МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

„КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМ. ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС

„ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ”

# Курсова робота

## з курсу «Паралельні обчислення»

Виконав:

студент групи ДА-71

Михайловин Роман

Варіант 11

Київ – 2020

**Теоретичні відомості**

**Індекс** - структура даних, яка допомагає СУБД швидше виявити окремі записи в файлі і скоротити час виконання запитів користувачів.

Індекс в базі даних аналогічний предметному покажчику в книзі. Це - допоміжна структура, пов'язана з файлом і призначена для пошуку інформації за тим же принципом, що і в книзі з предметним покажчиком.

**Індексування** - це процес перегляду файлів, електронних повідомлень та іншого вмісту на комп'ютері та класифікації інформації про них, наприклад вміст в них слів і метаданих. При виконанні пошуку після індексування, система аналізує індекс термінів, щоб швидше знайти результати.

**Хід роботи**

Програму розроблено на мові програмування с++. Для використання багатопоточності використано бібліотеку <thread>, для організації правильної роботи потоків використано м’ютекс (бібліотека <mutex>). Кількість потоків можна обрати. Відповідно до варіанту обрано такі дані:

1. aclImdb\test\neg – 2500 -2750

2. aclImdb\test\pos – 2500 -2750

3. aclImdb\train\neg – 2500 -2750

4. aclImdb\train\pos – 2500 -2750

5. aclImdb\train\unsup – 10000-11000

Програму завантажено на github:

<https://github.com/IscraSpark/course_work_parallel_computing>

Тестування програми

1 - 123с

2 - 87с

3 - 63с

4 - 69с

5 - 69с

6 - 66с

7 - 76с

8 - 80с

9 - 89с

10 - 89с

Рис.1

На діаграмі можемо бачити залежність часу роботи програми від кількості задіяних потоків. Під час тестування не враховувався час на вивід результату та запису в файл.

**Перевірка правильності роботи**



Рис.2 Результат роботи послідовного рішення



Рис.3 результат роботи паралельного рішення

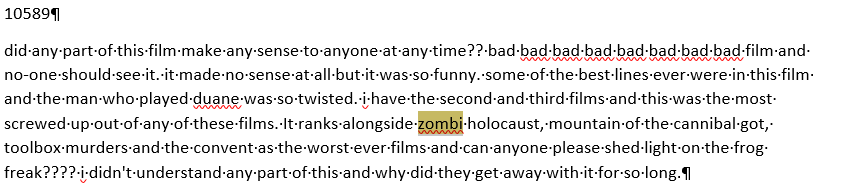


Рис.4 Вміст файлу 10589

Як бачимо результат однаковий при обох рішення і дане слово справді знаходиться у даному файлі.

Лістинг програми

Index.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <direct.h>

#include <io.h>

#include <string>

#include <fstream>

#include <map>

#include <vector>

#include <cctype>

#include <algorithm>

#include <thread>

#include <mutex>

#include <fcntl.h>

#include <chrono>

#include "clean.h"

#define START 2500

#define END 2750

// 10 000 - 11 000

//#include <regex>

using namespace std;

mutex mux;

map <wstring, int> dictionary;

int iter = 0;

string path1 = "aclImdb\\test\\neg\\",

path2 = "aclImdb\\test\\pos\\",

path3 = "aclImdb\\train\\neg\\",

path4 = "aclImdb\\train\\pos\\",

path5 = "aclImdb\\train\\unsup\\";

class Pages

{

private:

vector<int> x; // хранит страницы в которых встречалось слово

public:

void add(int el){

x.push\_back(el);

}

void show()

{

for (int i = 0; i < x.size(); i++)

wcout << x[i] <<" ";

}

int find(int page)

{

int flag = 0;

for (auto it = x.begin(); it < x.end();it++)

{

if (\*it == page)

{

flag = 1;

break;

}

}

return flag;

}

int sizev()

{

return x.size();

}

int get(int ind)

{

return x[ind];

}

};

Pages page[50000];

void indexer(string path, int begin, int end)

{

int index = 0,i;

Pages p[50000];

map <wstring, int> dict;

for (i = begin; i <= end; i++)

{

cout << i << " ";

\_finddata\_t data;

string buff;

buff = to\_string(i);

buff = path + buff + "\\*"; // запись пути к файлу

//cout << buff<<endl;

char a[50],b[50];

strcpy(a, buff.c\_str());

//cout << a;

intptr\_t handle = \_findfirst(a, &data); // поиск нужного файла

buff = path + data.name;

//cout << data.name << " ";

strcpy(b, buff.c\_str());

//cout << data.name<<endl;

wifstream fin(b);

if (!fin.is\_open()) // если файл не открыт

cout << "Can't open: " << data.name; // сообщить об этом

else

{

wstring word;

while (!fin.eof()) // заполнение вспомагательного словаря

{

fin >> word; // считать слово из файла

word = clean(word);

if (word.size() > 0)

{

auto it = dict.find(word); // проверка, встечалось ли это слово ранее

if (it != dict.end())

{

if (!p[it->second].find(i))

{

p[it->second].add(i);

