# 결과보고서

## f 함수

PI가 입력되면 해당 PI가 나타내는 minterm들을 리스트로 리턴함

| 입력 | 출력 |
| --- | --- |
| f('0--1', 0) | [1, 3, 5, 7] |
| f('-00-', 0) | [0, 8, 1, 9] |

```
def f(x, cur):
    if x == '':
        return [cur]
    if x[0] == '-':
        return f(x[1:], cur*2) + f(x[1:], cur*2+1)
    else:
        return f(x[1:], cur*2+int(x[0]))
```

## find_epi 함수

n과 PI들을 받고, EPI를 리턴하는 코드

yas에는 i번째 PI가 나타내는 minterm들을 저장함
yas의 각 원소마다 아래의 연산을 해줌

- 이미 해당 minterm을 다른 PI가 cover 하고 있다 == -1저장
- 아니다 처음 cover하는 minterm이다 == i+1저장

를 모든 PI에 대해 연산을 해주고 ang에 0, -1을 제외한 저장된 인덱스들을 ans2에 저장후 리턴

```
def find_epi(n, ans):
    ang = [ 0 for i in range(0,2**n)]
    for i in range(0, len(ans)):
        yas = f(ans[i], 0) # ex) 0--1 = [1, 3, 5, 7]
        for x in yas:
            if ang[x] != 0:
                ang[x] = -1
            else:
                ang[x] = i+1

    ans2 = []
    for i in ang:
        if i > 0 and i-1 not in ans2:
            ans2.append(i-1)
```

```
    ans2.sort()
    return ans2
```

## Column dominance 함수

nepi와 remain을 입력으로 받고 CD연산 후의 남아있는 minterm들을 second_remain으로 리턴

남아있는 minterm들(remain의 원소)끼리 아래의 비교를 해줌

- 현재 자신을 cover하는 minterm들의 집합끼리 비교
    - set_i가 set_j를 dominate한다 → check[j] = 1
    - set_j가 set_i를 dominate한다 → check[i] = 1

해당 연산을 모두 실행한 후 마지막에 check[i] == 0인 remain[i]만 저장 후 리턴

```
# Column dominance
# 각각의 PI에 대해서 하나의 minterm이 다른 하나의 minterm에 완전히 겹친다면 작은 것만 가져감
def column_dominance(nepi, remain):
    nn = len(remain)
    check = [0 for i in range(0,nn)]
    cover = [f(pi,0) for pi in nepi]
    for i in range(0, nn):
        for j in range(i+1, nn):
            set_i = set()
            set_j = set()
            for idx in range(0,len(cover)):
                if remain[i] in cover[idx]:
                    set_i.add(idx)
                if remain[j] in cover[idx]:
                    set_j.add(idx)

            if set_i == set_i & set_j:   # i가 j에 포함된다!
                check[j] = 1
            elif set_j == set_i & set_j: # j가 i에 포함된다!
                check[i] = 1

    second_remain = []
    for i in range(0,nn):
        if check[i] == 0:
            second_remain.append(remain[i])

    return second_remain
```

## Row_dominance 함수

nepi와 CD 후의 remain을 입력으로 받고, dominate 되지 않은 nepi를 second_epi에 저장 후 리턴

모든 nepi 쌍에 대하여 다음의 연산을 수행

- set_i == 남아 있는 minterm들(remain)중에서 nepi[i]가 cover하는 minterm의 집합
- set_i == set_j라면 size가 작은 nepi 제거
- 한 집합이 다른 집합을 포함한다면 포함되는 집합은 dominate됨

```
# Row dominance
# nepi에 대해서 자신을 dominance하는 다른 nepi 찾기
def row_dominance(nepi, remain):
    nn = len(nepi)
    check = [0 for i in range(0,nn)]
    cover = [f(pi,0) for pi in nepi]
    for i in range(0,nn):
        for j in range(i+1,nn):
            set_i = set(cover[i]) & set(remain)
            set_j = set(cover[j]) & set(remain)

            if set_i == set_j:
                check[i if len(cover[i]) < len(cover[j]) else j] = 1
            elif set_i == set_i & set_j:
                check[i] = 1
            elif set_j == set_i & set_j:
                check[j] = 1

    second_epi = []
    for i in range(0,nn):
        if check[i] == 0:
            second_epi.append(nepi[i])

    return second_epi
```

## 최종 결과

| 입력 | 출력 |
| --- | --- |
| 4 8 0 4 8 10 11 12 13 15 | ['--00', '101-', '11-1'] |
| 4 6 0 1 2 3 4 5 | ['00--', '0-0-'] |
| 5 13 0 2 3 4 5 7 9 14 15 16 20 23 27 | ['01001', '0111-', '11011', '-0111', '-0-00'] |