Logic System 2023 Assignment II

Simple CAD Tool Based on Quine-McCluskey Method

姓名：劉冠妤　　學號：E24116152

- How my program implement the combine process of QM method?

1. 轉換成二進制

首先，將 minterms（vector<string>）和 dontcares（vector<string>）由十進制轉換成二進制的 string，且利用 switch 語句控制長度為變數的個數（numVar）。程式碼如下（以 numVar=8 為例）：

```
126.     case 8:
127.         for (int i = 0; i < minterms.size(); i++)
128.         {
129.             bitset<8> binary(minterms[i]);
130.             terms.push_back(binary.to_string());
131.             MIN.push_back(binary.to_string());
132.         }
133.         for (int i = 0; i < dontcares.size(); i++)
134.         {
135.             bitset<8> binary(dontcares[i]);
136.             terms.push_back(binary.to_string());
137.             DON.push_back(binary.to_string());
138.         }
139.         break;
```

將轉換後的結果存進 terms（vector<string>），同時將 minterms 的部分存進 MIN（vector<string>），將 dontcares 的部分存進 DON（vector<string>）。

2. 初始化

在開始進行比對前，先確認 MIN、DON 和 terms 中沒有空的元素，避免之後的比對出現錯誤，並將用來標記 terms 有無被使用的向量 used（vector<bool>）初始為 false。程式碼如下：

```
172.     MIN.erase(remove(MIN.begin(), MIN.end(), ""), MIN.end());
173.     DON.erase(remove(DON.begin(), DON.end(), ""), DON.end());
174.     terms.erase(remove(terms.begin(), terms.end(), ""), terms.end());
175.     used.resize(terms.size(), false);
```

3. **合併開始**

用 round（int）紀錄現在是第幾輪的合併，從第一輪開始，利用 while 迴圈重複比對的過程。

```
177.     int round = 1;
178.     while (round <= numVar)
```

用兩層 for 迴圈兩兩比較 terms 中的元素，並在迴圈中創建 diff（string）且初始化為""（若省略此步驟會造成程式執行錯誤），用來記錄比較的結果。

```
180.         for (int i = 0; i < terms.size(); i++)
181.         {
182.             for (int j = i + 1; j < terms.size(); j++)
183.             {
184.                 string diff = "";
```

先判斷"-"的位置是否相同，若不同則直接跳下一個元素。

```
189.                 if ((terms[i][k] == '-' && terms[j][k] != '-') || (terms[i][k] != '-' && terms[j][k] == '-'))
190.                 {
191.                     continue;
192.                 }
```

若"-"的位置相同，則將兩兩元素不相同的位置標上"-"，相同的位置則保留原來的數字，比對完成後 diff 中所存的即是比對的結果，同時用 countDiffOnes（int）來記錄相差的 1 的個數。

```
193.                     else if (terms[i][k] != terms[j][k])
194.                     {
195.                         diff += "-";
196.                         if (terms[i][k] == '1' || terms[j][k] == '1')
197.                             countDiffOnes++;
198.                     }
199.                     else
200.                     {
201.                         diff += terms[i][k];
202.                     }
```

接下來，用 count（int）數 diff 中"-"的個數，若 count=round 且 countDiffOnes=1，才將 diff 存進 nexterms（vector<string>），且將該兩元素在 used 中標記為 true。

```
204.                int count = 0;
205.                for (const auto& c : diff)
206.                {
207.                    if (c == '-')
208.                        count++;
209.                }
210.                if (count == round && countDiffOnes == 1)
211.                {
212.                    nexterms.push_back(diff);
213.                    used[i] = true;
214.                    used[j] = true;
215.                }
```

現在，所有符合條件的新的合併結果都在 nexterms 中，此時先判斷 nexterms 這個向量的大小，若為 0 則表示沒有合理的合併結果，也就是合併已結束，此時離開迴圈。

```
291.            if (nexterms.size() == 0)
292.            {
293.                break;
294.            }
```

