**Алгоритми та складність. Лабораторна робота 1.**

**Багатофазне сортування злиттям.**

Грищенко Юрій К-29.

В цій роботі ми ознайомимось зі зовнішнім алгоритмом сортування — багатофазне сортування злиттям.

**Звичайне сортування злиттям**

Взагалі, сортування злиттям полягує у тому, що він ділить вхідні дані на менші відсортовані підмножини, потім ці підмножини зливаються один з одним до тих пір, поки ми не отримуємо одну відсортовану множину — а саме множину вхідних даних.

В звичайних алгоритмах сортування злиттям, підмножини зливаються попарно, тобто, наприклад, 64 підмножини зливаються в 32, потім в 16, 8, 4, 2, 1. Процес злиття двох підмножин розміру k можна легко виконати за О(k) операцій зрівняння. У випадку з масивом даних довжиною 64, ми спочатку виконуємо 32 злиття (по 1 елементу), потім 16 (по 2 елементи), потім 8 по 4, і т. д., тобто кожний крок займає O(n) операцій (де n — довжина масиву). Оскільки кількість кроків дорівнює log n + 1, то складність такого алгоритму дорівнює О(nlogn).

Це досить ефективний алгоритм як і для внутрішнього сортування (тобто, сортування даних, що вміщаються в ОЗУ компютера), так і для зовнішнього. Розглянемо зовнішнє сортування:

Нехай у нас є доступ до 3 файлів: пара вхідних файлів та один вихідний (output). Вхідні файли містять в собі невідсортовані дані, їх умовно розділяємо на певну кількість підмножин (наприклад, по 32 елементи) і кожен з них сортуємо внутрішньо. Потім беремо дві підможини з вхідних файлів, зливаємо їх, та записуємо в вихідний. Далі розподіляємо відсортовані підмножини з вихідного файлу назад у вхідні. Тоді на наступному кроці матимемо відсортовані підмножини по 64 елементи, і т.д.

Рахуючи час виконання цього алгоритму, врахуємо, що запис файлів на диск є дуже довгою операцією. Звичайне злиття сортуванням потребує запису всіх елементів у вихідний файл, а потім ще одного запису назад у вхідні. Злиття даних зменшує кількість підмножин на 2, а запис даних назад не змінює кількість підмножин (фактор — 1). Беремо середнє геометричне — фактор зменшення дорівнює квадратному корню 2, тобто приблизно 1.41.

**Багатофазне сортування злиттям**

Якщо у нас обмежена кількість файлів (до 8 штук), то можна використати ефективніший алгоритм зовнішнього сортування — **багатофазне сортування злиттям**.

Маємо N-1 вхідних файлів і 1 вихідний. Поділимо вхідні дані на підмножини (достатьно маленькі, щоб їх можна було відсортувати внутрішьно), а потім між розподілимо їх між вхідними файлами *нерівномірним чином*, а саме, спробуємо виконати такі умову:

* На кожному кроці злиття (крім останнього) один, і тільки один з вхідних файлів повинен стати порожнім, щоб його використати, як наступний вихідний файл.
* На останньому кроці всі файли повинні злитися в один.

Приклад: Нехай у нас 3 файли і 800 елементів, і ми в памяті компютера можемо зберегти 100 елементів. (припустимо, що підможини зі 100 елементів вже відсортовано за допомогою внутрішнього алгоритма сортування)

Жовтим показані результати злиття.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Відсорт. | Відсорт. | Відсорт. | Відсорт. | Відсорт. |
| Файл 1 | 0 | 3 x 200 | 1 x 200 | 0 | 1 x 800 |
| Файл 2 | 5 х 100 | 2 x 100 | 0 | 1 x 500 | 0 |
| Файл 3 | 3 х 100 | 0 | 2 x 300 | 1 x 300 | 0 |

Кількість підмножин на кожному кроці співпадає з послідовністю Фібоначчі.

Ми виконали 4 злиття, спочатку змінивши кількість підмножин на 5/8, потім на 3/5, 2/3 і врешті ½ — якщо взяти середнє геометричне, побачимо, що в середньому кількість підмножин зменшується в ~1.65 раз. При цьому ми працюємо не з усіма даними, а приблизно з 60% даних. Тобто ефективність кожного кроку — 1.65 / 0.6 = 2.75. Навіть якщо врахувати, що ми виконали 4 кроки замість 3, цей алгоритм все ж ефективніший за звичайне сортування злиттям.

Щоб знайти, як саме розподіляються дані по файлам, зручно працювати з кінця.

В кінці алгоритму маємо 1 відсортовану множину. Знаємо, що це результат злиття 1 підмножини з файлу 2 і 1 підмножини з файлу 3. Отже, на попередньому кроці «додаємо» в файл 2 та 3 по 1 підмножині. Далі, знаємо, що файл 2 — результат злиття 1 підмножини з файлу 1 і ще однієї з файлу 3, отже «додаємо» підмножини туди. Будемо «додавати» підможини до тих пір, поки їх кількість не стане >= INPUT\_SIZE / MAX\_MEMORY.

Для цього можна використати таку функцію:

void polyphase\_merge\_get\_runs(int\*& run\_distribution, int& buffer\_size, int& output\_file\_num,

int input\_size, int file\_count, int max\_memory, int& merges\_count)

{

merges\_count = 0;

run\_distribution = new int[file\_count];

for(int i = 0; i < file\_count; i++)

run\_distribution[i] = 0;

run\_distribution[0] = 1;

int prev\_output\_file\_num = 0;

int total\_runs;

while(true)

{

int runs\_to\_merge = run\_distribution[prev\_output\_file\_num];

for(int i = 0; i < file\_count; i++)

{

if(i == prev\_output\_file\_num)

run\_distribution[i] = 0;

else

run\_distribution[i] += runs\_to\_merge;

}

prev\_output\_file\_num = (prev\_output\_file\_num + 1) % file\_count;

total\_runs = 0;

for(int i = 0; i < file\_count; i++)

total\_runs += run\_distribution[i];

merges\_count++;

if(input\_size < total\_runs \* max\_memory)

break;

}

buffer\_size = input\_size / total\_runs + (input\_size % total\_runs != 0); //divide integers, round up

output\_file\_num = (prev\_output\_file\_num + file\_count - 1) % file\_count;

}

Ця функція обчислює відразу кілька параметрів: кількість кроків (merges\_count), розподілення підмножин (run\_distribution) та найбільш оптимальний розмір буфера (buffer\_size). Розмір буфера дізнаємося, бо можлива така ситуація, що MAX\_MEMORY набагато більше за INPUT\_SIZE / total\_tuns.

Далі ми зчитуємо відсортовані підмножини у кілька буферів, і виконуємо злиття: тобто обираємо найменший елемент з усіх буферів, переписуємо його в буфер для вихідного файлу, і переходимо на наступний buffer\_position. Для сортування елементів з буферів пропоную простий алгоритм insertion sort, він досить добре працює для маленьких масивів.

Взагалі алгоритм поліфазного сортування злиттям досить громіздкий, тому вкладаю окремо .cpp файл, він містить коментарі.

**Інтерфес користувача**

Програма автоматично генерує вхідний файл, сортує його і перевіряє на відсортованість.

**Використані джерела**

<https://en.wikipedia.org/wiki/Polyphase_merge_sort>

<https://en.wikipedia.org/wiki/K-way_merge_algorithm>