104291-Combinatorics-Algorithm-PA1-Q2

Contributors

`Zixuan Guan. guan09236@gtiit.edu.cn` who writes the codes, report and README.

`Houjian Qin. qin09376@gtiit.edu.cn` who gives constructive suggestions.

Algorithm Details

Algorithm for Generating and Sorting Subsets

Pseudo-code

Algorithm 1: Main algorithm for managing 2-colored permutations

```
Function GenerateAll2ColoredPermutations(n):
    Input: An integer n (size of the set)
    Output: Yields all 2-colored permutations of the set {1, ..., n}
    base_perm ← list of all permutations of {1, ..., n};
    for perm in base_perm do
        for colors in 0 to 2^n − 1 do
            yield [(perm[i], (colors >> i)&1) for i in range(n)];
        end
    end
end
Function PrintPermutation(perm):
    Input: A permutation perm
    Print perm as formatted string;
end
Function FindSuccessor(perm):
    Input: A permutation perm
    Output: The next permutation in lexicographical order, or None if last
    all_perms ← list of all 2-colored permutations of size of perm;
    index ← index of perm in all_perms;
    if index is last in all_perms then
        return None;
    return all_perms[index + 1];
end
Function PermutationRank(perm):
    Input: A permutation perm
    Output: The rank of perm among all permutations of its size
    all_perms ← list of all 2-colored permutations of size of perm;
    return index of perm in all_perms;
end
Function FindPermutationByRank(n, rank):
    Input: Size n, rank rank
    Output: The permutation at the given rank
    all_perms ← list of all 2-colored permutations of size n;
    if rank is within bounds of all_perms then
        return all_perms[rank];
    return None;
end
Function GetValidPermutationInput(n):
    Input: An integer n
    Output: A valid user-entered permutation
    repeat
        input_str ← get input from user;
        perm ← parse input_str into permutation;
        if perm is valid then
            return perm;
        print "Invalid input. Please try again.";
    until input is valid;
end
Function Main():
    Display menu and handle user choices using above functions;
end
```

Natural Language Description of the Algorithm

The algorithm consists of several key functions that interact to generate, manipulate, and analyze 2-colored permutations of a set. Here's a simplified, natural language description:

1. **Generating All 2-Colored Permutations:**
   - The algorithm starts by considering all permutations of a set of numbers from 1 to `n`.
   - For each permutation, it then examines every possible "coloring" combination. A "coloring" assigns a binary value (0 or 1) to each element, indicating two possible states for each element in the permutation.
   - These combinations are generated using binary numbers from 0 up to $2^n - 1$. Each bit in a binary number corresponds to an element in the permutation, indicating whether the element is in the first state (0) or the second state (1).
   - The result is a comprehensive list of all permutations, each with every possible coloring, providing a complete set of 2-colored permutations.
2. **Finding the Successor of a Permutation:**
   - To find the successor, the algorithm first generates all 2-colored permutations for a given set size.
   - It then locates the current permutation within this list and returns the next permutation in the sequence. If the current permutation is the last one in the list, the algorithm indicates that there is no successor.
3. **Determining the Rank of a Permutation:**
   - The rank of a permutation is its position in the list of all 2-colored permutations sorted lexicographically.
   - The algorithm calculates this by generating all permutations, sorting them, and then finding the position of the given permutation.
4. **Finding a Permutation by Rank:**
   - This function reverses the process of ranking: given a rank, it returns the permutation at that position in the sorted list of all 2-colored permutations.
5. **Validating User Input:**
   - The algorithm prompts the user to input the elements of a permutation along with their colors.
   - It checks if the input matches the expected format and size, ensuring that all elements are valid and the coloring is correctly specified.

Explanation

Ordering Rules

Criteria for Ordering 2-Colored Permutations The ordering of 2-colored permutations is governed by two primary criteria:

1. **Lexicographical Order Based on Elements:**
   - The permutations are first sorted by the elements themselves, ignoring their colors. This means that permutations are ordered as if they were simple lists of numbers. For example, `(1, 2, 3)` would come before `(1, 3, 2)`.
   - This sort of ordering is akin to how words are arranged in a dictionary, where the position of the letter in the word (or number in the permutation) takes precedence over other attributes.
2. **Secondary Ordering Based on Colors:**
   - Once the permutations are ordered by their elements, the secondary criterion—color—comes into play. This is where permutations that have the same elements but differ in their color assignments are sorted.
   - The color ordering is also lexicographical but follows the primary element order. For example, considering two permutations of the same set of elements `(1, 2, 3)` with color vectors `(0, 1, 0)` and `(0, 0, 1)`, the permutation `(1,0), (2,1), (3,0)` would come before `(1,0), (2,0), (3,1)` because the second element's color in the first permutation is greater than in the second while the rest are equal or evaluated later.

How Color Influences Ordering

- **Bitwise Representation:** Each color is essentially a binary digit associated with each element, where '0' might represent one state (e.g., off or false) and '1' another state (e.g., on or true).
- **Evaluation of Color:** During the sorting process, if two permutations have the same set of numbers, their ordering is determined by comparing their color values starting from the first element to the last. The first difference in color dictates the order.