若 nexterms 的大小不為 0，表示合併尚未結束，此時要將沒有使用到的 terms 加回 nexterms 中，且該 terms 若來自 DON 則不加入。完成後將 nexterms 存入 terms 中，並清空 nexterms 和 used，且重新設定 used 的大小為新的 terms 的大小，且初始化為 false，以進入下一輪的比對。程式碼如下：

```
295.         else
296.         {
297.             // Add unused terms
298.             for (int i = 0; i < terms.size(); i++)
299.             {
300.                 if (!used[i])
301.                 {
302.                     // Check if the term is also present in the DON vector
303.                     if (std::find(DON.begin(), DON.end(), terms[i]) == DON.end())
304.                     {
305.                         nexterms.push_back(terms[i]);
306.                     }
307.                 }
308.             }
309.             terms = nexterms;
310.             nexterms.clear();
311.             used.clear();
312.             used.resize(terms.size(), false);
313.         }
```

同時，透過 DON 和 nexdon（vector<string>）進行如上的比較過程，好讓每一輪都能確實過濾掉完全來自 dontcares 的元素，以維持正確性。

```
352.         DON = nexdon;
353.         nexdon.clear();
354.         round++;
```

迴圈的最後將 round 加一，以進行下一輪的對比，直到沒有新的合理的合併結果出現時，即完成合併過程。

4. 刪除來自 DON 的元素

比較結束後的結果在 terms 中，此時因為最後一輪尚未和更新過的 DON 比對，所以在迴圈結束後再檢查一次是否有來自 DON 的元素，若有則刪除，以維持正確性。

```
356.     for (int i = 0; i < terms.size(); i++)
357.     {
358.         for (int j = 0; j < DON.size(); j++)
359.         {
360.             bool same = true;
361.             for (int k = 0; k < terms[i].size(); k++)
362.             {
363.                 if (terms[i][k] != DON[j][k])
364.                 {
365.                     same = false;
366.                     break;
367.                 }
368.             }
369.             if (same)
370.             {
371.                 terms.erase(terms.begin()+i);
372.             }
373.         }
374.     }
```

