

Quine McClusky Algorithm

2020202031 김재현

CONTENTS

- A. Problem statement
- B. Pseudo code, flow chart
- C. Verification strategy &
corresponding examples with explanation
- D. Hard testcase

A. Problem statement

1) 접근

사용자로부터 첫 줄에 비트의 길이를, 두번째 줄부터는 한 줄 당 하나의 변수를 입력 받는다. 각 변수는 don't care minterms 인지 true minterms 인지에 대해 판별하도록 첫번째 인덱스에 'd' or 'm'이 입력되고, 공백 다음에 비트가 입력된다. 입력 받은 변수들을 Quine-McCluskey algorithm 을 통해 최소화시킨 Boolean expressions 와 이를 구현하는데 필요한 트랜지스터의 개수를 출력한다.

다음은 Quine-McCluskey algorithm 에 대한 실행방법이다.

- ① 사용자로부터 입력을 받는다.
- ② don't care minterms 과 true minterms 를 1 의 개수에 따라 그룹 지어 나열한다.
- ③ 인접한 두 그룹에서 hamming distance 가 1 인 두 minterms 를 찾고, 다른 비트 부분을 '-'로 대체한 결과를 다음 열에 위치시킨다.
- ④ 두 minterms 에 체크표시를 해둔다.
- ⑤ hamming distance 가 1 인 minterm 이 없는 minterms 들은 '*'표시를 해둔다.
- ⑥ 마지막 남은 minterms 가 없거나 더 이상 결합될 수 없을 때까지 이를 반복한다.
- ⑦ '*'표시된 minterms(prime implicants)를 PI table 의 각 행에, 처음 입력 받은 true minterms 를 각 열에 적는다.
- ⑧ PI 가 true minterms 를 포함하는 곳에 'X'표시를 한다.
- ⑨ 각 열을 비교하며 'X'가 하나만 있는 열에 해당하는 PI 를 essential PIs 로 분류하고, 해당하는 행과 열을 제거한다.
- ⑩ 위의 과정들을 거친 후, 남은 열을 모두 포함할 수 있는 최소한의 PIs 를 찾아낸다.
- ⑪ 찾아낸 PIs 결과값 식을 사용해서 트랜지스터의 개수를 계산한다.
- ⑫ 결과값 항들과 트랜지스터 개수를 파일에 출력한다.

2) 프로그램 구현

디지털논리회로 시간에 배운 Quine-McCluskey algorithm 의 여러 step 중 step2~3에서는 hamming distance 가 1 차이나는 것들을 결합하여 새로운 column 들을 계속해서 구성한다. Linked list 를 활용하여 요소들을 저장한다. 노드는 생성됨과 동시에 'd' or 'm'과 '*'표시를 가지고 생성된다. hamming distance 가 1 인 요소들이 가지는 서로 다른 비트에 '-'를 저장하고 결합하여 다음 column 의 요소들을 구성하고 새로운 Linked list 에 저장한다. 이런 식으로 각 column 들의 요소들을 각 list 에 저장하고, 이를 1 의 개수 순으로 정렬하고 중복되는 것들은 삭제한다. 또한 결합하는 과정에서 사용된 요소들은 'V'표시를 한다.

모든 list 들을 탐색하며 '*'표시를 가진 요소들은 Pls 로 판단하고 따로 PI 라는 배열에 저장합니다. 첫번째 list 에서 'm'표시를 가진 요소들은 true minterms 라고 판단하여 minterms 라는 배열에 저장합니다. PI 와 minterms 배열의 길이에 맞게 PI table 을 이차원 배열로 구성하고 공백으로 초기화한 후, Pls 가 커버할 수 있는 true minterms 가 존재하는 적절한 위치에 'X'를 저장한다.

table 의 각 열을 비교하며 'X' 표시가 하나인 경우 해당 true minterm 를 커버하는 Pls 는 essential PI 로 판단하고 해당 PI 를 새로운 list 에 저장한다. Pls 가 속한 행 중, 'X'표시가 된 부분의 true minterms 는 이미 Pls 가 커버하고 있으므로 같은 열에 있는 모든 'X'를 지운다. Pls 와 true minterms 에 사용되었음을 나타내는 '@'표시를 한다.

