In [2]:
```python
# This file must be able to determine the smaller Ramsey numbers
import numpy as np
import function_grouping_homomorphics as gh
import function_bron_kerbosch_clique_finder as bk
import function_matrix_to_edge_connection as mx
import function_complement_graph as cm
import function_adjacency1 as aj
import function_drawing as dr
```

In [ ]:
```python
#%%
def Rams_comp_update2(k, l):
    k = k
    l = l
    n = 1  # Start the search at 2 nodes and increment until conditions a
    all_graphs_meet_condition = False
    # Base case for k = 2 and l = 2
    # if k == 2 and l == 2:
    #     print("The Ramsey number is 2.")
    #     return n
    if l == 1 or k ==1 :
        all_graphs_meet_condition = True

    while not all_graphs_meet_condition:
        # Step 1: Build the adjacency matrices
        graphs = aj.all_adj_matrix(n)

        # Step 2: Find all the non-homomorphic graphs
        n_h_graphs1 = gh.group_matrices_by_ones(graphs)
        n_h_graphs = n_h_graphs1[0]  # Homomorphics function returns the

        # Step 3: Initialize a list to track which graphs meet the condit
        these_graphs_meet_condition_list = []

        # Step 4: Iterate over each graph and check for conditions
        for graph in n_h_graphs:

            # Step 5: Find the complement graph for l-set
            complement_graph = cm.complement_graphs(graph)

            # Step 6: Convert to dictionary format for processing
            k_graph = mx.adj_mat_dict(graph)
            l_graph = mx.adj_mat_dict(complement_graph)


            # Step 7: Find maximal cliques in both k_graph and l_graph
            clique_set_k = bk.MaximalCliquesFinder(k_graph)
            clique_set_k.find_cliques()
            clique_list_k = clique_set_k.list_of_cliques()

            clique_set_l = bk.MaximalCliquesFinder(l_graph)
            clique_set_l.find_cliques()
            clique_list_l = clique_set_l.list_of_cliques()

            # Step 8: Find the largest cliques in both k_graph and l_grap
            k_max = max(clique_list_k, key= len)
```

```
            l_max = max(clique_list_l, key= len)
            length_k = len(k_max)
            length_l = len(l_max)



            # Step 9: Check if the graph meets at least one condition
            if length_k < k and length_l < l:
                # If neither condition is met, break out of the loop and
                # these_graphs_meet_condition = 0
                graphs = []
                n += 1

                break  # Exit the inner loop to restart with a new graph
            elif length_k >= k or length_l >= l :
                # If the condition is met, mark this graph as successful
                these_graphs_meet_condition_list.append(1)

        # If all graphs meet at least one condition, we have found the Ra
        if len(these_graphs_meet_condition_list) == len(n_h_graphs):
            all_graphs_meet_condition = True
        else:
            continue

    return n
```