if v==0:
        l1.append(k)
    else:
        l2.append(k)

print('\nCluster1 {} \n@Cluster2{}'.format(l1,l2))

cummunitiesLouv_H1_list = [l1,l2] # list of list to calculate modularity later

# color the nodes according to their partition
cmap = cm.get_cmap('viridis', max(partition.values()) + 1)
nx.draw_networkx_nodes(H1, pos1, partition.keys(), node_size=40,
                        cmap=cmap, node_color=list(partition.values()))
nx.draw_networkx_edges(H1, pos1, alpha=0.5)
plt.show()
```

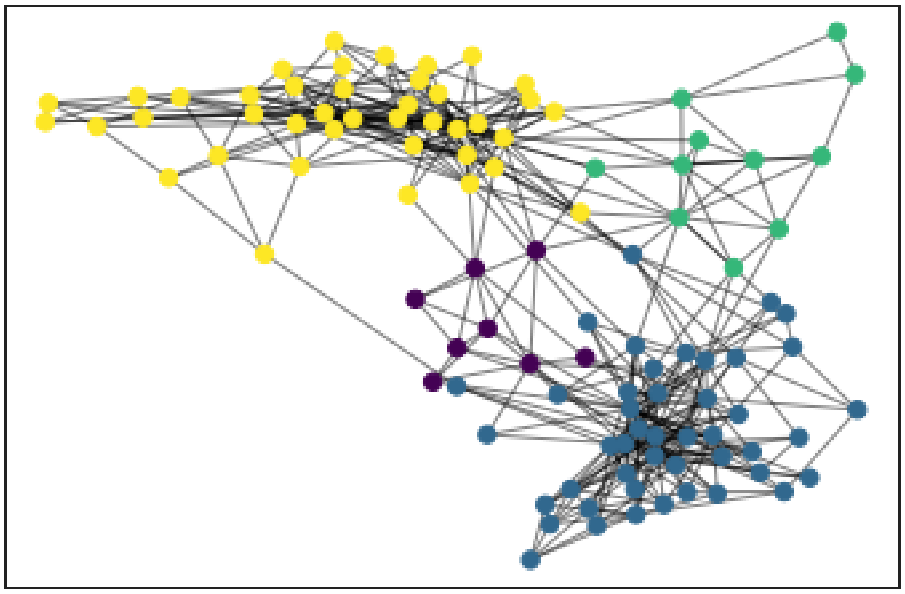
```
<class 'dict'>

Cluster1 ['beautiful', 'black', 'letter', 'room', 'eye', 'dark', 'night',
'fire', 'great', 'work', 'long', 'hard', 'red', 'large', 'white', 'strong',
'usual']
@Cluster2['agreeable', 'man', 'old', 'person', 'anything', 'short', 'arm',
'round', 'aunt', 'first', 'bad', 'air', 'boy', 'face', 'little', 'young',
'best', 'course', 'friend', 'love', 'part', 'thing', 'time', 'way', 'better',
'heart', 'mind', 'place', 'right', 'state', 'woman', 'word', 'door', 'bright',
'evening', 'morning', 'certain', 'day', 'other', 'child', 'happy', 'common',
'kind', 'dear', 'good', 'home', 'mother', 'pretty', 'open', 'early', 'full',
'master', 'moment', 'general', 'fancy', 'voice', 'head', 'hope', 'greater',
'hand', 'life', 'glad', 'new', 'late', 'whole', 'light', 'manner', 'bed',
'house', 'low', 'money', 'ready', 'small', 'strange', 'thought', 'lost',
'alone', 'nothing', 'miserable', 'natural', 'half', 'wrong', 'name', 'pleasant',
'possible', 'side', 'perfect', 'poor', 'quiet', 'same', 'something', 'true',
'family', 'world', 'year']
```



2.2.7 Max modularity communities for graph H2 -books

[215]:
```
### Use Louvain method for modularity

# compute the best partition
partition2 = community_louvain.best_partition(H2)

# cum_Louv_h2= [ list(x) for x in partition2]

# print(partition2)

# print('\n',cum_Louv_h2)
a,b,c = [],[],[]
for key, value in partition2.items():
    if value == 0:
        a.append(key)
    elif value==1:
        b.append(key)
    else:
        c.append(key)

print("\n a= ",a,"\n b=",b,"\n c=",c)

cummunititesLouv_H2_list = [a,b,c]

# color the nodes according to their partition
cmap = cm.get_cmap('viridis', max(partition2.values()) + 1)
nx.draw_networkx_nodes(H2, pos2, partition2.keys(), node_size=40,
                       cmap=cmap, node_color=list(partition2.values()))
nx.draw_networkx_edges(H2, pos2, alpha=0.5)
plt.show()
```

