Summary algorithm

수업시간에 배운 apriori algorithm의 pseudo code를 이용해 구성하였습니다..
코드를 구성한 방식

- Frequent item 들을 frequent item : count 형식인 dictionary 형태로 저장해 나중에 association rule 을 찾을 때 용이하도록 한다.
- List 는 dictionary의 key가 될 수 없어 frequent item 을 tuple 형태로 저장한다.
- 이 때 tuple끼리 비교가 어려우므로 비교할 때는 set의 형태로 변환해서 비교한다.
- 사용한 library

  Itertools.combination : candidate를 만들 때 조합을 이용

  sys : 시스템 입력 변수 처리
- 전역변수

  Transaction : list 형태로 input data를 저장

  minimum_support : 입력받은 support 를 저장

  mimimum_cnt: 입력받은 support를 토대로 transaction 에서 등장해야 하는 횟수를 저장

Detail Each function

프로그램 실행시 시작되는 메인 함수

```
def main():
    global minimum_cnt, minimum_support, transaction
    argv = sys.argv
    minimum_support = float(argv[1]) / 100

    input_file_name = argv[2]
    # input_file_name = 'input.txt'

    output_file_name = argv[3]
    # output_file_name = 'output.txt'

    # 파일을 읽고 각각의 transaction을 list 형태로 저장
    with open(input_file_name, 'r') as f:
        lines = f.readlines()
        for line in lines:
            transaction.append(line.split())

    minimum_cnt = len(transaction) * minimum_support
    # apriori 알고리즘 실행
    result = apriori()

    # 결과를 output file에 저장
    with open(output_file_name, 'w') as f:
        f.write(result)
```

입력 받은 인자들을 변수에 저장한다

```python
# apriori 알고리즘 메인 함수
def apriori():
    # 길이가 1인 초기 fregunt item set 을 세팅
    frequent_item = [init_frequent_item_set()]

    length = 1
    while True:
        prev_frequent_item = frequent_item[length - 1]

        candidate = make_candidate(prev_frequent_item, length + 1)
        frequent = scan_transaction(candidate)
        if len(frequent) == 0:
            break
        else:
            frequent_item.append(frequent)
            length += 1

    # 찾은 frequent item 을 바탕으로 association rule을 찾는다.
    return association_rule(frequent_item)
```

Candidate들이 frequent item이 되지 못하면 loop를 빠져나온다.

```python
# 길이가 1인 frequent item 을 세팅
def init_frequent_item_set():
    item_set = {}

    for transact in transaction:
        for item in transact:
            if item in item_set:
                item_set[item] += 1
            else:
                item_set[item] = 1

    return check_minimum_support(item_set)
```

Item_set에 item이 없으면 1로 등록 있다면 1을 더해준다.

```python
# item_set이 입력받은 minimum support 를 넘는지 확인
def check_minimum_support(item_set):
    frequent_item = {}
    for key, value in item_set.items():
        if value >= minimum_cnt:
            frequent_item[key] = value
    return frequent_item
```

구한 candidate들이 Minmum support를 넘는지 확인하는 일이 많아 따로 함수로 구현

```python
# 다음 후보를 찾는 함수
def make_candidate(frequent_item, length):
    if length == 2:
        return list(combinations(frequent_item, length))
    else:
        element = []
        for items in frequent_item:
            for item in items:
                if item not in element:
                    element.append(item)
        temp_candidate = list(combinations(element, length))
        candidate = []

        set_list = [set(item) for item in frequent_item]
        # Downward Closure
        for items in temp_candidate:
            is_candidate = True
            for subset in list(combinations(items, length - 1)):
                subset = set(subset)
                if subset not in set_list:
                    is_candidate = False
                    break

            # dictionary 키로 사용하기 위해 tuple 형태로 저장
            if is_candidate:
                candidate.append(tuple(items))

        return candidate
```

다음 candidate의 길이가 2면 바로 조합으로 실행

3 이상이라면 frequent_item에 있는 item 들을 한군데로 모은 후 조합을 실행

이 때 downward closure를 이용해 이전에 frequent_item에 없는 부분집합을 가진 item들은 후보에서 제외한다.

```python
# 찾은 후보를 대상으로 transaction을 스캔해 frequent item 인지 확인
def scan_transaction(candidate):
    item_set = {}

    for transact in transaction:
        for item in candidate:
            if set(transact) >= set(item):
                if item_set.get(item) is None:
                    item_set[item] = 1
                else:
                    item_set[item] += 1

    return check_minimum_support(item_set)
```

Set을 이용하면 transaction에 item이 있는지 손쉽게 비교할 수 있다.

```python
# confidence를 찾기위해 frequent item 중 후보의 count를 찾는 부분
def find_confidence(candidate, frequent_item):
    if len(candidate) == 1:
        candidate = candidate[0]
    for items in frequent_item:
        for item in items:
            if set(item) == set(candidate):
                return items.get(item)
```

Key가 tuple 이라 순서가 중요해서 if candidate in items 방식으로 구현하면 찾지 못하는 경우가 생겨 for문으로 구현

```python
# 찾은 frequent item 을 대상으로 association rule을 찾는다.
def association_rule(frequent_item):
    item_length = 1
    transaction_length = len(transaction)
    result = ''
    for item in frequent_item:
        for key, value in item.items():
            length = item_length
            while length > 1:
                candidates = list(combinations(key, length - 1))
                for candidate in candidates:
                    '''
                     candidate(A) 와 association(B) 의 support 는  A의 value or B의 value ( == A+B 의 value)
                     A→B confidence 는 A+B value / A  value
                    '''
                    association = set(map(int, (set(key) - set(candidate))))
                    support = float(value) / transaction_length * 100
                    support = round(support, 2)
                    confidence = float(value) / find_confidence(candidate, frequent_item) * 100
                    confidence = round(confidence, 2)
                    candidate = set(map(int, candidate))
                    result += "{}\t{}\t{:.2f}\t{:.2f}\n".format(str(candidate), str(association), support, confidence)
                length -= 1
        item_length += 1
    return result
```

Frequent item의 등록된 길이가 2이상 item 들에 대하여 부분 집합을 구해 association rule 을 가지는지 확인

Output spec을 맞추기 위해 support 와 confidence를 소수 2번째 자리까지 반올림하고

Candidate -> association을 set(map()) 형태로 변환해서 출력 형태를 맞춰준다.

Instruction for compile

제출한 source code는 .py 형태이므로 python3 명령어를 이용해 실행하면 된다.

Ex) python3 apriori.py 5 input.txt output.txt