Quine-McCluskey Algorithm
구현 결과 보고서

1. 구현

생성자로 input의 개수, minterm, don't care를 받아 QM Method를 수행하는 class를 만들었다. QM Method를 수행하는 각 과정이 어떤 규칙을 적용해서 어떻게 변화한 것인지 자세하게 표현하기 위해 최대한 손으로 직접 계산하는 과정과 유사하게 구현했다.

```
class QM:
    def __init__(self, numInput, minterm, dontcare):
        self.numInput = numInput  # input의 개수
        self.minterm = sorted(minterm)  # minterm들
        self.dontcare = sorted(dontcare)  # don't care들
        self.PI = []  # PI들
        self.EPI = []  # EPI들
        self.petrickResult = []  # petrick's method로 선택한 항
        self.table = None  # 각 행에 PI들이, 각 열에 minterm들이 배치된 표
        self.rows = 0  # self.table의 행 개수
        self.cols = 0  # self.table의 열 개수
        self.expression = []  # petrick's method를 수행하기 위한 boolean 표현식
    # PI들을 찾는 메소드
    def findPI(self):
    # 각 행에 PI를, 각 열에 minterm을 배치한 표를 만드는 메소드
    def makeTable(self):
    # 표를 줄일 수 있는 경우, 다시 만드는 메소드
    def remakeTable(self, direction, willDelete):
    # 현재 단계와 함께 표를 출력하는 메소드
    def printTable(self, progress=''):
    # 표에서 EPI를 찾는 메소드
    def findEPI(self):
    # 표에서 Row Dominance를 찾는 메소드
    def rowDominance(self):
    # 표에서 Column Dominance를 찾는 메소드
    def colDominance(self):
    # petrick's method를 수행하기 위해 표에서 POS 형태의 표현식을 만드는 메소드
    def makeExpression(self):
    # 현재 단계와 boolean 표현식을 출력하는 메소드
    def printExpression(self, progress=''):
    # petrick's method를 수행하는 메소드
    def petrick(self):
    # boolean 표현식을 분배법칙으로 간단하게 하는 메소드
    def distributive(self):
    # boolean 표현식을 전개하는 메소드
    def multiply(self):
    # boolean 표현식을 흡수법칙으로 간단하게 하는 메소드
    def absorption(self):
    # 결과를 출력하는 메소드
    def showResult(self):
    # 전체적인 과정을 수행하는 메소드
    def solve(self):
    # 모든 과정이 끝난 후 구한 표현식이 올바른지 검증하는 메소드
    def verification(self):
```

PI

PI를 찾는 첫 번째 과정은 각 minterm들을 이진법으로 나타냈을 때 1이 포함된 개수로 나열하는 것이다. 2차원 리스트 table을 만들어 첫 번째 인덱스로 1의 개수를 표현하고 그 안에 입력으로 주어진 minterm과 don't care를 딕셔너리 형태로 저장했다. 키로는 포함하고 있는 minterm들의 set인 minterm, 이진법으로 표현한 binary, 다른 implicant와 합쳐졌는지를 표시하는 combined로 총 3가지를 갖고있다.

```
table = [[] for _ in range(self.numInput + 1)]
for num in sorted(self.minterm + self.dontcare):
   numToBin = bin(num)[2:].zfill(self.numInput)
   table[numToBin.count('1')].append({'minterm': {num}, 'binary': numToBin, 'combined': False})
```

구성된 table을 for문 3개를 이용해 모든 조합 가능한 쌍을 확인하는데, 첫 번째 for문은 1의 개수를, 두 번째 for문은 첫 번째 for문에서 정해진 1의 개수만큼 1을 갖고있는 implicant들을, 세 번째 for문은 두 번째 for문보다 1을 하나 더 갖고있는 implicant들을 담당한다.

```
for numOne in range(self.numInput):
   for upper in table[numOne]:
       for lower in table[numOne + 1]:
```

