# Quin\_McCluskeyAlogrithm

Implementing Quin McCluskey algorithm for minimizing logic equations, observation and experimenting with variety of sets of equations to generate conclusion on variance with the algorithm.

INTRODUCTION:

Quine-McCluskey method is used to simplify the Boolean expressions to reduce the efforts by avoiding the calculation of the redundant terms. The other minimization methods like K map or using techniques of Boolean algebra may face limitations when given for large number of input. Maintaining K map for an equation sufficiently large number if input variables and QM approach present a simpler and efficient tool when it comes to minimization of even of input with high number of minterms.

PROBLEM STATEMENT:

Develop, implement and experiment Quine-McCluskey's algorithm for minimization of logic equations.

The characteristics of the algorithm are the following:

•The description of the logic equations is given in a text input file. You design the format of the input file or you can use another standard description.

•The algorithm should be tested for various logic equations of different sizes and number of variables.

•Consider sufficiently many test cases to study the performance of the algorithm in terms of execution time and the effectiveness of logic minimization.

PROPOSED SOLUTION:

# Step1: Arrange all the minterms according to the number of 1’s contained in binary representation in each of the minterm in each group.

For Example: All the minterms with zero 1’s in their representation will be grouped together in group 1.

# Step 2: In this step we must obtain the pairs across the adjacent groups. A matched pair will refer to the terms which differ from each other only by one variable. The terms which are not covered must be segregated separately.

For Example: Group zero will be formed by comparing the terms in earlier group zero and group one.

# Step 3: Similar groups will be formed using the function Comp\_pairs and the 2d array of new\_group will keep growing and the terms will be updated each time. The number of times such tables will be formed will depend on the number of variables in the function.

# Step 4: Printing the prime implicants.

The next step is to print all the prime implicants that we received in the last step. While printing the prime implicants, the numbers will have to be expressed again in their letter format. This has been achieved using the binary\_to\ \_letter . In this function if the value at a location is 1 then that variable is sent out as the letter at that position. If it is zero, then the bar function of that letter is sent out.

# Step 5: Preparing Prime Implicant Chart

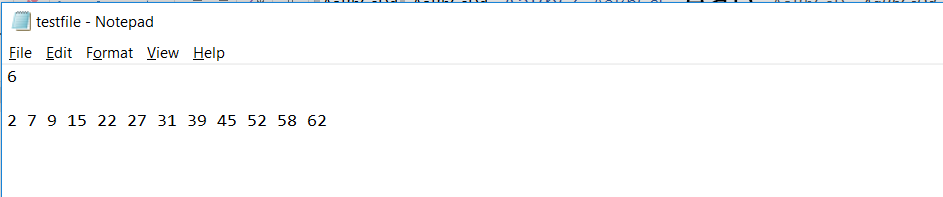
Out of the prime implicants that we received in the last step, we need to find only the essential prime implicants which needs to be exclusively covered. For doing so, we maintain a chart containing the prime implicants and the minterms against them which contains them. While preparing this chart we need to accommodate the terms in the list in the unchecked list. Therefore, we compare each of the term in the original list using the two for loops and compBinarySame function.

# Step 6: Now we need to find the essential prime implicants from the prime implicants. For doing so, we implemented the find\_minimum\_cost function. In this function we use the variable essential\_prime, first to calculate the essential prime and then to remove the redundant terms from the same. For calculating the essential prime, find\_prime function is used which scans the row which have only one entry and mark them as essential prime implicant. The other elements in the row and column in which one of such entry is present is covered automatically. In this way, the essential prime implicants are calculated and the by using the redundant function over it we remove any term which is repeated.

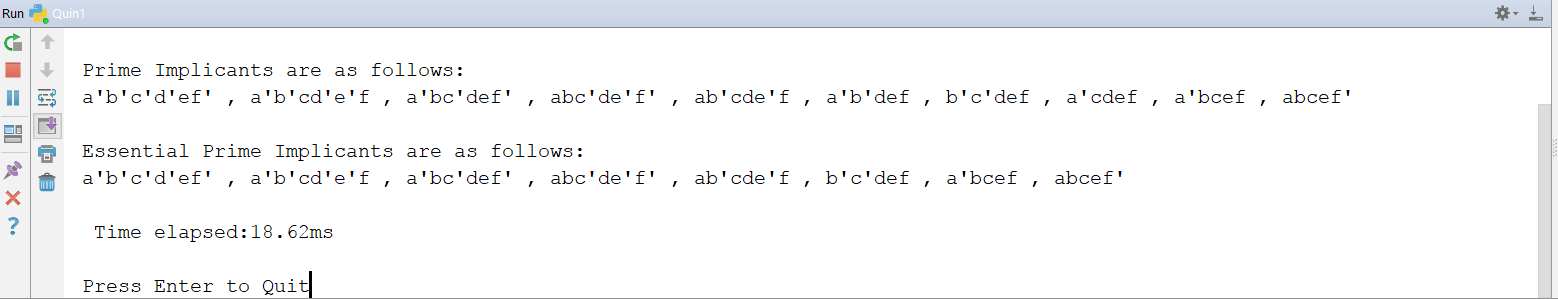
# Step 7: In the last step we will print the prime implicants and the essential prime implicants and also convert the binary representation into letter from which we used earlier.

