숫자를 주어진 길이의 2진수로 변환하는 함수

ex) bin_str(4, 4) = "0100"

```
def bin_str(n, l):
    result = bin(n)[2::]
    if len(result) < l:
        result = "0" * (l - len(result)) + result
    return result
```

두 이진수의 해밍 디스턴스가 1인지 확인하는 함수

ex) is_dist1("1011", "1001") = True

```
def is_dist1(a, b):
    count = 0
    for i in range(len(a)):
        if a[i] == "2" and b[i] != "2":
            return False
        if a[i] != b[i]:
            count += 1
        if count > 1:
            return False
    if count == 1:
        return True
    else:
        return False
```

2개의 이진수를 통합하는 함수

ex) combine("1001", "1011") = "1021" ('-' 대신 2를 기준으로 정렬하기 위해 2로 바꿈

```
def combine(a, b):
    result = ""
    for i in range(len(a)):
        if a[i] != b[i]:
            result += "2"
        else:
            result += a[i]
    return result
```

minterm을 key, PI를 value로 하는 딕셔너리를 리턴하는 함수

ex)

minterm = {0, 4, 8, 10, 11, 12, 13, 15}

PI_set = {"10-0"(=p1), "101-"(=p2), "110-"(=p3), "1-11"(=p4), "11-1"(=p5), "—00"(=p6)}

l = 4

ex = pi_cover(PI_set, minterm, l)

ex[0] = {p6}

ex[4] = {p6}

ex[8] = {p1, p6}

ex[10] = {p1, p2}

.

.

.

이런 식으로 딕셔너리 생성

```
def pi_cover(PI_set, minterm, l):
    if len(PI_set) == 0:
        return {}
    result = {}

    for i in minterm:
        result[i] = set()
        for j in PI_set:
            isPI = True
            for k in range(len(j)):
                if j[k] == "2" or j[k] == "-":
                    continue
                else:
                    if j[k] != bin_str(i, l)[k]:
                        isPI = False
                        break
            if isPI == True:
                result[i].add(j)
    return result
```

위와 반대

PI값을 key로, minterm값을 value로 갖는 딕셔너리 리턴

```
def minterm_cover(PI_set, minterm, l):
    if len(minterm) == 0:
        return {}
    result = {}

    for i in PI_set:
        result[i] = set()
        for j in minterm:
            isPI = True
            for k in range(len(i)):
                if i[k] == "2" or i[k] == "-":
                    continue
                else:
                    if i[k] != bin_str(j, l)[k]:
                        isPI = False
                        break
            if isPI == True:
                result[i].add(j)
    return result
```

epi를 찾는 함수

epi를 다 찾으면 PI_set에서 구한 epi를 뺌

찾은 epi값이 들어있는 set을 리턴

```
def find_epi(PI_set, minterm, l):
    PI_covered = pi_cover(PI_set, minterm, l)
    epi = set()
    has_epi = set()
    for i in PI_covered:
        if len(PI_covered[i]) == 1:
            for j in PI_covered[i]:
                epi.add(j)

    for i in epi:
        for j in PI_covered:
            if i in PI_covered[j]:
                has_epi.add(j)

    PI_set -= epi
    minterm -= has_epi

    return epi
```

앞에 집합이 뒤에 집합을 포함하는지 확인하는 함수

두 집합이 같을 때도 True를 리턴

```
def dominance(setA, setB):
    if (len(setA) - len(setB) == len(setA - setB)):
        return True
```

column dominance 함수

PI와 minterm으로 이루어진 두 집합을 인수로 받아서 pi_cover 함수에 넣고 그 결과로 나온value들(각 minterm을 커버하는 pi로 이루어진 set)을 각각 비교하여 둘중 하나가 나머지 하나에 포함되면 포함하는 minterm(pi 양이 많은거)을 제거함. 두 집합이 같을 때도 둘중 하나 제거함. continue 없으면 두 집합이 같을 때 둘 다 제거해서 오류가 발생함. 하나라도 제거한 경우 True를 리턴함

```
def column_dominance(PI_set, minterm, l):
    PI_covered = pi_cover(PI_set, minterm, l)
    remove_column = set()
    minterm_list = list(minterm)
    m = len(minterm)
    for i in range(m - 1):
        for j in range(i + 1, m):
            if dominance(PI_covered[minterm_list[i]],
PI_covered[minterm_list[j]]):
                remove_column.add(minterm_list[i])
                continue
            elif dominance(PI_covered[minterm_list[j]],
PI_covered[minterm_list[i]]):
                remove_column.add(minterm_list[j])

    if len(remove_column) == 0:
        return False
    else:
        minterm -= remove_column
        return True
```

