도전과제 결과보고서

RD, CD, PM 구현 완료

```
def draw_table(PI, minterm) :
    table = []
    new_PI = []
    sorted_PI = []

    # sort PI :
    for p in PI :
        new_PI.append(p.replace('-','2'))
    new_PI = sorted(new_PI)
    for q in PI :
        sorted_PI.append(q.replace('2','-'))


    for x in sorted_PI :
        check_minterm = []
        for mint in minterm :
            if contain_num(mint, x) :
                check_minterm.append('V')
            else :
                check_minterm.append(".")
        table.append(check_minterm)
    col = minterm
    ind = PI
    con = table
    df = pd.DataFrame(con, columns=col, index = ind)
    print(df)
```

draw_table(PI, minterm)

- 테이블이 어떻게 변하는 지 시각화를 통해 눈으로 확인하고 싶어서 구현한 함수입니다.

- minterm과 PI에 대해 dataframe을 구성하였고 과제를 진행하면서 이를 통해 테이블이 올바르게 변하는지 확인 할 수 있습니다.

RD

```python
def row_dominance(not_EPI_minterm, table) :
    new_table = []
    if len(table) == 1 :
        return not_EPI_minterm, table
    # PI와 해당 PI가 가지고 있는 not_EPI_minterm 값들을 담는 리스트를 만든다.
    for PI in table :
        a = set()
        for mint in not_EPI_minterm :
            if contain_num(mint, PI) :
                a.add(mint)
        new_table.append([PI, a])

    # 지배당하는 PI 제거 하기
    for i in range(len(new_table)-1) :
        for j in range(i+1, len(new_table)) :
            # 길이가 더 작은쪽에 대해서 길이가 더 큰 쪽의 부분집합인지 확인
            if len(new_table[i][1]) > len(new_table[j][1]) :
                if new_table[j][1].issubset(new_table[i][1]) : # issubset() -> 부분 집합인지 확인
                    new_table[j] = [0,set()]
            else :
                if new_table[i][1].issubset(new_table[j][1]) :
                    new_table[i] = [0,set()]

    # 리턴값
    ret_PI = [] # 지배안되는 PI 즉 리턴값
    for ret in new_table :
        if ret == [0,set()] :
            continue
        else :
            ret_PI.append(ret[0])
    # print("-----row dominance result-----")
    # print("해당 minterm: ", not_EPI_minterm, "지배하는 PI목록:", ret_PI)
    return not_EPI_minterm, ret_PI
```

1. PI 별로 어떤 minterm을 갖고 있는지 net_table에 저장해 두었다

2. issubset()을 이용하여 누군가의 부분집합일 경우 new_table 값을 [0,set()]으로 바꿈

3. new_table 속 [0,set()]값인 것들을 제거해주고 row dominance 후 남아있는 minterm과 PI값을 return함

CD

```
def col_dominance(not_EPI_minterm, PI) :
    new_table = []
    if len(not_EPI_minterm) == 1 :
        return not_EPI_minterm, PI

    # 해당 minterm이 가지고 있는 PI 값들을 담는 리스트를 만든다.
    for mint in not_EPI_minterm :
        a = set()
        for P in PI :
            if contain_num(mint, P) :
                a.add(P)
        new_table.append([mint, a])

    # 지배 하는 minterm을 제거한다
    delete_min = []
    for i in range(len(new_table)-1) :
        for j in range(i+1, len(new_table)) :
            # 길이가 더 작은쪽에 대해서 길이가 더 큰 쪽의 부분집합인지 확인
            if len(new_table[i][1]) > len(new_table[j][1]) :
                if new_table[j][1].issubset(new_table[i][1]) :
                    #new_table[i] = [0,set()]
                    delete_min.append(new_table[i])
            else :
                if new_table[i][1].issubset(new_table[j][1]) :
                    #new_table[j] = [0,set()]
                    delete_min.append(new_table[j])

    # 리턴값
    ret_minterm = [] # 지배안되는 PI 즉 리턴값
    for ret in new_table :
        if ret in delete_min :
            continue
        else :
            ret_minterm.append(ret[0])
    # print("-----col dominance result-----")
    # print("해당 minterm: ", ret_minterm, "PI목록:", PI)
    return ret_minterm, PI
```

1. 이번에는 minterm이 어떤 PI에 해당하는지에 대해 리스트를 만든다

2. 이번에는 지배 하는 minterm을 제거한다 row dominance와 다르게 이번에는 상위 집합을 delete_min이라는 리스트에 저장을 해두고 제거를 해주었다.

