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

Домашна работа 3

*курс Структури от данни и програмиране*

*за специалност Информатика*

*зимен семестър 2017/18 г.*

В това домашно трябва да напишете програма, която сканира всички файлове намиращи се под дадена директория и извежда размера на тези от тях, които отговарят на даден критерий.

Програмата ви трябва да получи от стандартния вход два параметъра – име на директория и шаблон за името на файловете, които да се включат в резултатите. След това програмата ви трябва да изведе всички файлове отговарящи на шаблона, в намаляващ ред, според техния размер. Пръв трябва да се изведе най-големият файл, след него – следващият по размер и т.н. Ако два файла имат еднакъв размер, те трябва да се подредят по лексикографската наредба, по техния пълен път. Например файлът C:\A\file.txt е по-напред в наредбата от файл с пълен път C:\B\aaa.txt.

Най-простият шаблон, който можете да реализирате, е по разширението на файловете. В този случай потребителят въвежда дадено разширение, като например “docx” и след това програмата ви трябва да изведе информация само за файлове, които имат това разширение. Забележете, че ако разширението, което търсим е “docx”, в резултатите НЕ трябва да се включва например файл с име “aaa.abcdocx”, т.е. прилагането на шаблона не може да се направи само с проверка дали разширението е суфикс на името на файла, а трябва да се потърси и разделящата точка.

Опционално (за повече точки) можете да реализирате търсене по шаблон съдържащ wildcards (символите \* и ?). Такъв шаблон е например “?doc\*.docx”, който би съвпаднал например с файловете “1doc-1.docx”, “2document.docx”, “3doc.docx” и т.н.

Търсенето трябва да става в дълбочина – както в подадената директория, така и във всички нейни поддиректории, техните поддиректории и т.н. Тъй като броят на директориите на дадено ниво може да бъде голям, помислете кой алгоритъм за обхождане ще използвате и как ще го реализирате, за да сте сигурни, че няма да имате проблем с паметта. Например обхождането не трябва да предизвиква препълване на стека, дори и при голям брой директории.

За улеснение може да считате, че в директорията, която ще бъде подадена на вашата програма, няма да се съдържат soft links, hard links или junctions (NTFS). Тоест ще работите с истинско дърво, а не с граф; може да считате, че в него нито може да има цикли, нито даден елемент може да се повтаря по няколко пъти. Също така, може да считате, че всички файлове ще са с размер по-малък от 232-1 (т.е. размерът се събира в 32-битова променлива без знак). Имената им ще съдържат само стандартни ASCII символи с кодове по-малки от 128 (няма нужда да използвате unicode). Дължината на името на всеки файл ще бъде по-малка от 260 символа (стойността на константата MAX\_PATH във Windows).

За извличане на всички файлове и поддиректории в дадена директория можете да използвате например (в края на документа ще намерите допълнителна информация):

* Windows: функциите FindFirstFile, FindNextFile и FindClose;
* Linux: функциите opendir, readdir и closedir;
* Платформено независимо: Boost Filesystem Library.

Сами трябва да реализирате всички изисквания в задачата. Например сами трябва да напишете алгоритъм, който проверява дали даден символен низ (име на файл), отговаря на даден шаблон. За целта НЕ МОЖЕ да се използва готовата функционалност, която предлагат горе-предложените функции (те поддържат търсене с wildcards). Също така сами трябва да реализирате подходящо обхождане на дървото на директориите и подходяща сортировка. НЕ МОЖЕ да се използват готови функции за сортиране, като например std::sort.

При решаването на задачата може да се използват наготово функциите и класовете в STL, например std::string и базовите колекции std::vector, std::list std::forward\_list, std::stack и std::queue. За всички структури от данни и алгоритми, които използвате наготово, трябва да разбирате как работят те и да можете да ги реализирате сами. Например ако използвате std::vector се предполага, че имате добро разбиране за структурата от данни динамичен масив и можете сами да напишете клас, който я реализира.

Обхождането на дървото трябва да стане чрез итератор. За целта създайте такъв, който обхожда всички директории под дадената и връща намиращите се в тях файлове. Създайте втори итератор, който разширява първия, като филтрира онези файлове, които не отговарят на шаблона. Използвайте итераторите, за да извлечете имената на файловете, които са ви нужни, за да решите задачата.

