Содержание:

Введение.

Прежде чем перейти к практической части моей курсовой работы, необходимо ознакомиться с ее теоретической частью во введении.

Итак, что такое алгоритмы и алгоритмические модели? Алгоритм - это точно определенный набор правил, который при заданном начальном сообщении представляет собой конечное число шагов для получения некоторого выходного сообщения. Все это определение было бы замечательным, если бы не одно "но". Более строгая интерпретация этого определения требует так называемых алгоритмических моделей. Например, рекурсивные функции (понятие алгоритма связано с вычислениями и числовыми функциями), машины Тьюринга (алгоритм выражается как описание процесса в машине, которая может выполнять лишь небольшое количество очень простых операций), обычные марковские алгоритмы (алгоритм - это произвольное алфавитное слово описанное как преобразование). В моей курсовой работе фигурируют только машины Тьюринга и обычные алгоритмы Маркова. После этого краткого вступления мы можем перейти к объяснению целей и задач данного исследования. Цель: проиллюстрировать определения алгоритма путём построения алгоритмических моделей Тьюринга и Маркова.

Задачи: составить программу машины Тьюринга в четвёрках, выполняющую заданное действие над словами, записанными на ленте;

1) разработать диаграмму Тьюринга решения задачи с использованием стандартных машин (r, l, R, L, K, a) и вспомогательных машин, определяемых задачей.
2) разработать нормальный алгоритм Маркова, обменивающий местами два троичных числа, разделённых знаком "^".
3) обобщить полученную информацию и сделать соответствующий вывод.

Для выполнения поставленных задач мне необходимы:

- эмулятор машины Тьюринга в четвёрках
- диаграммер для работы с диаграммами Тьюринга («VirtualTuringMachine.exe»)2
- эмулятор для нормальных алгоритмов Маркова
- текстовый редактор Microsoft Word (не хочется мне использовать Latex)

- мой репозиторий на GitHub, куда были выложены все описанные ниже алгоритмы(https://github.com/LernerF/capitoliy)
- Список необходимой литературы:
    - С.С.Гайсарян, В.Е.Зайцев «Курс информатики», Москва, Издательство Вузовская книга, 2013 г.

На этом вводный раздел о курсовой работе закончен. Настало время подумать о самой алгоритмической модели, но все же хотелось бы добавить материал от себя(данный список может присутствовать чисто для ознакомления):

1. "Introduction to the Theory of Computation" от Michael Sipser – это классический учебник по теории вычислений, который содержит подробные объяснения и примеры работы машин Тьюринга и алгоритмов Маркова.

2. "Computability and Logic" от George S. Boolos, John P. Burgess, Richard C. Jeffrey - эта книга предоставляет обширное введение в теорию вычислений, включая диаграммы, машины Тьюринга и алгоритмы Маркова.

3. "The Annotated Turing: A Guided Tour Through Alan Turing's Historic Paper on Computability and the Turing Machine" от Charles Petzold - этот текст представляет собой аннотированный комментарий к оригинальной статье Алана Тьюринга о машинах Тьюринга, что поможет вам понять историю и основные принципы работы машин Тьюринга.

4. "Theory of Computation" от Dexter C. Kozen - эта книга предлагает формальный и математический подход к теории вычислений, включая темы, связанные с машинами Тьюринга.

5. "Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science" от Martin Davis, Ron Sigal, Elaine J. Weyuker - этот учебник предоставляет введение в фундаментальные концепции теории вычислений, включая диаграммы, машины Тьюринга и алгоритмы Маркова.

Эти книги предоставляют различные подходы к изучению теории вычислений и могут помочь вам углубить свои знания в области диаграмм, машин Тьюринга и алгоритмов Маркова.

1. Машина Тьюринга.

1.1. Теоретическая часть, связанная с МТ

Пока я не начал описывать сделанную машину Тьюринга, необходимо дать ей более точное определение для простоты восприятия материала. Итак, Машиной Тьюринга называется упорядоченная четверка объектов T = (A, Q, P, q0), где T - символ МТ, A - конечное множество букв (рабочий алфавит), Q - конечное множество символов (имен состояний), q0 - имя начального состояния, P - множество упорядоченных четверок (q, a, v , q ′), q ,q ′∈ Q, a ∈A∪{λ}, v ∈ {l, r } ∪ A ∪ {λ} (программа), определяющее три функции: функцию выхода Fl: Q∗¯A→¯A (¯A =A∪{λ}),функцию переходов Ft :Q∗¯A → Q, и функцию движения головки Fv : Q ∗¯A → {l, r, s} (символ s означает, что головка неподвижна).

Это определение сложно и не понятно, поэтому более упрощенно оно будет выглядеть так:

1) начального состояния (то, в котором находится головка до выполнения следующей команды)
2) начальной буквы, расположенной там, где находится головка
3) команды, которую должна сделать машина Тьюринга. Это может быть:
   - переход на одну ячейку влево;
   - переход на одну ячейку вправо;
   - замена начальной буквы на другую;
   - ничего не делать, просто в пункте 4 сменить состояние (смысла в такой команде нет, поэтому её не упоминают).
4) состояния, в которое должна перейти головка.

Небольшая вставка: если в четвёрке не описано какое-нибудь действие (команда или смена состояние), то это приведёт к зацикливанию программы и всё будет очень плохо.

Например: 01,0,0,01 – у вас просто будет постоянно ставится ноль, и головка никогда не сдвинется с места.

Подводя итоги теоретической части, нужно сказать, что на вот этих «четвёрках» и построено определение алгоритма по Тьюрингу. С их помощью и задаются правила, по которым можно определить за конечное число шагов необходимое нам преобразование сообщения. Определение алгоритмов по Тьюрингу настолько фундаментально, что с его помощью можно описать всё, что только возможно представить в виде алгоритма.

А теперь пришло время продемонстрировать работу моей машины Тьюринга. Её задача заключалась в том, чтобы проверить делится ли число на 11 или нет. Сначала я постараюсь описать основополагающую информацию для выполнения данной лабораторной работы.

Примером такой информации является таблица истинности для проверки чисел на делимости (условно на 11).

Девятеричная система счисления	Троичная система счисления
5	0
6	0
7	0
8	0
10	0
11	1
13	0
15	0
22	1

Так можно сделать со всеми системами счисления.

Не стоит кстати про нормированность. Чтобы её соблюдать, пришлось ввести некоторые дополнительные команды, о которых я расскажу позднее.

Перед тем, как показывать работу программы добавлю сода несколько тестов для машины.

Входное сообщение	Выходное сообщение
11	1
10100110	0
121121	1
0	0

Таким образом, мы получим формальное определение алгоритма через алгоритмическую модель Тьюринга, где командами или состояниями я буду называть правила, связанные между собой и предназначенные для выполнения поставленной перед ними задачи.

