Report on Quine-McClauskey Tabular Algorithm in C++

John El Gallab
ID: 900193761
Kirolos Assaad
ID:900191250
under the supervision of Dr. Mohamed Shalan
Computer Science and Engineering Department
The American University in Cairo

Fall 2020

"This report has been compiled using overleaf and LaTeX 2.0 "

1 Objective

The main objective of this project is to explore the Quine McCluskey method's algorithm and how it works by minimizing the expected output of a set of minterms and don't care terms. The stated objective is to be fulfilled by implementing the said method using C++, which would oblige us to go over every possible case; such as having all possible minterms for a set number of variables or none of the minterms at all.

2 Introduction

To be able to understand how the McCluskey method works we need to look more into how it was created, as well as why it is better than some of the other minimizing methods like the Karnaugh Map and the Espresso heuristic logic minimizer.

2.1 Original Creators

The Quine McCluskey method carries this name as it was cocreated by two scientists at two different years; first of them being Willard Quine in 1952 and the other being Edward McCluskey in 1956. Even though the first discovery of this method was by Hugh McColl in 1878 and was later vouched for by Archie Blake in 1937, Quine and McCluskey took the whole credit for it.

2.2 How it works

This Method works by iterating through binary equivalent of all the minterms and don't care terms and comparing them. If there is a difference of one between the two minterms (e.g: 000 and 001), this number is eliminated and is replaced with a dash. This repeated across all minterms, across all the implicants obtained from the previous steps. The same iteration is repeated until there can be no further comparisons, which would give out all the prime implicants. The prime implicants are to be put in a coverage table, which consists of the minterms and the prime implicants obtained from the previous set of steps. If any of the minterms is only repeated once, the corresponding prime implicants are marked as essential, which means that they must be written for the function to work. Then, if any of the rest of the minterms pass through all the prime implicants they are crossed out as they would pass through all of the prime implicants, thus if any of them were to be used they would pass through it. If any of the prime implicants correspond to a minterm covered by another prime implicant, this minterm is called a dominated one, which is also crossed out as it will be covered by another minterm. The last couple of steps will be repeated until no further iterations can take place. The remaining prime implicants from the previous steps in the coverage table are to be added together to give out the sum of products of the function (SOP).

2.3 QM Algorithm verus K-Map Method

Although this method has its flaws such as being very time consuming, it still overpowers the other two most popular minimization methods known; the Karnaugh map (K-map) and the espresso method. The Karnaugh map method is thought to be one of the easiest methods used up to four variables, then the complexity of the map increases as multidimensional maps are needed to be able to successfully obtain the final Boolean expression. On the other hand, the Quine McCluskey method only requires repeating the same iteration method a set of times until no further iterations are possible.

2.4 QM Algorithm vs Espresso Method

The Espresso method is also one of the most well-known minimizers due to its very fast execution. Although this method is thought to be one of the fastest if not the fastest, it still has its flaws, as it gives up minimality for speed. When this method is used, usually the user is willing to give up the accuracy and the precision of the final Boolean expression, just for speed. Upon looking at the previous two examples, we can easily determine the Quine McCluskey method as one of the top ones, as it has low complexity, precision and minimality.

3 Documentation

In this section, all of the functions used in the code will be written with their a brief definition of what they are responsible for.

3.1 fetchFromFile

This function gives out the number of variables, a vector of all the minterms and another one for all the don't care terms, and a final one that has both the minterms and the don't care terms together.

3.2 decToBinInterior

This function converts its parameter from decimal to binary and returns it.

3.3 decToBinary

This function calls on the previous function (decToBinInterior) to loop over the vector needed to be converted.

3.4 padding

This function adds zeros to the beginning of a function so that all the minterms and don't care terms are of the same size, which is the variable size. (e.g.: 10 becomes 0010 if the number of variables inputted in 4)

3.5 printVector

This function takes a vector of strings and prints out all the terms it is carrying.

3.6 printDoubleVector

This function takes a vector of doubles and prints out all the terms it is carrying.

3.7 removeRedundancy

This function loops over a vector of strings and removes any duplicate in it.

3.8 removeRedundancyDecimal

This function loops over a vector of doubles and removes any duplicates in it.

3.9 checkDiff

This function checks the difference between two strings and if the number of differences is one, it is replaced with a dash.

3.10 getIteration:

This function pushes back the ith prime implicant.

3.11 primeImplicantGenerator

This function loops on the iteration function to get the whole list of prime implicants.

3.12 EPImin

This function gives out the minterms of all the essential prime implicants.

3.13 possibility

This function checks and pushes back one essential prime implicant if found at string a.

3.14 EPI

This function loops over the previous function getting the list of all essential prime implicants.

3.15 nonused

This function returns the decimal of all the minterms that have not been used due to not being essential.

3.16 boolexpression

This function takes the essential prime implicants and returns the Boolean expression of it.

3.17 printallPI

This function prints the binary representation of the prime implicants as well as the minterms and the don't care terms used in them.

3.18 allonescounter

This function checks the corner case of all the minterms and don't care terms are one which means the final prime implicant is just dashes and only returns one.

3.19 runProgram

This function runs the whole program and is called at in the int main function.

4 Corner Cases

List of all corner cases that out algorithm handles

4.1 All zeros

If a variable number is given, but there are no minterms or don't care terms (which means that all of their values are zero) the code exits and outputs 0.

4.2 All Ones

If all the possibilities of a variable is written either as minterms or don't care terms (resulting in a prime implicant of all dases) the function outputs one and exits the code.