```
 a=  ['1000 Years for Revenge', 'Bush vs. the Beltway', "Charlie Wilson's War",
'Sleeping With the Devil', 'The Man Who Warned America', 'Why America Slept',
'Ghost Wars', 'Dangerous Dimplomacy']
 b= ['Losing Bin Laden', 'A National Party No More', 'Bush Country',
'Dereliction of Duty', 'Legacy', 'Off with Their Heads', 'Persecution',
"Rumsfeld's War", 'Breakdown', 'Betrayal', 'Shut Up and Sing', 'Meant To Be',
'The Right Man', 'Ten Minutes from Normal', "Hillary's Scheme", 'The French
Betrayal of America', 'Tales from the Left Coast', 'Hating America', 'The Third
Terrorist', 'Endgame', 'Spin Sisters', 'The Death of Right and Wrong', 'Useful
Idiots', "The O'Reilly Factor", 'Let Freedom Ring', 'Those Who Trespass',
'Bias', 'Slander', 'The Savage Nation', 'Deliver Us from Evil', 'Give Me a
Break', 'The Enemy Within', 'The Real America', "Who's Looking Out for You?",
'The Official Handbook Vast Right Wing Conspiracy', 'Power Plays', 'Arrogance',
'The Perfect Wife', 'The Bushes', 'Things Worth Fighting For', 'Why Courage
Matters', 'Hollywood Interrupted', 'Fighting Back', 'We Will Prevail', 'The
Faith of George W Bush']
 c= ["All the Shah's Men", 'The Price of Loyalty', 'House of Bush, House of
Saud', 'Surprise, Security, the American Experience', 'Allies', 'Rise of the
Vulcans', 'Downsize This!', 'Stupid White Men', 'Rush Limbaugh Is a Big Fat
Idiot', 'The Best Democracy Money Can Buy', 'The Culture of Fear', 'America
Unbound', 'The Choice', 'The Great Unraveling', 'Rogue Nation', 'Soft Power',
'Colossus', 'The Sorrows of Empire', 'Against All Enemies', 'American Dynasty',
'Big Lies', 'The Lies of George W. Bush', 'Worse Than Watergate', 'Plan of
Attack', 'Bush at War', 'The New Pearl Harbor', 'Bushwomen', 'The Bubble of
American Supremacy', 'Living History', 'The Politics of Truth', 'Fanatics and
