|  |  |  |  |
| --- | --- | --- | --- |
|  | | для прик эмбл |  |
| МИНОБРНАУКИ РОССИИ | | | |
| Федеральное государственное бюджетное образовательное учреждениевысшего образования«Московский технический университет»МИРЭА | | | |
| ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ | | | |
| КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ | | | |

|  |  |
| --- | --- |
| **Курсовая работа** | |
| по дисциплине | |
| **«**Программирование**»** | |
| на тему **«Реализация универсальной машины Тьюринга»** | |
| Студенты группы ИКБО-05-16 | Кадров И.В.  Глазов В.Е.  Гуськов П.С.  Бойко С.В. |
| Научный руководитель  доцент кафедры вычислительной  тех­ники  Институт информационных  техноло­гий МИРЭА  к.ф.-м.н., доцент | Горшенин А.К. |
|  |  |

|  |  |
| --- | --- |
| Работа представлена к защите | «\_\_\_»\_\_\_\_\_\_201\_ г. |
|  |  |

Москва 2017

Содержание

Постановка задачи ………………………………………………….……….….3

Введение ………………………………………………………………………….3

1. Описание машины Тьюринга…………………………………………………4

1.1 Свойства машины Тьюринга как алгоритма………………………………..6

2. Сложность алгоритмов………………………………………………………...7

2.1 Сложность проблем………………………………………………………….10

3. Машина Тьюринга и алгоритмически неразрешимые проблемы………….12

4. Структура……………………………………………………………………....16

5. Практическая часть…………………………………………………………….17

6. Примеры реализации……………………………………………………….….18

Заключение………………………………………………………………………..20

Список использованной литературы……………………………………………21

Приложения……………………………………………………………………….22

**Постановка задачи**

Тьюринг показал на примерах, что существуют задачи, для которых применение формальных методов требует бесконечно много времени для решения, т.е. решения практически нет. В то время как человек решает эти задачи очень просто. Это имеет важное практическое значение  для разработчиков систем искусственного интеллекта.  Машина Тьюринга играет в понимании математики, в умении квалифицированно решать задачу такую же фундаментальную роль, как и постулаты Евклида. Целью курсовой работы было дать описание реализации универсальной машины Тьюринга и выполнить соответствующую практическую часть работы.

**Введение**

Машина Тьюринга – это очень простое вычислительное устройство. Она состоит из ленты бесконечной длины, разделенной на ячейки, и головки, которая перемещается вдоль ленты и способна читать и записывать символы. Также у машины Тьюринга есть такая характеристика, как состояние, которое может выражаться целым числом от нуля до некоторой максимальной величины. В зависимости от состояния машина Тьюринга может выполнить одно из трех действий: записать символ в ячейку, передвинуться на одну ячейку вправо или влево и установить внутреннее состояние.

Устройство машины Тьюринга чрезвычайно просто, однако на ней можно выполнить практически любую программу. Для выполнения всех этих действий предусмотрена специальная таблица правил, в которой прописано, что нужно делать при различных комбинациях текущих состояний и символов, прочитанных с ленты.

Развитие теории алгоритмов начинается с доказательства К. Геделем теорем о неполноте формальных систем, включающих арифметику, первая из которых была доказана в 1931 г. Возникшее в связи с этими теоремами предположение о невозможности алгоритмического разрешения многих математических проблем вызвало необходимость стандартизации понятия алгоритма. Первые стандартизованные варианты этого понятия были разработаны в 30-х годах XX века в работах А. Тьюринга, А. Черча и Э. Поста. Предложенные ими машина Тьюринга, машина Поста и лямбда-исчисление Черча оказались эквивалентными друг другу. Основываясь на работах Геделя, С. Клини ввел понятие рекурсивной функции, также оказавшееся эквивалентным вышеперечисленным.

Алан Тьюринг (1912–1954) – английский математик. Он дал определение алго­ритма через построение, названное машиной Тьюринга. В 1936 году Тьюринг предложил определение вычислимости, которое основано на анализе осуществ­ления алгоритма человеком, располагающим ручкой для письма и бумагой. Он рассматривает это как последовательность очень простых действий следую­щего вида:  
 В 1937 г. Алан Тьюринг расширил определение, описав "универсальную машину Тьюринга". Позже для решения определенных классов задач была вве­дена ее разновидность, которая позволяла выполнять не одну задачу, а не­сколько.

# 1. Описание машины Тьюринга

Алан Тьюринг в 1936 году опубликовал в трудах Лондонского математического общества статью "О вычислимых числах в приложении к проблеме разрешения", которая наравне с работами Поста и Черча лежит в основе современной теории алгоритмов.

Предыстория создания этой работы связана с формулировкой Давидом Гильбертом на Международном математическом конгрессе в Париже в 1900 году неразрешенных математических проблем. Одной из них была задача доказательства непротиворечивости системы аксиом обычной арифметики, которую Гильберт в дальнейшем уточнил как "проблему разрешимости" – нахождение общего метода, для определения выполнимости данного высказывания на языке формальной логики.

Статья Тьюринга как раз и давала ответ на эту проблему – вторая проблема Гильберта оказалась неразрешимой. Но значение статьи Тьюринга выходило далеко за рамки той задачи, по поводу которой она была написана.

Приведем характеристику этой работы, принадлежащую Джону Хопкрофту: "Работая над проблемой Гильберта, Тьюрингу пришлось дать четкое определение самого понятия метода. Отталкиваясь от интуитивного представления о методе как о некоем алгоритме, т.е. процедуре, которая может быть выполнена механически, без творческого вмешательства, он показал, как эту идею можно воплотить в виде подробной модели вычислительного процесса. Полученная модель вычислений, в которой каждый алгоритм разбивался на последовательность простых, элементарных шагов, и была логической конструкцией, названной впоследствии машиной Тьюринга".

Машина Тьюринга является расширением модели конечного автомата, расширением, включающим потенциально бесконечную память с возможностью перехода (движения) от обозреваемой в данный момент ячейки к ее левому или правому соседу.

Формально машина Тьюринга может быть описана следующим образом. Пусть заданы:

конечное множество состояний – Q, в которых может находиться машина Тьюринга;

конечное множество символов ленты – Г;

функция δ (функция переходов или программа), которая задается отображением пары из декартова произведения Q x Г (машина находится в состоянии qi и обозревает символ γi) в тройку декартова произведения Q х Г х {L,R} (машина переходит в состояние qi, заменяет символ γi на символ γj и передвигается влево или вправо на один символ ленты) – Q x Г-->Q х Г х {L,R}; один символ из Г-->е (пустой); подмножество Σ є Г – -> определяется как подмножество входных символов ленты, причем е є (Г – Σ); одно из состояний – q0 є Q является начальным состоянием машины.

Решаемая проблема задается путем записи конечного количества символов из множества Σ є Г – Si є Σ на ленту: eS1S2S3S4......... Sne.

после чего машина переводится в начальное состояние, и головка устанавливается у самого левого непустого символа (q0,↑w) –, после чего в соответствии с указанной функцией переходов (qi,Si) – ->(qj,Sk, L или R), машина начинает заменять обозреваемые символы, передвигать головку вправо или влево и переходить в другие состояния, предписанные функций переходов.

Остановка машины происходит в том случае, если для пары (qi,Si), функция перехода не определена.

Алан Тьюринг высказал предположение, что любой алгоритм в интуитивном смысле этого слова может быть представлен эквивалентной машиной Тьюринга. Это предположение известно, как тезис Черча-Тьюринга. Каждый компьютер может моделировать машину Тьюринга (операции перезаписи ячеек, сравнения и перехода к другой соседней ячейке с учетом изменения состояния машины). Следовательно, он может моделировать алгоритмы в любом формализме, и из этого тезиса следует, что все компьютеры (независимо от мощности, архитектуры и т.д.) эквивалентны с точки зрения принципиальной возможности решения алгоритмических задач.