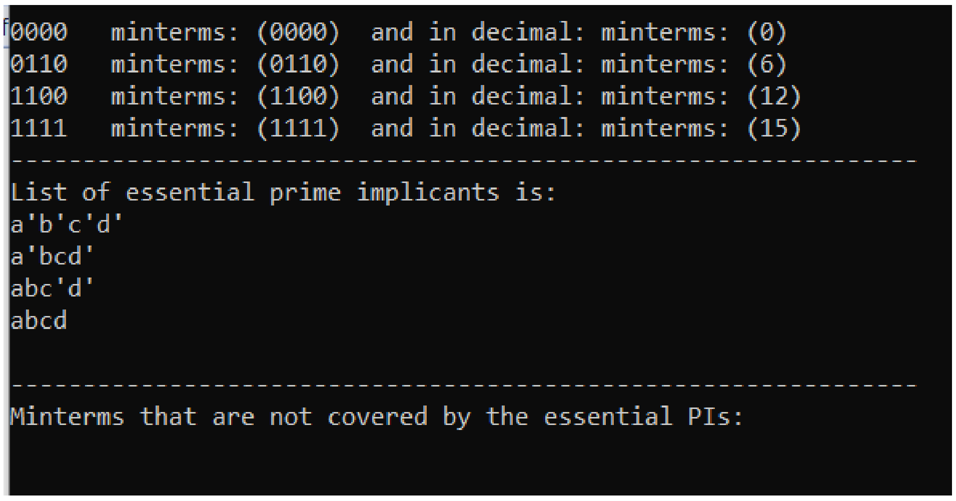
4.3 Mistakes in file

If any error was detected during the validation of the input file, the code exits and gives out a warning.

5 Test cases

Test cases have been test against values from other methods that are known to work to ensure that our program runs as expected. All test cases produced output as expected; thus, our algorithm is deemed accurate. Our test cases also cover corner cases that our code handles well. All inputs to test cases can be found within the text files attached to the project. If any of the below figure are not clear they are also attached with the project :)

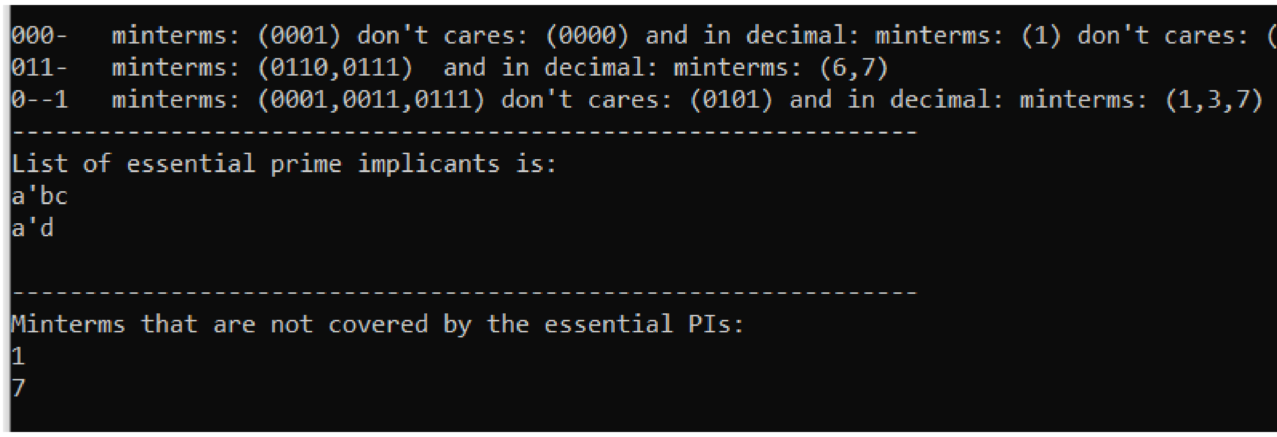
5.1 Test case 1



```
0000   minterms: (0000)  and in decimal: minterms: (0)
0110   minterms: (0110)  and in decimal: minterms: (6)
1100   minterms: (1100)  and in decimal: minterms: (12)
1111   minterms: (1111)  and in decimal: minterms: (15)
---------------------------------------------------------------
List of essential prime implicants is:
a'b'c'd'
a'bcd'
abc'd'
abcd

---------------------------------------------------------------
Minterms that are not covered by the essential PIs:
```

Figure: Output of Test Case 1

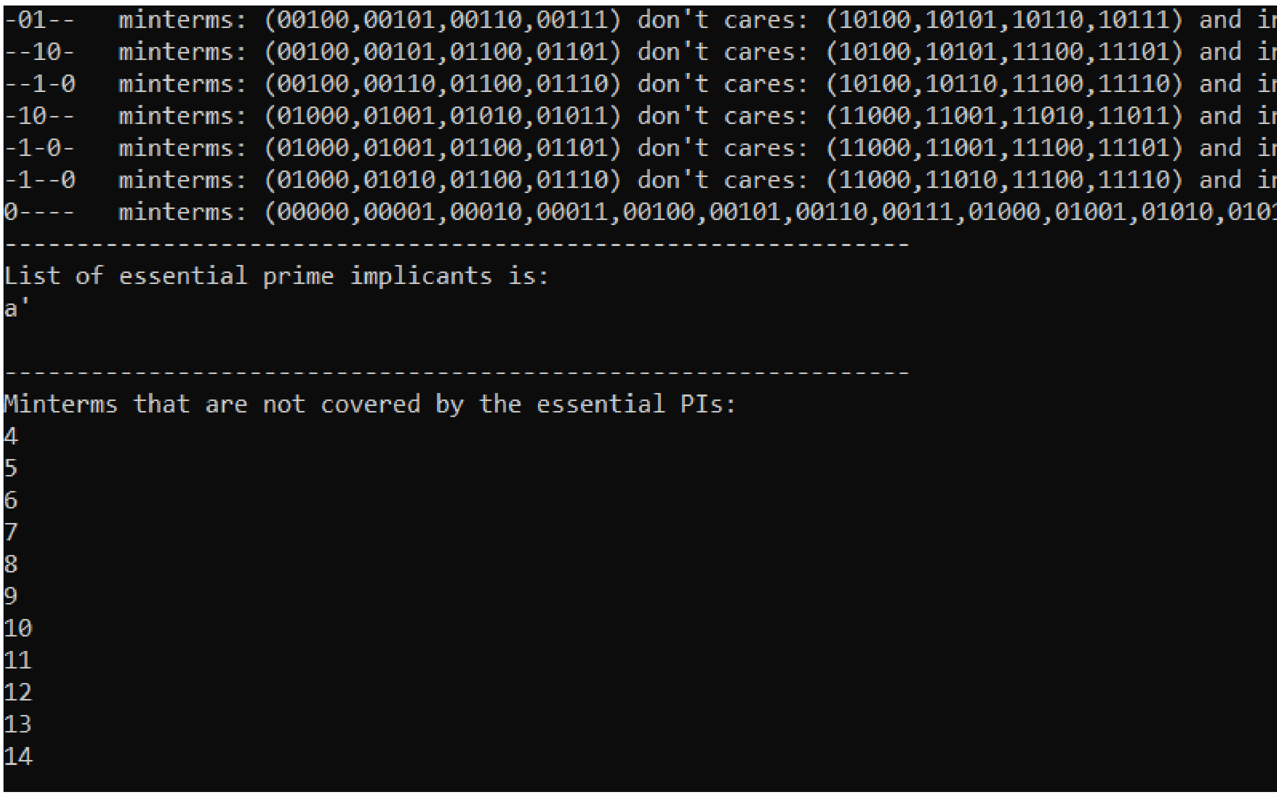
5.2 Test case 2



```
000-   minterms: (0001) don't cares: (0000) and in decimal: minterms: (1) don't cares: (
011-   minterms: (0110,0111)  and in decimal: minterms: (6,7)
0--1   minterms: (0001,0011,0111) don't cares: (0101) and in decimal: minterms: (1,3,7)
---------------------------------------------------------------
List of essential prime implicants is:
a'bc
a'd

---------------------------------------------------------------
Minterms that are not covered by the essential PIs:
1
7
```

Figure: Output of Test Case 2

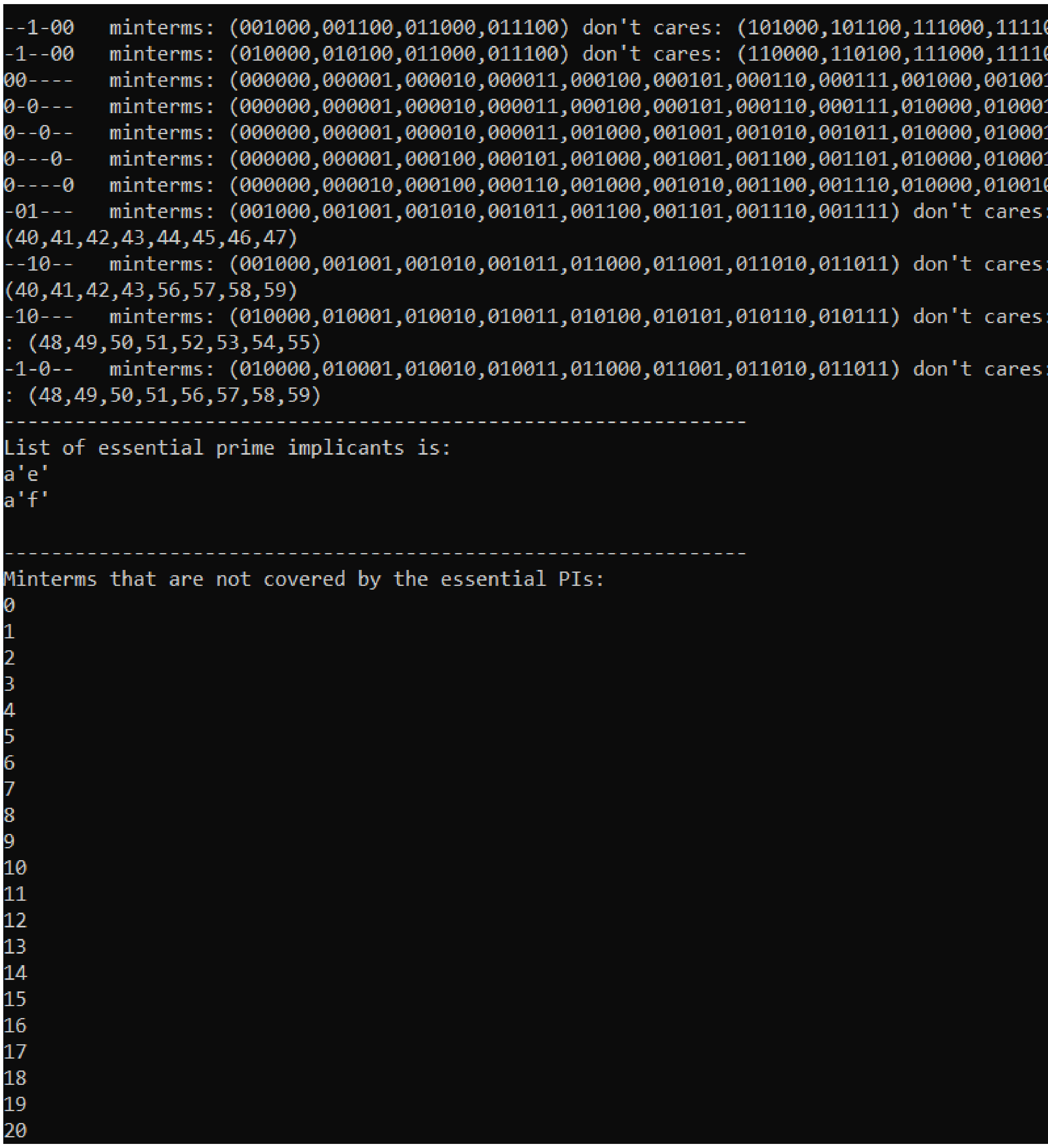
5.3 Test case 3



```
-01--   minterms: (00100,00101,00110,00111) don't cares: (10100,10101,10110,10111) and ir
--10-   minterms: (00100,00101,01100,01101) don't cares: (10100,10101,11100,11101) and ir
--1-0   minterms: (00100,00110,01100,01110) don't cares: (10100,10110,11100,11110) and ir
-10--   minterms: (01000,01001,01010,01011) don't cares: (11000,11001,11010,11011) and ir
-1-0-   minterms: (01000,01001,01100,01101) don't cares: (11000,11001,11100,11101) and ir
-1--0   minterms: (01000,01010,01100,01110) don't cares: (11000,11010,11100,11110) and ir
0----   minterms: (00000,00001,00010,00011,00100,00101,00110,00111,01000,01001,01010,0101
---------------------------------------------------------------
List of essential prime implicants is:
a'

---------------------------------------------------------------
Minterms that are not covered by the essential PIs:
4
5
6
7
8
9
10
11
12
13
14
```

Figure: Output of Test Case 3

5.4 Test case 4



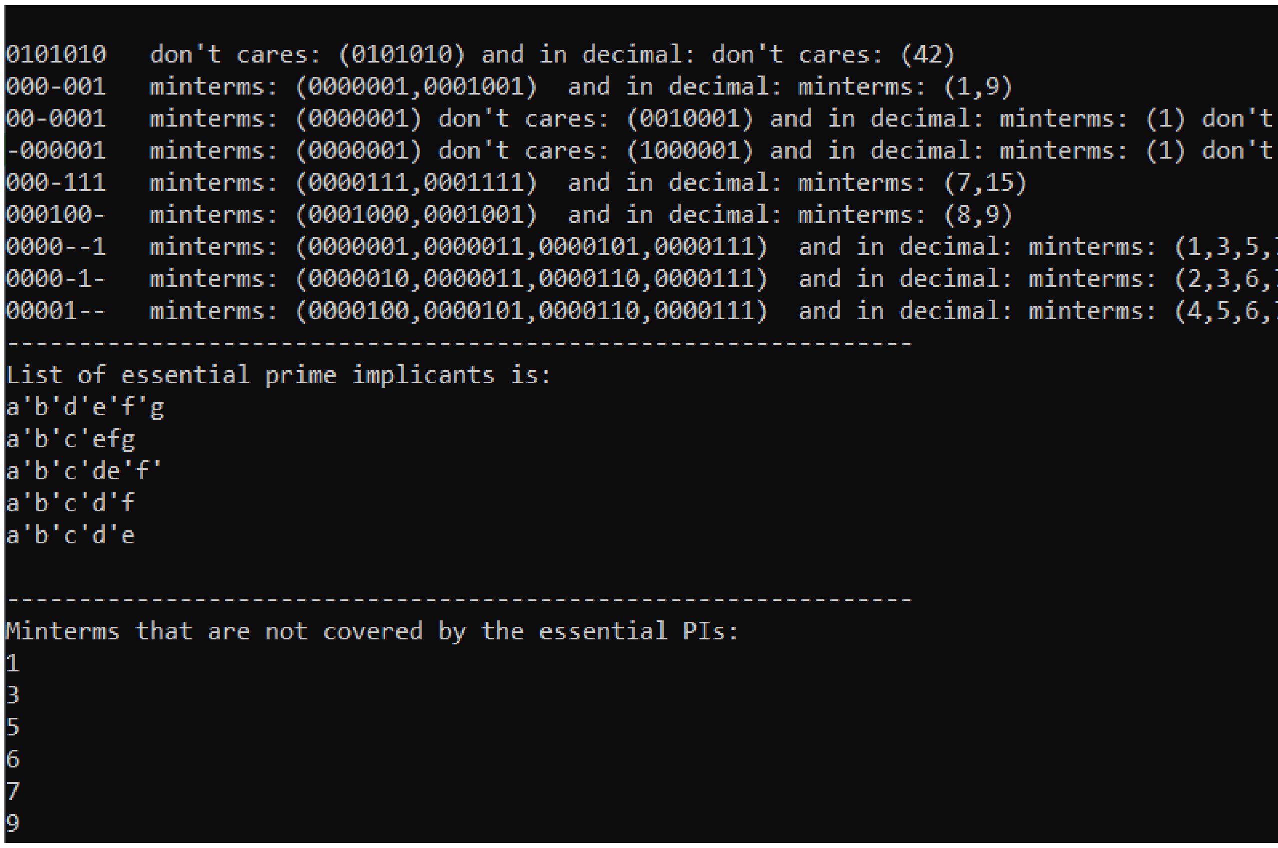
```
--1-00   minterms: (001000,001100,011000,011100) don't cares: (101000,101100,111000,11110
-1--00   minterms: (010000,010100,011000,011100) don't cares: (110000,110100,111000,11110
00----   minterms: (000000,000001,000010,000011,000100,000101,000110,000111,001000,00100
0-0---   minterms: (000000,000001,000010,000011,000100,000101,000110,000111,010000,01000
0--0--   minterms: (000000,000001,000010,000011,001000,001001,001010,001011,010000,01000
0---0-   minterms: (000000,000001,000100,000101,001000,001001,001100,001101,010000,01000
0----0   minterms: (000000,000010,000100,000110,001000,001010,001100,001110,010000,01001
-01---   minterms: (001000,001001,001010,001011,001100,001101,001110,001111) don't cares
(40,41,42,43,44,45,46,47)
--10--   minterms: (001000,001001,001010,001011,011000,011001,011010,011011) don't cares
(40,41,42,43,56,57,58,59)
-10---   minterms: (010000,010001,010010,010011,010100,010101,010110,010111) don't cares
: (48,49,50,51,52,53,54,55)
-1-0--   minterms: (010000,010001,010010,010011,011000,011001,011010,011011) don't cares
: (48,49,50,51,56,57,58,59)
---------------------------------------------------------------
List of essential prime implicants is:
a'e'
a'f'

---------------------------------------------------------------
Minterms that are not covered by the essential PIs:
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

Figure: Output of Test Case 4 part 1

```
21
22
23
24
25
26
27
28
```

Figure: Output of Test Case 4 part 2

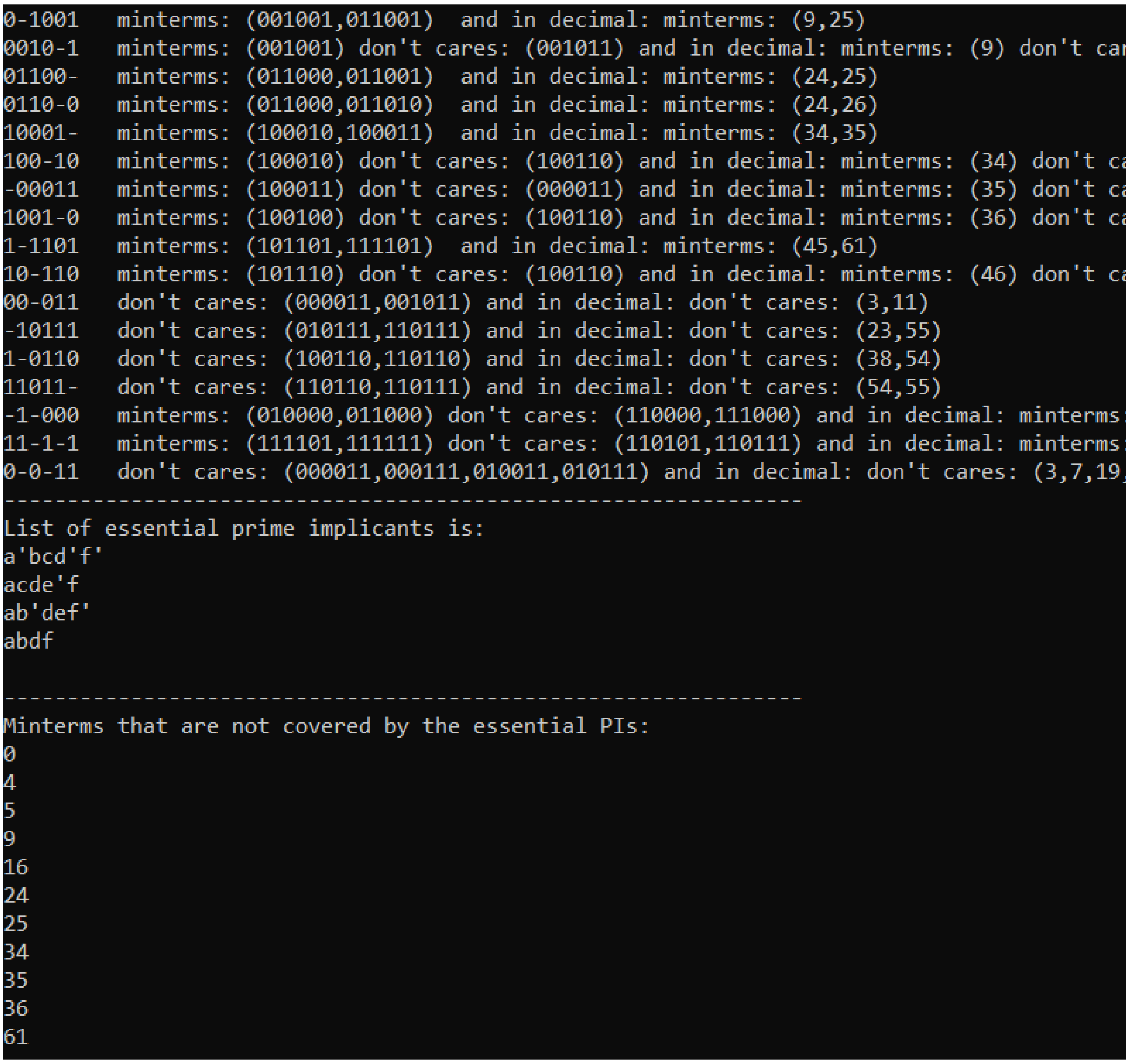
5.5 Test case 5



```

0101010   don't cares: (0101010) and in decimal: don't cares: (42)
000-001   minterms: (0000001,0001001)  and in decimal: minterms: (1,9)
00-0001   minterms: (0000001) don't cares: (0010001) and in decimal: minterms: (1) don't
-000001   minterms: (0000001) don't cares: (1000001) and in decimal: minterms: (1) don't
000-111   minterms: (0000111,0001111)  and in decimal: minterms: (7,15)
000100-   minterms: (0001000,0001001)  and in decimal: minterms: (8,9)
0000--1   minterms: (0000001,0000011,0000101,0000111)  and in decimal: minterms: (1,3,5,
0000-1-   minterms: (0000010,0000011,0000110,0000111)  and in decimal: minterms: (2,3,6,
00001--   minterms: (0000100,0000101,0000110,0000111)  and in decimal: minterms: (4,5,6,
---------------------------------------------------------------
List of essential prime implicants is:
a'b'd'e'f'g
a'b'c'efg
a'b'c'de'f'
a'b'c'd'f
a'b'c'd'e

---------------------------------------------------------------
Minterms that are not covered by the essential PIs:
1
3
5
6
7
9
```

Figure: Output of Test Case 5

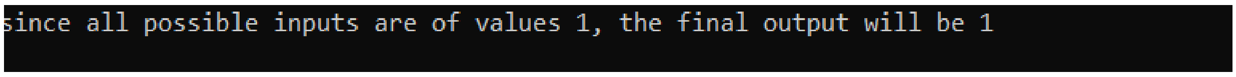
5.6 Test case 6



```
0-1001   minterms: (001001,011001)  and in decimal: minterms: (9,25)
0010-1   minterms: (001001) don't cares: (001011) and in decimal: minterms: (9) don't car
01100-   minterms: (011000,011001)  and in decimal: minterms: (24,25)
0110-0   minterms: (011000,011010)  and in decimal: minterms: (24,26)
10001-   minterms: (100010,100011)  and in decimal: minterms: (34,35)
100-10   minterms: (100010) don't cares: (100110) and in decimal: minterms: (34) don't ca
-00011   minterms: (100011) don't cares: (000011) and in decimal: minterms: (35) don't ca
1001-0   minterms: (100100) don't cares: (100110) and in decimal: minterms: (36) don't ca
1-1101   minterms: (101101,111101)  and in decimal: minterms: (45,61)
10-110   minterms: (101110) don't cares: (100110) and in decimal: minterms: (46) don't ca
00-011   don't cares: (000011,001011) and in decimal: don't cares: (3,11)
-10111   don't cares: (010111,110111) and in decimal: don't cares: (23,55)
1-0110   don't cares: (100110,110110) and in decimal: don't cares: (38,54)
11011-   don't cares: (110110,110111) and in decimal: don't cares: (54,55)
-1-000   minterms: (010000,011000) don't cares: (110000,111000) and in decimal: minterms
11-1-1   minterms: (111101,111111) don't cares: (110101,110111) and in decimal: minterms
0-0-11   don't cares: (000011,000111,010011,010111) and in decimal: don't cares: (3,7,19,
----------------------------------------------------------------
List of essential prime implicants is:
a'bcd'f'
acde'f
ab'def'
abdf

----------------------------------------------------------------
Minterms that are not covered by the essential PIs:
0
4
5
9
16
24
25
34
35
36
61
```

Figure: Output of Test Case 6

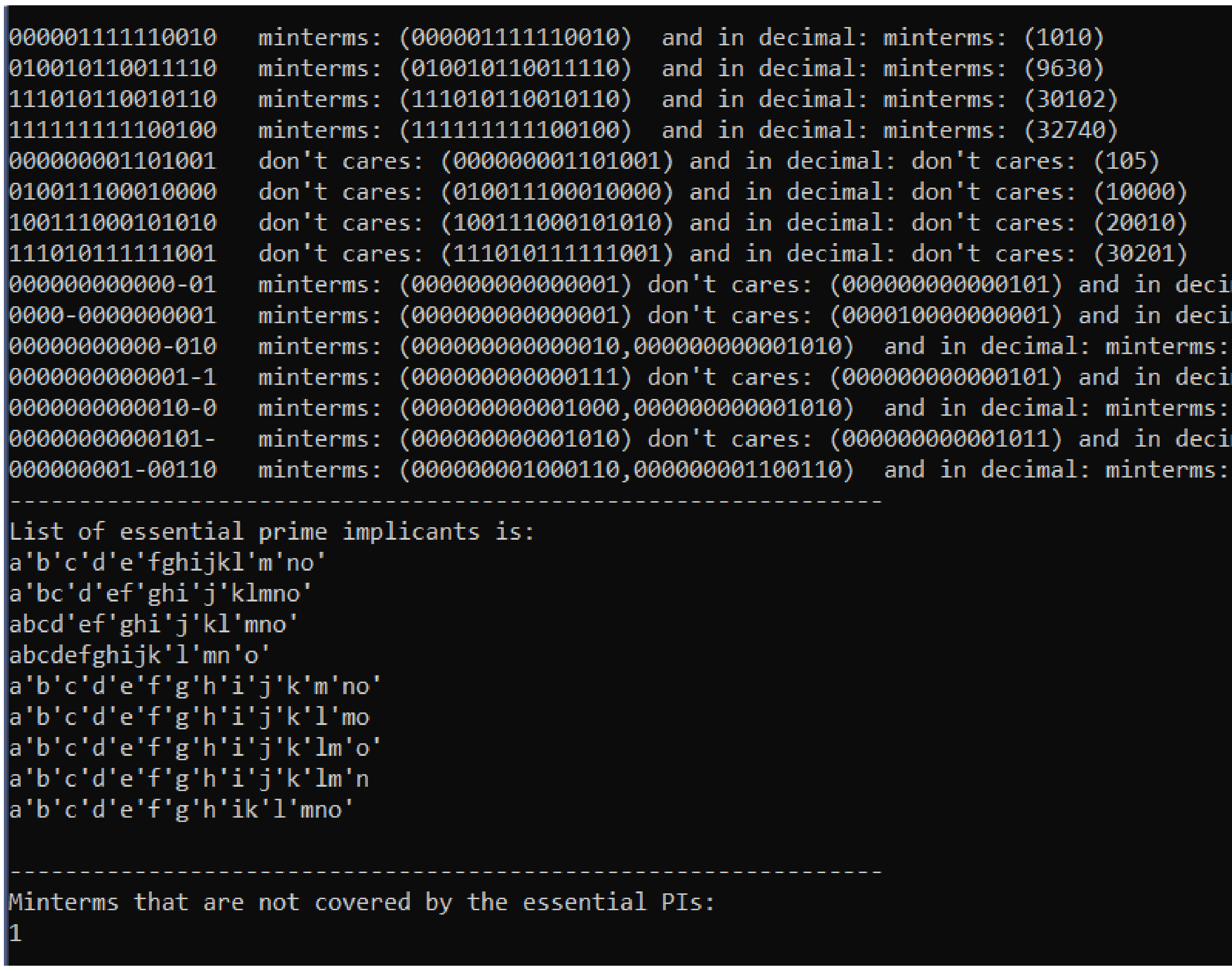
5.7 Test case 7



```
since all possible inputs are of values 1, the final output will be 1
```

Figure: Output of Test Case 7

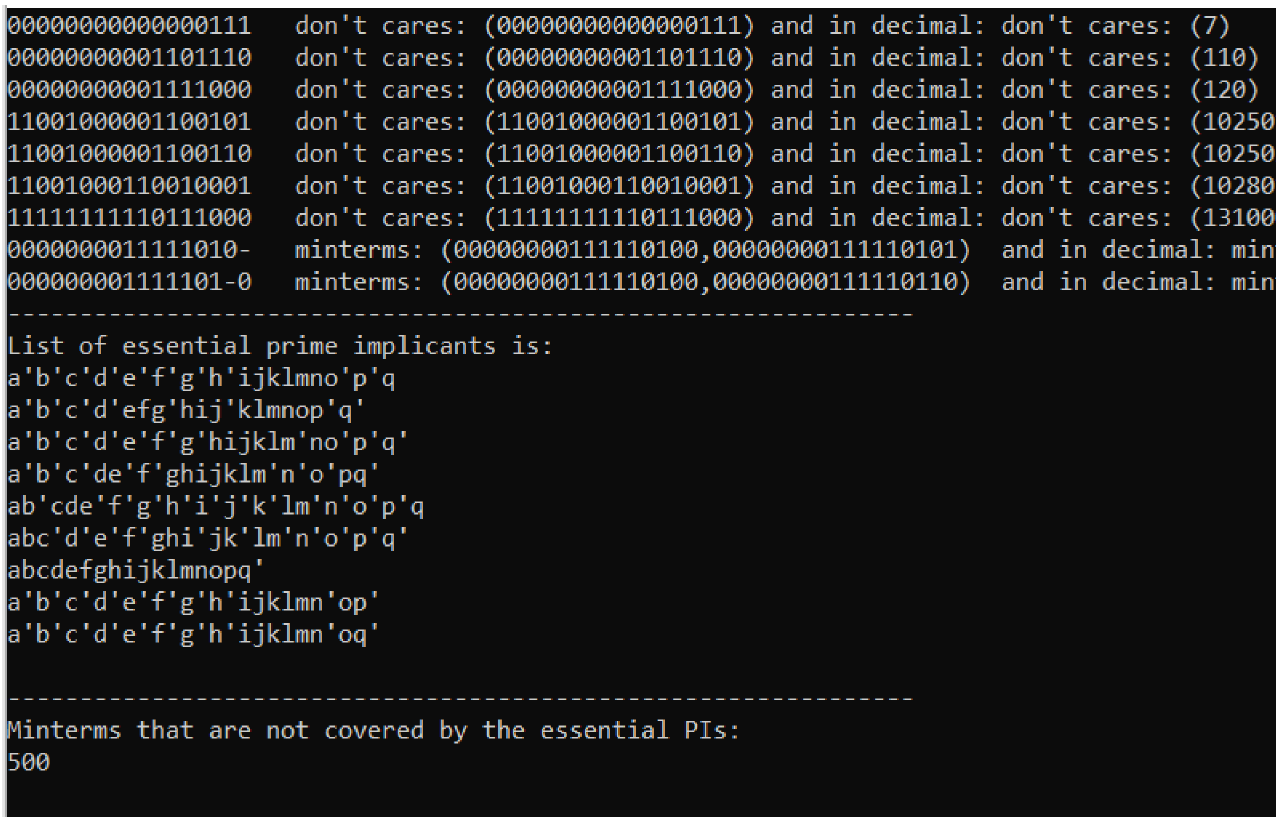
5.8 Test case 8



```
000001111110010   minterms: (000001111110010)  and in decimal: minterms: (1010)
010010110011110   minterms: (010010110011110)  and in decimal: minterms: (9630)
111010110010110   minterms: (111010110010110)  and in decimal: minterms: (30102)
111111111100100   minterms: (111111111100100)  and in decimal: minterms: (32740)
000000001101001   don't cares: (000000001101001) and in decimal: don't cares: (105)
010011100010000   don't cares: (010011100010000) and in decimal: don't cares: (10000)
100111000101010   don't cares: (100111000101010) and in decimal: don't cares: (20010)
111010111111001   don't cares: (111010111111001) and in decimal: don't cares: (30201)
000000000000-01   minterms: (000000000000001) don't cares: (000000000000101) and in deci
0000-0000000001   minterms: (000000000000001) don't cares: (000010000000001) and in deci
00000000000-010   minterms: (000000000000010,000000000001010)  and in decimal: minterms:
0000000000001-1   minterms: (000000000000111) don't cares: (000000000000101) and in deci
0000000000010-0   minterms: (000000000001000,000000000001010)  and in decimal: minterms:
00000000000101-   minterms: (000000000001010) don't cares: (000000000001011) and in deci
000000001-00110   minterms: (000000001000110,000000001100110)  and in decimal: minterms:
----------------------------------------------------------------
List of essential prime implicants is:
a'b'c'd'e'fghijkl'm'no'
a'bc'd'ef'ghi'j'klmno'
abcd'ef'ghi'j'kl'mno'
abcdefghijk'l'mn'o'
a'b'c'd'e'f'g'h'i'j'k'm'no'
a'b'c'd'e'f'g'h'i'j'k'l'mo
a'b'c'd'e'f'g'h'i'j'k'lm'o'
a'b'c'd'e'f'g'h'i'j'k'lm'n
a'b'c'd'e'f'g'h'ik'l'mno'

----------------------------------------------------------------
Minterms that are not covered by the essential PIs:
1
```

Figure: Output of Test Case 8

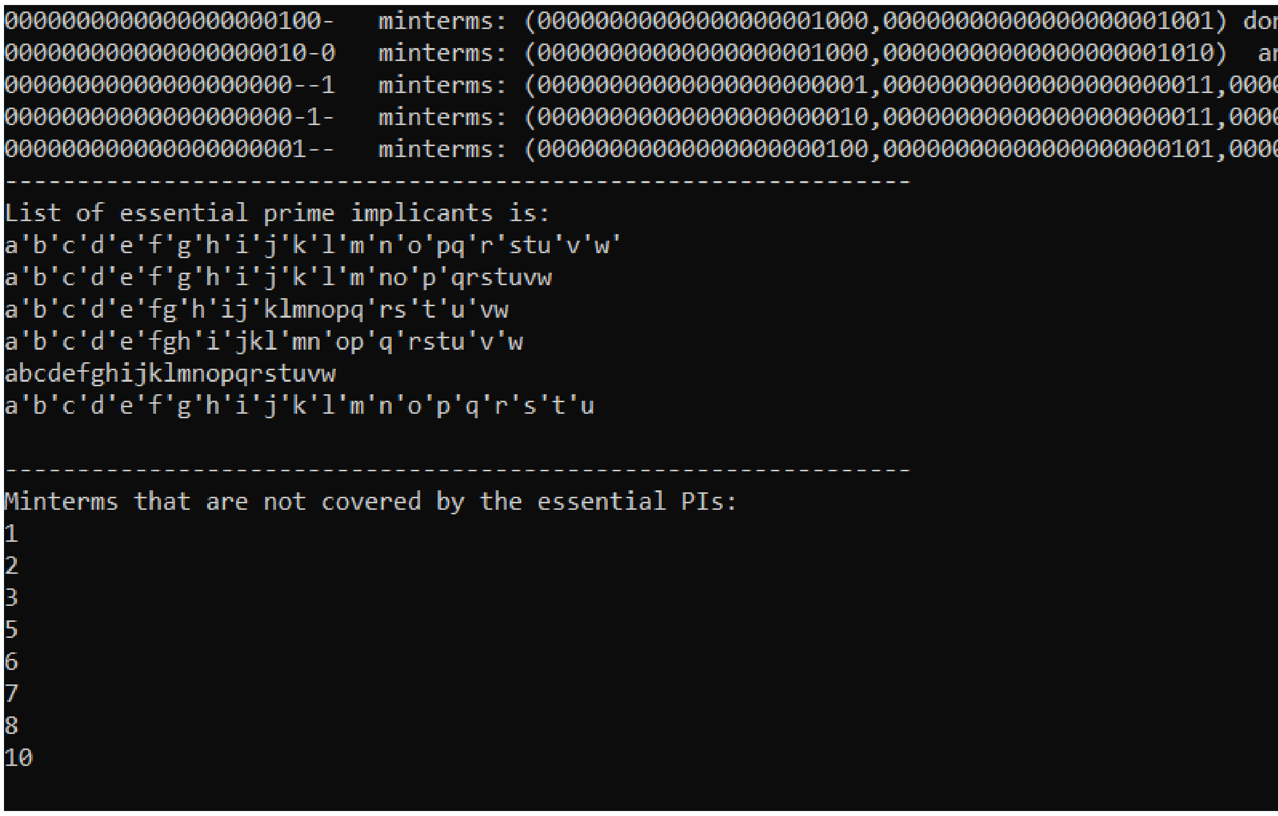
5.9 Test case 9



```
0000000000000111   don't cares: (0000000000000111) and in decimal: don't cares: (7)
00000000001101110  don't cares: (00000000001101110) and in decimal: don't cares: (110)
00000000001111000  don't cares: (00000000001111000) and in decimal: don't cares: (120)
11001000001100101  don't cares: (11001000001100101) and in decimal: don't cares: (10250
11001000001100110  don't cares: (11001000001100110) and in decimal: don't cares: (10250
11001000110010001  don't cares: (11001000110010001) and in decimal: don't cares: (10280
11111111110111000  don't cares: (11111111110111000) and in decimal: don't cares: (13100
0000000011111010-  minterms: (00000000111110100,00000000111110101)  and in decimal: min
000000001111101-0  minterms: (00000000111110100,00000000111110110)  and in decimal: min
---------------------------------------------------------------
List of essential prime implicants is:
a'b'c'd'e'f'g'h'ijklmno'p'q
a'b'c'd'efg'hij'klmnop'q'
a'b'c'd'e'f'g'hijklm'no'p'q'
a'b'c'de'f'ghijklm'n'o'pq'
ab'cde'f'g'h'i'j'k'lm'n'o'p'q
abc'd'e'f'ghi'jk'lm'n'o'p'q'
abcdefghijklmnopq'
a'b'c'd'e'f'g'h'ijklmn'op'
a'b'c'd'e'f'g'h'ijklmn'oq'

---------------------------------------------------------------
Minterms that are not covered by the essential PIs:
500
```

Figure: Output of Test Case 9

5.10 Test case 10



```
0000000000000000000100-   minterms: (00000000000000000001000,00000000000000000001001) dor
0000000000000000000010-0  minterms: (00000000000000000001000,00000000000000000001010)  ar
00000000000000000000--1   minterms: (00000000000000000000001,00000000000000000000011,0000
00000000000000000000-1-   minterms: (00000000000000000000010,00000000000000000000011,0000
000000000000000000001--   minterms: (00000000000000000000100,00000000000000000000101,0000
--------------------------------------------------------------
List of essential prime implicants is:
a'b'c'd'e'f'g'h'i'j'k'l'm'n'o'pq'r'stu'v'w'
a'b'c'd'e'f'g'h'i'j'k'l'm'no'p'qrstuvw
a'b'c'd'e'fg'h'ij'klmnopq'rs't'u'vw
a'b'c'd'e'fgh'i'jkl'mn'op'q'rstu'v'w
abcdefghijklmnopqrstuvw
a'b'c'd'e'f'g'h'i'j'k'l'm'n'o'p'q'r's't'u

--------------------------------------------------------------
Minterms that are not covered by the essential PIs:
1
2
3
5
6
7
8
10
```

Figure: Output of Test Case 10

6 Errors in the Submission

Although many errors were encountered during the process of creation, production, and debugging of this C++ code, no errors are present in the submitted algorithm and runs perfectly fine for all cases of variable size up to 23 variables.

7 readme file

Hello :) To access the source code you will either find it attached to the blackboard assignment itself or in github using this link under the main branch: Please make sure that you have signed into your github account inorder to be able to open the link :) https://github.com/johnelgallab/QM$project-------------------$

8 Contributions

Kirolos Morcos has created the following functions: functions 12,13,14,15,16,17,18, and 19. He has also debugged the remaining functions and helped in catching errors and corner cases in the remaining of the code.

John El Gallab has created the following functions: function 1,2,3,4,5,6,7,8,9,10, and 11. He has also debugged the remaining functions and helped in catching errors and corner cases in the remaining of the code.