1.2. Демонстрация сделанной МТ и описание команд

Так как вставить всю машину целиком и так подойти под необходимый объём курсовой будет слишком просто, я решил демонстрировать команды лишь для одной-двух цифр для экономии пространства. Кроме того, было бы проще показывать состояния не по номерам, как это обычно делается, а по названиям, которые я сам им дал. Так будет гораздо проще разобраться в работе программы.

Итак, начнём с команды первого перемещения головки в начало

Шаг 1: Обработка входных сообщений (`gtf`):

В данном коде gtf используется для определения отношений между сущностями. Каждое сообщение имеет формат <идентификатор1>, <оператор сравнения>, <идентификатор2>. Например, gtf, 0 , < , gtf означает условие, что значение сущности 0 должно быть меньше значения сущности gtf.

Шаг 2: Правила обработки ROD (`r0` до `r10`):

ROD представляют собой правила, в которых задаются условия на основе входных сообщений. Например, r0,0,>, r0 говорит о том, что если условие для сущности 0 выполняется, нужно перейти к следующему правилу или выполнить определенное действие.

Шаг 3: Генерация ответного сообщения (`yes` и `no`):

В зависимости от выполненных условий в правилах, система генерирует ответное сообщение. Например, yes, ,1 ,hltno, ,0,hlt говорит о том, что если определенные условия выполнены, генерируется сообщение yes, иначе - no.

Шаг 4: Команды остановки (`hlt` и `hlthlt`):

Код включает команды для прекращения выполнения. Например, hlt,0,>,hlt указывает, что выполнение прекращается, если условие для сущности 0 выполняется. hlthlt добавляет дополнительное условие для завершения.

Общий ход выполнения:

1. Обработка ввода: Система анализирует входные сообщения (`gtf`).
2. Обработка правил (`r0` до `r10`): Исходя из входных условий, система следует набору правил для принятия решений.
3. Генерация ответа: на основе выполненных условий в правилах генерируется ответное сообщение (`yes` или `no`).
4. Остановка при необходимости (`hlt` и `hlthlt`): Выполнение прекращается при наступлении определенных условий. Если у вас есть конкретные вопросы по шагам или элементам кода, которые вызывают затруднение, уточните, и я буду рад помочь.

Шаг 5.1: Обработка входных сообщений (`gtf`):

Каждое gtf сообщение определяет отношение между двумя идентификаторами сущностей с использованием оператора сравнения. - Пример: gtf, 0, <, gtf указывает, что значение сущности 0 должно быть меньше значения сущности gtf.

Шаг 5.2: Правила обработки ROD (`r0` до `r10`):

Правила (`r0` до r10`) задают условия для принятия решений, основанных на входных данных `gtf. - Например, r0, 0,>, r0 говорит о том, что если условие для сущности 0 выполняется, система переходит к следующему правилу или выполняет определенное действие.

Шаг 5.3: Генерация ответного сообщения (`yes` и `no`):

В зависимости от результата выполнения правил система генерирует ответное сообщение. - Пример: yes, ,1, hltno, ,0, hlt означает, что при определенных условиях система выдаст ответ yes, в противном случае - no.

Шаг 4: Команды остановки (`hlt` и `hlthlt`):

Команды hlt указывают, когда система должна прекратить выполнение. - Например, hlt,0,>,hlt означает, что выполнение завершится, если условие для сущности 0 выполняется. Дополнительное hlthlt предполагает наличие дополнительного условия для завершения.

Общий ход выполнения:

1. Обработка ввода (`gtf`): Анализ отношений между сущностями на основе входных данных.
2. Обработка правил (`r0` до `r10`): Система следует правилам для принятия решений на основе входных условий.
3. Генерация ответа (`yes` и `no`): В зависимости от выполненных условий в правилах, система генерирует ответное сообщение.
4. Остановка при необходимости (`hlt` и `hlthlt`): Выполнение завершается при наступлении определенных условий. Если у вас есть конкретные аспекты кода или шаги, которые вызывают затруднения, уточните, и я постараюсь предоставить более детальные пояснения.

Программа:

```
// go to the front of input message
gtf,0,<,gtf
gtf,1,<,gtf
gtf,2,<,gtf
gtf,3,<,gtf
gtf,4,<,gtf
gtf,5,<,gtf
gtf,6,<,gtf
gtf,7,<,gtf
gtf,8,<,gtf
gtf,9,<,gtf
gtf, ,>,r0

// RODs processing
r0,0,>,r0
r0,1,>,r1
r0,2,>,r2
r0,3,>,r3
```

r0,4,>,r4
r0,5,>,r5
r0,6,>,r6
r0,7,>,r7
r0,8,>,r8
r0,9,>,r9
r0, ,>,yes

r1,0,>,r10
r1,1,>,r0
r1,2,>,r1
r1,3,>,r2
r1,4,>,r3
r1,5,>,r4
r1,6,>,r5
r1,7,>,r6
r1,8,>,r7
r1,9,>,r8
r1, ,>,no

r2,0,>,r9
r2,1,>,r10
r2,2,>,r0
r2,3,>,r1
r2,4,>,r2
r2,5,>,r3
r2,6,>,r4
r2,7,>,r5
r2,8,>,r6
r2,9,>,r7
r2, ,>,no

r3,0,>,r8
r3,1,>,r9
r3,2,>,r10
r3,3,>,r0
r3,4,>,r1
r3,5,>,r2
r3,6,>,r3
r3,7,>,r4
r3,8,>,r5
r3,9,>,r6
r3, ,>,no

r4,0,>,r7
r4,1,>,r8
r4,2,>,r9
r4,3,>,r10
r4,4,>,r0
r4,5,>,r1
r4,6,>,r2
r4,7,>,r3
r4,8,>,r4
r4,9,>,r5
r4, ,>,no

r5,0,>,r6
r5,1,>,r7
r5,2,>,r8
r5,3,>,r9
r5,4,>,r10
r5,5,>,r0
r5,6,>,r1
r5,7,>,r2
r5,8,>,r3
r5,9,>,r4
r5, ,>,no

r6,0,>,r5
r6,1,>,r6
r6,2,>,r7
r6,3,>,r8
r6,4,>,r9
r6,5,>,r10
r6,6,>,r0
r6,7,>,r1
r6,8,>,r2
r6,9,>,r3
r6, ,>,no

r7,0,>,r4
r7,1,>,r5
r7,2,>,r6
r7,3,>,r7
r7,4,>,r8
r7,5,>,r9
r7,6,>,r10
r7,7,>,r0

```
r7,8,>,r1
r7,9,>,r2
r7, ,>,no

r8,0,>,r3
r8,1,>,r4
r8,2,>,r5
r8,3,>,r6
r8,4,>,r7
r8,5,>,r8
r8,6,>,r9
r8,7,>,r10
r8,8,>,r0
r8,9,>,r1
r8, ,>,no

r9,0,>,r2
r9,1,>,r3
r9,2,>,r4
r9,3,>,r5
r9,4,>,r6
r9,5,>,r7
r9,6,>,r8
r9,7,>,r9
r9,8,>,r10
r9,9,>,r0
r9, ,>,no

r10,0,>,r1
r10,1,>,r2
r10,2,>,r3
r10,3,>,r4
r10,4,>,r5
r10,5,>,r6
r10,6,>,r7
r10,7,>,r8
r10,8,>,r9
r10,9,>,r10
r10, ,>,no

// print answer message
yes, ,1,hlt
no, ,0,hlt
```

```
// halt (end of work)
hlt,0,>,hlt
hlt,1,>,hlt
hlt, ,#,hlt
```

На этом объяснение программы заканчивается и мы плавно переходим к оценки и выводам.

