В этой задаче вам надо поработать с классом SearchServer, который позволяет выполнять поиск в базе документов:

```
class SearchServer {
public:
  SearchServer() = default;
  explicit SearchServer(istream& document_input);

  void UpdateDocumentBase(istream& document_input);
  void AddQueriesStream(istream& query_input, ostream& search_results_output);
};
```

Рассмотрим его интерфейс.

Конструктор
Конструктор класса SearchServer принимает поток ввода, содержащий базу документов. При этом

- один документ — это одна строка входного потока;
- документы состоят из слов, разделённых одним или несколькими пробелами;
- слова состоят из строчных латинских букв. Например, код, приведённый ниже, загружает в объект класса SearchServer базу из трёх документов:

Метод AddQueriesStream(istream& query_input, ostream& search_results_output)
Метод AddQueriesStream выполняет собственно поиск. Он принимает входной поток поисковых запросов и выходной поток для записи результатов поиска. При этом

- один запрос — это одна строка в потоке query_input
- каждый поисковый запрос состоит из слов, разделённых одним или несколькими пробелами
- так же, как и в документах, слова в запросах состоят из строчных латинских букв

Результатом обработки поискового запроса является набор из **максимум пяти** наиболее релевантных документов. В реальных поисковых системах метрика релевантности устроена довольно сложно. В рамках нашей задачи в качестве метрики релевантности мы будем использовать суммарное количество вхождений всех слов запроса в документ. Например, допустим, у нас есть поисковая база из трёх документов: "london is the capital of great britain", "moscow is the capital of the russian federation", "paris is the capital of france", — и поисковый запрос "the best capital". Тогда метрика релевантности у наших документов будет такой:

- london is the capital of great britain — 2 (слово "the" входит в документ 1 раз, слово "best" — ни разу, слово "capital" — 1 раз)
- moscow is the capital of the russian federation — 3 (слово "the" входит в документ 2 раза, слово "best" — ни разу, слово "capital" — 1 раз)
- paris is the capital of france — 2 ("the" — 1, "best" — 0, "capital" — 1)

В итоге получается, что документ "moscow is the capital of the russian federation" оказывается наиболее релевантным запросу "the best capital".

Для каждого поискового запроса метод AddQueriesStream должен вывести в поток search_results_output одну строку в формате [текст запроса]: {docid: <значение>, hitcount: <значение>} {docid: <значение>, hitcount: <значение>} ..., где docid — идентификатор документа (см. ниже), а hitcount — значение метрики релевантности для данного документа (то есть суммарное количество вхождений всех слов запроса в данный документ).

Два важных замечания:

- Добавлять в результаты поиска документы, hitcount которых равен нулю, не нужно.
- при подсчёте hitcount нужно учитывать только слова целиком, то есть слово «there» не является вхождением слова «the»

Метод UpdateDocumentBase(istream& document_input)
Метод UpdateDocumentBase заменяет текущую базу документов на новую, которая содержится в потоке document_input. При этом документ из первой строки этого потока будет иметь идентификатор (docid) 0, документ из второй строки — идентификатор 1 и т.д. Точно так же должен назначать идентификаторы документам и конструктор класса SearchServer. Например, код

```
const string doc1 = "london is the capital of great britain";

const string doc2 = "moscow is the capital of the russian federation";

istringstream doc_input1(doc1 + '\n' + doc2);

SearchServer srv(doc_input1);


const string query = "the capital";

istringstream query_input1(query);

srv.AddQueriesStream(query_input1, cout);


istringstream doc_input2(doc2 + '\n' + doc1);

srv.UpdateDocumentBase(doc_input2);

istringstream query_input2(query);
```

```
srv.AddQueriesStream(query_input2, cout);
```

должен выводить

```
the capital: {docid: 1, hitcount: 3} {docid: 0, hitcount: 2}
```

```
the capital: {docid: 0, hitcount: 3} {docid: 1, hitcount: 2}
```

Первая часть задачи

Эта задача состоит из двух частей. В первой части вам дана корректная реализация класса SearchServer, которая работает недостаточно быстро. Вам нужно найти и устранить узкие места в её реализации. В тестирующую систему нужно сдать cpp-файл или архив из нескольких исходных файлов, содержащий вашу ускоренную реализацию.

Тестирование реализации будет проводиться на производительность и на целостность.

При тестировании на производительность, Ваша реализация будет тестироваться вот такой функцией (объект класса SearchServer будет создан один раз, и у него один раз будет вызван метод AddQueriesStream):


```
}
```

При этом:

- document_input содержит не более 50000 документов
- каждый документ содержит не более 1000 слов
- общее число различных слов во всех документах не превосходит 15000
- максимальная длина одного слова — 100 символов, слова состоят из строчных латинских букв и разделены одним или несколькими пробелами
- query_input содержит не более 500 000 запросов, каждый запрос содержит от 1 до 10 слов.

В отличие от тестирования на производительность, при тестировании на целостность и корректность работы исходного функционала сервера допускается вызов UpdateDocumentBase в том числе и у уже созданного и проинициализированного объекта класса SearchServer.