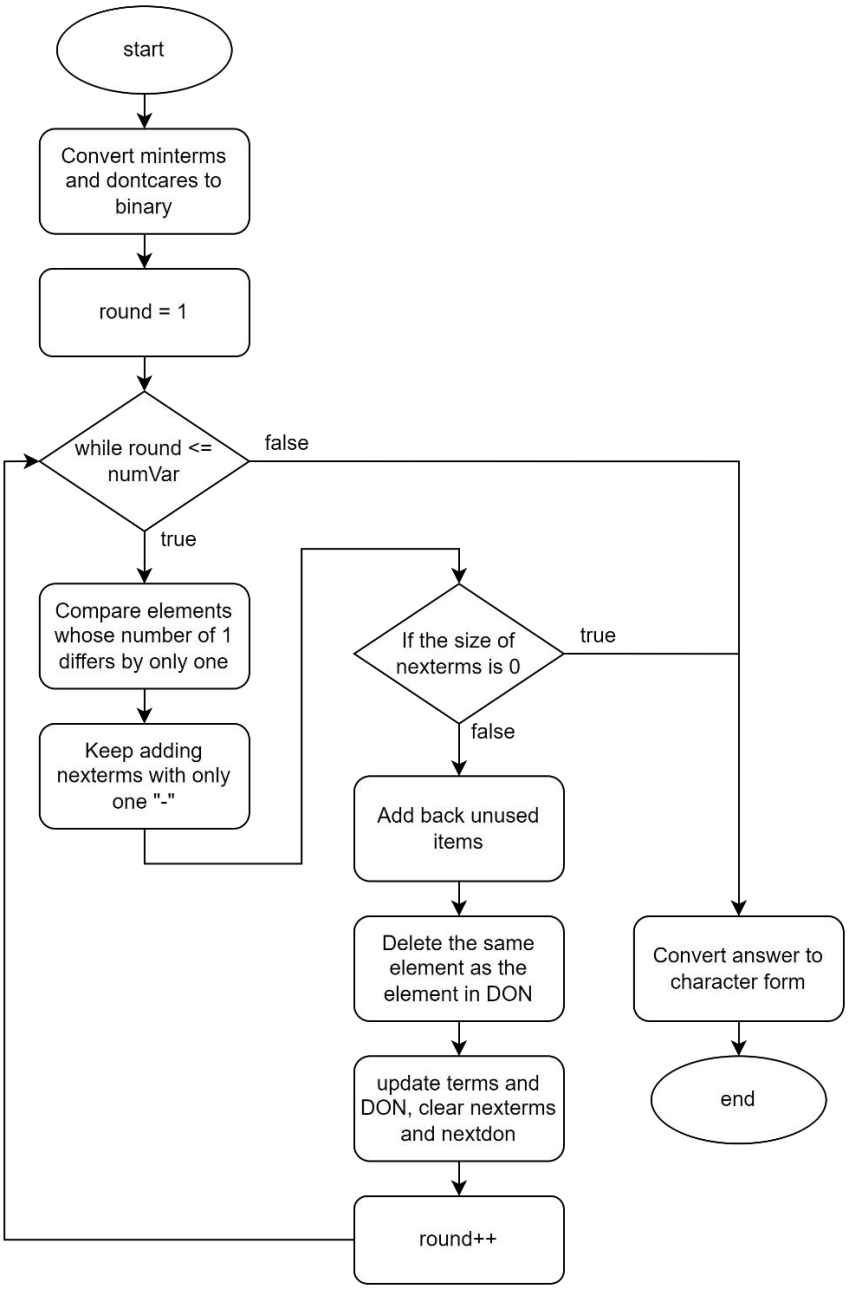
5. 刪除重複的元素

最後將 terms 中重複的元素刪除，便完成合併。刪除重複元素的方式為，先將所有元素用函式 sort 排序，排序後則可使用 unique 函式找出重複的元素並將該元素移動到向量的末尾，最後則用 erase 函式將該元素刪除。

```
375.     std::sort(terms.begin(), terms.end());
376.     auto last = unique(terms.begin(), terms.end());
377.     terms.erase(last, terms.end());
```

總的來說，合併的過程主要是透過 string 的形式儲存每個 minterm 和 dontcare 的二進位值，再利用 while 迴圈重複比對的過程，接著使用兩層的 for 迴圈進行比對相同位置的數字的異同，再每一輪的比較完成後，先確認新的合併結果的個數，若為 0 則表示比對完成，若還有新的合併結果，則加回未使用到的 terms 並刪除由 dontcares 所生成的合併結果，最後更新 terms。每一輪的比對都是用 terms 中的元素進行比對，而結果先暫存在 nexterms，直到比對完成才將結果放回 terms 中，透過兩個 vector<string>的交替使用實現這樣的合併過程。過程中適時的整理向量中的元素，將空的、重複的刪除，以避免錯誤的產生。

Flow chart:



- How my program finds all valid minimum SOPs?

1. 找出 essential prime implicant

透過三層 for 迴圈檢查 MIN 中只被包含一次的元素，matchIdx（int）紀錄包含該 MIN 的元素的 terms 元素的 Index，count（int）數 MIN 被包含的次數，若僅被包含一次，則將相對應的 terms 存入 t1（vector<string>），t1 所存的元素即為 essential prime implicant。

```
467.     for (int i = 0; i < MIN.size(); i++)
468.     {
469.         int matchIdx = -1; // Use this to record the match index
470.         int count = 0;
471.         for (int j = 0; j < terms.size(); j++)
472.         {
473.             bool same_digits = true;
474.             for (int k = 0; k < terms[j].size(); k++)
475.             {
476.                 if (terms[j][k] != '-' && terms[j][k] != MIN[i][k])
477.                 {
478.                     same_digits = false;
479.                     break;
480.                 }
481.             }
482.             if (same_digits)
483.             {
484.                 count++;
485.                 matchIdx = j;
486.             }
487.         }
488.         if (count == 1)
489.         {
490.             t1.push_back(terms[matchIdx]);
491.         }
492.     }
```