Fools', 'Bushwhacked', 'Disarming Iraq', 'Lies and the Lying Liars Who Tell
Them', "MoveOn's 50 Ways to Love Your Country", 'The Buying of the President
2004', 'Perfectly Legal', 'Hegemony or Survival', 'The Exception to the Rulers',
'Freethinkers', 'Had Enough?', "It's Still the Economy, Stupid!", "We're Right
They're Wrong", 'What Liberal Media?', 'The Clinton Wars', 'Weapons of Mass
Deception', "Dude, Where's My Country?", 'Thieves in High Places', 'Shrub',
'Buck Up Suck Up', 'The Future of Freedom', 'Empire']
```



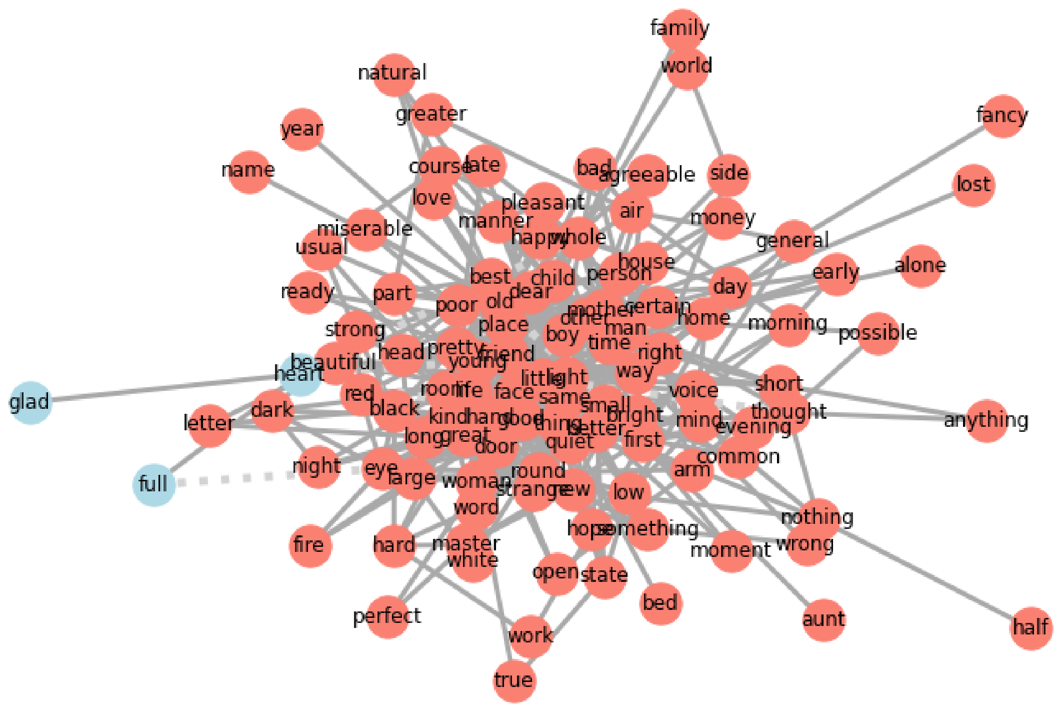
[148]:
```

```

2.3 C.|Hierarchical clustering girvan newman

2.3.1 for graph H1

[150]:
```
communities1, removed_edges1 = girvan_newman(H1)
# Edges that were NOT removed by GN algorithm
other_edges1 = set(H1.edges()) - set(removed_edges1)
```

[151]:
```
plt.figure(figsize=(12,8))
# plot edges, with edges removed by algorithm dashed instead of solid
nx.draw_networkx_edges(H1, pos1, width=3, edge_color='darkgray',␣
↪edgelist=other_edges1)
nx.draw_networkx_edges(H1, pos1, edgelist=removed_edges1, style='dotted',␣
↪edge_color='lightgray', width=5)
for community, color in zip(communities1, colors):
    nx.draw_networkx_nodes(H1, pos=pos1, nodelist=community, node_color=color,␣
↪node_size=640)
nx.draw_networkx_labels(H1, pos=pos1)
_ = plt.axis('off')
```

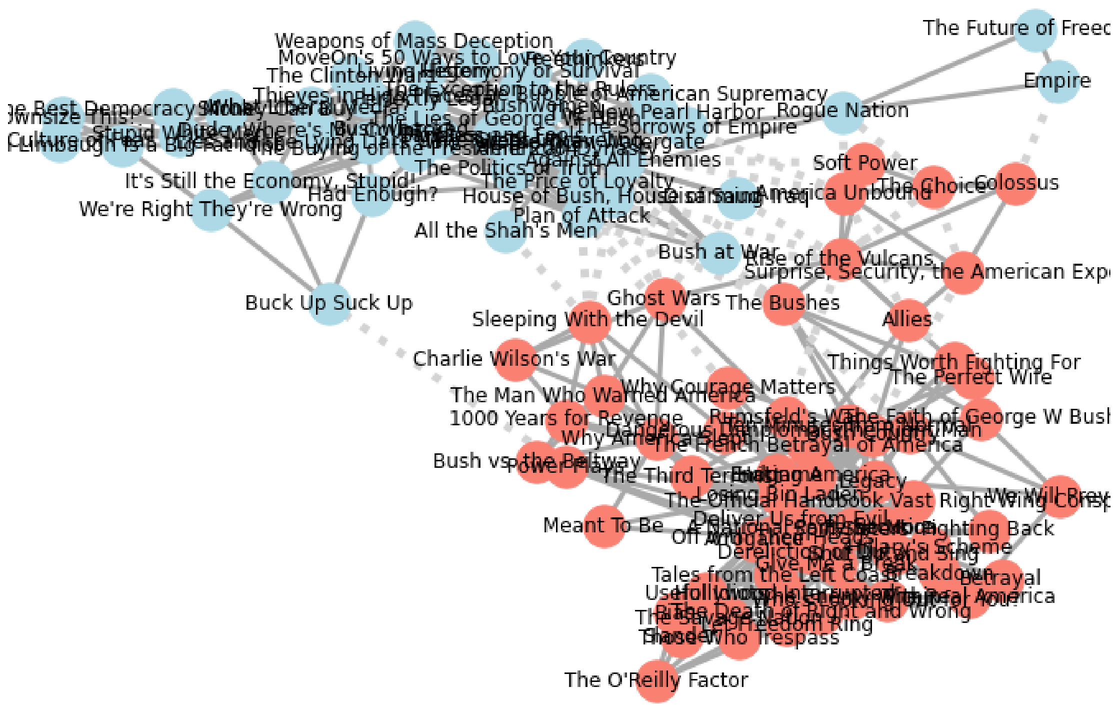


2.3.2 For graph H2 - girvan_newman

[153]:
```
communities2, removed_edges2 = girvan_newman(H2)
# Edges that were NOT removed by GN algorithm
other_edges2 = set(H2.edges()) - set(removed_edges2)
```

[155]:
```
plt.figure(figsize=(12,8))
# plot edges, with edges removed by algorithm dashed instead of solid
nx.draw_networkx_edges(H2, pos2, width=3, edge_color='darkgray',␣
↪edgelist=other_edges2)
```

