Оглавление

[Введение 2](#_Toc480463473)

[1. Аналитический раздел 4](#_Toc480463474)

[1.1 Описание предметной области 4](#_Toc480463475)

[1.2 Формализация постановки задачи 9](#_Toc480463476)

[1.3 Методы доступа к данным 10](#_Toc480463477)

[*1.3.1* Open Database Connectivity (ODBC) 10](#_Toc480463478)

[*1.3.2* Object Linking and Embedding (OLE DB) 12](#_Toc480463479)

[*1.3.3* Microsoft ActiveX Data Objects (ADO) 15](#_Toc480463480)

[1.4 Метод обработки данных 16](#_Toc480463481)

[1.5 Алгоритмы поиска информации 18](#_Toc480463482)

[*1.5.1* Алгоритм Кнута-Морриса-Пратта 18](#_Toc480463483)

[*1.5.2* Алгоритм Бойера-Мура 19](#_Toc480463484)

[*1.5.3* Двоичный алгоритм поиска подстроки 22](#_Toc480463485)

[Выводы 23](#_Toc480463486)

[Приложение А 24](#_Toc480463487)

# Введение

Бортовое радиоэлектронное оборудование (БРЭО) ***[\*]*** современного боевого летательного аппарата (ЛА) выполняет огромное количество многочисленных задач по обеспечению пилотирования, навигации, коммуникации и безопасности. С каждым годом, все больше и больше БРЭО на борту воздушных судн оснащаются встроенными вычислительными системами, начиная от поисковых прожекторов и заканчивая современными радарами. Объем программного обеспечения таких систем постоянно увеличивается и усложняется, в следствии чего растет потребность в изменениях и доработках.

Каждые доработки нуждаются в тщательном тестировании, поскольку любая неисправность БРЭО при испытаниях может повлечь за собой гибель летчика-испытателя. Затраты на корректировку таких систем могут достигать нескольких лет, что существенно замедляет процесс внедрения в эксплуатацию, поэтому поднимаются вопросы частичной автоматизации проверки работы оборудования. Создаются различные программные обеспечения и разрабатываются новые технологии тестирования, которые значительно сокращают время и минимизируют человеческий труд.

В связи с этим, актуальным является задача автоматизации методов и процессов верификации программного обеспечения. Полная ручная проверка достоверности системных комплексов для ЛА слишком трудоемкий процесс, который требует огромное количество человеческого ресурса и времени. Внедрение вспомогательных инструментальных средств позволит его сэкономить и частично исключить человеческий фактор, что приведет к резкому сокращению ошибок.

Целью работы является разработка и реализация унифицированного программного обеспечения для работы с тестовыми артефактами тестирования индикации. В рамках работы решаются следующие задачи:

* анализ задачи обработки информации из тестовых артефактов;
* анализ алгоритмов, позволяющих осуществлять оперативную корректировку данных в тестовых артефактах;
* тестирование программного обеспечения;
* анализ методов интеграции в комплекс программного обеспечения тестового автомата;

Разработка данного программного обеспечения позволит частично автоматизировать процесс подготовки тестовых артефактов для тестового автомата, что даст возможность уделять больше времени самому тестированию комплексов БРЭО.

# Аналитический раздел

В этом разделе производится анализ предметной области, рассматриваются различные подходы к обработке тестовых артефактов и побочные алгоритмы, необходимые для работы с тестовыми артефактами. Также в этом разделе производится формализация поставленной задачи.

## Описание предметной области

Основным источником информации для летчика современного боевого ЛА являются многофункциональные индикаторы (МФИ) ***[\*]***, оснащенные встроенной вычислительной системой. *МФИ* – средство отображения различной графической или телеинформации, поступающей от бортовых информационных систем, о количественном или качественном значении параметров. Для внутренней автоматизации тестирования функционального программного обеспечения индикаторов были разработаны специальный язык тестового сценария и тестовый автомат.