1.3. Сложностная оценка и выводы к машине Тьюринга

1. Сложностная оценка машины Тьюринга:

Машина Тьюринга - это абстрактная модель вычислений, которая состоит из бесконечной ленты, на которой записан входной символ, и головки, способной перемещаться по ленте и изменять ее содержимое.

Машина Тьюринга может быть использована для моделирования любого алгоритма, и ее сложность оценивается с учетом количества шагов, необходимых для завершения задачи.

2. Минусы машины Тьюринга:

Ограниченность модели: Машина Тьюринга имеет свои ограничения, например, она работает с бесконечной лентой и не учитывает реальные ограничения ресурсов компьютеров.

Недостаточная абстракция: Модель машины Тьюринга может быть недостаточно абстрактной для некоторых задач, таких как параллельные вычисления или работы с большими объемами данных.

Сложность анализа: Оценка сложности алгоритмов с использованием машины Тьюринга может быть сложной и трудоемкой задачей. Таким образом, машина Тьюринга имеет свои ограничения и минусы, которые не позволяют использовать ее в качестве универсального инструмента для оценки сложности алгоритмов.

Также хочу сделать собственный вывод о работе с машиной Тьюринга.

Работа была крайне не простой, для решения данный задачи с помощью машины выдающегося математика пришлось напрячь многие мышцы в том числе, находящиеся вне головного черепа, не скажу, что работа неактуальна, она наглядно показывает как работает алгоритм данный машины, и заставляет действительно посидеть и подумать над ее решением, к тому же неизменный плюс данной ЛБ в том, что можно показать скрин программы менее просвещенным в этой теме людям, которые своим искренним непониманием и

удивлением смогут поднять ваше тщеславие и повысить самооценку, собственно говоря для многих это отличный плюс. Эта работа была непростой, впереди нас ожидают сложные испытания, которые нам необходимо преодолеть. Эта работа подготавливает нас к ним.

Мне понравилось собирать и делать эту программу по кусочкам, но мне не понравилось, что это было весьма долго, и сначала было крайне непросто осознать что вообще делать, что, собственно, от меня хотят? Чему же я научился за эту ЛБ? Пожалуй, терпению, стойко стоят перед лицом сложных и непонятных задач, а также я впервые сделал алгоритм, который поэлементно считывает программу, это новый подход к решению разного рода задач, который я опробовал в этой работе, надеюсь се знание мне пригодится.

2. Диаграмма Тьюринга

2.1. Теоретическая часть, связанная с диаграммами Тьюринга, и тесты

Стандартные машины Тьюринга имеют ряд существенных недостатков. Из тех, что были обнаружены при выполнение лабораторной работы, можно выделить следующие:

1) Приходится прописывать состояния для перемещения головки для каждой буквы отдельно, что очень замедляет работу.
2) Даже моя простая программа имеет длину более тысячи строк, и в формате tu4 воспринять её практически невозможно даже мне. Чтобы этого избежать, приходилось давать состояниям длинные имена, что тоже не ускоряет и не упрощает работу.
3) Кроме того, я боюсь даже представить, насколько сложно было бы копировать слова в формате tu4. Если бы мне пришлось этим заняться, программа увеличилась бы ещё раза в полтора.

Чтобы решить эти проблемы и придать машине Тьюринга «человеческое лицо», были придуманы так называемые диаграммы Тьюринга.

Диаграммы Тьюринга представляют одни МТ через другие, более простые МТ иным, визуально-топологическим способом, причём, как будет показано далее, этот способ не менее строг и полон, нежели "обычные" МТ. Так, машина, копирующая на ленте записанное на ней слово, может быть представлена через МТ, которые ищут начало слова на ленте, конец слова на ленте, копируют одну из букв слова и т. д. Эти более простые МТ в свою очередь могут быть представлены через еще более простые МТ и т. д. Такой нисходящий процесс представления МТ через более простые МТ должен обязательно оборваться, так как рано или поздно мы сведем описание каждой из рассматриваемых МТ к элементарным действиям, введенным при определении МТ. При этом рассматриваемая МТ будет описана через элементарные МТ, т. е. такие, которые уже нельзя описать через более простые МТ, так как каждая из них выполняет всего одно элементарное действие и останавливается.

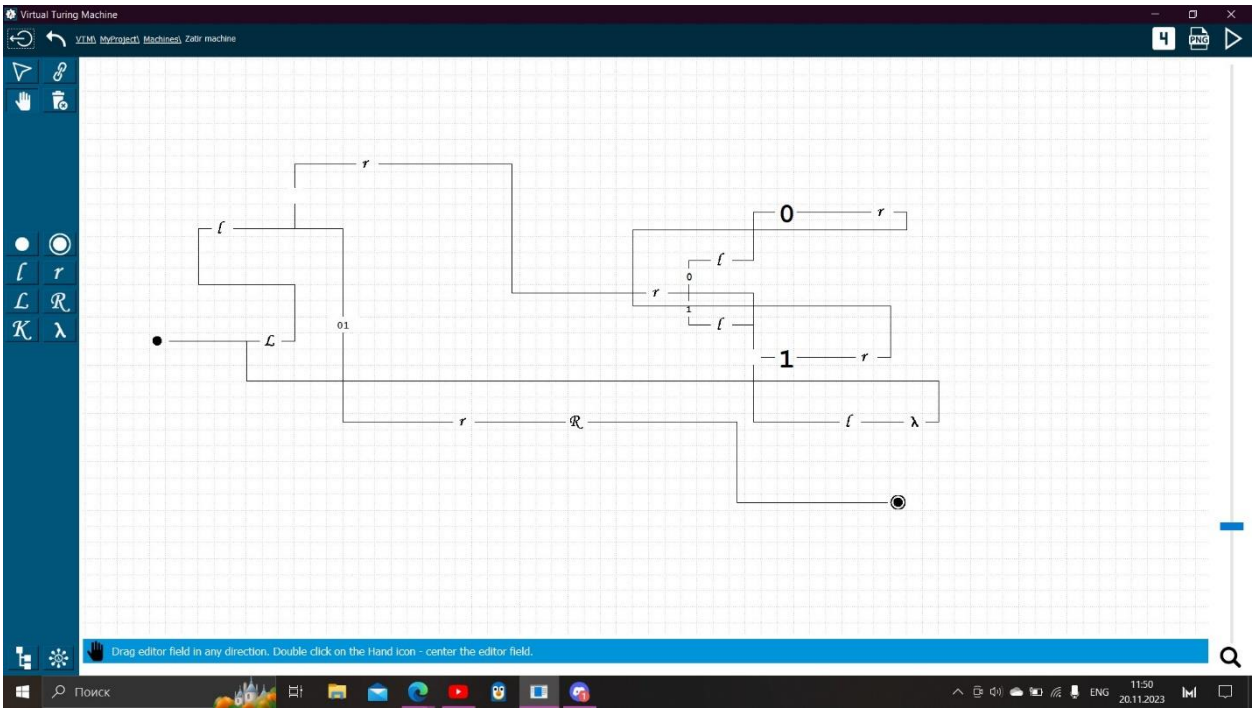
Эти элементарные МТ решают описанные мною проблемы 1) и 3). А топографический формат записи и возможность разбивать программу на части ликвидируют второй недостаток «обычных» машин Тьюринга.