```
nx.draw_networkx_edges(H2, pos2, edgelist=removed_edges2, style='dotted',␣
↪edge_color='lightgray', width=5)
for community, color in zip(communities2, colors):
    nx.draw_networkx_nodes(H2, pos=pos2, nodelist=community, node_color=color,␣
↪node_size=640)
nx.draw_networkx_labels(H2, pos=pos2)
_ = plt.axis('off')
```



2.3.3 d. | Spectral Clustering

[204]:
```
# Get adjacency-matrix as numpy-array
adj_mat1 = nx.to_numpy_matrix(H1)
adj_mat2 = nx.to_numpy_matrix(H2)


# Cluster
sc = SpectralClustering(2, affinity='precomputed', n_init=100)
sc.fit(adj_mat1)

sc2 = SpectralClustering(3, affinity='precomputed', n_init=100)
sc2.fit(adj_mat2)

# Compare ground-truth and clustering-results
```

```
print('\nAdjnoun H1-spectral clustering')
print(sc.labels_)
print('\njust for better-visualization: invert clusters (permutation)')
print(np.abs(sc.labels_ - 1))


# Compare ground-truth and clustering-results -- graph H2 3 cluster books
print('\nBooks H2spectral clustering')
print(sc2.labels_)
print('\njust for better-visualization: invert clusters (permutation)')
print(np.abs(sc2.labels_ - 1))
```

```
Adjnoun H1-spectral clustering
[0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 1 1
 0]

just for better-visualization: invert clusters (permutation)
[1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 1 1 1 1 1 0 1 0 0
 1]

Books H2spectral clustering
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 0 0 0 0 0 2 1 1 1 1 1 2 2 1 2 2 2 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2]

just for better-visualization: invert clusters (permutation)
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 0 1 1 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1]
```

2.3.4 B.| Measures

v. Modularity Measures

For Graph H1

[222]:
```
Q1_1 = nx.algorithms.community.modularity(H1,cummunities_gr_mod_h1) # greedy␣
↪modularity graph1
Q1_2 = modularity(H1,cummunitiesLouv_H1_list)
Q1_3 = nx.algorithms.community.modularity(H1,communities1)


print("Modularity for graph1 using: \n greedy method ={}, ,\nusing Louvain {},␣
↪\n using girvan-newman {}".format(Q1_1,Q1_2,Q1_3))
```

```
Modularity for graph1 using:
 greedy method =0.29469619377162637, ,
using Louvain 0.09459100346020617,
 using girvan-newman 0.009234602076124521
```

For graph H2

[218]:
```
Q2_1 = modularity(H2,cummunities_gr_mod_h2) # greedy modularity graph2
Q2_3 = modularity(H2,communities2)
Q2_2 = modularity(H2,cummunititesLouv_H2_list)

print("Modularity for graph2 using greedy method ={} ,\nusing Louvain␣
 ↪{},\nusing girvan-newman {}".format(Q2_1,Q2_2,Q2_3))
```

```
Modularity for graph2 using greedy method =0.5019744859395165 ,
using Louvain 0.48447149078832974,
using girvan-newman 0.4428915935232932
```

2.4 Part B

[225]:
```
### Generate random model
n = 500
m = 250
p = 0.2




Gpow = nx.generators.random_graphs.powerlaw_cluster_graph(n, m, p)

communities_Gpow = community_louvain.best_partition(Gpow)

clusters_id = set(communities_Gpow)

print("\n $Different Clusters/Groups of keys/IDs ",clusters_id)

print("\nAlls cummunities data",communities_Gpow)
```

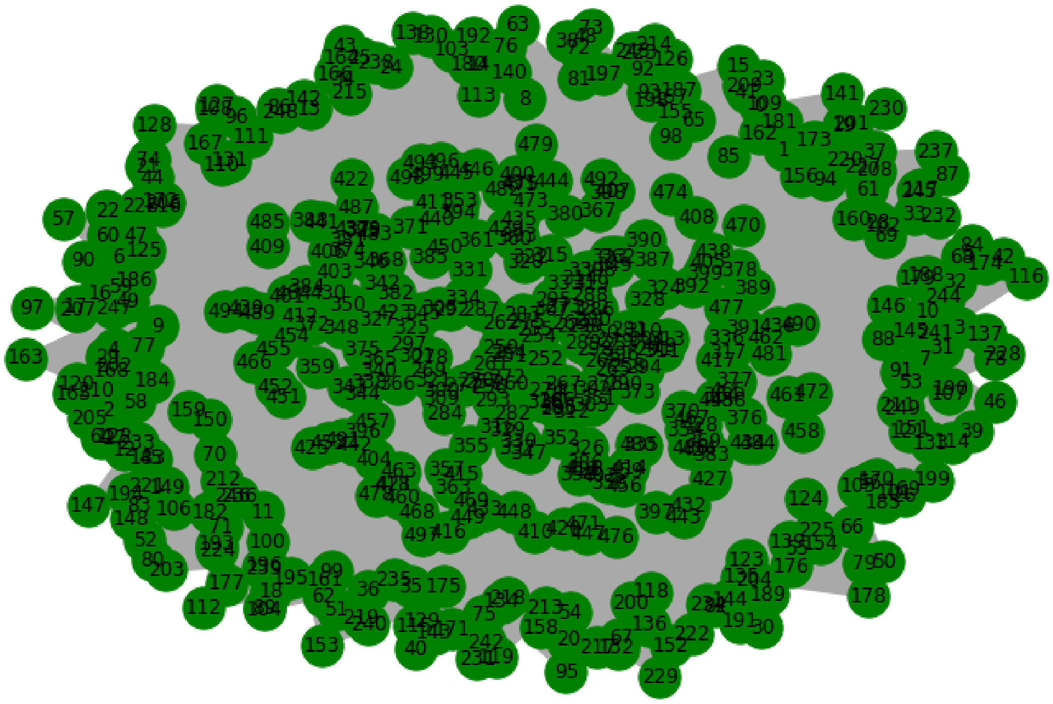
```
 $Different Clusters/Groups of keys/IDs  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71,
72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91,
92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108,
109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124,
125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140,
141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156,
157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172,
173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188,
189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204,
205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220,
221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236,
237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252,
253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268,
269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284,
285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300,
301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316,
317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332,
333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348,
349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364,
365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380,
381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396,
397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412,
413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428,
429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444,
445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460,
461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476,
477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492,
493, 494, 495, 496, 497, 498, 499}