이렇게 확보한 두 implicant를 비교해서 이진코드가 단 한 자리만 차이가 난다면 table과 똑같이 구성된 nextTable에 두 implicant를 합친 새로운 implicant를 저장한다. 그런데 다른 방식으로 합성된 똑같은 implicant가 이미 nextTable에 있을 수 있으므로 확인한 후 저장한다. 그리고 다음으로 넘어가기 전에 PI를 구별하기 위해 combined를 체크한다.

```
binary = upper['binary']
binary = binary[:i] + '-' + binary[i + 1:]
newElem = {'minterm': set(), 'binary': binary, 'combined': False}
newElem['minterm'] = upper['minterm'] | lower['minterm']
for elem in nextTable[numOne]:
   if newElem['minterm'] == elem['minterm']:
       break
else:
   nextTable[numOne].append(newElem)
upper['combined'] = True
lower['combined'] = True
```

현재 table을 모두 확인했다면 모든 implicant들의 combined를 확인해 이 값이 False인 경우 PI에 해당 implicant를 저장한다. combined는 더 이상 활용하지 않을 키이기 때문에 이 단계에서 삭제했다. 이후 nextTable이 나오지 않을 때까지 nextTable을 table에 저장하고 반복한다.

```
for numOne in range(len(table)):
   for implicant in table[numOne]:
       if not implicant['combined']:
           implicant.pop('combined')
           self.PI.append(implicant)
```

EPI

EPI를 찾고, Row Dominance와 Column Dominance를 적용하는 과정을 한눈에 보기 위해 사용되는 table을 이미 찾은 PI들로 구성한다. 각 행에는 PI들이, 각 열에는 minterm들을 배치해서 해당 행의 PI가 해당 열의 minterm을 포함하고 있는지 나타내도록 했다. PI를 찾을 때에는 cost를 줄이는 데에 don't care가 필요했지만 이제는 고려할 필요가 없어서 제외했다.

```
table = [[None] + self.minterm]
for i in range(len(self.PI)):
    row = [self.PI[i]]
    for num in self.minterm:
        row.append(num in self.PI[i]['minterm'])
    table.append(row)
```

표의 열들을 확인하면서 True가 하나밖에 없다면 해당 minterm을 포함하는 PI가 하나밖에 없다는 것을 의미하기 때문에 EPI에 저장하고 찾은 EPI들을 표에서 삭제해 표를 더 줄여나간다.

```
EPI = set()
for c in range(1, self.cols):
    taken = False
    for r in range(1, self.rows):
        if self.table[r][c]:
            epiR = r
            if taken:
                break
            taken = True
    else:
        if taken:
            EPI.add(epiR)
```

Row Dominance

PI 딕셔너리들은 minterm키로 포함하고 있는 minterm들의 set를 갖고있다. 하지만 talbe이 이미 축약되어있다면 현재의 table 정보와 PI의 minterm이 다를 수 있기 때문에 table을 기준으로 각 행마다 포함하고 있는 minterm에 대한 정보를 set으로 만들어 부분집합 관계인지를 확인하고, 이 결과를 통해 dominance한지 판별한다. 단, 한 PI가 자신보다 cost가 더 낮은 PI를 dominance한 경우는 무시한다.

```
pivot = {c for c in range(1, self.cols) if self.table[i][c]}
compare = {c for c in range(1, self.cols) if self.table[j][c]}
if compare.issubset(pivot):
    if len(self.table[i][0]['minterm']) >= len(self.table[j][0]['minterm']):
        willDelete.add(j)
```

Column Dominance

Row Dominance의 경우와 마찬가지로 구성된 table에서 각 열에 대해서 minterm을 포함하는 PI들을 set으로 저장한 다음 Row Dominance와 같은 방식으로 진행한다. 단, Row Dominance와 다르게 dominance하는 쪽이 지워져야 하고 cost를 신경쓰지 않아도 된다.

```
pivot = {r for r in range(1, self.rows) if self.table[r][i]}
compare = {r for r in range(1, self.rows) if self.table[r][j]}
if pivot.issubset(compare):
   willDelete.add(j)
```

EPI와 Row Dominance, Column Dominance는 모든 minterm을 포함하게 되거나, table을 더 이상 축약할 수 없거나, 더 이상 table에 유의미한 값이 남아있지 않을 때까지 번갈아가며 계속해서 시행한다.

```
shrunk = True
        while shrunk:
            shrunk = False
            for func in [self.findEPI, self.rowDominance, self.colDominance]:
                if self.rows >= 2 and self.cols >= 2 and func():
                    shrunk = True
```

Petrick's Method

더 이상 table을 간단하게 만들 수 없으면서 minterm이 남아있으면, 남은 minterm들 빠짐없이 포함하는 것을 보장하도록 POS형태로 식을 만들어 준다. 처음에는 expressioin 리스트 내부에 각각 곱하기로 연결된 항들을 나타내는 set을 넣고, set의 내부에 더하기로 연결된 항을 나타내는 set을 또 넣으려고 생각했는데, python의 set는 원소로 set를 넣을 수 없었다. set을 사용한 이유는 항을 전개해 나가는 과정에서 중복되는 항을 자연스럽게 제거하기 위한 목적이었다. 그래서 set 대신 add처럼 자신이 변경되는 메소드를 사용할 수 없지만 set의 원소로 포함될 수 있는 frozenset을 사용했다.

```
for c in range(1, self.cols):
   term = set()
   for r in range(1, self.rows):
       if self.table[r][c]:
           term.add(frozenset([r]))
   self.expression.append(term)
```

이제 식을 전개해서 SOP 형태로 만들어야 한다. 단순하게 모든 항을 곱해서 전개한다고 해도 컴퓨터의 계산능력이라면 간단하게 수행할 수 있겠지만, 직접 계산을 할 때에는 Algebraic Manipulation을 통해 표현식을 간단하게 줄여나가야 편리한다. 그래서 이 과정도 구현해보았다.

전개를 하기 전에 분배법칙을 먼저 적용했다. 두 항에 공통적으로 포함되는 부분이 있다면, 새로운 항에 공통부분과 나머지 부분들을 서로 곱해 나온 항들을 넣어주었다. 중첩된 for문들을 한 번에 break하는 방법을 고민하다 두 개씩 짝을 지어 묶었다.

```
for l, r in [(l, r) for l in range(len(self.expression)) for r in range(l + 1, len(self.expression))]:
    if l in willDelete or r in willDelete:
        continue

    for i, j in [(i, j) for i in self.expression[l] for j in self.expression[r]]:
        if i == j:
            newTerm = {i}
            for x, y in [(x, y) for x in self.expression[l] - {i} for y in self.expression[r] - {j}]:
                newTerm.add(frozenset(x | y))
            self.expression.append(newTerm)
            willDelete.extend([l, r])
            break
```

한 가지 아쉬운 부분은 고민을 해봐도 분배법칙을 통해 식을 간단하게 변형한 이후 다시 분배법칙을 적용할 수 있는지 결론을 내리지 못했다는 점이다. 그래서 임시방편으로 표현식의 변화가 없을 때까지 분배법칙을 반복해서 적용하도록 했다.

```
while self.distributive():
    self.printExpression('After Apply (X + Y)(X + Z) = X + YZ')
```

분배법칙을 통해 식을 간략하게 만든 이후에는 모든 항을 전개해서 SOP 형태로 만든다. 모든 항을 +로 연결하면 expression 리스트에는 set 하나만 들어가게 된다. 이를 통해 전개가 끝났는지를 판별했다. 중복된 항이 나오더라도 frozenset과 set을 통해 자연스럽게 없어지도록 했다.

```
while len(self.expression) > 1:
    termA = self.expression.pop()
    termB = self.expression.pop()
    newTerm = set()
    for a in termA:
        for b in termB:
            newTerm.add(frozenset(a | b))
    self.expression.append(newTerm)
```

마지막으로 흡수법칙을 적용한다. 위 과정들을 거치고 나면 expression 리스트 안에는 하나의 set이 들어있고, 이 set 안에 frozenset들이 있다. frozenset도 issubset 메소드를 사용할 수 있기 때문에 이를 이용해서 흡수법칙을 적용할 수 있는지 확인하고 삭제해줬다.

```
expression = list(self.expression.pop())
pivot = expression[i]
compare = expression[j]
if pivot.issubset(compare):
    willDelete.append(j)
```

표현식을 SOP 형태로 바꾼 이후에는 각 항마다 cost를 계산해 최적의 항을 고른다.

```
(expression, ) = self.expression
for term in expression:
    cost = 0
    for r in term:
        cost += self.numInput - len(self.table[r][0]['minterm']) ** 1/2 + 1
    if len(term) > 1:
        cost += len(term) + 1
```

이전에 찾아둔 EPI들과 Petrick's Method로 찾은 항을 합친 것이 최종 결과이다. 마지막으로 이렇게 구한 결과가 올바른지 검증한다. 들어올 수 있는 모든 입력에 대해 출력이 참이 되는 output 집합을 구한 뒤 이 집합을 minterm과 dontcare와 비교한다. 우선 minterm은 모두 켜져야 하기 때문에 minterm이 output의 부분집합인지 확인한다. 다음으로 minterm과 dontcare가 아니라면 반드시 꺼져야 하기 때문에 output이 minterm과 dontcare를 합친 집합의 부분집합인지 확인했다.

```
output = []
for num in range(0, 2**self.numInput):
    result = False
    numToBin = bin(num)[2:].zfill(self.numInput)
    for term in self.EPI + self.petrickResult:
        for i in range(self.numInput):
            if term['binary'][i] == '-':
                continue
            if term['binary'][i] != numToBin[i]:
                break
        else:
            result = True
            break
    if result:
        output.append(num)
```

2. 실행

필요한 정보들을 입력한다.

```
Number Of Input : 6
Minterm : 0 1 2 5 10 15 17 19 20 25 26 30 31 33 41 43 44 45 48 51 52 53 54 57 58 59 62
Don't Care : 9 28 35 40 42 50 55 56 60
```

implicant들을 한 번 결합한 결과

# of 1s	Minterm	Binary	Combined
0	{0}	000000	○
1	{1}	000001	○
	{2}	000010	○
2	{5}	000101	○
	{9}	001001	○
	{10}	001010	○
	{17}	010001	○
	{20}	010100	○
	{33}	100001	○
	{40}	101000	○
	{48}	110000	○
3	{19}	010011	○
	{25}	011001	○
	{26}	011010	○
	{28}	011100	○
	{35}	100011	○
	{41}	101001	○
	{42}	101010	○
	{44}	101100	○
	{50}	110010	○
	{52}	110100	○
	{56}	111000	○
4	{15}	001111	○
	{30}	011110	○
	{43}	101011	○
	{45}	101101	○
	{51}	110011	○
	{53}	110101	○
	{54}	110110	○
	{57}	111001	○
	{58}	111010	○
	{60}	111100	○
5	{31}	011111	○
	{55}	110111	○
	{59}	111011	○
	{62}	111110	○

implicant들을 두 번 결합한 결과

# of 1s	Minterm	Binary	Combined
0	{0, 1}	00000-	
	{0, 2}	0000-0	
1	{1, 5}	000-01	
	{1, 9}	00-001	○
	{1, 17}	0-0001	○
	{1, 33}	-00001	○
	{2, 10}	00-010	
2	{9, 25}	0-1001	○
	{9, 41}	-01001	○
	{10, 26}	0-1010	○
	{10, 42}	-01010	○
	{17, 19}	0100-1	
	{17, 25}	01-001	○
	{20, 28}	01-100	○
	{20, 52}	-10100	○
	{33, 35}	1000-1	○
	{33, 41}	10-001	○
	{40, 41}	10100-	○
	{40, 42}	1010-0	○
	{40, 44}	101-00	○
	{40, 56}	1-1000	○
	{48, 50}	1100-0	○
	{48, 52}	110-00	○
	{48, 56}	11-000	○
3	{51, 19}	-10011	
	{25, 57}	-11001	○
	{26, 30}	011-10	○
	{26, 58}	-11010	○
	{28, 30}	0111-0	○
	{28, 60}	-11100	○
	{43, 35}	10-011	○
	{51, 35}	1-0011	○
	{41, 43}	1010-1	○
	{41, 43}	1010-1	○
	{41, 45}	101-01	○
	{41, 57}	1-1001	○
	{42, 43}	10101-	○
	{42, 58}	1-1010	○
	{44, 45}	10110-	○
	{44, 60}	1-1100	○
	{50, 51}	11001-	○
	{50, 54}	110-10	○
	{50, 58}	11-010	○
	{52, 53}	11010-	○
	{52, 54}	1101-0	○
	{52, 60}	11-100	○
	{56, 57}	11100-	○
	{56, 58}	1110-0	○
	{56, 60}	111-00	○
4	{31, 15}	0-1111	
	{30, 31}	01111-	
	{62, 30}	-11110	○
	{59, 43}	1-1011	○
	{51, 55}	110-11	○
	{59, 51}	11-011	○
	{53, 55}	1101-1	○
	{54, 55}	11011-	○
	{62, 54}	11-110	○
	{57, 59}	1110-1	○
	{58, 59}	11101-	○
	{58, 62}	111-10	○
	{60, 62}	1111-0	○

Founded PI

```
{'minterm': {0, 1}, 'binary': '00000-'}
{'minterm': {0, 2}, 'binary': '0000-0'}
{'minterm': {1, 5}, 'binary': '000-01'}
{'minterm': {2, 10}, 'binary': '00-010'}
{'minterm': {17, 19}, 'binary': '0100-1'}
{'minterm': {51, 19}, 'binary': '-10011'}
{'minterm': {31, 15}, 'binary': '0-1111'}
{'minterm': {30, 31}, 'binary': '01111-'}
```

implicant들을 세 번 결합한 결과

# of 1s	Minterm	Binary	Combined
1	{1, 25, 9, 17}	0--001	
	{1, 41, 9, 33}	-0-001	
2	{9, 57, 25, 41}	--1001	
	{10, 26, 42, 58}	--1010	
	{20, 28, 52, 60}	-1-100	
	{33, 35, 43, 41}	10-0-1	
	{40, 41, 42, 43}	1010--	○
	{40, 41, 44, 45}	101-0-	
	{40, 41, 56, 57}	1-100-	○
	{40, 42, 56, 58}	1-10-0	○
	{40, 56, 44, 60}	1-1-00	
	{48, 50, 52, 54}	110--0	○
	{48, 50, 56, 58}	11-0-0	○
	{48, 56, 52, 60}	11--00	○
3	{62, 26, 58, 30}	-11-10	
	{62, 28, 30, 60}	-111-0	
	{43, 35, 59, 51}	1--011	
	{41, 43, 59, 57}	1-10-1	○
	{59, 42, 43, 58}	1-101-	○
	{50, 51, 54, 55}	110-1-	
	{59, 50, 51, 58}	11-01-	
	{62, 50, 58, 54}	11--10	○
	{52, 53, 54, 55}	1101--	
	{62, 52, 54, 60}	11-1-0	○
	{56, 57, 58, 59}	1110--	○
	{56, 58, 60, 62}	111--0	○

```
Founded PI
{'minterm': {1, 25, 9, 17}, 'binary': '0--001'}
{'minterm': {1, 41, 9, 33}, 'binary': '-0-001'}
{'minterm': {9, 57, 25, 41}, 'binary': '--1001'}
{'minterm': {10, 26, 42, 58}, 'binary': '--1010'}
{'minterm': {20, 28, 52, 60}, 'binary': '-1-100'}
{'minterm': {33, 35, 43, 41}, 'binary': '10-0-1'}
{'minterm': {40, 41, 44, 45}, 'binary': '101-0-'}
{'minterm': {40, 56, 44, 60}, 'binary': '1-1-00'}
{'minterm': {62, 26, 58, 30}, 'binary': '-11-10'}
{'minterm': {62, 28, 30, 60}, 'binary': '-111-0'}
{'minterm': {43, 35, 59, 51}, 'binary': '1--011'}
{'minterm': {50, 51, 54, 55}, 'binary': '110-1-'}
{'minterm': {59, 50, 51, 58}, 'binary': '11-01-'}
{'minterm': {52, 53, 54, 55}, 'binary': '1101--'}
```

implicant들을 네 번 결합한 결과

# of 1s	Minterm	Binary	Combined
2	{40, 41, 42, 43, 56, 57, 58, 59}	1-10--	
	{48, 50, 52, 54, 56, 58, 60, 62}	11---0	

```
Founded PI
{'minterm': {40, 41, 42, 43, 56, 57, 58, 59}, 'binary': '1-10--'}
{'minterm': {48, 50, 52, 54, 56, 58, 60, 62}, 'binary': '11---0'}
```

찾은 PI들로 표를 구성한 결과

After Find PI	0	1	2	5	10	15	17	19	20	25	26	30	31	33	41	43	44	45	48	51	52	53	54	57	58	59	62
{'minterm': {0, 1}, 'binary': '00000-'}	○	○																									
{'minterm': {0, 2}, 'binary': '0000-0'}	○		○																								
{'minterm': {1, 5}, 'binary': '000-01'}		○		○																							
{'minterm': {2, 10}, 'binary': '00-010'}			○		○																						
{'minterm': {17, 19}, 'binary': '0100-1'}							○	○																			
{'minterm': {51, 19}, 'binary': '-10011'}								○												○							
{'minterm': {31, 15}, 'binary': '0-1111'}						○							○														
{'minterm': {30, 31}, 'binary': '01111-'}												○	○														
{'minterm': {1, 25, 9, 17}, 'binary': '0--001'}		○					○			○																	
{'minterm': {1, 41, 9, 33}, 'binary': '-0-001'}		○												○	○												
{'minterm': {9, 57, 25, 41}, 'binary': '--1001'}										○					○									○			
{'minterm': {10, 26, 42, 58}, 'binary': '--1010'}					○						○														○		
{'minterm': {20, 28, 52, 60}, 'binary': '-1-100'}									○												○						
{'minterm': {33, 35, 43, 41}, 'binary': '10-0-1'}														○	○	○											
{'minterm': {40, 41, 44, 45}, 'binary': '101-0-'}															○		○	○									
{'minterm': {40, 56, 44, 60}, 'binary': '1-1-00'}																	○										
{'minterm': {62, 26, 58, 30}, 'binary': '-11-10'}											○	○													○		○
{'minterm': {62, 28, 30, 60}, 'binary': '-111-0'}												○															○
{'minterm': {43, 35, 59, 51}, 'binary': '1--011'}																○				○						○	
{'minterm': {50, 51, 54, 55}, 'binary': '110-1-'}																				○			○				
{'minterm': {59, 50, 51, 58}, 'binary': '11-01-'}																				○					○	○	
{'minterm': {52, 53, 54, 55}, 'binary': '1101--'}																					○	○	○				
{'minterm': {40, 41, 42, 43, 56, 57, 58, 59}, 'binary': '1-10--'}															○	○								○	○	○	
{'minterm': {48, 50, 52, 54, 56, 58, 60, 62}, 'binary': '11---0'}																			○		○		○		○		○

```
Founded EPI
{'minterm': {1, 5}, 'binary': '000-01'}
{'minterm': {31, 15}, 'binary': '0-1111'}
{'minterm': {20, 28, 52, 60}, 'binary': '-1-100'}
{'minterm': {40, 41, 44, 45}, 'binary': '101-0-'}
{'minterm': {52, 53, 54, 55}, 'binary': '1101--'}
{'minterm': {48, 50, 52, 54, 56, 58, 60, 62}, 'binary': '11---0'}
```

EPI를 찾고 표를 재구성한 결과

After Find EPI	0	2	10	17	19	25	26	30	33	43	51	57	59
{'minterm': {0, 1}, 'binary': '00000-'}	○												
{'minterm': {0, 2}, 'binary': '0000-0'}	○	○											
{'minterm': {2, 10}, 'binary': '00-010'}		○	○										
{'minterm': {17, 19}, 'binary': '0100-1'}				○	○								
{'minterm': {51, 19}, 'binary': '-10011'}					○						○		
{'minterm': {30, 31}, 'binary': '01111-'}								○					
{'minterm': {1, 25, 9, 17}, 'binary': '0--001'}				○		○							
{'minterm': {1, 41, 9, 33}, 'binary': '-0-001'}									○				
{'minterm': {9, 57, 25, 41}, 'binary': '--1001'}						○						○	
{'minterm': {10, 26, 42, 58}, 'binary': '--1010'}			○				○						
{'minterm': {33, 35, 43, 41}, 'binary': '10-0-1'}									○	○			
{'minterm': {40, 56, 44, 60}, 'binary': '1-1-00'}													
{'minterm': {62, 26, 58, 30}, 'binary': '-11-10'}							○	○					
{'minterm': {62, 28, 30, 60}, 'binary': '-111-0'}								○					
{'minterm': {43, 35, 59, 51}, 'binary': '1--011'}										○	○		○
{'minterm': {50, 51, 54, 55}, 'binary': '110-1-'}											○		
{'minterm': {59, 50, 51, 58}, 'binary': '11-01-'}											○		○
{'minterm': {40, 41, 42, 43, 56, 57, 58, 59}, 'binary': '1-10--'}										○		○	○

```
{'minterm': {0, 2}, 'binary': '0000-0'} Dominate {'minterm': {0, 1}, 'binary': '00000-'}
{'minterm': {1, 25, 9, 17}, 'binary': '0--001'} Dominate {'minterm': {40, 56, 44, 60}, 'binary': '1-1-00'}
{'minterm': {33, 35, 43, 41}, 'binary': '10-0-1'} Dominate {'minterm': {1, 41, 9, 33}, 'binary': '-0-001'}
{'minterm': {62, 26, 58, 30}, 'binary': '-11-10'} Dominate {'minterm': {30, 31}, 'binary': '01111-'}
{'minterm': {62, 26, 58, 30}, 'binary': '-11-10'} Dominate {'minterm': {62, 28, 30, 60}, 'binary': '-111-0'}
{'minterm': {43, 35, 59, 51}, 'binary': '1--011'} Dominate {'minterm': {50, 51, 54, 55}, 'binary': '110-1-'}
{'minterm': {43, 35, 59, 51}, 'binary': '1--011'} Dominate {'minterm': {59, 50, 51, 58}, 'binary': '11-01-'}
```

Row Dominance를 적용한 결과

After Apply Row Dominance	0	2	10	17	19	25	26	30	33	43	51	57	59
{'minterm': {0, 2}, 'binary': '0000-0'}	○	○											
{'minterm': {2, 10}, 'binary': '00-010'}		○	○										
{'minterm': {17, 19}, 'binary': '0100-1'}				○	○								
{'minterm': {51, 19}, 'binary': '-10011'}					○						○		
{'minterm': {1, 25, 9, 17}, 'binary': '0--001'}				○		○							
{'minterm': {9, 57, 25, 41}, 'binary': '--1001'}						○						○	
{'minterm': {10, 26, 42, 58}, 'binary': '--1010'}			○				○						
{'minterm': {33, 35, 43, 41}, 'binary': '10-0-1'}									○	○			
{'minterm': {62, 26, 58, 30}, 'binary': '-11-10'}							○	○					
{'minterm': {43, 35, 59, 51}, 'binary': '1--011'}										○	○		○
{'minterm': {40, 41, 42, 43, 56, 57, 58, 59}, 'binary': '1-10--'}										○		○	○

```
2 Dominate 0
26 Dominate 30
43 Dominate 33
```

Column Dominance를 적용한 결과

After Apply Column Dominance	0	10	17	19	25	30	33	51	57	59
{'minterm': {0, 2}, 'binary': '0000-0'}	○									
{'minterm': {2, 10}, 'binary': '00-010'}		○								
{'minterm': {17, 19}, 'binary': '0100-1'}			○	○						
{'minterm': {51, 19}, 'binary': '-10011'}				○				○		
{'minterm': {1, 25, 9, 17}, 'binary': '0--001'}			○		○					
{'minterm': {9, 57, 25, 41}, 'binary': '--1001'}					○				○	
{'minterm': {10, 26, 42, 58}, 'binary': '--1010'}		○								
{'minterm': {33, 35, 43, 41}, 'binary': '10-0-1'}							○			
{'minterm': {62, 26, 58, 30}, 'binary': '-11-10'}						○				
{'minterm': {43, 35, 59, 51}, 'binary': '1--011'}								○		○
{'minterm': {40, 41, 42, 43, 56, 57, 58, 59}, 'binary': '1-10--'}									○	○

```
Founded EPI
{'minterm': {0, 2}, 'binary': '0000-0'}
{'minterm': {33, 35, 43, 41}, 'binary': '10-0-1'}
{'minterm': {62, 26, 58, 30}, 'binary': '-11-10'}
```

