

# Apriori implementation assignment

2013011800 구장희

## # Summary of Algorithm

### [Apriori 클래스]

```
class Apriori{
private:
    int length;
    long double minSupport;
    map<vector<int>,long double> C;
    map<vector<int>,long double> L;
    vector<vector<int>> transactions;
    map<int,vector<vector<int>>> frequentPatterns;
    vector<tuple<vector<int>, vector<int>, long double, long double> > associationRules;
public:
    Apriori(long double _minSupport) { ... }

    void readInputFile(string filename) { ... }

    void writeToOutputFile(string filename) { ... }

    string vectorToString(vector<int> vec) { ... }

    void subroutine_readInputFile(string str) { ... }

    void runJob() { ... }

    bool isHaveSameItem(vector<int> small , vector<int> large) { ... }

    bool seekNextLength(vector<int> small , vector<int> large) { ... }

    bool isContains(vector<int> small , vector<int> large) { ... }

    bool isSameVector(vector<int> small , vector<int> large) { ... }

    void generateAssociationRules(vector<int> itemList1 , vector<int> itemList2) { ... }

    void generateC(int length) { ... }

    void joiningAndPrunning() { ... }

    void generateL() { ... }

    long double getSupport(vector<int> itemList) { ... }

};
```

## [알고리즘 요약]

- main 함수에서
  - argv로 minsupport , input.txt , output.txt 를 받는다.
  - Apriori class의 객체 apriori를 생성(생성자의 인자로 minsupport를 준다.)
  - apriori가 input.txt를 한줄(row)씩 읽으면서
    - 각 row를 `vector<int> transaction`으로, (이 과정에서 item을 오름차순으로 정렬)
    - 그 모두를 `vector<vector<int>> transactions` 에 저장한다.
  - apriori가 Apriori algorithm을 수행한다.
    - 처음 길이를 1로 해서
    - while-loop 돌면서
      - 해당 길이의 item list와 그 support를 가지고 C 를 만든다.
      - 만들어진 C.size() ==0 이면 break;
      - C를 가지고 L을 만든다.
      - 길이++;
    - association rule을 만든다.
  - apriori가 output.txt에 associationrule을 쓴다.

## # Detailed description of codes(for each function)

```
class Apriori{
private:
    int length;
    long double minSupport;
    map<vector<int>,long double> C;
    map<vector<int>,long double> L;
    vector<vector<int>> transactions;
    map<int,vector<vector<int>>> frequentPatterns;
    vector<tuple<vector<int>, vector<int>, long double, long double> > associationRules;
public:
```

- Apriori(long double \_minSupport) { ... }
  - priori class의 생성자로 , 시작길이(length)를 1로 설정하고 minsupport를 set한다.
- void readInputFile(string filename) { ... }
  - input file을 읽어서 한 줄씩(string으로) subroutine\_readInputFile(string str)를 호출한다.
- void writeToOutputFile(string filename) { ... }
  - association rule들을 output file에 쓴다.
- string vectorToString(vector<int> vec) { ... }
  - vector를 받아서 string으로 반환한다.
- void subroutine\_readInputFile(string str) { ... }
  - readInputFile의 sub routine으로
  - string(한 줄)씩 인자로 받아서, '\t'을 제외한 integer를 list에 저장한다.
  - 그 list를 오름차순으로 정렬해서 transactions에 저장한다.
- void runJob() { ... }
  - apriori 알고리즘을 수행하는 부분(처음 길이를 1로)
  - while-loop 돌면서
    - 해당 길이의 item list와 그 support를 가지고 C 를 만든다.
    - 만들어진 C.size() ==0 이면 break;
    - C를 가지고 L을 만든다.
    - 길이++;
  - association rule을 만든다.
    - frequentPatterns에 저장된 모든 frequent item list에 길이를 기준으로 두 쌍을 뽑는다.
      - 두 쌍의 길이가 같다면, 각각 선택된 모든 vector<int>의 쌍에 대해서
        - 두 item list가 공통의 item을 가지지 않고 두 list를 cartesian product한 결과의 list가 frequentPatterns에 속한다면
        - 두 item list를 인자로 generateAssociationRules를 호출한다.
      - 두 쌍의 길이가 다르면, 각각 선택된 모든 vector<int>의 쌍에 대해서
        - 길이가 긴 item list가 길이가 작은 item list를 포함하지 않고, 두 list를 합친 list가 frequentPatterns에 속한다면
        - 두 item list를 인자로 generateAssociationRules를 호출한다.
- bool isHaveSameItem(vector<int> small , vector<int> large) { ... }
  - 두 vector<int>를 받아서 , 두 vector가 공통의 아이템을 가지고 있는지 여부를 bool로 반환한다.