Alls cummunities data {0: 2, 1: 1, 2: 3, 3: 4, 4: 5, 5: 0, 6: 3, 7: 1, 8: 2, 9:
4, 10: 4, 11: 0, 12: 1, 13: 1, 14: 5, 15: 5, 16: 0, 17: 2, 18: 2, 19: 5, 20: 0,
21: 0, 22: 0, 23: 5, 24: 0, 25: 1, 26: 4, 27: 4, 28: 5, 29: 0, 30: 5, 31: 3, 32:
4, 33: 3, 34: 2, 35: 3, 36: 0, 37: 0, 38: 2, 39: 0, 40: 5, 41: 3, 42: 0, 43: 2,
44: 1, 45: 4, 46: 0, 47: 1, 48: 2, 49: 3, 50: 1, 51: 0, 52: 1, 53: 0, 54: 3, 55:
4, 56: 0, 57: 2, 58: 2, 59: 2, 60: 3, 61: 0, 62: 1, 63: 4, 64: 4, 65: 2, 66: 1,
67: 5, 68: 4, 69: 3, 70: 5, 71: 1, 72: 5, 73: 0, 74: 4, 75: 2, 76: 4, 77: 1, 78:
4, 79: 3, 80: 2, 81: 5, 82: 4, 83: 0, 84: 4, 85: 4, 86: 2, 87: 1, 88: 4, 89: 5,
90: 2, 91: 2, 92: 0, 93: 3, 94: 0, 95: 5, 96: 2, 97: 1, 98: 5, 99: 5, 100: 5,
101: 0, 102: 3, 103: 1, 104: 0, 105: 1, 106: 1, 107: 4, 108: 4, 109: 5, 110: 5,
111: 5, 112: 5, 113: 1, 114: 4, 115: 0, 116: 4, 117: 0, 118: 5, 119: 3, 120: 1,
121: 4, 122: 4, 123: 1, 124: 1, 125: 4, 126: 1, 127: 4, 128: 5, 129: 3, 130: 5,
131: 3, 132: 3, 133: 5, 134: 1, 135: 3, 136: 3, 137: 2, 138: 2, 139: 3, 140: 0,
141: 5, 142: 3, 143: 4, 144: 1, 145: 5, 146: 4, 147: 4, 148: 1, 149: 4, 150: 5,
151: 0, 152: 4, 153: 3, 154: 4, 155: 4, 156: 1, 157: 2, 158: 5, 159: 5, 160: 0,
161: 5, 162: 5, 163: 4, 164: 2, 165: 0, 166: 4, 167: 1, 168: 1, 169: 0, 170: 0,
171: 3, 172: 5, 173: 1, 174: 0, 175: 4, 176: 1, 177: 2, 178: 3, 179: 3, 180: 5,
181: 1, 182: 1, 183: 1, 184: 1, 185: 1, 186: 0, 187: 4, 188: 0, 189: 0, 190: 4,
191: 0, 192: 3, 193: 0, 194: 3, 195: 2, 196: 5, 197: 2, 198: 1, 199: 4, 200: 3,
201: 5, 202: 4, 203: 2, 204: 5, 205: 4, 206: 5, 207: 0, 208: 0, 209: 2, 210: 0,
211: 4, 212: 1, 213: 1, 214: 1, 215: 4, 216: 4, 217: 1, 218: 0, 219: 5, 220: 0,
221: 1, 222: 0, 223: 2, 224: 2, 225: 1, 226: 1, 227: 3, 228: 3, 229: 0, 230: 4,
231: 5, 232: 1, 233: 4, 234: 3, 235: 2, 236: 0, 237: 3, 238: 1, 239: 5, 240: 1,
241: 4, 242: 2, 243: 2, 244: 2, 245: 5, 246: 1, 247: 1, 248: 4, 249: 4, 250: 3,
251: 0, 252: 0, 253: 0, 254: 3, 255: 5, 256: 4, 257: 2, 258: 3, 259: 2, 260: 3,
261: 0, 262: 4, 263: 0, 264: 0, 265: 5, 266: 4, 267: 4, 268: 5, 269: 1, 270: 1,
271: 2, 272: 2, 273: 4, 274: 4, 275: 5, 276: 2, 277: 5, 278: 1, 279: 1, 280: 2,
281: 2, 282: 5, 283: 3, 284: 5, 285: 1, 286: 4, 287: 1, 288: 4, 289: 1, 290: 1,
291: 5, 292: 2, 293: 4, 294: 1, 295: 5, 296: 5, 297: 3, 298: 5, 299: 0, 300: 4,
301: 1, 302: 4, 303: 3, 304: 0, 305: 0, 306: 1, 307: 1, 308: 4, 309: 1, 310: 4,
311: 0, 312: 4, 313: 1, 314: 0, 315: 1, 316: 1, 317: 4, 318: 0, 319: 1, 320: 2,
321: 5, 322: 1, 323: 3, 324: 1, 325: 1, 326: 0, 327: 0, 328: 1, 329: 0, 330: 3,
331: 2, 332: 2, 333: 0, 334: 0, 335: 5, 336: 4, 337: 3, 338: 3, 339: 5, 340: 2,
341: 1, 342: 3, 343: 5, 344: 5, 345: 0, 346: 5, 347: 3, 348: 2, 349: 3, 350: 5,
351: 5, 352: 0, 353: 2, 354: 5, 355: 3, 356: 4, 357: 4, 358: 0, 359: 4, 360: 0,
361: 0, 362: 4, 363: 2, 364: 0, 365: 3, 366: 5, 367: 1, 368: 5, 369: 4, 370: 3,
371: 3, 372: 4, 373: 1, 374: 2, 375: 1, 376: 4, 377: 0, 378: 4, 379: 0, 380: 4,
381: 0, 382: 4, 383: 3, 384: 4, 385: 1, 386: 4, 387: 4, 388: 2, 389: 2, 390: 4,
391: 2, 392: 4, 393: 2, 394: 4, 395: 4, 396: 1, 397: 0, 398: 3, 399: 3, 400: 2,
401: 2, 402: 3, 403: 5, 404: 5, 405: 1, 406: 2, 407: 5, 408: 3, 409: 1, 410: 0,
411: 1, 412: 5, 413: 1, 414: 1, 415: 0, 416: 1, 417: 0, 418: 4, 419: 0, 420: 0,
421: 0, 422: 1, 423: 3, 424: 5, 425: 3, 426: 1, 427: 0, 428: 1, 429: 1, 430: 5,
431: 1, 432: 3, 433: 5, 434: 1, 435: 5, 436: 5, 437: 2, 438: 0, 439: 5, 440: 0,
441: 0, 442: 3, 443: 1, 444: 1, 445: 0, 446: 4, 447: 1, 448: 0, 449: 0, 450: 0,
451: 2, 452: 1, 453: 4, 454: 5, 455: 1, 456: 3, 457: 1, 458: 5, 459: 3, 460: 4,
461: 3, 462: 4, 463: 0, 464: 2, 465: 5, 466: 5, 467: 5, 468: 5, 469: 4, 470: 4,
471: 0, 472: 2, 473: 4, 474: 4, 475: 4, 476: 4, 477: 3, 478: 4, 479: 1, 480: 5,
481: 4, 482: 5, 483: 1, 484: 0, 485: 1, 486: 2, 487: 2, 488: 3, 489: 4, 490: 1,
491: 4, 492: 2, 493: 2, 494: 2, 495: 3, 496: 4, 497: 0, 498: 3, 499: 2}
```

[235]:
```python
## Group dictionary structure of the graph by values,
## So each value maps to a list of different keys

