# Часть 1

Зачем нужен класс InvertedIndex?

Обратный (или инвертированный) индекс широко применяется в реальных поисковых системах — https://ru.wikipedia.org/wiki/Инвертированный\_индекс. Идея очень проста. Допустим, у нас есть N документов, каждый из которых содержит не более K слов, а также поисковый запрос, состоящий из Q слов. Если каждое из Q слов поискать в каждом документе, то это будет иметь асимптотику O(QNK) . Если же мы построим обратный индекс, то есть для каждого слова построим список документов, в которых оно встречается, то поисковый запрос будет обрабатываться за O(QN) . Таким образом, мы достигаем лучшей асимптотики обработки поискового запроса.

Измеряйте

Мы говорили, что для поиска узких мест в программе надо проводить измерения. Если вам не удаётся найти узкое место, измерьте, что работает дольше: построение индекса или обработка поисковых запросов.

Узкое место в обработке запросов

По нашим замерам получилось, что обработка запросов занимает гораздо больше времени, чем построение индекса. Теперь надо найти узкое место в методе AddQueriesStream . Для этого разобьём обработку запросов на 4 стадии: разбиение на слова обращение к индексу сортировка документов формирование результата Померяйте, какая из этих четырёх стадий занимает больше всего времени при обработке запросов.

Ускоряем подсчёт hitcount'а

По нашим замерам получилось, что медленнее всего работает подсчёт hitcount'а для документов, а именно вот этот кусок кода:

map docid\_count;

for (const auto& word : words)

{

For (const size\_t docid : index.Lookup(word)) {

docid\_count[docid]++; }

}

Давайте посмотрим, какую он имеет асимптотику. Если запрос состоит из Q слов и всего у нас имеется N документов, то асимптотика этого кода получается O((QN log N) \* f(Lookup)) , где f(Lookup) — это асимптотика метода Lookup у класса InvertedIndex . Подумайте, как нам уменьшить её до O(QN \* f(Lookup)) , учитывая, что у N <= 50 000 и идентификаторы документов — это целые числа от 0 до N - 1 . Избавление от log N в асимптотике в данном случае весьма важно. По условию задачи N <= 50 000 , следовательно log N <= 16 . То есть мы можем ускорить этот цикл в 16 раз. Так как он выполняется для каждого поискового запроса, то и обработка всех запросов может быть, в теории, ускорена в 16 раз

Замена map на вектор

Так как всего документов не больше 50 000, то мы можем заменить map docid\_count на vector , это снизит асимптотику обращения к индексу до O(QN \* f(Lookup)) . Кроме того, мы можем вынести вектор за пределы цикла for (string current\_query; getline(query\_input, current\_query); ) , чтобы один раз выделить под него память, а не делать это на каждой итерации цикла.

Ускорение сортировки

После замены map'а на вектор, по нашим замерам, узким местом в обработке запросов стала сортировка. Мы в ней выполняем сортировку всего вектора из 50 000 элементов, хотя нам нужно только 5 наиболее релевантных документов. Можем ли мы заменить алгоритм sort на какой-то другой стандартный алгоритм, который позволяет быстрее находить 5 наиболее релевантных документов?

Ускорение сортировки

Для ускорения сортировки документов можно воспользоваться алгоритмом partial\_sort. Алгоритм sort позволяет найти первые пять максимумов за O(N log N) , а алгоритм partial\_sort — за O(N) .