## 1.1 Свойства машины Тьюринга как алгоритма

На примере машины Тьюринга хорошо прослеживаются свойства алгоритмов. Попросите учащихся показать, что машина Тьюринга обладает всеми свойствами алгоритма.

**Дискретность**. Машина Тьюринга может перейти к (к + 1) – му шагу только после выполнения каждого шага, т.к. именно каждый шаг определяет, каким будет (к + 1) – й шаг.

**Понятность**. На каждом шаге в ячейку пишется символ из алфавита, автомат делает одно движение (Л, П, Н), и машина Тьюринга переходит в одно из описанных состояний.

**Детерминированность**. В каждой клетке таблицы машины Тьюринга записан лишь один вариант действия. На каждом шаге результат определен однозначно, следовательно, последовательность шагов решения задачи определена однозначно, т.е. если машине Тьюринга на вход подают одно и то же входное слово, то выходное слово каждый раз будет одним и тем же.

**Результативность**. Содержательно результаты каждого шага и всей последовательности шагов определены однозначно, следовательно, правильно написанная машина Тьюринга за конечное число шагов перейдет в состояние q0, т.е. за конечное число шагов будет получен ответ на вопрос задачи.

**Массовость**. Каждая машина Тьюринга определена над всеми допустимыми словами из алфавита, в этом и состоит свойство массовости. Каждая машина Тьюринга предназначена для решения одного класса задач, т.е. для каждой задачи пишется своя (новая) машина Тьюринга.

# 2. Сложность алгоритмов

Что же мы знаем об алгоритмах? Алгоритм – понятная и точная  последовательность действий, описывающая процесс преобразования объекта из начального состояния в конечное. Исполнителем алгоритма может быть как человек (кулинарные рецепты, различные инструкции, алгоритмы математических вычислений), так и техническое устройство. Различные машины (компьютеры, промышленные роботы, современная бытовая техника) являются формальными исполнителями алгоритмов. От формального исполнителя не требуется понимание сущности решаемой задачи, но требуется точное выполнение последовательности команд.

Алгоритм можно записывать различными способами (словесное описание, графическое описание – блок схема, программа на одном из языков программирования и т.д.). Программа – это алгоритм, записанный на языке программирования.

Сама же сложность алгоритма определяется вычислительными мощностями, необходимыми для его выполнения. Вычислительная сложность алгоритма часто измеряется двумя параметрами: Т (временная сложность) и S (пространственная сложность, или требования к памяти). И Т, и S обычно представляются в виде функций от n, где n – это размер входных данных. (Существую и другие способы измерения сложности: количество случайных бит, ширина канала связи, объем данных и т.п.).

Обычно вычислительная сложность алгоритма выражается с помощью нотации "О большого", т. е описывается порядком величины вычислительной сложности. Это просто член разложения функции сложности, быстрее всего растущий с ростом n, все члены низшего порядка игнорируются. Например, если временная сложность данного алгоритма равна 4n2+7n+12, то вычислительная сложность порядка n2, записываемая как О(n2).

Временная сложность, измеренная таким образом не зависит от реализации. Не нужно знать ни точное время выполнения различных инструкций, ни число битов, используемых для представления различных переменных, ни даже скорость процессора. Один компьютер может быть на 50 процентов быстрее другого, а у третьего шина данных может быть в два раза шире, но сложность алгоритма, оцененная по прядку величины, не изменится. Это не жульничество, при работе с алгоритмами настолько сложными, как описанные в этой книге, всем прочим можно пренебречь (с точностью до постоянного множителя) в сравнении со сложностью по порядку величины.

Эта нотация позволяет увидеть, как объем входных данных влияет на требования к времени и объему памяти. Например, если Т= О(n), то удвоение входных данных удвоит и время выполнения алгоритма. Если Т=О(2n), то добавление одного бита к входным данным удвоит время выполнения алгоритма.

Обычно алгоритмы классифицируются в соответствии с их временной или пространственной сложностью. Алгоритм называют постоянным, если его сложность не зависит от n: 0(1). Алгоритм является линейным, если его временная сложность О(n). Алгоритмы могут быть квадратичными, кубическими и т.д. Все эти алгоритмы – полиноминальны, их сложность – О(m), где m – константа. Алгоритмы с полиномиальной временной сложностью называются алгоритмами с полиномиальным временем

Алгоритмы, сложность которых равна О(tf(n)), где t – константа, большая, чем 1, a f(n) – некоторая полиномиальная функция от n, называются экспоненциальными. Подмножество экспоненциальных алгоритмов, сложность которых равна О(сf(n)), где с – константа, a f(n) возрастает быстрее, чем постоянная, но медленнее, чем линейная функция, называется суперполиноминальным.

В идеале, криптограф хотел бы утверждать, что алгоритм, лучший для взлома спроектированного алгоритма шифрования, обладает экспоненциальной временной сложностью. На практике, самые сильные утверждения, которые могут быть сделаны при текущем состоянии теории вычислительной сложности, имеют форму "все известные алгоритмы вскрытия данной криптосистемы обладают суперполиномиальной временной сложностью". То есть, известные нам алгоритмы вскрытия обладают суперполиномиальной временной сложностью, но пока невозможно доказать, что не может быть открыт алгоритм вскрытия с полиномиальной временной сложностью. Развитие теории вычислительной сложности, возможно, когда-нибудь позволит создать алгоритмы, для которых существование алгоритмов с полиномиальным временем вскрытия может быть исключено с математической точностью.

С ростом n временная сложность алгоритмов может стать настолько огромной, что это повлияет на практическую реализуемость алгоритма.

При условии, что единицей времени для нашего компьютера является микросекунда, компьютер может выполнить постоянный алгоритм за микросекунду, линейный – за секунду, а квадратичный – за 11.6 дня. Выполнение кубического алгоритма потребует 32 тысяч лет, что в принципе реализуемо, компьютер, конструкция которого позволила бы ему противостоять следующему ледниковому периоду, в конце концов, получил бы решение. Выполнение экспоненциального алгоритма тщетно, независимо от экстраполяции роста мощи компьютеров, параллельной обработки или контактов с инопланетным супер разумом.

Взглянем на проблему вскрытия алгоритма шифрования грубой силой. Временная сложность такого вскрытия пропорциональна количеству возможных ключей, которое экспоненциально зависит от длины ключа. Если n – длина ключа, то сложность вскрытия грубой силой равна 0(2n). Сложность вскрытия грубой силой при 56-битовом ключе составляет 256, а при 112-битовом ключе – 2112. В первом случае вскрытие возможно, а во втором – нет.

## 2.1 Сложность проблем

Теория сложности также классифицирует и сложность самих проблем, а не только сложность конкретных алгоритмов решения проблемы. Теория рассматривает минимальное время и объем памяти, необходимые для решения самого трудного варианта проблемы на теоретическом компьютере, известном как машина Тьюринга. Машина Тьюринга представляет собой конечный автомат с бесконечной лентой памяти для чтения записи и является реалистичной моделью вычислений.

Проблемы, которые можно решить с помощью алгоритмов с полиномиальным временем, называются решаемыми, потому что для разумных входных данных обычно могут быть решены за разумное время. (Точное определение "разумности" зависит от конкретных обстоятельств) Проблемы, которые невозможно решить за полиномиальное время, называются нерешаемыми, потому что вычисление их решений быстро становится невозможным. Нерешаемые проблемы иногда называют трудными. Проблемы, которые могут быть решены только с помощью суперполиномиальных алгоритмов, вычислительно нерешаемы, даже при относительно малых значениях n.