3. col_dominance이후 table에 남은 minterm과 PI리스트를 리턴했다.

PM

```python
# 식 전개
def Petrick(PI, minterm, EPIs) :
    EPIs.append("after Petrick")
    print("Petrick")
    busket = []
    nickname = []
    cnt = 1
    for x in PI :
        nick = "P" + str(cnt)
        cnt += 1
        nickname.append((nick, x))

    for i in minterm :
        a = []
        for nick, P in nickname :
            if contain_num(i,P) :
                a.append(nick)
        busket.append(a)

    for i in range(1, len(busket)) :
        tmp = mul(busket[0], busket[i])
        busket[0] = tmp

    min_len = 999999 # 가장 적은 경우 뽑음
    for F in busket[0] :
        if len(set(F.split('P'))) < min_len :
            P = set(F.split('P'))
            min_len = len(set(F.split('P')))
    P = list(P)
    #print(P[1:], min_len-1)
    #select_PI = []
    for nick, Pi in nickname :
        if nick[1:] in P[1:] :
            EPIs.append(Pi)

    return PI, minterm, EPIs

def mul(a,b) :
    ret = list()
    for x in a :
        for y in b :
            ret.append(x + y)
    return ret
```

1. 패트릭 방법이후 선택된 것임을 확인하기 위해 "after Petrick"를 EPIs 리스트에 추가해 두었다

2. 그리고 PI들은 '010-'와 같이 생겼기 때문에 뒤의 연산을 편하게 하기위해 nickname을 만들어 주었다. 예를 들어 첫번째 PI 는 P1, 그 다음 부터는 P2, P3, P4 이런 순서대로 닉네임을 지어주었다.

3. mul()함수를 통해 다항식을 전개하였다.

4. mul()함수를 설명하자면 ['ac', 'ad', 'bc', 'bd']

5. 그리고 set함수를 이용하여 P1P2P1 과 같은 연산들은 split와 set을 활용하여 {1,2}로 바꾸어 주었다.

6. 이중 가장 길이가 짧은 set을 찾았고 이 안에 있는 숫자들은 P뒤에 있는 숫자들이므로 이를 활용하여 nickname이전의 본래의 PI모습의 형태로 EPIs에 담아 주었다.

