High Performance Computing

Homework #1: Part B

Due: Monday September 11 2017 before 11:59 PM (Midnight)

Email-based help Cutoff: 5:00 PM on Sun, Sept 10 2017

Maximum Points For This Part: 35

Submission Instructions

This homework assignment must be turned-in electronically via Canvas. You may suitably organize your C++ source code into as many additional methods as needed. Upload just your source files onto Canvas individually. **Do not submit archive files such as: zip,7zip,rar,tar,tar.gz,rpm,deb etc.**

Objective

The objective of this homework is to develop a basic C++ program to implement a frequently used gather operation in a memory efficient manner.

Grading Rubric:

This is an advanced course and consequently the expectations in this course are higher. Accordingly, the program submitted for this homework must pass necessary tests in order to qualify for earning a full score.

NOTE: Program that do not compile, have methods longer than 25 lines, use `new` operator, are just skeleton code, or hard coded solutions will be assigned zero score.

Scoring for this assignment will be determined as follows assuming your program compiles (and is not skeleton code):

- **-1 Points**: for each warning generated by `g++` when compiling your C++ program with the `-Wall` (show all warnings) flag.
- **-1 Points**: for each warning generated by the CSE departments' C++ style checker (a slightly relaxed version from Google Inc).
  NOTE: Points will be deducted for violating stylistic qualities of the program such as: program follows formatting requirements (spacing, indentation, suitable variable names with appropriate upper/lowercase letters, etc.). The program should include suitable comments at appropriate points in each method to elucidate flow of thought/logic in each method. Program strives to appropriately reuse as much code as possible.

- **Graduate Students**: Your solution must print the top *k* high frequency n-grams sorted in alphabetical order to earn full credit for this homework.

Problem statement

This homework exercise expects you to extract and display high frequency *n*-grams (read: https://en.wikipedia.org/wiki/N-gram) from a given text file. For this homework:

- An *n*-gram ($0 < n < \infty$) is defined as a series of consecutive "word"s.
- A "word" is defined as consecutive set of 1 or more alphanumeric (`'A'...'Z'`, `'a'...'z'`, `'0'...'9'`) characters (see `std::isalnum`) separated by one or more non-alphanumeric characters or whitespaces. All characters in a word are converted to lowercase (see `std::tolower`)
- The top *k* highest frequency *n*-grams must be displayed with the highest frequency *n*-grams first. Graduate students only: The top *k* *n*-grams with the same frequency must be sorted in alphabetical order to earn full credit for this homework.

Starter Code:

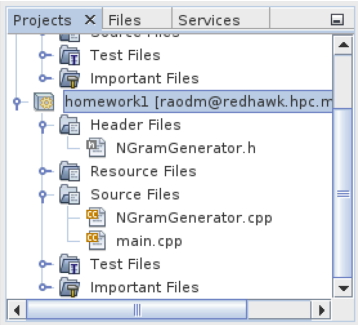
In order to streamline this homework the following file(s) are supplied for this homework:

i. `main.cpp`: **Do not modify this file**. Do review the source code in this file. This file provides an implementation for the main method that processes command-line arguments. You will need to add this file to your `NetBeans` project. Since this file should remain unmodified, do not submit `main.cpp`.

ii. `NGramGenerator.h`: This is a simple header file that is used by `main.cpp`. You may extend/modify this file (add instance variables, methods, etc.) as you see fit.

iii. `NGramGenerator.cpp`: This is essentially a blank source file that simply serves as a placeholder. You will implement the various methods in the header file in this source file.

You should create a suitable `NetBeans` C++ project and include the supplied starter code into your project. The adjacent screenshot shows an example of a `NetBeans` project with the files for your reference. You can either scp the starter code to `redhawk` or copy-paste the code to corresponding source files in the project.

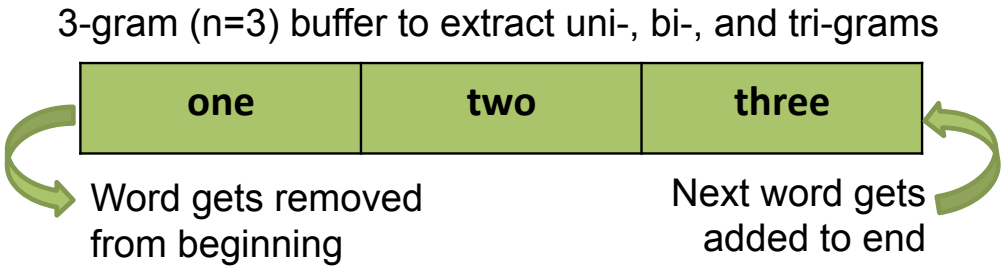


Additional information & Tips

An *n*-gram is just a series of consecutive words separated by 1 blank space character. For example, given the following sequence: `"one.two,  three!/*four"`,

- The unigrams (1 word) are: `"one"`, `"two"`, `"three"`, `"four"`
- The bigrams (2 words) are: `"one two"`, `"two three"`, `"three four"`
- The trigrams (3 words) are: `"one two three"`, `"two three four"`

