2

$$\phi_i(v) = \sum_{S \subseteq D \setminus \{i\}} \frac{|S|!(d - |S| - 1)!}{d!} (v(S \cup \{i\}) - v(S)).$$

2(a)

In [2]:
```
from itertools import permutations

# from the article, change it as you wish
coalition_values = {
    frozenset(): 0,
    frozenset("1"): 1,
    frozenset("2"): 1,
    frozenset("3"): 1,
    frozenset(("1", "2")): 2,
    frozenset(("1", "3")): 2,
    frozenset(("2", "3")): 2,
    frozenset(("1", "2", "3")): 3,
}

def compute_shapley_values(coalition_values, player):
    players = max(coalition_values, key=lambda x: len(x))
    contributions = []

    for permutation in permutations(players):
        player_index = permutation.index(player)
        coalition_before = frozenset(permutation[:player_index])  # excluding player
        coalition_after = frozenset(permutation[: player_index + 1])  # player joined
        #print("After",coalition_after)
        contributions.append(coalition_values[coalition_after] - coalition_values[coal

    return sum(contributions) / len(contributions)  # average, results in Shapley valu

for player in ("1", "2", "3"):
    print(player, compute_shapley_values(coalition_values, player))
```

```
1 1.0
2 1.0
3 1.0
```

In [3]:
```
from itertools import combinations
from math import factorial

coalition_values = {
    frozenset(): 0,
    frozenset("1"): 1,
    frozenset("2"): 1,
    frozenset("3"): 1,
    frozenset(("1", "2")): 2,
    frozenset(("1", "3")): 2,
    frozenset(("2", "3")): 2,
```

```
    frozenset(("1", "2", "3")): 3,
}

def powerset(feature_set):
    """Creating all subsets of a given set."""
    for i in range(len(feature_set) + 1):
        for feature_subset in combinations(feature_set, i):
            yield set(feature_subset)

def compute_shapley_values(coalition_values, i):
    N = max(coalition_values, key=lambda x: len(x))
    n = len(N)
    contribution = 0

    for S in powerset(N - {i}):
        scalar = factorial(len(S)) * factorial(n - len(S) - 1)
        coalition_before = frozenset(S)
        coalition_after = frozenset(S | {i})
        contribution += scalar * (coalition_values[coalition_after] - coalition_values

    return contribution / factorial(n)

for player in ("1", "2", "3"):
    print(player, compute_shapley_values(coalition_values, player))
```

```
1 1.0
2 1.0
3 1.0
```

2(b)

In [4]:

```
from itertools import combinations
from math import factorial

# These coalition values were calculated manually from the
# function in the question
coalition_values = {
    frozenset(): 0,
    frozenset("1"): 1,
    frozenset("2"): 2,
    frozenset("3"): 3,
    frozenset(("1", "2")): 3,
    frozenset(("1", "3")): 4,
    frozenset(("2", "3")): 5,
    frozenset(("1", "2", "3")): 6,
}

def powerset(feature_set):
    """Creating all subsets of a given set."""
    for i in range(len(feature_set) + 1):
        for feature_subset in combinations(feature_set, i):
            yield set(feature_subset)

def compute_shapley_values(coalition_values, i):
    N = max(coalition_values, key=lambda x: len(x))
    n = len(N)
    contribution = 0

    for S in powerset(N - {i}):
```

```
        scalar = factorial(len(S)) * factorial(n - len(S) - 1)
        coalition_before = frozenset(S)
        coalition_after = frozenset(S | {i})
        contribution += scalar * (coalition_values[coalition_after] - coalition_values

    return contribution / factorial(n)

for player in ("1", "2", "3"):
    print(player, compute_shapley_values(coalition_values, player))
```

```
1 1.0
2 2.0
3 3.0
```

2(c)

In [5]:

```
from itertools import combinations
from math import factorial

def shapley_values(v, players):
    n = len(players)

    shapley = {player: 0 for player in players}

    for player in players:
        for S in powerset(players):
            if player in S:
                S_without_player = set(S) - {player}
                marginal_contribution = v(S) - v(S_without_player)
                weight = (factorial(len(S_without_player)) * factorial(n - len(S))) /
                shapley[player] += weight * marginal_contribution

    return shapley

def powerset(s):
    return [set(subset) for l in range(len(s) + 1) for subset in combinations(s, l)]

def v(S):
    return S.intersection({2}).__len__() + S.intersection({3}).__len__() + 2*S.interse

# Set of players
players = {1, 2, 3, 4, 5}

shapley_values_dict = shapley_values(v, players)
for player, value in shapley_values_dict.items():
    print(f"Player {player}: {value:.2f}")
```

```
Player 1: 0.00
Player 2: 1.00
Player 3: 1.00
Player 4: 2.00
Player 5: 0.00
```

In [ ]: