REPORT

국민대학교
KOOKMIN UNIVERSITY

제목	RD 도전과제
담당교수	주용수
학과	소프트웨어전공
학년	익명
학번	익명
이름	익명
제출일	2022-05-29

1. 구현 코드

```
from itertools import product

def hamming_distance(m1, m2):
    dist = 0

    for n1, n2 in zip(m1, m2):
        if n1 != n2:
            dist += 1
    return dist


def merge(m1, m2, impicants, answer):
    if hamming_distance(m1, m2) == 1:
        merge_str = ""
        for bit in range(len(m1)):
            if m1[bit] == m2[bit]:
                merge_str += m1[bit]
            else:
                merge_str += "2"
        impicants[merge_str.count('1')].append(merge_str)
        if m1 in answer:
            answer.remove(m1)
        if m2 in answer:
            answer.remove(m2)
        if merge_str not in answer:
            answer.append(merge_str)


def solution(minterm):
    answer = []
    n = minterm[0]
    d = minterm[2]
    save = minterm.copy()
    impicants = [[[] for _ in range(n-i+1)] for i in range(n)]
    max_len, term_cnt = f'0{n}b', minterm[1]

    for each_minterm in minterm[3:]:
        s = format(each_minterm, max_len)
        impicants[0][s.count('1')].append(s)
        answer.append(s)

    for total in range(len(impicants)):
        for i in range(len(impicants[total]) - 1):
            for j in range(len(impicants[total][i])):
                for k in range(len(impicants[total][i + 1])):
                    m1, m2 = impicants[total][i][j], impicants[total][i + 1][k]
                    merge(m1, m2, impicants[total+1], answer)
    n -= 1

    answer.sort(key=int)

    pi_map = {}
    to_find_epi_list = []

    for pi in answer:
        minterm = []
        for i in product(["1","0"], repeat=pi.count("2")):
            if '2' in pi:
                replace = pi.replace("2", i[0], 1)
                for j in range(1, len(i)):
                    replace = replace.replace("2", i[j], 1)
                minterm.append(replace)
                to_find_epi_list.append(replace)
                pi_map[pi] = minterm
            else:
                pi_map[pi] = [pi]
                to_find_epi_list.append(pi)
```

```
    to_find_epi_list = [minterm for minterm in to_find_epi_list if
to_find_epi_list.count(minterm) == 1]

    # EPI 찾는 과정
    find_epi_set = set()
    for pi, minterm in pi_map.items():
        for epi in to_find_epi_list:
            if epi in minterm:
                find_epi_set.add(pi)

    # PI 들 중에 Minimum Cover 를 RD 로 찾는 과정

    not_covered = []

    print(f'제거 하기 전의 pi 와 커버하는 minterm 의 상태: {pi_map}\n')

    for each_minterm in save[3:-d]:
        s = format(each_minterm, max_len)
        for epi in find_epi_set:
            if s in pi_map[epi]:
                continue
            else:
                not_covered.append(s)

    print(f'커버되지 않은 minterm: {not_covered}\n')

    for epi in find_epi_set:
        del pi_map[epi]

    print(f'epi 제거 후: {pi_map}\n')


    for pi in list(pi_map):
        if len(set(not_covered) & set(pi_map[pi])) == 0:
            del pi_map[pi]
    print(f'don\' care term 제거 후: {pi_map}\n')

    for pi in list(pi_map):
        pi_map[pi] = set(pi_map[pi]) & set(not_covered)

    print(f'epi 가 커버하는 minterm 제거: {pi_map}\n')

    for pi in list(pi_map):
        for pi2 in list(pi_map):
            if len(set(pi_map[pi]) - set(pi_map[pi2])) == 0 and pi != pi2:
                del pi_map[pi]
                break

    print(f'row dominance 실시 후 상태: {pi_map}\n')

    second_epi = pi_map.keys()
    find_epi_list = list(find_epi_set)
    find_epi_list.sort(key=int)
    answer.append("EPI")
    answer += find_epi_list
    answer.append("Second EPI")
    answer += second_epi

    for i in range(len(answer)):
        answer[i] = answer[i].replace("2", "-")
    return answer



print(solution([4, 8, 2, 0, 4, 8, 10, 11, 12, 13, 15]))
```

구현 과정은 다음과 같다. 우선 1, 2차 과제에서 구한 EPI 코드를 변형하여 사용하는 것을 선택하였는데

EPI를 구할 때 PI들이 커버하는 minterm을 역으로 추적하여 모든 조합의 수를 dictionary에 PI를 key값으로 value에는 커버하는 minterm을 저장 하고나서 모든 key값의 minterm의 출현 빈도수를 카운트 하여 한번만 출현하는 minterm을 커버하는 PI가 EPI라고 판단하는 방식으로 했다. 이를 이용하여 Secondary EPI를 구할 때는 우선 EPI가 커버하는 minterm을 커버한 것으로 보고 not_covered 리스트에 커버가 안된 minterm을 추가한다. 그리고 나서 pi_map에는 각 PI와 PI가 커버하는 minterm이 저장되어 있으므로 일단 EPI를 제거하고 Don't care term을 제거 한다. 그러고 나면 row dominance를 할 수 있을지도 모르는 PI들만 남게 되는데 이때 PI에서 커버하는 minterm중에 EPI가 커버하는 minterm은 고려할 필요가 없으므로 제거한다. 그러면 순수하게 커버되지 않은 minterm들을 포함하는 상태가 나오게 되는데 여기서 pi_map을 이중으로 돌면서 set으로 minterm이 저장되어 있는 리스트를 변환하고 차지합 연산을 통해 어떤 PI가 지배하고 있는지 조건문을 통해 확인하고 지배당한 PI는 제거한다. 이렇게 되면 interchangeable 한 PI가 있어도 맨 처음 마주한 것을 제거하는 방식을 하기 때문에 랜덤으로 뽑을 필요가 없게 된다. 그리고 나면 Secondary EPI가 나오게 되고 마지막에 출력하는 것으로 종료한다.