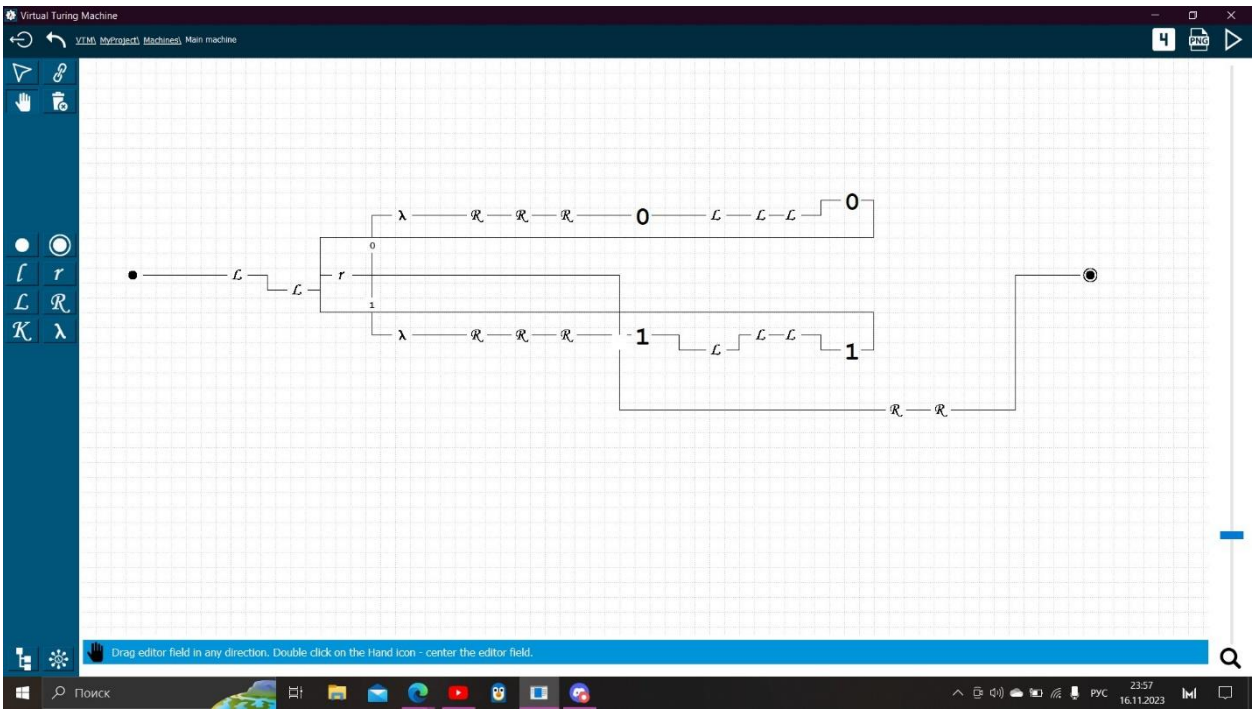
Как раз на элементарных МТ стоит остановиться поподробнее.

1) Машина сдвига на одну ячейку влево (обозначается l)
2) Машина сдвига на одну ячейку вправо (обозначается r)
3) Машина сдвига влево до конца слова (обозначается L)
4) Машина сдвига вправо до конца слова (обозначается R)
5) Машина копирования слова (обозначается K)
6) Машина постановки символа (по умолчанию λ)

На этих машинах как раз и основана моя диаграмма Тьюринга, которую я продемонстрирую далее. Вычисление двоичного логического сдвига второго числа влево на число разрядов, равное первому

2.2. Диаграмма машин Тьюринга





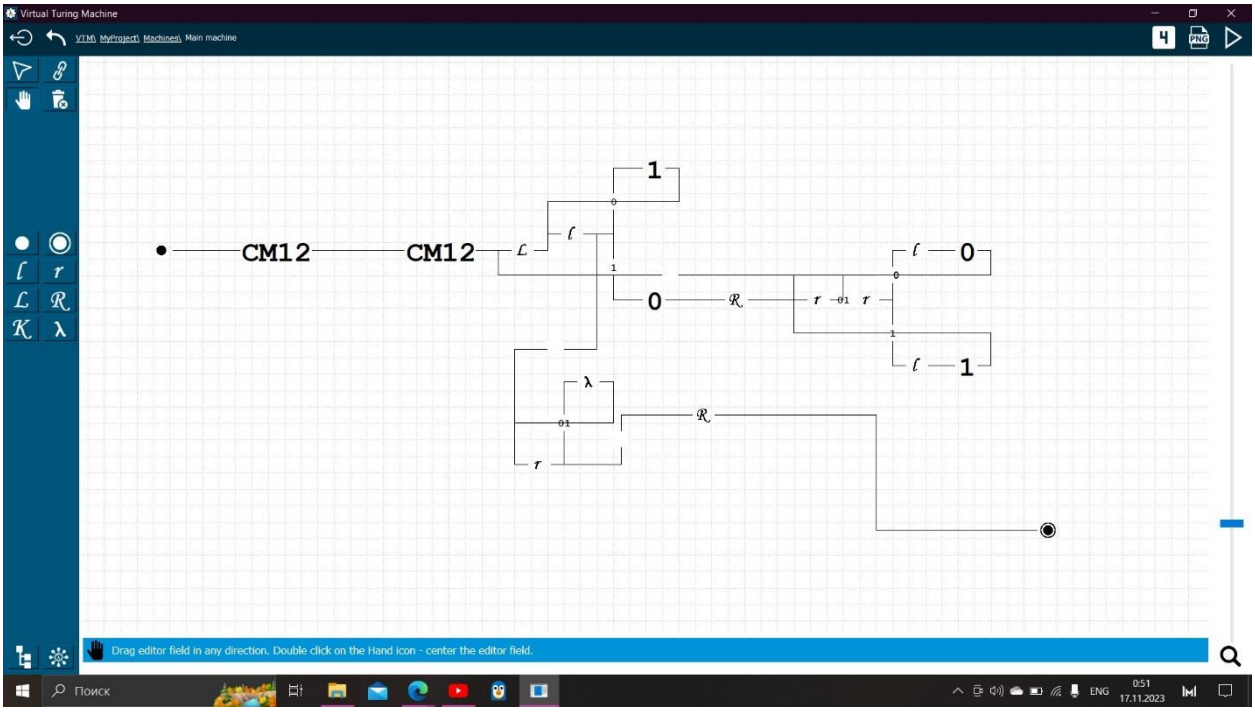


Диаграмма состоит из нескольких частей, поэтому имеет смысл продемонстрировать её по частям.

