Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики
Факультет информационных технологий и программирования
Кафедра «Компьютерные Технологии»

Волков И. Р.

Отчет по курсовой работе
«External Memory: куча»

Санкт-Петербург
2015

Оглавление

1. Введение

1.1. Постановка задачи

Реализовать структуру данных куча, хранящуюся в external memory и работающую эффективно (то есть с оптимальным числом операций чтения и записи в external memory).

1.2. Пояснение сложности задачи

В external memory можно читать и писать только блоками фиксированного размера, поэтому обычная куча работала бы на external memory крайне неэффективно: для каждого обращения к одному любому элементу читался/записывался бы сразу целый блок.

2. Реализация

Для реализации поставленной задачи был выбран язык программирования C++.
Задача была разбита на 3 части:

1. Разработка класса **ExternalStorage**, который занимается взаимодействием с external memory по модели external memory model (чтение и запись возможны только фиксированными блоками).
2. Разработка класса **ExternalHeap**, использующий для взаимодействия с данными **ExternalStorage**.
3. Разработка тестов для класса **ExternalHeap**.

2.1. ExternalStorage

Класс **ExternalStorage** конструируется от двух параметров: имени файла и размера блока. Затем можно использовать следующие методы:

- *readBlock(номер блока)* — читает блок из файла и возвращает, как **vector<T>**, где **T** – это тип данных в блоке (**T** является параметром шаблона для **ExternalStorage**);
- *writeBlock(номер блока, **vector<T>**)* – записывает блок в файл.

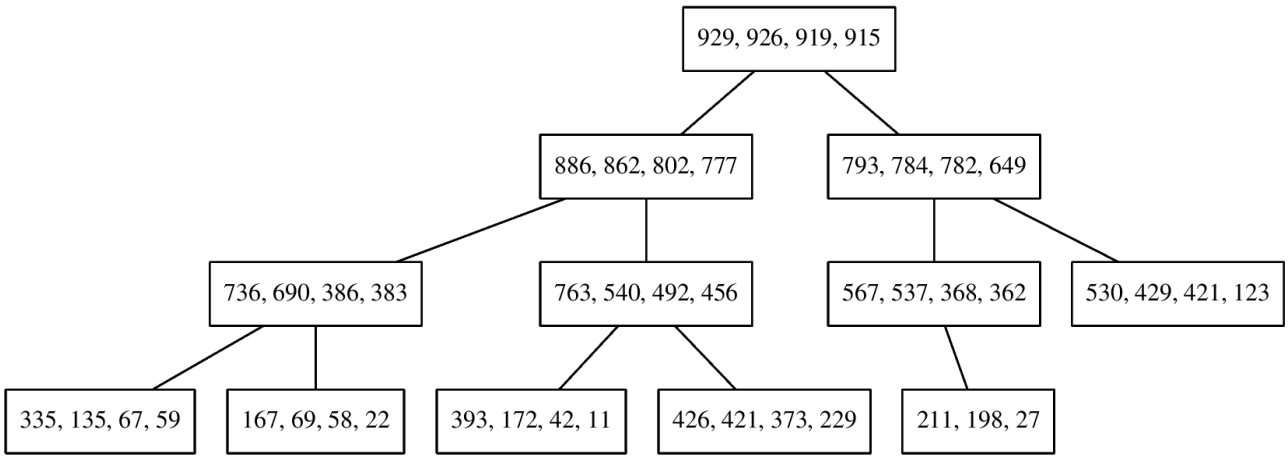
Кроме того, класс **ExternalStorage** умеет считать количество чтений и записей блоков.

2.2. ExternalHeap

Класс **ExternalHeap** является реализацией кучи на external memory.
Поддерживаются следующие свойства кучи:

- Каждая вершина — блок упорядоченных по убыванию элементов кучи.
- Каждый элемент в блоке-родителе больше (нестрого) каждого элемента в блоке-сыне (можно сформулировать иначе: минимальный элемент в блоке-родителе больше или равен максимальному элементу в блоке-сыне).
- Только последняя вершина кучи может быть недозаполнена.

