논리회로설계 도전 과제
결과 보고서

1. 서론

전체적인 구현 설계는 EPI를 구하고 EPI에 해당하는 MINTERM , PI (세로,가로)를 모두 지워주고 시작을 한다. COLUM DOMINANCE, ROW DOMINANCE를 적용시키고 , EPI를 한번 더 적용시키는 것이 1번의 순환이며 루틴이다. 이 반복을 PM을 구현하지 않았다는 가정 하에 PI와 MINTERM이 모두 없어질 때 까지 반복하는 것이 구현 목적이었다. (TEST CASE는 변수의 길이가 4,민텀 개수 9 , 해당 민텀 2,3,5,7,8,10,12,13,15)

2. 본론

EPI를 먼저 구한다.

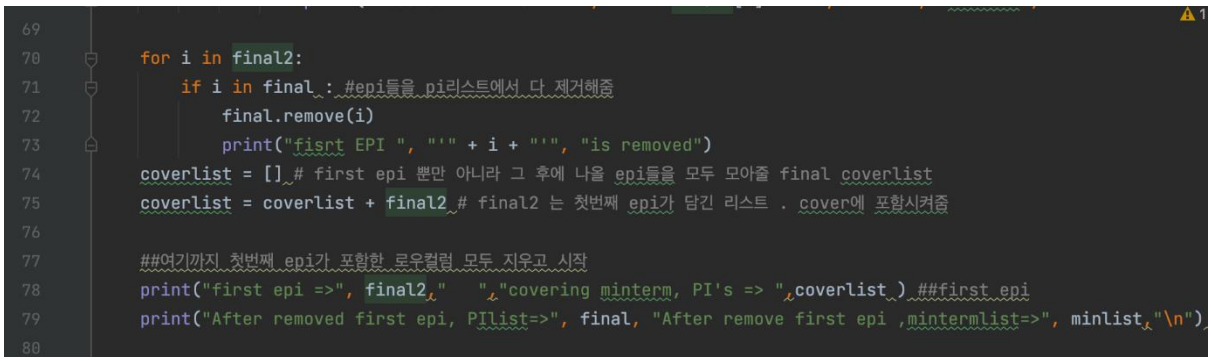
```
39
40      print("start pi's ",final,"minterms",answer2)
41      print("-----------------------------------------------------------------------------------------------------------------------------------")
42      print("EPI Checking and Using EPI to remove ")
43      final2 = [] ## EPi를 담을 리스트
44      minlist =[] ## minterm들을 담을 리스트
45      for i in answer2:
46          minlist.append(i) # answer2는 민텀을 이진수로 바꾼 것들을 넣어둔 배열 -> 그대로 minlist에 넣어줌
47
48      for i in answer2:
49          epi = []
50          mt = i
51          for j in final: #final은 그냥 pi들을 담은 리스트 final2는 epi를 담은 리스트
52              s = ""
53              for l in range(len(i)):
54                  if j[l] == '-':
55                      s += i[l]
56                  else:
57                      s += j[l]
58              if i == s: # 한글자씩 비교하여 빈 스트링에 넣어주고 '-'는 민텀의 그 자리 숫자로 바꾸어줌
59                  epi.append(j) # 같으면 포함되는 것 (체크) 이므로 epi라는 배열에 넣어줌
60
61          if len(epi) == 1:  #이진수 민텀하나와 pi들을 차례로 하나씩 다 비교한 후 epi배열을 봤을때 길이가 1이면 담당하는 pi가 하나라는 것, 즉 epi
62              if epi[0] not in final2: #중복을 제거하기 위한 코드
63                  final2.append(epi[0])
64          for k in range(0,len(final2)):
65              if final2[k] in epi:
66                  if mt in minlist: #epi를 확인하는 배열에서 epi를 가지고있다면 그 때의 민텀들도 싹다 지워줌
67                      minlist.remove(mt)
68                      print("Because of first EPI", "'" + final2[k] + "'", "====>", "Minterm", "'" + mt + "'","is removed ")
69
```

40번 라인에서는 그 전에 구한 PI와 MINTERM리스트를 출력해준다.

그 후에 EPI를 담을 리스트와 MINTERM을 담을 리스트를 따로 준비를 해준다 .

그 후에 MINTERM과 PI들을 하나하나씩 ,그리고 한 글자씩 비교를 해준다 . 그 전에 새로운 스트링 문자열을 만들어주고 '-'라면 민텀의 자리수를 , 아니라면 PI의 자리수를 스트링에 붙혀준다 . 그 결과 스트링이 MINTERM과 같다면 그 해당하는 PI는 체크가 된 것 임을 알 수 있다. 그리고나서 EPI라는 리스트에 넣어주었는데 이 세로로 보았을 때 이 EPI리스트의 길이가 1이라면 포함하는게 그 PI밖에 없다는 의미이므로 EPI리스트에 들어간 PI는 EPI임이 명확하다. 그래서 그 해당하는 EPI가 중복되어 APPEND될 수 있으므로 중복을 not in을 사용하여 제거해주었다 . 그 후에 이 epi를 체크하는 민텀들은 다 지워줘야하므로 epi를 확인하는 배열에서 epi를 가지고 있다면 다 제거해주었다. 그리고 epi가 지운 minterm들을 출력해주었다 .

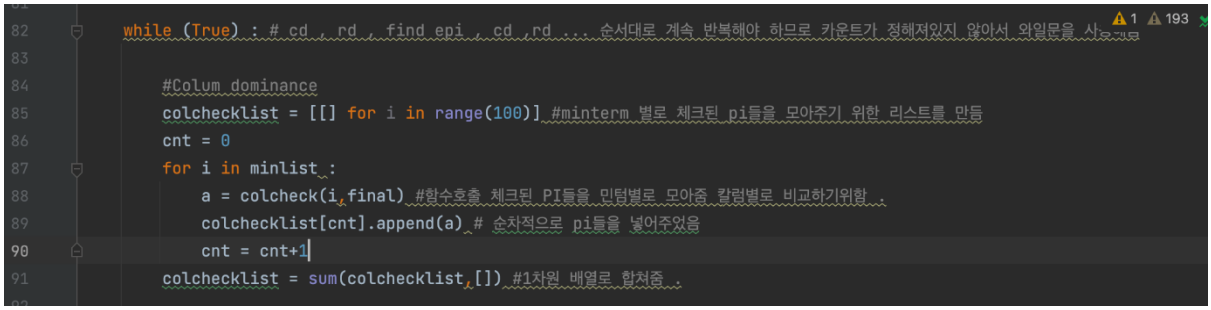
EPI를 PI리스트에서 제거해주었다.



```
69
70      for i in final2:
71          if i in final : #epi들을 pi리스트에서 다 제거해줌
72              final.remove(i)
73              print("fisrt EPI ", "'" + i + "'", "is removed")
74      coverlist = [] # first epi 뿐만 아니라 그 후에 나올 epi들을 모두 모아줄 final coverlist
75      coverlist = coverlist + final2 # final2 는 첫번째 epi가 담긴 리스트 . cover에 포함시켜줌
76
77      ##여기까지 첫번째 epi가 포함한 로우컬럼 모두 지우고 시작
78      print("first epi =>", final2,"   ","covering minterm, PI's => ",coverlist ) ##first epi
79      print("After removed first epi, PIlist=>", final, "After remove first epi ,mintermlist=>", minlist,"\n")
80
```

Coverlist는 first epi와 그 후에 second, third epi. … 들을 모두 모아주는 final cover이다 . epi가 나올 때 마다 coverlist에 append해주었다. 그리고 첫번째 epi를 출력해주고 cover를 출력해주었다 . 또한 epi가 지워지고 난 후의 pi리스트와 minterm리스트를 모두 출력해주었다.

무한루프를 cd,rd,find epi로직으로 반복시켜 돌려주었다. 카운트가 정해져있지 않다.



```
82      while (True) : # cd , rd , find epi , cd ,rd .... 순서대로 계속 반복해야 하므로 카운트가 정해져있지 않아서 와일문을 사용
83
84          #Colum dominance
85          colchecklist = [[] for i in range(100)] #minterm 별로 체크된 pi들을 모아주기 위한 리스트를 만듬
86          cnt = 0
87          for i in minlist :
88              a = colcheck(i,final) #함수호출 체크된 PI들을 민텀별로 모아줌 칼럼별로 비교하기위함 .
89              colchecklist[cnt].append(a) # 순차적으로 pi들을 넣어주었음
90              cnt = cnt+1
91          colchecklist = sum(colchecklist,[]) #1차원 배열로 합쳐줌 .
```

함수호출을 고민하다가, 손으로 풀면서 생각한게 어쩌면 이건 나올 때 까지 하는 것이니까 무한루프를 돌리면 되지않을까 해서 사용하게 되었다. 구현하다보니 함수호출이 더 효율적 일 수도 있다는 생각이 들었다. 제일 먼저 COLUM DOMINANCE를 구현하였다 .

CD는 함수호출로 체크된 PI들을 민텀별로 칼럼별로 비교하기위해 모아주었다 . 그리고 순차적으로 PI들을 COUNTING을 통해 정렬해주었다 .

위에서 호출한 함수의 내용이다.

Colcheck는 CD에서 사용되는 함수로 minterm값과 pi리스트를 인자값으로 전달시켜 각 minterm들이 체크하고있는 pi들을 구하기 위한 함수이다. 로직은 epi를 구현한 방식과 같게 한글자씩 비교하여 스트링에 넣어주고 그 값을 이진수 minterm과 비교하여 같으면 체크하는 방식으로 구현해주었다 . rowcheck도 동일하다.

```
295  def colcheck(min,final): #각 minterm이 체크하고있는 pi들을 구하기 위한 함수이다 .
296      checklist = []
297      for j in final:
298          s = ""
299          for l in range(len(j)):
300              if j[l] == '-':
301                  s += min[l]
302              else:
303                  s += j[l]
304          if min == s:
305              checklist.append(j)
306      return checklist
307
308  def rowcheck(final,min ) :
309      checklist = []
310      for j in min:
311          s = ""
312          for l in range(len(j)):
313              if final[l] == '-':
314                  s += j[l]
315              else:
316                  s += final[l]
317          if j == s:
318              checklist.append(j)
319      return checklist
320
321
```

이제 CD에서 칼럼별로 체크된 것들을 비교하기 시작한다 . colchecklist에는 칼럼별로 체크하고있는(포함하고있는)PI들이 주어진다.

```
        colrmlist =[] # 지우기위한 리스트를 만듬 remove사용시 인덱스가 꼬여서 마지막에 리스트에 포함된 것들만 지워주기 위함 .
        print("Before CD ,PIlist=>", final, "Before CD, mintermlist=>", minlist)
        for i in range(0, len(colchecklist)):
            for j in range(0,len(colchecklist)):
                if len(colchecklist[i]) == len(colchecklist[j]):  ## 원소개수가 같으면 지배관계가 안나타남 .
                    if i == j: ## 자신이 자신을 검사할때 그냥 넘겨줌
                        continue
                    elif i < j: # 왼쪽에서 오른쪽으로만 검사하면 모든경우 검사가능
                        cnt = 0
                        for k in colchecklist[i]:
                            for p in colchecklist[j]:
                                if k == p:
                                    cnt += 1
                        if cnt == len(colchecklist[j]):  ##interchangable
                            if final[j] not in colchecklist: #interchangable중 하나만 고르기 위하여 . 들어가있으면 어펜드 x
                                print("removed interchangable col", "'" + final[i] + "'", ">-->", "'" + final[j] + "'")
                                rowrmlist.append(final[j]) #뒤에꺼를 임의로 골라줌
                elif len(colchecklist[i]) > len(colchecklist[j]):  ## 앞의 체크리스트가 뒤 체크리스트를 지배할 수 있는 경우
                    cnt= 0 #체크하는 pi가 일치하는 개수
                    for k in colchecklist[i]:
                        for p in colchecklist[j]:
                            if (k == p): #체크하는 pi가 같으면
                                cnt = cnt + 1
                    if len(colchecklist[j]) == cnt : ##체크하는 pi개수가 더 적은 체크리스트 길이와 같으면 포함관계.
                        print("col dominance occured","'"+minlist[i]+"'", ">-->", "'"+minlist[j]+"'")
                        colrmlist.append(minlist[i]) ## checklist[i]의 민텀이 지워져야하므로 i번째가 minlist의 민텀이다.

        for i in colrmlist:
            if i in minlist:
                minlist.remove(i) #cd 후 민텀자체를 세로로 다 지워줘야함.
```

