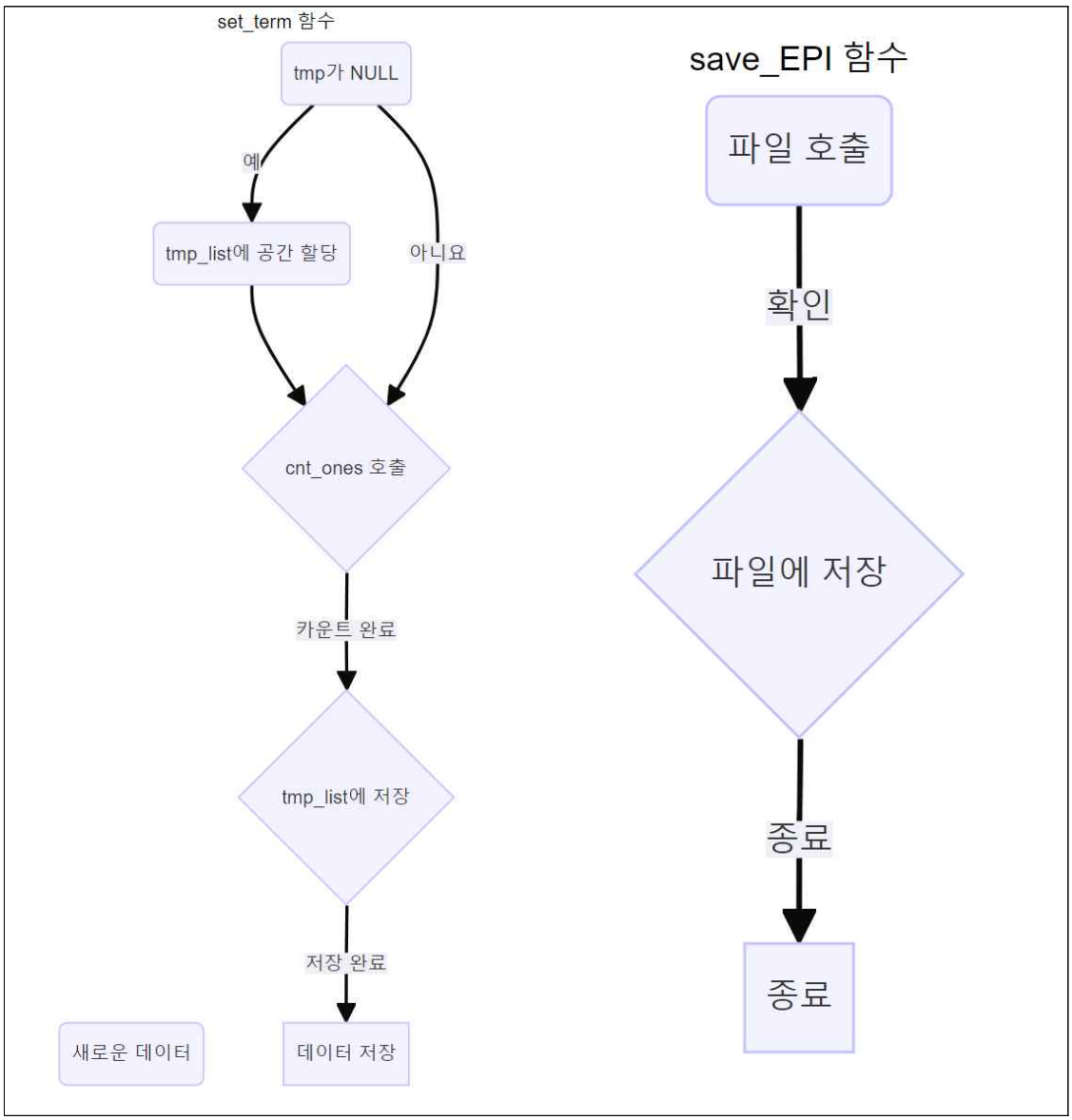
디지털 논리회로1
콰인-맥클러스키 알고리즘
과제 보고서

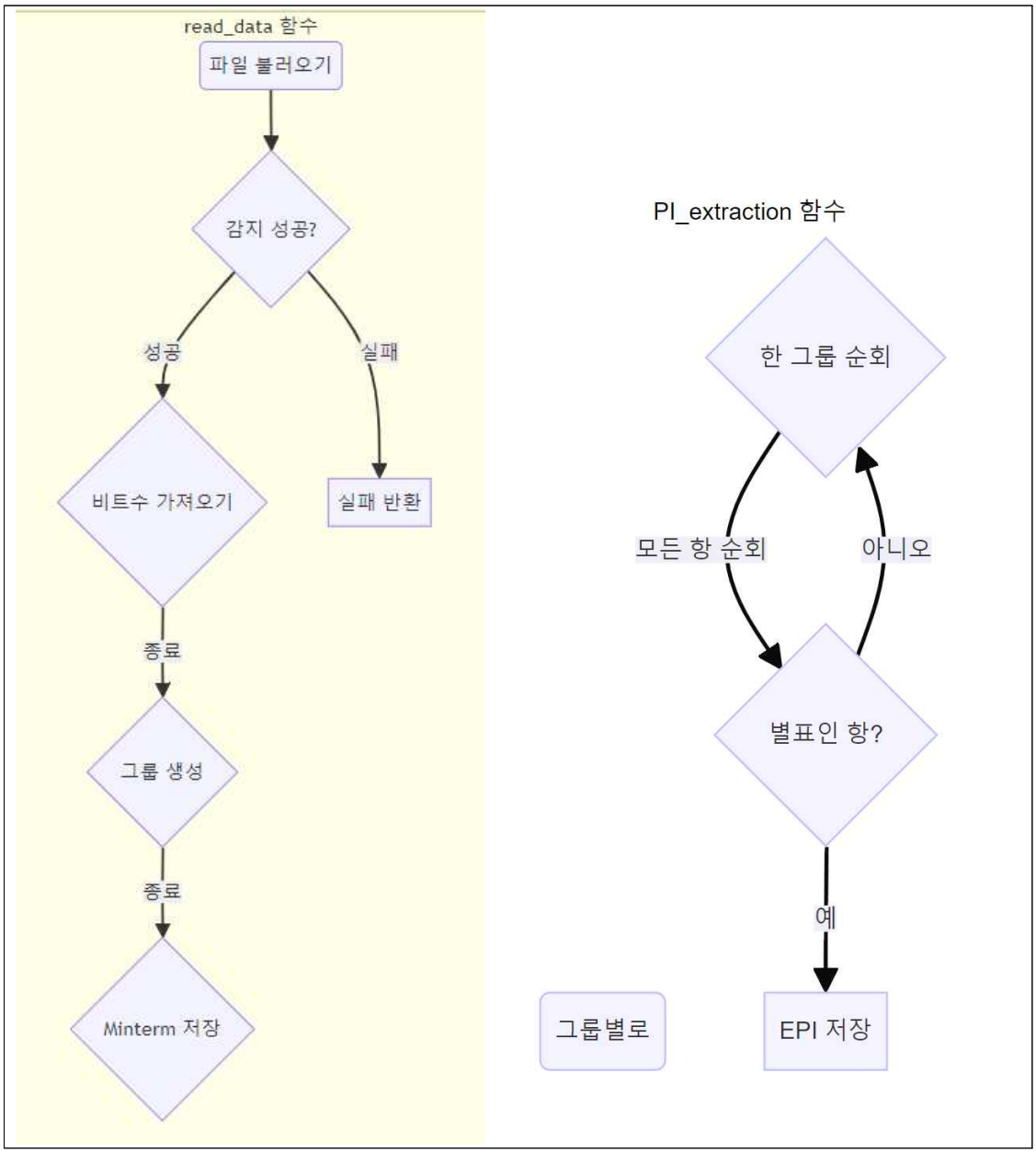
교수님: 유지현 교수님
과목: 디지털논리회로1
학번: 2023202032
이름: 남호준

