Cliques

1. Use NetworkX to create a graph and add edges.

```
G = nx.random_graphs.random_regular_graph(3, 30, seed=42)
```

---

2. Implement a function to find all cliques of a specific size.

```
def find_cliques_n(graph: Graph, n: int) -> list:
    cliques = []
    for clique in nx.find_cliques(graph):
        if len(clique) == n:
            cliques.append(clique)

    return cliques
```

---

3. Test the function with a graph and a given clique size `n`.

```
G = nx.random_graphs.random_regular_graph(3, 30, seed=42)

result = find_cliques_n(G, 2)
print(result)
# [[0, 25], [0, 26], [0, 18], [1, 4], [2, 8], [2, 19], [2, 5], [3, 10], [3, 21], [3,
13], [4, 11], [4, 13], [5, 9], [5, 15], [6, 24], [7, 24], [7, 27], [7, 20], [8, 18],
[8, 29], [9, 14], [9, 23], [10, 20], [10, 23], [11, 24], [11, 21], [12, 25], [12, 26],
[12, 19], [13, 16], [14, 21], [14, 22], [15, 27], [15, 22], [16, 25], [16, 20], [17,
27], [19, 28], [22, 28], [23, 18], [28, 29], [29, 26]]
```

---

4. Using some simple random graphs, discuss how the computation time growing with the size of the graph (number of nodes and/or number of edges).

```
G = nx.random_graphs.random_regular_graph(3, 30, seed=42)
result = find_cliques_n(G, 2)
G = nx.random_graphs.random_regular_graph(4, 40, seed=42)
result = find_cliques_n(G, 2)
G = nx.random_graphs.random_regular_graph(5, 50, seed=42)
result = find_cliques_n(G, 2)
```

The function `find_cliques_n(graph, n)` finds all cliques in the graph with size `n`. Internally, it calls the `find_cliques(graph)` function from NetworkX, the implementation of which is based on the algorithm published by Bron and Kerbosch (1973) [1], as adapted by Tomita, Tanaka and Takahashi (2006) [2]. This algorithm has the worst-case time complexity of $O(3^{n/3})$, with n being the size of the graph [2]. Given that `find_cliques_n(graph, n)` iterates through all the cliques returned by the function, if we consider number of said cliques as m, then the function overall will have a time complexity of $O(3^{n/3}m)$.

Hamiltonian Circuits

1. What is the value returned by `find_hamiltonian_circuit`? (type and possible values)

```
def find_hamiltonian_circuit(graph: nx.Graph) -> Optional[List[int]]:
    """Find the Hamiltonian circuit in the given graph if it exists."""
    path: List[int] = [-1] * len(graph.nodes())
```

```python
    # Let vertex 0 be the first vertex in the path
    path[0] = list(graph.nodes())[0]

    if not hamiltonian_circuit_util(graph, path, 1):
        print("No Hamiltonian circuit exists")
        return None
    else:
        # Return to the starting point
        path.append(path[0])
        print("Hamiltonian circuit exists:\n", path)
        return path
```

The `find_hamiltonian_circuit` function returns a value of type `Optional[List[int]]`.

This means that the function can return either a list of integers or None.

The list of integers represents a Hamiltonian circuit in the given graph. A Hamiltonian circuit is a path that visits each node in the graph exactly once and returns to the starting node. The integers in the list are the nodes in the order they are visited in the circuit.

If no Hamiltonian circuit exists in the graph, the function returns None.

---

2. How is the backtracking implemented? Explain the relevant line(s) of code.

```python
def hamiltonian_circuit_util(graph: nx.Graph, path: List[int], pos: int) -> bool:
    """Utility function to solve Hamiltonian Circuit problem using Backtracking."""
    # Base case: if all vertices are in the path and there is an edge from the last
included vertex to the first vertex
    if pos == len(graph.nodes()) and graph.has_edge(path[pos - 1], path[0]):
        return True

    # Try different vertices as the next candidate in Hamiltonian Circuit
    for v in graph.nodes():
        if is_valid_vertex(graph, v, pos, path):
            path[pos] = v
            if hamiltonian_circuit_util(graph, path, pos + 1):
                return True
            # Remove current vertex if it doesn't lead to a solution
            path[pos] = -1

    return False
```

The function iterates over all vertices in the graph with for v in `graph.nodes()`:

For each vertex v, it checks if v can be added to the current path with if `is_valid_vertex(graph, v, pos, path)`:. A vertex can be added if it is adjacent to the last vertex in the path and it has not been included in the path yet.

