Solution: Subsets

Let's solve the Subsets problem using the Subsets pattern.

We'll cover the following

Statement

Given an array of integers, `nums`, find all possible subsets of `nums`, including the empty set.

> **Note:** The solution set must not contain duplicate subsets. You can return the solution in any order.

Constraints:

- $1 \leq$ `nums.length` ≤ 10
- $-10 \leq$ `nums[i]` ≤ 10
- All the numbers of `nums` are unique.

Pattern: Subsets

Problems such as this one, where we need to find all possible subsets of a given set, can be efficiently solved using the subsets pattern. This pattern involves generating all possible subsets of a given set by using binary representations of indices to represent which elements should be included in each subset. This approach allows us to solve a wide range of problems that involve generating all possible subsets of a set.

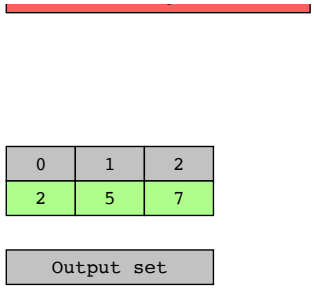
Solution

The idea is to generate binary numbers from 0 to $2^{nums.length}$. The number of bits in the binary numbers will equal `nums.length`. These integers will be added to the subset whose corresponding bits are set to 1 in the binary number.

> **Note:** The ordering of bits for picking integers from the set doesn't matter. We can pick integers from left to right and produce the same output as picking integers from right to left.

The size of the given set is 3, so 2^3 = 8 subsets will be formed. The value of "i" will iterate from 0 to 7. The output set is currently empty.

```
Size of integer set = 3
```



1 of 9

> **Note:** In the following section, we will gradually build the solution. Alternatively, you can skip straight to just the code.

Step-by-step solution construction

The first step in solving this problem is calculating the total subsets generated from the given numbers. The total number of possible subsets for a set of size n is 2^n. We'll use this formula to calculate the total subsets for our input numbers.

Java

```java
import java.util.*;
class FindSubsets {

    static List<List<Integer>> findAllSubsets(int[] nums) {
        List<List<Integer>> setsList = new ArrayList<>();
        if (nums.length != 0) {

            // finds number of subsets by taking power of length of input array
            int subsetsCount = (int) (Math.pow(2, nums.length));

            System.out.println("   Total subsets: "+ subsetsCount);
        } else {
            System.out.println("   Total subsets: 1");
            List<Integer> subset = new ArrayList<>();
            setsList.add(subset);
        }
        return setsList;
    }

    public static void main(String[] args) {
        int[][] inputSets = {
                {},
                {2, 5, 7},
                {1, 2},
                {1, 2, 3, 4},
                {7, 3, 1, 5}
        };
```

Subsets

Once we have the count of all possible subsets, we can generate each subset using a binary representation of the indices. We will initiate a loop from `i = 0` to `i = subsetsCount - 1`, where `subsetsCount` represents the total subsets. In each loop iteration, we will form a subset using the value of `i` as a bitmask.

Using `i` as a bitmask enables us to represent which elements from the original set should be included in the current subset. The set bits in `i` correspond to the indices of the elements to choose from the original set. For example, consider a set `[1, 2, 3]` and `i = 2`. In binary representation, 2 is denoted as `010`. Since the second bit is set to 1, our subset will include the second element from the original set, which is 2. Therefore, the subset will be `[2]`. Similarly, for `i = 3`, the binary representation is `011`. In this case, the set bits in `i` correspond to the first and second elements from the original set, resulting in the subset `[1, 2]`. Within each iteration of the loop (from 0 to `subsetsCount - 1`), we will perform the following steps:

- We will initialize an empty set called `subset` to store the subset being constructed.
- During each iteration, we will iterate over the set `nums` elements using the variable `j`, representing the indices of the elements in `nums`.
- For each `j`, we will call the `getBit(i, j)` function to determine if the j^{th} bit of `i` is set to 1.
- If `getBit(i, j)` returns 1, indicating that the j^{th} bit is set to 1, and `nums[j]` is not already present in `subset`, we will add `nums[j]` to `subset`.
- Finally, we will add the set `subset` to `setsList`.

Once the loop completes, the list `setsList` will contain all the subsets, including the empty set. This approach allows us to generate all possible subsets of the given set by considering each possible combination of elements using the binary representation of the indices.

Java

```java
import java.util.*;
class FindSubsets {
    static int getBit(int num, int bit) {
        // shifts the first operand the specified number of bits to the left
        int temp = (1 << bit);

        // if binary number and current subset count are equal,
        // return 1 else return 0
        temp = temp & num;
        if (temp == 0) {
            return 0;
        }
        return 1;
    }

    static List<List<Integer>> findAllSubsets(int[] nums) {
        List<List<Integer>> setsList = new ArrayList<>();
        if (nums.length != 0) {
            // finds number of subsets by taking power of length of input array
            int subsetsCount = (int) (Math.pow(2, nums.length));

            for (int i = 0; i < subsetsCount; ++i) {
                // Set is created to store each subset
                List<Integer> subset = new ArrayList<>();
                for (int j = 0; j < nums.length; ++j) {
                    // if a specific bit is 1, chooses that number from the original set
                    // and add it to the current subset
                    if (getBit(i, j) == 1) {
```

Subsets

Just the code

Here's the complete solution to this problem:

Java

```java
import java.util.*;
```

```
class FindSubsets {
    static int getBit(int num, int bit) {
        int temp = (1 << bit);
        temp = temp & num;
        if (temp == 0) {
            return 0;
        }
        return 1;
    }

    static List<List<Integer>> findAllSubsets(int[] nums) {
        List<List<Integer>> setsList = new ArrayList<>();
        if (nums.length != 0) {
            int subsetsCount = (int) (Math.pow(2, nums.length));
            for (int i = 0; i < subsetsCount; ++i) {
                List<Integer> subset = new ArrayList<>();
                for (int j = 0; j < nums.length; ++j) {
                    if (getBit(i, j) == 1) {
                        int x = nums[j];
                        subset.add(x);
                    }

                }
                setsList.add(subset);
            }
        } else {
```

Subsets

To recap, the solution to this problem can be divided into the following parts:

1. Calculate the subsets count using the formula 2^n.
2. Iterate from 0 to the subsets count.
3. Generate subsets using bitwise operations.
4. Add each generated subset to the list of subsets.
5. Return the list of subsets.

Time complexity

The time complexity of this solution is exponential, specifically $O(2^n \cdot n)$, where n represents the number of integers in the given array.

Space complexity

The space complexity of this solution is $O(n)$, where n represents the number of integers in the given array. This analysis excludes the space utilized by the output array, `setsList`, and only considers the space used by the `subset` set.

Back
Subsets

Next
Permutations

Mark as Completed