Что еще хуже, Алан Тьюринг доказал, что некоторые проблемы принципиально неразрешимы. Даже отвлекаясь от временной сложности алгоритма, невозможно создать алгоритм решения этих проблем.

Задачи можно разбить на классы в соответствии со сложностью их решения. Вот важнейшие из них и предполагаемые соотношения между ними: P<=NP<=EXPTIME

Находящийся слева класс P включает все задачи, которые можно решить за полиномиальное время. В класс NP входят все задачи, которые можно решить за полиномиальное время только на недетерминированной машине Тьюринга (это вариант обычной машины Тьюринга, которая может делать предположения). Такая машина предполагает решение задачи – либо "удачно угадывая", либо перебирая все предположения параллельно – и проверяет свое предположение за полиномиальное время.

Класс NP включает в себя класс P, поскольку любую задачу, решаемую за полиномиальное время на детерминированной (обычной) машине Тьюринга, можно решить и на недетерминированной за полиномиальное время, просто этап предположения опускается.

Если все задачи класса NP решаются за полиномиальное время и на детерминированной машине, то P=NP. Тем не менее, никем не доказано, что P<>NP (или P=NP). Однако, большинство специалистов, занимающихся теорией сложности, убеждены, что это классы неравны.

Как ни странно, можно доказать, что некоторые NP-задачи настолько же трудны, что и любая задача этого класса. Такие задачи называются NP-полными. То есть, если такая задача решается за полиномиальное время, то P=NP.

Таким образом, для программиста NP-полнота означает полный перебор, причем сложность этого перебора будет экспоненциальной или факториальной. Но следует понимать, что не всякий полный перебор имеет такую сложность. Например, если решать задачи из предыдущего выпуска полным перебором, то сложность полученных алгоритмов будет полиномиальной – O(n2) для задачи про подпоследовательности и O(n6) для задачи про подматрицы.

Наконец, существует класс задач EXPTIME. Эти задачи решаются за экспоненциальное время. В настоящее время можно доказать, что EXPTIME-полные задачи невозможно решить за детерминированное полиномиальное время. Кроме того, доказано, что P<>EXPTIME.

# 3. Машина Тьюринга и алгоритмически неразрешимые проблемы

За время своего существования человечество придумало множество алгоритмов для решения разнообразных практических и научных проблем. Зададимся вопросом – а существуют ли какие-нибудь проблемы, для которых невозможно придумать алгоритмы их решения?

Утверждение о существовании алгоритмически неразрешимых проблем является весьма сильным – мы констатируем, что мы не только сейчас не знаем соответствующего алгоритма, но мы не можем принципиально никогда его найти.

Успехи математики к концу XIX века привели к формированию мнения, которое выразил Д. Гильберт – "в математике не может быть неразрешимых проблем", в связи с этим формулировка проблем Гильбертом на конгрессе 1900 года в Париже была руководством к действию, констатацией отсутствия решений в данный момент.

Первой фундаментальной теоретической работой, связанной с доказательством алгоритмической неразрешимости, была работа Курта Геделя – его известная теорема о неполноте символических логик. Это была строго формулированная математическая проблема, для которой не существует решающего ее алгоритма. Усилиями различных исследователей список алгоритмически неразрешимых проблем был значительно расширен. Сегодня принято при доказательстве алгоритмической неразрешимости некоторой задачи сводить ее к ставшей классической задаче – "задаче останова".

Имеет место быть следующая теорема: не существует алгоритма (машины Тьюринга), позволяющего по описанию произвольного алгоритма и его исходных данных (и алгоритм и данные заданы символами на ленте машины Тьюринга) определить, останавливается ли этот алгоритм на этих данных или работает бесконечно.

Таким образом, фундаментально алгоритмическая неразрешимость связана с бесконечностью выполняемых алгоритмом действий, т.е. невозможностью предсказать, что для любых исходных данных решение будет получено за конечное количество шагов.

Тем не менее, можно попытаться сформулировать причины, ведущие к алгоритмической неразрешимости, эти причины достаточно условны, так как все они сводимы к проблеме останова, однако такой подход позволяет более глубоко понять природу алгоритмической неразрешимости:

а) Отсутствие общего метода решения задачи.

Проблема 1: Распределение девяток в записи числа.

Определим функцию f(n) = i, где n – количество девяток подряд в десятичной записи числа, а i – номер самой левой девятки из n девяток подряд: =3,141592… f(1) = 5.

Задача состоит в вычислении функции f(n) для произвольно заданного n.

Поскольку число является иррациональным и трансцендентным, то мы не знаем никакой информации о распределении девяток (равно как и любых других цифр) в десятичной записи числа. Вычисление f(n) связано с вычислением последующих цифр в разложении, до тех пор, пока мы не обнаружим n девяток подряд, однако у нас нет общего метода вычисления f(n), поэтому для некоторых n вычисления могут продолжаться бесконечно – мы даже не знаем в принципе (по природе числа) существует ли решение для всех n.

Проблема 2: Вычисление совершенных чисел.

Совершенные числа – это числа, которые равны сумме своих делителей, например: 28 = 1+2+4+7+14.

Определим функцию S(n) = n-ое по счёту совершенное число и поставим задачу вычисления S(n) по произвольно заданному n. Нет общего метода вычисления совершенных чисел, мы даже не знаем, множество совершенных чисел конечно или счетно, поэтому наш алгоритм должен перебирать все числа подряд, проверяя их на совершенность. Отсутствие общего метода решения не позволяет ответить на вопрос о останове алгоритма. Если мы проверили М чисел при поиске n-ого совершенного числа – означает ли это, что его вообще не существует?

Проблема 3: Десятая проблема Гильберта.

Пусть задан многочлен n-ой степени с целыми коэффициентами – P, существует ли алгоритм, который определяет, имеет ли уравнение P=0 решение в целых числах?

Ю.В. Матиясевич показал, что такого алгоритма не существует, т.е. отсутствует общий метод определения целых корней уравнения P=0 по его целочисленным коэффициентам.

б) Информационная неопределенность задачи.

Проблема 4: Позиционирование машины Поста на последний помеченный ящик.

Пусть на ленте машины Поста заданы наборы помеченных ящиков (кортежи) произвольной длины с произвольными расстояниями между кортежами и головка находится у самого левого помеченного ящика. Задача состоит установке головки на самый правый помеченный ящик последнего кортежа.

Попытка построения алгоритма, решающего эту задачу приводит к необходимости ответа на вопрос – когда после обнаружения конца кортежа мы сдвинулись вправо по пустым ящикам на М позиций и не обнаружили начало следующего кортежа – больше на ленте кортежей нет или они есть где-то правее? Информационная неопределенность задачи состоит в отсутствии информации либо о количестве кортежей на ленте, либо о максимальном расстоянии между кортежами – при наличии такой информации (при разрешении информационной неопределенности) задача становится алгоритмически разрешимой.

в) Логическая неразрешимость (в смысле теоремы Геделя о неполноте).

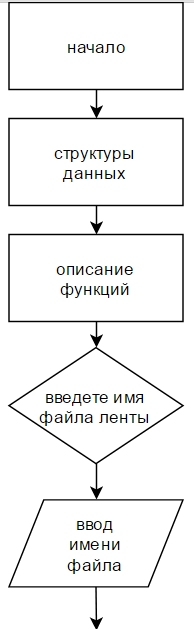
Проблема 5: Проблема "останова" (см. теорема).

