In [ ]:
```python
#bron-kerbosch

#it also requires the graphs to be converted so that we have each node as
#defn of the other nodes its connected to.

class MaximalCliquesFinder:
    def __init__(self, graph): #initialisation, of the adj_mat as self
        self.graph = graph
        self.maximal_cliques = []

    def find_cliques(self):
        nodes = list(self.graph.keys())
        self._extend([], nodes, []) #--compsub[], candidates, not[], this
        #implement the recursion.
        # a = self.maximal_cliques
        # return a #new line to return cliques

    def _extend(self, compsub, candidates, not_set): #looping through new
        if not candidates and not not_set: #if both 'candidate' and 'not'
            self.maximal_cliques.append(compsub)
            return

        # Branch and bound: Choose a pivot
        pivot = candidates[0] if candidates else not_set[0] #1st element
        #pivot = max(candidates, key=lambda node: len(self.graph[node]))

        # Iterate through candidates not connected to the pivot
        for candidate in candidates[:]:
            if candidate in self.graph[pivot]:
                continue

            # New sets for recursion
            new_compsub = compsub + [candidate]
            new_candidates = [v for v in candidates if v in self.graph[ca
            new_not_set = [v for v in not_set if v in self.graph[candidat

            # Recursive call to extend compsub
            self._extend(new_compsub, new_candidates, new_not_set)

            # Move candidate to not_set
            candidates.remove(candidate)
            not_set.append(candidate)

    def print_cliques(self):
        for clique in self.maximal_cliques:
            print(clique)

    def list_of_cliques(self): #This line so we can work with he list of
        a = list(self.maximal_cliques)
        return a
```