Вот так мы хотя бы можем увидеть, как мы начинаем работу программы с левой стороны. В моей программе копирование выполняется совершенно иначе, нежели в java, я не мог просто взять и через k сделать копирование по

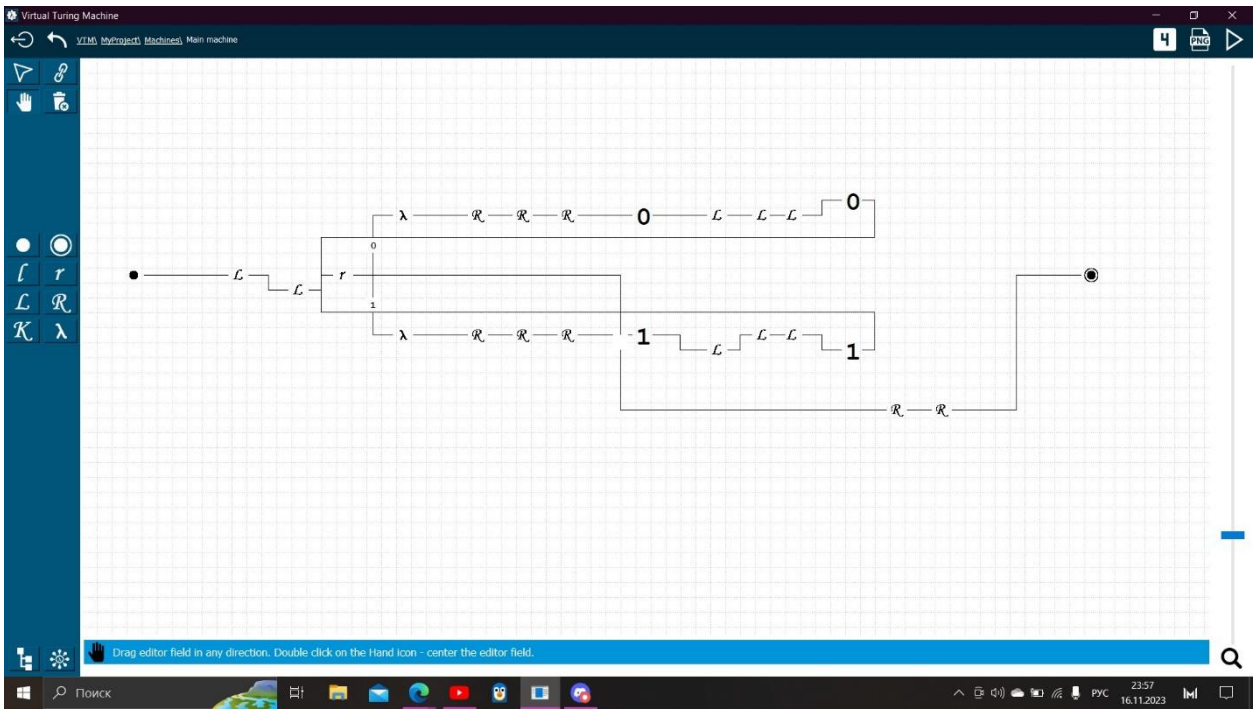
словам, мне пришлось по элементам копировать два слова, при этом не забывая затирать пробелы, с точки зрения реализации это было не сложно, головка машины пробегалась по каждому элементу, после чего бежала обратно, делая пробел, и ставя символ в соответсвующее поле.

Затирание нулей та проблема, с которой я столкнулся при попытках сломать программу, чтобы устранить это, я создал подмашину, которая брала на себя данную работу. Логика этой минипрограммы очень проста, если шло несолько нулей подряд впереди, она затирала их, пока не встретит символ отличный от нуля, при этом каждый раз выполняя цикл

Почему на ленте появляется ноль – это вопрос уже к подмашинам, о которых я расскажу позднее. Он необходим для обозначения количества нулей или единиц, которые я поставил слева от числа, чтобы его количество цифр было дополненно. Эти цифры и сам ноль (хотя там может быть единица и двойка необходимо убрать для нормированности вычислений). Для этого головка перемещается в начало входного сообщения и ещё одним прыжком доходит до левого нуля, считывает, а после и стирает его. В зависимости от числа головка смещается вправо и либо ничего не стирает, а просто идёт в конец второго слова и завершает работу программы, либо перед этим стирает одну или две первых цифры входного сообщения. На этом работа основной диаграммы и всей программы в целом подходит к концу. Пришло время перейти к вспомогательным диаграммам, которые я вскользь уже упоминал.

2.3. Диаграммы Zatir и Perenos.

Свои разъяснения я начну с диаграммы Perenos. Вот и она сама:



Проблема программы в которой я реализовал диаграму Тьюринга состоит в том, что там напрочь отсутсвует функция копирования, поэтому самый первый этап любой диагрммы Тьюринга мне пришлось делать вручную, буквально по элеиентам, копируюя каждый символ. Работа этой диагрммы начинается сразу после обработки первой цифры входного числа.Наше головка стоит справа от двух слов, первым шагом мы идем в самый левый край, а далее иде, тотальная обработка чисел, неизменно, какое число нам не попадется, мы копируем его и выносим вправо после второго слова, потом мы возвращаемся обратно, дижемся вправо копируем, идем в левый край и так далее, мы делаем это до тех пор, пока не встретим ничего. Программа довольно проста и является довольно быстрой, недаром даже она сама напоминает некоторую ракету или пулю, понимаете здесь скрывается некий символизм. Эта программа фундаментальна и играет ключевую роль, без нее, в отличии от Perenos, никак не справится. Но разберем ее поподробней:

Шаг 1: первый этап это сместить курсор на два слова влево, далее мы переходим на символ вправо и переходим к следущему шагу

Шаг 2: у нас идет раздление на ‘три путя’, мы рассмотримм подробно каждый из них, но сейчас выделю один момент, третий путь является