여기서 KEY POINT는 그 비교하는 , 배열의 길이 즉 , 체크의 개수이다. 체크된 개수가 같으면 지배관계가 나타나지 않고 INTERCHANGABLE이 나타나며 아닐 경우 지배관계가 나타날 수 있다. 또한 두 배열을 비교했을 때 같은 체크된 것들이 같은 경우 (원소가 같은 경우) 카운트를 해주어 만약 그 카운트가, 원소의 개수가 적은 민텀의 배열길이와 같으면 그것은 지배관계가 나타난다. 정리하자면 같은 것의 개수 = 원소가 적은 것의 배열길이 라면 지배관계이다. Interchangable의 경우에는 중복을 제거해주고 임의로 뒤의 것들을 제거해주었다 . 그리고 DOMINANCE관계가 나타날 경우에는 지배관계를

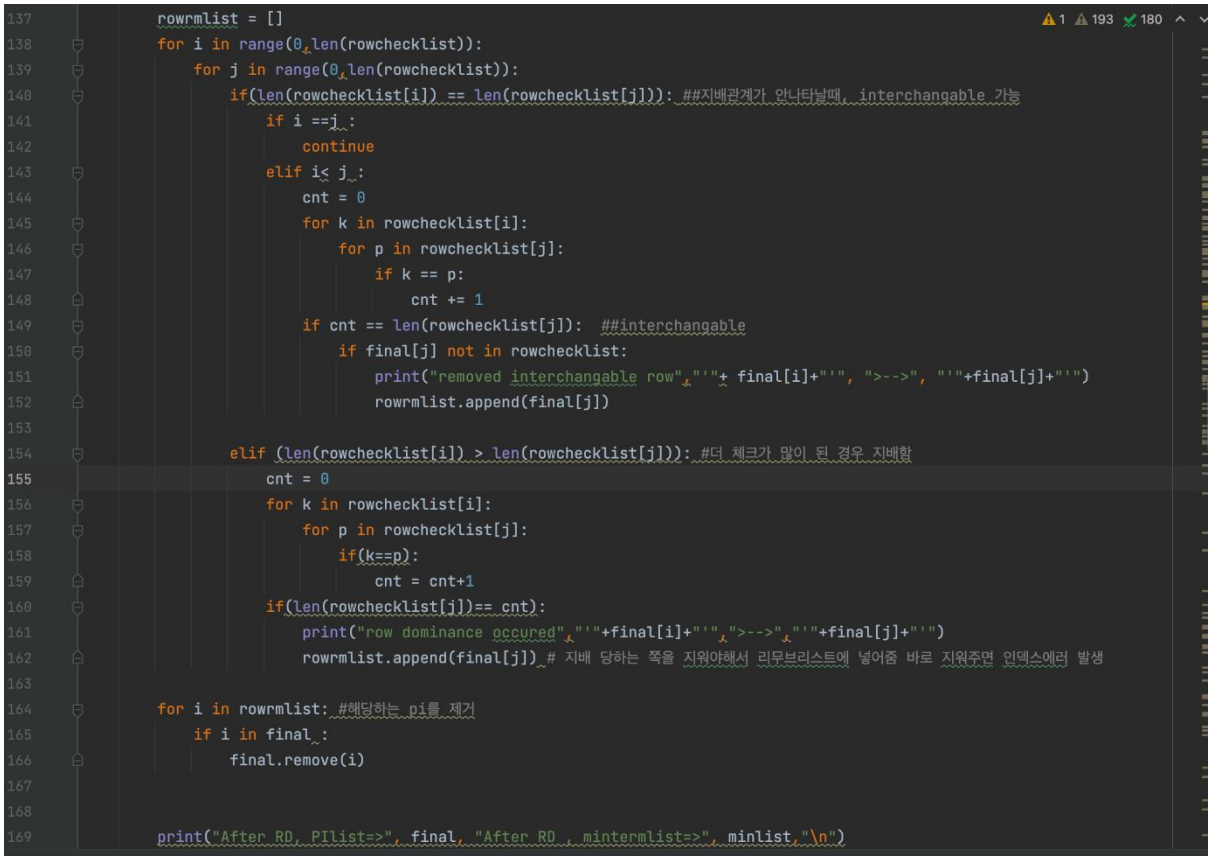
지배하는쪽 >--> 지배당하는쪽으로 출력하여 지배하는 쪽을 colrmlist에 append하여 그 리스트에 들어있는 원소들을 한번에 지워주었다 (인덱스 오류때문)

다음은 ROW DOMINANCE이다 . COLUM DOMINANCE와 같은 로직으로 진행되었다.

```
#ROW dominance
print("Before RD, PIlist=>", final, "Before RD , mintermlist=>", minlist)
rowchecklist = [[] for i in range(100)]
cnt2 = 0
for i in final:  # col dominance와 똑같은 방식으로 구현됨 . 함수호출을 하지않아서 변수이름이 조금씩 상이함 .
    b = rowcheck(i,minlist)
    rowchecklist[cnt2].append(b)
    cnt2 =cnt2 +1
rowchecklist = sum(rowchecklist, [])
```

COLUM DOMINACE와 같이 함수를 호출하였는데 여기서는 CD와는 다르게 pi들과 minterm리스트를 인자값으로 전송하고 minterm들을 돌려받아 rowchecklist에 넣어준다 .

위와 같이 rowrmlist를 만들어 인덱스 오류를 발생시키지 않도록 구현했다 .



```
rowrmlist = []
for i in range(0,len(rowchecklist)):
    for j in range(0,len(rowchecklist)):
        if(len(rowchecklist[i]) == len(rowchecklist[j])): ##지배관계가 안나타날때, interchangable 가능
            if i ==j :
                continue
            elif i< j :
                cnt = 0
                for k in rowchecklist[i]:
                    for p in rowchecklist[j]:
                        if k == p:
                            cnt += 1
                if cnt == len(rowchecklist[j]):  ##interchangable
                    if final[j] not in rowchecklist:
                        print("removed interchangable row","'"+ final[i]+"'", ">-->", "'"+final[j]+"'")
                        rowrmlist.append(final[j])

        elif (len(rowchecklist[i]) > len(rowchecklist[j])): #더 체크가 많이 된 경우 지배함
            cnt = 0
            for k in rowchecklist[i]:
                for p in rowchecklist[j]:
                    if(k==p):
                        cnt = cnt+1
            if(len(rowchecklist[j])== cnt):
                print("row dominance occured","'"+final[i]+"'",">-->","'"+final[j]+"'")
                rowrmlist.append(final[j])# 지배 당하는 쪽을 지워야해서 리무브리스트에 넣어줌 바로 지워주면 인덱스에러 발생

for i in rowrmlist: #해당하는 pi를 제거
    if i in final:
        final.remove(i)


print("After RD, PIlist=>", final, "After RD , mintermlist=>", minlist,"\n")
```

여기서 비교하는 것은 row끼리의 비교이므로 pi들이 가지고있는 minterm check를 비교하는 것이다. 똑같이 배열길이를 이용하여 지배관계를 조건을 달아주었고 CD와는 다르게 지배 당하는 쪽을 지워줘야 하므로 지배당하는 쪽을 rowrmlist에 넣어주고 해당 pi또한 삭제시켜주었다 . 그리고 RD가 적용된 후의 pi와 minterm을 출력해주었다.

RD,CD가 끝난 후에 EPI를 다시 구해준다 .

```
170
171         final3 = []
172         minlist2 = []
173
174         ##epi를 구해야함 .
175         print("checking NEXT EPI .... ")
176         print("Before checking EPI, PIlist=>", final, "Before checking EPI , mintermlist=>", minlist)
177         for i in minlist_:_# rd에서 또 다른 변수 사용
178             minlist2.append(i)
179
180         for i in minlist:
181             epi = []
182             mt = i
183             for j in final:
184                 s = ""
185                 for l in range(len(i)):
186                     if j[l] == '-':
187                         s += i[l]
188                     else:
189                         s += j[l]
190                 if i == s:
191                     epi.append(j)
192
193             if len(epi) == 1:
194                 if epi[0] not in final3:
195                     final3.append(epi[0])
196
197             for k in range(0, len(final3)):
198                 if final3[k] in epi:
199                     if mt in minlist2:
200                         minlist2.remove(mt)
201                         print("Because of first EPI", "'" + final3[k] + "'", "====>", "Minterm", "'" + mt + "'","is removed
202
```

EPI 코드가 실행되기 전 PI,MINTERM을 출력하고 시작한다. 위에서 구현했던 EPI와 같은 방식으로 PI와 MINTERM을 하나하나 한글자씩 비교하여 EPI리스트에 넣어주고 배열 길이가 1이라면 FINAL3이라는 EPI리스트에 넣어주는 방식으로 진행되었다. 위와 동일하게 구해진 EPI로 인해 삭제되는 MINTERM,PI들을 출력해주었다.