반복을 통한 테이블 간단하게 하기

```
def loop_dominance(minterm) :
    # find all PI and EPI
    PI, EPI = find_PI_EPI(minterm)

    # petrick 예시
    # PI = ['000-','00-0','0-01','0-10', '01-1','011-']
    # EPI = []
    # minterm = [0,1,2,5,6,7]

    EPIs = [] # 선택된 PI들 즉 EPI
    #print("start PI: ", PI,"EPI: ", EPI, "minterm: ", minterm[2:])
    minterm = minterm[2:]
    cnt = 1

    while True :
        # simplify the table
        print()
        print(cnt, "번 반복")
        cnt = cnt + 1
        print("테이블 정보")
        print()
        draw_table(PI, minterm)
        print()
        print()

        NEPI = list(set(PI) - set(EPI)) # EPI를 제거한 PI list -> NEPI
        for sel in EPI :
            EPIs.append(sel)

        EPI_minterm_set = set()
        for x in minterm :
            for y in EPI :
                if contain_num(x,y) :
                    EPI_minterm_set.add(x)

        not_EPI_minterm = list(set(minterm) - EPI_minterm_set)
        if len(not_EPI_minterm) == 0 : PI = []

        #print("EPI에 포함된 minterm제거: ", not_EPI_minterm)
        "EPI를 제거할 때는 EPI의 열과 해당되는 minterm 제거"
        print("EPI 제거 후 테이블 정보")
        print()
        draw_table(NEPI, not_EPI_minterm)
        print("EPI:", EPI)
        print()
        if len(NEPI) == 0 :
            print("NEPI가 존재하지 않음")
            print()
            print()
            PI = []
            minterm = []
            break
```

```
    # Apply column dominance law
    col_minterm, col_PI = col_dominance(not_EPI_minterm, NEPI)
    print("col_dominance 이후 테이블 정보")
    print()
    draw_table(col_PI, col_minterm)
    print()
    print()

    # Apply row dominance law
    row_minterm, row_PI = row_dominance(col_minterm, col_PI)
    print("row_dominance 이후 테이블 정보")
    print()
    draw_table(row_PI, row_minterm)
    print()
    print()

    # 다시 반복하는지
    if PI == row_PI and minterm == row_minterm :
        PI, minterm, EPIs = Petrick(PI, minterm, EPIs)
        break
    elif len(not_EPI_minterm) == 0 :
        PI = []
        print()
        print("테이블에 남아있는 PI, minterm이 없습니다.")
        draw_table(PI, row_minterm)
        print()

        return PI, row_minterm, EPIs
    else :
        PI = row_PI
        minterm = row_minterm
        EPI = find_EPI(minterm, PI)
return PI, minterm, EPIs # 남아있는 PI, minterm 그리고 지금까지 찾은 EPI모음
```

1. 테스트 케이스

```
"""
    1. Petrick 활용
    petrick으로 가는 예시
    과제 형식의 입력 ([변수의 개수, minterm의 개수, minterm #1, minterm #2, ....])
    minterm = [4,6, 0,1,2,5,6,7]
"""
PI, minterm, EPIs =  loop_dominance([4,6, 0,1,2,5,6,7])
print("dominance 후 최종 테이블")
print()
draw_table(PI, minterm)
print()
print("dominance 반복과정을 하며 선택된 EPI들",EPIs)


"""
    2. 예시2
    과제 형식의 입력 ([변수의 개수, minterm의 개수, minterm #1, minterm #2, ....])
    minterm = [4,8,0,4,8,10,11,12,13,15]
"""

PI, minterm, EPIs =  loop_dominance([4,8,0,4,8,10,11,12,13,15])
print("dominance 후 최종 테이블")
print()
draw_table(PI, minterm)
print()
print("dominance 반복과정을 하며 선택된 EPI들",EPIs)

"""
    3. 예시3
    과제 형식의 입력 ([변수의 개수, minterm의 개수, minterm #1, minterm #2, ....])
    minterm = [4, 11, 0,2,5,6,7,8,10,12,13,14,15]
"""

PI, minterm, EPIs =  loop_dominance( [4, 11, 0,2,5,6,7,8,10,12,13,14,15])
print("dominance 후 최종 테이블")
print()
draw_table(PI, minterm)
print()
print("dominance 반복과정을 하며 선택된 EPI들",EPIs)

"""
    4. 예시4
    과제 형식의 입력 ([변수의 개수, minterm의 개수, minterm #1, minterm #2, ....])
    minterm = [5, 20, 0,2,5,6,7,8,10,12,13,14,15, 22,23,24,25,26,27,28,29,30]

"""
PI, minterm, EPIs =  loop_dominance([5, 20, 0,2,5,6,7,8,10,12,13,14,15, 22,23,24,25,26,27,28,29,30])
print("dominance 후 최종 테이블")
print()
draw_table(PI, minterm)
print()
print("dominance 반복과정을 하며 선택된 EPI들",EPIs)
```

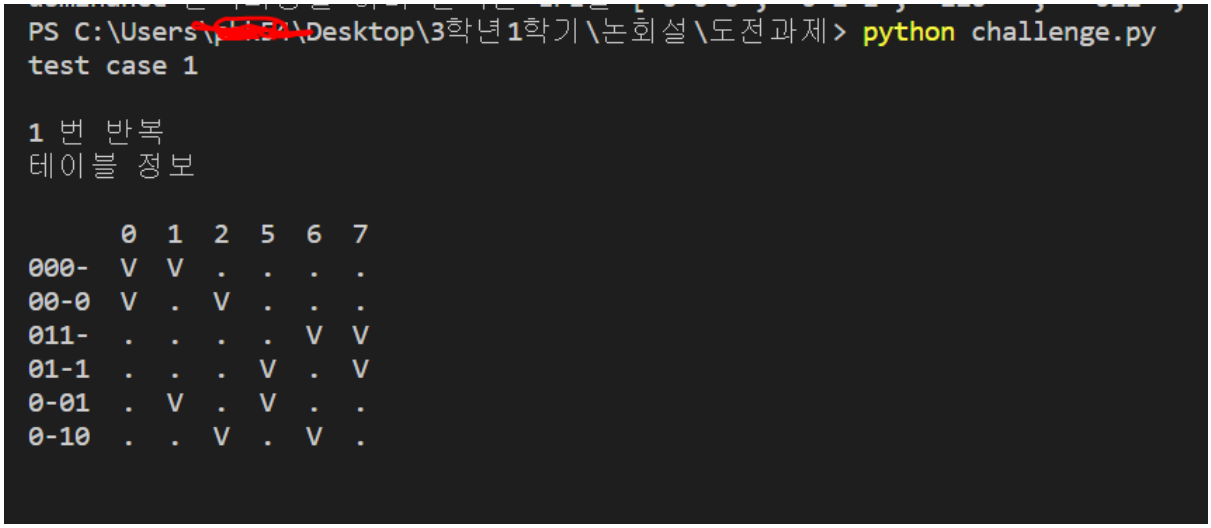
각각 [4,6, 0,1,2,5,6,7], [4,8,0,4,8,10,11,12,13,15], [4, 11, 0,2,5,6,7,8,10,12,13,14,15], [5, 20, 0,2,5,6,7,8,10,12,13,14,15, 22,23,24,25,26,27,28,29,30] 의 테스트 케이스를 사용하였다

테스트 케이스의 형식은 과제와 동일하다.

입력 형식 : ([변수의 개수, minterm의 개수, minterm #1, minterm #2, ...])

테스트 1 결과 [4,6, 0,1,2,5,6,7]

가장 minterm에 대한 PI 정보이다.



EPI, CD, RD가 적용이 되지 않는 것을 눈으로 확인 할 수 있다.

```
2 번 반복
테이블 정보

     0  1  2  5  6  7
011-  .  .  .  .  V  V
0-10  .  .  V  .  V  .
00-0  V  .  V  .  .  .
0-01  .  V  .  V  .  .
01-1  .  .  .  V  .  V
000-  V  V  .  .  .  .


EPI 제거 후 테이블 정보

     0  1  2  5  6  7
011-  .  .  .  .  V  V
0-10  .  .  V  .  V  .
00-0  V  .  V  .  .  .
0-01  .  V  .  V  .  .
01-1  .  .  .  V  .  V
000-  V  V  .  .  .  .
EPI: []

col_dominance 이후 테이블 정보

     0  1  2  5  6  7
011-  .  .  .  .  V  V
0-10  .  .  V  .  V  .
00-0  V  .  V  .  .  .
0-01  .  V  .  V  .  .
01-1  .  .  .  V  .  V
000-  V  V  .  .  .  .


row_dominance 이후 테이블 정보

     0  1  2  5  6  7
011-  .  .  .  .  V  V
0-10  .  .  V  .  V  .
00-0  V  .  V  .  .  .
0-01  .  V  .  V  .  .
01-1  .  .  .  V  .  V
000-  V  V  .  .  .  .
```