*Test Script Language (TSL)* – инструмент написания тестовых сценариев, призванный облегчить написание скриптов и уменьшить вероятность появления ошибок при написании тестовых процедур. Отличительной особенностью является возможность использовать основные конструкции языка С++: функции, циклы, структуры и массивы. Основные команды сценария:

* ITEM\_EX – начало тестового сценария.
* END\_ITEM\_EX – окончание тестового сценария.
* SET – установка значения сигнала.
* CHECK – запрос у оператора результата сравнения ожидаемого и реального результата.
* DISPLAY – вывод в окно тестового автомата информации для оператора.
* SPEC – описание проверяемых требований в тесте.
* DisplaySpecMessage – описание и отображение данных спецификаций с созданием матрицы трассировки проверяемых требований в отчете.
* SetAndDisplay – задание значения сигналов вместе с описанием заданных значений в инструкции оператору.
* DoTesting – выполнение тестового объекта.

Основным элементом тестового сценария, служащим проверить достоверность отображения информации, является тестовый объект. *Тестовый объект* ***[\*]*** – структура, заполненная ожидаемыми значениями отображения в соответствии со спецификацией требований программного обеспечения (СТПО). Структура содержит наименование теста, количество принимаемых индикатором параметров, их тестовые результаты и значения. На рисунке 1.1 приведен пример заполнения тестового объекта. Окончание ввода тестовых результатов и значений характеризуются соответствующими терминаторами *WTR\_TERM* и *OTD\_TERM*.

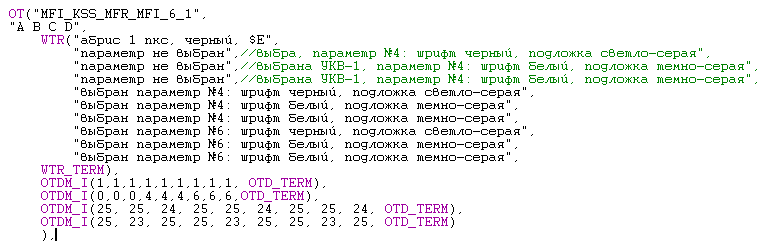
****

Рис. 1.1 – Пример заполнения тестового объекта.

*Тестовый автомат («Тестер»)* – утилита, позволяющая проводить тестирование МФИ в соответствии со сценарием. Компьютер, оснащенный тестовым автоматом, подключается через штатные каналы информационного взаимодействия к многофункциональным индикаторам. В соответствии со сценарием проверки, программа подает на вход индикатора набор сигналов для каждого теста, и МФИ производит отображение информации. Тестировщик сравнивает ожидаемый результат с полученным и принимает решение о ее корректности. По окончанию всех тестов программное обеспечение выдает отчет.

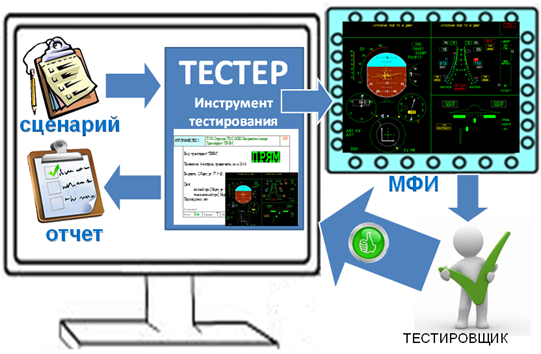


Рис. 1.2 – Работа тестового автомата («Тестера»).

Для корректного отображения информации на МФИ «Тестер» должен содержать в себе подробную информацию о существующих сигналах индикатора, чтобы при поступлении набора от тестового сценария изменить соответствующие сигналы на индикаторе. Данные характера содержат протоколы информационного взаимодействия (ПИВ). ПИВ – файл, формата Microsoft Office Excel, содержащий таблицы параметров информационного взаимодействия, выполненные в формате специальной электронной формы (СЭФ). Обработанные протоколы являются основными тестовыми артефактами.

