Finding Min-cover

LANGUAGE: PYTHON

Project structure

- mincover
  - product.py
    - minterm, implicants 등 term의 product를 위한 자료형 제공(set 기반)
  - solution.py
    - find_pis(), find_epis(), find_min_cover() 등 문제 풀이 함수 제공
  - main.py
    - 테스트케이스 테스트
  - test.py

product.py

- 각 product의 겹치는 부분과 다른 부분을 마스크, 부분의 형태로 제공하여 implicant 여부 확인 등에 사용

```python
# check overlapping portion of all the terms, and return overlapping mask and portion.
@cached_property
def overlapping(self):
    mask = 2 ** self.size - 1
    portion, *terms = self.terms
    for term in terms:
        mask &= ~(portion ^ term)
    return mask, portion

# return list of non-overlapping portions of all the terms.
@cached_property
def differences(self):
    mask, _ = self.overlapping
    return list(set(map(lambda t: t & ~mask, self.terms)))
```

product.py

- implicant를 '-01-'와 같은 형태로 출력하거나 implicant가 아닌 경우 [0001, 0010] 과 같이 출력

```
def __str__(self):
    if self.is_implicant:
        ovr_mask, ovr_portion = self.overlapping
        return ''.join(['-' if (ovr_mask >> i) & 1 == 0 \
                        else str((ovr_portion >> i) & 1) for i in reversed(range(self.size))])
    return ''.join(['[', *', '.join([bin(t)[2:].rjust(self.size, '0') for t in self.terms]), ']'])
```

- implicant의 정렬값 함수: 자리는 그대로 3진법으로 변환 > 안 겹치는 부분 * 2

```
def sort_key(self, _sort_one_first: bool = None):
    sort_one_first = _sort_one_first if _sort_one_first is not None else Product.sort_one_first
    if self.is_implicant:
        ovr_mask, ovr_portion = self.overlapping
        overlapping = int(bin(ovr_portion)[2:], 3)
        non_overlapping = int(bin((2 ** self.size - 1) ^ ovr_mask)[2:], 3)
        return overlapping * 2 + non_overlapping if sort_one_first else overlapping + non_overlapping * 2
    else:
        return -1
```

product.py

- 해당 product가 implicant인지 확인
  - 다른 부분의 개수(일반적으로 term의 개수와 일치)
  - 겹치는 부분의 길이로부터 얻은 다른 부분의 길이
  - 다른 부분의 개수가 2의 다른 부분의 길이 제곱과 같은 경우 implicant

```
# checks if the Product is a valid implicant.
@cached_property
def is_implicant(self):
    # make sure non-overlapping portion includes all possible cases
    mask, _ = self.overlapping
    diffs = self.differences
    return len(diffs) == 2 ** (self.size - bin(mask)[2:].count('1'))
```

solution.py

- 기존 과제에서 풀었던 것들이라 자세한 설명 생략
  - Product.combineable(): 둘다 implicant이고 차이가 한 비트에서만 나는지 체크

```python
def find_pis(n, minterms: Iterable[int], dterms: Iterable[int] = None):
    if dterms is None:
        dterms = []
    terms = list(map(lambda t: Product([t], size=n), [*minterms, *dterms]))
    implicants = set()
    while terms:
        # find all combined
        combined = {i + j for i in terms for j in terms if i is not j and i.combinable(j)}
        new_implicants = {t for t in terms if all(map(lambda a: t not in a, combined))}
        implicants |= new_implicants
        terms = combined

    return [i for i in implicants if not any(map(lambda j: j != i and i in j, implicants))]


def find_epis(n, minterms: Iterable[int], _pis: Iterable[Product] = None, dterms: Iterable[int] = None):
    pis = _pis if _pis else find_pis(n, minterms, dterms)
    count = {term: sum(1 for _ in pis for t in _ if term == t) for term in minterms}
    # print(count)
    return [i for i in pis if any(map(lambda t: t in count and count[t] == 1, i.terms))]
```