다시 반복한 후에도 table이 변하지 않는 것을 확인 할 수 있고

이로 인해 Petrick method의 방법을 활용하여 PI를 선택하게 된다.

```
Petrick
dominance 후 최종 테이블

     0  1  2  5  6  7
011-  .  .  .  .  V  V
0-10  .  .  V  .  V  .
00-0  V  .  V  .  .  .
0-01  .  V  .  V  .  .
01-1  .  .  .  V  .  V
000-  V  V  .  .  .  .

dominance 반복과정을 하며 선택된 EPI들 ['after Petrick', '011-', '00-0', '0-01']
```

최종 결과로는 table은 dominance로는 줄어들지 않았고

Petrick과정을 거친후 '011-' '00-0', '0-01'이 선택된 것을 알 수 있다.

결과 : ['after Petrick', '011-', '0-01', '00-0']

테스트 2 결과 [4,8,0,4,8,10,11,12,13,15]

```
1 번 반복
테이블 정보

     0  4  8  10 11 12 13 15
101-  .  .  .  V  V  .  .  .
10-0  .  .  V  V  .  .  .  .
110-  .  .  .  .  .  V  V  .
11-1  .  .  .  .  .  .  V  V
1-11  .  .  .  .  V  .  .  V
--00  V  V  V  .  .  V  .  .


EPI 제거 후 테이블 정보

     10 11 13 15
101-  V  V  .  .
10-0  V  .  .  .
1-11  .  V  .  V
110-  .  .  V  .
11-1  .  .  V  V
EPI: ['--00']

col_dominance 이후 테이블 정보

     10 11 13 15
101-  V  V  .  .
10-0  V  .  .  .
1-11  .  V  .  V
110-  .  .  V  .
11-1  .  .  V  V


row_dominance 이후 테이블 정보

     10 11 13 15
101-  V  V  .  .
1-11  .  V  .  V
11-1  .  .  V  V
```

처음 반복문을 돌 EPI인 '—00'에 해당하는 PI와 그에 해당하는 minterm을 지운다.

그리고 RD를 거치면서 테이블이 간단해 지는 것을 알 수 있다

RD에서 '110-'이 '11-1'에 포함되고 '10-0'이 '101-'에 포함되어 제거되는 것을 확인 할 수 있다.

```
2 번 반복
테이블 정보

     10 11 13 15
1-11  .  V  .  V
11-1  .  .  V  V
101-  V  V  .  .


EPI 제거 후 테이블 정보

Empty DataFrame
Columns: []
Index: []
EPI: ['101-', '11-1']

col_dominance 이후 테이블 정보

Empty DataFrame
Columns: []
Index: []


row_dominance 이후 테이블 정보

Empty DataFrame
Columns: []
Index: []



테이블에 남아있는 PI, minterm이 없습니다.
Empty DataFrame
Columns: []
Index: []
```

```
dominance 후 최종 테이블

Empty DataFrame
Columns: []
Index: []

dominance 반복과정을 하며 선택된 EPI들 ['--00', '101-', '11-1']
```

2번째 반복을 통해서는 '101-', '11-1' 이 EPI이므로 해당 EPI를 테이블에서 제거하고 minterm도 제거 되므로 table이 비는 것을 확인 할 수 있다.

결과 : ['--00', '101-', '11-1']

테스트 3 결과 [4, 11, 0,2,5,6,7,8,10,12,13,14,15]

```
test case 3

1 번 반복
테이블 정보

     0  2  5  6  7  8  10 12 13 14 15
11--  .  .  .  .  .  .  . V  V  V  V
1--0  .  .  .  .  .  V  V  V  .  V  .
-0-0  V  V  .  .  .  V  V  .  .  .  .
-11-  .  .  .  V  V  .  .  .  .  V  V
-1-1  .  .  V  .  V  .  .  .  V  .  V
--10  .  V  .  V  .  .  V  .  .  V  .


EPI 제거 후 테이블 정보

     12 6  14
11--  V  .  V
--10  .  V  V
1--0  V  .  V
-11-  .  V  V
EPI: ['-0-0', '-1-1']

col_dominance 이후 테이블 정보

     12 6
11--  V  .
--10  .  V
1--0  V  .
-11-  .  V


row_dominance 이후 테이블 정보

     12 6
1--0  V  .
-11-  .  V
```

'-0-0', '-1-1'이 EPI이므로 EPI와 해당 minterm이 테이블에서 제거되는 것을 볼 수 있다.