2. 找出未被 essential prime implicant 包含的 minterms

接下來，利用三層的 for 迴圈找出有哪些 MIN 的元素尚未被目前的 essential prime implicant 所包含，對於每一個 MIN 皆檢查所有的 t1 元素是否有出現包含該 MIN 元素的，包含的定義即檢查兩者是否除了"-"以外的每個位置的字符皆相同，若有不同則將 same_digits（bool）標記為 false 並跳至下一個 terms 元素，若檢查完某一 terms 元素的所有位置而 same_digits 仍為 true，表示該 terms 元素包含該 MIN 元素，則將 have（bool）標記為 true，並跳至下一個 MIN 元素，若檢查完所有 terms 而 have 仍為 false 則表示該 MIN 元素未被包含，將其加入 non（vector<string>），non 所存的元素即為尚未被包含的 MIN 元素。

```
495.     for (int i = 0; i < MIN.size(); i++)
496.     {
497.         bool have = false;
498.         for (int j = 0; j < t1.size(); j++)
499.         {
500.             bool same_digits = true;
501.             for (int k = 0; k < t1[j].size(); k++)
502.             {
503.                 if (t1[j][k] != '-' && t1[j][k] != MIN[i][k])
504.                 {
505.                     same_digits = false;
506.                     break;
507.                 }
508.             }
509.             if (same_digits)
510.             {
511.                 have = true;
512.                 break;
513.             }
514.         }
515.         if(!have)
516.         {
517.             non.push_back(MIN[i]);
518.         }
519.     }
```

3. 找出 minimum SOPs

接下來進入重要的環節，先宣告 solutions（vector<vector<string>>），用來儲存找到的 sop 組合，每一個組合為一個 vector<string>，同時宣告 t2（vector<string>）複製一份 t1 的內容，即 essential prime implicants，將尚未被包含的 minterms（non）、所有的 prime implicants（terms）、essential prime implicants（t2）和接組合結果的向量（solutions）傳入 solve 這個 function。

```
521.     vector<vector<string>> solutions;
522.     vector<string> t2(t1);
523.     solve(non, terms, t2, solutions);
```

solve 函式：（所有傳入的向量皆是 pass by reference）
先在函式內部新建一個 nonTerms（vector<string>），儲存尚未被使用的 terms。

```
71.     for(auto &term : terms)
72.     {
73.         if(find(t2.begin(), t2.end(), term) == t2.end())
74.         {
75.             nonTerms.push_back(term);
76.         }
77.     }
```

接下來先看 else 內的內容。創建 combinations（vector<vector<string>>）儲存所有 nonTerms 可能的組合，如 nonTerms 有三個元素，可能產生的集合有 23 個。將 nonTerms 的大小、for 迴圈的 i 值、t2 和 combinations 傳入函式 generateCombinations，產生 i 個個數的 nonTerms 元素組合，檢查傳回來的 combinations 中的每一組 vector<string>，用一 t2_candidate（vector<string>）複製 t2 的內容，避免改動到原本的數值，將 combinations 的元素插入 t2_candidate 的末尾，此時的 t2_candidate 所存的為一可能的解，再將 non 和 t2_candidate 傳入 checkValidity 函式檢查是否符合所需的條件，若符合表 t2_candidate 為一 sop，此時將 t2_candidate 存進 solutions。

```
    for(int i = 0; i <= nonTerms.size(); ++i) {
        //cout<<i<<endl;
        if (firstSolutionSize != -1 && i > firstSolutionSize)
        {
            break; // 如果當前的組合大小超過第一個解的大小，則停止檢查
        }
        else{
            vector<vector<string>> combinations;
            generateCombinations(nonTerms.size(), i, nonTerms, t2, combinations);
            for(auto &combination : combinations)
            {
                vector<string> t2_candidate = t2;
                t2_candidate.insert(t2_candidate.end(), combination.begin(), combination.end());
                if(checkValidity(non, t2_candidate))
                {
                    solutions.push_back(t2_candidate);
                    if (firstSolutionSize == -1)
                    {
                        firstSolutionSize = t2_candidate.size() - t2.size(); // 如果找到第一個解，則記錄其大小
                    }
                }
            }
        }

    }
```

