결과 보고서

1. Row_dominance

```
def Row_dominance(tmp_minterm):
    tmp_minterm.sort(key = len)
    for i in range(len(tmp_minterm) - 1):
        count = 0
        row = i + 1
        pos = 0
        for j in range(len(tmp_minterm) - 1 - i):
            for k in range(len(tmp_minterm[i])):
                if tmp_minterm[i][pos] in tmp_minterm[row]:
                    count += 1
                    pos += 1

            if count == len(tmp_minterm[i]):
                del tmp_minterm[i][:]
                break

            row += 1
            pos = 0
            count = 0

    while 1:
        if [] in tmp_minterm:
            tmp_minterm.remove([])
            continue
        break

    print("Row_dominance :", tmp_minterm)
    return tmp_minterm    # row_dominance후에 남은 row들
```

매개변수로 들어오는 배열은 Remove_EPI_minterm 함수를 통해서 EPI에 해당하는 minterm을 제거하고 남은 minterm을 pi별로 저장하고 있다.

Row_dominance는 가지고 있는 minterm이 적을수록 제거될 가능성이 크기때문에 각 배열의 길이를 기준으로 오름차순으로 정렬시켜준다.

count는 겹치는 minterm의 수이다.
row는 지배하는 배열의 index이다.
pos는 지배당하는 배열의 col값이다.

tmp_minterm[i][pos]값이 tmp_minterm[row]에 있으면 count에 1을 더하고, pos에도 1을 더한다.

만약 count의 값이 tmp_miterm[i]의 길이와 같다면 이것은 tmp_minterm[i]에 있는 모든 minterm을 tmp_minterm[row]가 모두 cover한다는 의미와 같기 때문에 tmp_minterm[i]을 제거해주고, 안쪽 반복을 끝낸다.

while문에서는 for문을 통해 요소가 제거된 빈 배열들을 제거한다.

---

2. Column_dominance

```
def Column_dominance(tmp_minterm):
    check = []
    for i in range(len(tmp_minterm)):
        for j in tmp_minterm[i]:
            if j not in check:
                check.append(j)
    check.sort(key = int)
    col_minterm = [[0]*(len(tmp_minterm)+1) for row in range(len(check))]

    for i in range(len(check)):   #2차 배열의 시작에 check의 요소들을 적어준다.
        col_minterm[i][0] = check[i]

    for i in range(len(tmp_minterm)):       # 해당 minterm을 가지는 pi체크
        for j in range(len(tmp_minterm[i])):
            for k in check:
                if tmp_minterm[i][j] == k:
                    col_minterm[check.index(k)][i+1] = 1
                    break

    col_minterm.sort(key = lambda x : x.count(1))
```

매개 변수로 들어오는 배열은 pi별로 minterm을 저장하고있다.

check는 tmp_minterm배열에 있는 모든 minterm을 중복없이 저장한다.

col_minterm은 남은 pi의 개수보다 1개 많게 col을 설정한다.

col_minterm의 배열들의 첫 번째 요소는 해당 minterm의 binary표현법이다.

tmp_minterm[i]의 요소들을 돌면서 만약 해당 요소들이 check에 있다면 해당 minterm이 check에서 가지는 index를 이용하여 col_minterm에 저장하고, 다음 minterm으로 넘어간다.

for문이 완료되면 col_minterm을 1의 개수를 기준으로 오른차순 정렬한다.

```
for i in range(len(col_minterm) - 1):
    if col_minterm[i] == []:
        continue
    compare_row = 1
    loop = True
    while (loop):
        count = 0
        if i+compare_row == len(col_minterm):
            break
        if col_minterm[i+compare_row] == []:
            compare_row += 1
            continue
        for j in range(len(col_minterm[i]) - 1):
            if ((col_minterm[i][j+1] == 1) and (col_minterm[i+compare_row][j+1] == 1)):
                count += 1
            if count == col_minterm[i].count(1):
                loop = False
                del col_minterm[i+compare_row][:]
                break
        compare_row += 1

while 1:
    if [] in col_minterm:
        col_minterm.remove([])
        continue
    break
```

해당 for문을 통해서 column_dominance가 이루어진다.

col_minterm[i]가 빈 배열이면 다음 반복으로 넘어간다.

compare_row는 지배하는 colmun의 index에 더해주는 값이다.

i와 compare_row가 col_minterm의 길이와 같으면 while문을 종료한다.

지배하는 column이 빈 배열이라면 compare_row에 1을 더해주고 다음 반복으로 넘어간다.

for문을 돌면서 만약 각각의 colmun을 가지는 pi가 같다면 count에 +1을 해준다.

count가 col_minterm[i].count(1)과 같다는 것은 col_minterm[i]가 지배당하는 column이라는 의미임으로 지배하는 column, 즉 col_minterm[i+compare_row]의 모든 요소를 지워주고, for문을 종료한다.

밑의 while문에서는 for문을 통해 요소가 제거된 빈 배열들을 제거한다.

```
    check.clear()   #배열 재사용
    for i in range(len(col_minterm[0])-1):  #배열에 남은 pi 갯수만큼 빈 배열 추가
        check.append([])

    for i in range(len(col_minterm)):   #col_minterm돌면서 1인 것을 check에 추가.
        for j in range(len(col_minterm[i])- 1):   #j는 pi의 번호
            if col_minterm[i][j+1] == 1:
                check[j].append(col_minterm[i][0])

    while 1:
        if [] in check:
            check.remove([])
            continue
        break

    print("Column_dominance :", check)

    return check   #Column_dominance 후에 정리된 minterm을 가진 pi
```

이제 사용하지 않는 check를 이용하기 위해서 check를 빈 배열로 만들어줌.

col_minterm에 있는 pi의 개수만큼 check에 빈 배열을 추가한다.

for문을 통해 col_minterm을 돌면서 1이 있으면 해당 pi는 col_minterm[i]의 minterm을 가지고 있다는 의미 임으로, check[j]에 minterm을 추가한다.

for문 종료 후 while문을 통해서 minterm을 하나도 가지고 있지 않은 pi는 제거해준다.

3. Petrick_method

```
def Petrick_method(remove_minterm):
    result = []
    for i in range(len(remove_minterm)):
        for j in range(len(remove_minterm[i])):
            for k in range(len(remove_minterm)-i-1):
                if remove_minterm[i][j] in remove_minterm[i+k+1]:
                    result.append(remove_minterm[i])
                    result.append(remove_minterm[i+k+1])
    return result
```

로 사용된다.
solution함수에서 반환된 result를 사용해 값을 출력한다.

매개변수로 들어오는 배열은 pi별로 minterm을 저장하고 있다.

for문을 돌면서 pi가 가지고 있는 minterm을 다른 pi가 가지고 있으면 result 배열에 두 pi를 추가해준다.

ex) Result의 index 0과 1의 값은 (p1+p2)으

```
if (len(remove_minterm) == 0):
    result = optimization_form(input_count, EPI, second_EPi_list, remove_minterm)   #최적화 폼으로 변환
    print("F =", end=' ')
    for i in range(len(result) - 1):
        print("{} + ".format(result[i]), end='')
    print(result[-1])
else:
    print("Petrick")
    remove_minterm = Petrick_method(remove_minterm)
    remove_minterm = change_form(remove_minterm, input_count)
    result = optimization_form(input_count, EPI, second_EPi_list, remove_minterm)

    print("F =", end=' ')
    for i in range(len(result) - len(remove_minterm)):
        print("{} + ".format(result[i]), end='')
    plus = 0
    for i in range(len(result)-len(remove_minterm), len(result)):
        if i+plus == len(result): break
        print("({} + {})".format(result[i+plus], result[i+1+plus]), end='')
        plus += 1
    return result
```

change_form은 '-'가 포함된 형태로 변환하는 것이다.

optimization_form은 '-'가 포함된 형태를 알파벳으로 변경하는 함수이다.

for문을 통해 optimization_form함수에서 반환된 배열을 식으로 표현한다.

```
def optimization_form(input_count, EPI, second_EPI, remove_minterm):
    alphabet_list = list(ascii_uppercase)
    alphabet_list = alphabet_list[:input_count]
    result = []

    for t in EPI,second_EPI,remove_minterm:
        for i in range(len(t)):
            s = ''.join(alphabet_list)
            count = 0
            for j in range(len(s)):
                if (t[i][j] == '0'):
                    s = s[:j+count+1] + '\'' + s[j+count + 1:]
                    count += 1
                elif t[i][j] == '-':
                    s = s[:j + count] + s[j+count+1:]
                    count -= 1
            result.append(s)

    return result
```

EPI, second_EPI, remove_minterm을 돌면서 '0'을 만나면 해당 index뒤에 `를 붙혀준다.

'-'를 만나면 해당 index에 있는 값을 지워준다.