Example of Ordering Rule Application Consider permutations of the set `{1, 2}` with possible colors 0 and 1:

- Permutations might include:
  - `[(1,0), (2,0)]`
  - `[(1,0), (2,1)]`
  - `[(1,1), (2,0)]`
  - `[(1,1), (2,1)]`
- **Ordering:**
  - First, sort by the first element ignoring color: All permutations where `1` appears first are grouped together.
  - Within those groups, sort by the second element, still ignoring color.
  - Finally, apply color sorting: `[(1,0), (2,0)]` would be sorted before `[(1,0), (2,1)]` because the color of the second element in the first permutation is less than the color in the second permutation.

2-Colored Subset Generation Explanation The generation of 2-colored permutations of a set extends the basic concept of subset generation by not only deciding the presence or absence of elements but also assigning a "color" or additional binary state to each element. Here's how it works:

- **Binary Representation Extended:** Each element in the original set can exist in two states: included with color 0 or included with color 1. This is represented using two bits per element instead of one, effectively doubling the range of binary numbers needed to represent all combinations.
- **Generating 2-Colored Permutations:**
  - Consider a set with elements `{1, 2, 3}`. We create permutations of these elements, then extend each permutation by considering all possible color combinations.
  - For a permutation like `(1, 2, 3)`, each element can be either 0 or 1 (two colors), resulting in `2^3 = 8` combinations per permutation, such as `(1,0), (2,0), (3,0)` or `(1,1), (2,0), (3,1)`, and so on.
- **Permutation Construction:**
  - For each permutation of the set, a binary number is generated for each possible color combination. This binary number is used to assign colors to each element in the permutation.
  - **Example:** For permutation `(1, 2, 3)` and binary color representation `101`, the colored permutation would be `[(1,1), (2,0), (3,1)]`.
- **Efficiency:** The process of generating these 2-colored permutations involves iterating through each permutation of the set and then iterating through each binary color combination. This approach ensures comprehensive coverage of all possible permutations and their color variations.

Sorting and Successor Finding Explanation Sorting and finding successors within these 2-colored permutations follow specific logical steps:

- **Sorting by Lexicographical Order:**
  - Permutations are sorted not just by the numeric sequence of their elements but also considering their colors. This is more complex than sorting regular subsets because each element carries an additional binary attribute.
  - The sorting first considers the natural order of elements and then their colors. For instance, `[(1,0), (2,0), (3,0)]` will precede `[(1,0), (2,1), (3,0)]` due to the color of the second element.

- **Finding Successors:**
  - To find the successor of a given permutation within the sorted list, the algorithm determines the permutation's position in this list and returns the next permutation in sequence.
  - If the permutation is the last in the list, there is no successor, which is a common boundary condition in algorithms dealing with ordered data.
- **Practical Utility:**
  - Sorting these permutations allows for efficient querying, retrieval, and analysis based on both element order and their color states.
  - Applications can range from simulations where permutations represent different states of a system, to combinatorial problems where each color might represent a different attribute or condition.

Detailed Runs

Provide a detailed example using specific values to illustrate the process:

This shows how to run the codes. First enter the directory which contains `main.py` then run the program.

```
python3 main.py
```

Then you can choose a functionality as you want. In our case the first functionality is our goal so we choose 1. Then it requires we enter the size of the n-set. We enter 4. Then we got the result as desired. And the program return to the menu.

```
Menu:
1. Generate all 2-colored permutations of length n and display their ranks
2. Find the successor of a given 2-colored permutation
3. Find a permutation given its rank
4. Find the rank of a given permutation
5. Exit
Enter your choice: 1
Enter the size of the permutation (n): 4
Rank: 0 - (1,0) (2,0) (3,0) (4,0)
Rank: 1 - (1,1) (2,0) (3,0) (4,0)
Rank: 2 - (1,0) (2,1) (3,0) (4,0)
Rank: 3 - (1,1) (2,1) (3,0) (4,0)
Rank: 4 - (1,0) (2,0) (3,1) (4,0)
Rank: 5 - (1,1) (2,0) (3,1) (4,0)
Rank: 6 - (1,0) (2,1) (3,1) (4,0)
Rank: 7 - (1,1) (2,1) (3,1) (4,0)
Rank: 8 - (1,0) (2,0) (3,0) (4,1)
Rank: 9 - (1,1) (2,0) (3,0) (4,1)
Rank: 10 - (1,0) (2,1) (3,0) (4,1)
Rank: 11 - (1,1) (2,1) (3,0) (4,1)
Rank: 12 - (1,0) (2,0) (3,1) (4,1)
Rank: 13 - (1,1) (2,0) (3,1) (4,1)
Rank: 14 - (1,0) (2,1) (3,1) (4,1)
Rank: 15 - (1,1) (2,1) (3,1) (4,1)
Rank: 16 - (1,0) (2,0) (4,0) (3,0)
Rank: 17 - (1,1) (2,0) (4,0) (3,0)
Rank: 18 - (1,0) (2,1) (4,0) (3,0)
Rank: 19 - (1,1) (2,1) (4,0) (3,0)
Rank: 20 - (1,0) (2,0) (4,1) (3,0)
Rank: 21 - (1,1) (2,0) (4,1) (3,0)
Rank: 22 - (1,0) (2,1) (4,1) (3,0)
Rank: 23 - (1,1) (2,1) (4,1) (3,0)
Rank: 24 - (1,0) (2,0) (4,0) (3,1)
```

```
Rank: 25 - (1,1) (2,0) (4,0) (3,1)
Rank: 26 - (1,0) (2,1) (4,0) (3,1)
Rank: 27 - (1,1) (2,1) (4,0) (3,1)
Rank: 28 - (1,0) (2,0) (4,1) (3,1)
Rank: 29 - (1,1) (2,0) (4,1) (3,1)
Rank: 30 - (1,0) (2,1) (4,1) (3,1)
Rank: 31 - (1,1) (2,1) (4,1) (3,1)
Rank: 32 - (1,0) (3,0) (2,0) (4,0)
Rank: 33 - (1,1) (3,0) (2,0) (4,0)
Rank: 34 - (1,0) (3,1) (2,0) (4,0)
Rank: 35 - (1,1) (3,1) (2,0) (4,0)
Rank: 36 - (1,0) (3,0) (2,1) (4,0)
Rank: 37 - (1,1) (3,0) (2,1) (4,0)
Rank: 38 - (1,0) (3,1) (2,1) (4,0)
Rank: 39 - (1,1) (3,1) (2,1) (4,0)
Rank: 40 - (1,0) (3,0) (2,0) (4,1)
Rank: 41 - (1,1) (3,0) (2,0) (4,1)
Rank: 42 - (1,0) (3,1) (2,0) (4,1)
Rank: 43 - (1,1) (3,1) (2,0) (4,1)
Rank: 44 - (1,0) (3,0) (2,1) (4,1)
Rank: 45 - (1,1) (3,0) (2,1) (4,1)
Rank: 46 - (1,0) (3,1) (2,1) (4,1)
Rank: 47 - (1,1) (3,1) (2,1) (4,1)
Rank: 48 - (1,0) (3,0) (4,0) (2,0)
Rank: 49 - (1,1) (3,0) (4,0) (2,0)
Rank: 50 - (1,0) (3,1) (4,0) (2,0)
Rank: 51 - (1,1) (3,1) (4,0) (2,0)
Rank: 52 - (1,0) (3,0) (4,1) (2,0)
Rank: 53 - (1,1) (3,0) (4,1) (2,0)
Rank: 54 - (1,0) (3,1) (4,1) (2,0)
Rank: 55 - (1,1) (3,1) (4,1) (2,0)
Rank: 56 - (1,0) (3,0) (4,0) (2,1)
Rank: 57 - (1,1) (3,0) (4,0) (2,1)
Rank: 58 - (1,0) (3,1) (4,0) (2,1)
Rank: 59 - (1,1) (3,1) (4,0) (2,1)
Rank: 60 - (1,0) (3,0) (4,1) (2,1)
Rank: 61 - (1,1) (3,0) (4,1) (2,1)
Rank: 62 - (1,0) (3,1) (4,1) (2,1)
Rank: 63 - (1,1) (3,1) (4,1) (2,1)
Rank: 64 - (1,0) (4,0) (2,0) (3,0)
Rank: 65 - (1,1) (4,0) (2,0) (3,0)
Rank: 66 - (1,0) (4,1) (2,0) (3,0)
Rank: 67 - (1,1) (4,1) (2,0) (3,0)
Rank: 68 - (1,0) (4,0) (2,1) (3,0)
Rank: 69 - (1,1) (4,0) (2,1) (3,0)
Rank: 70 - (1,0) (4,1) (2,1) (3,0)
Rank: 71 - (1,1) (4,1) (2,1) (3,0)
Rank: 72 - (1,0) (4,0) (2,0) (3,1)
Rank: 73 - (1,1) (4,0) (2,0) (3,1)
Rank: 74 - (1,0) (4,1) (2,0) (3,1)
Rank: 75 - (1,1) (4,1) (2,0) (3,1)
Rank: 76 - (1,0) (4,0) (2,1) (3,1)
Rank: 77 - (1,1) (4,0) (2,1) (3,1)
```

```
Rank: 78 - (1,0) (4,1) (2,1) (3,1)
Rank: 79 - (1,1) (4,1) (2,1) (3,1)
Rank: 80 - (1,0) (4,0) (3,0) (2,0)
Rank: 81 - (1,1) (4,0) (3,0) (2,0)
Rank: 82 - (1,0) (4,1) (3,0) (2,0)
Rank: 83 - (1,1) (4,1) (3,0) (2,0)
Rank: 84 - (1,0) (4,0) (3,1) (2,0)
Rank: 85 - (1,1) (4,0) (3,1) (2,0)
Rank: 86 - (1,0) (4,1) (3,1) (2,0)
Rank: 87 - (1,1) (4,1) (3,1) (2,0)
Rank: 88 - (1,0) (4,0) (3,0) (2,1)
Rank: 89 - (1,1) (4,0) (3,0) (2,1)
Rank: 90 - (1,0) (4,1) (3,0) (2,1)
Rank: 91 - (1,1) (4,1) (3,0) (2,1)
Rank: 92 - (1,0) (4,0) (3,1) (2,1)
Rank: 93 - (1,1) (4,0) (3,1) (2,1)
Rank: 94 - (1,0) (4,1) (3,1) (2,1)
Rank: 95 - (1,1) (4,1) (3,1) (2,1)
Rank: 96 - (2,0) (1,0) (3,0) (4,0)
Rank: 97 - (2,1) (1,0) (3,0) (4,0)
Rank: 98 - (2,0) (1,1) (3,0) (4,0)
Rank: 99 - (2,1) (1,1) (3,0) (4,0)
Rank: 100 - (2,0) (1,0) (3,1) (4,0)
Rank: 101 - (2,1) (1,0) (3,1) (4,0)
Rank: 102 - (2,0) (1,1) (3,1) (4,0)
Rank: 103 - (2,1) (1,1) (3,1) (4,0)
Rank: 104 - (2,0) (1,0) (3,0) (4,1)
Rank: 105 - (2,1) (1,0) (3,0) (4,1)
Rank: 106 - (2,0) (1,1) (3,0) (4,1)
Rank: 107 - (2,1) (1,1) (3,0) (4,1)
Rank: 108 - (2,0) (1,0) (3,1) (4,1)
Rank: 109 - (2,1) (1,0) (3,1) (4,1)
Rank: 110 - (2,0) (1,1) (3,1) (4,1)
Rank: 111 - (2,1) (1,1) (3,1) (4,1)
Rank: 112 - (2,0) (1,0) (4,0) (3,0)
Rank: 113 - (2,1) (1,0) (4,0) (3,0)
Rank: 114 - (2,0) (1,1) (4,0) (3,0)
Rank: 115 - (2,1) (1,1) (4,0) (3,0)
Rank: 116 - (2,0) (1,0) (4,1) (3,0)
Rank: 117 - (2,1) (1,0) (4,1) (3,0)
Rank: 118 - (2,0) (1,1) (4,1) (3,0)
Rank: 119 - (2,1) (1,1) (4,1) (3,0)
Rank: 120 - (2,0) (1,0) (4,0) (3,1)
Rank: 121 - (2,1) (1,0) (4,0) (3,1)
Rank: 122 - (2,0) (1,1) (4,0) (3,1)
Rank: 123 - (2,1) (1,1) (4,0) (3,1)
Rank: 124 - (2,0) (1,0) (4,1) (3,1)
Rank: 125 - (2,1) (1,0) (4,1) (3,1)
Rank: 126 - (2,0) (1,1) (4,1) (3,1)
Rank: 127 - (2,1) (1,1) (4,1) (3,1)
Rank: 128 - (2,0) (3,0) (1,0) (4,0)
Rank: 129 - (2,1) (3,0) (1,0) (4,0)
Rank: 130 - (2,0) (3,1) (1,0) (4,0)
```

```
Rank: 131 - (2,1) (3,1) (1,0) (4,0)
Rank: 132 - (2,0) (3,0) (1,1) (4,0)
Rank: 133 - (2,1) (3,0) (1,1) (4,0)
Rank: 134 - (2,0) (3,1) (1,1) (4,0)
Rank: 135 - (2,1) (3,1) (1,1) (4,0)
Rank: 136 - (2,0) (3,0) (1,0) (4,1)
Rank: 137 - (2,1) (3,0) (1,0) (4,1)
Rank: 138 - (2,0) (3,1) (1,0) (4,1)
Rank: 139 - (2,1) (3,1) (1,0) (4,1)
Rank: 140 - (2,0) (3,0) (1,1) (4,1)
Rank: 141 - (2,1) (3,0) (1,1) (4,1)
Rank: 142 - (2,0) (3,1) (1,1) (4,1)
Rank: 143 - (2,1) (3,1) (1,1) (4,1)
Rank: 144 - (2,0) (3,0) (4,0) (1,0)
Rank: 145 - (2,1) (3,0) (4,0) (1,0)
Rank: 146 - (2,0) (3,1) (4,0) (1,0)
Rank: 147 - (2,1) (3,1) (4,0) (1,0)
Rank: 148 - (2,0) (3,0) (4,1) (1,0)