}

}

else

{

dict.insert(pair<wstring, int>(word, index));

p[index].add(i);

index++;

}

}

}

fin.close(); // закрываем файл

}

}

mux.lock();

// перенос данных в основной словарь

for (auto it = dict.begin(); it != dict.end(); it++)

{

//dictionary.insert(pair<string, int>(it->first, iter));

auto iterator = dictionary.find(it->first);

if (iterator != dictionary.end())

{

for (i = 0; i < p[it->second].sizev(); i++)

{

if (!page[iterator->second].find(p[it->second].get(i)))

{

page[iterator->second].add(p[it->second].get(i));

}

}

}

else

{

dictionary.insert(pair<wstring, int>(it->first, iter));

for (i = 0; i < p[it->second].sizev(); i++)

{

page[iter].add(p[it->second].get(i));

}

iter++;

}

}

mux.unlock();

}

void router(int box1, int box2, int box3, int box4, int bigbox\_start, int bigbox\_end) // указывает путь и количество данных которые нужно обработать

{

if (box1)

{

indexer(path1, START, END);

}

if (box2)

{

indexer(path2, START, END);

}

if (box3)

{

indexer(path3, START, END);

}

if (box4)

{

indexer(path4, START, END);

}

if (bigbox\_start < bigbox\_end)

{

indexer(path5, bigbox\_start, bigbox\_end);

}

}

void create\_processes(int n) // распределяет задания между потоками

{

thread th[10];

if (n == 1)

{

router(1, 1, 1, 1, 10000, 11000);

}

if (n == 2)

{

th[0] = thread(router, 1, 1, 1, 1, 0, 0);

th[1] = thread(router, 0, 0, 0, 0, 10000, 11000);

}

if (n == 3)

{

th[0] = thread(router, 1, 1, 1, 0, 0, 0);

th[1] = thread(router, 0, 0, 0, 1, 10000, 10500);

th[2] = thread(router, 0, 0, 0, 0, 10501, 11000);

}

if (n == 4)

{

th[0] = thread(router, 1, 1, 0, 0, 0, 0);

th[1] = thread(router, 0, 0, 1, 1, 0, 0);

th[2] = thread(router, 0, 0, 0, 0, 10000, 10500);

th[3] = thread(router, 0, 0, 0, 0, 10501, 11000);

}

if (n == 5)

{

th[0] = thread(router, 1, 1, 0, 0, 0, 0);

th[1] = thread(router, 0, 0, 1, 1, 0, 0);

th[2] = thread(router, 0, 0, 0, 0, 10000, 10333);

th[3] = thread(router, 0, 0, 0, 0, 10334, 10666);

th[4] = thread(router, 0, 0, 0, 0, 10667, 11000);

}

if (n == 6)

{

th[0] = thread(router, 1, 1, 0, 0, 0, 0);

th[1] = thread(router, 0, 0, 1, 1, 0, 0);

th[2] = thread(router, 0, 0, 0, 0, 10000, 10250);

th[3] = thread(router, 0, 0, 0, 0, 10251, 10500);

th[4] = thread(router, 0, 0, 0, 0, 10501, 10750);

th[5] = thread(router, 0, 0, 0, 0, 10751, 11000);

}

if (n == 7)

{

th[0] = thread(router, 1, 0, 0, 0, 0, 0);

th[1] = thread(router, 0, 1, 0, 0, 0, 0);

th[2] = thread(router, 0, 0, 1, 0, 0, 0);

th[3] = thread(router, 0, 0, 0, 1, 0, 0);

th[4] = thread(router, 0, 0, 0, 0, 10000, 10333);

th[5] = thread(router, 0, 0, 0, 0, 10334, 10666);

th[6] = thread(router, 0, 0, 0, 0, 10667, 11000);

}

if (n == 8)

{

th[0] = thread(router, 1, 0, 0, 0, 0, 0);

th[1] = thread(router, 0, 1, 0, 0, 0, 0);

th[2] = thread(router, 0, 0, 1, 0, 0, 0);

th[3] = thread(router, 0, 0, 0, 1, 0, 0);

th[4] = thread(router, 0, 0, 0, 0, 10000, 10250);

th[5] = thread(router, 0, 0, 0, 0, 10251, 10500);

th[6] = thread(router, 0, 0, 0, 0, 10501, 10750);

th[7] = thread(router, 0, 0, 0, 0, 10751, 11000);

}

if (n == 9)

{

th[0] = thread(router, 1, 0, 0, 0, 0, 0);

th[1] = thread(router, 0, 1, 0, 0, 0, 0);

th[2] = thread(router, 0, 0, 1, 0, 0, 0);

th[3] = thread(router, 0, 0, 0, 1, 0, 0);

th[4] = thread(router, 0, 0, 0, 0, 10000, 10200);

th[5] = thread(router, 0, 0, 0, 0, 10201, 10400);

th[6] = thread(router, 0, 0, 0, 0, 10401, 10600);

th[7] = thread(router, 0, 0, 0, 0, 10601, 10800);

th[8] = thread(router, 0, 0, 0, 0, 10801, 11000);