**Допълнение (23.12.2017 г.):** Възможно е броят на файловете, които отговарят на даден шаблон, да бъде толкова голям, че информацията за тях (пълен път + размер) да не може да се побере в оперативната памет. Във вашата програма трябва да намерите начин да се справите с този проблем. Решение, при което цялата информацията се запазва във вектор, който след това се сортира с помощта на std::sort и се извежда на екрана, не е валидно за задачата.

По-долу са предложени няколко възможни варианта, които можете да реализирате. Първите адресират проблема с големия обем данни, като в тях е предложено как да подходите към проблема. В останалите се улеснява условието на задачата, като се допуска, че данните, с които се работи могат да се поберат в паметта. В тези случаи обаче се изисква да реализирате конкретен алгоритъм за работа с дърво.

1. Директориите се обхождат и информацията за намерените файлове, които отговарят на критерия за търсене, се записва във временен файл на диска. Този файл се сортира с помощта на алгоритъм за външно сортиране и информацията се извежда на екрана. Накрая файлът се изтрива.
2. Тъй като основният обем информация идва от пътищата на файловете, може да се потърси начин тази информация да се компресира или да се изнесе на диска, а сортирането да стане или върху компресираните данни или върху друга структура от данни, която пази размерите на файловете и (може би) индекс, по който може да се намерят имената им в компресираната структура (или на диска).
3. Улесняваме задачата, като допускаме, че информацията може да се съхрани в паметта. Улесняваме и условието за сортиране, като файловете трябва да се изведат на екрана лексикографски по техния път. В този случай програмата трябва да бъде интерактивна и да поддържа 3 команди – quit, refresh и print. quit се използва за излизане от програмата. При нейното стартиране, тя сканира входната директория и построява в паметта дърво, което я представя. С командата refresh, дървото се обхожда отново и в дървото се отразяват настъпилите промени след последното обхождане (например може да са изтрити някои файлове, а други да са добавени). С командата print се извежда на екрана съдържането на дървото, като файловете са подредени лексикографски по техния пълен път. Също така, при извеждането трябва да се маркират по някакъв начин промените спрямо предишното обхождане. Например може да се изведе символ плюс (+) пред всички новодобавени файлове и минус (-) пред всички изтрити. print може да приема един опционален параметър – филтър, който указва кои файлове да се изведат. Филтърът е във формата, който е описан в условието на задачата (например “print docx”). Ако този параметър не се подаде (потребителят въвежда просто “print”), трябва да се изведат всички файлове. В този случай отново трябва да се реализират два итератора, като единият сканира входната директория и връща всички файлове от нея и се използва за построяването на дървото. Вторият сканира дървото и връща само онези файлове, които отговарят на даден филтър.

## Примери за обхождане на директории (API)

### Windows

По-долу можете да намерите няколко прости примера за обхождане на директория. Примерът е за “плоско” обхождане – само за една директория. Помислете как да го адаптирате така, че да можете да обходите цялото дърво.

<https://github.com/semerdzhiev/sdp-samples/blob/master/Other/DirScan.cpp>

<https://gist.github.com/poseidon4o/cc6e4789704d559911741aa4d8267762>

За по-подробен пример и информация за функциите в API-то, вижте например:

<https://msdn.microsoft.com/en-us/library/aa365200(VS.85).aspx>

<https://msdn.microsoft.com/en-us/library/windows/desktop/aa365740(v=vs.85).aspx>

### Linux

Кратък пример за използване на функциите:

<https://www.gnu.org/software/libc/manual/html_node/Simple-Directory-Lister.html>

За повече информация:

<https://www.gnu.org/software/libc/manual/html_node/Opening-a-Directory.html>

<http://man7.org/linux/man-pages/man3/opendir.3.html>

<https://www.gnu.org/software/libc/manual/html_node/Reading_002fClosing-Directory.html>

### Boost

За повече информация за Filesystem Library:

<http://www.boost.org/doc/libs/1_36_0/libs/filesystem/doc/index.htm>