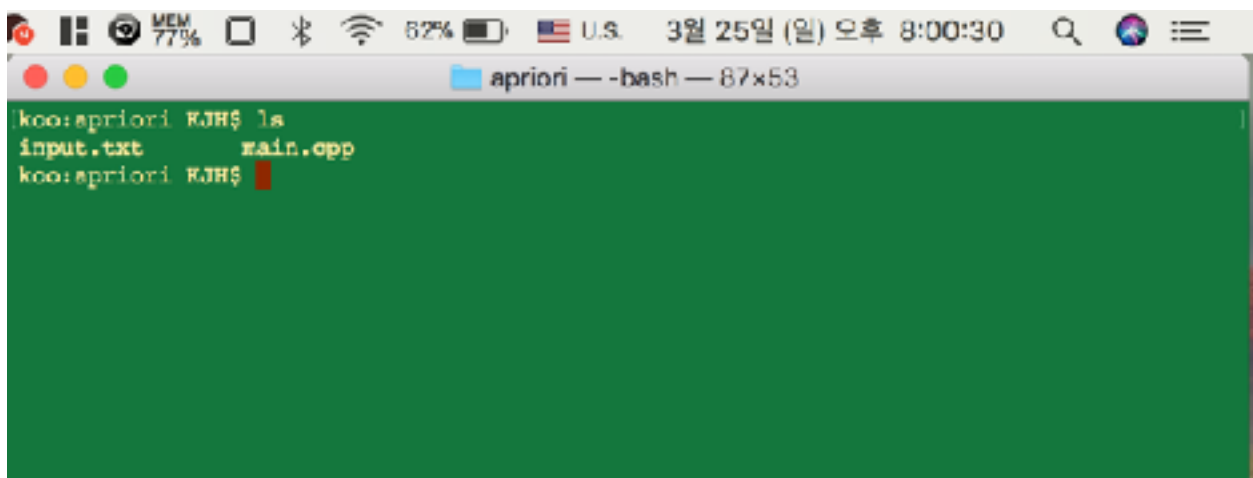
- `bool seekNextLength(vector<int> small , vector<int> large) { ... }`
  - 두 `vector<int>`를 받아서 , 두 `vector`의 cartesian product을 해서 새로운 `vector<int>`를 만들고
  - 그 `vector<int>`가 `frequentPatterns`에 속하는지 여부를 `bool`로 반환한다.
- `bool isContains(vector<int> small , vector<int> large) { ... }`
  - 두 `vector<int>`를 받아서, `large vector<int>`가 `small vector<int>`를 포함하고 있는지 그 여부를 `bool`로 반환한다.
- `bool isSameVector(vector<int> small , vector<int> large) { ... }`
  - 두 `vector<int>`를 받아서, 두 `vector<int>`가 같은지 여부를 `bool`로 반환한다.
- `void generateAssociationRules(vector<int> itemList1 , vector<int> itemList2) { ... }`
  - 두 `vector<int>`를 받아서, 순서를 바꿔가면서
  - association rule를 만든다.({`vector<int>` , `vector<int>` , `long double` , `long double`})
  - 이를 `associationRules`에 넣는다.
- `void generateC(int length) { ... }`
  - 만들 item-list의 길이를 인자로 받아서 그 support와 함께 C를 만든다.
  - 길이가 1일 경우(one-item list)
    - 모든 transaction을 scan하면서 one-item set을 만든다.
    - one-item set에 속한 모든 아이템과 그 support를 C에 넣는다.
  - 길이가 2이상일 경우
    - `joiningAndPrunning()` 호출한다.
- `void joiningAndPrunning() { ... }`
  - 길이가 2 이상의 itemlist에 대해서 joining & prunning을 수행
  - L에서 해당 길이의 모든 item list에 대해서
  - 마지막 item만 다른 두 item list의 쌍을 찾아서 joining 한다.
    - 공통의 item-list에 다른 두개의 item을 정렬해서 새로운 item list를 만든다.
  - joining의 결과로 남은 모든 item list에 대해서 prunning한다.
    - 한 아이템씩 제외한 item list가 L에 있는지 확인하고 있으면 그 item list와 support를 C에 저장.
    - 한 아이템씩 제외한 item list가 L에 있는지 확인하고 없으면 버린다.
- `void generateL() { ... }`
  - C안의 모든 item list-support의 쌍에서 `support<minsupport` 이면 그 item list를 삭제한다.
  - 삭제되지 않은 모든 item list를 길이과 함께 `frequentPatterns`에 저장한다.
- `long double getSupport(vector<int> itemList) { ... }`
  - item list를 인자로 받아서 그 support를 반환한다.
    - itemList보다 길이가 같거나 긴, 모든 transaction에 대해서
    - itemList의 길이, 해당 transaction의 길이만큼 nested for-loop을 돌면서,
    - itemList의 모든 item을 transaction이 포함할 경우 그 횟수(hit)를 increment.
    - `support = (hit/# of transaction)*100.0` 를 반환한다.