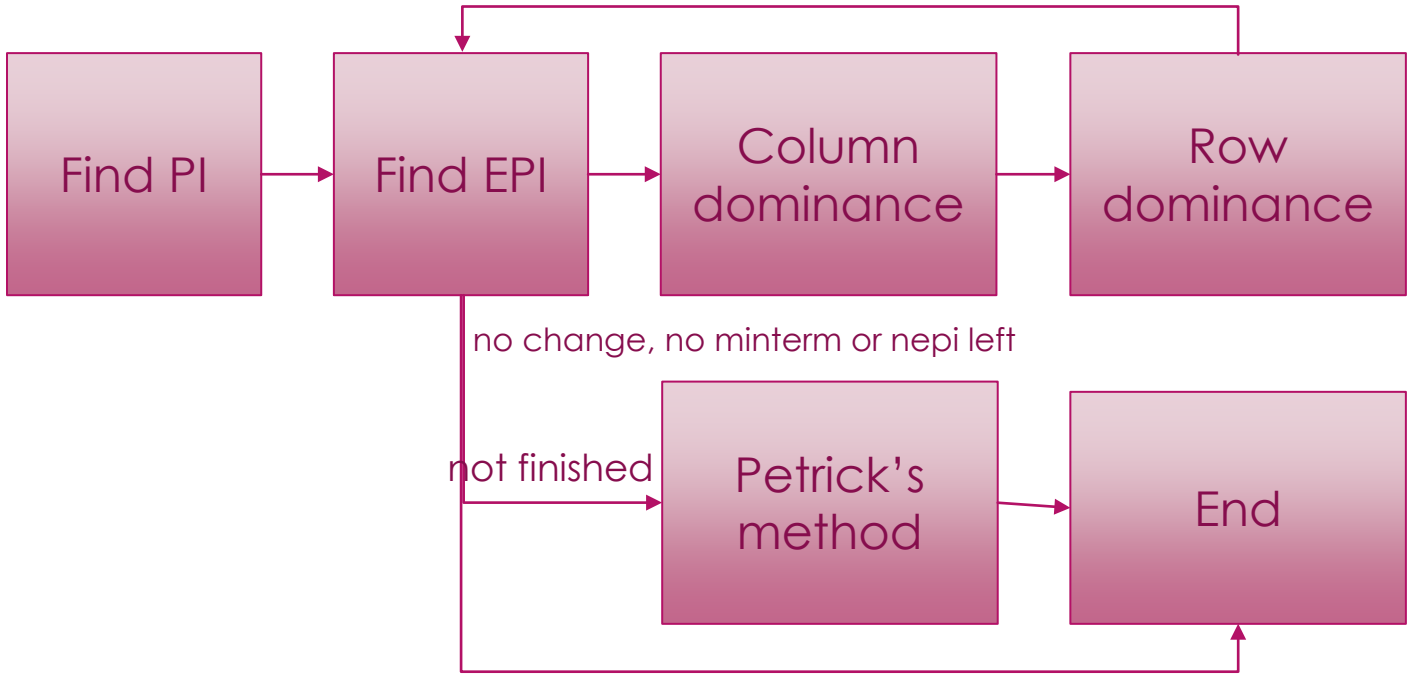
solution.py – 입력과 출력

- n, minterms, dterms을 입력으로 받아 EPI들과 NEPI들을 반환함

```
def find_min_cover(n, _minterms: Iterable[int],
                   dterms: Iterable[int] = None,
```

```
        print_process(5, "Apply Petrick's method", pis=pis, epis=epis, nepis=nepis, minterms=minterms)

    return epis, nepis
```

solution.py – 대략적 과정



solution.py

- 과정 1. find PIs

```
# Process(1): Find all PIs
pis = set(_pis if _pis else find_pis(n, _minterms, dterms))
epis = set()
nepis = set(pis)
minterms = set(_minterms)
```