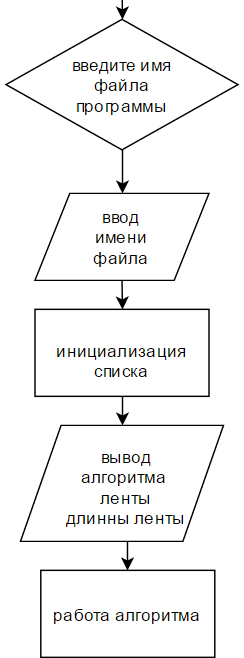
Проблема 6: Проблема эквивалентности алгоритмов.

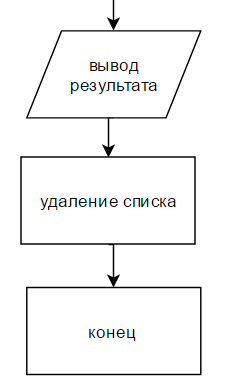
По двум произвольным заданным алгоритмам (например, по двум машинам Тьюринга) определить, будут ли они выдавать одинаковые выходные результаты на любых исходных данных.

Проблема 7: Проблема тотальности.

По произвольному заданному алгоритму определить, будет ли он останавливаться на всех возможных наборах исходных данных. Другая формулировка этой задачи – является ли частичный алгоритм Р всюду определённым?

**Структура программы «Универсальная машина Тьюринга»**

****

****

**Реализация программы:**

функции алгоритма выполнения машины Тьюринга

****

псевдокод алгоритма работы с файлами

****

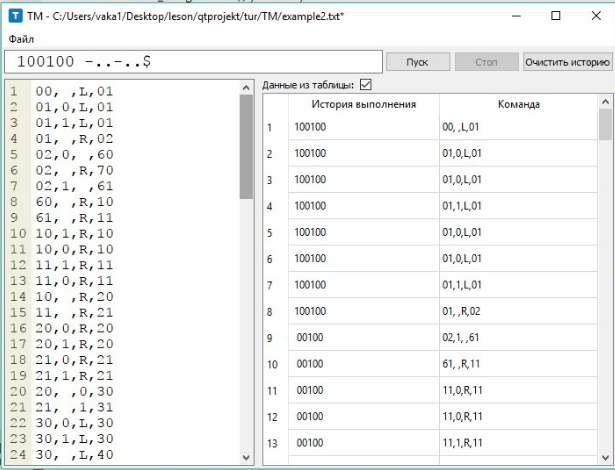
Элементы некоторых виджетов в интерфейсе

****

Примеры сигналов для взаимодействия пользователя с интерфейсом

****

Пример работы программы через интерфейс

****

**Заключение**

Теория сложности также классифицирует и сложность самих проблем, а не только сложность конкретных алгоритмов решения проблемы. Теория рассматривает минимальное время и объем памяти, необходимые для решения самого трудного варианта проблемы на теоретическом компьютере, известном как машина Тьюринга. Машина Тьюринга представляет собой конечный автомат с бесконечной лентой памяти для чтения записи и является реалистичной моделью вычислений.

Задачи можно разбить на классы в соответствии со сложностью их решения. Вот важнейшие из них и предполагаемые соотношения между ними: P<=NP<=EXPTIME

Находящийся слева класс P включает все задачи, которые можно решить за полиномиальное время. В класс NP входят все задачи, которые можно решить за полиномиальное время только на недетерминированной машине Тьюринга (это вариант обычной машины Тьюринга, которая может делать предположения). Такая машина предполагает решение задачи – либо "удачно угадывая", либо перебирая все предположения параллельно – и проверяет свое предположение за полиномиальное время.

Класс NP включает в себя класс P, поскольку любую задачу, решаемую за полиномиальное время на детерминированной (обычной) машине Тьюринга, можно решить и на недетерминированной за полиномиальное время, просто этап предположения опускается.

Если все задачи класса NP решаются за полиномиальное время и на детерминированной машине, то P=NP. Тем не менее, никем не доказано, что P<>NP (или P=NP). Однако, большинство специалистов, занимающихся теорией сложности, убеждены, что это классы неравны.

**Список использованной литературы**

1. Гуц А.К. Математическая логика и теория алгоритмов. – Омск: Издательство Наследие. Диалог-Сибирь, 2003. – 108 с.
2. Игошин В.И. Задачи и упражнения по математической логике и теории алгоритмов / В.И. Игошин. – 3-е изд., стер. – М.: Издательский центр "Академия", 2007. – 304 с.
3. Лавров И.А. Математическая логика: учеб. пособие для студ. высш. учеб. заведений / И.А. Лавров; под ред. Л.Л. Максимовой. – М.: Издательский центр "Академия", 2006. – 240 с.
4. Михайлов А.Б., Рыжова Н.И., Швецкий М.В. Упражнения по основам математической логики. Формальные системы первого порядка. Учебное пособие для студентов математического факультета – Санкт-Петербург: РГПУ. 1997. – 127 с.
5. Рощин А.Г., Половов Р.М. Теория автоматов. Часть I. Тексты лекций – Москва: МГТУ ГА, 2001. – 76 с.
6. Фалина Н.М. Машина Тьюринга // Информатика. – №26. – 2005. – с. 12-15.
7. Фалевич Б.Я. Теория алгоритмов. – М.: ИНФРА-М, 2006. – с. 324.

**Приложения**

QT += core gui

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

TARGET = Sturing3

TEMPLATE = app

SOURCES += main.cpp \

controller.cpp \

ui.cpp \

sturing.cpp \

codeeditor.cpp \

history.cpp \

historyitem.cpp \

filecontroller.cpp

HEADERS += \

controller.h \

ui.h \

sturing.h \

codeeditor.h \

history.h \

historyitem.h \

filecontroller.h

CONFIG += c++11

RESOURCES += \

resources.qrc

#ifndef CODEEDITOR\_H

#define CODEEDITOR\_H

#include <QPlainTextEdit>

#include <QObject>

QT\_BEGIN\_NAMESPACE

class QPaintEvent;

class QResizeEvent;

class QSize;

class QWidget;

QT\_END\_NAMESPACE

class LineNumberArea;

class CodeEditor : public QPlainTextEdit

{

Q\_OBJECT

public:

CodeEditor(QWidget \*parent = 0);

void lineNumberAreaPaintEvent(QPaintEvent \*event);

int lineNumberAreaWidth();

protected:

void *resizeEvent*(QResizeEvent \*event) Q\_DECL\_OVERRIDE;

private slots:

void updateLineNumberAreaWidth(int newBlockCount);

void highlightCurrentLine();

void updateLineNumberArea(const QRect &, int);

void rectSend(const QRect &, int);

void highlightLine(int);

private:

QWidget \*lineNumberArea;

};

class LineNumberArea : public QWidget

{

public:

LineNumberArea(CodeEditor \*editor) : QWidget(editor) {

codeEditor = editor;

}

QSize *sizeHint*() const Q\_DECL\_OVERRIDE {

return QSize(codeEditor->lineNumberAreaWidth(), 0);

}

protected:

void *paintEvent*(QPaintEvent \*event) Q\_DECL\_OVERRIDE {

codeEditor->lineNumberAreaPaintEvent(event);

}

private:

CodeEditor \*codeEditor;

};

#endif

#ifndef CONTROLLER\_H

#define CONTROLLER\_H

#include <QObject>

#include <vector>

#include <string>

#include <QtWidgets>

#include "sturing.h"

#include "ui.h"

using namespace std;

class Controller : public QObject

{

Q\_OBJECT

private:

STuring\* turing;

QApplication\* app;

UI\* ui;

bool isTableParametres;

QSettings\* settings;

public:

explicit Controller(QApplication \*app\_, QObject \*parent = 0);

signals:

public slots:

void loadRecentFile(QString path);

void tmRun();

void setRunable(bool);

void tmStop();

void updateFromTable(int, int);

void changeTableParameter(bool);

void addHistory(QString state, QString line, int pointer, QString command);

void tmLineChanged();

void setHistoryEnabled(bool);

void saveSettings();

};

#endif

#ifndef FILECONTROLLER\_H

#define FILECONTROLLER\_H

#include <QFileDialog>

#include <QTextStream>

#include <string>

#include <QDebug>

using namespace std;

class FileController : public QObject

{

Q\_OBJECT

QFile\* file;

QString path;

bool fileOpenned;

QString src, line;

QVector<QString>lines;

public:

explicit FileController(QObject \*parent = 0);

signals:

void opennedFile(QString src, QString line);

void saveFileSign();

void sendPath(QString);

public slots:

QString getPath(QString p);

void clear();

void openDialogEnable();

void saveDialogEnable();

void saveFile(QString src, QString line);

void createNewFile();

QString getPathString();

void openFromPath(QString);

};

#endif

#ifndef HISTORY\_H

#define HISTORY\_H

#include <QtWidgets>

#include "historyitem.h"

class History : public QTableWidget

{

Q\_OBJECT

private:

QVector<HistoryItem>historyItem;

int colWidth;

int numberHistoryItem;

public:

History();

void addItem(QString state, QString line, int pointer);

void addItem(QString state, QString line, int pointer, QString command);

void updateTable();

bool isFirstHave();

void clearPart();

QString getState(int n);

QString getLine(int n);

int getPointer(int n);

int getNumberHistoryItem();

bool isEmpty();

public slots:

void changeNumberHistoryItem(int r, int c);

void clearAllHistory();

};

#endif

#ifndef HISTORYITEM\_H

#define HISTORYITEM\_H

#include <QString>

using namespace std;

class HistoryItem

{

private:

QString state, line, command;

int pointer;

public:

HistoryItem(QString state\_, QString line\_, int pointer\_);

HistoryItem(QString state\_, QString line\_, int pointer, QString command\_);

HistoryItem();

QString getState();

QString getLine();

QString getCommand();

int getPointer();

};

#endif

#ifndef STURING\_H

#define STURING\_H

#include <QObject>

#include <QtWidgets>

using namespace std;

class STuring : public QObject

{

Q\_OBJECT

public:

int pointer;

bool stopped;

QString nowState, line;

QVector<QString>stackSrc;

bool errorsHave;

bool isRunning;

QTime time;

void go();

void executeCommand(QString& cmd);

void changeLetter(QChar let);

void moveRight();

void moveLeft();

void stop();

QString getState(QString& cmd);

QString getReadLetter(QString& cmd);

QString getWriteLetter(QString& cmd);

QString getNextState(QString& cmd);

QString get(QString& cmd, int k);

QString uncomment(QString& str);

bool testOfExecute(QString& cmd);

public:

explicit STuring(QObject \*parent = 0);

signals:

void getData();

void getSource();

void Runable(bool);

void updateLine(QString);

void updatePointer(int);

void getSettings();

void commandExecuted(QString state, QString line, int pointer, QString command);

void testErrors(/\*QVector<QString>, QString\*/);

public slots:

void run(QString src\_);

void maxSpeedEnable(bool);

void setStandardSettings(QString line\_);

void setCustomSettings(QString state\_, QString line\_, int pointer\_);

};

#endif

#ifndef UI\_H

#define UI\_H

#include <QObject>

#include <QtWidgets>

#include <string>

#include <QDebug>

#include "codeeditor.h"

#include "history.h"

#include "filecontroller.h"

using namespace std;

class UI : public QObject

{

Q\_OBJECT

private:

void createActions();

public:

FileController\* fControl;

QString programmNameString;

QIcon icon;

QWidget mainWindow;

QLineEdit\* tmLine;

QPushButton\* tmRunBtn;

QPushButton\* tmStopBtn;

QPushButton\* clearHistoryBtn;

CodeEditor\* tmSrc;

QHBoxLayout\* upLayout;

QVBoxLayout\* turingLayout;

QTime time;

QHBoxLayout\* menuLayout;

QLabel\* infoLbl;

QFont fontLine, fontSrc, fontLbl;

QHBoxLayout\* historyCkbLayout;

QLabel\* historyLbl;

QCheckBox\* historyCkb;

QAction\* saveAction;

QAction\* openAction;

QMenuBar\* menuBar;

QMenu\* fileMenu;

QApplication\* app;

QWidget aboutDialog;

QPixmap logo;

QLabel\* logoLbl;

QLabel\* programName;

QLabel\* aboutLbl;

QVBoxLayout\* dialogMainLayout;

QHBoxLayout\* appLayout;

QVBoxLayout\* rightLayout;

QVBoxLayout\* mainVerticalSeparator;

History\* history;

QString allWindowTitle;

int dialogW = 300, dialogH = 200;

bool fileSaved;

public:

explicit UI(QApplication \*app\_, QObject \*parent = 0);

signals:

void saveSettings();

public slots:

void setSrcSize();

void setPointer(int);

void setLine(QString);

void dialogShow();

void createTableHistory();

void openFile(QString, QString);

void saveFile();

void fileNameWindow(QString str);

void createNewFile();

void addUnSaved();

void addSaved();

};

#endif

#include <QtWidgets>

#include "codeeditor.h"

CodeEditor::CodeEditor(QWidget \*parent) : QPlainTextEdit(parent)

{

lineNumberArea = new LineNumberArea(this);

connect(this, SIGNAL(blockCountChanged(int)), this, SLOT(updateLineNumberAreaWidth(int)));

connect(this, SIGNAL(updateRequest(QRect,int)), this, SLOT(updateLineNumberArea(QRect,int)));

connect(this, SIGNAL(cursorPositionChanged()), this, SLOT(highlightCurrentLine()));

updateLineNumberAreaWidth(0);

highlightCurrentLine();

}

void CodeEditor::rectSend(const QRect& rect, int dy) {

QRect r2;

r2 = rect;

r2.setWidth(rect.width() + 7);

qDebug() << r2.width();

updateLineNumberArea(r2, dy);

}

int CodeEditor::lineNumberAreaWidth()

{

int digits = 1;

int max = qMax(1, blockCount());

while (max >= 10) {

max /= 10;

++digits;

}

int space = 3 + fontMetrics().width(QLatin1Char('9')) \* digits;

return space + 7;

}

void CodeEditor::updateLineNumberAreaWidth(int /\* newBlockCount \*/)

{

setViewportMargins(lineNumberAreaWidth(), 0, 0, 0);

}

void CodeEditor::updateLineNumberArea(const QRect &rect, int dy)

{

if (dy)

lineNumberArea->scroll(0, dy);

else

lineNumberArea->update(0, rect.y(), lineNumberArea->width(), rect.height());

if (rect.contains(viewport()->rect()))

updateLineNumberAreaWidth(0);

}

void CodeEditor::*resizeEvent*(QResizeEvent \*e)

{

QPlainTextEdit::*resizeEvent*(e);

QRect cr = contentsRect();

lineNumberArea->setGeometry(QRect(cr.left(), cr.top(), lineNumberAreaWidth(), cr.height()));

}

void CodeEditor::highlightCurrentLine()

{

QList<QTextEdit::ExtraSelection> extraSelections;

if (!isReadOnly()) {

QTextEdit::ExtraSelection selection;