};

## # Instructions for compiling source codes at TA's computer

- 환경
  - OS : Mac OS
  - Language : C++11

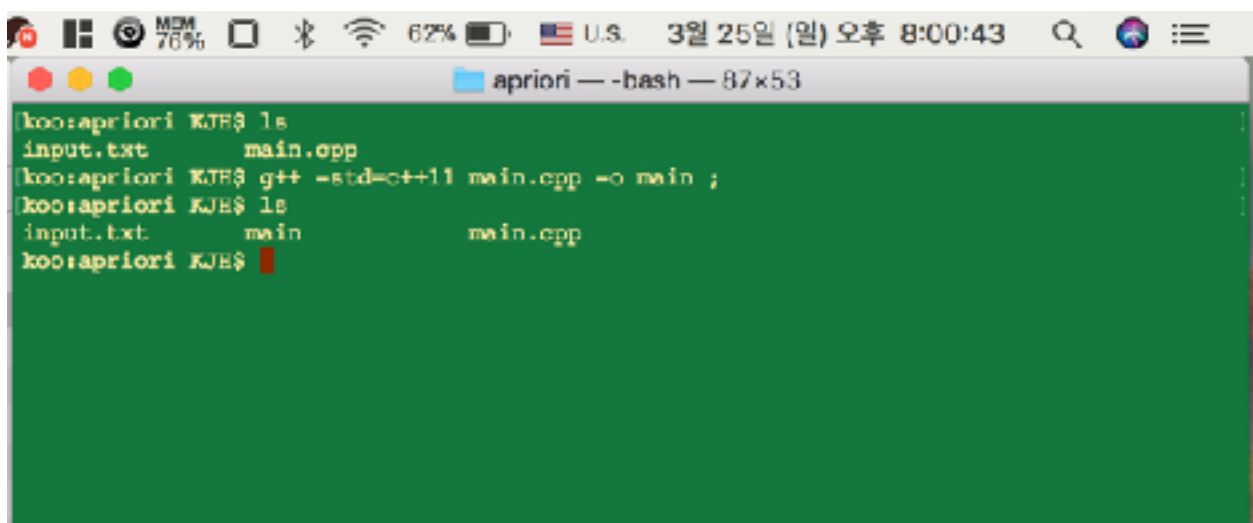
컴파일 이전



```
koo:apriori KJH$ ls
input.txt      main.cpp
koo:apriori KJH$
```

A screenshot of a macOS terminal window titled 'apriori — -bash — 87x53'. The terminal shows the user 'koo' at the directory 'apriori' running the command 'ls'. The output lists 'input.txt' and 'main.cpp'. The prompt 'koo:apriori KJH\$' is shown twice.

컴파일 방법 & 결과 (\$g++ -std=c++11 main.cpp -o main)



```
koo:apriori KJH$ ls
input.txt      main.cpp
koo:apriori KJH$ g++ -std=c++11 main.cpp -o main ;
koo:apriori KJH$ ls
input.txt      main                main.cpp
koo:apriori KJH$
```

A screenshot of a macOS terminal window titled 'apriori — -bash — 87x53'. The terminal shows the user 'koo' at the directory 'apriori' running 'ls', then the compilation command 'g++ -std=c++11 main.cpp -o main ;', and finally 'ls' again. The output after compilation shows 'input.txt', 'main', and 'main.cpp'. The prompt 'koo:apriori KJH\$' is shown four times.

실행( \$ ./main 5 input.txt output.txt)

```
koc:apriori KJH$ ls
input.txt      main.cpp
koc:apriori KJH$ g++ -std=c++11 main.cpp -o main ;
koc:apriori KJH$ ls
input.txt      main                main.cpp
koc:apriori KJH$ ./main 5 input.txt output.txt
koc:apriori KJH$ ls
input.txt      main                main.cpp      output.txt
koc:apriori KJH$
```

결과

```
koc:apriori KJH$ ls
input.txt      main.cpp
koc:apriori KJH$ g++ -std=c++11 main.cpp -o main ;
koc:apriori KJH$ ls
input.txt      main                main.cpp
koc:apriori KJH$ ./main 5 input.txt output.txt
koc:apriori KJH$ ls
input.txt      main                main.cpp      output.txt
koc:apriori KJH$ cat out
cat: out: No such file or directory
koc:apriori KJH$ cat output.txt
{0}    {1}    6.60    24.63
{1}    {0}    6.60    22.15
{0}    {2}    8.60    32.09
{2}    {0}    8.60    32.58
{0}    {3}    5.60    20.90
{3}    {0}    5.60    18.67
{0}    {4}    5.60    20.90
{4}    {0}    5.60    22.76
{0}    {5}    7.40    27.61
{5}    {0}    7.40    29.37
{0}    {6}    6.80    25.37
{6}    {0}    6.80    30.09
{0}    {7}    7.40    27.61
{7}    {0}    7.40    30.83
{0}    {8}    11.80   44.03
{8}    {0}    11.80   26.11
{0}    {9}    7.60    28.36
{9}    {0}    7.60    27.34
```

## # Any other specification of your implementation and testing

제가 Mac OS사용중이라서요.

.exe 파일 만들때 친구 랩탑(Windows) 빌려서 해서

거기서의 컴파일 방법(스크린샷) 첨부합니다.