}

if (n >= 10)

{

th[0] = thread(router, 1, 0, 0, 0, 0, 0);

th[1] = thread(router, 0, 1, 0, 0, 0, 0);

th[2] = thread(router, 0, 0, 1, 0, 0, 0);

th[3] = thread(router, 0, 0, 0, 1, 0, 0);

th[4] = thread(router, 0, 0, 0, 0, 10000, 10166);

th[5] = thread(router, 0, 0, 0, 0, 10167, 10332);

th[6] = thread(router, 0, 0, 0, 0, 10333, 10500);

th[7] = thread(router, 0, 0, 0, 0, 10501, 10666);

th[8] = thread(router, 0, 0, 0, 0, 10667, 10832);

th[9] = thread(router, 0, 0, 0, 0, 10833, 11000);

}

if (n > 10) n = 10;

for (int j = 0; j < n; j++)

{

if (th[j].joinable())

th[j].join();

}

}

int main(){

//setlocale(LC\_ALL, "rus");

int n;

chrono::system\_clock::time\_point start = chrono::system\_clock::now();

cout << "Enter number of threads:";

cin >> n;

while (n <= 0)

{

cout << "Imposible, enter >0:";

cin >> n;

}

create\_processes(n);

//router(0, 0, 0, 0, 10000, 11000);

wofstream fout("result.txt");

for (auto it = dictionary.begin(); it != dictionary.end(); it++) // вывод сформированого словаря

{

wcout << it->first << ": ";

fout << it->first << ": ";

page[it->second].show();

for (int j = 0; j < page[it->second].sizev(); j++)

fout << page[it->second].get(j) << " ";

wcout << endl;

fout << endl;

}

fout.close();

auto end = chrono::system\_clock::now();

auto diff = chrono::duration\_cast <chrono::seconds > (end - start).count();

cout << "Work time:" << diff;

system("pause");

return 0;

}

Clean.cpp

//#include <string>

#include "clean.h"

#pragma once

wstring clean(wstring word) // убирает нежелательные символы

{

int n;

n = word.find(L"<br />");

if (n != wstring::npos){

word.erase(n, n + 3);

n = word.find(L"<br />");

if (n != wstring::npos){

word.erase(n, n + 3);

}

}

n = word.find(L'(');

if (n != wstring::npos){

word.erase(n, n + 1);

}

n = word.find(L')');

if (n != wstring::npos){

word.erase(n, n + 1);

}

n = word.find(L'(');

if (n != wstring::npos){

word.erase(n, n + 1);

}

n = word.find(L'.');

if (n != wstring::npos){

word.erase(n, n + 1);

}

n = word.find(L',');

if (n != wstring::npos){

word.erase(n, n + 1);

}

n = word.find(L'[');

if (n != wstring::npos){

word.erase(n, n + 1);

}

n = word.find(L']');

if (n != wstring::npos){

word.erase(n, n + 1);

}

n = word.find(L':');

if (n != wstring::npos){

word.erase(n, n + 1);

}

n = word.find(L';');

if (n != wstring::npos){

word.erase(n, n + 1);

}

n = word.find(L'!');

if (n != wstring::npos){

word.erase(n, n + 1);

}

n = word.find(L'?');

if (n != wstring::npos){

word.erase(n, n + 1);

}

n = word.find(L'<');

if (n != wstring::npos){

word.erase(n, n + 1);

}

n = word.find(L'>');

if (n != wstring::npos){

word.erase(n, n + 1);

}

n = word.find(L'{');

if (n != wstring::npos){

word.erase(n, n + 1);

}

n = word.find(L'}');

if (n != wstring::npos){

word.erase(n, n + 1);

}

n = word.find(L'\\');

if (n != wstring::npos){

word.erase(n, n + 1);

}

n = word.find(L'/');

if (n != wstring::npos){

word.erase(n, n + 1);

}

n = word.find(L'\"');

if (n != wstring::npos){

word.erase(n, n + 1);

n = word.find(L'\"');

if (n != wstring::npos){

word.erase(n, n + 1);

}

}

n = word.find(L'\'');

if (n != wstring::npos && (n == 0 || n <= word.size() - 1)){

word.erase(n, n + 1);

n = word.find(L'\'');

if (n != wstring::npos && (n == 0 || n <= word.size() - 1)){

word.erase(n, n + 1);

}

}

n = word.find(L'\*');

if (n != wstring::npos){

word.erase(n, n + 1);

n = word.find(L'\*');

if (n != wstring::npos){

word.erase(n, n + 1);

}

}

n = word.find(L"...");

if (n != wstring::npos){

word.erase(n, n + 3);

}

for (int j = 0; j < word.size(); j++)

{

if (word[j] >= 65 && word[j] <= 90)

{

word[j] += 32;

}

}

//transform(word.begin(), word.end(), word.begin(), tolower);

return word;

}

Висновок: Результатом виконання курсової роботи стала програма, що дозволяє знайти та проіндексувати слова у заданих завданням файлах. Проект викладений у репозиторії на github. Інструкція до програми міститься в README.md.