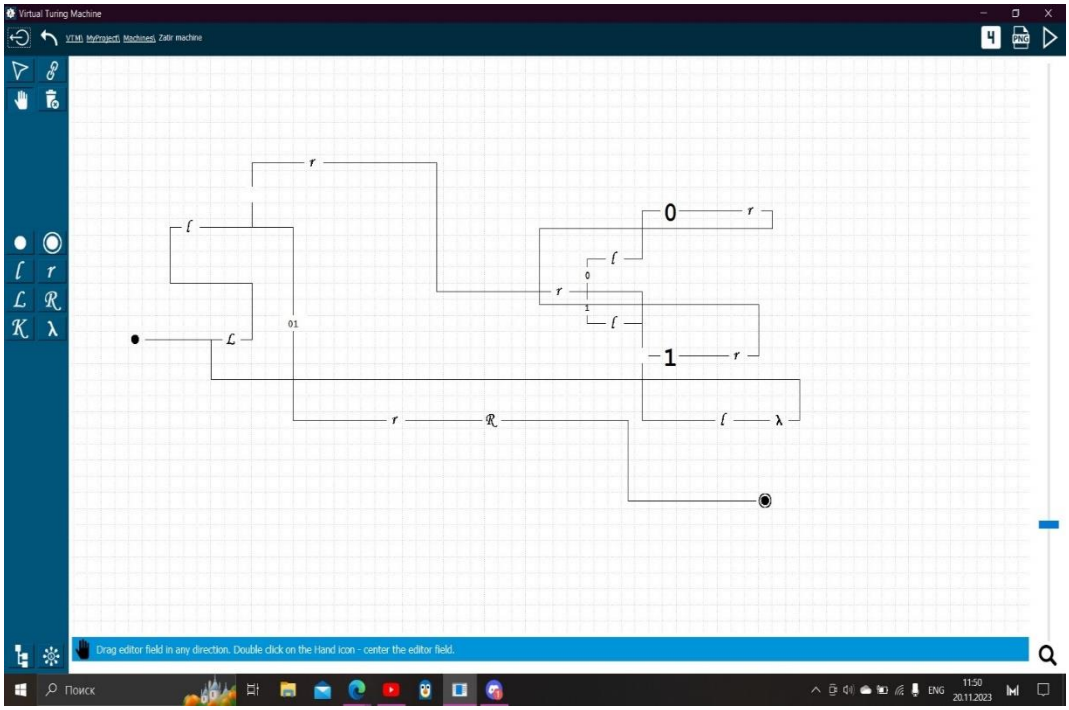
последним и задействуется машиной, только после выполнения двух первых, которые могут идти в разнобой между собой.

Шаг 3: Если мы видим 0, то мы фиксируем этот символ, идем на три слова вправо, ставим 0, далее передвигаемся обратно на три слова влево и видим соответсвенно опять 0, цикл повторется.

Шаг 4: Мы описали как действует программа с 0, как его переносит, стоит сказать что с 1 происходят абсолютно аналогичные вещи, потому что по факту нам без разницы какое число стоит, важно чтобы это было число, поэтому я сюда просто продублирую Шаг 3: если мы видим 1, то мы фиксируем этот символ, идем на три слова вправо, ставим 1, далее передвигаемся обратно на три слова влево и видим соответсвенно опять 1, цикл повторется.

Шаг 5: Это заключительный шаг, этап нашей программы, если мы видим пробел или же ничего, то мы переносим курсор на два слова вправо и заканчиваем программу подмашины Perenos.

Zatir machine



Эта подмашина не играет такой ключевой роли в отличии от двух предыдущич, описанныч мною диаграмм, которые были необходимы для работы. Данная подмашина служит скорей для наведения ‘красоты’. Она убавляет все лишние пробелы. Дело в том, что при выполнении двух других подмашин, у нас образуются пробелы между двумя словами, не соответвующие определенным нормам, дабы это исправить, была созлдана эта машина. Вот принцип работы

Шаг 1: мы идем влево, к началу первого слова, далее сдвигаемся на один элемент влево еще раз, чтобы оказаться на пробеле, если мы на нем оказываемся, то переходим к Шагу 2, если пробела не будет то тогда мы выбираем Шаг 3

Шаг 2: Увидев пробел(если он лишний) мы должны его ликвидировать, но для этого нам необходимо убедиться, что это единсвенный пробел, поэтому дл начала мы сдвигаемся вправо, увидев второй пробел, мы переходим вправо, увидев либо 0, либо единицу, мы возвращаемся обратно, на пробел, затираем его и ставим число, стоящее перед ним, и так делаем до тех пор, пока все пробелы не закончаться.

Шаг 3: если же ситуция инаяя(мы стоим не на пробеле), то все в порядке, затирать ничего не нужно, мы можем проверять дальше, свдигаясь по элементно, мы возвращемся к Шагу 1. Мы возвращаемся до тех, пока не убедимся что полностью все лишнии пробелы затерты, далее последний этап, то есть конец этой программы. Мы возвращаемся в самое начало, если курсоср видит 1 или 0, то мы сдвигаемся на один элемент вправо, далее в конец слова и конец.

2.4. Диаграмма Main Machine

Стоило разместить эти диаграмме в самом начале, так как она считается самой главной диаграммой, грубо говоря диаграммой, куда ведут все дороги, но подумал, что для начала стоит разобраться с мелкими конструкциями, дабы потом полностью понимая процессы, разобраться что и как происходит.

Что ж, начнем с самого начала, первым что у нас стоит это машина CM12, но на самом деле это ни что иное как машина Perenos, которая как я рассказывал отвечает за копирование всех символов и перенос их на один пробел правее наших изначальных слов. Этот процесс стоит повторит два раза, поскольку слова у нас два, поэтому и копируем мы два раза, дублируем нашу диаграмму. Далее, после того как мы выполнили наши первые шаги, переходим к главной составляющей нашей Main Machine.

Но перед тем, как перейти к анализу главной диаграммы, я хочу разобрать на “словах” как она работает, то есть объяснить логику, механику этой задачи, это очень важно, поскольку не понимая этой основы, мы не сможем реализовать нашу программу, а следовательно, не выполним, поставленную перед нами задачу.