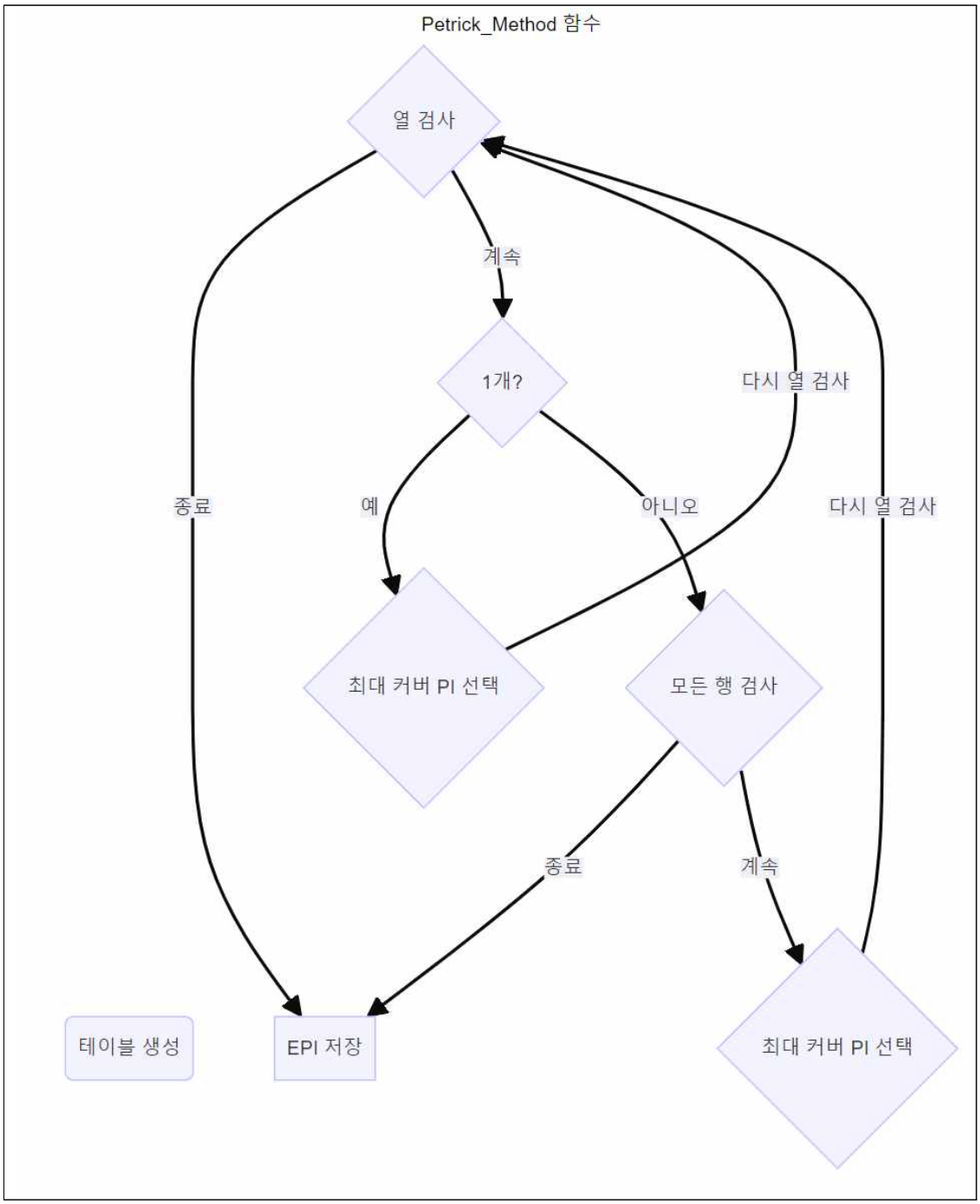
1. 문제 설명

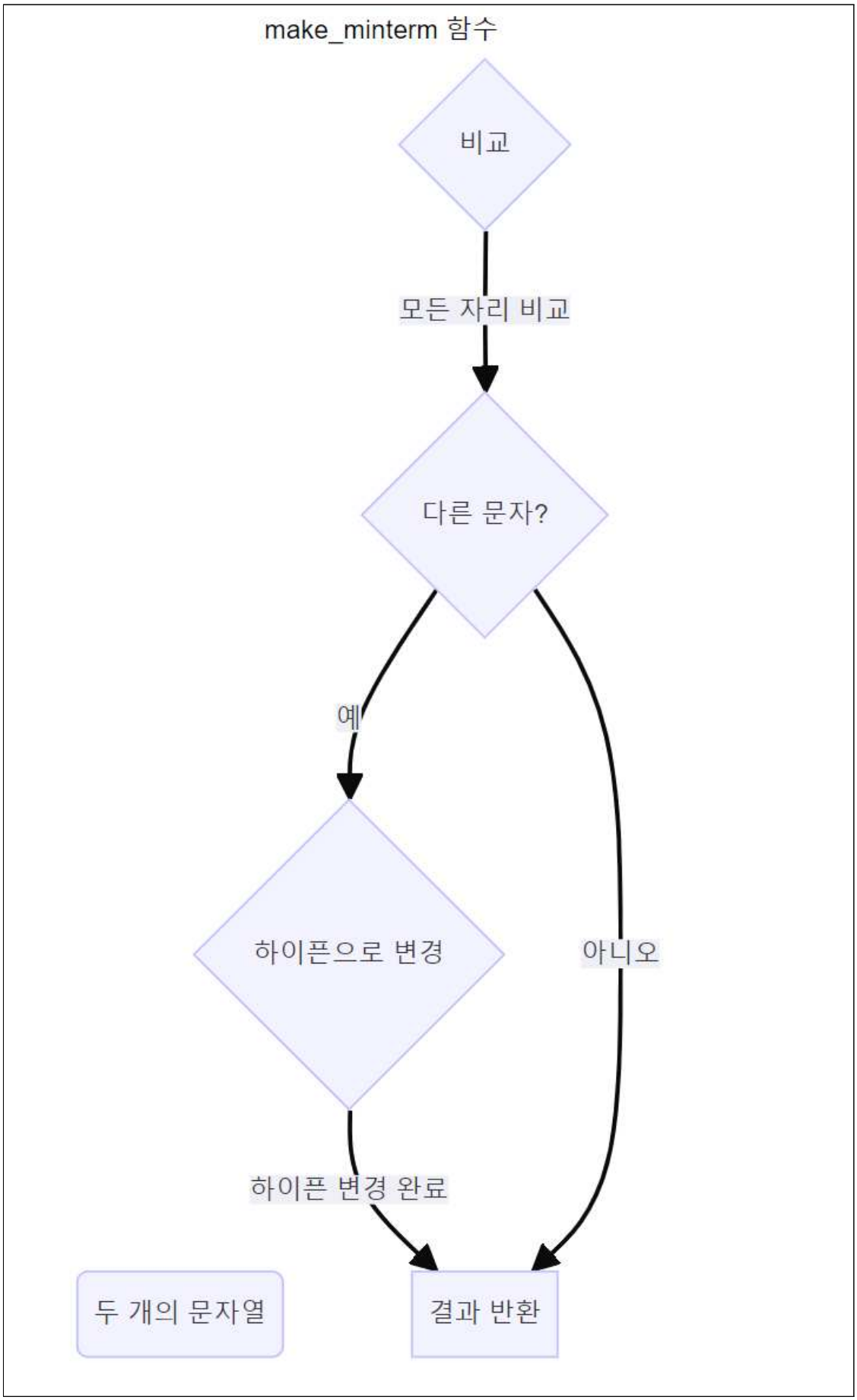
콰인-맥클러스키 알고리즘은 디지털 논리 회로를 최소한의 AND나 OR 게이트를 사용하여 표현하는 데 사용되는 방법 중 하나이다. 카르노 맵은 변수가 4개까지는 가능하지만, 이 알고리즘은 적지 않은 변수의 개수가 들어와도 해결할 수 있는 알고리즘이며, 컴퓨터로 실행하기에 원활한 알고리즘이다. 부울식을 최소화하는 데에 사용한다. 해당 과제에선 비트 수와 minterm을 입력받아 최소화하는 알고리즘을 프로그래밍한다.