- 과정 2. find EPIs

```
while True:
    # Process(2): Find all EPIs
    epis |= set(find_epis(n, minterms, pis))
    minterms -= {t for i in epis for t in i}
    nepis -= epis
    if print_detail:
        print_process(2, "Find all EPIs", pis=pis, epis=epis, nepis=nepis, minterms=minterms)
```

```
    # quit if no NEPIs nor minterms remained
    if not nepis or not minterms:
        return epis, nepis
```

solution.py - 과정 3. Column dominance

- 각 minterm과 관련된 PI들을 구해 PI 집합의 포함관계 분석
- 포함되는 것들을 관심있는 minterm에서 제거

```python
def column_dominance(_minterms: Iterable[int], _epis: Iterable[Product], _nepis: Iterable[Product]):
    epis, nepis, minterms = set(_epis), set(_nepis), set(_minterms)
    dominators = {}
    assoc_impls = {t: set(i for i in nepis if t in i) for t in minterms}
    for t in minterms:
        dominator = next((other for other in minterms if t != other and assoc_impls[t].issubset(assoc_impls[other])),
                         None)
        if dominator:
            dominators[t] = dominator
    subservient = {t for t in minterms if t in dominators.keys()}
    minterms -= subservient
    return epis, nepis, minterms
```

solution.py - 과정 4. Row dominance

- 다른 PI의 minterm을 모두 포함하고 cost가 더 작거나 같은 PI들(dominants)를 구하고 필요 없는 PI를 배제

```
def row_dominance(_minterms: Iterable[int], _epis: Iterable[Product], _nepis: Iterable[Product],
                  cost: Callable[[Product], float]):
    epis, nepis, minterms = set(_epis), set(_nepis), set(_minterms)
    dominants = get_row_dominants(nepis, minterms, cost)
    while dominants:
        for d in dominants:
            epis.add(d)
            nepis.remove(d)
            minterms -= d.terms
        if not minterms:
            break
        dominants = get_row_dominants(nepis, minterms, cost)
    return epis, nepis, minterms
```

solution.py - 과정 4. Row dominance

- dominator 중에서 dominate되지 않는 dominator만을 dominant로 선택
  - 포함관계가 계층 구조에서 가장 높은 것 하나만 체크

```
def get_row_dominants(nepis: Set[Product], minterms: Set[int], cost: Callable[[Product], float]):
    dominators = {}
    for i in nepis:
        if not i.terms.intersection(minterms):
            continue
        dominator = next((other for other in nepis if i != other and i.is_dominated_by(other, cost, minterms)), None)
        if dominator:
            dominators[i] = dominator
    # print(dominators)
    # choose dominant implicants those not dominated by others or sharing same cost and same terms
    dominants = [i for i in nepis if i in dominators.values() and
                 (i not in dominators.keys() or dominators[i] in dominators and dominators[
                     dominators[i]] == i and i.sort_key() < dominators[i].sort_key())]
    return dominants
```

solution.py - 과정 5. Petrick's method

- 조건을 만족하는 논리식은 각 minterm을 만족하기 위해 필요한 PI들의 maxterm의 형태로 나타남
- 모든 항을 동시에 만족하는 경우의 수를 구하고 비용이 최소화되는 경우를 선택
  → O(2^n)

```
# for each term and its associated NEPIs, find combinations of NEPIs that satisfies the term
p_maxterms = {term: [any(case & (1 << nepis.index(i)) != 0 for i in impls) for case in range(max_bits)]
             for term, impls in assoc_impls.items()}
# find combinations of NEPIs that satisfies all the terms
p_total = [all(p_maxterms[term][i] for term in minterms) for i in range(max_bits)]
```

```
# optimize cost
optimizing_epis_bit, cost = min(
    ((p, sum(cost(nepis[i]) for i in range(size) if p >> i & 1 == 1)) for p in range(max_bits) if p_total[p]),
    key=lambda x: x[1])
optimizing_epis = {nepis[i] for i in range(size) if optimizing_epis_bit >> i & 1 == 1}
```