다음 탐색부터는 Pls 중 해당하는 행에 가장 많은 'X'가 포함된 PI 를 상대로 이전 문단의 과정을 반복하면 list 에 minimum set of Pls 만이 저장되고, 이를 통해 트랜지스터의 개수를 구할 수 있다. minimum set of Pls 의 개수가 곧 OR gate 의 입력 개수이므로 2 개 이상이면 트랜지스터의 개수를 $2n+2$ 개 더해준다. minimum set of Pls 의 각 PI 에서 '-'를 제외한 비트의 수가 곧 AND gate 의 입력 개수이므로 2 개 이상이면 $2n+2$ 만큼 트랜지스터의 개수를 더해준다. 모든 minimum set of Pls 에서 0 인 입력이 있는 부분은 인버터를 필요로 하기 때문에 트랜지스터를 $2n$ 개 더해준다.

마지막으로 list 와 트랜지스트 개수를 파일에 출력해준다.

B. Problem statement

1) Pseudo code

(val.cpp)

val_manager::isEmpty :

if(list is empty) return 1;

else return 0;

val_manager::setData :

if(list is empty)

add node and head, tail point to node;

else

add node and tail point to node;

val_manager::deleteSame :

current = list 의 head;

while(current 가 마지막 노드를 가리키기 전까지){

while(current 와 target 의 1 의 개수가 같을 때만){

if(current and target are same)

target 을 delete 및 이전 노드가 다음 노드를 가리키게 함;

target 이 다음 노드 참조;

current 가 다음 노드 참조;

}

val_manager::sortBy1

// 각 노드들에 저장되어 있는 비트 '1'의 개수를 비교하며 BubbleSort 진행

for (i = 노드의 개수 - 1; i > 0; i--){

for (j = 처음 노드; j < i; j++){

if(j 번째 노드 1 의 개수 > j+1 번째 노드 1 의 개수)

두 노드의 위치 switching;

deleteSame 실행;

val_manager::combining :

current = list 의 head;

while(current 가 마지막 노드를 가리키기 전까지){

```

        if(current 와 target 의 서로 다른 비트가 1 개라면){
            그 비트에 '-'을 저장한 문자열을 새로운 문자열에 저장하고, current 와
            target 에 'V' 표시;
        }
        current 가 다음 노드 참조;
    }
}

```

```

val_manager::printall :
current = list 의 head;
while(current 가 마지막 노드를 가리킬 때까지){
    current 의 str 을 출력;
    current 가 다음 노드 참조;
}

```

(main.cpp)

```

open file for reading;
비트 수 읽기 -> num;
Linked list 를 관리하는 포인터배열 arr 할당;
파일 내 true minterms 와 don't care minterms 를 첫번째 list 에 저장;
첫번째 list sortBy1 으로 정렬;
for(i = 1; i<=num; i++)
    i-1 번째 list 를 combining 으로 결합하여 i 번째 list 에 저장;
width = PI table 의 열 개수;
while (첫번째 list 의 마지막 노드까지) {
    if (current 가 true minterm 일 때)
        width 1 증가;
    list 의 다음 노드 참조;
}
height = PI table 행 개수;
height x width 크기의 이차원 배열 table 생성 및 공백으로 초기화;
PIs 를 저장할 height 크기의 포인터 배열 PI 생성;
while(list 가 비어 있지 않을 때){
    current = list 의 head;
    while(current 가 존재할 때){