If v can be added to the path, it is added with `path[pos] = v`.

The function then makes a recursive call to itself with if `hamiltonian_circuit_util(graph, path, pos + 1)`: to try to add the next vertex to the path. The pos + 1 argument means that the next position in the path is considered.

If the recursive call returns True, this means that a Hamiltonian circuit has been found, and the function returns True with `return True`.

If the recursive call returns False, this means that adding `v` to the path does not lead to a Hamiltonian circuit. In this case, `v` is removed from the path with `path[pos] = -1`. This is the backtracking step: it undoes the addition of `v` to the path, and the function will continue with the next iteration of the loop to try a different vertex.

This process continues until all vertices have been tried. If no Hamiltonian circuit is found, the function returns False with `return False`.

---

3. Run the program on various small graph. Plot the results and check if the solutions are correct (check also with graphs for which no hamiltonian path exist)

```python
def plot_graph(graph: nx.Graph, path: Optional[List[int]]):
    """Plot the graph with the Hamiltonian circuit if it exists."""
    pos = nx.spring_layout(graph)
    nx.draw(graph, pos, with_labels=True)
    if path is not None:
        edges = [(path[i - 1], path[i]) for i in range(1, len(path))]
        nx.draw_networkx_edges(graph, pos, edgelist=edges, edge_color='r', width=2)
    plt.show()

if __name__ == "__main__":
    # Example usage
    graphs = [
         nx.Graph([(0, 1), (0, 2), (1, 2), (2, 3), (3, 0)]),  # Hamiltonian circuit
exists
        nx.Graph([(0, 1), (0, 2), (1, 2), (2, 3)]),  # No Hamiltonian circuit
         nx.Graph([(0, 1), (0, 2), (1, 2), (2, 3), (3, 4), (4, 0)])  # Hamiltonian
circuit exists
    ]

    for G in graphs:
        path = find_hamiltonian_circuit(G)
        plot_graph(G, path)
```