Rank: 149 - (2,1) (3,0) (4,1) (1,0)
Rank: 150 - (2,0) (3,1) (4,1) (1,0)
Rank: 151 - (2,1) (3,1) (4,1) (1,0)
Rank: 152 - (2,0) (3,0) (4,0) (1,1)
Rank: 153 - (2,1) (3,0) (4,0) (1,1)
Rank: 154 - (2,0) (3,1) (4,0) (1,1)
Rank: 155 - (2,1) (3,1) (4,0) (1,1)
Rank: 156 - (2,0) (3,0) (4,1) (1,1)
Rank: 157 - (2,1) (3,0) (4,1) (1,1)
Rank: 158 - (2,0) (3,1) (4,1) (1,1)
Rank: 159 - (2,1) (3,1) (4,1) (1,1)
Rank: 160 - (2,0) (4,0) (1,0) (3,0)
Rank: 161 - (2,1) (4,0) (1,0) (3,0)
Rank: 162 - (2,0) (4,1) (1,0) (3,0)
Rank: 163 - (2,1) (4,1) (1,0) (3,0)
Rank: 164 - (2,0) (4,0) (1,1) (3,0)
Rank: 165 - (2,1) (4,0) (1,1) (3,0)
Rank: 166 - (2,0) (4,1) (1,1) (3,0)
Rank: 167 - (2,1) (4,1) (1,1) (3,0)
Rank: 168 - (2,0) (4,0) (1,0) (3,1)
Rank: 169 - (2,1) (4,0) (1,0) (3,1)
Rank: 170 - (2,0) (4,1) (1,0) (3,1)
Rank: 171 - (2,1) (4,1) (1,0) (3,1)
Rank: 172 - (2,0) (4,0) (1,1) (3,1)
Rank: 173 - (2,1) (4,0) (1,1) (3,1)
Rank: 174 - (2,0) (4,1) (1,1) (3,1)
Rank: 175 - (2,1) (4,1) (1,1) (3,1)
Rank: 176 - (2,0) (4,0) (3,0) (1,0)
Rank: 177 - (2,1) (4,0) (3,0) (1,0)
Rank: 178 - (2,0) (4,1) (3,0) (1,0)
Rank: 179 - (2,1) (4,1) (3,0) (1,0)
Rank: 180 - (2,0) (4,0) (3,1) (1,0)
Rank: 181 - (2,1) (4,0) (3,1) (1,0)
Rank: 182 - (2,0) (4,1) (3,1) (1,0)
Rank: 183 - (2,1) (4,1) (3,1) (1,0)
```

```
Rank: 184 - (2,0) (4,0) (3,0) (1,1)
Rank: 185 - (2,1) (4,0) (3,0) (1,1)
Rank: 186 - (2,0) (4,1) (3,0) (1,1)
Rank: 187 - (2,1) (4,1) (3,0) (1,1)
Rank: 188 - (2,0) (4,0) (3,1) (1,1)
Rank: 189 - (2,1) (4,0) (3,1) (1,1)
Rank: 190 - (2,0) (4,1) (3,1) (1,1)
Rank: 191 - (2,1) (4,1) (3,1) (1,1)
Rank: 192 - (3,0) (1,0) (2,0) (4,0)
Rank: 193 - (3,1) (1,0) (2,0) (4,0)
Rank: 194 - (3,0) (1,1) (2,0) (4,0)
Rank: 195 - (3,1) (1,1) (2,0) (4,0)
Rank: 196 - (3,0) (1,0) (2,1) (4,0)
Rank: 197 - (3,1) (1,0) (2,1) (4,0)
Rank: 198 - (3,0) (1,1) (2,1) (4,0)
Rank: 199 - (3,1) (1,1) (2,1) (4,0)
Rank: 200 - (3,0) (1,0) (2,0) (4,1)
Rank: 201 - (3,1) (1,0) (2,0) (4,1)
Rank: 202 - (3,0) (1,1) (2,0) (4,1)
Rank: 203 - (3,1) (1,1) (2,0) (4,1)
Rank: 204 - (3,0) (1,0) (2,1) (4,1)
Rank: 205 - (3,1) (1,0) (2,1) (4,1)
Rank: 206 - (3,0) (1,1) (2,1) (4,1)
Rank: 207 - (3,1) (1,1) (2,1) (4,1)
Rank: 208 - (3,0) (1,0) (4,0) (2,0)
Rank: 209 - (3,1) (1,0) (4,0) (2,0)
Rank: 210 - (3,0) (1,1) (4,0) (2,0)
Rank: 211 - (3,1) (1,1) (4,0) (2,0)
Rank: 212 - (3,0) (1,0) (4,1) (2,0)
Rank: 213 - (3,1) (1,0) (4,1) (2,0)
Rank: 214 - (3,0) (1,1) (4,1) (2,0)
Rank: 215 - (3,1) (1,1) (4,1) (2,0)
Rank: 216 - (3,0) (1,0) (4,0) (2,1)
Rank: 217 - (3,1) (1,0) (4,0) (2,1)
Rank: 218 - (3,0) (1,1) (4,0) (2,1)
Rank: 219 - (3,1) (1,1) (4,0) (2,1)
Rank: 220 - (3,0) (1,0) (4,1) (2,1)
Rank: 221 - (3,1) (1,0) (4,1) (2,1)
Rank: 222 - (3,0) (1,1) (4,1) (2,1)
Rank: 223 - (3,1) (1,1) (4,1) (2,1)
Rank: 224 - (3,0) (2,0) (1,0) (4,0)
Rank: 225 - (3,1) (2,0) (1,0) (4,0)
Rank: 226 - (3,0) (2,1) (1,0) (4,0)
Rank: 227 - (3,1) (2,1) (1,0) (4,0)
Rank: 228 - (3,0) (2,0) (1,1) (4,0)
Rank: 229 - (3,1) (2,0) (1,1) (4,0)
