3539. Find Sum of Array Product of Magical Sequences

Solved

Hard | Topics | Companies | Hint

You are given two integers, `m` and `k`, and an integer array `nums`.

A sequence of integers `seq` is called **magical** if:

- `seq` has a size of `m`.
- `0 <= seq[i] < nums.length`
- The **binary representation** of $2^{seq[0]} + 2^{seq[1]} + ... + 2^{seq[m - 1]}$ has `k` **set bits**.

The **array product** of this sequence is defined as `prod(seq) = (nums[seq[0]] * nums[seq[1]] * ... * nums[seq[m - 1]])`.

Return the **sum** of the **array products** for all valid **magical** sequences.

Since the answer may be large, return it **modulo** $10^9 + 7$.

A **set bit** refers to a bit in the binary representation of a number that has a value of 1.

Example 1:

> **Input:** m = 5, k = 5, nums = [1,10,100,10000,1000000]
>
> **Output:** 991600007
>
> **Explanation:**
>
> All permutations of `[0, 1, 2, 3, 4]` are magical sequences, each with an array product of 10^{13}.

Example 2:

> **Input:** m = 2, k = 2, nums = [5,4,3,2,1]
>
> **Output:** 170
>
> **Explanation:**
>
> The magical sequences are `[0, 1]`, `[0, 2]`, `[0, 3]`, `[0, 4]`, `[1, 0]`, `[1, 2]`, `[1, 3]`, `[1, 4]`, `[2, 0]`, `[2, 1]`, `[2, 3]`, `[2, 4]`, `[3, 0]`, `[3, 1]`, `[3, 2]`, `[3, 4]`, `[4, 0]`, `[4, 1]`, `[4, 2]`, and `[4, 3]`.

Example 3:

> **Input:** `m = 1, k = 1, nums = [28]`
>
> **Output:** 28
>
> **Explanation:**
>
> The only magical sequence is `[0]`.

Constraints:

- `1 <= k <= m <= 30`
- `1 <= nums.length <= 50`
- `1 <= nums[i] <= 10`^{8}

Python:

```
MOD = 10**9 + 7
from functools import lru_cache
import math
from typing import List

class Solution:
    def magicalSum(self, total_count: int, target_odd: int, numbers: List[int]) -> int:

        @lru_cache(None)
        def dfs(remaining, odd_needed, index, carry):
            if remaining < 0 or odd_needed < 0 or remaining + carry.bit_count() < odd_needed:
                return 0
            if remaining == 0:
                return 1 if odd_needed == carry.bit_count() else 0
            if index >= len(numbers):
                return 0

            ans = 0
            for take in range(remaining + 1):
```

```
            ways = math.comb(remaining, take) * pow(numbers[index], take, MOD) % MOD
            new_carry = carry + take
            ans += ways * dfs(remaining - take, odd_needed - (new_carry % 2), index + 1,
new_carry // 2)
            ans %= MOD
        return ans

    return dfs(total_count, target_odd, 0, 0)
```