실행 예시

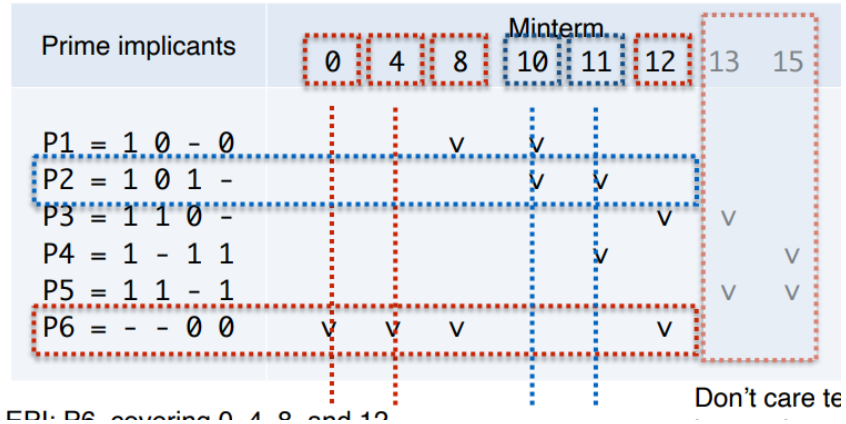
- hw1/hw2 기본 테스트케이스

```
Case 1: n=3, minterms=[0, 1, 2, 3], dterms=None, input_cost=None, output_cost=No
ne, petrick_only=None
default testcase for hw1 / hw2

Process(1): Find all PIs:pis=[0--], epis=EMPTY, nepis=[0--], minterms={0, 1, 2,
3}
Process(2): Find all EPIs:pis=[0--], epis=[0--], nepis=EMPTY, minterms=EMPTY
>> EPIs=[0--], NEPIs=[]
elapsed time: 1.0004043579101562ms

--------------------------------------------------------------------------------
```

실행 예시

- 수업자료 예시(lecture5_tabular_method/p.4~)
  - 수업자료에서와 같이 처음 row dominance를 적용한 후 관심 있는 minterm이 모두 소거된 모습

- Final solution: f = P2+P6 = $x_1x_2'x_3+x_3'x_4'$




```
--------------------------------------------------------------------------------

Case 2: n=4, minterms=[0, 4, 8, 10, 11, 12], dterms=[13, 15], input_cost=None, o
utput_cost=None, petrick_only=None
from class material #1 (lecture5_tabular_method/p.4~)

Process(1): Find all PIs:pis=[--00, 10-0, 101-, 1-11, 110-, 11-1], epis=EMPTY, n
epis=[--00, 10-0, 101-, 1-11, 110-, 11-1], minterms={0, 4, 8, 10, 11, 12}
Process(2): Find all EPIs:pis=[--00, 10-0, 101-, 1-11, 110-, 11-1], epis=[--00],
 nepis=[10-0, 101-, 1-11, 110-, 11-1], minterms={10, 11}
Process(3): Apply column dominance:pis=[--00, 10-0, 101-, 1-11, 110-, 11-1], ep
s=[--00], nepis=[10-0, 101-, 1-11, 110-, 11-1], minterms={10, 11}
Process(4): Apply row dominance:pis=[--00, 10-0, 101-, 1-11, 110-, 11-1], epis=
--00, 101-], nepis=[10-0, 1-11, 110-, 11-1], minterms=EMPTY
Process(2): Find all EPIs:pis=[--00, 10-0, 101-, 1-11, 110-, 11-1], epis=[--00,
101-], nepis=[10-0, 1-11, 110-, 11-1], minterms=EMPTY
>> EPIs=[--00, 101-], NEPIs=[10-0, 1-11, 110-, 11-1]
elapsed time: 2.001523971557617ms

--------------------------------------------------------------------------------
```

실행 예시

- 수업자료 예시 (lecture5_tabular_method/p.17~)
  - 수업자료에서와 같이 column dominance와 row dominance만으로는 min cover를 구하지 못함

  → Petrick's method

Prime implicants	Minterm 0	1	2	5	6	7
P1 = 0 0 - (0,1) a'b'	v	v				
P2 = 0 - 0 (0,2) a'c'	v		v			
P3 = - 0 1 (1,5) b'c		v		v		
P4 = - 1 0 (2,6) bc'			v		v	
P5 = 1 - 1 (5,7) ac				v		v
P6 = 1 1 - (6,7) ab					v	v

```
--------------------------------------------------------------------------------

Case 3: n=3, minterms=[0, 1, 2, 5, 6, 7], dterms=None, input_cost=None, output_
ost=None, petrick_only=False
from class material #2 (lecture5_tabular_method/p.17)

Process(1): Find all PIs:pis=[00-, 0-0, -01, -10, 1-1, 11-], epis=EMPTY, nepis=
00-, 0-0, -01, -10, 1-1, 11-], minterms={0, 1, 2, 5, 6, 7}
Process(2): Find all EPIs:pis=[00-, 0-0, -01, -10, 1-1, 11-], epis=EMPTY, nepis
[00-, 0-0, -01, -10, 1-1, 11-], minterms={0, 1, 2, 5, 6, 7}
Process(3): Apply column dominance:pis=[00-, 0-0, -01, -10, 1-1, 11-], epis=EMP
Y, nepis=[00-, 0-0, -01, -10, 1-1, 11-], minterms={0, 1, 2, 5, 6, 7}
Process(4): Apply row dominance:pis=[00-, 0-0, -01, -10, 1-1, 11-], epis=EMPTY,
nepis=[00-, 0-0, -01, -10, 1-1, 11-], minterms={0, 1, 2, 5, 6, 7}
Process(2): Find all EPIs:pis=[00-, 0-0, -01, -10, 1-1, 11-], epis=EMPTY, nepis
[00-, 0-0, -01, -10, 1-1, 11-], minterms={0, 1, 2, 5, 6, 7}
Process(5): Apply Petrick's method:pis=[00-, 0-0, -01, -10, 1-1, 11-], epis=[0-
, -01, 11-], nepis=[00-, -10, 1-1], minterms=EMPTY
>> EPIs=[0-0, -01, 11-], NEPIs=[00-, -10, 1-1]
elapsed time: 2.002239227294922ms

--------------------------------------------------------------------------------
```

실행 예시

- cf) Petrick's method만 사용
  - 같은 결과

Prime implicants	Minterm 0	1	2	5	6	7
P1 = 0 0 - (0,1) a'b'	v	v				
P2 = 0 - 0 (0,2) a'c'	v		v			
P3 = - 0 1 (1,5) b'c		v		v		
P4 = - 1 0 (2,6) bc'			v		v	
P5 = 1 - 1 (5,7) ac				v		v
P6 = 1 1 - (6,7) ab					v	v

```
--------------------------------------------------------------------------------

Case 4: n=3, minterms=[0, 1, 2, 5, 6, 7], dterms=None, input_cost=None, output_
ost=None, petrick_only=True
from class material #2 (lecture5_tabular_method/p.17) - only with Petrick's met
od

Process(1): Find all PIs:pis=[00-, 0-0, -01, -10, 1-1, 11-], epis=EMPTY, nepis=
00-, 0-0, -01, -10, 1-1, 11-], minterms={0, 1, 2, 5, 6, 7}
Process(2): Find all EPIs:pis=[00-, 0-0, -01, -10, 1-1, 11-], epis=EMPTY, nepis
[00-, 0-0, -01, -10, 1-1, 11-], minterms={0, 1, 2, 5, 6, 7}
Process(5): Apply Petrick's method:pis=[00-, 0-0, -01, -10, 1-1, 11-], epis=[0-
, -01, 11-], nepis=[00-, -10, 1-1], minterms=EMPTY
>> EPIs=[0-0, -01, 11-], NEPIs=[00-, -10, 1-1]
elapsed time: 2.0012855529785156ms

--------------------------------------------------------------------------------
```

실행 방법

- 제공된 Testcase 이용
  - Python main.py [--detail]
- 추가적인 Testcase 이용
  - Python main.py [--detail] [--petrick-only] {<N> <m0>,<m1>,...,<mn> }
- README.md 참고