Вот как может выглядеть такая куча, если размер блока — 4 элемента:



Обозначим количество элементов в куче — N, а количество элементов в блоке — B. Тогда количество блоков равно $\frac{N}{B}$ (с округлением вверх), а высота кучи — $\log_2\left(\frac{N}{B}\right)$ (обозначим как H).

Список операций, которые поддерживает **ExternalHeap**:

- *insert(**T**)* – добавление элемента;
- *insert(**vector<T>**)* – добавление блока элементов;
- *size()* – возвращает количество элементов в куче;
- *empty()* – возвращает пуста ли куча;
- *getMax()* – возвращает максимальный элемент кучи;
- *getMaxBlock()* – возвращает блок максимальных элементов;
- *extractMax()* – извлекает максимальный элемент из кучи;
- *extractMaxBlock()* – извлекает блок максимальных элементов из кучи.

Стоить отметить, что операции, работающие с элементами по одному, имеют такое же алгоритмическое время работы (измеряемое в количестве чтений/записей в external memory), как операции, работающие с блоками элементов. Поэтому всегда, когда есть возможность, предпочтительнее работать с блоками (например, для *heapsort*).

Как работают операции, не изменяющие кучу (*size*, *empty*, *getMax*, *getMaxBlock*) – очевидно. Поэтому далее рассмотрим, как работает добавление и извлечение элементов.

2.2.1. Добавление элемента/элементов

Прежде всего, элемент(ы) добавляются в самый конец кучи (как в обычной куче). Затем для вершины (или иногда для двух вершин), в которую они добавились, вызывается private-метод *siftUp*, который поднимает большие элементы наверх для восстановления свойства кучи. Для того, чтобы это сделать, он использует функцию *remerge*, которая в вершину-родитель складывает б**о**льшие элементы, а в вершину-сына — м**е**ньшие. *siftUp* поднимается наверх и восстанавливает свойство кучи до тех пор, пока оно не перестанет нарушаться. Таким образом, *siftUp* производит *O(H)* чтений/записей блоков.

2.2.2. Извлечение максимального элемента/элементов

Максимальные элементы находятся в корневой вершине кучи. После извлечения на их место переносим самые последние элементы кучи, а затем для восстановления свойства кучи вызываем private-метод *siftDown*. Он спускает маленькие элементы вниз. Для того, чтобы это сделать, он вначале использует функцию *remerge* для сыновей вершины (и таким образом, элементы одного из сыновей становятся больше элементов другого), а затем ещё раз использует функцию *remerge*, но в этот раз для вершины и б**о**льшего сына. Теперь свойство кучи может нарушиться только для бывшего б**о**льшего сына и его сыновей. *siftDown* спускается по куче вниз и чинит свойство кучи до тех пор, пока оно не перестанет нарушаться. Таким образом, *siftDown* производит *O(H)* чтений/записей блоков.