Как видно из «Приложения А» протокол информационного взаимодействия содержит заголовок в виде таких параметров, как:

* Источник – устройство, передающее сигналы;
* Режим – способ передачи данных (симплексный, полудуплексный, дуплексный, мультиплексный);
* Приемник – устройство, принимающее сигналы;
* Линия – соединение отдельных функциональных блоков между собой. Состав и количество линий определяет возможные способы передачи данных.

Таблица содержит в себе необходимые характеристики сигналов, от наименования сигнала до номера набора параметров, в котором он передается. Для дополнительной информации присутствует поле «Примечание», в котором содержатся различные пояснения к сигналу, вплоть до описания логики. Правила составления ПИВ в формате СЭФ определены стандартом предприятия (СТО 730.0162-2012).

Подготовка протоколов информационного взаимодействия для тестового автомата производится в несколько этапов:

* Проверка оформления ПИВ (на соответствие стандартам предприятия);
* Оперативная корректировка параметров сигналов, если имеется несоответствие данных;
* Преобразование информации о каждом сигнале в формат, необходимый тестовому автомату, в виде текстового документа (\*.txt).

Для работы с информацией о сигналах, тестовому автомату необходимо принимать каждый сигнал отдельно. Поэтому протоколы информационного взаимодействия в виде таблицы преобразовываются в формат текстового документа, где каждый сигнал имеют специальную структуру. В дальнейшем тестовый автомат преобразовывает сигналы в двоичный код.

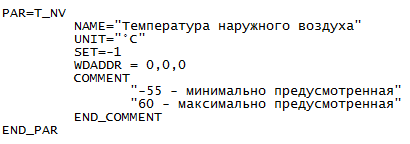


Рис. 1.3 – Формат сигнала, требуемый тестовым автоматом.

На рисунке 1.3 приведен формата сигнала, необходимый тестовому автомату, где *PAR, END\_PAR* – начало и окончание блока параметра, содержащего всю доступную информацию о сигнале, *NAME* – наименование сигнала, *UNIT* - размерность, *SET* – номер подкадра (-1, если в кадре отсутствуют подкадры), *WADDR* – адрес параметра в наборе, *COMMENT, END\_COMMENT* – начало и окончание блока примечания.

Данные этапы, на сегодняшний день, производятся вручную. На работу с одним протоколом уходит порядка двух месяцев у одного сотрудника, поэтому был поставлен вопрос о создании программного обеспечения для улучшения условий работы с протоколами информационного взаимодействия.

## Формализация постановки задачи

Основываясь на проведенном анализе предметной области, постановка решаемой в данной работе задачи может быть сформулирована следующим образом.

Необходимо реализовать программное обеспечение для работы с протоколами информационного взаимодействия. Входными данными являются файлы формата Microsoft Excel, содержащие в себе таблицы значений. Данные таблицы должны быть проверены на соответствие стандартам предприятия. Для оперативного изменения данных требуется редактор с функцией поиска сигналов, так как в одном протоколе их насчитывается огромное количество. Выходными данными являются текстовые документы, содержащие в себе информацию о сигналах в формате, требуемом тестовым автоматом.

Также необходимо рассмотреть методы интеграции данного ПО в программный комплекс отдела тестирования.

## Методы доступа к данным

Таблицы протоколов информационного взаимодействия содержатся в файлах Microsoft Excel. Для работы с данными у компании Microsoft имеется несколько собственных API. В данном подразделе будет выполнен обзор этих программных интерфейсов, с целью последующего выбора оптимального подхода.

### Open Database Connectivity (ODBC)

Стандарт открытого доступа к базам данных является типовым программным интерфейсом для построения приложений управления БД в среде Microsoft Windows. Основная идея состоит в использовании специализированных драйверов баз данных подобно тому, как Windows применяет драйвера для работы различных устройств. Разработчики могут создавать свои собственные драйвера ODBC или же использовать разработанные другими фирмами. Список баз данных, для которых Microsoft уже разработало специализированные драйвера, можно посмотреть в***[\*]***.