Rank: 230 - (3,0) (2,1) (1,1) (4,0)
Rank: 231 - (3,1) (2,1) (1,1) (4,0)
Rank: 232 - (3,0) (2,0) (1,0) (4,1)
Rank: 233 - (3,1) (2,0) (1,0) (4,1)
Rank: 234 - (3,0) (2,1) (1,0) (4,1)
Rank: 235 - (3,1) (2,1) (1,0) (4,1)
Rank: 236 - (3,0) (2,0) (1,1) (4,1)
```

```
Rank: 237 - (3,1) (2,0) (1,1) (4,1)
Rank: 238 - (3,0) (2,1) (1,1) (4,1)
Rank: 239 - (3,1) (2,1) (1,1) (4,1)
Rank: 240 - (3,0) (2,0) (4,0) (1,0)
Rank: 241 - (3,1) (2,0) (4,0) (1,0)
Rank: 242 - (3,0) (2,1) (4,0) (1,0)
Rank: 243 - (3,1) (2,1) (4,0) (1,0)
Rank: 244 - (3,0) (2,0) (4,1) (1,0)
Rank: 245 - (3,1) (2,0) (4,1) (1,0)
Rank: 246 - (3,0) (2,1) (4,1) (1,0)
Rank: 247 - (3,1) (2,1) (4,1) (1,0)
Rank: 248 - (3,0) (2,0) (4,0) (1,1)
Rank: 249 - (3,1) (2,0) (4,0) (1,1)
Rank: 250 - (3,0) (2,1) (4,0) (1,1)
Rank: 251 - (3,1) (2,1) (4,0) (1,1)
Rank: 252 - (3,0) (2,0) (4,1) (1,1)
Rank: 253 - (3,1) (2,0) (4,1) (1,1)
Rank: 254 - (3,0) (2,1) (4,1) (1,1)
Rank: 255 - (3,1) (2,1) (4,1) (1,1)
Rank: 256 - (3,0) (4,0) (1,0) (2,0)
Rank: 257 - (3,1) (4,0) (1,0) (2,0)
Rank: 258 - (3,0) (4,1) (1,0) (2,0)
Rank: 259 - (3,1) (4,1) (1,0) (2,0)
Rank: 260 - (3,0) (4,0) (1,1) (2,0)
Rank: 261 - (3,1) (4,0) (1,1) (2,0)
Rank: 262 - (3,0) (4,1) (1,1) (2,0)
Rank: 263 - (3,1) (4,1) (1,1) (2,0)
Rank: 264 - (3,0) (4,0) (1,0) (2,1)
Rank: 265 - (3,1) (4,0) (1,0) (2,1)
Rank: 266 - (3,0) (4,1) (1,0) (2,1)
Rank: 267 - (3,1) (4,1) (1,0) (2,1)
Rank: 268 - (3,0) (4,0) (1,1) (2,1)
Rank: 269 - (3,1) (4,0) (1,1) (2,1)
Rank: 270 - (3,0) (4,1) (1,1) (2,1)
Rank: 271 - (3,1) (4,1) (1,1) (2,1)
Rank: 272 - (3,0) (4,0) (2,0) (1,0)
Rank: 273 - (3,1) (4,0) (2,0) (1,0)
Rank: 274 - (3,0) (4,1) (2,0) (1,0)
Rank: 275 - (3,1) (4,1) (2,0) (1,0)
Rank: 276 - (3,0) (4,0) (2,1) (1,0)
Rank: 277 - (3,1) (4,0) (2,1) (1,0)
Rank: 278 - (3,0) (4,1) (2,1) (1,0)
Rank: 279 - (3,1) (4,1) (2,1) (1,0)
Rank: 280 - (3,0) (4,0) (2,0) (1,1)
Rank: 281 - (3,1) (4,0) (2,0) (1,1)
Rank: 282 - (3,0) (4,1) (2,0) (1,1)
Rank: 283 - (3,1) (4,1) (2,0) (1,1)
Rank: 284 - (3,0) (4,0) (2,1) (1,1)
Rank: 285 - (3,1) (4,0) (2,1) (1,1)
Rank: 286 - (3,0) (4,1) (2,1) (1,1)
Rank: 287 - (3,1) (4,1) (2,1) (1,1)
Rank: 288 - (4,0) (1,0) (2,0) (3,0)
Rank: 289 - (4,1) (1,0) (2,0) (3,0)
```

```
Rank: 290 - (4,0) (1,1) (2,0) (3,0)
Rank: 291 - (4,1) (1,1) (2,0) (3,0)
Rank: 292 - (4,0) (1,0) (2,1) (3,0)
Rank: 293 - (4,1) (1,0) (2,1) (3,0)
Rank: 294 - (4,0) (1,1) (2,1) (3,0)
Rank: 295 - (4,1) (1,1) (2,1) (3,0)
Rank: 296 - (4,0) (1,0) (2,0) (3,1)
Rank: 297 - (4,1) (1,0) (2,0) (3,1)
Rank: 298 - (4,0) (1,1) (2,0) (3,1)
Rank: 299 - (4,1) (1,1) (2,0) (3,1)
Rank: 300 - (4,0) (1,0) (2,1) (3,1)
Rank: 301 - (4,1) (1,0) (2,1) (3,1)
Rank: 302 - (4,0) (1,1) (2,1) (3,1)
Rank: 303 - (4,1) (1,1) (2,1) (3,1)
Rank: 304 - (4,0) (1,0) (3,0) (2,0)
Rank: 305 - (4,1) (1,0) (3,0) (2,0)
Rank: 306 - (4,0) (1,1) (3,0) (2,0)
Rank: 307 - (4,1) (1,1) (3,0) (2,0)
Rank: 308 - (4,0) (1,0) (3,1) (2,0)
Rank: 309 - (4,1) (1,0) (3,1) (2,0)