```

```

        '*' 표기된 노드들의 문자열을 PI 에 저장;
        current 가 다음 노드 참조;
    }
    다음 list 를 참조;
}
minterms 를 저장할 width 크기의 포인터 배열 minterm 생성;
current = 첫번째 list 의 head;
while(current 가 존재할 때){
    'm' 표기된 노드들의 문자열을 minterm 에 저장;
    current 가 다음 노드 참조;
}
for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
        if(PI[i]가 minterm[j]를 커버한다면)
            table[i][j]에 'X' 저장
    }
}
minimum set of PIs 를 저장할 리스트 minPIs 생성;
for (int j = 0; j < width; j++) {
    xNum = 0 //한 열에서의 x 개수;
    for (int i = 0; i < height; i++) {
        if (table[i][j]에 'X'표시가 돼있으면){
            xNum 1 증가;
            idx = i; // 'X'의 행 인덱스 저장
        }
    }
    if (xNum 0이 1 이면) {
        minPIs 에 해당 PI 를 저장하고 해당 PI 에 '@' 표시;
        while(해당 행에 있는 모든 'X'에 대하여){
            if (그 열에 'X'가 있으면){
                'X'표시 있는 해당 열의 minterm 에 '@'표시;
                그 열에 있는 'X' 전부 삭제;
            }
        }
    }
}
}
}

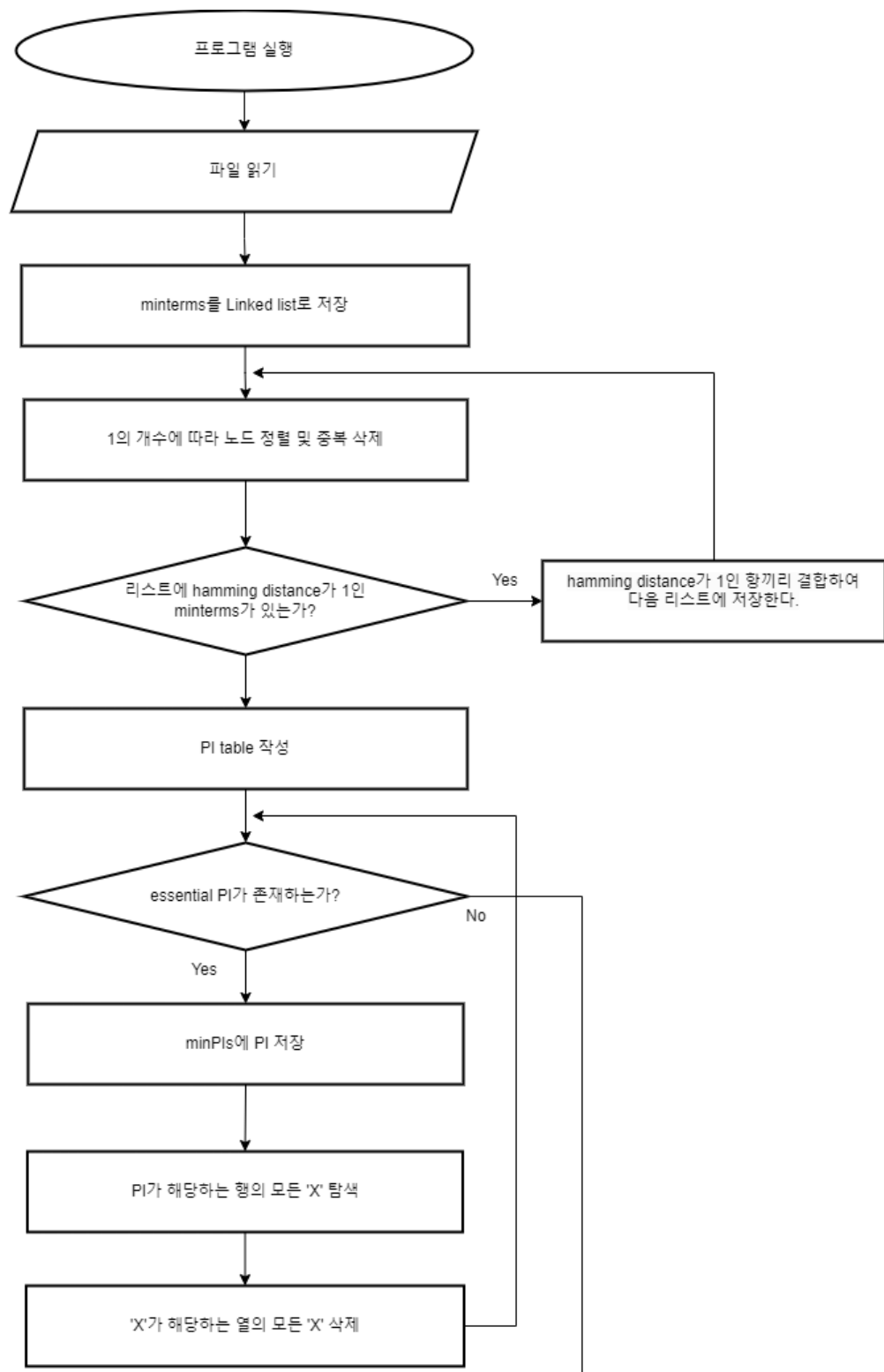
```