grouped_dict = defaultdict(list)
for key, val in sorted(communities_Gpow.items()):
    grouped_dict[val].append(key)
print("\n**Grouped dictionary is : " + str(dict(grouped_dict)))

print("\n\n##,type= ",type(grouped_dict.values()),grouped_dict.values())
cummunities_l_pow = []
for vv in grouped_dict.values():
    cummunities_l_pow.append(vv)

print("\n**List of grouped-keys'list ",cummunities_l_pow)
```

```
**Grouped dictionary is : {2: [0, 8, 17, 18, 34, 38, 43, 48, 57, 58, 59, 65, 75,
80, 86, 90, 91, 96, 137, 138, 157, 164, 177, 195, 197, 203, 209, 223, 224, 235,
242, 243, 244, 257, 259, 271, 272, 276, 280, 281, 292, 320, 331, 332, 340, 348,
353, 363, 374, 388, 389, 391, 393, 400, 401, 406, 437, 451, 464, 472, 486, 487,
492, 493, 494, 499], 1: [1, 7, 12, 13, 25, 44, 47, 50, 52, 62, 66, 71, 77, 87,
97, 103, 105, 106, 113, 120, 123, 124, 126, 134, 144, 148, 156, 167, 168, 173,
176, 181, 182, 183, 184, 185, 198, 212, 213, 214, 217, 221, 225, 226, 232, 238,
240, 246, 247, 269, 270, 278, 279, 285, 287, 289, 290, 294, 301, 306, 307, 309,
313, 315, 316, 319, 322, 324, 325, 328, 341, 367, 373, 375, 385, 396, 405, 409,
411, 413, 414, 416, 422, 426, 428, 429, 431, 434, 443, 444, 447, 452, 455, 457,
479, 483, 485, 490], 3: [2, 6, 31, 33, 35, 41, 49, 54, 60, 69, 79, 93, 102, 119,
129, 131, 132, 135, 136, 139, 142, 153, 171, 178, 179, 192, 194, 200, 227, 228,
234, 237, 250, 254, 258, 260, 283, 297, 303, 323, 330, 337, 338, 342, 347, 349,
355, 365, 370, 371, 383, 398, 399, 402, 408, 423, 425, 432, 442, 456, 459, 461,
477, 488, 495, 498], 4: [3, 9, 10, 26, 27, 32, 45, 55, 63, 64, 68, 74, 76, 78,
82, 84, 85, 88, 107, 108, 114, 116, 121, 122, 125, 127, 143, 146, 147, 149, 152,
154, 155, 163, 166, 175, 187, 190, 199, 202, 205, 211, 215, 216, 230, 233, 241,
248, 249, 256, 262, 266, 267, 273, 274, 286, 288, 293, 300, 302, 308, 310, 312,
317, 336, 356, 357, 359, 362, 369, 372, 376, 378, 380, 382, 384, 386, 387, 390,
392, 394, 395, 418, 446, 453, 460, 462, 469, 470, 473, 474, 475, 476, 478, 481,
489, 491, 496], 5: [4, 14, 15, 19, 23, 28, 30, 40, 67, 70, 72, 81, 89, 95, 98,
99, 100, 109, 110, 111, 112, 118, 128, 130, 133, 141, 145, 150, 158, 159, 161,
162, 172, 180, 196, 201, 204, 206, 219, 231, 239, 245, 255, 265, 268, 275, 277,
282, 284, 291, 295, 296, 298, 321, 335, 339, 343, 344, 346, 350, 351, 354, 366,
368, 403, 404, 407, 412, 424, 430, 433, 435, 436, 439, 454, 458, 465, 466, 467,
468, 480, 482], 0: [5, 11, 16, 20, 21, 22, 24, 29, 36, 37, 39, 42, 46, 51, 53,
56, 61, 73, 83, 92, 94, 101, 104, 115, 117, 140, 151, 160, 165, 169, 170, 174,
186, 188, 189, 191, 193, 207, 208, 210, 218, 220, 222, 229, 236, 251, 252, 253,