Технология ODBC позволяет разработчику создавать и распространять клиент-серверные программы, не зависящие от конкретной системы управления базами данных. Всю привязку к определенной СУБД выполняет диспетчер драйверов. *Диспетчер драйверов* ***[\*]*** – библиотека динамической загрузки, которая осуществляет доступ к БД. Основной функцией диспетчера является загрузка необходимого драйвера. Также он выполняет обработку некоторых инициализированных и информационных вызовов ODBC, передачу вызовов функций ODBC от приложения драйверу и контроль состояния. При необходимости, диспетчер может регистрировать в журнале все вызовы функций ODBC приложением. В журнале записывается имя каждой свободной от ошибок функции вместе со значениями входных аргументов и именами выходных. Прежде чем передать вызов драйверу, отвечающем за подключение к базе данных, диспетчер проверяет аргументы функций и корректность изменения состояния, а также другие условия отсутствия ошибок. Это дает возможность освободить БД от обработки большинства ошибок.

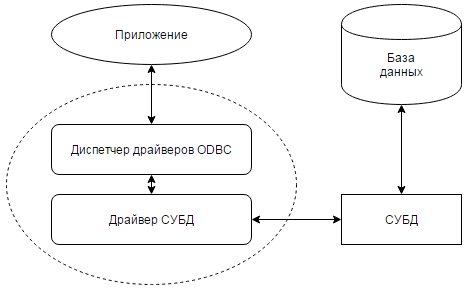


Рис. 1.4 – Архитектура ODBC.

Для установления соединения и управления драйвером СУБД, диспетчеру требуется информация о его местонахождении и типе драйвера. Данную информацию предоставляет именованный ресурс ODBC – DSN (Data Source Name). Существуют три типа именованного ресурса DSN: пользовательский, системный и файловый. Различия заключается в методе хранения требуемой информации и ее доступе. Информация для пользовательского DSN содержится в реестре Windows того компьютера, на котором создано соединение с данными. Системные DNS относятся к компьютеру, а не к пользователю. Таким образом, система или пользователь, обладающий соответствующими привилегиями, необходимыми для доступа к источнику данных, имеют право работать с БД. Информация для файлового DNS хранится в файле, доступном для чтения любому приложению.

Основная задача драйвера СУБД обрабатывать запросы, поступающие от приложения и преобразовывать их в набор команд API СУБД и, таким образом, производить какие-либо действия с базой данных. Также он отвечает за то, чтобы стандартные команды ODBC выполнялись корректно, поскольку в некоторых случаях источник данных не поддерживает некоторые функции. В таких ситуациях их выполняет драйвер СУБД. Также на драйвере лежит функция приведения кодов ошибок, поступающих от источника, к стандартным в ODBC.

Определяют два типа драйверов – одноуровневые и многоуровневые. Одноуровневые обрабатывают вызовы ODBC и операторы SQL. Многоуровневые обрабатывают только вызовы ODBC, оставляя право за СУБД осуществлять обработку SQL-запросов.

### Object Linking and Embedding (OLE DB)

Технология OLE DB представляет из себя низкоуровневый интерфейс для доступа к базам данным, разработанный компанией Microsoft. Отличительно особенностью является наличие доступа к нереляционным данным, таким как почтовые сообщения, электронные таблицы, текстовые документы и многие другие. В спецификации OLE DB определен набором COM-интерфейсов (*Component Object Model* ***[\*]*** – технологический стандарт компании Microsoft, предназначенный для создания программного обеспечения на основе взаимодействующих компонентов), инкапсулирующих различные сервисы управления данными и предоставляющих однотипный доступ.

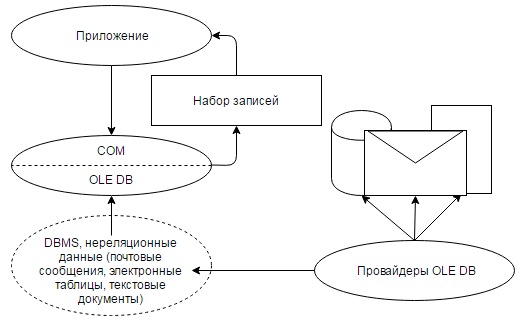


Рис. 1.5 – Архитектура OLE DB.