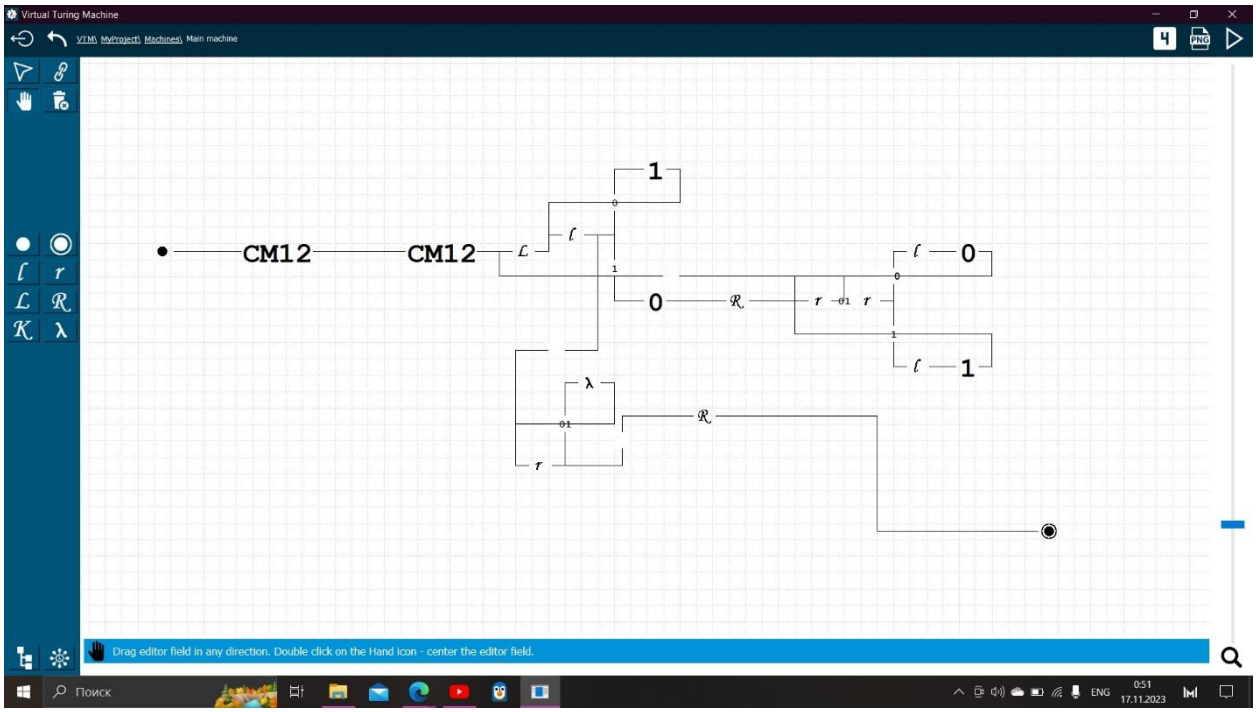
Итак, думаю стоит начать рассказывать базу теории с вопроса стоящего в самой задаче.

Что такое логический сдвиг? Логический сдвиг – это операция, при которой все биты в битовом значении сдвигаются на определенное количество позиций влево или вправо.

При этом, в зависимости от типа сдвига, освободившиеся биты либо заполняются нулями, либо копируются из соседних битов. Логический сдвиг может использоваться для ускорения операций умножения и деления на степень двойки, а также для манипуляций с битовыми значениями.

Наконец, мы поняли, что же такое логический сдвиг, осталось понять как нам с помощью курсора и замены символов реализовать это на одной лишь диаграмме. Самый ключевой здесь момент это, способ, при помощи которого, мы сможем посчитать то, на сколько нам надо сдвинуть левое слово.

Это можно сделать только одним способом. Благодаря вычитанию, оно будет действовать также, как и обычное вычитание, условно, в столбик. Если мы будем видеть единицу, то мы просто заменим ее на ноль, если же мы будем видеть 0, то заменим его на единицу, и перейдем к следующему числу, и далее будем действовать согласно Шагам, что я опишу снизу.



Мы находимя на данный момент около симола L, означающего перенос на лево другого слова, мы сдвигаемся на одни элемент влево, после чего курсосром можем оказаться либо на 0, либо на 1, либо же просто пробеле.

Если мы оказываемся на 0, то мы вычитаем из нашего числа 1, то есть производим перенос числа, как мы делаем при обыночном вычитании в столбик,то есть заменяем 0 на 1, и идем обратно к L, то есть производя цикл

Если мы оказываемся на 1, то заменяем 1 на 0, тут нет ничего сложного, после замены мы сдвигемся вправо на слово, на еще один символ вправ, и длаее производим цикл с r, соединяя его с условием, что если у нас единица, то мы пишем единицу, если ноль, то пишем ноль, далее опять возвращемся к L, зацикливая его с этим действием тоже.

Далее, последний, то есть заключительный этап нашей главной диаграммы и по совместительсву центральной машиной Main Machine, это процесс окончания программы. Мы должны в этот момент находится у l, от которой и шли наши два варианта, что ж, теперь стоит рассказать о третьем варианте, это если мы будем на пробеле, если у нас элемент окажеться пробелом, то мы идем правее, дабы проверить, что после наши элементы, будут отличны, то есть будут либо 1, либо 0. После того как мы проверим у нас будет два варинта:

1. Если правый элемент оказывается пробелом, то это означает, что мы выполнили программу, она сработала, и нам остается только ее завершить, что мы собственно говоря и делаем.
2. Второй же вариант не такой радужный как первый, наш элемент отличен от пробела, а это означает, что мы не до конца выполнили нашу программу и нам стоит заменить наши элементы и снова вернуться к правому элементу.

2.6. Итоговая оценка диаграммы Тьюринга и выводы

Работа с диаграммой Тьюринга может быть важной частью курсовой работы, особенно если вы занимаетесь изучением теории вычислимости или алгоритмов.

Диаграмма Тьюринга представляет собой графическое представление работы машины Тьюринга, которая является абстрактной моделью вычислительного устройства.

При анализе диаграммы Тьюринга в курсовой работе важно рассмотреть различные аспекты, такие как описание состояний, символов на лентах, переходы между состояниями и т.д. Вы также можете провести анализ сложности алгоритма, представленного в диаграмме Тьюринга, и оценить его временную и пространственную сложность.

Но также при работе с диаграммой Тьюринга можно столкнуться с несколькими проблемами, включая:

1.Сложность представления:

Диаграммы Тьюринга могут стать сложными и запутанными, особенно для более сложных задач. Это может затруднить понимание работы машины Тьюринга и усложнить отладку и анализ.

2. Ограниченность пространства:

При решении некоторых задач машина Тьюринга может потребовать большое количество состояний и символов на ленте, что может привести к ограничениям в представлении всей работы машины на диаграмме.

3. Трудности визуализации:

Некоторые операции и переходы машины Тьюринга могут быть трудны для визуализации на диаграмме, особенно если они включают большое количество шагов или условий.

4. Ошибки и опечатки:

При создании диаграммы Тьюринга могут возникнуть ошибки или опечатки, которые могут привести к неправильному поведению машины. Это может потребовать дополнительного времени на отладку и исправление ошибок.

5.Сложность моделирования:

Диаграммы Тьюринга могут быть сложными для моделирования более сложных алгоритмов и проблем, так как требуют детального представления всех состояний и переходов. Для решения этих проблем важно хорошо

понимать принципы работы машин Тьюринга, использовать четкие и структурированные диаграммы, а также проводить тщательную отладку и проверку созданных диаграмм.

В целом, работа с диаграммой Тьюринга для курсовой работы может быть интересным и содержательным заданием, которое позволит вам углубить свои знания в области теории вычислений и алгоритмов.

