ЛР3: Булев индекс

Задание

Требуется построить поисковый индекс, пригодный для булева поиска, по подготовленному в ЛР1 корпусу документов. Требования к индексу:

- Самостоятельно разработанный, бинарный формат представления данных. Формат необходимо описать в отчёте, в побайтовом представлении.
- Формат должен предполагать расширение, так как в следующих работах он будет меняться под требования новых лабораторных работ.
- Использование текстового представления или готовых баз данных не допускается.
- Кроме обратного индекса, должен быть создан «прямой» индекс, содержащий в себе как минимум заголовки документов и ссылки на них (понадобятся для выполнения ЛР4, при генерации страницы поисковой выдачи).
- Для термов должна быть как минимум понижена капитализация.

В отчёте должно быть отмечено как минимум:

- Выбранное внутренне представление документов после токенизации.
- Выбранный метод сортировки, его достоинства и недостатки для задачи индексации.

Среди результатов и выводов работы нужно указать:

- Количество термов.
- Средняя длина терма. Сравнить со средней длинной токена, вычисленной в ЛР1 по курсу ОТЕЯ. Объяснить причину отличий.
- Скорость индексации: общую, в расчёте на один документ, на килобайт текста.
- Оптимальна ли работа индексации? Что можно ускорить? Каким образом? Чем она ограниченна? Что произойдёт, если объём входных данных увеличится в 10 раз, в 100 раз, в 1000 раз?

Метод решения

1. Выбрать алгоритм индексации
2. Выбрать бинарный формат представления данных
3. Реализовать алгоритм
4. Провести анализ решения

- В качестве алгоритма индексации был выбран алгоритм BSBI

- Для сохранения индексов в побайтовом представлении был выбран инструмент Boost.Serialization. Данная библиотека позволяет преобразовывать объекты в последовательность байтов, которая может быть сохранена и загружена для восстановления объектов. Доступны различные форматы данных для определения правил генерации последовательностей байтов.

Результаты выполнения

Процесс индексации коллекции из ~1.8млн документов занимает ~50 минут.
В качестве ограничения используемой индексатором RAM был программно указан лимит в размере 100тыс статей на индекс, что соответствует ~550мб.

Исходный код

```
class BlockedSortBasedIndexer {
public:
    BlockedSortBasedIndexer(std::size_t blockSize);

    void makeIndex(
        const std::filesystem::path& inputFilePath,
        const std::filesystem::path& outputFilePath);
private:
    std::size_t blockSize_;
    std::shared_ptr<tokenization::Tokenizer> tokenizer_;

    BlockProcessor blockProcessor_;
};

}
```

```cpp
namespace {

std::filesystem::path makeFilename(const std::filesystem::path& dir, std::size_t blockNum)
{
    return dir / ("block_" + std::to_string(blockNum) + ".records   ");
}

template<class Iterator>
void cleanTempFiles(Iterator begin, Iterator end)
{
    for (auto it = begin; it != end; ++it) {
        std::remove(it->c_str());
    }
}

void dumpRecords(const std::vector<Record>& records, const std::filesystem::path& outputPath)
{
    common::PerformanceHandler performanceHandler("dump_records");
    std::ofstream ofs(outputPath, std::ios::binary);
    common::serialization::write(ofs, records.size());
    for(const auto& record: records) {
        common::serialization::write(ofs, record);
    }
}

std::vector<document::Document> readNextBlock(const std::size_t blockSize, document::DocumentReader& reader)
{
    common::PerformanceHandler performanceHandler("read_block");
    std::vector<document::Document> block;
    block.reserve(blockSize);
    for(int i = 0; i < blockSize && !reader.allDocsRead(); ++i) {
        block.push_back(reader.readNext());
    }
    return block;
}

}
```

```cpp
BlockedSortBasedIndexer::BlockedSortBasedIndexer(std::size_t blockSize)
    : blockSize_(blockSize)
    , tokenizer_(std::make_shared<tokenization::SimpleTokenizer>())
{
}

void BlockedSortBasedIndexer::makeIndex(
    const std::filesystem::path& inputFilePath,
    const std::filesystem::path& outputDirectory)
{
    common::PerformanceHandler performanceHandler("make_index");

    if (!std::filesystem::exists(outputDirectory)) {
        std::filesystem::create_directories(outputDirectory);
    }
    assert(std::filesystem::is_directory(outputDirectory));

    document::DocumentReader reader(inputFilePath, blockSize_, tokenizer_.get());

    std::size_t blocksProcessed = 0;
    std::vector<std::filesystem::path> tempFiles;
    auto documentIndex = document::createDocumentIndex();

    while (!reader.allDocsRead()) {
        auto block = readNextBlock(blockSize_, reader);

        for(const auto& doc: block) {
            documentIndex->appendDocument(doc);
        }

        auto records = blockProcessor_.processBlock(block.begin(), block.end());

        std::sort(records.begin(), records.end());

        ++blocksProcessed;

        tempFiles.push_back(makeFilename(outputDirectory, blocksProcessed));
        dumpRecords(records, tempFiles.back());
    }

    RecordMerger::merge(tempFiles, outputDirectory);
    cleanTempFiles(tempFiles.begin(), tempFiles.end());

    std::ofstream dictOfs(outputDirectory / "dict.bin");
    common::serialization::write(dictOfs, blockProcessor_.dictionary());

    std::ofstream documentIndexStream(outputDirectory / "document_index.bin");
    common::serialization::write(documentIndexStream, *documentIndex);
```

Выводы:

В результате выполнения данной лабораторной работы нам удалось проиндексировать документы новостей из wikinews. Процесс оказался достаточно трудоемким в следствие особенностей

языка c++. Понравилось использовать библиотеку boost как для операций с файловой системой, так и для сериализации/десериализации данных.