Inequivalent Boolean Functions

A Boolean function of n variables can be represented by its truth table, which gives its value for all 2^n combinations of the input variables. This can be represented by a string of length 2^n containing only characters `0` and `1`. If the n variables are $x_0, x_1, \ldots, x_{n-1}$, then the value of the function for an input $x_i = b_i, b_i \in \{0, 1\}$, is given by the character in the string at position $\sum_{i=0}^{n-1} b_i * 2^i$. Thus the OR function of 2 variables is represented by the string `0111` and the AND by `0001`.

Two such functions $f(x_0, \ldots, x_{n-1})$ and $g(x_0, \ldots, x_{n-1})$ are said to be equivalent if there exists a permutation p of $\{0, 1, \ldots, n-1\}$ such that for all sequences $b_0, \ldots, b_{n-1}$ of n bits, $f(b_0, b_1, \ldots, b_{n-1}) = g(b_{p[0]}, b_{p[1]}, \ldots, b_{p[n-1]})$. In other words, the expression for g is obtained from that of f by permuting the input variables.

Given a sequence of boolean functions represented as strings, you have to compute the number of distinct (inequivalent) functions in the sequence.

There is no simple algorithm to check equivalence and essentially the only way is try all possible permutations. However, the number of such checks can be reduced by using hash functions. Suppose there is a function h that maps strings to integers in the range $\{0, \ldots, B-1\}$ with the property that equivalent functions map to the same value. Then we only need to check equivalence between functions whose hash value is the same. A simple hash function is the number of 1's in the string.

In the first part of the problem, you can use this simple function, and the brute force algorithm for checking equivalence. The inputs will be small in size for this part, and correctness should be ensured.

In the 2nd part, you have to try to think of possible other hash functions, that preserve equivalence, but may reduce checks further. There is a trade-off between the time required to compute the hash function, and the number of checks it saves. Experiment with this and also better ways of checking equivalence. If all functions are in fact equivalent, then no hash function can separate them. However, in such cases, there is good chance the brute force algorithm will find the permutation showing equivalence quickly. But if the functions are not equivalent, essentially all permutations have to be tried before concluding this, and avoiding such checks using hash functions can reduce time.

Input/Output The first line will specify n the number of variables and m the number of strings. The next m lines will contain a string of `0,1` of length 2^n, one per line. The output should be the number of inequivalent functions among the m given. For the first part, $n \leq 8$ and $m \leq 1000$. For the 2nd part, you should try to do as best as possible, with say $2^n * m \leq 10^7$, and n at most 20. Experiment with random strings containing the same number of `1`'s. There is no time limit as such, will compare with whatever I can do with simple hash functions.

For the brute force equivalence, you can just try generating all permutations and checking for each. The `next_permutation` function defined in the algorithm header can be used to generate permutations. For the second part, a better way of doing this, by eliminating some possibilities, may be required.

```
Sample Input  Sample Output
2 5           3
0000
0100
0101
0011
0010
```

Submission: Submit a single file named RollNo_10.cpp