而 firstSolutionSize（int）的初始值為-1，因為是從 i 為 0 開始組合，也就是第一個回傳的組合大小為 0，即從僅包含 essential prime implicant 開始檢查，若找到第一個解，則記錄該解的 combination 大小，該大小即為 minimum Sops 的大小，接下來的迴圈只要跑完所有這個大小的 combination 就可以了，因再更大的解就不符合 minimum Sops 的條件。

generateCombinations 函式：

先創建了一個 bitmask 字串，用於標記組合中的元素，前 r 個位元被設為 1，表示這些位元將在組合中使用，其餘的位元則設為 0。而 r 所對應的傳入值為 solve 函式中的迴圈的 i 值，而 n 為 nonTerms 的大小。接下來，使用 do...while 迴圈來生成所有可能的組合。在每次迴圈迭代期間，程式碼會檢查 bitmask 中哪些位元被設為 1，然後將相應的 terms 元素添加到新的 combination 容器中。最後，將 combination 加入到 combinations 容器中。迴圈的條件是使用 prev_permutation 函式對 bitmask 進行前置排列，以便生成下一個組合。即此迴圈將生成所有可能的 r 個元素的組合。

```
{
    string bitmask(r, 1);
    bitmask.resize(n, 0);

    do {
        vector<string> combination;
        for (int i = 0; i < n; ++i)
        {
            if (bitmask[i])
            {
                combination.push_back(terms[i]);
            }
        }
        combinations.push_back(combination);
    } while (prev_permutation(bitmask.begin(), bitmask.end()));
}
```

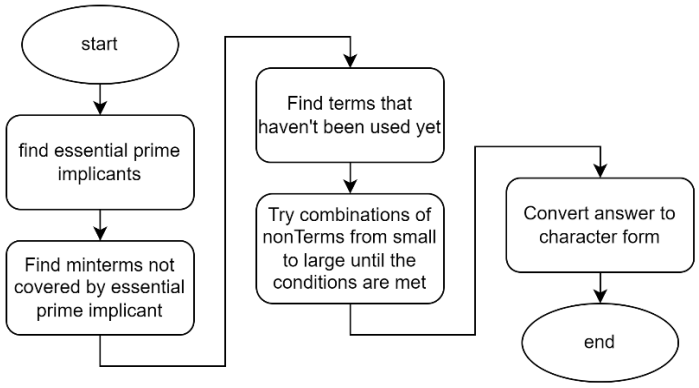
checkValidity 函式：

透過嵌套的迴圈進行比對，對於 non 中的每個字串 n 都會遍歷 t2 中的每個字串 t 進行比較。此函式主要檢查是否兩字串除了"-"以外的數字部分皆相同，如果兩個字符不相等且 t 字串中的字符不是 '-'，則 same_digits（bool）被設置為 false，並退出內部迴圈，表找到了不匹配的情況。如果 same_digits 保持為 true，則表示 t 字串中的字符與 n 字串中的字符匹配。在這種情況下，matched（bool）被設置為 true 並退出內部迴圈。最後，在每次外部迴圈迭代之後，檢查 matched 的值。如果 matched 為 false，則表示在 t2 中沒有找到與 n 匹配的字串，因此函式返回 false，若所有的 non 字串都找到了匹配，則函式返回 true。

```
20.  bool checkValidity(vector<string> &non, vector<string> &t2)
21.  {
22.      for(auto &n : non)
23.      {
24.          bool matched = false;
25.          for(auto &t : t2)
26.          {
27.              bool same_digits = true;
28.              for (int k = 0; k < t.size(); k++)
29.              {
30.                  if (t[k] != '-' && t[k] != n[k])
31.                  {
32.                      same_digits = false;
33.                      break;
34.                  }
35.              }
36.              if (same_digits)
37.              {
38.                  matched = true;
39.                  break;
40.              }
41.          }
42.          if (!matched) return false;
43.      }
44.      return true;
45.  }
```