3. Нормальные алгоритмы Маркова (НАМ)

3.1. Теоретическая часть, связанная с НАМ

Нормальный алгоритм Маркова (НАМ) представляет собой упорядоченный набор правил-продукций — пар слов (цепочек знаков, в том числе пустых цепочек длины 0), соединенных между собой символами → или 7 →. Каждая продукция представляет собой формулу замены части входного слова, совпадающей с левой частью формулы, на ее правую часть. И левая, и правая части продукций могут быть пустыми: либо выполняется безусловная подстановка правой части, либо удаляется часть исходного слова. Однако поскольку пустое слово присутствует и слева, и справа от каждой буквы преобразуемого слова, то подстановка с пустой левой частью зацикливается, и соответствующий алгоритм неприменим ни к какому входному слову. Если удается применить какую-то формулу подстановки, заменив вхождение ее

левой части в исходном слове на правую часть, происходит возврат в начало алгоритма, и снова ищутся вхождения левой части первой продукции в измененное слово. Если же какую-то продукцию не удалось применить, проверяется следующая за ней, и так далее. Процесс выполнения нормального алгоритма заканчивается в одном из двух случаев: либо все формулы оказались неприменимыми, т. е . в обрабатываемом слове нет вхождений левой части ни одной формулы подстановки; либо только что применилась так называемая терминальная (завершающая) продукция, в которой правую и левую часть разделяет символ 7 →. Терминальных продукций в одном НАМ может быть несколько.

Стоит ещё упомянуть и порядок обработки правил.

1. если применимо несколько правил, то берется правило, которое встречается в описании алгоритма первым;

2. если правило применимо в нескольких местах обрабатываемого слова, то выбирается самое левое из этих мест.

Существует два простых достаточных признака применимости НАМ ко всем входным словам:

1. левые части всех продукций непустые, а в правых частях нет букв, входящих в левые части;

2. в каждом правиле правая часть короче левой.

Необходимость в этих правилах и признаках возникла из-за отсутствия в нормальных алгоритмах Маркова головки, курсора или ещё чего-нибудь, что обозначало бы расположение рабочей ячейки, как в машинах и диаграммах Тьюринга. В результате этого, реализация алгоритмов Маркова очень отличается от Тьюринга даже на уровне идеи, что отразилось и на моей работе. Даже формат вывода отличается (в случае с Марковым он может быть ненормированным, а значит мне не нужно ничего копировать)

Итак, моей задачей было создание нормального алгоритма Маркова, меняющего местами два троичных числа, разделённых знаком “^”.

Как обычно, перед осмотром программы нужно прописать пару тестов.

Входные данные	Выходные данные
3	10
4	11
5	12
6	20
7	21
8	22

3.2. Демонстрация сделанного НАМ

На этот раз алгоритм у меня получился небольшой, поэтому я продемонстрирую его целиком, постепенно объясняя смысл работы каждого блока правил.

Итак вот подробный отчет для задачи по алгоритмам Маркова для перевода числа из 9 системы счисления в 3:

Шаг 1: Первое что нам необходимо и нужно понять, это как именно взаимосвязаны между собой 3 и 9 системы счисления, если мы посмотрим, то все очень просто: есть банальная таблица перевода чисел из 3 в 9 систему счисления, и по ней мы просто переносим все взаимосвязи чисел в алгоритм, для понимания я сейчас приведу эту таблицу:

0	0
1	1
2	2
3	10
4	11
5	12
6	20
7	21
8	22

Шаг 2: Мы поняли взаимосвязь, теперь просто переносим эту таблицу в наш алгоритм, при этом изначально перед числом поставим t, чтобы потом ее переносить вместе с конвертацией чисел

Шаг 3: Алгоритм Маркова так устроен, что то, что нам надо сделать в начале, нужно засунуть в конец, это сделано для того, чтобы программа не зациклилась, и мы смогли перевести нужные нам данные.

Шаг 4: Мы выполнили в принципе все задание, осталось дело за малым, а именно у нас может возникнуть ситуации при которо1 в выходных данных у нас выведется не 0, а 00, или же ситуация когда выведется не значащий 0, к примеру 01 или же 002, такого быть не должно поэтому мы используем k, чтобы избавиться от 0 в начале.

t0->0t

t1->1t

t2->2t

t3->10t

t4->11t

t5->12t

t6->20t

t7->21t

t8->22t

t->.

k00->k0

k01->k1

k02->k2

k03->k3

k04->k4

k05->k5

k06->k6

k07->k7

k08->k8

k->t

->k

Думаю, теперь стоит обратить внимание на сложность данного алгоритма.

3.3. Сложностная оценка сделанного НАМ и выводы

Уже не вижу смысла рассказывать о том, как я буду оценивать сложность алгоритма. Скажу лишь то, что я не буду ставить в числа незначащие нули и пробелы, чтобы обеспечить хоть какую-нибудь точность вычислений.

Влажностная оценка НАМ (нормализованная амплитуда модуляции) и алгоритмы Маркова не имеют прямого отношения к сложностной оценке машин Тьюринга.

Однако, я могу рассказать о недостатках и возможных улучшениях этих концепций.

1. Недостатки влажностной оценки НАМ:
   - Чувствительность к помехам: Влажностная оценка НАМ может быть чувствительной к помехам и шумам, что может привести к искажению данных.
   - Ограниченная применимость: Влажностная оценка НАМ может быть неэффективной для определения влажности в некоторых условиях, например, при наличии конденсации или в условиях экстремальной температуры.

2. Недостатки алгоритмов Маркова:
   - Ограниченность выразительности: Алгоритмы Маркова могут быть недостаточно выразительными для моделирования сложных систем или задач.
   - Сложность разработки: Разработка алгоритмов Маркова для решения определенных задач может быть сложной и требовать глубоких знаний в области теории автоматов и вычислений. Чтобы улучшить эти концепции, можно рассмотреть следующие подходы:

- Для влажностной оценки НАМ: Использование более сложных моделей обработки сигналов, а также учет других параметров окружающей среды, которые могут влиять на точность оценки.
- Для алгоритмов Маркова: Разработка более сложных и гибких моделей, способных учитывать большее количество переменных и условий, а также использование методов машинного обучения для автоматического создания и улучшения алгоритмов Маркова.

ИТОГИ

Итак, в этой курсовой работе я проиллюстрировал определения алгоритма путём построения алгоритмических моделей Тьюринга и Маркова. Как оказалось, это определение каждый раз отличалось.

В случае с машиной Тьюринга алгоритмом называлась совокупность состояний, в которых описаны конкретные действия, переходя через которые головка выполняла определённые операции, приводящие программу к решению поставленной задачи.

В диаграммах Тьюринга вместо состояний головка ориентировалась на последовательность элементарных машин Тьюринга и сделанных программистом подмашин, соединённых между собой определённым образом.

В нормальных алгоритмах Маркова результат работы достигался путём последовательного выполнения правил, которые постепенно преобразовывали входную строку нужным нам образом.

К этим умозаключениям и силлогизмам я пришёл во время выполнения своих задач курсовой работы. Все они были мною успешно, хоть и не ех проблем проделаны.

В конце хотелось бы добавить, что лично для меня более всего удобней и проще было работать в алгоритмах Маркова, думаю, справедливо могу считать это самым удобным и простым способом решения задач.