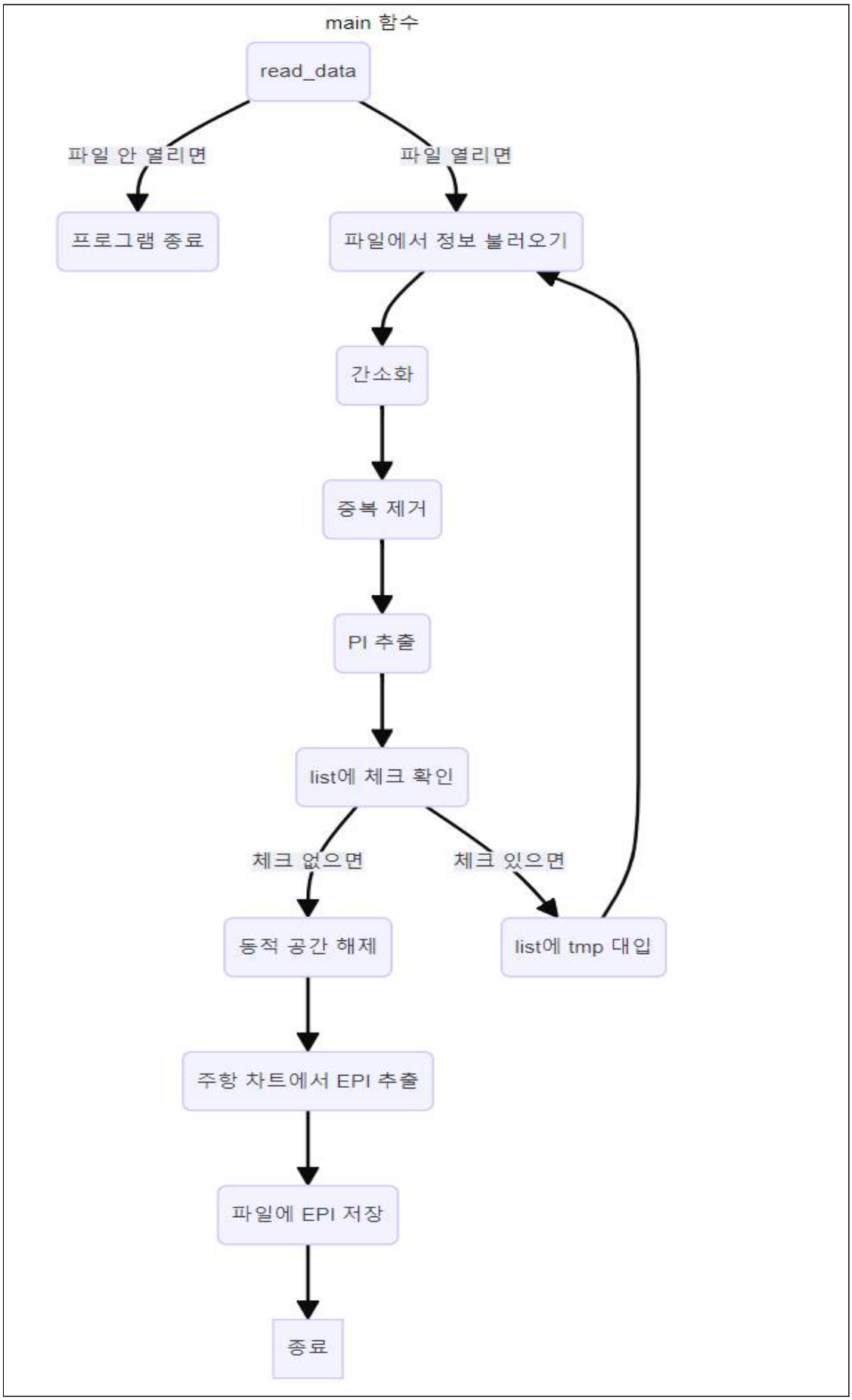
2. 의사코드와 순서도

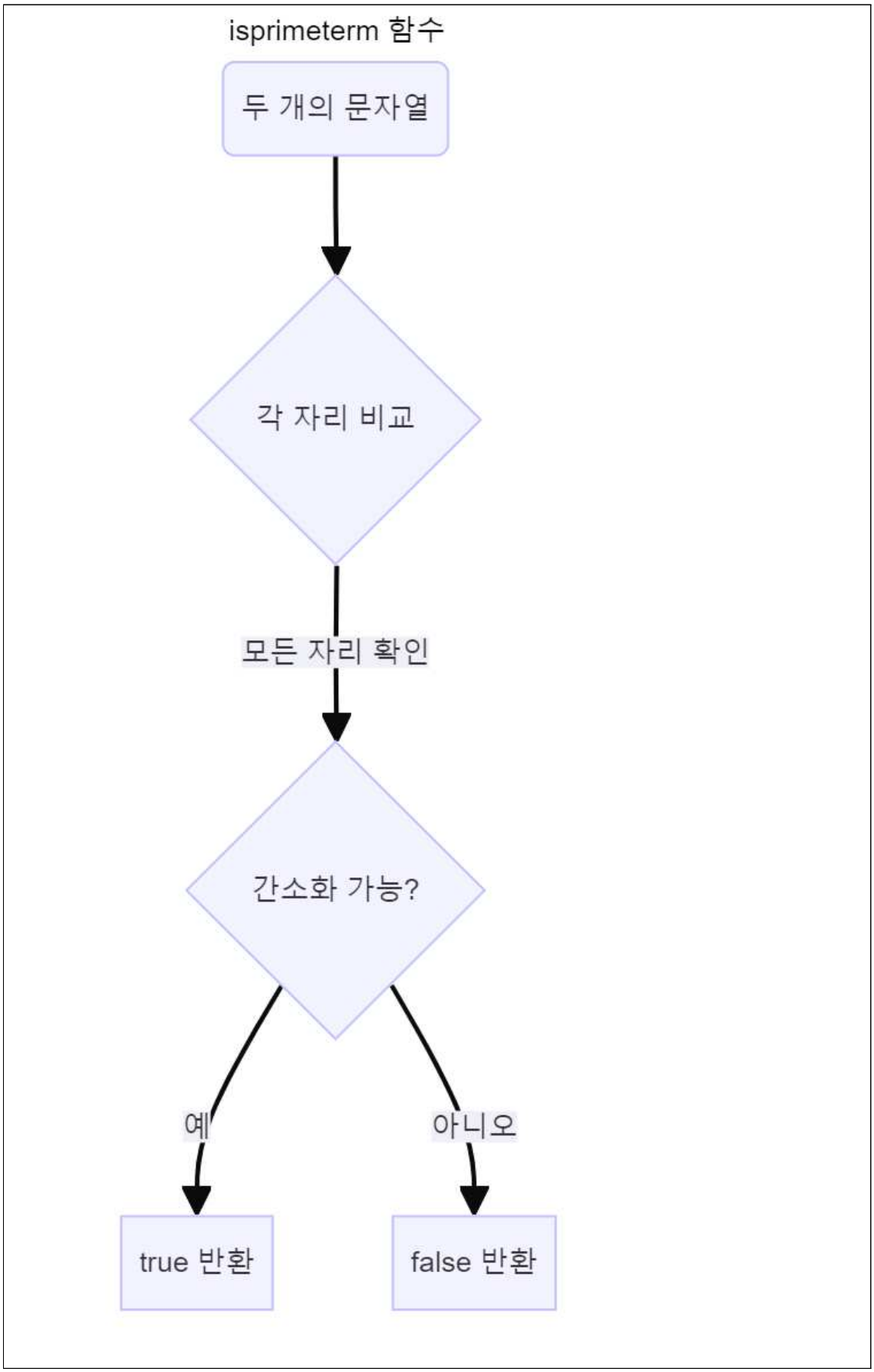


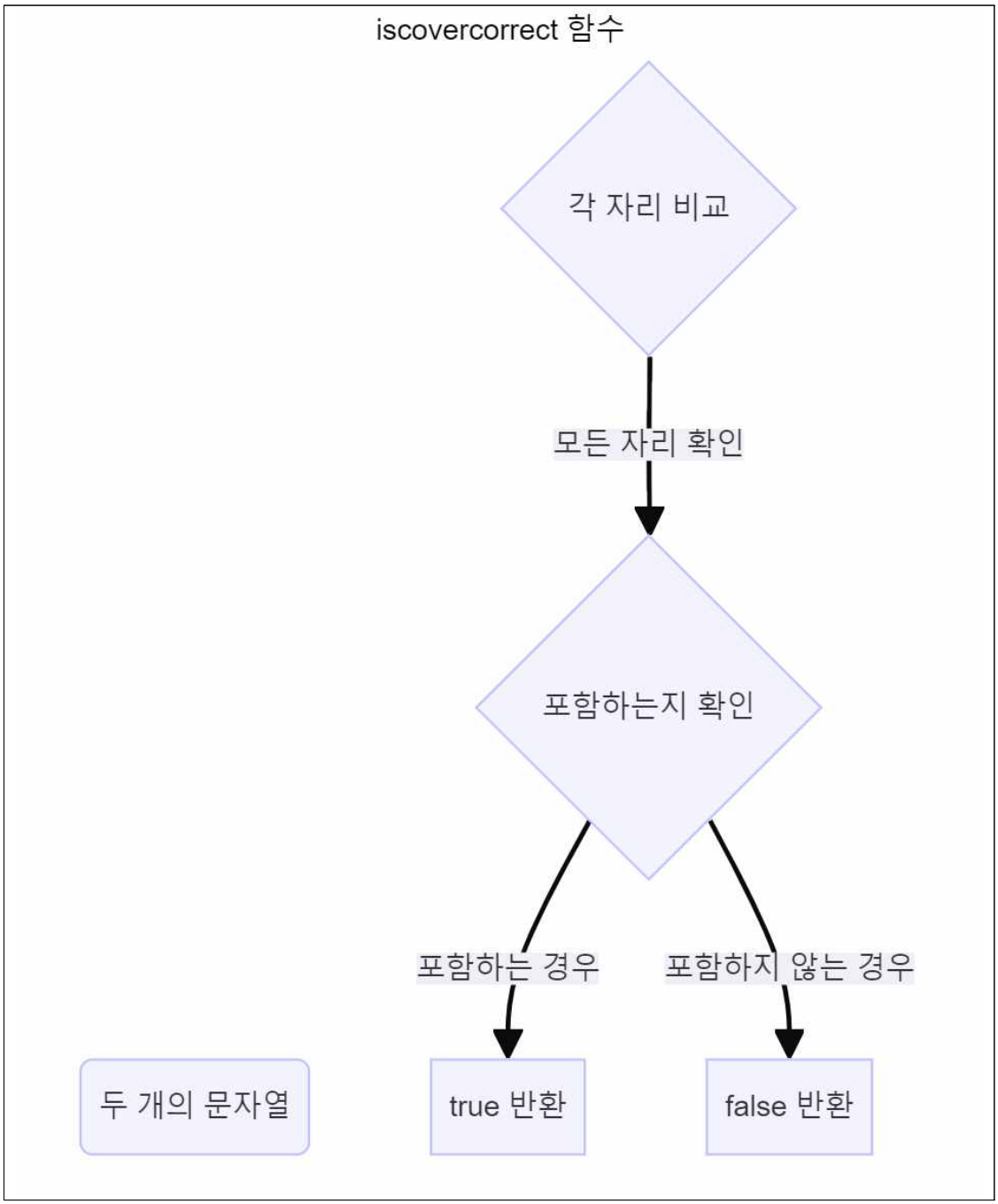

read_data 함수
파일 불러오기
감지 성공?
성공
실패
비트수 가져오기
실패 반환
종료
그룹 생성
종료
Minterm 저장