Rank: 310 - (4,0) (1,1) (3,1) (2,0)
Rank: 311 - (4,1) (1,1) (3,1) (2,0)
Rank: 312 - (4,0) (1,0) (3,0) (2,1)
Rank: 313 - (4,1) (1,0) (3,0) (2,1)
Rank: 314 - (4,0) (1,1) (3,0) (2,1)
Rank: 315 - (4,1) (1,1) (3,0) (2,1)
Rank: 316 - (4,0) (1,0) (3,1) (2,1)
Rank: 317 - (4,1) (1,0) (3,1) (2,1)
Rank: 318 - (4,0) (1,1) (3,1) (2,1)
Rank: 319 - (4,1) (1,1) (3,1) (2,1)
Rank: 320 - (4,0) (2,0) (1,0) (3,0)
Rank: 321 - (4,1) (2,0) (1,0) (3,0)
Rank: 322 - (4,0) (2,1) (1,0) (3,0)
Rank: 323 - (4,1) (2,1) (1,0) (3,0)
Rank: 324 - (4,0) (2,0) (1,1) (3,0)
Rank: 325 - (4,1) (2,0) (1,1) (3,0)
Rank: 326 - (4,0) (2,1) (1,1) (3,0)
Rank: 327 - (4,1) (2,1) (1,1) (3,0)
Rank: 328 - (4,0) (2,0) (1,0) (3,1)
Rank: 329 - (4,1) (2,0) (1,0) (3,1)
Rank: 330 - (4,0) (2,1) (1,0) (3,1)
Rank: 331 - (4,1) (2,1) (1,0) (3,1)
Rank: 332 - (4,0) (2,0) (1,1) (3,1)
Rank: 333 - (4,1) (2,0) (1,1) (3,1)
Rank: 334 - (4,0) (2,1) (1,1) (3,1)
Rank: 335 - (4,1) (2,1) (1,1) (3,1)
Rank: 336 - (4,0) (2,0) (3,0) (1,0)
Rank: 337 - (4,1) (2,0) (3,0) (1,0)
Rank: 338 - (4,0) (2,1) (3,0) (1,0)
Rank: 339 - (4,1) (2,1) (3,0) (1,0)
Rank: 340 - (4,0) (2,0) (3,1) (1,0)
Rank: 341 - (4,1) (2,0) (3,1) (1,0)
Rank: 342 - (4,0) (2,1) (3,1) (1,0)
```

```
Rank: 343 - (4,1) (2,1) (3,1) (1,0)
Rank: 344 - (4,0) (2,0) (3,0) (1,1)
Rank: 345 - (4,1) (2,0) (3,0) (1,1)
Rank: 346 - (4,0) (2,1) (3,0) (1,1)
Rank: 347 - (4,1) (2,1) (3,0) (1,1)
Rank: 348 - (4,0) (2,0) (3,1) (1,1)
Rank: 349 - (4,1) (2,0) (3,1) (1,1)
Rank: 350 - (4,0) (2,1) (3,1) (1,1)
Rank: 351 - (4,1) (2,1) (3,1) (1,1)
Rank: 352 - (4,0) (3,0) (1,0) (2,0)
Rank: 353 - (4,1) (3,0) (1,0) (2,0)
Rank: 354 - (4,0) (3,1) (1,0) (2,0)
Rank: 355 - (4,1) (3,1) (1,0) (2,0)
Rank: 356 - (4,0) (3,0) (1,1) (2,0)
Rank: 357 - (4,1) (3,0) (1,1) (2,0)
Rank: 358 - (4,0) (3,1) (1,1) (2,0)
Rank: 359 - (4,1) (3,1) (1,1) (2,0)
Rank: 360 - (4,0) (3,0) (1,0) (2,1)
Rank: 361 - (4,1) (3,0) (1,0) (2,1)
Rank: 362 - (4,0) (3,1) (1,0) (2,1)
Rank: 363 - (4,1) (3,1) (1,0) (2,1)
Rank: 364 - (4,0) (3,0) (1,1) (2,1)
Rank: 365 - (4,1) (3,0) (1,1) (2,1)
Rank: 366 - (4,0) (3,1) (1,1) (2,1)
Rank: 367 - (4,1) (3,1) (1,1) (2,1)
Rank: 368 - (4,0) (3,0) (2,0) (1,0)
Rank: 369 - (4,1) (3,0) (2,0) (1,0)
Rank: 370 - (4,0) (3,1) (2,0) (1,0)
Rank: 371 - (4,1) (3,1) (2,0) (1,0)
Rank: 372 - (4,0) (3,0) (2,1) (1,0)
Rank: 373 - (4,1) (3,0) (2,1) (1,0)
Rank: 374 - (4,0) (3,1) (2,1) (1,0)
Rank: 375 - (4,1) (3,1) (2,1) (1,0)
Rank: 376 - (4,0) (3,0) (2,0) (1,1)
Rank: 377 - (4,1) (3,0) (2,0) (1,1)
Rank: 378 - (4,0) (3,1) (2,0) (1,1)
Rank: 379 - (4,1) (3,1) (2,0) (1,1)
Rank: 380 - (4,0) (3,0) (2,1) (1,1)
Rank: 381 - (4,1) (3,0) (2,1) (1,1)
Rank: 382 - (4,0) (3,1) (2,1) (1,1)
Rank: 383 - (4,1) (3,1) (2,1) (1,1)

Menu:
1. Generate all 2-colored permutations of length n and display their ranks
2. Find the successor of a given 2-colored permutation
3. Find a permutation given its rank
4. Find the rank of a given permutation
5. Exit
Enter your choice:
```

