Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий комплекс  
«Інститут прикладного системного аналізу»

Відділення другої вищої та післядипломної освіти

**ЛАБОРАТОРНА РОБОТА №2**

(варіант 15)

з курсу «Алгоритми та Структури даних»

на тему «Сортування Великих Даних»

Виконав: студент 3-го курсу

групи ІС-зп71

Бутузов О.В.

Прийняв: викл. Древаль М.М.

Захищено з оцінкою

« » 2017 р.

Київ – 2017

# Умова завдання

Завдання розподіляється на дві частини:

1. Створення файлу що містить 64бітні числа у довільном порядку.
2. Сортування цього змісту цього файлу (за допомогою алгоритмів сортування злиттям (merge) та купкою (heap)), усунення дублікатів з зберіганням інформації де число зустрілось уперше (з урахуванням що ми будемо розбивати файл на більш дрібніші).

# Розвязання задачі

Частина 1:

Реалізація доволі проста програма (код якої можна побачити у файлі generate.cpp) – файл продовжується писатись допоки не буде перевищено ліміт розміру файлу (що визначається в коді програми). Генерація довільного 64бітного числа реалізована по аналогії з схожою реалізацією мовою програмування Go ( де кожен з згенерованих 8 байтів зсувається вліво число що еквівалентне n\*8 (де n номер байту), а фінальним довільним числом є побітова або операція з ліва направо (в порядку little endian)), в нашої реалізації ми формуємо довільннe кількість байтів (в межах 1… 8 ) де проводимо аналогічну вищеописаній операцію – в результаті ми маємо більшу дисперсію значень, ніж в рідному алгоритмі.

Частина 2:

Реалізацію програми (код якої можна побачити в sort\_big\_file.cpp ) можна розбити на кілька етапів:

1. Розбиття нашого великого файлу на купу дрібних. Розмір дрібного файлу ми визначали за допомогою наперед заданого числа – що в рамках програмного забезпечення означав частку ділення від максимально-допустимого обсягу оперативної пам’яті що ми могли виділити для сортування масиву 64бітних чисел до розміру одного числа (тобто грубо кажучи приблизно 500.000.000 для 4 Gb пам’ті).
2. Кожен з “дрібних” (насправді розмір може бути ніяк не дрібним і залежить лише від розміру що займають довільних N значень) читаєтся в порядку зростання, завантажується в пам’ять де і сортується обраним нами методом (що визначено в коді) сортується по одному за раз.
3. Відсортовані файли ми склеюємо за тим же принципом що ми використовуємо при алгоритмі сортування злиттям – використовуючи вже файли як наше сховище даних. Цей процес рекурсивний і ми продовжуємо його допоки не залишиться лише 1 не злитого файлу що і буде результатом виконання нашої програми.

При розробці було використано RAM диск що дозволило трошка підвищити ефективність роботи программи.

# Лістинг програм

// random numbers file generator.

// clang++ generate.cpp -o ./generate -std=c++14

#include "code.cpp"

int main(){

// seed for rand

srand( time(NULL) );

std::ofstream data\_in;

data\_in.open ( name("data.in") );

//

uint64 random\_number(0); // our random number container

uint64 totalsize(0); // total size we using now.

//

// Total Allowed size;

uint64 size(0);

// 5 Gb

size = uint64( 1024) \* uint64(1024) \* uint64(1024) \* uint(5);

// Generating Numbers List

while( true ) {

int index(get\_random\_index());

random\_number = random(index);

data\_in << random\_number << std::endl;

// update size counter.

totalsize += uint64( num\_uint64\_length( random\_number ) + 1 );

if ( totalsize > size ){

break;

}

}

data\_in.close();

}

// large file sorter

// clang++ sort\_big\_file.cpp -o ./sort -std=c++14

#include "code.cpp"

int main(){

// Splitting Large Fiels into chunks

int sorted(0), unsorted( split\_big\_file( name("data.in") ) );

// Sorting Files.

for (int i=1; i <= unsorted; i++ ){

std::string unsorted\_F(name("unsorted", i ));

std::string sorted\_F(name("sorted", i ));

if ( sort\_file( unsorted\_F ) ) {

sorted++;

rename( unsorted\_F.c\_str(), sorted\_F.c\_str() );

}

}

// std::cout << "Sorted " << sorted << std::endl;

// std::cout << "Un merged " << sorted << std::endl;

// std::cout << "Un sorted " << unsorted << std::endl;

if ( (unsorted - sorted) > 0 ){

throw;

}

merge\_files( sorted );

return 0;

}

# Результати роботи программи

Результатом роботи програми є відсортований в порядку зростання 64бітних чисел файл що такою містить інформацію де число зустрілось уперше (в якому з “кусочних” файлів).

1. **Висновки**

Ми можемо сформувати наші висновки у двох основних напрямках:

Робота программи і швидкодія проведеня операцій:

В будь якій операції що включає IO операції – це вузьке місце. Операції запису та читання були найповільнішими в порівняні з операцією сортування що проводилась цілком в пам’яті. Важливо було також попередньо визначити скільки елементів ми спроможні відсортувати за раз і використовувати це оптимальне значення і лише потів записати результат сортування (тоюто альтернативно кращий варіант це читати з основного файла не в кілька-надцять не сортованих файлів, а в масив який відразу по досягненню ліміта сортувати і вже сортоване записувати до файлу який пізніше буде склеєсно з іншими).

Алгоритми сортування що ми використали показали себе приблизно однаковими за швидкодією, але обидва мали свої плюси та мінуси: мінусом алгоритму сортування злиттям – можна назвати просторову складність що не завжди є варіантом коли маємо справу з лімітованим апаратним забезпеченням. В нашій реалізації - мінуси алгоритму сортування heap не булу взято до уваги, оскільки сортування та склеювання результатів велось лінійно від більшого до меншого і старші результати сортування просто ігнорувались. Але слід зазначити що, heap sort є не стабільним методом сортування що змінює порядок, і зазвичай має використовуватись лише там, де цей порядок нічого не значить.