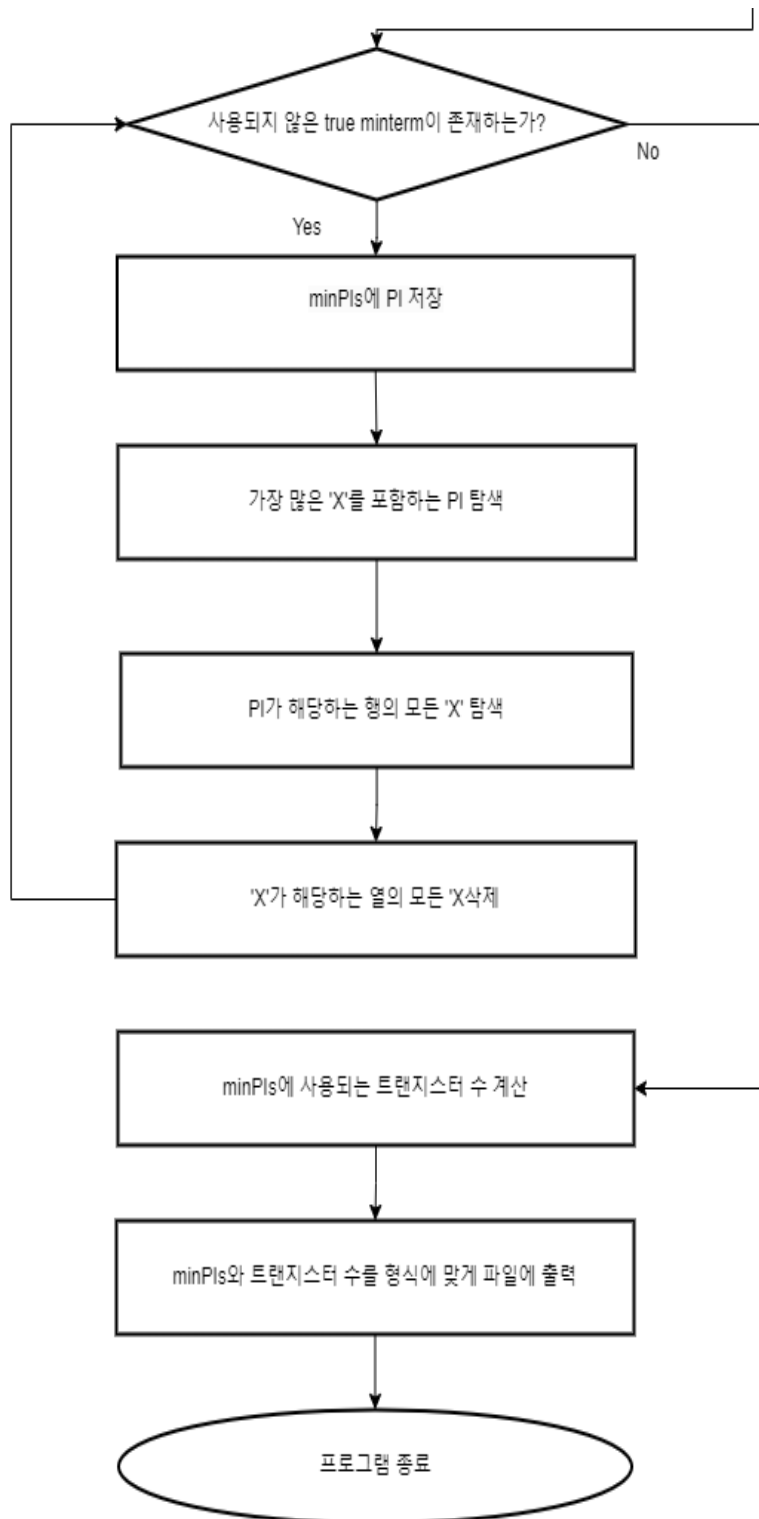
```
for(k=0; k<height; k++){  
    가장 많은 'X'를 포함하는 행을 탐색;  
    해당 PI 를 minPIs 에 저장;  
    해당 행에 있는 'X'를 포함하는 열들에 있는 'X'를 모두 삭제;  
}
```

```
open file for writing;  
minPIs 를 sortBy1 으로 정렬;  
minPIs 의 요소들을 개행으로 구분하여 파일에 출력;  
minPIs 에 저장된 최소 후보항을 통해 트랜지스터 개수 계산;  
형식에 맞게 파일에 출력;
```

```
close file for both reading and writing;  
deallocate all allocated memory;  
프로그램 종료
```