Рассматривая работу интерфейса OLE DB, выделяют две сущности: потребители и поставщики. В роли потребителей служат приложения, которым необходимо получить данные из долговременного источника. Для доступа нужно выбрать поставщика, соответствующего типу базы данных, к которой требуется доступ, и использовать базовые интерфейсы для взаимодействия, описанные в ***[\*]***.

*Поставщик (провайдер)* – набор COM-объектов, обслуживающих вызовы интерфейса из пользовательского объекта. Основная задача заключается в передаче информации в табличном формате из хранилища данных приложению-получателю. Каждый из COM-объектов содержит несколько интерфейсов, которые подразделяются на обязательные и необязательные. С помощью реализации обязательных провайдер гарантирует минимальный уровень функциональности. В OLE DB интерфейсы определяются для нескольких типов объектов:

* Источник данных;
* Сеанс;
* Набор строк;
* Команда;
* Транзакция.

Все объекты, кроме строк и хранилищ, реализуются с помощью шаблонов поставщика. *Шаблоны* – предварительно упакованные реализации для всех необходимых интерфейсов.

Когда приложению-потребителю требуются данные, оно создает объект источника данных с целью запуска поставщика. Он содержит сведения о проверке подлинности авторизации, которые используются для предоставления разрешений потребителю. Используя интерфейс *IDBCreateSession*, объект источника данных создает один или несколько сеансов, с помощью которых потребитель подключается к источнику данных (рисунок 1.6). Каждый сеанс управляет своими взаимодействиями с базой данных независимо от других существующих сеансов и для управления инициализирует объекты команд (или таблиц) и набора строк.



Рис. 1.6 – шаблон поставщика OLE DB.

Наборы строк не предоставляют доступа к хранилищу данных, а лишь содержат информацию о нем – метаданные. При представлении информации используются три вида наборов: одиночный набор, групповой набор и массивный набор. Доступ к строкам осуществляется через класс команд (или таблиц). С помощью него выполняются базовые операции для работы с наборами строк (обновление, удаление, вставка). Запись и извлечение данных происходит через объект метода доступа, который содержит набор привязок между полями набора строк и элементами данных, объявленные в потребителе.

### Microsoft ActiveX Data Objects (ADO)

*Microsoft ActiveX Data Objects (ADO)* – набор программных объектов, построенных по технологиям ActiveX (COM), позволяющий получать данные и управлять ими на самых разных источниках. ADO является высокоуровневым интерфейсом, и для работы с источником использует разные подключения к базам данных, в том числе OLE DB и ODBC.

Объектная модель ADO ***[\*]*** достаточно проста, и содержит всего три главных объекта:

* Объект *Connection* – позволяет установить соединение с источником данных и управлять им. Ошибки, которые возникают при попытке подключения помещаются в сопутствующую коллекцию *Errors*.
* Объект *Command* – представляет набор команд, с помощью которых производится выполнение определенных операций на хранилище данных (выполнение запроса, создание или изменение объекта, изменение данных и т.п.). Если источник данных является SQL-совместимым, то объект *Command* представляет SQL-команды. Данному объекту сопутствует коллекция *Parameters* – наборы параметров, передающихся запросу или хранимой процедуре.
* Объект *Recordset* – представляет набор строк, полученных с источника или сгенерированный другим способом. Ему сопутствует коллекция *Fields*, которая представляет информацию о столбцах в этом набор строк (имя, тип, размерность и т.п.), а также сами данные.

Для каждого из этих трех объектов предусмотрена коллекция *Properties,* которая определяет соответственно свойства соединения, команды или набора строк.

## Метод обработки данных

Обработка таблиц протоколов информационного взаимодействия основывается на проверке соответствия спецификациям требований к изложению, построению и оформлению. Решение задачи напрямую, а именно посимвольное сравнение строк и дальнейшего разбора на ошибки значений сигналов, является неэффективным. Исходя из этого поставлена задача разработки метода обработки данных. Для его создания требовалось изучить основные положения в спецификациях к построению и оформлению. При анализе была выявлена четкая постановка инструкций по построению таблиц, поэтому основная идея будущего алгоритма лежит в использовании шаблонов.

