Logic System Assignment 1

A simple CAD tool based on K-map

Due date: `2023/04/30 23:59:59`

1. Description

- This assignment is to let students know how to automate the process taught on textbook by writing a program of K-map.
- This program should be able to handle **2~4 variables**.
- The **prime implicants** and the **essential prime implicants** of the K-map should be indicated.
- This program should also show the **minimum SOP** (Sum of Product).

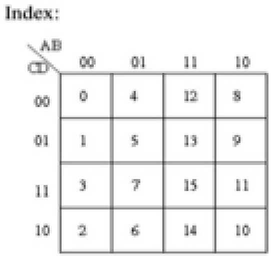
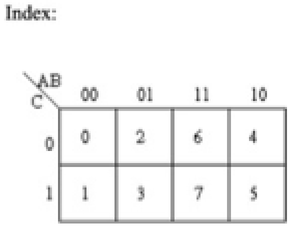
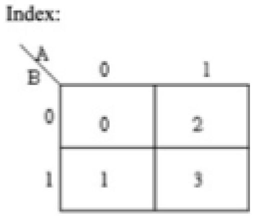
2. Requirement

1. Implement the function: `solveKmap`, in **C**, **C++** or **Java**

   The function takes in 3 parameters as input, which contains ***number of variables***, ***minterm*** and ***don't care term*** information:

   - `numVar`: Number of variables, ranging from (2~4)
   - `minterms`: Minterm value index, ranging from $(2^v - 1)$ where v = `numVar`
   - `dontcares`: Don't care term index, ranging from $(2^v - 1)$ where v = `numVar`

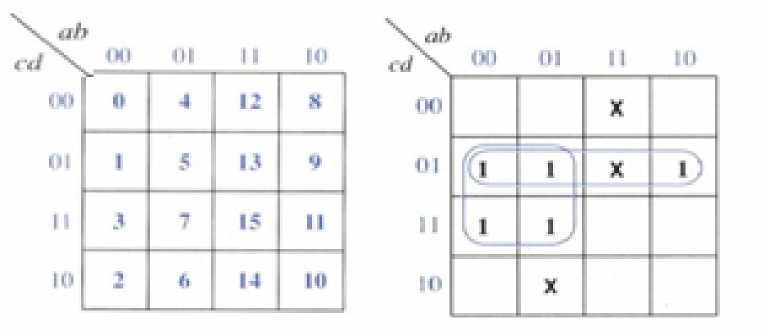
   You should first initialize the terms in the K-map, by creating a two-dimensional arrays to allocate all the 1's, 0's and X's terms of the K-map. The corresponding index of each entry in the K-map is as follow:

4-variable	3-variable	2-variable
		

e.g., the following input:

```
numVar = 4
minterms = [1, 3, 5, 7, 9]
dontcares = [6, 12, 13]
```

corresponds to:



cd \ ab	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

cd \ ab	00	01	11	10
00			X	
01	1	1	X	1
11	1	1		
10		X		

After that, you should design your own algorithm to find the **prime implicants**, **essential prime implicants**, and **minimum SOP** of the K-map (see *Hint* section below).

2. Return the solution to the K-map according to the inputs

   After your algorithm solves the K-map, the function should return the solution to it, which contains **the K-map**, **prime implicants**, **essential prime implicants**, and **minimum SOP** information:

   - `numVar`: Number of variables for this K-map (can copy from input directly)
   - `kmap`: The Karnaugh Map, 2-D array of char, use `'0'` `'1'`, or `'x'` for representation
   - `primes`: Prime implicants, array of string
   - `essentials`: Essential prime implicants, array of string
   - `minimumSop`: Minimum SOP, array of string

For more informations, please refer to *specification* section below.

3. Submit your code to CASOJ

   After you finish implementing the function, you need to submit your code to CASOJ, an online judge server, to check the correctness of your program. (See CASOJ tutorial)

   We prepared several test cases (including boundary conditions, corner cases) to test your program. You should try your best to pass as many test cases as possible.

   You can write your `main` function locally for debugging, but need not submit to CASOJ (otherwise it may cause compile errors).

   Note: The grading of your assignment is heavily based on your score (pass rate) on CASOJ.