1) Flow chart

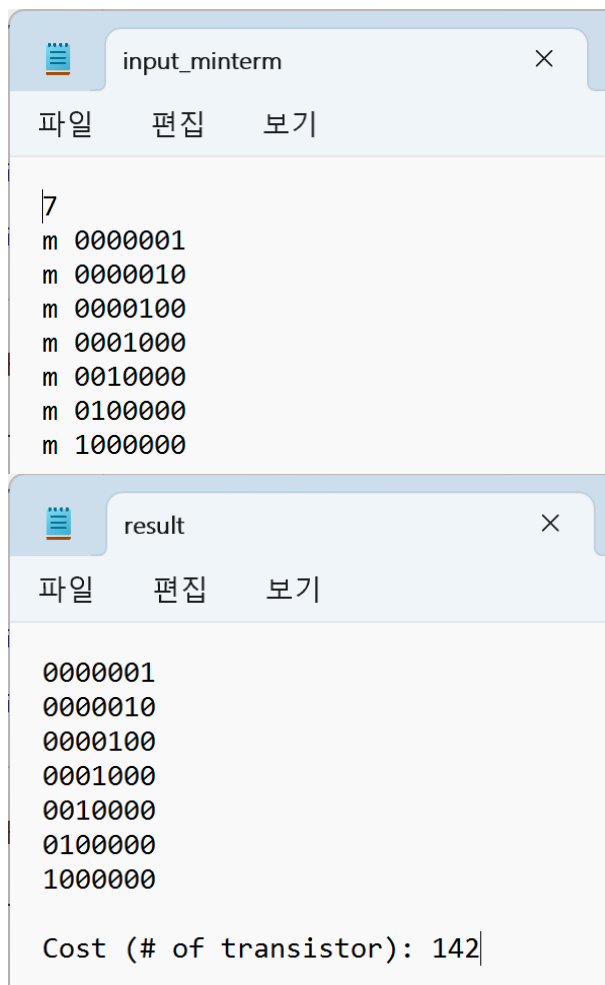




C. Verification strategy &

corresponding examples with explanation

1) testcase 1 (파일 명: input1)



The image shows two overlapping windows from a software application. The top window, titled 'input_minterm', contains a list of minterms for a 7-bit input, each preceded by 'm'. The bottom window, titled 'result', shows the same list of minterms without the 'm' prefix, followed by a line indicating the total cost in terms of transistors.