그리고 CD에서 민텀 14가 6과 12의 superset이므로 col_dominance에 의해서 민텀 14가 테이블에서 사라진 것을 눈으로 확인 할 수 있다.

그 이후에 RD에서 interchangeable한 관계가 나타났음을 col_dominance이후 테이블에서 확인 할 수 있다. 이후에 '1—0', '-11-'이 테이블에 남아 있는 것을 볼 수 있다.

```
2 번 반복
테이블 정보

     12 6
1--0  V  .
-11-  .  V


EPI 제거 후 테이블 정보

Empty DataFrame
Columns: []
Index: []
EPI: ['-11-', '1--0']

NEPI가 존재하지 않음


dominance 후 최종 테이블

Empty DataFrame
Columns: []
Index: []

dominance 반복과정을 하며 선택된 EPI들 ['-0-0', '-1-1', '-11-', '1--0']
```

2번째 반복에서 '1—0', '-11-'이 둘 다 EPI이므로 테이블에서 다 제거되므로 테이블 속에 NEPI가 존재하지 않아 반복수행이 종료되게 된다.

이 과정에서 선택된 PI들은 ['-0-0', '-1-1', '-11-', '1--0']임을 알 수 있다.

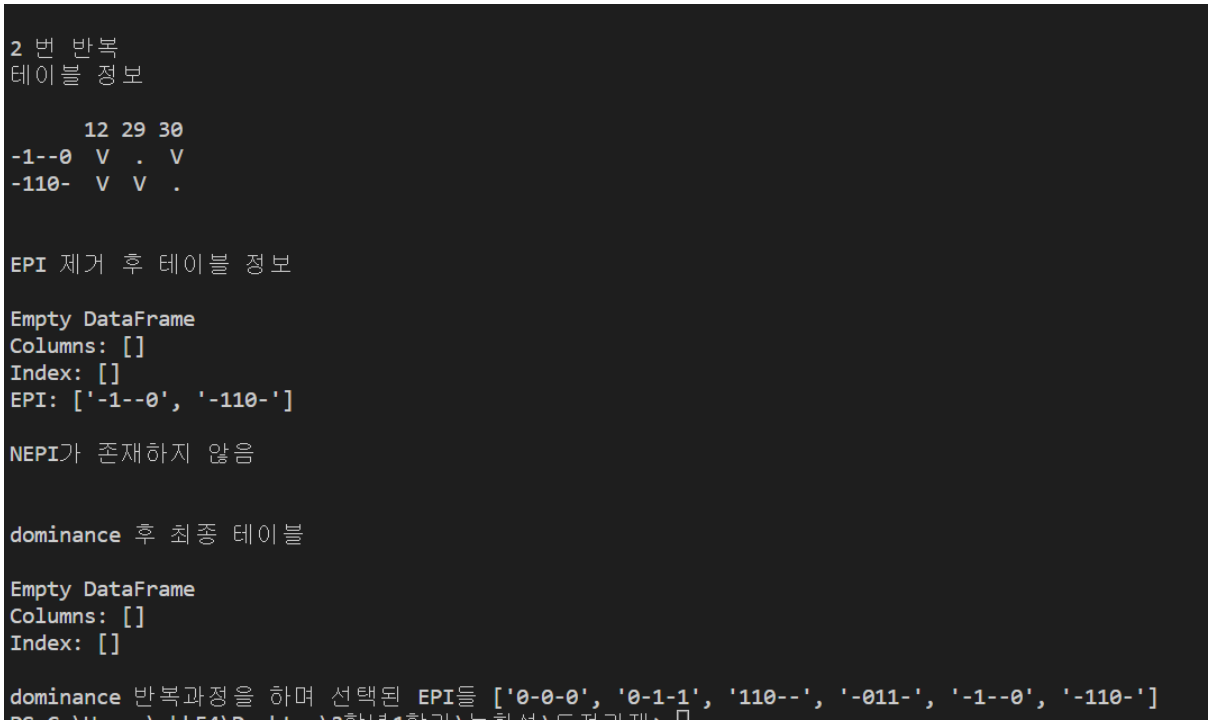
테스트 3 결과 [5, 20, 0,2,5,6,7,8,10,12,13,14,15, 22,23,24,25,26,27,28,29,30]

```
test case 4

1 번 반복
테이블 정보

      0  2  5  6  7  8  10 12 13 14 15 22 23 24 25 26 27 28 29 30
011--  .  .  .  .  .  .  . V  V  V  V  .  .  .  .  .  .  .  .  .
0-0-0  V  V  .  .  .  V  V  .  .  .  .  .  .  .  .  .  .  .  .  .
0-11-  .  .  .  V  V  .  .  .  .  V  V  .  .  .  .  .  .  .  .  .
0-1-1  .  .  V  .  V  .  .  .  V  .  V  .  .  .  .  .  .  .  .  .
0--10  .  V  .  V  .  .  V  .  .  V  .  .  .  .  .  .  .  .  .  .
110--  .  .  .  .  .  .  .  .  .  .  .  .  .  V  V  V  V  .  .  .
11-0-  .  .  .  .  .  .  .  .  .  .  .  .  .  V  V  .  .  V  V  .
-011-  .  .  .  V  V  .  .  .  .  .  .  V  V  .  .  .  .  .  .  .
-110-  .  .  .  .  .  .  .  V  V  .  .  .  .  .  .  .  .  V  V  .
-1--0  .  .  .  .  .  .  V  V  V  .  V  .  .  .  V  .  V  .  V  .  V
--110  .  .  .  V  .  .  .  .  .  V  .  V  .  .  .  .  .  .  .  V

EPI 제거 후 테이블 정보

      12 14 28 29 30
011--  V  V  .  .  .
-1--0  V  V  V  .  V
11-0-  .  .  V  V  .
0-11-  .  V  .  .  .
--110  .  V  .  .  V
0--10  .  V  .  .  .
-110-  V  .  V  V  .
EPI: ['0-0-0', '0-1-1', '110--', '-011-']

col_dominance 이후 테이블 정보

      12 29 30
011--  V  .  .
-1--0  V  .  V
11-0-  .  V  .
0-11-  .  .  .
--110  .  .  V
0--10  .  .  .
-110-  V  V  .


row_dominance 이후 테이블 정보

      12 29 30
-1--0  V  .  V
-110-  V  V  .
```

첫 테이블 이후 해당 EPI인 '0-0-0','0-1-1','110—','-011-'를 테이블에서 지우고 해당하는 민텀에 대해서도 테이블에서 제거해 주었다.

CD에서는 14, 28 라인이 각각 30, 29의 superset이므로 제거되는 것을 볼 수 있다.

또 RD에서는 '011—','11-0-','0-11-'이 '-110-'의 부분집합이고 '—110'이 '-1—0'의 부분집합이여서 제거 되는 것을 확인 할 수 있다.



두번째 반복을 통해서는 해당 테이블의 PI들은 다 EPI이므로 테이블에 있는 PI들이 다 제거되는 것을 볼 수 있다.

그 이후에는 NEPI가 존재하지 않으므로 dominance 후 최종 테이블은 empty한 것을 볼 수 있다.

선택된 EPI 결과로는 ['0-0-0', '0-1-1', '110--', '-011-', '-1--0', '-110-']가 나온 것을 확인 할 수 있다.

결과 : ['0-0-0', '0-1-1', '110--', '-011-', '-1--0', '-110-']