我所使用到的演算法為**位元運算法（Bit Manipulation）**，是一種基於位元運算的技巧，用於生成組合。利用二進位位元的特性，將組合表示為一個位元遮罩（bitmask），其中每個位元代表一個元素的選擇狀態。通過迭代遮罩的所有可能狀態，可以生成所有的組合。在這裡，我將尚未被使用到的 terms 在 generateCombinations 函式中迭代遮罩出所有可能的組合，且從大小最小的開始，滿足 minimum SOPs 的要求。

Flow chart:



- Test cases

第一組測資：

```
·   numVar = 8
·   minterms = [0, 2, 5, 6, 7, 8, 10, 12, 13, 14, 15]
·   dontcares = [225]
```

輸出結果：

```
=====================    =====================    =====================
Column 1                 Column 2                 Column 3
=====================    =====================    =====================
v 00000000: 0            v 000000-0: 0 2            0000-0-0: 0 2 8 10
---------------------    v 0000-000: 0 8          x 0000-0-0: 0 2 8 10
v 00000010: 2            ---------------------    ---------------------
v 00001000: 8            v 00000-10: 2 6            0000--10: 2 6 10 14
---------------------    v 0000-010: 2 10         x 0000--10: 2 6 10 14
v 00000101: 5            v 000010-0: 8 10           00001--0: 8 10 12 14
v 00000110: 6            v 00001-00: 8 12         x 00001--0: 8 10 12 14
v 00001010: 10           ---------------------    ---------------------
v 00001100: 12           v 000001-1: 5 7            0000-1-1: 5 7 13 15
---------------------    v 0000-101: 5 13         x 0000-1-1: 5 7 13 15
v 00000111: 7            v 0000011-: 6 7            0000-11-: 6 7 14 15
v 00001101: 13           v 0000-110: 6 14         x 0000-11-: 6 7 14 15
v 00001110: 14           v 00001-10: 10 14          000011--: 12 13 14 15
---------------------    v 0000110-: 12 13        x 000011--: 12 13 14 15
v 00001111: 15           v 000011-0: 12 14        ---------------------
d 11100001: 225          ---------------------
---------------------    v 0000-111: 7 15
                         v 000011-1: 13 15
                         v 0000111-: 14 15
                         ---------------------
```

- [v]表示該項被使用
- [d]表示該項為 dontcare
- [-]表示該位置所對應變數為 dontcare
- [x]表示該項重複

```
=====================
Prime Implicant Chart
=====================
                   |   0|   2|   5|   6|   7|   8|  10|  12|  13|  14|  15|
-------------------+----+----+----+----+----+----+----+----+----+----+----+
a'b'c'd'g h'       |    |x   |    |x   |    |    |x   |    |    |x   |    |
a'b'c'd'f'h'       |x   |x   |    |    |    |x   |x   |    |    |    |    |
a'b'c'd'f h        |    |    |x   |    |x   |    |    |    |x   |    |x   |
a'b'c'd'f g        |    |    |    |x   |x   |    |    |    |    |x   |x   |
a'b'c'd'e h'       |    |    |    |    |    |x   |x   |x   |    |x   |    |
a'b'c'd'e f        |    |    |    |    |    |    |    |x   |x   |x   |x   |
-------------------+----+----+----+----+----+----+----+----+----+----+----+
```