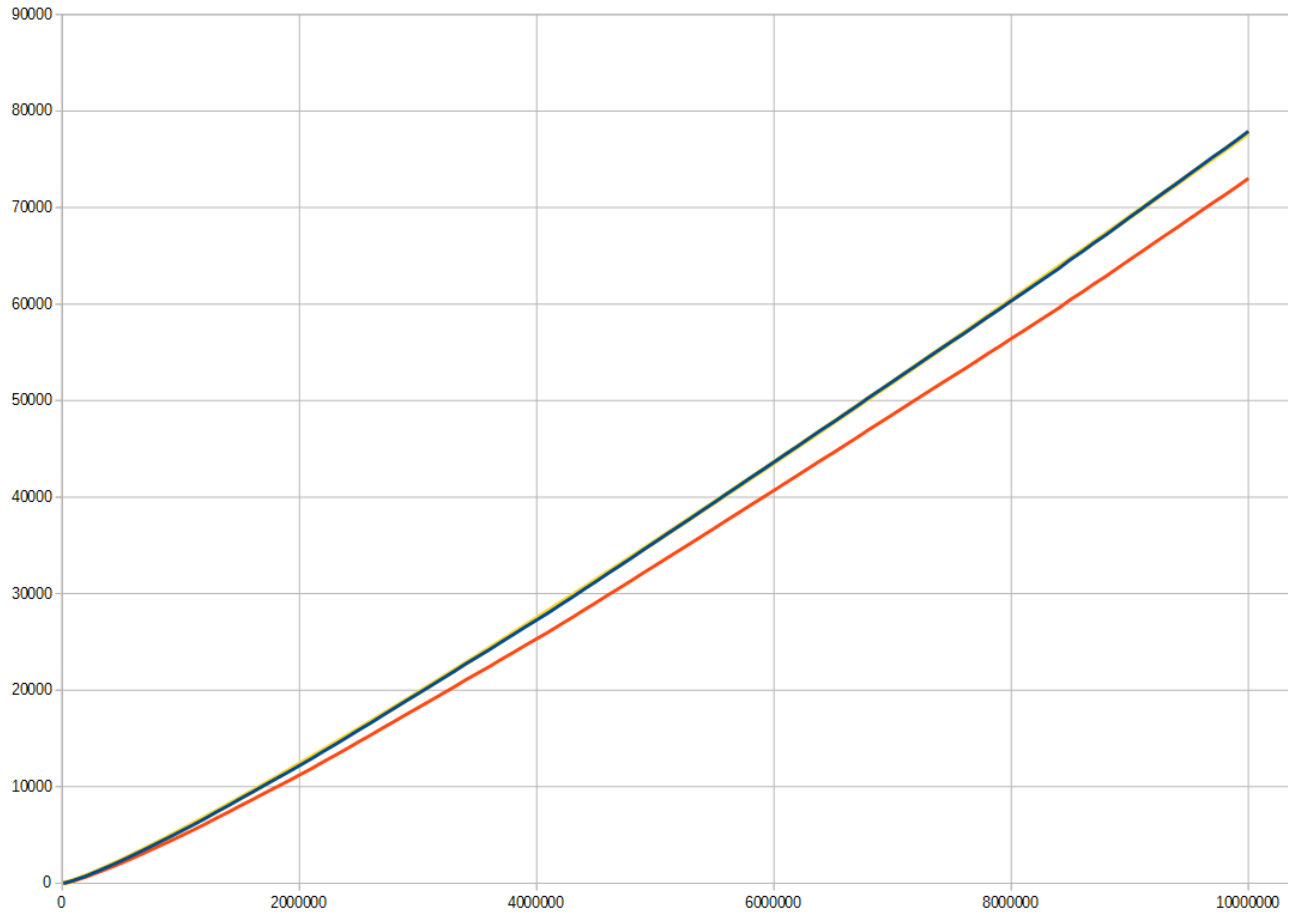
2.3. Тесты

Для тестирования **ExternalHeap** были разработаны следующие unit-тесты:

- Два небольших теста, проверяющих работоспособность основных операций.
- Тест, производящий *heapsort* набора случайных чисел. При этом можно задать длину набора, размер блока и как добавлять/извлекать элементы: блоками или по одному.

3. Результаты

В результате тестирования был построен график зависимости количества операций чтения/записи блоков (взят размер блока 4096 элементов) от размера данных для *heapsort*:



Синяя линия — зависимость количества чтений от числа элементов, красная линия — зависимость количества записей от числа элементов, желтая линия (проходит под синей) построена по формуле $C\frac{N}{4096}\log_2\left(\frac{N}{4096}\right)$, при этом взято $C = 2.83$.

Эти результаты подтверждают, что операции *insert* и *extractMax* работают за *O(H)*.

Результаты также сохранены в файле *results.ods*.

4. Источники

1. Исходный код ExternalStorage, ExternalHeap и тестов
   https://github.com/Tsar/external_heap
2. Описание кучи в external memory
   http://www.diku.dk/forskning/performance-engineering/Jesper/heaplab/heapsurvey_html/node23.html
3. Библиотека для создания unit-тестов (C++)
   https://code.google.com/p/googletest/