QColor lineColor = QColor(0, 154, 255, 20);

selection.format.setBackground(lineColor);

selection.format.setProperty(QTextFormat::FullWidthSelection, true);

selection.cursor = textCursor();

selection.cursor.clearSelection();

extraSelections.append(selection);

}

setExtraSelections(extraSelections);

}

void CodeEditor::highlightLine(int line) {

QList<QTextEdit::ExtraSelection> extraSelections;

if (!isReadOnly()) {

QTextEdit::ExtraSelection selection;

QTextCursor cur;

cur.setPosition(5, QTextCursor::KeepAnchor);

QColor lineColor = QColor(0, 154, 255, 20);

selection.format.setBackground(lineColor);

selection.format.setProperty(QTextFormat::FullWidthSelection, true);

selection.cursor.setPosition(line);

qDebug() << selection.cursor.position();

selection.cursor.clearSelection();

extraSelections.append(selection);

}

setExtraSelections(extraSelections);

}

void CodeEditor::lineNumberAreaPaintEvent(QPaintEvent \*event)

{

QPainter painter(lineNumberArea);

painter.fillRect(event->rect(), QColor(255, 230, 0, 20));

QTextBlock block = firstVisibleBlock();

int blockNumber = block.blockNumber();

int top = (int) blockBoundingGeometry(block).translated(contentOffset()).top();

int bottom = top + (int) blockBoundingRect(block).height();

while (block.isValid() && top <= event->rect().bottom()) {

if (block.isVisible() && bottom >= event->rect().top()) {

QString number = QString::number(blockNumber + 1);

painter.setPen(QColor(70, 70, 70, 255));

painter.drawText(5, top, lineNumberArea->width(), fontMetrics().height(),

Qt::AlignLeft, number);

}

block = block.next();

top = bottom;

bottom = top + (int) blockBoundingRect(block).height();

++blockNumber;

}

}

#include "controller.h"

Controller::Controller(QApplication \*app\_, QObject \*parent) : QObject(parent) {

app = app\_;

ui = new UI(app\_);

turing = new STuring();

isTableParametres = false;

settings = new QSettings("settings.conf", QSettings::IniFormat);

QString recentPath = settings->value("pathFile/recentFile", "").toString();

QObject::connect(ui->tmRunBtn, SIGNAL(clicked()), this, SLOT(tmRun()));

QObject::connect(turing, SIGNAL(Runable(bool)), this, SLOT(setRunable(bool)));

QObject::connect(ui->tmStopBtn, SIGNAL(clicked()), this, SLOT(tmStop()));

QObject::connect(turing, SIGNAL(updateLine(QString)), ui, SLOT(setLine(QString)));

QObject::connect(turing, SIGNAL(updatePointer(int)), ui, SLOT(setPointer(int)));

QObject::connect(ui->historyCkb, SIGNAL(toggled(bool)), this, SLOT(changeTableParameter(bool)));

QObject::connect(turing, SIGNAL(commandExecuted(QString,QString,int, QString)), this, SLOT(addHistory(QString, QString, int, QString)));

QObject::connect(ui->history, SIGNAL(cellClicked(int,int)), this, SLOT(updateFromTable(int, int)));

QObject::connect(ui->tmSrc, SIGNAL(textChanged()), ui->history, SLOT(clearAllHistory()));

QObject::connect(ui->tmLine, SIGNAL(textChanged(QString)), this, SLOT(tmLineChanged()));

QObject::connect(turing, SIGNAL(testErrors()), this, SLOT(errorTest()));

QObject::connect(app, SIGNAL(aboutToQuit()), this, SLOT(saveSettings()));

QObject::connect(app, SIGNAL(aboutToQuit()), ui, SLOT(saveFile()));

QObject::connect(ui, SIGNAL(saveSettings()), this, SLOT(saveSettings()));

loadRecentFile(recentPath);

}

void Controller::loadRecentFile(QString path) {

ui->fControl->openFromPath(path);

}

void Controller::saveSettings() {

settings->setValue("pathFile/recentFile", ui->fControl->getPathString());

settings->sync();

}

void Controller::setHistoryEnabled(bool b) {

ui->history->setHidden(b);

ui->historyCkb->setHidden(b);

ui->historyLbl->setHidden(b);

ui->clearHistoryBtn->setHidden(b);

}

void Controller::tmLineChanged() {

if(!turing->isRunning && ui->tmLine->hasFocus()) {

ui->history->clearAllHistory();

}

}

void Controller::updateFromTable(int r, int c) {

ui->tmLine->setText(ui->history->getLine(r));

ui->tmLine->setSelection(ui->history->getPointer(r), 1);

}

void Controller::addHistory(QString state, QString line, int pointer, QString command) {

if(command.isEmpty()) {

if(ui->history->getNumberHistoryItem() == 0) {

ui->history->addItem(state, line, pointer, command);

}

}

else {

ui->history->addItem(state, line, pointer, command);

}

}

void Controller::changeTableParameter(bool b) {

isTableParametres = b;

ui->history->setEnabled(b);

}

void Controller::tmRun() {

QString src = ui->tmSrc->document()->toPlainText();

if(!isTableParametres) {

ui->history->clearAllHistory();

}

if(isTableParametres && !ui->history->isEmpty()) {

ui->history->clearPart();

turing->setCustomSettings(ui->history->getState(ui->history->getNumberHistoryItem()),

ui->history->getLine(ui->history->getNumberHistoryItem()),

ui->history->getPointer(ui->history->getNumberHistoryItem())

);

}

else {

turing->setStandardSettings(ui->tmLine->text());

}

turing->run(src);

}

void Controller::tmStop() {

setRunable(false);

}

void Controller::setRunable(bool r) {

turing->isRunning = r;

ui->tmRunBtn->setEnabled(!r);

ui->tmStopBtn->setEnabled(r);

ui->clearHistoryBtn->setEnabled(!r);

ui->tmSrc->setReadOnly(r);

ui->tmLine->setReadOnly(r);

}

#include "filecontroller.h"

FileController::FileController(QObject \*parent) : QObject(parent)

{

fileOpenned = 0;

}

void FileController::clear() {

src.clear();

lines.clear();

line.clear();

path.clear();

}

QString FileController::getPathString() {

return path;

}

void FileController::openDialogEnable() {

QString pathTmp = QFileDialog::getOpenFileName(0, "Открыть файл...", "", "\*.txt");

if(!pathTmp.isEmpty()) {

clear();

path = pathTmp;

file = new QFile(path);

if(file->*open*(QIODevice::ReadWrite)) {

QTextStream tStream(file);

QString fileData = tStream.readAll();

QStringList list = fileData.split('\n');

for(int i = 0; i < list.size(); ++i) {

lines.push\_back(list.at(i));

}

if(lines.size() >= 1) {

line = lines[0];

for(int i = 1; i < lines.size(); ++i) {

src.push\_back(lines[i]);

src.push\_back('\n');

}

}

fileOpenned = 1;

}

emit opennedFile(src, line);

emit sendPath(path);

}

}

void FileController::openFromPath(QString p) {

if(!p.isEmpty()) {

clear();

path = p;

file = new QFile(path);

if(file->*open*(QIODevice::ReadWrite)) {

QTextStream tStream(file);

QString fileData = tStream.readAll();

QStringList list = fileData.split('\n');

for(int i = 0; i < list.size(); ++i) {

lines.push\_back(list.at(i));

}

if(lines.size() >= 1) {

line = lines[0];

for(int i = 1; i < lines.size(); ++i) {

src.push\_back(lines[i]);

src.push\_back('\n');

}

}

fileOpenned = 1;

}

emit opennedFile(src, line);

emit sendPath(path);