PI_extraction 함수
한 그룹 순회
모든 항 순회
아니오
별표인 항?
예
그룹별로
EPI 저장

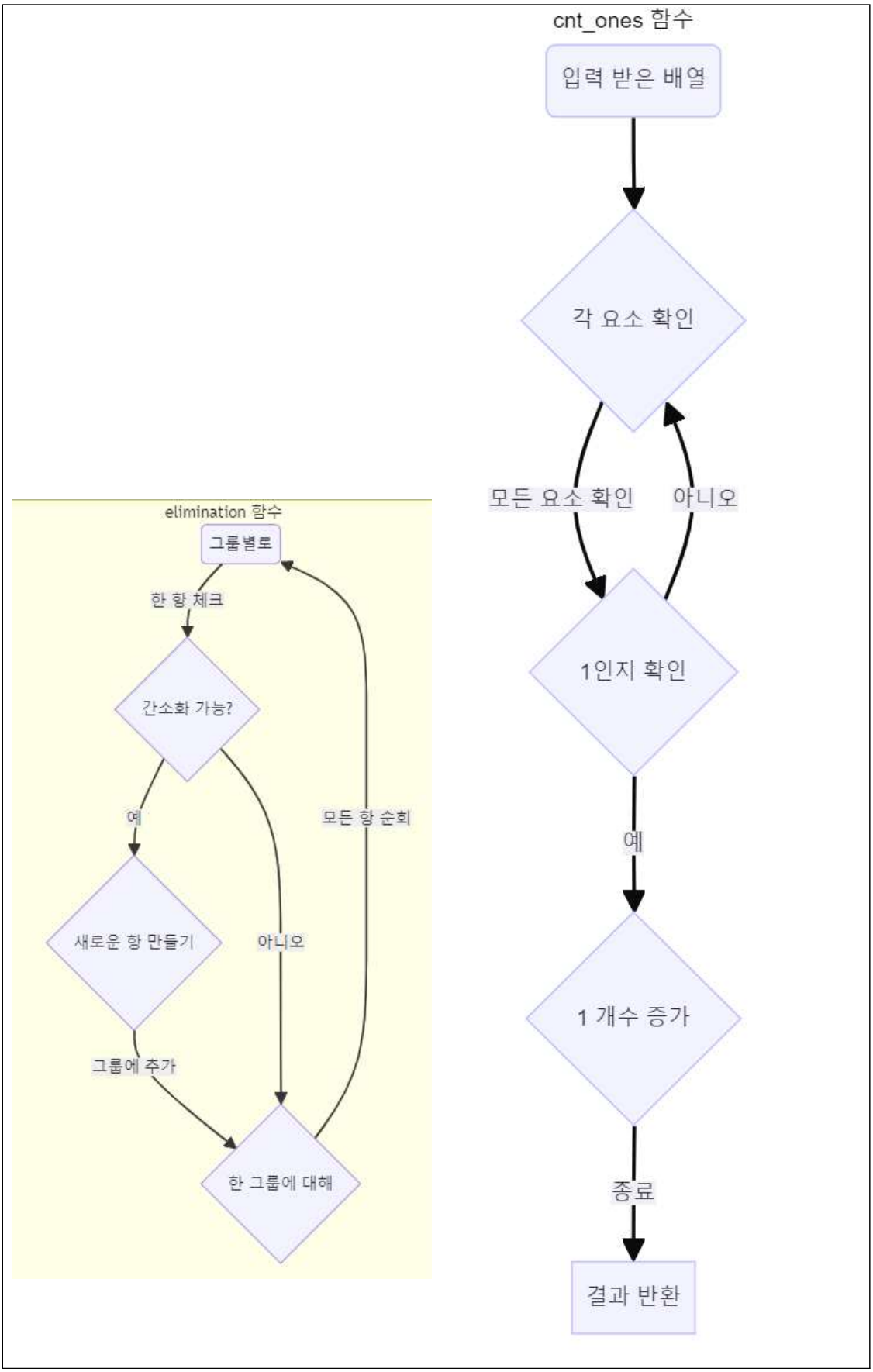

Petrick_Method 함수
열 검사
계속
1개?
다시 열 검사
종료
예
아니오
다시 열 검사
최대 커버 PI 선택
모든 행 검사
종료
계속
테이블 생성
EPI 저장
최대 커버 PI 선택