EPI를 찾고 표를 재구성한 결과

After Find EPI	10	17	19	25	51	57	59
{'minterm': {2, 10}, 'binary': '00-010'}	○						
{'minterm': {17, 19}, 'binary': '0100-1'}		○	○				
{'minterm': {51, 19}, 'binary': '-10011'}			○		○		
{'minterm': {1, 25, 9, 17}, 'binary': '0--001'}		○		○			
{'minterm': {9, 57, 25, 41}, 'binary': '--1001'}				○		○	
{'minterm': {10, 26, 42, 58}, 'binary': '--1010'}	○						
{'minterm': {43, 35, 59, 51}, 'binary': '1--011'}					○		○
{'minterm': {40, 41, 42, 43, 56, 57, 58, 59}, 'binary': '1-10--'}						○	○

```
{'minterm': {10, 26, 42, 58}, 'binary': '--1010'} Dominate {'minterm': {2, 10}, 'binary': '00-010'}
```

Row Dominance를 적용한 결과

After Apply Row Dominance	10	17	19	25	51	57	59
{'minterm': {17, 19}, 'binary': '0100-1'}		○	○				
{'minterm': {51, 19}, 'binary': '-10011'}			○		○		
{'minterm': {1, 25, 9, 17}, 'binary': '0--001'}		○		○			
{'minterm': {9, 57, 25, 41}, 'binary': '--1001'}				○		○	
{'minterm': {10, 26, 42, 58}, 'binary': '--1010'}	○						
{'minterm': {43, 35, 59, 51}, 'binary': '1--011'}					○		○
{'minterm': {40, 41, 42, 43, 56, 57, 58, 59}, 'binary': '1-10--'}						○	○

```
Founded EPI
{'minterm': {10, 26, 42, 58}, 'binary': '--1010'}
```

EPI를 찾고 표를 재구성한 결과

After Find EPI	17	19	25	51	57	59
{'minterm': {17, 19}, 'binary': '0100-1'}	○	○				
{'minterm': {51, 19}, 'binary': '-10011'}		○		○		
{'minterm': {1, 25, 9, 17}, 'binary': '0--001'}	○		○			
{'minterm': {9, 57, 25, 41}, 'binary': '--1001'}			○		○	
{'minterm': {43, 35, 59, 51}, 'binary': '1--011'}				○		○
{'minterm': {40, 41, 42, 43, 56, 57, 58, 59}, 'binary': '1-10--'}					○	○

Petrick's Method를 수행한 결과

```
F = (P3 + P1)(P2 + P1)(P3 + P4)(P5 + P2)(P6 + P4)(P5 + P6)

After Apply (X + Y)(X + Z) = X + YZ
F = (P1 + P2P3)(P3P6 + P4)(P5 + P2P6)

After Multiplying Out
F = P2P3P5P6 + P2P3P6 + P2P3P4P5 + P1P4P5 + P1P2P4P6 + P2P3P4P6 + P1P3P5P6 + P1P2P3P6

After Apply X + XY = X
F = P2P3P6 + P2P3P4P5 + P1P4P5 + P1P2P4P6 + P1P3P5P6

Cost of P1P2P4P6 is 25
Cost of P1P3P5P6 is 24
Cost of P1P4P5 is 20
Cost of P2P3P4P5 is 26
Cost of P2P3P6 is 18

P2P3P6 Has Chosen
```

최종 결과를 출력한 모습

```
< Result >
F = A'B'C'E'F + A'CDEF + BDE'F' + AB'CE' + ABC'D + ABF' + AB'D'F + A'B'C'D'F' + BCEF' + CD'EF' + BC'D'EF + A'D'E'F + ACD'
```

www.32x4.com 에서 동일한 입력으로 테스트한 결과와 같다.

y = ACD' + ABF' + BCEF' + AB'D'F + CD'EF' + BDE'F' + ABC'D + AB'CE' + A'D'E'F + A'B'C'E'F + A'CDEF + A'B'C'D'F' + BC'D'EF

검산을 한 결과

```
Result Of Verification : [0, 1, 2, 5, 9, 10, 15, 17, 19, 20, 25, 26, 28, 30, 31, 33, 35, 40, 41, 42, 43, 44, 45, 48, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 62]
Is Minterm Covered By Result? : True
Is Result Covered By Minterm + Don't Care? : True
```