Описание задания №1 «Поиск шаблонов в тексте»

Версия 1.0

10 октября 2014 г.

1 Описание базового задания

Общий план задания — надо реализовать несколько классов, ищущих набор подстрок в тексте. Все классы наследуются от базового интерфейса (см. далее).

Задание состоит из подзаданий. Подзадание — набор требований, которым должен удовлетворять очередной класс. Чтобы сдать подзадание, нужно написать набор тестов, проверяющих требования, и собственно класс, который эти тесты проходит. Набор тестов должен быть максимально полным и тестировать весь функционал данного класса. Разрешается реализовывать дополнительные методы, но их тоже надо тестировать. Разрешается строго не следовать названиям, которые используются в этом документе. Разрешаются минимальные отклонения от предлагаемого дизайна базовых классов.

К заданию надо приложить файл с описанием того, где находится реализация того или иного задания.

Помимо собственно тестов, некоторые баллы начисляются за качество кода класса (5 баллов за задание, если не указано иное). Стиль кода разрешается любой внутренне непротиворечивый (возможно, отличный от используемого в этом документе).

2 Базовый интерфейс

2.1 ICharStream

```
class ICharStream {
public:
    virtual char GetChar();
    virtual bool IsEmpty() const;
};
```

Интерфейс `ICharStream` реализует поток символов.

`GetChar()` — получить следующий символ из потока. Если поток пуст, вызов этой функции должен бросить исключение.

`IsEmpty()` — узнать, пуст ли поток.

Запрещается создавать бесконечные потоки.

2.2 IMetaTemplateMatcher

```
typedef int TStringID;
typedef std::pair<size_t, TString> TOccurence;
typedef std::vector<TOccurence> TMatchResults;

class IMetaTemplateMatcher {
public:
    virtual TStringID AddTemplate(const std::string &template);
    virtual TMatchResults MatchStream(ICharStream &stream);
};
```

`AddTemplate(template)` — добавить строку `template` в набор шаблонов, вернуть уникальный идентификатор для этой строки. Если строка уже присутствует в наборе, кинуть исключение.

`MatchStream(input)` — обработать поток символов `input`, вернуть структуру с перечнем всех вхождений всех шаблонов из текущего набора. Элемент `TMatchResults`, равный (i, j), означает, что в i-ом символе заканчивается шаблон с идентификатором j. Можно считать, что все символы лежат в диапазоне кодов ASCII от 32 до 255 (если это не так, кинуть исключение; не забудьте при этом, что `char` — знаковый тип).

3 Подзадания базового задания

Если с классом пытаются выполнить некорректное действие, надо бросить информативный exception с описанием ошибки. В частности, если действие в общей постановке задачи разрешено, но в данном подзадании его обрабатывать не требуется (например, гарантируется, что все строки в словарь добавляются заранее, но вам пытаются добавить строку после обработки текста), надо бросить exception типа `TNotSupportedException` или подобного с информативным сообщением (например, `"adding templates after processing streams is not supported in TStaticTemplateMatcher"`).

Обозначения: $streamSize$ — количество символов потока, $templatesSize$ — суммарное количество символов всех шаблонов, $templatesCount$ — количество шаблонов, $totalOccurencesCount$ — суммарное количество вхождений всех шаблонов.

Расход памяти для всех классов, реализующих `IMetaTemplateMatcher` должен составлять $O(templatesSize)$ вне обработки потока, может достигать $O(templatesSize+totalOccurencesCount)$ при обработке потока (если явно не указано иное).

3.1 Наивный поиск (`TNaiveTemplateMatcher`, 5 баллов)

- Суммарное время добавления шаблонов — $O(templatesSize)$.
- Время обработки потока должно составлять $O(streamSize \cdot templatesSize)$.

3.2 Поиск одного шаблона (`TSingleTemplateMatcher`, 10 баллов)

- Во время обработки всех потоков $templatesCount = 1$.
- Время добавления шаблона и предобработки — $O(templatesSize)$.
- Время обработки потока — $O(inputSize)$.

3.3 Статический поиск нескольких шаблонов (`TStaticTemplateMatcher`, 20 баллов)

- Суммарное время добавления шаблонов и предобработки — $O(templatesSize)$.
- Все добавления строк идут перед обработкой потоков.
- Время обработки потока — $O(inputSize)$.

4 Бонусные задания

Бонусное задание может состоять в реализации нового класса, либо добавлении метода к одному из базовых (в этом случае тесты базового класса не должны ломаться).

4.1 `TSingleTemplateMatcher::AppendCharToTemplate`, 5 баллов

`TSingleTemplateMatcher::AppendCharToTemplate(char c)` — добавить символ в конец шаблона. Если шаблон еще не добавлен, кинуть исключение.

Все требования к базовому классу сохраняются.

- Время работы метода — амортизированно $O(1)$.
- Вызовы `AppendCharToTemplate` могут чередоваться с обработкой потоков.

4.2 `TSingleTemplateMatcher::PrependCharToTemplate`, 5 баллов

`TSingleTemplateMatcher::PrependCharToTemplate(char c)` — добавить символ в начало шаблона. Если шаблон еще не добавлен, кинуть исключение.

Все требования к базовому классу сохраняются.

- Время работы метода — амортизированно $O(1)$.
- Вызовы `PrependCharToTemplate` могут чередоваться с обработкой потоков.

4.3 `TWildcardSingleTemplateMatcher`, 10 баллов

Реализует интерфейс `IMetaTemplateMatcher`. На момент обработки потоки $templatesCount = 1$. При сопоставлении шаблона подстроке текста символ шаблона «`?`» может соответствовать любому символу текста.

Разрешается, чтобы во время обработки потока расход памяти и времени достигал $O(k \cdot inputSize)$, где k — количество символов «`?`» в шаблоне. В остальном, применимы все требования `TSingleTemplateMatcher`.

4.4 `TDynamicTemplateMatcher`, 20 баллов

Реализует `IMetaTemplateMatcher`. Запросы `AddTemplate` и `MatchStream` могут идти в любом порядке.

- Суммарное время добавления шаблонов — $O(templatesSize \cdot \log templatesCount)$.
- Время обработки потока — $O(inputSize \cdot \log templatesCount)$.

4.5 `T2DSingleTemplateMatcher`, 25 баллов

Общая задача: найти все вхождения заданной подматрицы матрицы символов A, совпадающие с матрицей B. Обе матрицы можно задавать в виде `std::vector<std::string>`. Время работы и расход памяти должны составлять $O(|A| + |B|)$. Интерфейс класса разработайте самостоятельно.