- [x]表示包含該 minterm

```
numVar = 8
primes = a'b'c'd'g h', a'b'c'd'f'h', a'b'c'd'f h, a'b'c'd'f g, a'b'c'd'e h', a'b'c'd'e f
minimumSops = a'b'c'd'gh' + a'b'c'd'f'h' + a'b'c'd'fh + a'b'c'd'eh'
a'b'c'd'gh' + a'b'c'd'f'h' + a'b'c'd'fh + a'b'c'd'ef
a'b'c'd'f'h' + a'b'c'd'fh + a'b'c'd'fg + a'b'c'd'eh'
a'b'c'd'f'h' + a'b'c'd'fh + a'b'c'd'fg + a'b'c'd'ef
```

第二組測資：

```
numVar = 9
minterms = [24, 37, 83, 138, 217, 228, 269, 354, 368, 376, 415, 476, 508]
dontcares = [54, 175, 214, 301, 316, 332, 336, 358, 398, 412, 428, 473]
```

輸出結果：

```
=====================
Column 1
=====================
  000011000: 24
---------------------
  000100101: 37
  010001010: 138
d 101010000: 336
---------------------
  001010011: 83
  011100100: 228
v 100001101: 269
v 101100010: 354
v 101110000: 368
d 000110110: 54
d 101001100: 332
---------------------
```

```
---------------------
v 011011001: 217
v 101111000: 376
d 011010110: 214
d 100101101: 301
d 100111100: 316
d 101100110: 358
d 110001110: 398
d 110011100: 412
d 110101100: 428
---------------------
v 111011100: 476
d 010101111: 175
d 111011001: 473
---------------------
  110011111: 415
v 111111100: 508
---------------------
```

```
=====================
Column 2
=====================
  101-10000: 336 368
---------------------
  100-01101: 269 301
  101100-10: 354 358
  10111-000: 368 376
---------------------
  -11011001: 217 473
  11-011100: 412 476
---------------------
  111-11100: 476 508
---------------------
```

- [v]表示該項被使用
- [d]表示該項為 dontcare
- [-]表示該位置所對應變數為 dontcare

```
====================
Prime Implicant Chart
====================
                   |  24|  37|  83| 138| 217| 228| 269| 354| 368| 376| 415| 476| 508|
-------------------+----+----+----+----+----+----+----+----+----+----+----+----+----+
b c d'e f g'h'i    |    |    |    |    |x   |    |    |    |    |    |    |    |    |
a'b'c'd'e f g'h'i' |x   |    |    |    |    |    |    |    |    |    |    |    |    |
a'b'c'd e'f'g h'i  |    |x   |    |    |    |    |    |    |    |    |    |    |    |
a'b'c d'e f'g'h i  |    |    |x   |    |    |    |    |    |    |    |    |    |    |
a'b c'd'e'f g'h i' |    |    |    |x   |    |    |    |    |    |    |    |    |    |
a'b c d e'f'g h'i' |    |    |    |    |    |x   |    |    |    |    |    |    |    |
a b'c'e'f g h'i    |    |    |    |    |    |    |x   |    |    |    |    |    |    |
a b'c e f'g'h'i'   |    |    |    |    |    |    |    |    |x   |    |    |    |    |
a b'c d e'f'h i'   |    |    |    |    |    |    |    |x   |    |    |    |    |    |
a b'c d e g'h'i'   |    |    |    |    |    |    |    |    |x   |x   |    |    |    |
a b d'e f g h'i'   |    |    |    |    |    |    |    |    |    |    |    |x   |    |
a b c'd'e f g h i  |    |    |    |    |    |    |    |    |    |    |x   |    |    |
a b c e f g h'i'   |    |    |    |    |    |    |    |    |    |    |    |x   |x   |
-------------------+----+----+----+----+----+----+----+----+----+----+----+----+----+
```

