Софийски университет „Св. Климент Охридски“
Факултет по математика и информатика

Трето домашно

курсове *Структури от данни;*
Структури от данни и програмиране

специалности *Информатика, Информационни системи и*
Компютърни науки (1-ви поток)

зимен семестър 2021/22 г.

Редакции

07.01.2022 - Сложността е сменена на средна.

Описание на задачата

Дадени са текстови файлове. За улеснение, считаме, че всеки един такъв файл се състои от нула, една или повече думи, без да придаваме някакъв смисъл на това какви точно са думите и какво означават. В рамките на задачата под “дума” ще разбираме максимална, непрекъсната поредица от непразни (non-whitespace) символи. Думите се разделят помежду си от произволен брой празни (whitespace) символи. Забележете, че не е нужно думите да се състоят само от букви. Например в низа

```
"1234 \t\t(*abc*(  \r  \n abd \r\n"
```

се съдържат думите:

- `1234`
- `(*abc*(`
- abd

Програмата ви трябва да може да анализира текстови файлове и да извежда на екрана в каква степен те съдържат едни и същи думи.

Например нека са дадени следните файлове:

file1 съдържа:
one two three four two one

file2 съдържа:
two one four one one

file1 съдържа 6 думи и 4 от тях се съдържат във file2 (66%)
file2 съдържа 5 думи и 4 от тях се съдържат във file1 (80%)

Забележете, че броят на срещанията на дадена дума във файла има значение. Например, ако един файл съдържа три срещания на думата “one”, а друг само две, това не е пълно съвпадение. Само две от думите от първия файл се съдържат във втория, а третото срещане на “one” няма еквивалент.

Вие трябва да разработите програма, която получава като параметри от командния ред (argc/argv) пътищата до два файла. Програмата трябва да ги сравни и да изведе на екрана информация за съвпаденията между всяка двойка файлове. Програмата трябва да може да работи с големи файлове (например десетки или стотици мегабайти). Когато разработвате решението си, подгответе примерни файлове с голям обем и се уверете, че то може да работи с тях, а също и че приключва за разумно време. За разумно време можем да приемем **средна** сложност от вида O(n*log(n)+m*log(m)), където n и m са дължините на 2-та документа.

Класове

Решението ви трябва да съдържа клас Comparator, който може да се използва за сравнение на два потока от символи. Пълно описание на класа можете да намерите във файла interface.h (за повече информация вижте секцията “Тестове” по-долу). Тук можете да намерите кратко описание на това какво трябва да прави класът.

Comparator трябва да има функция

```
ComparisonReport compare(std::istream& a, std::istream& b)
```

функцията получава два входни потока от символи a и b. Тя анализира съдържанието им и връща резултата от сравнението.

Ако по време на работата на функцията възникне някаква грешка, това трябва да се сигнализира чрез хвърляне на подходящо изключение. То трябва да бъде или някое от стандартните изключения в C++ библиотеката, или ако се налага да използвате собствен тип, той трябва да бъде пряк или непряк наследник на std::exception.

Функцията трябва да връща обект от тип ComparisonReport. Този обект има членове, които докладват кои са думите, които са общи и за двата потока, а също и кои са

уникални за всеки от тях. Описанието на ComparisonReport също можете да намерите в предоставените файлове. В решението си НЕ ТРЯБВА да променяте този клас.

ComparisonReport използва обекти от тип WordMultiset. Това е клас, който представя мултимножество от думи. Този клас също трябва да реализирате сами. Интерфейсът му е описан в interface.h. Няма ограничение за броя на уникалните думи, които да се съдържат в обект от тип WordMultiset, нито в броя на срещанията на всяка от тях.

Тъй като това домашно има за цел да упражни работата с хешове, при решаването му ще трябва да изберете подходяща хеш-функция и да работите с хешове, които се получават от входа, който ще получите. Можете сами да изберете кой алгоритъм да използвате за хеш-функцията. Например, възможен вариант е да използвате std::hash от стандартната библиотека, MD5 (https://en.wikipedia.org/wiki/MD5) или някой друг алгоритъм.

Специално искаме да уточним, че извън хеширането (което искаме да упражним в това домашно), за решаване на задачата има и други подходи, някои даже по-ефективни.

Ако изберете да работите с хеш различен от този в стандартната библиотека на C++ (std::hash), можете да свалите готова библиотека или сами да реализирате хеш функция. Например ако решите да използвате MD5 в своето решение, можете да свалите следната библиотека: http://www.zedwood.com/article/cpp-md5-function

Независимо кой вариант изберете, при предаването на работата, ако използвате библиотека за хеширане, трябва явно да посочите коя и да включите в архива всичко нужно, за да може решението ви да се изгради. Тоест ако използвате външна библиотека, трябва да включите и нейните файлове и всичко трябва да бъде разположено по правилен начин в архива (например в поддиректории, ако така сте организирали кода си), за да може решението да се изгради. За целта, освен файловете с изходен код, предайте и проектните файлове (make файлове, CMakeLists, .vcxproj/.sln и т.н., според това как сте разработвали решението си). Ако решите да използвате дадена външна библиотека, уверете се, че лицензът ѝ позволява това (както употребата, така и разпространението на кода).

В решението на задачата НЕ МОГАТ да се използват наготово стандартни реализации на дърво или хеш. Например не могат да се използват следните контейнери от STL: std::set, std::multiset, std::map, std::multimap, std::unordered_set и т.н.). Ако използвате хеш-таблица или дърво, трябва да напишете кода сами. Единственото изключение се прави за класа WordMultiset, в неговата функция words(), която трябва да върне обект от тип std::multiset<std::string>. В този случай трябва да създадете обект от този тип и да го попълните с нужната информация.

Тестове

Тестове за задачата можете да намерите в хранилището на курса, на следния адрес https://github.com/semerdzhiev/sdp-2021-22/tree/main/tests/3. Вашето решение трябва като минимум да минава тези тестове. Ако решението ви минава всички тестове това не означава автоматично получаване на пълен брой точки. Тестовете проверяват няколко конкретни случая и не са изчерпателни. Препоръчваме да добавите свои такива, които обхващат повече случаи.

За да може тестовете да проверяват коректността на вашето решение трябва да спазите няколко условия при реализация на кода си.

Във файла interface.h ще намерите класовете, за които стана дума по-горе в текста. Трябва да имплементирате всички функции на Comparer и WordMultiset. Ако счетете за нужно, можете да добавите нови членове в тези два класа. Класът ComparisonReport НЕ БИВА да се променя.

В test.cpp ще намерите всички тестове, които вашето решение трябва да минава успешно. Тестовете трябва да минават без промяна на кода в този файл.

Ако счетете за уместно, добавете допълнителни тестове, с които да проверите решението си. Препоръчваме да ги отделите в друг файл и да оставите оригиналния test.cpp непроменен.

Ако смятате, че някой тест противоречи на условието, пишете в Discord канала на курса и тагнете някой от екипа.

ВАЖНО: Не забравяйте, че решението на задачата трябва да включва както реализация на определена функционалност, която проверяваме с тестовете, така и самостоятелна програма, която анализира файлове. Това означава че трябва да има 2 подпроекта (project във Visual Studio или target в CMake) – единият да пуска тестовете, а другият да реализира крайното приложение.

При реализацията е разрешено да използвате само посочените долу класове от стандартната библиотека. Всички останали структури от данни и/или алгоритми, които са ви необходими при решаването на задачата трябва да реализирате сами.

- forward_list
- list
- pair
- priority_queue
- queue
- stack
- string
- vector