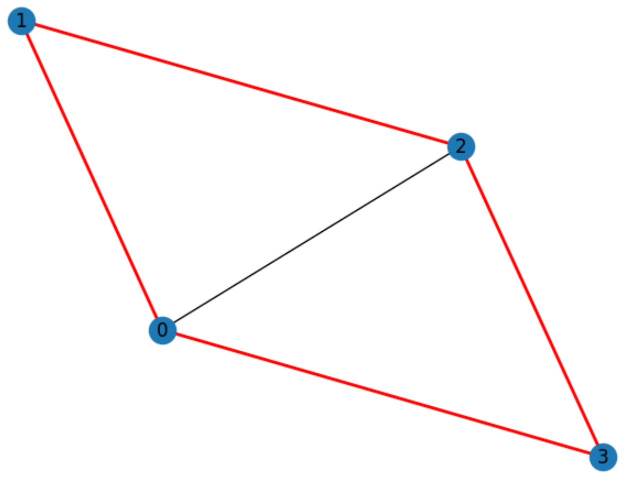

Figure 1: `Hamiltonian circuit exists: [0, 1, 2, 3, 0]`

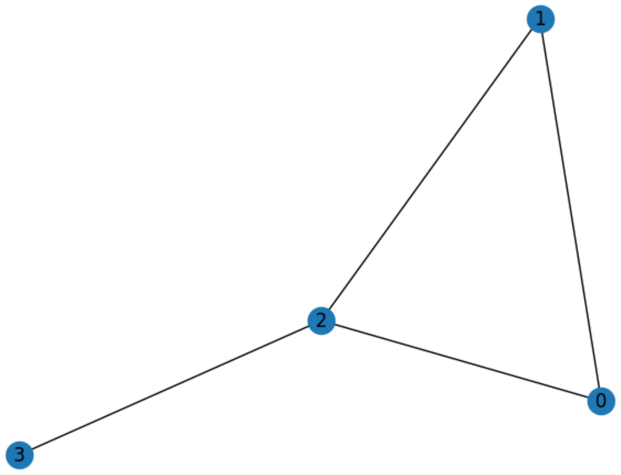

Figure 2: `No Hamiltonian circuit exists`

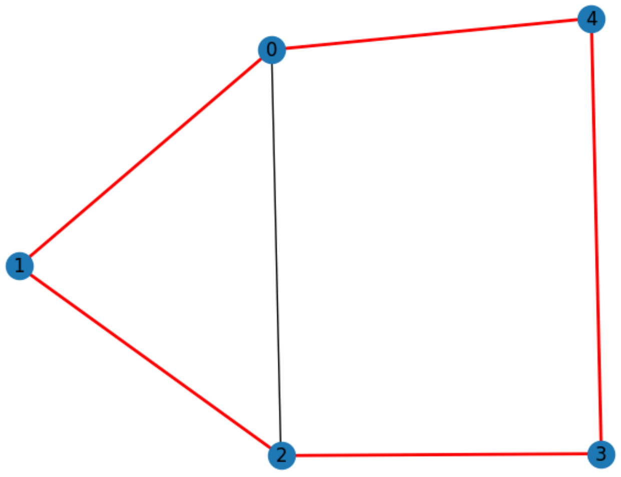

Figure 3: `Hamiltonian circuit exists: [0, 1, 2, 3, 4, 0]`

---

4. Modify the code to count the number of times `hamiltonian_circuit_util` is called (eg using a global variable). Then try running on graphs with various values of n and look how the number of calls is growing with n. Discuss the results.

```
call_count = 0

def hamiltonian_circuit_util(graph: nx.Graph, path: List[int], pos: int) -> bool:
    global call_count
    call_count += 1

    if pos == len(graph.nodes()) and graph.has_edge(path[pos - 1], path[0]):
        return True

    for v in graph.nodes():
        if is_valid_vertex(graph, v, pos, path):
            path[pos] = v
            if hamiltonian_circuit_util(graph, path, pos + 1):
                return True
            path[pos] = -1

    return False

for n in range(1, 10):
    G = nx.complete_graph(n)
    path = find_hamiltonian_circuit(G)
    print(f"For n={n}, hamiltonian_circuit_util was called {call_count} times")
    call_count = 0  # Reset the count for the next run

# No Hamiltonian circuit exists
# For n=1, hamiltonian_circuit_util was called 1 times
# Hamiltonian circuit exists: [0, 1, 0]
# For n=2, hamiltonian_circuit_util was called 2 times
# Hamiltonian circuit exists: [0, 1, 2, 0]
# For n=3, hamiltonian_circuit_util was called 3 times
# Hamiltonian circuit exists: [0, 1, 2, 3, 0]
# For n=4, hamiltonian_circuit_util was called 4 times
# Hamiltonian circuit exists: [0, 1, 2, 3, 4, 0]
# For n=5, hamiltonian_circuit_util was called 5 times
# Hamiltonian circuit exists: [0, 1, 2, 3, 4, 5, 0]
# For n=6, hamiltonian_circuit_util was called 6 times
# Hamiltonian circuit exists: [0, 1, 2, 3, 4, 5, 6, 0]
# For n=7, hamiltonian_circuit_util was called 7 times
# Hamiltonian circuit exists: [0, 1, 2, 3, 4, 5, 6, 7, 0]
# For n=8, hamiltonian_circuit_util was called 8 times
# Hamiltonian circuit exists: [0, 1, 2, 3, 4, 5, 6, 7, 8, 0]
# For n=9, hamiltonian_circuit_util was called 9 times
```

The number of calls to `hamiltonian_circuit_util` grows exponentially with n. This is because the function uses a backtracking algorithm to find a Hamiltonian circuit, which has a worst-case time complexity of O(n!). For each vertex, it tries to add all other vertices to the path, leading to n! possible paths to check. This exponential growth means that the function can become very slow for large values of n.

Bibliography

[1] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph", Communications of the ACM, vol. 16, pp. 575–577, 1973, doi: 10.1145/362342.362367.

[2] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments", Theoretical Computer Science, vol. 363, pp. 28–42, 2006, doi: 10.1016/j.tcs.2006.06.015.