row dominance 함수

PI 값을 key로, 각 PI가 커버하는 minterm으로 이루어진 set을 value로 갖는 딕셔너리를 만들고 value끼리 비교해서 포함되는(작은거) PI를 제거함.

나머지는 column dominance 함수와 같음.

```
def row_dominance(PI_set, minterm, l):
    Minterm_covered = minterm_cover(PI_set, minterm, l)
    remove_row = set()
    PI_list = list(PI_set)
    p = len(PI_set)
    for i in range(p - 1):
        for j in range(i + 1, p):
            if dominance(Minterm_covered[PI_list[i]],
Minterm_covered[PI_list[j]]):
                remove_row.add(PI_list[j])
                continue
            elif dominance(Minterm_covered[PI_list[j]],
Minterm_covered[PI_list[i]]):
                remove_row.add(PI_list[i])
    if len(remove_row) == 0:
        return False
    else:
        PI_set -= remove_row
        return True
```

minterm 입력받고 각 minterm 이진수로 바꿔서 1의 개수를 key로, 각 민턴의 set을 value로 갖는 딕셔너리 num_of1 생성

```
def solution(minterm):
    answer = []
    PI_set = set()

    l = minterm[0]

    minterm = set(minterm[2::])

    num_of1 = {}
    for i in minterm:
        n = bin_str(i, l).count("1")
        if n in num_of1:
            num_of1[n].add(bin_str(i, l))
        else:
            num_of1[n] = set()
            num_of1[n].add(bin_str(i, l))
```

1의 개수가 1개 차이나는 값끼리 서로 비교해서 해밍 디스턴스가 1이면 둘이 합쳐서 next_num_of1에 저장(합친거 1의 개수를 key로하는 딕셔너리), 합쳐지지 않은 값(Combined에 저장되지 않은 값)은 PI_set에 넣어줌.

Combined에 아무것도 들어있지 않으면(합쳐진 값이 없음) break, 하나라도 합쳐졌으면 next_num_of1으로 다시 반복

```
while (True):
    Combined = set()
    next_num_of1 = {}
    for i in range(l):
        j = i + 1
        if i in num_of1 and j in num_of1:
            for before in num_of1[i]:
                for after in num_of1[j]:
                    if is_dist1(before, after):
                        Combined.add(before)
                        Combined.add(after)
                        tem = combine(before, after)
                        n = tem.count("1")
                        if n in next_num_of1:
                            next_num_of1[n].add(tem)
                        else:
                            next_num_of1[n] = set()
                            next_num_of1[n].add(tem)

    for i in num_of1:
        for j in num_of1[i]:
            if j not in Combined:
                PI_set.add(j)
    if len(Combined) == 0:
        break
    num_of1 = next_num_of1
```

epi 찾기, column dominance, row dominance 반복

찾은 epi를 answer에 넣어주고 PI_set에서 찾은 epi 제거

column dominance와 row dominance가 둘다 작동하지 않으면 break, 하나라도 작동했으면 epi 찾기로 돌아가서 다시 반복

```
while(True):
    epi = find_epi(PI_set, minterm, l)
    answer += list(epi)
    PI_set -= epi

    col = column_dominance(PI_set, minterm, l)
    row = row_dominance(PI_set, minterm, l)

    if (col == False) and (row == False):
        break
```

minterm이 모두 제거되지 않았을 때 Petrik's Method 사용(미구현)

```
if len(minterm) != 0:
    #Petrik's Method 미구현
    pass
```

answer를 정렬(지금은 '-' 대신 '2' 들어가있음)후 '2'를 '-'로 교체

answer를 리턴

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

return answer
```

실행

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

실행결과

['101-', '11-1', '--00']