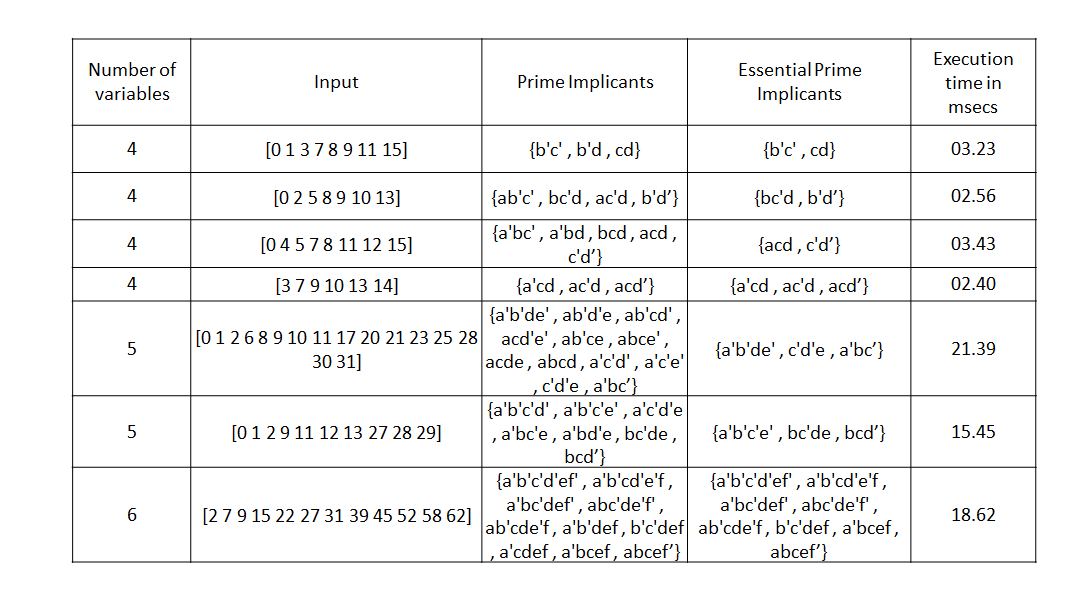
RESULTS:

After parsing the input ‘testfile.txt’, we tried various combinations of min-terms for different files from 4-variable, 5-variable up to 9- variable input.

**Sample input file:**

**Sample output file:**



Comparison results for some of the results for essential prime implicants after removing redundant min-terms are as follows:

For above table, if we plot execution time comparison graph, we come onto conclusion, that despite of more variable input, execution time depends on number of input min-terms.

As we can see, the change is linearly proportional to number of min-terms.

On comparison with K-map for same number of variable, we found that, Quine–McCluskey algorithm serves more efficient than K-map when it comes to minimizing Boolean functions for input variable after 4-variable, as after 4-variable finding solution for K-map becomes complicated.

**Don’t care condition:**

To check the running status for don’t care condition,

1. We entered the following inputs in the system: for a 4-input variable.
2. Next, we gave some of the inputs as Don’t care, out of this: .

OBSERVATION:

In the first case, we got our Essential Prime Implicants(EPI) as AD + A’BD + B’CD’, whereas in the second case we got minimized output as AD+A’CD’+BC’D+B’CD’.

We observed that, this is because, in Don’t care condition, we consider don’t care cases while making chart, but while making match tables, we eliminate them as they need not be considered at the last, where min-terms are matched with redundant and other essential pairs.

CONCLUSION:

* In this project, we successfully implemented the Quine McClusky method used for logic minimization.
* We designed the input files which had the number of input variables and the minterms in each logic equation.
* We analyzed the Quine McClusky method implemented here by running it on various sizes of number of variables. We understood the usage of the Quine McClusky when the number of variables increases and the traditional methods like k map becomes complicated.
* During this experiment we also analyzed the beahviour of Quine Mcclusky in terms of time. We found that as the number of minterms increases the time required also increases. Even if the number of variables are less or remain the same, the time required increases with increase in number of minterms.