Конструкции шаблонов строятся на основе отношений – неких универсальных связей, установленных инструкциями или разработчиком. Связи являются бинарными, т.е. переходы осуществляются от одного узла к другому, а в качестве узлов подразумеваются неделимые единицы. Если существуют два различных узла *А* и *В*, и между ними присутствует универсальная бинарная связь, то такая связь является зависимостью. В совокупности, множество таких зависимостей должно покрывать все возможные связи между значениями.

*Базовый шаблон* – установленное правило, определяющее зависимость в анализируемых ячейках таблицы. Конструкция шаблона формируется из последовательности неделимых единиц, слов или знаков. Характеризуется уникальным идентификатором. Для каждого базового шаблона определяется набор альтернативных. *Альтернативный шаблон* (формула 1.1) – установленное правило, противоречащее базовому, которое позволяет определять причину нарушения зависимости. Отличительной особенностью является наличие производимой операции при обнаружении в ячейке таблицы. Кроме того, данные шаблоны обладают приоритетом.

*Приоритет* – формальное свойство, влияющее на очередность расположения конструкций в наборе. При одинаковом приоритете шаблоны располагаются исходя из алфавитной сортировки по идентификатору или усмотрению разработчика. Установка очередности альтернативных конструкций зависит от значимости причины нарушения зависимости.

*Операция* – действие, выполняемое при соответствии конструкции ячейки таблицы одному из шаблонов. Как правило, этим действием является запись ошибки в определенную структуру, с помощью которой потом формируется отчет об обработке тестового артефакта. Если отличия базового шаблона и альтернативного незначительно, то действием возможно автоматическое исправление ячейки.

Обработка ячеек таблицы в анализируемом тестовом артефакте строится на основе поиска схожей конструкции в множестве шаблонов. Пусть существует набор значений, множество   
базовых шаблонов. Тогда для каждого отдельного базового шаблона определено множество альтернативных шаблонов.

Все значения из набора *P* последовательно сопоставляются с выделенными отдельными базовыми шаблонами. При соответствии конструкций производится переход к следующей ячейке. В противном случае происходит последовательный поиск альтернативного шаблона из соответствующего множества . После обнаружения совпадения с альтернативной конструкцией происходит выполнение операции, закрепленной для данного шаблона.

Таких совпадений может быть несколько, поэтому сопоставления происходит до тех пор, пока не будут проверены все шаблоны. Для окончания работы всего алгоритма необходимо обработать последовательно все ячейки таблицы.

## Алгоритмы поиска информации

В тестовых артефактах содержится огромное количество значений, с которыми необходимо работать, поэтому поиск подстроки в строке является одной из важнейших задач. Примитивный алгоритм основан на переборе всех подстрок, длина которых равна длине шаблона поиска, и посимвольном сравнении таких подстрок с шаблонами поиска. Данный способ не является быстродействующим и в лучшем случае будет произведено *n-m+1* сравнений, где *n* - длинна строки, *m* – длинна необходимой подстроки, а в худшем и вовсе снижается скорость до *O(n\*m)*, что не подходит для решения поставленной задачи.

Алгоритмов существует огромное разнообразие, поэтому необходимо отталкиваться исходя от данных, на которых будет производится поиск. После изучения таблиц данных были выявлены следующие особенности:

* Огромное количество строк, отличающихся одним или несколькими символами (плохая выборка данных);
* Размер каждой строки не превышает длину алфавита.

Для рассмотрения, на основе выявленных особенностей таблиц данных, были выбраны алгоритм Кнута-Морриса-Пратта ***[\*]***, алгоритм Бойера-Мура ***[\*]*** и двоичный алгоритм поиска подстроки ***[\*]***.