}

}

void FileController::saveFile(QString src, QString line) {

if(path.isEmpty()) {

saveDialogEnable();

}

else {

QFile file(path);

file.*open*(QIODevice::ReadWrite);

QByteArray srcBytes;

QByteArray lineBytes;

string lineStd = line.toStdString();

//srcBytes += "\n";

for(int i = 0; i < lineStd.size(); ++i) {

lineBytes.push\_back(lineStd[i]);

}

string srcStd = src.toStdString();

for(int i = 0; i < srcStd.size(); ++i) {

qDebug() << srcStd[i];

srcBytes.push\_back(srcStd[i]);

}

file.*resize*(0);

file.write(lineBytes);

file.write("\n");

file.write(srcBytes);

}

}

QString FileController::getPath(QString p) {

if(p.size() >= 5) {

if(p[p.size() - 1] == 'r' && p[p.size() - 2] == 'u' && p[p.size() - 3] == 't' && p[p.size() - 4] == 's' && p[p.size() - 5] == '.') {

return p;

} else {return p + ".txt";}

} else {return p + ".txt";}

}

void FileController::saveDialogEnable() {

QFileDialog sDialog;

QString pathTmp = sDialog.getSaveFileName(0, tr("Сохранить файл как..."), "", tr("STuring files (\*.txt)"));

if(!pathTmp.isEmpty()) {

lines.clear();

path = pathTmp;

emit saveFileSign();

file = new QFile(path);

QFileInfo info(\*file);

if(info.suffix().isEmpty()) {

info.suffix() = ".txt";

}

file->*open*(QIODevice::ReadWrite);

emit sendPath(path);

}

}

void FileController::createNewFile() {

clear();

emit opennedFile(src, line);

emit sendPath("Новый");

}

#include "history.h"

History::History()

{

setColumnCount(2);

setMinimumWidth(400);

setMaximumWidth(800);

colWidth = 180;

QStringList header;

header << "История выполнения" << "Команда";

setHorizontalHeaderLabels(header);

for(int i = 0; i < colorCount(); ++i) {setColumnWidth(i, colWidth);}

this->setSelectionMode(QAbstractItemView::SingleSelection);

this->setEditTriggers(QTableWidget::NoEditTriggers);

numberHistoryItem;

QObject::connect(this, SIGNAL(cellPressed(int,int)), this, SLOT(changeNumberHistoryItem(int,int)));

}

bool History::isEmpty() {

return (historyItem.size() == 0);

}

int History::getNumberHistoryItem() {

return numberHistoryItem;

}

void History::changeNumberHistoryItem(int r, int c) {

numberHistoryItem = r;

}

void History::addItem(QString state, QString line, int pointer) {

if(!isFirstHave()) {

historyItem.push\_back(HistoryItem(state, line, pointer));

}

updateTable();

}

bool History::isFirstHave() {

for(int i = 0; i < historyItem.size(); ++i) {

if(!historyItem[i].getCommand().isEmpty()) {return 1;}

}

return 0;

}

void History::addItem(QString state, QString line, int pointer, QString command) {

historyItem.push\_back(HistoryItem(state, line, pointer, command));

updateTable();

}

void History::clearAllHistory() {

historyItem.clear();

numberHistoryItem = 0;

updateTable();

}

void History::clearPart() {

historyItem.remove(numberHistoryItem + 1, historyItem.size() - numberHistoryItem - 1);

updateTable();

}

void History::updateTable() {

setRowCount(historyItem.size());

for(int i = 0; i < rowCount(); ++i) {

setItem(i, 0, new QTableWidgetItem(historyItem[i].getLine()));

setItem(i, 1, new QTableWidgetItem(historyItem[i].getCommand()));

}

this->scrollToBottom();

}

QString History::getState(int n) {

return historyItem[n].getState();

}

QString History::getLine(int n) {

return historyItem[n].getLine();

}

int History::getPointer(int n) {

return historyItem[n].getPointer();

}

#include "historyitem.h"

HistoryItem::HistoryItem(QString state\_, QString line\_, int pointer\_) : state(state\_), line(line\_), pointer(pointer\_) {

}

HistoryItem::HistoryItem(QString state\_, QString line\_, int pointer\_, QString command\_) : state(state\_), line(line\_), pointer(pointer\_), command(command\_) {

}

HistoryItem::HistoryItem() : state("00"), line(""), pointer(0), command("") {

}

QString HistoryItem::getState() {

return state;

}

QString HistoryItem::getLine() {

return line;

}

QString HistoryItem::getCommand() {

return command;

}

int HistoryItem::getPointer() {

return pointer;

}

#include <QApplication>

#include "controller.h"

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

Controller\* mainController;

mainController = new Controller(&a);

return a.exec();

}

#include "sturing.h"

STuring::STuring(QObject \*parent) : QObject(parent), errorsHave(0)

{

}

void STuring::run(QString src\_) {

emit Runable(true);

stopped = false;

stackSrc.clear();

QString tmp1 = src\_;

QStringList srcList = src\_.split('\n');

for(int i = 0; i < srcList.size(); ++i) {

QString cmd = srcList.at(i);

cmd = uncomment(cmd);

if(cmd.size() > 0) {

stackSrc.push\_back(cmd);

}

}

emit testErrors();

if(!stopped && !errorsHave) {

go();

}

emit Runable(false);

}

void STuring::setStandardSettings(QString line\_) {

nowState = "00";

line = line\_;

pointer = line.size();

line.push\_back(' ');

}

void STuring::setCustomSettings(QString state\_, QString line\_, int pointer\_) {

nowState = state\_;

line = line\_;

pointer = pointer\_;

}

void STuring::go() {

emit Runable(true);

for(int i = 0; i < stackSrc.size() && !stopped && isRunning; ++i) {

if(testOfExecute(stackSrc[i])) {

time.start();

for(;time.elapsed() < 100;) {

qApp->processEvents();

}

executeCommand(stackSrc[i]);

emit updateLine(line);

emit updatePointer(pointer);

i = -1;

}

if(i == stackSrc.size() - 1) {

i = -1;

stopped = true;

}

}

emit Runable(false);

}

QString STuring::get(QString& cmd, int k) {

int s = 0;

QString tmp;

for(int i = 0; i < cmd.size(); ++i) {

if(cmd[i] == ',') {++s; ++i;}

if(s == k && i < cmd.size()) tmp.push\_back(cmd[i]);

if(s > k) return tmp;

}

return tmp;

}

QString STuring::getState(QString& cmd) {

return get(cmd, 0);

}

QString STuring::getReadLetter(QString& cmd) {

return get(cmd, 1);

}

QString STuring::getWriteLetter(QString& cmd) {

return get(cmd, 2);

}

QString STuring::getNextState(QString& cmd) {

return get(cmd, 3);

}

QString STuring::uncomment(QString& str) {

QString tmp;

for(int i = 0; i < str.size(); ++i) {

if(str[i] == '/' && i < str.at(i).unicode() - 1) {

if(str[i+1] == '/') {

return tmp;

}

else {

tmp.push\_back(str[i]);

continue;

}

}

tmp.push\_back(str[i]);

}

}

bool STuring::testOfExecute(QString &cmd) {

QString state = getState(cmd);

QString readLetter = getReadLetter(cmd);

return state == nowState && readLetter[0] == line[pointer];

}

void STuring::executeCommand(QString& cmd) {

QString writeLetter = getWriteLetter(cmd);

QString nextState = getNextState(cmd);

if(writeLetter == "R") {

moveRight();

}

else if(writeLetter == "L") {

moveLeft();

}

else if(writeLetter == "S") {

stopped = true;

