Find Content Children - Assign Cookies Problem Documentation

Table of Contents

1. Problem Statement

We have a group of children, each with a greed factor g[i], which represents the minimum size of a cookie they require to be content. We also have a list of cookies, each with a size s[j].

The goal is to maximize the number of content children by giving each child at most one cookie. A child is content if they receive a cookie of size greater than or equal to their greed factor.

Constraints:

- $1 \leq g.length \leq 30{,}000$
- $0 \leq s.length \leq 30{,}0000$
- $1 \leq g[i], s[j] \leq 2^{31} - 1$

2. Intuition

The problem can be solved optimally using a greedy approach:

- Sort both lists (g and s) in ascending order.
- Try to assign the smallest cookie that can satisfy a child.
- If a cookie is too small for the current child, move to a larger cookie.
- Continue until all cookies are assigned or all children are satisfied.

This ensures that we use smaller cookies efficiently and maximize the number of content children.

3. Key Observations

- A child with a smaller greed factor should be assigned the smallest possible cookie.
- Sorting both arrays helps in efficiently matching children with cookies.
- We can use two pointers to traverse both lists and determine the maximum number of content children.
- If a cookie is too small for a child, move to the next larger cookie.
- Once all cookies are used or all children are satisfied, stop the process.

4. Approach

i. Sort both lists:
   a. g (children's greed factors) in ascending order.
   b. s (cookie sizes) in ascending order.

ii. Use two pointers:
   a. i for g (children).
   b. j for s (cookies).

iii. Match greed factor with cookie size:
   a. If s[j] >= g[i], assign the cookie to the child (content_children += 1) and move both pointers.
   b. Otherwise, move only the cookie pointer (j), searching for a larger cookie.

iv. Continue until:
   a. All children are satisfied, **or** All cookies are used.

5. Edge Cases

No cookies available (s = []) → Output 0

All children have a higher greed factor than any cookie size → Output 0

More cookies than children → Maximum children get satisfied

Children with the same greed factor → Handled via sorting

All children can be satisfied with given cookies → Output len(g)

Very large input sizes (g.length = 30,000, s.length = 30,000) → Efficiently handled in O(n log n) time

6. Complexity Analysis

Time Complexity

- Sorting g and s → O(nlogn)
- Greedy traversal using two pointers → O(n)
- Overall Complexity: O(nlogn)

Space Complexity

- Sorting is in-place, using O(1) extra space.
- No extra data structures used, so space complexity is O(1)

7. Alternative Approaches

Brute Force (Inefficient)

- Try every cookie for every child ($O(n^2)$)
- Nested loops to check every combination
- Not feasible for large inputs (TLE - Time Limit Exceeded).

Binary Search Approach

- Sort g and s
- Use Binary Search (O(log n)) to find the smallest valid cookie for each child
- Overall Complexity: O(nlogn), same as greedy but more complex implementation.

8. Test Cases

Test Case 1: Basic Example

```
g = [1, 2, 3]
s = [1, 1]
sol = Solution()
print(sol.findContentChildren(g, s))  # Output: 1
```

Explanation: The only assignable cookie (size 1) satisfies one child (greed 1).

Test Case 2: More Cookies Than Children

```
g = [1, 2]
s = [1, 2, 3]
print(sol.findContentChildren(g, s))  # Output: 2
```

Explanation: Both children get satisfied with cookies of size 1 and 2.

Test Case 3: No Cookies

```
g = [1, 2, 3]
s = []
print(sol.findContentChildren(g, s))  # Output: 0
```

Explanation: No cookies available.

Test Case 4: No Child Can Be Satisfied

```
g = [5, 10, 15]
s = [1, 2, 3]
print(sol.findContentChildren(g, s))  # Output: 0
```

Explanation: No cookie is large enough for any child.

Test Case 5: Large Input

```
g = [i for i in range(1, 30001)]
s = [i for i in range(1, 30001)]
print(sol.findContentChildren(g, s))  # Output: 30000
```

Explanation: Each child gets a cookie exactly matching their greed factor.

9. Final Thoughts

- This problem is a classic greedy algorithm example.
- Sorting both lists helps maximize the number of satisfied children efficiently.
- The two-pointer approach ensures optimal performance.
- Alternative approaches, like binary search, exist but add unnecessary complexity.
- Time Complexity: O(nlogn) due to sorting, which is optimal.
- This solution performs well even for large constraints. 🚀