### Алгоритм Кнута-Морриса-Пратта

Идея алгоритма строится на использовании префикс-функции. Для более понятного объяснения введем определения префикса и суффикса. *Префикс­* – подстрока, начинающаяся с начала строки. *Суффикс* – подстрока, заканчивающаяся в конце строки. Тогда *префикс-функцией* ***[\*]*** является функцией, которая для каждого элемента строки показывает длину наибольшего префикса строки, исключая вырожденные случаи, где префикс совпадает с этой строкой и одновременно является ее суффиксом.

Рассмотрим сам алгоритм Кнута-Морриса-Пратта ***[\*]***. Как и в примитивном алгоритме поиска подстроки в строке, шаблон сдвигается по строке слева направо с целью обнаружения совпадения. Однако ключевым отличием является то, что при помощи вычисленной префикс-функции избегаются бесполезные сдвиги. Пусть нам нужно найти подстроку *t* в строке *S*. Тогда необходимо найти префикс-функцию от строки , где # - символ, гарантировано не встречающийся ни в одной из строк. Если эта префикс-функция содержит значения равные длине *t*, значит *t* входит в *S*, а – последний символ вхождения *t* в *S*.

Время работы алгоритма линейно зависит от объема входных данных, поэтому разработать асимптотически эффективный алгоритм невохможно.

### Алгоритм Бойера-Мура

Алгоритм, разработанный двумя учеными, считается наиболее быстрым среди алгоритмов общего назначения. Отличительной чертой считается предобработка искомой строки, благодаря которой сравнение производится не во всех позициях.

Сравнение шаблона *x* справа налево, начиная с самого правого, один за другим с символами исходной строки *y.* При последовательном совпадении всех символов шаблона с наложенными символами строки, подстрока считается найденной, и поиск заканчивается. В случае несовпадения какого-либо символа он использует две предварительно вычисляемых эвристических функций, чтобы сдвинуть позицию для начала сравнения вправо. Таким образом алгоритм Бойера-Мура ***[\*]*** для сдвига позиции выбирает между двумя функциями, называемыми эвристиками хорошего суффикса и плохого символа.

При совпадении в шаблоне одного или больше символов с текстом, шаблон сдвигается по правилу хорошего суффикса в зависимости от того, какой суффикс совпал. Различают два варианта сдвига хорошего суффикса.

Если существуют такие подстроки равные u, что они полностью входят в x и идут справа от символов, отличных от , то сдвиг происходит к самой правой из них, отличной от *u*. Понятно, что таким образом мы не пропустим никакую строку, так как сдвиг происходит на следующую слева подстроку *u* от суффикса. После выравнивания шаблона по этой подстроке сравнение шаблона опять начнется с его последнего символа. На новом шаге алгоритма можно строку *u*, по которой был произведён cдвиг, не сравнивать с текстом.

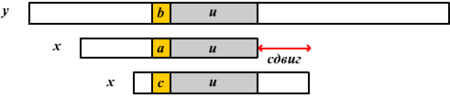


Рис. 1.7 – Сдвиг хорошего суффикса. Встречается вся подстрока.

Если не существует таких подстрок, то смещение состоит в выравнивании самого длинного суффикса *u* подстроки   
с соответствующим префиксом *x*. Поскольку такой подстроки нет, то единственным вариантом остается попадание в эту подстроку префикса.

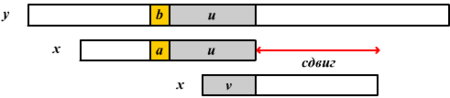


Рис. 1.8 – Сдвиг хорошего суффикса. Суффикс встречается повторно.

В таблице плохих символов указывается последняя позиция в шаблоне (исключая последнюю букву) каждого из символов алфавита. Для всех символов, не вошедших в шаблон, пишем *m*. Предположим, что у нас не совпал символ *c* из текста на очередном шаге с символом из шаблона. Очевидно, что в таком случае мы можем сдвинуть шаблон до первого вхождения этого символа *c* в шаблоне, потому что совпадений других символов точно не может быть. Если в шаблоне такого символа нет, то можно сдвинуть весь шаблон полностью.