4. Write a document report with some explanations, e.g., execution flow or tricks in your program

5. Upload this assignment to course **Moodle**

   - **Files to submit**:

     a. `Kmap.[c, cpp, java]`: single file, whose content must be exactly the same as what you submitted to CASOJ (otherwise no score for your program)

     b. `report.pdf`

   - All files should be put in a directory named `StudentID_HW1` and **compress** it into `.tar` or `.zip`, e.g., `E24112345_HW1.zip`

     ```
     E24112345_HW1
     ├── Kmap.c
     └── report.pdf

     1 directory, 2 files
     ```

3. Grading

- **Program (80%)**

  - **Basic function (70%)**

    Function correctness would be tested by CASOJ with several test cases.

    The final score would be your score on CASOJ.

  - **Coding style and comments (10%)**

    Please write human-readable code.

- **Document Report (20%)**

  - **Code explanation**

    Please explain how your algorithm works.

  - **Some test results**

    Show some examples to verify your program.

- ***Bonus (10%)***

  You can get extra 10% if your program runs faster than TA's (You have to pass all test cases first).

4. Specification

This assignment can only be finished in **C**, **C++**, or **Java**. Please choose one of these programming languages and see the corresponding specification below.

C

1. Function prototype and return structure

```c
struct KmapSolution {
    int numVar;
    char **kmap;
    char **primes;
    int primesSize;
    char **essentials;
    int essentialsSize;
    char **minimumSop;
    int minimumSopSize;
};

struct KmapSolution *solveKmap(
    int numVar, int *minterms, int mintermsSize,
    int *dontcares, int dontcaresSize)
{
    struct KmapSolution sol = malloc(sizeof(KmapSolution));

    // implement your solution here

    return sol;
}
```

2. Variable descriptions

Input

- `int numVar` : Integer, number of variables
- `int *minterms` : Array of int, minterm value index
- `int mintermsSize` : Size of `minterms` array
- `int *dontcares` : Array of int, don't care term index
- `int dontcaresSize` : Size of `dontcares` array

e.g. $F(a,b,c,d) = \sum m(2,5,7,10,11,14) + \sum d(3,10,15)$

```
numVar = 4
minterms = [2, 5, 7, 10, 11, 14]
mintermsSize = 6
dontcares = [3, 10, 15]
dontcaresSize = 3
```

Output

   - `int numVar`: Integer, number of variables
   - `char **kmap`: 2-D array of char, the K-map, represented by `'0'`, `'1'`, or `'x'`
   - `char **primes`: Array of string, prime implicants
   - `int primesSize`: Size of `primes` array
   - `char **essentials`: Array of string, essential prime implicants
   - `int essentialsSize`: Size of `essentials` array
   - `char **minimumSop`: Array of string, minimum SOP
   - `int minimumSopSize`: Size of `minimumSop` array

     e.g. solution to the input above:

     ```
     numVar = 4
     kmap = [['0', '0', '0', '0'],
             ['0', '1', '1', '0'],
             ['x', '1', 'x', '1'],
             ['1', '0', '1', 'x']]
     primes = ["ac", "bd", "b'c", "cd"]
     primesSize = 4
     essentials = ["ac", "bd", "b'c"]
     essentialsSize = 3
     minimumSop = ["ac", "bd", "b'c"]
     minimumSopSize = 3
     ```

3. Notes
   - You can assume function inputs are error-free.
   - Returned `struct` needs to be malloced.
   - For every array in C, you need to specify its size (e.g., `primes` -> `primesSize`).
   - Every array needs to be malloced before returning (assume caller calls `free()`).
   - The order of your answer doesn't matter (e.g., `["ab", "bc"]` and `["bc, "ab"]` are both OK).
   - Your returned `struct KmapSolution *` should be printed by the following function without error:

```c
void printKmapSolution(struct KmapSolution *sol)
{
    int nRow, nCol;
    switch (sol->numVar) {
    case 2:
        nRow = 2; nCol = 2;
        break;
    case 3:
        nRow = 2; nCol = 4;
        break;
    case 4:
        nRow = 4; nCol = 4;
        break;
    }

    printf("K-map:\n");
    for (int i = 0; i < nRow; ++i) {
        for (int j = 0; j < nCol; ++j)
            printf("%c ", sol->kmap[i][j]);
        printf("\n");
    }

    printf("Prime implicants: ");
    for (int i = 0; i < sol->primesSize; ++i) {
        printf("%s", sol->primes[i]);
        if (i != sol->primesSize - 1) printf(", ");
    }
    printf("\n");

    printf("Essential prime implicants: ");
    for (int i = 0; i < sol->essentialsSize; ++i) {
        printf("%s", sol->essentials[i]);
        if (i != sol->essentialsSize - 1) printf(", ");
    }
    printf("\n");

    printf("Minimum SOP: ");
    for (int i = 0; i < sol->minimumSopSize; ++i) {
        printf("%s", sol->minimumSop[i]);
        if (i != sol->minimumSopSize - 1) printf(" + ");
    }
    printf("\n");
}
```

