Sum Pro - An efficient linear approach to detect set equality

Bhargav R
PES Institute of Technology,
Bangalore South Campus,
Karnataka, India
bhargavr105@gmail.com

Abstract:
We present a new algorithm to test the equality of tests, which is highly efficient in terms of space and time complexity. Most existing methodologies make use of specialized data structures such as 'tries' to perform operations on sets and provide probabilistic results. In this paper, we introduce an algorithm that can perform with improved efficiency irrespective of the underlying data structure used. When compared with algorithms in existing literature, this method can determine equality with reduced memory requirements \( O(1) \) and in a single pass \( O(n) \). We show both, mathematically and through experiments that the proposed method fares better than its predecessors.

Keywords:
Analysis of algorithms, Data structure independent, Equality, Linear approach, One-pass, Set theory.

I. INTRODUCTION
Set equality refers to the process of determining if two sets contain the same elements irrespective of their ordering. Though this might seem like a trivial problem, it is still noteworthy due to the fact that set equality is critically used in cryptographic ciphers, plagiarism detection and other mathematical set operations.

Existing methodologies such as the subset-check test and the sorted-set test for equality; though obtain the necessary result without errors, are not efficient in either space or time. Constant-time equality tests can be highly efficient with their running time independent of the input size. However, they fail grossly in the sense that the validity of the results are only probabilistic. Such tests can, with a high probability determine if two sets are equal with a known chance of false positives. We will show that the presented algorithm can perform efficiently with respect to both time and space complexity in all scenarios with zero-chances of false positives.

II. ORGANIZATION OF PAPER
We first look at the definitions and notations involved to give us the necessary background information. This will help us understand the existing and proposed methodologies with much more ease.

We then move on to looking into the existing methodologies and where they fall through the cracks. This will give us an idea about the cons of the algorithms in present literature. This is followed by a discussion on the proposed algorithm, the proof and the areas where it exceeds. We further look at a comparison of the existing algorithms and the proposed algorithm in terms of space and time efficiency.

III. DEFINITIONS AND NOTATIONS
Algorithm: An algorithm is a step-by-step procedure for calculation, data processing, or automated reasoning.

Equal sets: Two sets are equal if they both have the same members.

Linear algorithms: An algorithm that takes one iteration of the input data to solve the problem and output the results.

Data structure: In computer science, a data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently.

Tries: In computer science, a trie, also called digital tree or prefix tree or sometimes radix tree, is an ordered tree data structure that is used to store a dynamic set or associative array where the keys are usually strings.

Time efficiency: It is a concept used in computer science to estimate of the running time as a function as the size of the input data. The result is normally expressed using Big O notation

Space efficiency: It is defined by using the Big O notation taking four aspects of memory into consideration: The amount of memory needed to hold the code for the algorithm; the amount of memory needed for the input data; the amount of memory needed for any output data (some algorithms, such as sorting, often just rearrange the input data and don't need any space for output data) and the amount of memory needed as working space during the calculation.
IV. STATE OF THE ART

Subset check algorithm:

To find out if sets S1 and S2 are equal, this algorithm uses the following condition.

```plaintext
if (S1 is a subset of S2 and S2 is a subset of S1) {
    return true
} else {
    return false
}
```

Though this is a very simple algorithm, finding if a set is a subset of another is time inefficient. If the size of S1 is n1 and the size of S2 is n2, the time taken is O(n1 * n2).

Sorted set equality algorithm:

This algorithm relies on storing a copy of the existing sets in sorted order.

```plaintext
Sorted_S1 = sort (S1)
Sorted_S2 = sort (S2)
N = Max of (Size of set S1, Size of set S2)
for (index from 0 to N) {
    if Sorted_S1 [index] != Sorted_S2 [index] {
        return false
    }
}
return true
```

This algorithm requires addition storage space of O(n) not to mention, a running time of at-least O(n log n) for sorting the sets. This renders the algorithm better in time efficiency than the previous, but worse in space efficiency.

William Pugh’s algorithm:

This paper makes use of the trie data structure to determine equality of sets. This method involves creating a binary hash trie to store the elements of the given sets along with a hash key, which determines their equality. The space complexity of this algorithm is O(n) with only probabilistic results.

Other algorithms:

There are other algorithms proposed by Daniel M Yellin, Tarjan [2] ewhich provide set operability with an efficiency O(log n) and one by and Tak Wah Lam, and Ka Hing Lee [3] which provides an efficiency of O(log n). Though these are major improvements over previous algorithms in terms of running time, they fail to provide an optimized use of space.