}

else {

changeLetter(writeLetter[0]);

}

nowState = nextState;

emit commandExecuted(nowState, line, pointer, cmd);

}

void STuring::changeLetter(QChar let) {

line[pointer] = let;

}

void STuring::moveRight() {

pointer++;

if(pointer >= line.size()) {

line.push\_back(' ');

}

}

void STuring::moveLeft() {

if(pointer > 0) {

pointer--;

}

}

#include "ui.h"

UI::UI(QApplication\* app\_, QObject \*parent) : QObject(parent)

{

programmNameString = "TM";

app = app\_;

fControl = new FileController();

tmLine = new QLineEdit();

tmRunBtn = new QPushButton("Пуск");

tmStopBtn = new QPushButton("Стоп");

clearHistoryBtn = new QPushButton("Очистить историю");

tmSrc = new CodeEditor();

infoLbl = new QLabel("");

upLayout = new QHBoxLayout();

menuLayout = new QHBoxLayout();

createActions();

turingLayout = new QVBoxLayout();

appLayout = new QHBoxLayout();

appLayout->setSpacing(3);

appLayout->setMargin(0);

upLayout->setSpacing(3);

upLayout->addWidget(tmLine);

upLayout->addWidget(tmRunBtn);

upLayout->addWidget(tmStopBtn);

upLayout->addWidget(clearHistoryBtn);

upLayout->setMargin(3);

history = new History();

rightLayout = new QVBoxLayout();

rightLayout->setMargin(3);

fontLine.setPointSize(14);

fontLine.setFamily("Monospace");

fontSrc.setFamily("Monospace");

fontLine.setStyleHint(QFont::TypeWriter);

fontSrc.setStyleHint(QFont::TypeWriter);

fontSrc.setPointSize(12);

fontLbl.setFamily("Monospace");

fontLbl.setStyleHint(QFont::TypeWriter);

fontLbl.setPointSize(12);

tmLine->setFont(fontLine);

tmSrc->setFont(fontSrc);

infoLbl->setFont(fontLbl);

turingLayout = new QVBoxLayout();

turingLayout->setSpacing(3);

turingLayout->setMargin(3);

turingLayout->addWidget(tmSrc);

appLayout->addLayout(turingLayout);

historyCkbLayout = new QHBoxLayout();

historyLbl = new QLabel("Данные из таблицы: ");

historyCkb = new QCheckBox(false);

historyCkbLayout->addWidget(historyLbl);

historyCkbLayout->addWidget(historyCkb);

historyCkbLayout->setAlignment(Qt::AlignLeft);

rightLayout->addLayout(historyCkbLayout);

rightLayout->addWidget(history);

createTableHistory();

appLayout->addLayout(rightLayout);

mainVerticalSeparator = new QVBoxLayout();

mainVerticalSeparator->addLayout(menuLayout);

mainVerticalSeparator->addLayout(upLayout);

mainVerticalSeparator->addLayout(appLayout);

mainVerticalSeparator->setMargin(0);

mainVerticalSeparator->setSpacing(0);

tmStopBtn->setEnabled(false);

setSrcSize();

icon.addFile(":/ico");

logo = QPixmap(":/ico");

logoLbl = new QLabel();

logoLbl->setPixmap(logo.scaled(100, 100, Qt::KeepAspectRatio));

programName = new QLabel(programmNameString);

QFont programNameFont = programName->font();

programNameFont.setBold(true);

programName->setFont(programNameFont);

history->setEnabled(false);

dialogMainLayout = new QVBoxLayout();

dialogMainLayout->setAlignment(Qt::AlignCenter);

dialogMainLayout->insertWidget(-1, programName, 0, Qt::AlignHCenter);

dialogMainLayout->insertWidget(-1, logoLbl, 0, Qt::AlignHCenter);

dialogMainLayout->insertWidget(-1, aboutLbl, 0, Qt::AlignHCenter);

mainWindow.setLayout(mainVerticalSeparator);

mainWindow.setWindowTitle(programmNameString);

mainWindow.setWindowIcon(icon);

mainWindow.setMinimumSize(700, 500);

mainWindow.show();

QObject::connect(tmSrc, SIGNAL(textChanged()), this, SLOT(setSrcSize()));

QObject::connect(clearHistoryBtn, SIGNAL(clicked()), history, SLOT(clearAllHistory()));

QObject::connect(fControl, SIGNAL(opennedFile(QString,QString)), this, SLOT(openFile(QString, QString)));

QObject::connect(fControl, SIGNAL(saveFileSign()),this, SLOT(saveFile()));

QObject::connect(fControl, SIGNAL(sendPath(QString)), this, SLOT(fileNameWindow(QString)));

QObject::connect(tmLine, SIGNAL(textChanged(QString)), this, SLOT(addUnSaved()));

QObject::connect(tmSrc, SIGNAL(textChanged()), this, SLOT(addUnSaved()));

fControl->createNewFile();

fileSaved = 1;

}

void UI::openFile(QString src, QString line) {

history->clearAllHistory();

tmSrc->document()->setPlainText(src);

tmLine->setText(line);

addSaved();

}

void UI::createTableHistory() {

history->updateTable();

}

void UI::dialogShow() {

aboutDialog.setGeometry((2\*mainWindow.x() + mainWindow.width())/ 2 - dialogW/2, (2\*mainWindow.y() + mainWindow.height())/ 2 - dialogH/2, dialogW, dialogH);

aboutDialog.show();

}

void UI::saveFile() {

QString src =tmSrc->document()->toPlainText();

QString line = tmLine->text();

qDebug() << src;

fControl->saveFile(src, line);

addSaved();

emit saveSettings();

}

void UI::fileNameWindow(QString str) {

mainWindow.setWindowTitle(programmNameString + " - " + str);

allWindowTitle = mainWindow.windowTitle();

}

void UI::addUnSaved() {

mainWindow.setWindowTitle(allWindowTitle + "\*");

fileSaved = 0;

}

void UI::addSaved() {

mainWindow.setWindowTitle(allWindowTitle);

fileSaved = 1;

}

void UI::createNewFile() {

fileNameWindow(" - Новый");

fControl->createNewFile();

addSaved();

}

void UI::createActions() {

menuBar = new QMenuBar();

fileMenu = new QMenu("Файл");

fileMenu->addAction("Новый", fControl, SLOT(createNewFile()), Qt::CTRL + Qt::Key\_N);

fileMenu->addAction("Открыть...", fControl, SLOT(openDialogEnable()), Qt::CTRL + Qt::Key\_O);

fileMenu->addAction("Сохранить", this, SLOT(saveFile()), Qt::CTRL + Qt::Key\_S);

fileMenu->addAction("Сохранить как...", fControl, SLOT(saveDialogEnable()), Qt::CTRL + Qt::SHIFT + Qt::Key\_S);

fileMenu->addAction("Выход", app, SLOT(quit()), Qt::CTRL + Qt::Key\_Q);

menuBar->addMenu(fileMenu);

menuBar->addSeparator();

menuLayout->addWidget(menuBar);

}

void UI::setSrcSize() {

QString str = "Символы: ";

int numChars = tmSrc->document()->characterCount() - 1;

QString numCharsS = QString::number(numChars);

str += numCharsS;

str += "; Строки: ";

int numStrings = tmSrc->document()->lineCount();

QString numStringsS = QString::number(numStrings);

str += numStringsS;

str += ".";

infoLbl->setText(str);

}

void UI::setPointer(int n) {

tmLine->setSelection(n, 1);

}

void UI::setLine(QString s) {

tmLine->setText(s);

}

Логотип для програмы: ST\_new.png