And the result should look like:

```
K-map:
0 0 0 0
0 1 1 0
x 1 x 1
1 0 1 1
Prime implicants: ac, bd, b'c, cd
Essential prime implicants: ac, bd, b'c
Minimum SOP: ac + bd + b'c
```

C++

1. Function prototype and return structure

```cpp
#include <vector>
#include <string>

using namespace std;

struct KmapSolution {
    int numVar;
    char **kmap;
    vector<string> primes;
    vector<string> essentials;
    vector<string> minimumSop;
};

KmapSolution *solveKmap(int numVar, const vector<int> minterms,
                        const vector<int> dontcares)
{
    KmapSolution *sol = new KmapSolution;

    // implement your solution here

    return sol;
}
```

2. Variable descriptions

Input

- `int numVar`: Integer, number of variables
- `vector<int> minterms`: Vector of int, minterm value index
- `vector<int> dontcares`: Vector of int, don't care term index

e.g. $F(a, b, c, d) = \sum m(2, 5, 7, 10, 11, 14) + \sum d(3, 10, 15)$

```
numVar = 4
minterms = [2, 5, 7, 10, 11, 14]
dontcares = [3, 10, 15]
```

Output

- `int numVar`: Integer, number of variables
- `char **kmap`: 2-D array of char, the K-map, represented by `'0'`, `'1'`, or `'x'`
- `vector<string> primes`: Vector of string, prime implicants
- `vector<string> essentials`: Vector of string, essential prime implicants
- `vector<string> minimumSop`: Vector of string, minimum SOP

  e.g. solution to the input above:

```
numVar = 4
kmap = [['0', '0', '0', '0'],
        ['0', '1', '1', '0'],
        ['x', '1', 'x', '1'],
        ['1', '0', '1', 'x']]
primes = ["ac", "bd", "b'c", "cd"]
essentials = ["ac", "bd", "b'c"]
minimumSop = ["ac", "bd", "b'c"]
```

3. Notes

- You can assume function inputs are error-free.
- Returned `struct` needs to be allocated with `new`.
- The order of your answer doesn't matter (e.g., `["ab", "bc"]` and `["bc, "ab"]` are both OK).
- Your returned `struct KmapSolution *` should be printed by the following function without error:

```cpp
void printKmapSolution(KmapSolution *sol)
{
    int nRow, nCol;
    switch (sol->numVar) {
    case 2:
        nRow = 2; nCol = 2;
        break;
    case 3:
        nRow = 2; nCol = 4;
        break;
    case 4:
        nRow = 4; nCol = 4;
        break;
    }

    cout << "K-map:\n";
    for (int i = 0; i < nRow; ++i) {
        for (int j = 0; j < nCol; ++j)
            cout << sol->kmap[i][j] << " ";
        cout << endl;
    }

    cout << "Prime implicants: ";
    for (int i = 0; i < sol->primes.size(); ++i) {
        cout << sol->primes[i];
        if (i != sol->primes.size() - 1) cout << ", ";
    }
    cout << endl;

    cout << "Essential prime implicants: ";
    for (int i = 0; i < sol->essentials.size(); ++i) {
        cout << sol->essentials[i];
        if (i != sol->essentials.size() - 1) cout << ", ";
    }
    cout << endl;

    cout << "Minimum SOP: ";
    for (int i = 0; i < sol->minimumSop.size(); ++i) {
        cout << sol->minimumSop[i];
        if (i != sol->minimumSop.size() - 1) cout << " + ";
    }
    cout << endl;
}
```