V. PROPOSED ALGORITHM

We introduce an algorithm that can perform the equality test on two given sets S1 and S2 with a running time of efficiency O(n) and with constant space efficiency i.e.; O(1).

A pseudo-code is given below.

```plaintext
# Input to function : Set S1, Set S2
# Output of function : true or false
S1 = read set 1 from input
S2 = read set 2 from input
if Size(S1) != Size(S2) {
    return false
} else {
    # Initialize the following variables
    Sum_S1 = 0
    Sum_S2 = 0
    Product_S1 = 1
    Product_S2 = 1
    ZerosInS1 = 0
    ZerosInS2 = 0
    foreach element in S1
        if (element is zero) {
            ZerosInS1 += 1
        } else {
            Sum_S1 += element
            Product_S1 *= element
        }
    foreach element in S2
        if (element is zero)
            ZerosInS2 += 1
        else
            Sum_S2 += element
            Product_S2 *= element
    if (Sum_S1 = Sum_S2 and Product_S1 = Product_S2 and ZerosInS1 = ZerosInS2 ) {
        return true
    } else {
        return false
    }
}
```

# End of algorithm

We prove the above algorithm by mathematical induction. This involves proving it for number of elements 0, 1 and 2. Proving the same for a size K, assuming it is already proven for size K-1, follows this.

VI. PROOF OF CORRECTNESS

We denote the size of the sets by N.

Step 1: For N = 0

If both sets are empty, they are equal sets by definition.

Step 2: For N = 1

Let us assume:
Set S1 has the elements \{a\}
Set S2 has the elements \{b\}
By definition:
The two sets will be equal, if and only if \( a = b \). We pass these sets as inputs to the algorithm and discuss the results below, for various possibilities of the values of the set elements.

**Case 1**: \( a = 0 \) and \( b \neq 0 \)

The values obtained are as follows –

\[
\begin{align*}
\text{Sum}_{S1} & = 0 \\
\text{Product}_{S1} & = 1 \\
\text{ZerosInS1} & = 1 \\
\text{Sum}_{S2} & = b \\
\text{Product}_{S2} & = b \\
\text{ZerosInS1} & = 0
\end{align*}
\]

We find that the two sets are not equal as they fail the condition –

\[
\begin{align*}
\text{Sum}_{S1} & = \text{Sum}_{S2} \\
\text{Product}_{S1} & = \text{Product}_{S2} \\
\text{ZerosInS1} & = \text{ZerosInS2}
\end{align*}
\]

**Case 2**: \( a \neq 0 \), \( b = 0 \)

The values obtained are as follows –

\[
\begin{align*}
\text{Sum}_{S1} & = a \\
\text{Product}_{S1} & = a \\
\text{ZerosInS1} & = 0 \\
\text{Sum}_{S2} & = b \\
\text{Product}_{S2} & = b \\
\text{ZerosInS1} & = 0
\end{align*}
\]

We find that the two sets are equal as they satisfy the condition –

\[
\begin{align*}
\text{Sum}_{S1} & = \text{Sum}_{S2} \\
\text{Product}_{S1} & = \text{Product}_{S2} \\
\text{ZerosInS1} & = \text{ZerosInS2}
\end{align*}
\]

**Step 3**: For \( N = 2 \)

Let us assume,

Set \( S1 \) has the elements \( \{a_1, a_2\} \)

Set \( S2 \) has the elements \( \{b_1, b_2\} \)

By definition, the two sets are equal if and only if they satisfy either of the following two conditions.

- \( a_1 = b_1 \) and \( a_2 = b_2 \)
- \( a_1 = b_2 \) and \( a_2 = b_1 \)

We prove this case by contradiction.

If the condition given below is satisfied -

\[
\begin{align*}
\text{Sum}_{S1} & = \text{Sum}_{S2} \\
\text{Product}_{S1} & = \text{Product}_{S2} \\
\text{ZerosInS1} & = \text{ZerosInS2}
\end{align*}
\]

It implies:

\[
\begin{align*}
a_1 + a_2 & = b_1 + b_2 \\
(1) \\
a_1 * a_2 & = b_1 * b_2 \\
(2) \\
\text{number of zeros in S1} & = \text{number of zeros in S2}
\end{align*}
\]

Rewriting (2), we obtain -

\[
a_1 = b_1 + b_2 - a_2 \quad (3)
\]

Replacing \( a_1 \) in (2) and rearranging the terms,

\[
\begin{align*}
(b_1 + b_2 - a_2) * a_2 & = b_1 * b_2 \\
(a_1 * a_2) + (b_2 * a_2) & = b_1 * b_2 \\
a_2 * (b_1 - a_2) & = b_2 (b_1 - a_2) \\
(4)
\end{align*}
\]

If \( b_1 - a_2 = \) zero, then \( b_1 = a_2 \).

By (1), \( b_2 = a_1 \).

This proves the two sets are equal.

If \( b_1 - a_2 \neq 0 \), we have after dividing (4) by \( b_1 - a_2 \):

\[
a_2 = b_2. \quad (6)
\]

Replacing \( a_2 = b_2 \) in (1), we get \( a_1 = b_1 \).

This proves the two sets are equal.

**Step 4**: For \( N = k \)

Assuming the conditions are satisfied for \( N = k \), we have:

\[
\begin{align*}
\text{Sum}_{S1} & = a.1 + a.2 + a.3 + \ldots + a.k \\
\text{Product}_{S1} & = a.1 * a.2 * a.3 * \ldots * a.k \\
\text{ZerosInS1} & = \text{Number of zeros in S1} \\
\text{Sum}_{S2} & = b.1 + b.2 + b.3 + \ldots + b.k \\
\text{Product}_{S2} & = b.1 * b.2 * b.3 * \ldots * b.k \\
\text{ZerosInS1} & = \text{Number of zeros in S2}
\end{align*}
\]

We also have:

\[
\begin{align*}
\text{Sum}_{S1} & = \text{Sum}_{S2} \quad (7) \\
\text{Product}_{S1} & = \text{Product}_{S2} \quad (8) \\
\text{ZerosInS1} & = \text{ZerosInS2} \quad (9)
\end{align*}
\]

We need to prove the algorithm still is valid for \( N = k + 1 \)

We shall assume a new element is added to each set – \( a.k+1 \) to \( S1 \) and \( b.k+1 \) to \( S2 \).

Equations (7), (8) and (9) will still be satisfied if \( a.k+1 = b.k+1 \) thus proving \( S1 = S2 \)

If \( a.k+1 \neq b.k+1 \), (1) will not be satisfied, which denotes \( S1 \neq S2 \); also proving the validity of the algorithm.

**VII. COMPARISON OF THE VARIOUS ALGORITHMS**

We provide here, a comparison of the existing methodologies and the proposed algorithm in terms of space and time efficiency. We also provide comments for certain algorithms, which give us, further insight into their performance and behavior in edge cases.

Efficiency is measured using the Big O notation, as it is a standard for analyzing algorithmic space and time complexity.
TABLE I. COMPARISON OF ALGORITHMS

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Space Complexity</th>
<th>Time Complexity</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subset check algorithm</td>
<td>$O(1)$</td>
<td>$O(n^2)$</td>
<td>-</td>
</tr>
<tr>
<td>Sorted set equality algorithm</td>
<td>$O(n^2)$</td>
<td>$O(n)$</td>
<td>-</td>
</tr>
<tr>
<td>William Pugh's algorithm</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>Probabilistic results</td>
</tr>
<tr>
<td>Daniel M Yellin's algorithm</td>
<td>$O(n)$</td>
<td>$O(\log_2 n)$</td>
<td>A time efficiency of $O(n)$ is required for</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>construction of the data structure</td>
</tr>
<tr>
<td>Tak Wah Lam and Ka Hing Lee's algorithm</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>A time efficiency of $O(n)$ is required for</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>construction of the data structure</td>
</tr>
<tr>
<td>The proposed algorithm</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>A time complexity of $O(1)$ can be achieved for</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>sets that are structured using a Hash map.</td>
</tr>
</tbody>
</table>

VIII. CONCLUSION

In this paper, we introduced an improved algorithm could determine the equality of sets with a space efficiency of $O(1)$ and a time efficiency of $O(n)$. We also showed that the proposed methodology does not have the disadvantages of a probabilistic algorithm. This gives us the opportunity of ruling of false positives. By using simple counter variables, the proposed algorithm also fares better than the algorithms that require tries or binary hashes, which require a running time of $O(n)$ for merely constructing the structure, though the set operations can be completed in $O(\log n)$. We regard this algorithm as the first step to developing a structure with even better space and time efficiency by making use of a Hash map for insertions and deletions.

IX. ACKNOWLEDGEMENTS

I would like to thank Dr. Gowri Srinivasa, PES Institute of Technology - Bangalore South Campus, who took the time to provide valuable guidance. Without her support, it would not have been possible to complete this research study.

X. REFERENCES