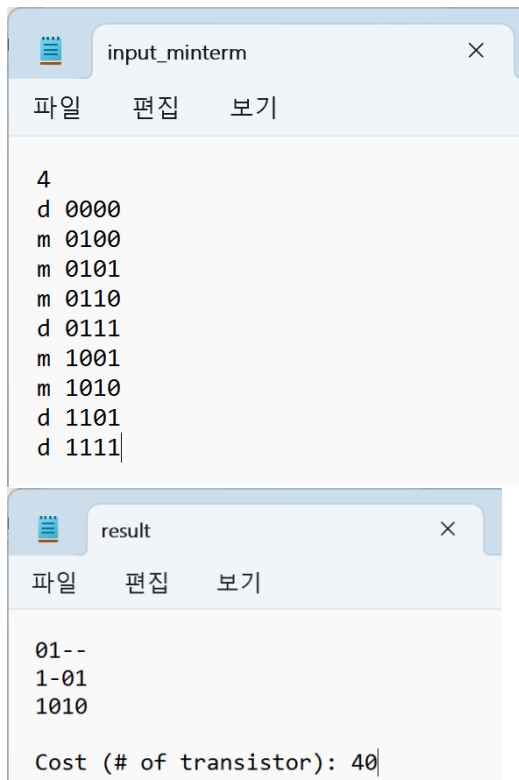
```
input_minterm
파일 편집 보기
7
m 0000001
m 0000010
m 0000100
m 0001000
m 0010000
m 0010000
m 0100000
m 1000000

result
파일 편집 보기
0000001
0000010
0000100
0001000
0010000
0010000
0100000
1000000

Cost (# of transistor): 142
```

위 케이스는 hamming distance 가 1 이 아닌 minterms 만을 입력한 것으로, 모든 minterms 가 essential PI 가 된다. 따라서 입력된 모든 minterms 가 minimum set of PIs 가 된다.

2) testcase2 (파일 명: input2)



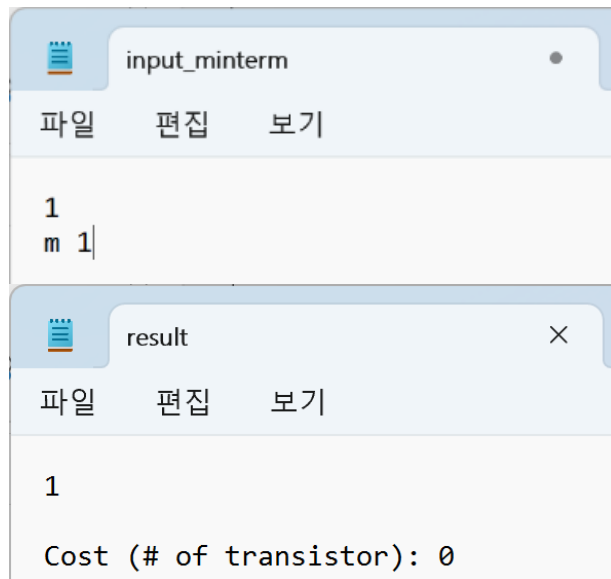
```
input_minterm
4
d 0000
m 0100
m 0101
m 0110
d 0111
m 1001
m 1010
d 1101
d 1111

result
01--
1-01
1010

Cost (# of transistor): 40
```

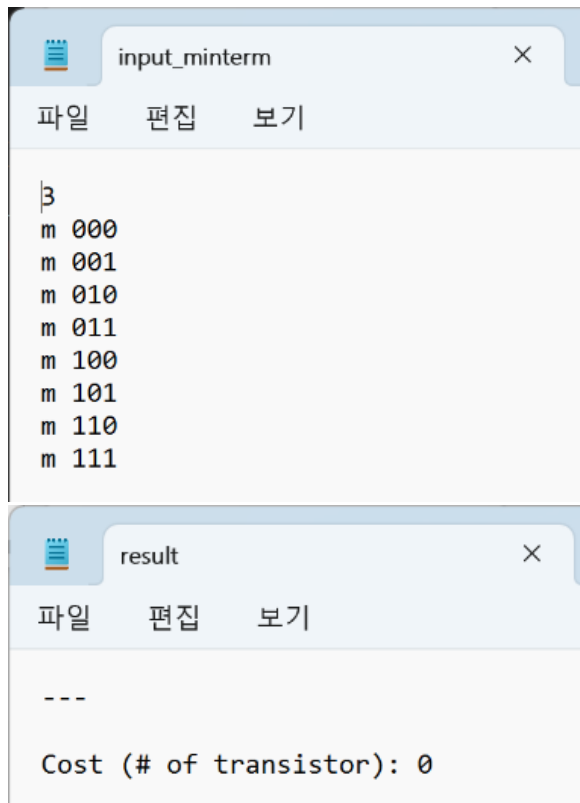
디지털논리회로 수업 자료에 있는 예시로, essential PI 가 곧 minimum set of PIs 가 되는 예제다.

3) testcase 3 (파일 명: input3)



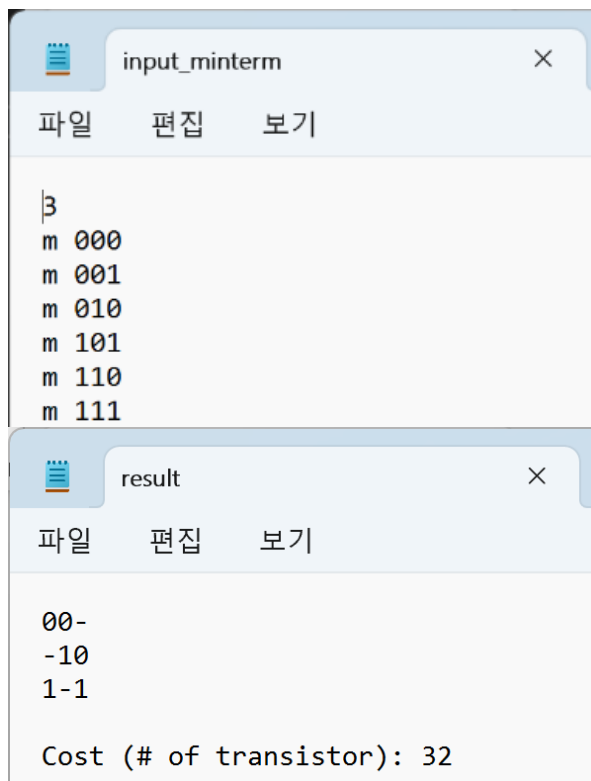
비트 길이가 1 이면 AND gate 에 들어갈 입력이 한 개이므로 AND gate 가 사용되지 않고, minimum set of PIs 또한 1 한 개이므로 OR gate 도 사용되지 않으며, Inverter 도 사용되지 않아 트랜지스터가 총 0 개 소비된다.

4) testcase 4 (파일 명: input4)



가능한 모든 경우의 수를 true minterm 으로 입력한 상황이다. 모든 경우가 다 참이다. 따라서 minimum set of Pls 는 ---이고 소비되는 트랜지스터 또한 0 이다.

5) testcase5 (파일 명: input5)



위의 예제는 essential PIs 가 존재하지 않는 케이스다. essential PIs 가 존재하지 않으므로 가장 많은 'X'를 커버하는 PI 를 탐색하는 단계로 넘어가 minimum set of PIs 를 구한다.

6) testcase6 (파일 명: input6)

input_minterm
×

파일
편집
보기

```

40
m5245867176582469191105777852910287846686
m8413961100339325340987556427841095587495
d9197768236478759903100222908567465530516
d7987191340924082506437018196446734775744
m8230034960843716533391389685696254496255
m9769480555639976084027569691216731903929
d5989207903315419529086380073874926033664
d8861450383933014887920760395431899903079
d2669596838857608822640138070792841616103
d3567350629874210050131406029808199430341
d6888013031959415097751356260697963748589
m0598305249407396617314680970242863428978
m4347842686157639039025761131588667898614
m0974859549717389270982060758939062848315
m0503126110456175115328240005696376096819
m253615407033576024644575076393881877813
m5070443504850697933246311742031709475275
m0874633502905170228345380086420610984379
m8753484995573869382598075175078573552464
m3313361747176679306889658232383929748530
m6554616047147141802416744170729633783959
m0095368456761803194329582953329753782761
d6058927015651443730662382506084512707395
d1044022605862809862275902195143803781138
d5207951930841956371474510839264720903392
m467066628041628265939035324427237674085
m5382383288901299872340120788296618246250
m4803261080786405540342812030451549850535
m1941845468339407592736310347145289714898
m7743834111247900039515890707401123399959
m2399840534674430918526636692803643802295
m1068241591284351762369185812374615773978
m5495605219403819581081032285150046885668
m0900430451299525695977548959359241947977
m3611331610935227222903497096571891985133
m2121452541660659549207590882986145941201
d2992733757932509840374321379577946437534
d0602107132619621276474864246847847421649
d9106344237719301467384216864378813780028
m2925858880391508071639613645839810245652
d4511620001324758780033484074264115969822
m0781360221029297154343943848477183708449
d0858291402045690699199853941152672200983
d9208535900362446556949629260321255397488
d1570344196633953271918098991785213486289
m4513571253648542337626112278239660032562
m748779639735067294977733306765162292724
m3346047444685469340506779033221721012561
m7615831862916403474288562083455163862259
m3177196673006756637962919112932969590043

```

result
×
+

파일
편집
보기

```

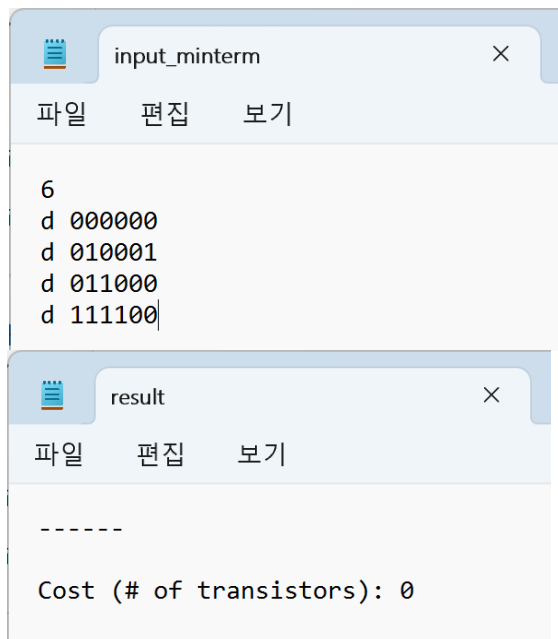
753484995573869382598075175078573552464
670666280416282659399035324427237674085
399840534674430918526636692803643802295
48779639735067294977733306765162292724
230034960843716533391389685696254496255
598305249407396617314680970242863428978
974859549717389270982060758939062848315
874633502905170228345380086420610984379
900430451299525695977548959359241947977
769480555639976084027569691216731903929
536154070333576024644575076393881877813
070443504850697933246311742031709475275
313361747176679306889658232383929748530
095368456761803194329582953329753782761
382383288901299872340120788296618246250
803261080786405540342812030451549850535
413961100339325340987556427841095587495
941845468339407592736310347145289714898
925858880391508071639613645839810245652
781360221029297154343943848477183708449
513571253648542337626112278239660032562
346047444685469340506779033221721012561
615831862916403474288562083455163862259
245867176582469191105777852910287846686
347842686157639039025761131588667898614
495605219403819581081032285150046885668
177196673006756637962919112932969590043
554616047147141802416744170729633783959
743834111247900039515890707401123399959
068241591284351762369185812374615773978
121452541660659549207590882986145941201
503126110456175115328240005696376096819
611331610935227222903497096571891985133

Cost (# of transistor): 694

```

40 비트짜리 minterm 을 50 개 입력했다. 일일이 minimum set of Pls 와 트랜지스터의 개수를 개산할 수는 없지만 잘 실행됨을 확인할 수 있다.

D. Hard testcase



모든 입력된 minterms 가 don't care minterms 일 때, minimum set of PIs 는 ('-' * 비트의 수) 가 되어야 하고, 소비되는 트랜지스터는 0 이어야한다. 하지만 모든 입력이 don't care 인 경우는 따로 예외 처리를 해주지 않는다면 오류가 날 확률이 크다고 생각한다.