261, 263, 264, 299, 304, 305, 311, 314, 318, 326, 327, 329, 333, 334, 345, 352,
358, 360, 361, 364, 377, 379, 381, 397, 410, 415, 417, 419, 420, 421, 427, 438,
440, 441, 445, 448, 449, 450, 463, 471, 484, 497]}


##,type=  <class 'dict_values'> dict_values([[0, 8, 17, 18, 34, 38, 43, 48, 57,
58, 59, 65, 75, 80, 86, 90, 91, 96, 137, 138, 157, 164, 177, 195, 197, 203, 209,
223, 224, 235, 242, 243, 244, 257, 259, 271, 272, 276, 280, 281, 292, 320, 331,
332, 340, 348, 353, 363, 374, 388, 389, 391, 393, 400, 401, 406, 437, 451, 464,
472, 486, 487, 492, 493, 494, 499], [1, 7, 12, 13, 25, 44, 47, 50, 52, 62, 66,
71, 77, 87, 97, 103, 105, 106, 113, 120, 123, 124, 126, 134, 144, 148, 156, 167,
168, 173, 176, 181, 182, 183, 184, 185, 198, 212, 213, 214, 217, 221, 225, 226,
232, 238, 240, 246, 247, 269, 270, 278, 279, 285, 287, 289, 290, 294, 301, 306,
307, 309, 313, 315, 316, 319, 322, 324, 325, 328, 341, 367, 373, 375, 385, 396,
405, 409, 411, 413, 414, 416, 422, 426, 428, 429, 431, 434, 443, 444, 447, 452,
455, 457, 479, 483, 485, 490], [2, 6, 31, 33, 35, 41, 49, 54, 60, 69, 79, 93,
102, 119, 129, 131, 132, 135, 136, 139, 142, 153, 171, 178, 179, 192, 194, 200,
227, 228, 234, 237, 250, 254, 258, 260, 283, 297, 303, 323, 330, 337, 338, 342,
347, 349, 355, 365, 370, 371, 383, 398, 399, 402, 408, 423, 425, 432, 442, 456,
459, 461, 477, 488, 495, 498], [3, 9, 10, 26, 27, 32, 45, 55, 63, 64, 68, 74,
76, 78, 82, 84, 85, 88, 107, 108, 114, 116, 121, 122, 125, 127, 143, 146, 147,
149, 152, 154, 155, 163, 166, 175, 187, 190, 199, 202, 205, 211, 215, 216, 230,
233, 241, 248, 249, 256, 262, 266, 267, 273, 274, 286, 288, 293, 300, 302, 308,
310, 312, 317, 336, 356, 357, 359, 362, 369, 372, 376, 378, 380, 382, 384, 386,
387, 390, 392, 394, 395, 418, 446, 453, 460, 462, 469, 470, 473, 474, 475, 476,
478, 481, 489, 491, 496], [4, 14, 15, 19, 23, 28, 30, 40, 67, 70, 72, 81, 89,
95, 98, 99, 100, 109, 110, 111, 112, 118, 128, 130, 133, 141, 145, 150, 158,
159, 161, 162, 172, 180, 196, 201, 204, 206, 219, 231, 239, 245, 255, 265, 268,
275, 277, 282, 284, 291, 295, 296, 298, 321, 335, 339, 343, 344, 346, 350, 351,
354, 366, 368, 403, 404, 407, 412, 424, 430, 433, 435, 436, 439, 454, 458, 465,
466, 467, 468, 480, 482], [5, 11, 16, 20, 21, 22, 24, 29, 36, 37, 39, 42, 46,
51, 53, 56, 61, 73, 83, 92, 94, 101, 104, 115, 117, 140, 151, 160, 165, 169,
170, 174, 186, 188, 189, 191, 193, 207, 208, 210, 218, 220, 222, 229, 236, 251,
252, 253, 261, 263, 264, 299, 304, 305, 311, 314, 318, 326, 327, 329, 333, 334,
345, 352, 358, 360, 361, 364, 377, 379, 381, 397, 410, 415, 417, 419, 420, 421,
427, 438, 440, 441, 445, 448, 449, 450, 463, 471, 484, 497]])

