CAD for VLSI Design
Programming Assignment 3 Report

Student:黃柏燁
Student ID:109521018

I. Introduction

本次作業主要是要求使用 C/C++實作出 Quine-McCluskey and Petrick Method Algorithm 做出邏輯化簡的程式。

II. Data Structure

在 Quine-McCluskey，使用 class 創建 Linked-list 作為主要的資料結構。
在 Petrick Method，使用 2D vector 作為主要的資料結構。

III. Algorithms

題目規定必須使用 Quine-McCluskey and Petrick Method。

- Quine-McCluskey:找出所有可簡化的 Minterms，即是 Prime Implicants。

  Example:

  欲化簡 Boolean function 為下列式子:

  $$f_{A,B,C,D} = A'BC'D' + AB'C'D' + AB'CD' + AB'CD + ABC'D' + ABCD$$

  首先先列出所有 Minterms 以及用二進位表示中有幾個 1 來分類:

1的數目	極小項	二進位表示
1	m4	0100
	m8	1000
2	m9	1001
	m10	1010
	m12	1100
3	m11	1011
	m14	1110
4	m15	1111

  可以開始把 Minterm 同其他 Minterm 組合在一起。如果兩個項只有一

個位元的數值不同，則可以這個位的數值可以替代為一個橫槓，來指示這個數字無關緊要。
不再組合的項標記上 **"*"**。

1的數目	極小項	二進位表示	大小為2的蘊涵項	大小為4的蘊涵項
1	m4	0100	m(4,12) -100*	m(8,9,10,11) 10--*
	m8	1000	m(8,9) 100-	m(8,10,12,14) 1--0*
	--	--	m(8,10) 10-0	--
	--	--	m(8,12) 1-00	--
2	m9	1001	m(9,11) 10-1	m(10,11,14,15) 1-1-*
	m10	1010	m(10,11) 101-	--
	--	--	m(10,14) 1-10	--
	m12	1100	m(12,14) 11-0	--
3	m11	1011	m(11,15) 1-11	--
	m14	1110	m(14,15) 111-	--
4	m15	1111	--	--

- Petrick Method: 將找到的所有 Prime Impicants 列成一個項目表，去尋找出最少所需 Prime Impicants 可以達到跟簡化前功能一樣。
  Example:
  接續前面得到的 Prime Implicants，我們可以列出以下表格:

	4	8	10	11	12	15		=>	A	B	C	D
m(4,12)*	X				X		-100	=>	-	1	0	0
m(8,9,10,11)		X	X	X			10--	=>	1	0	-	-
m(8,10,12,14)		X	X		X		1--0	=>	1	-	-	0
m(10,11,14,15)*			X	X		X	1-1-	=>	1	-	1	-

第二個 Prime Impicants 能被第三個和第四個所覆蓋，而第三個 Prime Impicants 能被第二個和第一個所覆蓋，因此都不是唯一能 cover 的。如果一個 Prime Impicants 是唯一能 cover 的，它必須包含在最小化的 Boolean function。在某些情況下，唯一能 cover 的 Prime Impicants 不能 cover 所有的 Minterm，此時可以組合這兩個唯一能 cover 的 Prime Impicants 與剩下所有 Prime Impicants 中的一個而生成:

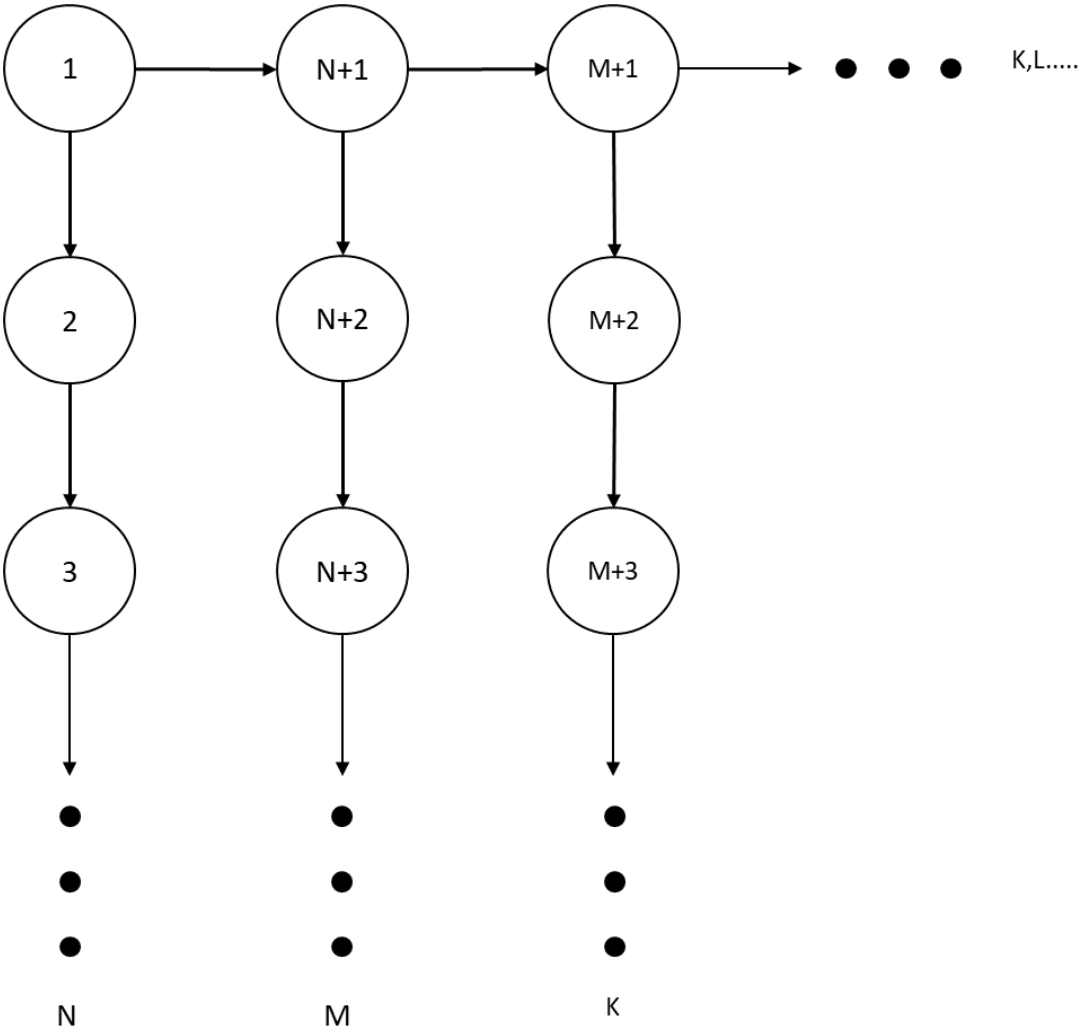
$$f_{A,B,C,D} = BC'D' + AB' + AC$$

$$f_{A,B,C,D} = BC'D' + AD' + AC$$

最終的等式在功能上等價於最初的（冗長）等式。

IV. Implement

以下為我的 Linked-list 的樣子:



利用兩個 pointer，可以模擬出 Quien-McCluskey 在比較時的表格，不同 minterm 互相比較的 pointer 是往下，當比較完得到化簡後的 minterm 則往右邊 push，整個資料結構是動態成長的，因為在比較的階段，我們並無法保證要比較幾次才可以全部比較完，若是開固定 size 的 2D array，則有可能會空間浪費的問題，亦或是當處理的資料量龐大時，可能導致空間不夠用的問題，因此我選擇動態空間來存取，而不是一般的 array。

V. Makefile

```
# Compiler
CXX = g++

# Path
SRC_PATH = .
BUILD_PATH = build

# Executable
EXE = go

# Source
SOURCES = $(SRC_PATH)/PA3_109521018.cpp  $(SRC_PATH)/linkedlist.cpp
OBJECTS = $(BUILD_PATH)/PA3_109521018.o $(BUILD_PATH)/linkedlist.o

# Compiler flags
CXXFLAG = -O3 -Wall
INCLUDE = -I$(SRC_PATH)

# Make-command list
.PHONY: all run clean

# Target: Dependencies
#       Shell-command
all: $(BUILD_PATH) $(EXE)


run: $(EXE)
        ./go $(INPUT) $(OUTPUT)

clean:
        @echo "Removing objects"
        @rm -rf $(BUILD_PATH)
        @echo "Removing executable file"
        @rm -rf $(EXE)

$(EXE): $(OBJECTS)
        @echo "Generating executable file: $^ -> $@"
        @$(CXX) $^ -o $@

$(BUILD_PATH)/%.o: $(SRC_PATH)/%.cpp
        @echo "Compiling: $< -> $@"
        @$(CXX) -std=c++11 $(CXXFLAGS) $(INCLUDE) -c $< -o $@

$(BUILD_PATH):

        @echo "Creating object directory"
        @mkdir -p $@

~
```

VI. Hardest part of the assignment

實現 algorithm 時，雖然這次的概念不難，但是要將表格時現在 code 中，我在實現的過程中遇到不少困難，尤其這次改用 class 創建，對於不是很熟悉使用 class 的我，在思考如何利用 class function 和 constuctor 控制及操控，更用說創建一個 linked-list，加入指標後更讓我頭痛許久，真的讓我花了不少時間。