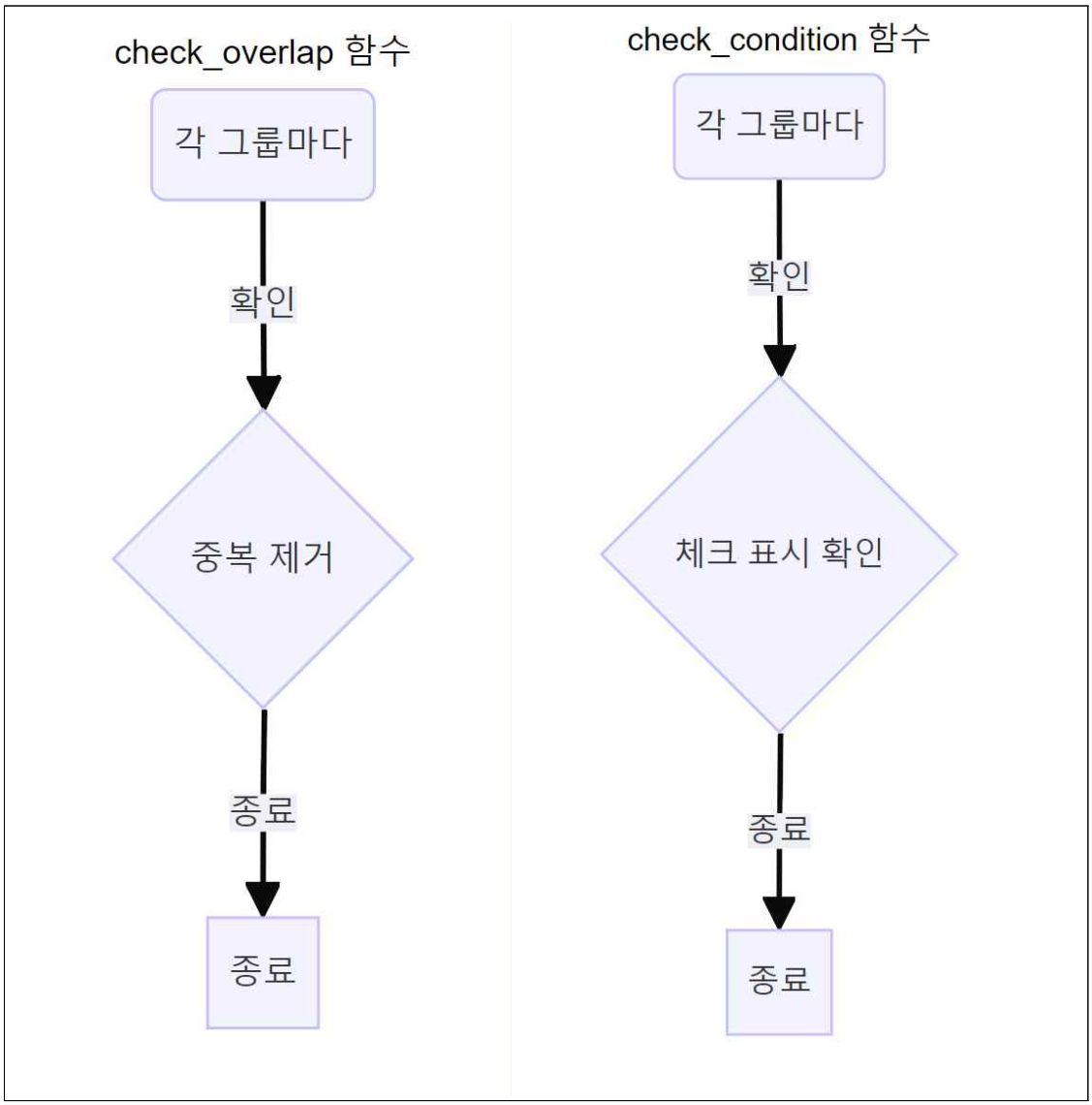

make_minterm 함수
비교
모든 자리 비교
다른 문자?
예
하이픈으로 변경
아니오
하이픈 변경 완료
두 개의 문자열
결과 반환

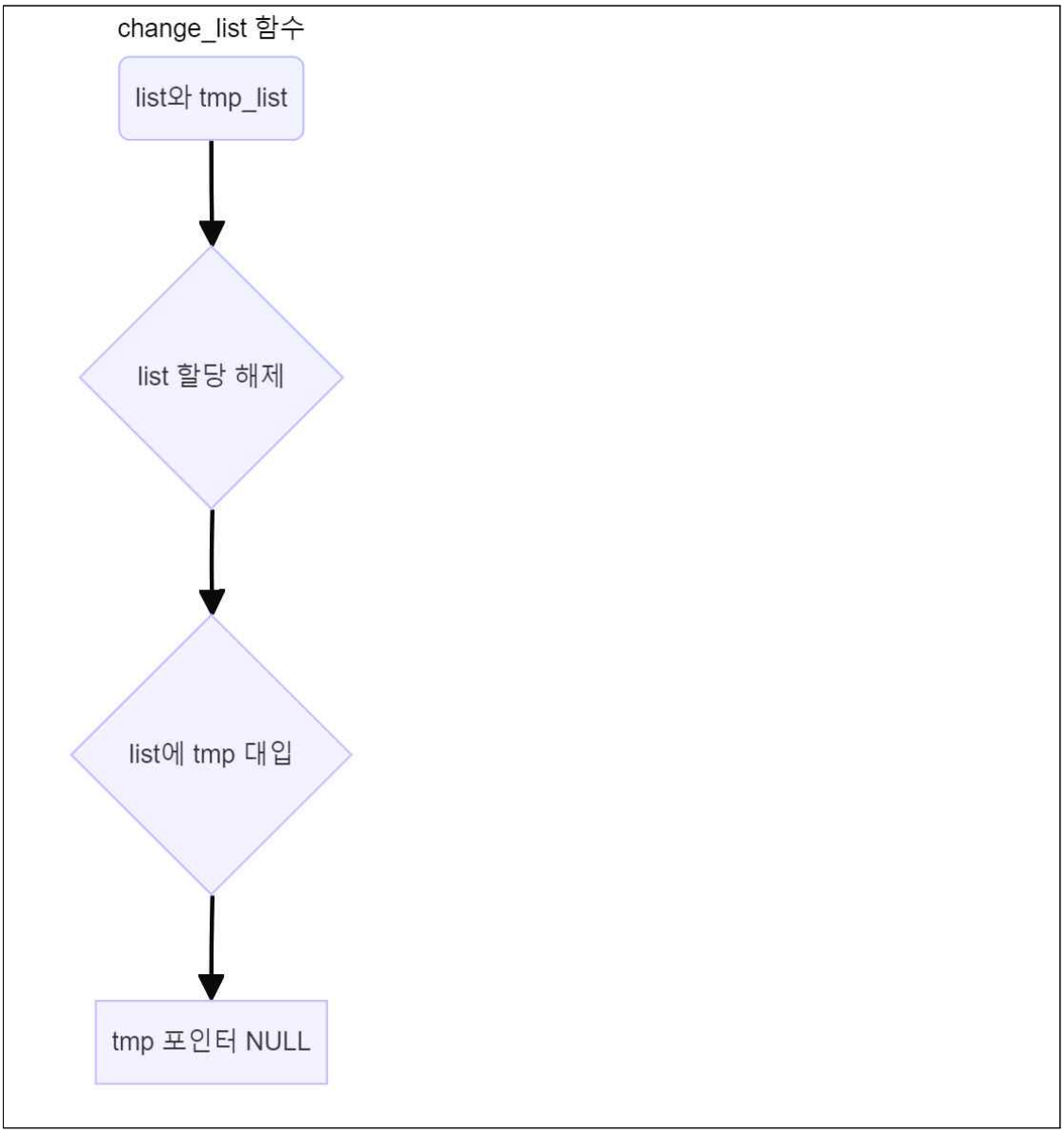

main 함수
read_data
파일 안 열리면
파일 열리면
프로그램 종료
파일에서 정보 불러오기
간소화
증복 제거
PI 추출
list에 체크 확인
체크 없으면
체크 있으면
등적 공간 해제
list에 tmp 대입
주항 차트에서 EPI 추출
파일에 EPI 저장
종료


isprimeterm 함수
두 개의 문자열
각 자리 비교
모든 자리 확인
간소화 가능?
예
아니오
true 반환
false 반환


iscovercorrect 함수
각 자리 비교
모든 자리 확인
포함하는지 확인
포함하는 경우
포함하지 않는 경우
두 개의 문자열
true 반환
false 반환


cnt_ones 함수
입력 받은 배열
각 요소 확인
모든 요소 확인
아니오
1인지 확인
예
1 개수 증가
종료
결과 반환
elimination 함수
그룹별로
한 항 체크
간소화 가능?
예
아니오
모든 항 순회
새로운 항 만들기
그룹에 추가
한 그룹에 대해


check_overlap 함수
각 그룹마다
확인
중복 제거
종료
종료
check_condition 함수
각 그룹마다
확인
체크 표시 확인
종료
종료




```
function check_condition()
    for i <- 0 to n_1 - 1 do
        size <- size of list[i].minterm
        for j <- 0 to size - 1 do
            if list[i].minterm[j].second == true then
                return false
            end if
        end for
    end for
    return true
end function
```

Listing 1: check_condition

```
function make_minterm(a, b)
    new_term <- pair(a, false)
    len <- length of a
    for i <- 0 to len - 1 do
        if a[i] != b[i] then
            new_term.first[i] <- '-'
        end if
    end for
    return new_term
end function
```

Listing 4: make_minterm

```
function set_term(new_data)
    if tmp_list == NULL then
        tmp_list <- new group[n_1]
        for i <- 0 to n_1 - 1 do
            tmp_list[i].data <- i
        end for
    end if
    cnt <- cnt_ones(new_data.first)
    tmp_list[cnt].minterm.push_back(pair(new_data.first, new_data.second)
        )
end function
```

Listing 5: set_term

```
function check_overlap()
    if tmp_list != NULL then
        for i <- 0 to n_1 - 1 do
            tmp_list[i].minterm.erase(unique(tmp_list[i].minterm.begin(),
                tmp_list[i].minterm.end()), tmp_list[i].minterm.end())
        end for
    end if
end function
```

Listing 6: check_overlap

```
function elimination()
    for i <- 1 to n_1 - 1 do
        size <- size of list[i - 1].minterm
        for j <- 0 to size - 1 do
            for k <- 0 to size of list[i].minterm - 1 do
                if isprimeterm(list[i - 1].minterm[j].first, list[i].
                    minterm[k].first) then
                    set_term(make_minterm(list[i - 1].minterm[j].first,
                        list[i].minterm[k].first))
                    list[i - 1].minterm[j].second <- true
                    list[i].minterm[k].second <- true
                end if
            end for
        end for
    end for
end function
```

Listing 1: Elimination

```
function PI_extraction()
    for i <- 0 to n_1 - 1 do
        for j <- 0 to size of list[i].minterm - 1 do
            if list[i].minterm[j].second == false then
                PI.push_back(list[i].minterm[j].first)
            end if
        end for
    end for
end function
```

Listing 2: PI_extraction

```
function isprimeterm(a, b)
    len <- length of a
    cnt <- 0
    for i <- 0 to len - 1 do
        if cnt > 1 then
            return false
        end if
        if |a[i] - b[i]| == 1 then
            cnt <- cnt + 1
        else if |a[i] - b[i]| == 3 or |a[i] - b[i]| == 4 then
            return false
        end if
    end for
    return true
end function
```

Listing 3: isprimeterm

```
function Petrick_Method()
    rows <- size of PI
    cols <- size of minterm
    table <- new int[rows][cols]
    for i <- 0 to rows - 1 do
        for j <- 0 to cols - 1 do
            if iscovercorrect(minterm[j], PI[i]) then
                table[i][j] <- 1
            else
                table[i][j] <- 0
            end if
        end for
    end for
    for i <- 0 to cols - 1 do
        cnt <- 0
        for j <- 0 to rows - 1 do
            if table[j][i] == 1 then
                cnt <- cnt + 1
            end if
        end for
        if cnt == 1 then
            for j <- 0 to rows - 1 do
                if table[j][i] == 1 then
                    EPI.push_back(PI[j])
                    for k <- 0 to cols - 1 do
                        table[k][i] <- 0
                    end for
                    break
                end if
            end for
        end if
    end for
    cover <- new vector<int>()
    while true do
        stop <- 0
        for i <- 0 to rows - 1 do
            cnt <- 0
            for j <- 0 to cols - 1 do
                if table[i][j] == 1 then
                    cnt <- cnt + 1
                end if
            end for
            cover.push_back(cnt)
            stop <- stop + cnt
        end for
        if stop == 0 then
            break
        end if
        max_id <- 0
```

```
        for i <- 0 to rows - 1 do
            if cover[max_id] < cover[i] then
                max_id <- i
            end if
        end for
        EPI.push_back(PI[max_id])
        for i <- 0 to cols - 1 do
            if table[max_id][i] == 1 then
                for j <- 0 to rows - 1 do
                    table[j][i] <- 0
                end for
            end if
        end for
    end while
    delete[] table
end function
```

Listing 1: Petrick_Method

```
function iscovercorrect(a, b)
    len <- length of a
    for i <- 0 to len - 1 do
        if a[i] != b[i] and a[i] != '-' and b[i] != '-' then
            return false
        end if
    end for
    return true
end function
```

Listing 2: iscovercorrect

```
function change_list()
    delete[] list
    list <- tmp_list
    tmp_list <- NULL
end function
```

Listing 3: change_list

```
function cnt_ones(arr)
    cnt <- 0
    for i <- 0 to length of arr - 1 do
        if arr[i] == '1' then
            cnt <- cnt + 1
        end if
    end for
    return cnt
end function
```

Listing 4: cnt_ones

```
function read_data()
    open read_file "input_minterm.txt"
    if read_file fails then
        return true
    end if
    read first line of read_file
    n <- convert line to integer
    n_1 <- n + 1
    list <- new group[n_1]
    for i <- 0 to n_1 - 1 do
        list[i].data <- i
    end for
    while not end of read_file do
        read a line from read_file
        tmp <- count ones in line
        list[tmp].minterm.push_back(pair(line, false))
        minterm.push_back(line)
    end while
    close read_file
    return false
end function
```

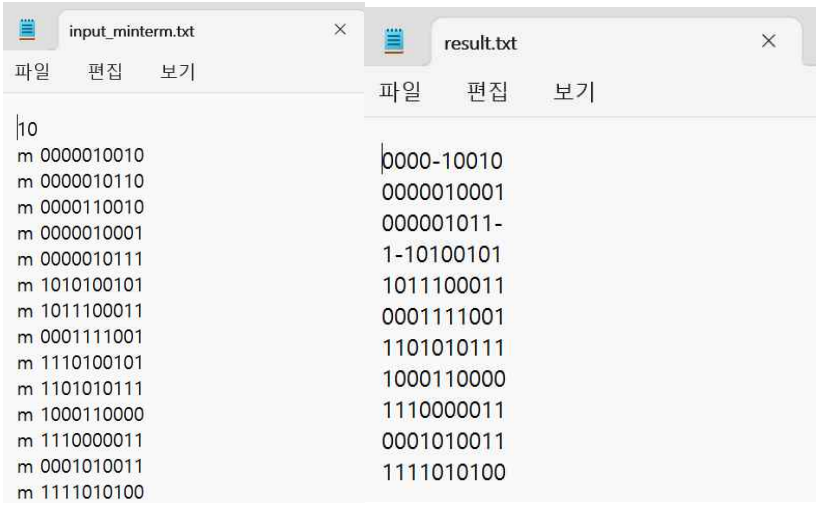
Listing 5: read_data

```
function save_EPI()
    open write_file "result.txt"
    for i <- 0 to size of EPI - 1 do
        write EPI[i] to write_file
        write newline to write_file
    end for
    close write_file
end function
```

Listing 6: save_EPI

위와 같이, 각 함수에 대해, 순서도와 의사 코드를 작성하였다.
큰 흐름으로 볼 때, 정보를 입력받아 간소화 후, 주항 차트를 이용하여 EPI를 추출 후 파일에 저장하는 프로그램이다.
하나만의 PI에 포함된 minterm을 먼저 EPI로 선정 후 그 항이 가진 다른 항을 전부 0으로 없앤다. 이후 각 PI가 가진 유효한 minterm의 커버 가능 범위를 cover로 체크하여 가장 큰 값을 체크하고 EPI로 선정하는 과정을 반복하여 table 변수 안에 모든 1이 0이 될 때까지 하나씩 EPI로 뽑는 과정이 중요한 알고리즘(Petrik_method함수 내용)이다.

3. 검증 사례 및 설명



아래는 위의 파일 내 데이터이다.

//////////

10

m 0000010010

m 0000010110

m 0000110010

m 0000010001

m 0000010111

m 1010100101

m 1011100011

m 0001111001

m 1110100101

m 1101010111

m 1000110000

m 1110000011

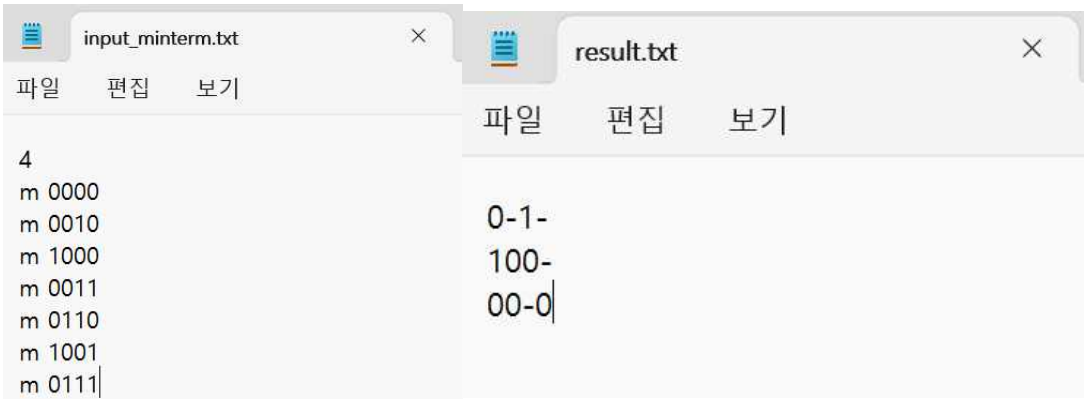
m 0001010011

m 1111010100

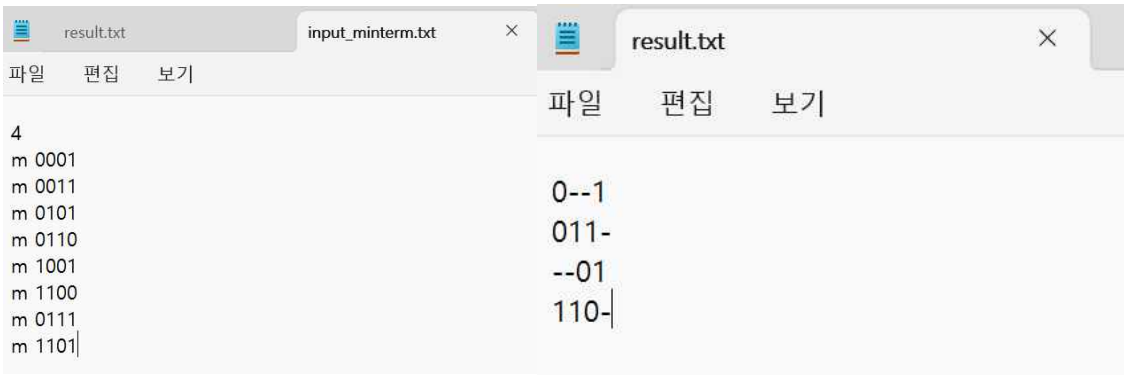
//////////////////

여기서 0000-10010, 000001011-, 1-10100101로 2개씩 데이터가 간소화 되어 총 14개의 데이터에서 11개의 항으로 줄어들었다.

아래는 입력과 출력으로 나온 파일이다.

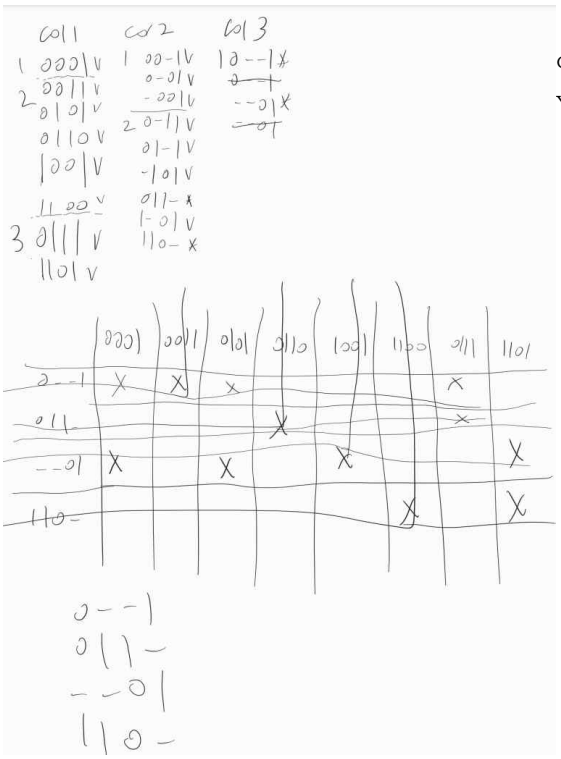


위의 식도 7개에서 3개로 줄어들었다. (각 항별로 3개 포함, 2개 포함, 2개 포함)



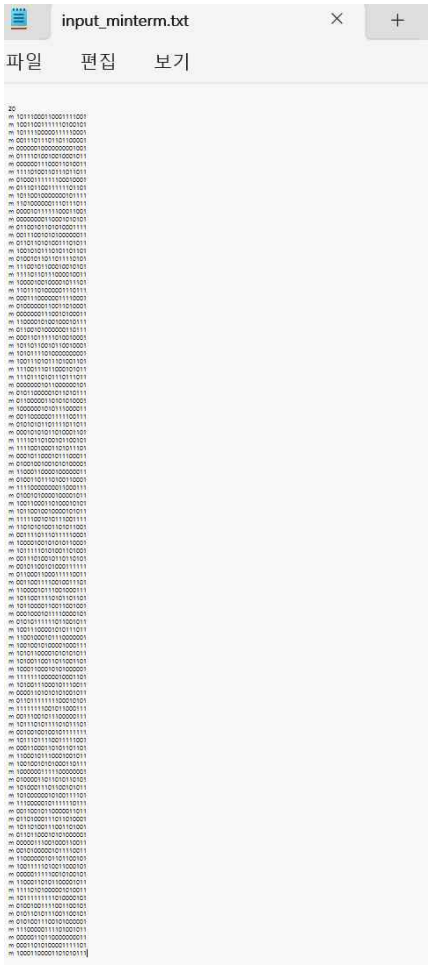
위의 식은 8개의 항에서 0——1이 4개 포함, 011-이 2개 포함, --01이 4개 포함, 110-이 2개 포함하여 최소항이 되었다.

아래는 위의 콰인-맥클러스키 알고리즘을 손으로 계산한 것이다.



아래의 최종 부울식은
Y=A'D+A'BC+C'D+ABC'이다.

4. 풀기 매우 어렵다고 생각하는 예제
아래의 사진은 입력과 출력을 캡쳐한 것이다.
데이터의 양이 많아 잘 보이지 않는다.
아래의 실제 입력과 출력은 다음과 같다.
파일 이름은 input_minterm8.txt이다.



입력 파일(input_minterm8.txt)

////////////////

20

m 10111000110001111001

m 10011001111110100101

m 10111100000111110001

m 00111011101101100001

m 00000010000000001001

m 01111010010010001011

m 00000011100011010011

m 11110100110111011011

m 01000111111100010001

m 01110110011111101101

m 10110010000000101111

m 11010000001110111011

m 00001011111100011001

m 00000000110001010101

m 01100101101010001111

m 00111001010100000011

m 01101101010011101011

m 10010101110101101101

m 01001011011011110101

m 11100101100010010101

m 11110110111000010011

m 10000100100001011101

m 11011101000001110111

m 00011100000011110001

m 01000000110011010001

m 00000001110010100011

m 11000010100100010111

m 01100101000000110111

m 00011011111010010001

m 10110110010110010001

m 10101111010000000001

m 10011101011101001101
m 11100111011000101011
m 11101110101110111011
m 00000001011000000101
m 01011000001011010111
m 01100000110101010001
m 10000001010111000011
m 00110000001111100111
m 01010101101111011011
m 00010101011010001101
m 11110110100101100101
m 11110010001101011101
m 00010110001011100011
m 01001001001010100001
m 11000110000100000011
m 01001101110100110001
m 11110000000011000111
m 01001010000100001011
m 10011000110100010101
m 10110010010000101011
m 11111001010111001111
m 11010101001101011001
m 00111101110111110001
m 10000100101010110001
m 10111110101001101001
m 00111010010110110101
m 00101100101000111111
m 01100011000111110011
m 00110011110010011101
m 11000010111001000111
m 10110011110101101101
m 10110000110011001001
m 00010001011110000101
m 01010111111011001011

m 10011100001010111011
m 11001000101110000001
m 10010010100001000111
m 10101100001010101011
m 10100110011011001101
m 10001100010101000001
m 11111110000010001101
m 10100111000101110011
m 00001101010101001011
m 01101111111100010101
m 11111111001011000111
m 00111001011100000111
m 10111010111101011101
m 00100100100101111111
m 10111011110011111001
m 00011000110101101101
m 11000101110001001011
m 10010010101000110111
m 10000001111100000001
m 01000011011010110101
m 10100011101100101011
m 10100000010100111101
m 11100000101111110111
m 00110010110000011011
m 01101000111011010001
m 10110100111001101001
m 01101100010101000001
m 00000111001000110011
m 00101000001011110011
m 11000000101101100101
m 10011111010011000101
m 00000111110010100101
m 11000110101100001011
m 11110101000001010011

m 10111111111010000101
m 01001001111001100101
m 01011010111001100101
m 01010011100101000001
m 11100000111101001011
m 00000110110000000011
m 00011010100001111101
m 10001100001101010111
//////////////////////

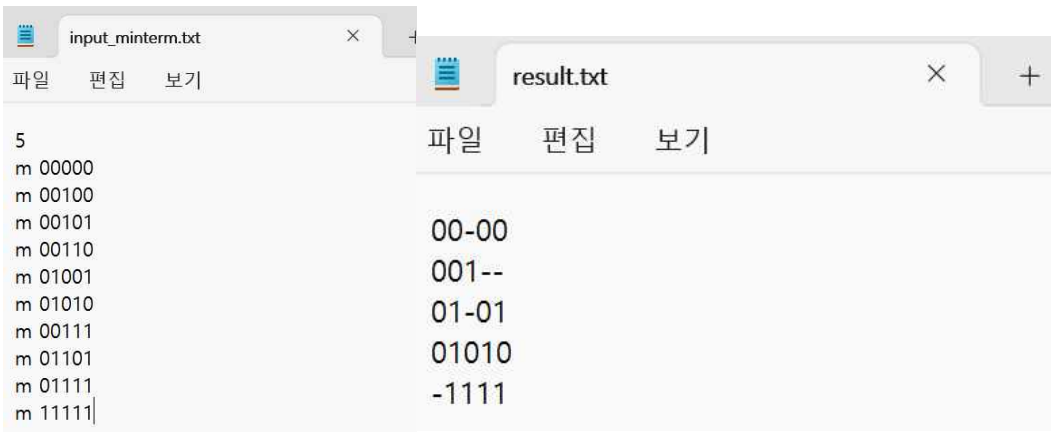
출력(result.txt)
//////////////////
10111000110001111001
10011001111110100101
10111100000111110001
00111011101101100001
00000010000000001001
01111010010010001011
00000011100011010011
11110100110111011011
01000111111100010001
01110110011111101101
10110010000000101111
11010000001110111011
00001011111100011001
00000000110001010101
01100101101010001111
00111001010100000011
01101101010011101011
10010101110101101101
01001011011011110101
11100101100010010101
11110110111000010011
10000100100001011101

1101110100000111O111
0001110000001111O001
0100000011001101O001
0000000111001O100011
1100001010010001O111
0110010100000011O111
0001101111101O010001
1011011001011001O001
1010111101000000O001
1001110101110100l101
1110011101100010l011
1110111010111011l011
0000000101100000O101
0101100000101101O111
0110000011010101O001
1000000101011100O011
0011000000111110O111
0101010110111101l011
0001010101101000l101
1111011010010110O101
1111001000110101l101
0001011000101110O011
0100100100101010O001
1100011000010000O011
0100110111010011O001
1111000000001100O111
0100101000010000l011
1001100011010001O101
1011001001000010l011
1111100101011100l111
1101010100110101l001
0011110111011111O001
1000010010101011O001
1011111010100110l001

00111010010110110101
00101100101000111111
01100011000111110011
00110011110010011101
11000010111001000111
10110011110101101101
10110000110011001001
00010001011110000101
01010111111011001011
10011100001010111011
11001000101110000001
10010010100001000111
10101100001010101011
10100110011011001101
10001100010101000001
11111110000010001101
10100111000101110011
00001101010101001011
01101111111100010101
11111111001011000111
00111001011100000111
10111010111101011101
00100100100101111111
10111011110011111001
00011000110101101101
11000101110001001011
10010010101000110111
10000001111100000001
01000011011010110101
10100011101100101011
10100000010100111101
11100000101111110111
00110010110000011011
01101000111011010001

10110100111001101001
01101100010101000001
00000111001000110011
00101000001011110011
11000000101101100101
10011111010011000101
00000111110010100101
11000110101100001011
11110101000001010011
10111111111010000101
01001001111001100101
01011010111001100101
01010011100101000001
11100000111101001011
00000110110000000011
00011010100001111101
10001100001101010111
//////////////////////////
위의 결과는 해밍 거리가 1인 경우의 조합이 없는 경우이다.
따라서 식의 간소화가 되지 않는다.

5. 예제를 사용한 테스트



예제를 사용했을 때, 같은 결과를 얻을 수 있었다.