다음은 위와 동일한 코드인데 한번 더 작성해주었다.

```
for i in minlist:
    epi = []
    mt = i
    for j in final:
        s = ""
        for l in range(len(i)):
            if j[l] == '-':
                s += i[l]
            else:
                s += j[l]
        if i == s:
            epi.append(j)

    if len(epi) == 1:
        if epi[0] not in final3:
            final3.append(epi[0])

    for k in range(0, len(final3)):
        if final3[k] in epi:
            if mt in minlist2:
                minlist2.remove(mt)
                print("Because of first EPI", "'" + final3[k] + "'", "====>", "Minterm", "'" + mt + "'","is removed

# epi가 표에서 늦게(오른쪽) 등장하면 그 전의 민텀들을 지우지 못한다. 그래서 final3에 epi들을 저장시켜놓고 똑같은 포문을 한번
# 더 돌려주면 왼쪽에 있는 epi를 포함하는 민텀까지 다 지울 수 있다. 비효율적인 방법이긴 하다 .


for i in final3:
    if i in final:
        final.remove(i)
        print("next EPI ","'"+i+"'", "is removed") # epi에 해당하는 pi 지우기
```

그 이유는 세로로 한줄씩 검사를 시작하는데 , 표에서 EPI가 맨 좌측에 나온다면 FINAL3에 EPI가 들어가게 되고 그 이후엔 EPI리스트에 들어간 것이 (CHECK된 것이) 그 중에 FINAL3에 있는 즉 EPI가 포함하고 있는 MINTERM이라면 지워줘야 한다. 하지만 EPI가 맨 첫번째가 아닌 나중에 나오게 된다면 지워지지 않으므로 한번 더 작성해주었다. 그리고 해당 EPI를 PI리스트에서 지워주었다 . 비효율적인 코드였던 것 같지만 다른 방법이 생각이 나지않았던 것 같다.

예외처리와 종료

```
        for i in final3:
            if i in final:
                final.remove(i)
                print("next EPI "+"'"+i+"'", "is removed") # epi에 해당하는 pi 지우기

        if len(final)>0 and len(minlist2) ==0 :# 예외처리 cd를 먼저 하기 때문에 민텀이 다 지워져도 row가 남을때가 생김.
            final.remove(final[0])

        coverlist = coverlist +final3 ## next epi들을 final cover에 포함시켜줌
        print("Next epi= ", final3," covering minterm, PI's", coverlist)
        print("After checking epi ,final=>", final, "After checking epi ,minlist =>", minlist2)

        if(len(final) ==0 and len(minlist2) == 0): ## pilist와 minlist가 다 사라졌으면 끝
            print("FINAL Cover  ==> ", coverlist)
            break

        ## 아직 pi와 MInterm이 남아있다면 계속 와일문을 돌아야함 앞에 minList가 다르므로 다시 minlist를 초기화하고 갱신된 minlist2의 원소를 넣어줌
        if(len(final)>0 and len(minlist2) >0):
            minlist = []
            for i in minlist2:
                minlist.append(i)
            continue
```

COLUM DOMINANCE를 먼저 적용하므로 MINTERM들은 다 지워졌는데 PI가 동그러니 남아있는 경우가 나온다 . 그 경우에는 해당 PI마저 지워주고 종료될 수 있게 해주었다. 그리고 다음 구해진 EPI들을 COVER에 넣어주고 PI , MINTERM들을 다시 출력해주었다. 만약 PI와 MINTERM이 아무것도 없다면 무한루프를 빠져나가게 해주었고 만약 원소들이 남아있다면 계속 CD,RD를 진행해줘야하므로 MINLIST를 갱신해주고 CONTINUE해주었다.

아래 내용은 출력값이다. 맨 윗 부분에서 pi와 minterm을 모두 보여주고 시작한다.

EPI체킹부터 시작하여 EPI로 지워지는 MINTERM, EPI들을 출력해주었다. CD,RD,를 진행하기 전 후로 계속해서 ECHO CHECKING을 위하여 PI와 MINTERM을 출력해주었다.

```
/Library/Frameworks/Python.framework/Versions/Current/bin/python3 /Users/ansuhyeon/PycharmProjects/pythonProject/main.py
start pi's  ['001-', '0-11', '10-0', '110-', '1-00', '-010', '-1-1'] minterms ['0010', '0011', '0101', '0111', '1000', '1010', '1100', '1101', '1111']
----------------------------------------------------------------------------------------------------
EPI Checking and Using EPI to remove
Because of first EPI '-1-1' ====> Minterm '0101' is removed
Because of first EPI '-1-1' ====> Minterm '0111' is removed
Because of first EPI '-1-1' ====> Minterm '1101' is removed
Because of first EPI '-1-1' ====> Minterm '1111' is removed
fisrt EPI  '-1-1' is removed
first epi => ['-1-1']     covering minterm, PI's =>  ['-1-1']
After removed first epi, PIlist=> ['001-', '0-11', '10-0', '110-', '1-00', '-010'] After remove first epi ,mintermlist=> ['0010', '0011', '1000', '1010', '1100']

Before CD ,PIlist=> ['001-', '0-11', '10-0', '110-', '1-00', '-010'] Before CD, mintermlist=> ['0010', '0011', '1000', '1010', '1100']
After CD ,PIlist=> ['001-', '0-11', '10-0', '110-', '1-00', '-010'] After CD, mintermlist=> ['0010', '0011', '1000', '1010', '1100']

Before RD, PIlist=> ['001-', '0-11', '10-0', '110-', '1-00', '-010'] Before RD , mintermlist=> ['0010', '0011', '1000', '1010', '1100']
row dominance occured '001-' >--> '0-11'
row dominance occured '1-00' >--> '110-'
After RD, PIlist=> ['001-', '10-0', '1-00', '-010'] After RD , mintermlist=> ['0010', '0011', '1000', '1010', '1100']

checking NEXT EPI ....
Before checking EPI, PIlist=> ['001-', '10-0', '1-00', '-010'] Before checking EPI , mintermlist=> ['0010', '0011', '1000', '1010', '1100']
Because of first EPI '001-' ====> Minterm '0011' is removed
Because of first EPI '1-00' ====> Minterm '1100' is removed
Because of first EPI '001-' ====> Minterm '0010' is removed
Because of first EPI '1-00' ====> Minterm '1000' is removed
next EPI  '001-' is removed
next EPI  '1-00' is removed
Next epi=  ['001-', '1-00']  covering minterm, PI's ['-1-1', '001-', '1-00']
After checking epi ,final=> ['10-0', '-010'] After checking epi ,minlist => ['1010']
Before CD ,PIlist=> ['10-0', '-010'] Before CD, mintermlist=> ['1010']
After CD ,PIlist=> ['10-0', '-010'] After CD, mintermlist=> ['1010']
```

```
checking NEXT EPI ....
Before checking EPI, PIlist=> ['001-', '10-0', '1-00', '-010'] Before checking EPI , mintermlist=> ['0010', '0011', '1000', '1010', '1100']
Because of first EPI '001-' ====> Minterm '0011' is removed
Because of first EPI '1-00' ====> Minterm '1100' is removed
Because of first EPI '001-' ====> Minterm '0010' is removed
Because of first EPI '1-00' ====> Minterm '1000' is removed
next EPI  '001-' is removed
next EPI  '1-00' is removed
Next epi=  ['001-', '1-00']  covering minterm, PI's ['-1-1', '001-', '1-00']
After checking epi ,final=> ['10-0', '-010'] After checking epi ,minlist => ['1010']
Before CD ,PIlist=> ['10-0', '-010'] Before CD, mintermlist=> ['1010']
After CD ,PIlist=> ['10-0', '-010'] After CD, mintermlist=> ['1010']

Before RD, PIlist=> ['10-0', '-010'] Before RD , mintermlist=> ['1010']
removed interchangable row '10-0' >--> '-010'
After RD, PIlist=> ['10-0'] After RD , mintermlist=> ['1010']

checking NEXT EPI ....
Before checking EPI, PIlist=> ['10-0'] Before checking EPI , mintermlist=> ['1010']
Because of first EPI '10-0' ====> Minterm '1010' is removed
next EPI  '10-0' is removed
Next epi=  ['10-0']  covering minterm, PI's ['-1-1', '001-', '1-00', '10-0']
After checking epi ,final=> [] After checking epi ,minlist => []
FINAL Cover  ==>  ['-1-1', '001-', '1-00', '10-0']
```

위 테스트케이스는 CD는 나오지않았고 RD와 EPI만 값이 도출되었다 . 마지막 FINAL COVER의 값이 잘 나온 것을 볼 수 있었다. 위 test case는 COLUMN DOMINANCE가 나타나지 않는 데 아래의 test case는 col dominance가 나타나서 첨부하였다.

Testcase=> 민텀의 길이가 4이고 민텀의 개수가 11(민텀=>(0,2,5,6,7,8,10,12,13,14,15)

```
/Library/Frameworks/Python.framework/Versions/Current/bin/python3 /Users/ansuhyeon/PycharmProjects/pythonProject/main.py
start pi's  ['11--', '1--0', '-0-0', '-11-', '-1-1', '--10'] minterms ['0000', '0010', '0101', '0110', '0111', '1000', '1010', '1100', '1101', '1110', '1111']
-----------------------------------------------------------------------------------------------------------
EPI Checking and Using EPI to remove
Because of first EPI '-0-0' ====> Minterm '0000' is removed
Because of first EPI '-0-0' ====> Minterm '0010' is removed
Because of first EPI '-1-1' ====> Minterm '0101' is removed
Because of first EPI '-1-1' ====> Minterm '0111' is removed
Because of first EPI '-0-0' ====> Minterm '1000' is removed
Because of first EPI '-0-0' ====> Minterm '1010' is removed
Because of first EPI '-1-1' ====> Minterm '1101' is removed
Because of first EPI '-1-1' ====> Minterm '1111' is removed
fisrt EPI  '-0-0' is removed
fisrt EPI  '-1-1' is removed
first epi => ['-0-0', '-1-1']     covering minterm, PI's =>  ['-0-0', '-1-1']
After removed first epi, PIlist=> ['11--', '1--0', '-11-', '--10'] After remove first epi ,mintermlist=> ['0110', '1100', '1110']

Before CD ,PIlist=> ['11--', '1--0', '-11-', '--10'] Before CD, mintermlist=> ['0110', '1100', '1110']
col dominance occured '1110' >--> '0110'
col dominance occured '1110' >--> '1100'
After CD ,PIlist=> ['11--', '1--0', '-11-', '--10'] After CD, mintermlist=> ['0110', '1100']

Before RD, PIlist=> ['11--', '1--0', '-11-', '--10'] Before RD , mintermlist=> ['0110', '1100']
removed interchangable row '11--' >--> '1--0'
removed interchangable row '-11-' >--> '--10'
After RD, PIlist=> ['11--', '-11-'] After RD , mintermlist=> ['0110', '1100']
```

```
checking NEXT EPI ....
Before checking EPI, PIlist=> ['11--', '-11-'] Before checking EPI , mintermlist=> ['0110', '1100']
Because of first EPI '-11-' ====> Minterm '0110' is removed
Because of first EPI '11--' ====> Minterm '1100' is removed
next EPI  '-11-' is removed
next EPI  '11--' is removed
Next epi=  ['-11-', '11--']  covering minterm, PI's ['-0-0', '-1-1', '-11-', '11--']
After checking epi ,final=> [] After checking epi ,minlist => []
FINAL Cover  ==>  ['-0-0', '-1-1', '-11-', '11--']

Process finished with exit code 0
```


3. 결론

RD와 CD를 해결하면서 사용한 것은 EPI에서 생각해낸 배열의 길이비교 (체크 개수비교)가 크게 작용했던 것 같다. ROW 별로 민텀들을 체크(포함)하는 배열을 만들고 COL 별로 PI들을 체크(포함)하는 배열을 만들어 비교한 것이 전체적인 로직이었던 것 같다.

PM까지는 비록 구현하지 못했지만 CD,RD까지 구현에 성공했다는 것 만으로도 만족스러운 결과였던 것 같다. 효율적으로 로직을 구현하지는 못했어도 열심히 했기 때문에 여기까지 한 것에 대해 의미를 더 높게 두려고 한다.