And the result should look like:

```
K-map:
0 0 0 0
0 1 1 0
x 1 x 1
1 0 1 1
Prime implicants: ac, bd, b'c, cd
Essential prime implicants: ac, bd, b'c
Minimum SOP: ac + bd + b'c
```

Java

1. Function prototype and return structure

```java
import java.util.ArrayList;

class Kmap {

  static class Solution {
    public int numVar;
    public char[][] kmap;
    public ArrayList<String> primes;
    public ArrayList<String> essentials;
    public ArrayList<String> minimumSop;
  }

  public static Solution solveKmap(
    int numVar, ArrayList<Integer> minterms,
    ArrayList<Integer> dontcares) {
    Solution sol = new Solution();

    // implement your solution here

    return sol;
  }
}
```

2. Variable descriptions

Input

- `int numVar`: Integer, number of variables
- `ArrayList<Integer> minterms`: ArrayList of integer, minterm value index
- `ArrayList<Integer> dontcares`: ArrayList of integer, don't care term index

e.g. $F(a,b,c,d) = \sum m(2,5,7,10,11,14) + \sum d(3,10,15)$

```
numVar = 4
minterms = [2, 5, 7, 10, 11, 14]
dontcares = [3, 10, 15]
```

Output

- `int numVar`: Integer, number of variables
- `char[][] kmap`: 2-D array of char, the K-map, represented by `'0'`, `'1'`, or `'x'`
- `ArrayList<String> primes`: ArrayList of String, prime implicants
- `ArrayList essentials`: ArrayList of String, essential prime implicants
- `ArrayList minimumSop`: ArrayList of String, minimum SOP

  e.g. solution to the input above:

```
numVar = 4
kmap = [['0', '0', '0', '0'],
        ['0', '1', '1', '0'],
        ['x', '1', 'x', '1'],
        ['1', '0', '1', 'x']]
primes = ["ac", "bd", "b'c", "cd"]
essentials = ["ac", "bd", "b'c"]
minimumSop = ["ac", "bd", "b'c"]
```