- [x]表示包含該 minterm

```
numVar = 9
primes = b c d'e f g'h'i, a'b'c'd'e f g'h'i', a'b'c'd e'f'g h'i, a'b'c d'e f'g'h i, a'b c'd'e'f g'h i',
a'b c d e'f'g h'i', a b'c'e'f g h'i, a b'c e f'g'h'i', a b'c d e'f'h i', a b'c d e g'h'i', a b d'e f g
h'i', a b c'd'e f g h i, a b c e f g h'i'
minimumSops = bcd'efg'h'i + a'b'c'd'efg'h'i' + a'b'c'de'f'gh'i + a'b'cd'ef'g'hi + a'bc'd'e'fg'hi' + a'bcd
e'f'gh'i' + ab'c'e'fgh'i + ab'cde'f'hi' + ab'cdeg'h'i' + abc'd'efghi + abcefgh'i'
```

第三組測資：

```
numVar = 10
minterms = [0, 1, 16, 17, 128, 343, 512, 640, 1023]
dontcares = [341]
```

輸出結果：

```
====================
Column 1
====================
v 0000000000: 0
--------------------
v 0000000001: 1
v 0000010000: 16
v 0010000000: 128
v 1000000000: 512
--------------------
v 0000010001: 17
v 1010000000: 640
--------------------
d 0101010101: 341
--------------------
v 0101010111: 343
--------------------
  1111111111: 1023
--------------------
```

```
====================
Column 2
====================
v 000000000-: 0 1
v 00000-0000: 0 16
v 00-0000000: 0 128
v -000000000: 0 512
--------------------
v 00000-0001: 1 17
v 000001000-: 16 17
v -010000000: 128 640
v 10-0000000: 512 640
--------------------
  01010101-1: 341 343
--------------------
```

```
====================
Column 3
====================
  00000-000-: 0 1 16 17
x 00000-000-: 0 1 16 17
  -0-0000000: 0 128 512 640
x -0-0000000: 0 128 512 640
--------------------
```

- [v]表示該項被使用
- [d]表示該項為 dontcare
- [-]表示該位置所對應變數為 dontcare
- [x]表示該項重複

```
====================
Prime Implicant Chart
====================
                   |   0|   1|  16|  17| 128| 343| 512| 640|1023|
-------------------+----+----+----+----+----+----+----+----+----+
b'd'e'f'g'h'i'j'    |x   |    |    |    |x   |    |x   |x   |    |
a'b'c'd'e'g'h'i'    |x   |x   |x   |x   |    |    |    |    |    |
a'b c'd e'f g'h j   |    |    |    |    |    |x   |    |    |    |
a b c d e f g h i j |    |    |    |    |    |    |    |    |x   |
-------------------+----+----+----+----+----+----+----+----+----+
```

- [x]表示包含該 minterm

```
numVar = 10
primes = b'd'e'f'g'h'i'j', a'b'c'd'e'g'h'i', a'b c'd e'f g'h j, a b c d e f g h i j
minimumSops = b'd'e'f'g'h'i'j' + a'b'c'd'e'g'h'i' + a'bc'de'fg'hj + abcdefghij
```

- Difficulties I encountered in the process

在完成前面的步驟後，第一次上傳 CASOJ 時僅有 19 分，經過了幾次的修正才成功答對所有測資。

1. 未清空向量中多餘的元素
   在輸出的合併過程中，我發現我的 terms 會出現空的元素，影響後續的動作，所以我在找出符合條件的 terms 後，先將 terms 中空的元素刪除，就解決了此問題。

2. 沒有刪除來自 dontcare 的 terms
   輸出的結果看見不符合的答案，仔細比對後發現該項完全是由 dontcare 合併而來，因此我在合併 terms 的迴圈中加入 dontcare 的合併，並在每輪刪除，將答案控制在正確範圍。

3. Solutions 答案錯誤
   起初我的 solve 函式中的迴圈起始值為 i=1，故 generateCombinations 函式所生成的 combination 大小由 1 開始，這忽略了 essential prime implicant 本身就是 minimum SOP 的情形，將起始值更正為 i=0 後，就能夠正確回答所有測資。