For the situation of 5 and 6. Users can try by their own. Since it's too large for the markdown. So I omit the result here.

Now we get what we want. Other choices are also compiled and usable. However, it's not our main goal. If

you are curiosity, you can check it out. Now we exit the program by choose 5.

```
Menu:
1. Generate and print all subsets/rank
2. Find and print the rank of a given subset
3. Find and print a subset given its rank
4. Find and print the successor of a given subset
5. Exit
Enter your choice: 5
Exiting...
```

Note: I have tested it on mac so it's fine for Linux to run it.

Why the Algorithm Works

Functions and Their Roles

1. `print_permutation` **Function:**
   - **Purpose:** This function is responsible for printing a permutation in a human-readable format, displaying each element and its associated color.
   - **Mechanism:**
     - It iterates over the permutation tuples, formatting each element and its color into a string in the form `(element,color)`.
   - **Effectiveness:** By clearly displaying the permutations and their colors, this function aids in debugging and visualization, making it easier to understand the output of the algorithm.
2. `generate_all_2_colored_permutations` **Function:**
   - **Purpose:** Generates all possible permutations of a set with an added dimension—color. Each element in a permutation can have one of two colors, effectively doubling the conceptual space of permutations.
   - **Mechanism:**
     - First generates all permutations of numbers from 1 to `n`.
     - For each permutation, iterates over all possible binary combinations to apply two colors (0 or 1) to each element in the permutation.
   - **Effectiveness:** This approach leverages the efficiency of bitwise operations to explore the color space, combined with the systematic generation of number permutations to exhaustively generate all possible 2-colored permutations. The method ensures that no permutations are missed and that they are generated in a lexicographical order based on number and color.
3. `find_successor`, `permutation_rank`, `find_permutation_by_rank` **Functions:**
   - **Purpose:** These functions handle advanced navigational and retrieval operations on the permutations:
     - `find_successor` retrieves the next permutation in lexicographical order.
     - `permutation_rank` calculates the rank or index of a given permutation in the complete sorted list of permutations.
     - `find_permutation_by_rank` retrieves a permutation based on its rank.
   - **Mechanism:**
     - Both `find_successor` and `find_permutation_by_rank` manipulate and traverse through the generated permutations, using logical conditions to determine order and rank based on predefined lexicographical rules.
   - **Effectiveness:** These functions are critical for applications requiring ordered data access, such as searching algorithms and optimizations that rely on the position of data within a sorted array.

More Information

1. **Comprehensive Generation of Permutations:**
   - By using permutations from the itertools module and combining them with binary operations for color assignments, the algorithm effectively covers every possible variation of the base set, ensuring

a thorough exploration of the permutation space.

2. **Efficiency in Sorting and Access:**
   - The natural lexicographical order used in generating permutations simplifies sorting and accessing specific permutations or calculating their rank. This inherent order is maintained throughout the generation process, allowing subsequent functions to leverage this ordering for efficient computation and retrieval.
3. **Optimized Memory and Computational Practices:**
   - Python's generators (`yield`) are used in permutation generation to handle large sets of data efficiently without requiring all permutations to be stored in memory simultaneously. This approach minimizes memory usage and enhances performance, especially crucial when dealing with large `n`.
4. **Use of Python Standard Libraries:**
   - Utilizing built-in Python libraries like `itertools` for permutation generation and `chain` from `itertools` to flatten list structures provides reliable and efficient mechanisms for complex combinatorial tasks. These libraries are optimized for performance and are widely used in the industry for such purposes.