JavaScript:

```
const MOD = 1000000007n;

function magicalSum(m, k, nums) {
  const n = nums.length;
  const numsB = nums.map(BigInt);

  // Precompute powtab[i][c] = nums[i]^c mod MOD for c in [0..m]
  const powtab = Array.from({ length: n }, () => Array(m + 1).fill(0n));
  for (let i = 0; i < n; i++) {
    powtab[i][0] = 1n;
    for (let c = 1; c <= m; c++) {
      powtab[i][c] = (powtab[i][c - 1] * numsB[i]) % MOD;
    }
  }

  // Precompute combinations comb[r][c] = C(r, c) mod MOD for r,c in [0..m]
  const comb = Array.from({ length: m + 1 }, () => Array(m + 1).fill(0n));
  for (let i = 0; i <= m; i++) {
    comb[i][0] = 1n;
    for (let j = 1; j <= i; j++) {
      comb[i][j] = (comb[i - 1][j - 1] + comb[i - 1][j]) % MOD;
    }
  }

  // dp[rem][carry][ones] holds the running total after processing some prefix of indices:
  // rem picks left to place, current carry value, ones bits produced so far
  let dp = Array.from({ length: m + 1 }, () =>
    Array.from({ length: m + 1 }, () => Array(k + 1).fill(0n))
  );
  dp[m][0][0] = 1n; // start with all m picks remaining, carry = 0, ones = 0

  for (let i = 0; i < n; i++) {
    const next = Array.from({ length: m + 1 }, () =>
      Array.from({ length: m + 1 }, () => Array(k + 1).fill(0n))
```

```
  );
  const powi = powtab[i];
  for (let rem = 0; rem <= m; rem++) {
    for (let carry = 0; carry <= m; carry++) {
      for (let ones = 0; ones <= k; ones++) {
        const base = dp[rem][carry][ones];
        if (base === 0n) continue;
        // Choose c copies of index i among the rem remaining positions
        for (let c = 0; c <= rem; c++) {
          const t = c + carry;           // add c to current bit with carry in
          const bit = t & 1;             // output bit at this position
          const ones2 = ones + bit;      // update ones count
          if (ones2 > k) continue;
          const carry2 = t >> 1;         // carry to the next bit
          const rem2 = rem - c;

          // Transition weight = comb[rem][c] (ways to place c copies) * nums[i]^c
          let add = base;
          add = (add * comb[rem][c]) % MOD;
          add = (add * powi[c]) % MOD;

          next[rem2][carry2][ones2] = (next[rem2][carry2][ones2] + add) % MOD;
        }
      }
    }
  }
  dp = next;
}

// Finish: only states with rem = 0 are valid
// Leftover carry still contributes popcount(carry) ones
let ans = 0n;
for (let carry = 0; carry <= m; carry++) {
  const extra = popcount(carry);
  const need = k - extra;
  if (need >= 0 && need <= k) {
    ans = (ans + dp[0][carry][need]) % MOD;
  }
}
return Number(ans);

function popcount(x) {
  let cnt = 0;
  while (x) {
```

```
      x &= x - 1;
      cnt++;
    }
    return cnt;
  }
}
```

Java:

```
// https://www.youtube.com/@0x3f
class Solution {
    private static final int MOD = 1_000_000_007;
    private static final int MX = 31;

    private static final long[] F = new long[MX]; // F[i] = i!
    private static final long[] INV_F = new long[MX]; // INV_F[i] = i!^-1

    static {
        F[0] = 1;
        for (int i = 1; i < MX; i++) {
            F[i] = F[i - 1] * i % MOD;
        }

        INV_F[MX - 1] = pow(F[MX - 1], MOD - 2);
        for (int i = MX - 1; i > 0; i--) {
            INV_F[i - 1] = INV_F[i] * i % MOD;
        }
    }

    private static long pow(long x, int n) {
        long res = 1;
        for (; n > 0; n /= 2) {
            if (n % 2 > 0) {
                res = res * x % MOD;
            }
            x = x * x % MOD;
        }
        return res;
    }

    public int magicalSum(int m, int k, int[] nums) {
        int n = nums.length;
        int[][] powV = new int[n][m + 1];
        for (int i = 0; i < n; i++) {
            powV[i][0] = 1;
```

```java
            for (int j = 1; j <= m; j++) {
                powV[i][j] = (int) ((long) powV[i][j - 1] * nums[i] % MOD);
            }
        }

        int[][][][] memo = new int[n][m + 1][m / 2 + 1][k + 1];
        for (int[][][] a : memo) {
            for (int[][] b : a) {
                for (int[] c : b) {
                    Arrays.fill(c, -1);
                }
            }
        }
        return (int) (dfs(0, m, 0, k, powV, memo) * F[m] % MOD);
    }

    private long dfs(int i, int leftM, int x, int leftK, int[][] powV, int[][][][] memo) {
        int c1 = Integer.bitCount(x);
        if (c1 + leftM < leftK) { // 可行性剪枝
            return 0;
        }
        if (i == powV.length) {
            return leftM == 0 && c1 == leftK ? 1 : 0;
        }
        if (memo[i][leftM][x][leftK] != -1) {
            return memo[i][leftM][x][leftK];
        }
        long res = 0;
        for (int j = 0; j <= leftM; j++) { // 枚举 I 中有 j 个下标 i
            // 这 j 个下标 i 对 S 的贡献是 j * pow(2, i)
            // 由于 x = S >> i，转化成对 x 的贡献是 j
            int bit = (x + j) & 1; // 取最低位，提前从 leftK 中减去，其余进位到 x 中
            if (bit <= leftK) {
                long r = dfs(i + 1, leftM - j, (x + j) >> 1, leftK - bit, powV, memo);
                res = (res + r * powV[i][j] % MOD * INV_F[j]) % MOD;
            }
        }
        return memo[i][leftM][x][leftK] = (int) res;
    }
}
```