Homework 1

COL 761

- You need to do the homework in your already formed team of 3. Make sure to upload you code to the github repo you mentioned in HW0.
- Due: **3rd September 11:59PM.**
- Your code must compile and execute on HPC. You will get 0 for compilation or execution errors
- No multi-threading or parallelization is allowed
- **Do not copy code from your friend or from the internet. We have previous year's submissions and downloaded all available libraries of Apriori algorithm and PrefixSpan from the web and all submitted codes will be checked against these as well as those submitted in this homework. Any plagiarized code will result in an F grade for the course straight-away.**

---

1. [**5 points**] Mention your github repo. Make sure that this is the same github repo as what you mentioned in HW0.
2. This question is on frequent itemset mining. Implement Apriori Algorithm to mine frequent itemsets. Apply it on the Dataset: http://fimi.uantwerpen.be/data/webdocs.dat.gz. You may assume all items are integers.
    a. **[20 points]** Please name your bash file RollNo.sh. For example, if MCS162913 is your roll number, your file should be named MCS162913.sh. Executing the command "./RollNo.sh -apriori webdocs.dat X <filename>" should generate a file filename.txt containing the frequent itemsets at >=X% support threshold with the apriori algorithm. Notice that X is in percentage and not the absolute count. Your implementations must ensure that the transactions are **not** loaded into main memory. This means, that it is not allowed to parse the complete input data and save it into an array or similar data structure. However, the frequent patterns and candidate sets can be stored in memory.

    filename.txt should strictly follow the following format.

        i. Each frequent itemset must be on a new line.
        ii. The items must be space separated and in ascending order of ASCII code.

    Your grade will be (F-score)*20.

    b. **[20 points]** Compare the performance of your implemented apriori algorithm with the following FP-tree implementation https://borgelt.net/fpgrowth.html (download package "fpgrowth.zip" and unzip it, cd fpgrowth/fpgrowth/src, make all, run using ./fpgrowth -sSUPPORT% inputfile outfile. For more details visit https://borgelt.net/doc/fpgrowth/fpgrowth.html). Executing the command "./RollNo.sh webdocs.dat -plot" should generate a plot using matplotlib where the x axis varies the support threshold and y axis contains the corresponding running times. It should plot the running times of FP-tree and Apriori algorithms at support thresholds of 5%, 10%,

25%, 50%, and 90%. Explain the results that you observe. You may add a timeout if your code fails to finish even after 1 hour.

3. [**25 points**] Implement PrefixSpan. Use it on the finished paths sequence dataset at SNAP: Web data: Wikispeedia navigation paths (stanford.edu). In this dataset, every itemset within a sequence is of length 1. You will need to extract only the sequence of page titles and ignore all remaining fields (only the column "path" is of significance). Note that the '<' symbol means a back-click. You would to add a pre-processing step further to add relevant page title when seeing a back-click. You can save the final pre-processed input in a file named 'paths_finished.dat'. Executing the command "./RollNo.sh -prefixspan paths_finished.dat X <filename>" should generate a file filename.txt containing the frequent sub-sequences at >=X% support threshold. Notice that X is in percentage and not the absolute count.

   filename.txt should strictly follow the following format.

   i. Each frequent itemset must be on a new line.
   ii. The items must be space separated and in ascending order of ASCII code.

Bash scripts you need to provide:

- compile.sh that compiles your code with respect to all implementations. Specifically running ./compile.sh in your submission folder should create all the binaries that you require. Any optimization flags like O3 for g++ should be included here itself
- RollNo.sh as specified earlier
- install.sh that should execute cloning of your team's repository in the current directory followed by executing bash commands to load all the required HPC modules. Inside the repository we should be able to locate EntryNo-Assgn1.zip corresponding to your Homework 1 submission. **Note this script will be run as source install.sh (to make use of HPC module load alias).**

Submission Instructions:

- Add **col761-2021** as a collaborator for your private repository
- Make sure to fill this form https://forms.gle/UjGL8FHsmYin69UQ7 **, one per team**
- There should be only **one** submission per group.
- Upload EntryNo-Assgn1.zip file to the root directory of your GitHub repository. This entry should be of any one of your team member. Ex. MCS162913-Assgn1.zip. On unzipping it should produce one folder. The folder should have the same name as zip file. This folder should contain all the source files and all the bash scripts. In addition, it should all contain a README.txt explaining all the files you bundled, explanation of q2-part b and entry numbers and names of **all team** members.
- RollNo.sh is the main script that will be used in q2-parts a, b, and q3.
- **You also need to submit install.sh script on moodle(Do not zip it), one per team**

- Make sure to **use same roll number** to give name of all submission files (EntryNo-Assgn1.zip , RollNo.sh) and install.sh should be submitted by same team member.
- Since your submissions will be auto graded, it is essential to ensure your submissions conform to format specified
- Later of timestamp of your last pushed commit and moodle submission of install.sh will be treated as your submission time

Compiler Specification:

- GCC version 7.1.0
- Java version 1.8
- Python3 version 3.6.5
- Python2 version 2.7.13