**List of grouped-keys'list  [[0, 8, 17, 18, 34, 38, 43, 48, 57, 58, 59, 65, 75,
80, 86, 90, 91, 96, 137, 138, 157, 164, 177, 195, 197, 203, 209, 223, 224, 235,
242, 243, 244, 257, 259, 271, 272, 276, 280, 281, 292, 320, 331, 332, 340, 348,
353, 363, 374, 388, 389, 391, 393, 400, 401, 406, 437, 451, 464, 472, 486, 487,
492, 493, 494, 499], [1, 7, 12, 13, 25, 44, 47, 50, 52, 62, 66, 71, 77, 87, 97,
103, 105, 106, 113, 120, 123, 124, 126, 134, 144, 148, 156, 167, 168, 173, 176,
181, 182, 183, 184, 185, 198, 212, 213, 214, 217, 221, 225, 226, 232, 238, 240,
246, 247, 269, 270, 278, 279, 285, 287, 289, 290, 294, 301, 306, 307, 309, 313,
315, 316, 319, 322, 324, 325, 328, 341, 367, 373, 375, 385, 396, 405, 409, 411,
413, 414, 416, 422, 426, 428, 429, 431, 434, 443, 444, 447, 452, 455, 457, 479,
483, 485, 490], [2, 6, 31, 33, 35, 41, 49, 54, 60, 69, 79, 93, 102, 119, 129,
131, 132, 135, 136, 139, 142, 153, 171, 178, 179, 192, 194, 200, 227, 228, 234,
237, 250, 254, 258, 260, 283, 297, 303, 323, 330, 337, 338, 342, 347, 349, 355,
365, 370, 371, 383, 398, 399, 402, 408, 423, 425, 432, 442, 456, 459, 461, 477,
488, 495, 498], [3, 9, 10, 26, 27, 32, 45, 55, 63, 64, 68, 74, 76, 78, 82, 84,
85, 88, 107, 108, 114, 116, 121, 122, 125, 127, 143, 146, 147, 149, 152, 154,
155, 163, 166, 175, 187, 190, 199, 202, 205, 211, 215, 216, 230, 233, 241, 248,
249, 256, 262, 266, 267, 273, 274, 286, 288, 293, 300, 302, 308, 310, 312, 317,
336, 356, 357, 359, 362, 369, 372, 376, 378, 380, 382, 384, 386, 387, 390, 392,
394, 395, 418, 446, 453, 460, 462, 469, 470, 473, 474, 475, 476, 478, 481, 489,
491, 496], [4, 14, 15, 19, 23, 28, 30, 40, 67, 70, 72, 81, 89, 95, 98, 99, 100,
109, 110, 111, 112, 118, 128, 130, 133, 141, 145, 150, 158, 159, 161, 162, 172,
180, 196, 201, 204, 206, 219, 231, 239, 245, 255, 265, 268, 275, 277, 282, 284,
291, 295, 296, 298, 321, 335, 339, 343, 344, 346, 350, 351, 354, 366, 368, 403,
404, 407, 412, 424, 430, 433, 435, 436, 439, 454, 458, 465, 466, 467, 468, 480,
482], [5, 11, 16, 20, 21, 22, 24, 29, 36, 37, 39, 42, 46, 51, 53, 56, 61, 73,
83, 92, 94, 101, 104, 115, 117, 140, 151, 160, 165, 169, 170, 174, 186, 188,
189, 191, 193, 207, 208, 210, 218, 220, 222, 229, 236, 251, 252, 253, 261, 263,
264, 299, 304, 305, 311, 314, 318, 326, 327, 329, 333, 334, 345, 352, 358, 360,
361, 364, 377, 379, 381, 397, 410, 415, 417, 419, 420, 421, 427, 438, 440, 441,
445, 448, 449, 450, 463, 471, 484, 497]]
```

Plot Gpow graph

[226]:
```python
posGpow = nx.spring_layout(Gpow)

plt.figure(figsize=(12,8))
nx.draw_networkx(Gpow, pos=posGpow, node_color='green', edge_color='darkgray',␣
 ↪width=3, node_size=640)
limits = plt.axis('off')
```



[237]:
```
mod = modularity(Gpow,cummunities_l_pow)
print("\n Modularity of Gpow graph, communities with Louvain method",mod)
```

```
 Modularity of Gpow graph, communities with Louvain method 0.027635665795158124
```