One straightforward strategy to extract *n*-grams is to have a list of *n* words in a `std::vector` in the form of queue and process them. New words are added to the end of the vector (via `vec.push_back`) and new word is removed from the beginning (via: `vec.erase(vec.begin())` as suggested in the figure below:



Recollect that a "word" is defined as consecutive set of 1 or more alphanumeric (`'A'...'Z'`, `'a'...'z'`, `'0'...'9'`) characters. The following approach is suggested for constructing words:

- Process character-by-character. Read characters using `operator>>`.
- **Note**: `operator>>` will skip whitespaces (tabs, blanks, newlines, etc.). However, to detect end of a word you need to read whitespaces as well. Consequently, prior to processing a stream ensure you suppress skipping of whitespaces (e.g., `is >> std::noskipws;`). Google for `std::noskipws` (no-skip-whitespace) for additional details.
- Use the `std::isalnum` method to detect if a character is a alphanumeric character.
- Use the `std::tolower` method to convert characters to lower case.

A suitable `std::unordered_map` is the most convenient data structure to count the number of occurrences of a given *n*-gram.

Base case (20 points): In order to earn any points for this homework all the n-grams extracted from a given input file must be printed. See sample base case outputs further below.

Extra feature(s) required for full credit (15 points):
Once you have counted all the *n*-grams, the top *k* highest frequency n-grams must be printed. The high frequency *n*-grams can be extracted in order using a suitable `std::priority_queue` with a custom comparison lambda as shown in the online

documentation example for its constructor. However, the following additional constraint must be met:

- **Undergraduate students only**: The *n*-grams with same frequency maybe printed in any order.
- **Graduate students only**: In addition to the *k* highest frequency n-grams, your solution must print the top *k* high frequency n-grams sorted in alphabetical order to earn full credit for this homework. This can be accomplished using a suitable comparison lambda for `std::priority_queue` and some extra processing to print entries in order.

Running programs via NetBeans

Running the program via NetBeans requires you to specify necessary command-line arguments to the program as illustrated in video demonstration on:
Canvas→Pages→View All Pages→Video Demonstrations.

Needless to add you will need to use the `NetBeans` debugger to help with troubleshooting your program.

Sample Outputs

Expected outputs from multiple independent runs of the completed program are shown below.

Base case test #1 [minimum functionality required to earn any points]
Since the base case does not require sorted output the Linux sort command is used to generate consistent outputs for functional testing.

```
$ ./homework1 --file simple.txt --min-len 1 --max-len 2 --top-k 100 | sort -nr -k2 -t:
cc: 2
bb: 2
aa: 2
cc cc: 1
bb cc: 1
bb bb: 1
aa bb: 1
aa aa: 1
```

Base case test #2 [minimum functionality required to earn any points]
Since the base case does not require sorted output the Linux sort command is used to generate consistent outputs for functional testing.

```
$ ./homework1 --file simple.txt --min-len 3 --max-len 5 --top-k 100 | sort -nr -k2 -t:
bb cc cc: 1
bb bb cc cc: 1
bb bb cc: 1
aa bb bb cc cc: 1
aa bb bb cc: 1
aa bb bb: 1
aa aa bb bb cc: 1
aa aa bb bb: 1
aa aa bb: 1
```

Undergraduate students additional functionality test #1:

```
$ ./homework1 --file simple.txt --min-len 1 --max-len 2 --top-k 3 | sort
cc: 2
bb: 2
aa: 2
```

Undergraduate students additional functionality test #2:

```
./homework1 --file ngram.txt --min-len 3 --top-k 13 | sort -k1,2nr -t:
n gram models: 10
an n gram: 9
natural language processing: 8
out of vocabulary: 7
syntactic n grams: 7
an n gram model: 6
i n 1: 6
n gram model: 6
of n grams: 6
serve as the: 6
x i 1: 6
x i n: 6
x i n 1: 6
```

Graduate students full functionality test #1:

Note: The command does not use `sort` as output should already be sorted.

```
./homework1 --file ngram.txt --min-len 1 --top-k 2
aa: 2
bb: 2
```

Graduate students full functionality test #2:

Note: The command does not use `sort` as output should already be sorted.

```
./homework1 --file simple.txt --min-len 1 --top-k 4
aa: 2
bb: 2
cc: 2
aa aa: 1
```

Graduate students full functionality test #3:

Note: The command does not use `sort` as output should already be sorted.

```
./homework1 --file ngram.txt --min-len 3 --top-k 13
n gram models: 10
an n gram: 9
natural language processing: 8
out of vocabulary: 7
syntactic n grams: 7
an n gram model: 6
i n 1: 6
n gram model: 6
of n grams: 6
serve as the: 6
x i 1: 6
x i n: 6
x i n 1: 6
```

Turn-in:

Submit just the C++ source file that meets the requirements of this part of the homework to Canvas. No credit will be given for submitting code that does not compile or just skeleton code. Verify that your program meets all the requirements as stated in the grading rubric. Ensure your C++ source files are named with the stipulated naming convention. Upload all the necessary C++ source files to onto Canvas. Do not submit zip/7zip/tar/gzip files. Upload each source file independently.