3. Notes

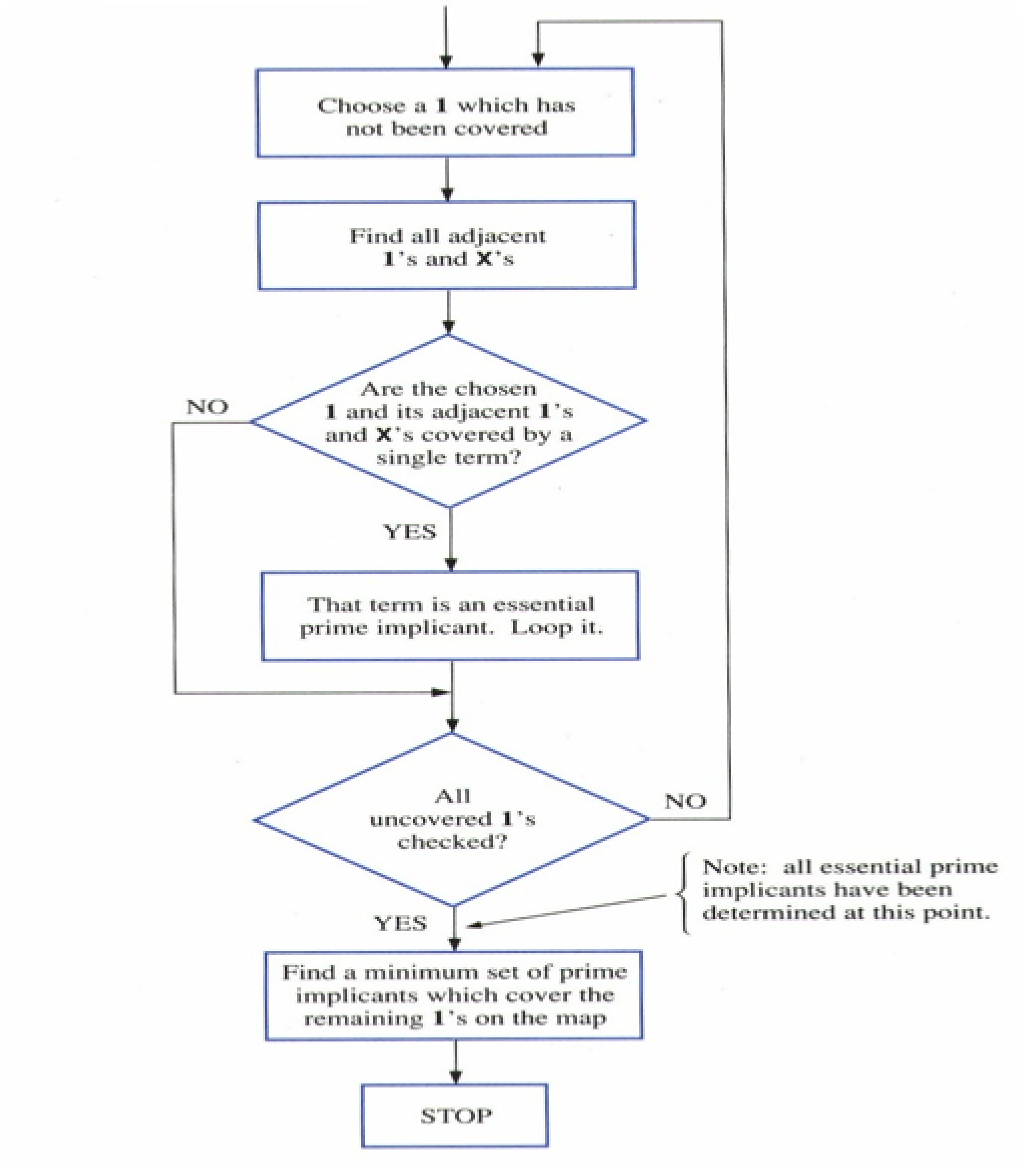
- You can assume function inputs are error-free.
- The order of your answer doesn't matter (e.g., `["ab", "bc"]` and `["bc, "ab"]` are both OK).
- Your returned `Solution` should be printed by the following function without error:

```java
public static void printKmapSolution(Kmap.Solution sol) {
  int nRow = 0, nCol = 0;
  switch (sol.numVar) {
    case 2:
      nRow = 2; nCol = 2;
      break;
    case 3:
      nRow = 2; nCol = 4;
      break;
    case 4:
      nRow = 4; nCol = 4;
      break;
  }

  System.out.println("K-map:");
  for (int i = 0; i < nRow; ++i) {
    for (int j = 0; j < nCol; ++j)
      System.out.printf("%c ", sol.kmap[i][j]);
    System.out.println();
  }

  System.out.print("Prime implicants: ");
  for (int i = 0; i < sol.primes.size(); ++i) {
    System.out.print(sol.primes.get(i));
    if (i != sol.primes.size() - 1) System.out.print(", ");
  }
  System.out.println();

  System.out.print("Essential prime implicants: ");
  for (int i = 0; i < sol.essentials.size(); ++i) {
    System.out.print(sol.essentials.get(i));
    if (i != sol.essentials.size() - 1) System.out.print(", ");
  }
  System.out.println();

  System.out.print("Minimum SOP: ");
  for (int i = 0; i < sol.minimumSop.size(); ++i) {
    System.out.print(sol.minimumSop.get(i));
    if (i != sol.minimumSop.size() - 1) System.out.print(" + ");
  }
  System.out.println();
}
```

And the result should look like:

```
K-map:
0 0 0 0
0 1 1 0
x 1 x 1
1 0 1 1
Prime implicants: ac, bd, b'c, cd
Essential prime implicants: ac, bd, b'c
Minimum SOP: ac + bd + b'c
```

5. Hint (lecture5 p.26)


Choose a 1 which has not been covered
Find all adjacent 1's and X's
Are the chosen 1 and its adjacent 1's and X's covered by a single term?
NO
YES
That term is an essential prime implicant. Loop it.
All uncovered 1's checked?
NO
YES
Note: all essential prime implicants have been determined at this point.
Find a minimum set of prime implicants which cover the remaining 1's on the map
STOP

6. Contact information

Email: course@caslab.ee.ncku.edu.tw