Если символ исходного текста встречается в шаблоне *x*, то происходит его выравнивание с его самым правым появлением в подстроке

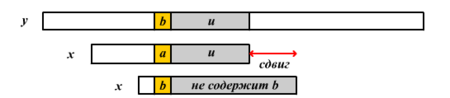


Рис. 1.9 – Сдвиг плохого символа. Символ «а» входит в «x».

Если не встречается в шаблоне x, то ни одно вхождение x в y не может включать в себя и левый конец окна сравнения совмещен с символом, непосредственно идущим после , то есть символ

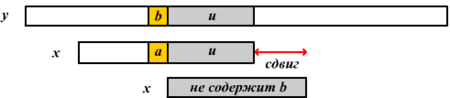


Рис. 1.10 – Сдвиг плохого символа. Символ «b» не входит в «x».

Необходимо учитывать, что сдвиг плохого символа может быть отрицательным, поэтому исходя из ранее приведенных свойств этих функций берется значение равное максимуму между сдвигом хорошего суффикса и сдвигом плохого символа.

### Двоичный алгоритм поиска подстроки

Основная идея алгоритма строится на понимании того, что в современных компьютерах битовый сдвиг и побитовое ИЛИ - атомарные операции. Используя данные операции, можно произвести оптимизацию примитивного алгоритма, благодаря которой за одну операцию производится до 32(64) сравнений одновременно, в зависимости от разрядности машины.

Рассмотрим подробнее работу алгоритма ***[\*]*** на примере. Пусть *subStr –* искомая строка, а *str –* строка, в которой осуществляется поиск. Тогда *n* и *N* – длины строк соответственно. Построим матрицу *M* размером и заполним ее по следующему принципу: элементтогда и только тогда, когда первые *i* символов *subStr* точно совпадают с *i* символами *str*, заканчивая на позиции *j.* В других случаях элемент равняется нулю. В таком случае, искомая строка считается найденной, если в последней строке матрицы существует хотя бы одна единица. Пример заполнения показан в таблице 1.1.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Str  subStr | T | E | M | P | \_ | N | P | T | E | T |
| T | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| E | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| M | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Таблица 1.1 – Пример работы алгоритма двоичного поиска

Для построения матрицы необходимо ввести понятие двоичного вектора. *Двоичный вектор* – строка, длина которой равна длине искомой подстроки, и в которой стоит единица в тех позициях, где в подстроке стоит буква *X.* Тогда столбцы формируются на основании предыдущего столбца по формуле 1.2:

Где *M[j] – j-*ыйстолбец матрицы *M, j –* индекс столбца*, BitShift() –* функция побитового смещения*, BinVector() –* функция формирования двоичного вектора*, str[j] – j-*ый символ строки, в которой производится поиск*.*

Данный алгоритм является очень гибким и при небольшой модификации приспосабливается под приблизительный поиск. Эффективен с подстроками, длина которых не превышает длину машинного слова.

### Выводы

В аналитическом разделе был проведен анализ предметной области, в ходе которого была сформулирована задача обработки тестовых артефактов, и предложен вариант метода обработки тестовых артефактов с использованием шаблонных конструкций. Также были проанализированы методы поиска и получения информации из тестовых артефактов.

# **Приложение А** Специальная электронная форма для представления информационного взаимодействия по внешнему интерфейсу.

ЭЛЕКТРОННАЯ КОПИЯ

СТО 730.0162-2012

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица «обозначение приложения».«номер таблицы» - «название таблицы» | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |
| **Источник** |  |  |  |  |  |  |  |  |  |
| **Режим** |  |  |  |  |  |  |  |  |  |
| **Приемник** |  |  |  |  |  |  |  |  |  |
| **Линия** |  |  |  |  |  |  |  |  |  |
| **№ слова** | **Наименование параметра** | **Обозначение сигнала** | **Размерность** | **Мин. значение** | **Макс. значение** | **Цена старшего разряда** | **Цена младшего разряда** | **Используемые разряды** | **Примечание** |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |