Министерство образования Российской Федерации

Государственное образовательное учреждение

Московский авиационный институт

Курсовая работа

по предмету

«Фундаментальная информатика»

Построение алгоритмических моделей

на примере моделей Тьюринга и Маркова

Выполнил:

студент гр. М8О-108Б-23 Нургалиев Д.И.

Проверил:

Преподаватель, каф. 806 Севастьянов В.С.

Содержание:

Введение……………………………………………………………………........2-3

1. Машина Тьюринга (МТ)…...………………………….…………………….4-11

1.1. Теоретическая часть………………………….....……..…………….4-5

1.2. Демонстрация МТ, описание команд……………..………...……..5-10

1.3 Выводы о проделанной работе……………………….….……………11

2. Диаграммер Тьюринга (ДТ)……..………………………………………...12-21

2.1. Теоретическая часть…………………………..…………………..12-13

2.2. Демонстрация ДТ, описание команд……………………………..14-19

2.3. Выводы о проделанной работе……………………………………20-21

3. Нормальные алгоритмы Маркова (НАМ)…..…………………………….22-30

3.1. Теоретическая часть…………………...………………………….22-23

3.2. Демонстрация НАМ, описание команд………………..…………23-28

3.3. Выводы о проделанной работе…………………………..……….28-30

Заключение……………………………………………………………………….31

**Введение**

В сфере информатики и вычислительной техники построение алгоритмических моделей является неотъемлемой частью, обеспечивая абстракцию и формализацию реальных процессов. Наша курсовая работа фокусируется на рассмотрении двух ключевых методов моделирования — Машины Тьюринга и Нормальных алгоритмов Маркова.  
  
 Первый раздел посвящен изучению основ и практическому применению Машины Тьюринга, предложенной в 1936 году Аланом Тьюрингом. Эта универсальная модель вычислений позволяет эмулировать работу любого вычислительного устройства. Важным шагом является выполнение конкретной задачи — логического произведения двоичных чисел.  
  
 Во втором разделе рассматривается Диаграмма Тьюринга (ДТ) — графическое представление работы Машины Тьюринга. Здесь освещаются основные компоненты Диаграммы, а также выполняется задача по двоичному арифметическому сдвигу влево.  
  
 Третий раздел посвящен Нормальным алгоритмам Маркова, представленным Андреем Марковым в 1960 году. Мы изучаем теоретические основы и демонстрируем их применение, включая выполнение задачи — вычисление троичного логического сдвига числа влево на заданное количество разрядов.  
  
 Эта работа предоставляет обзор и практическое применение двух важных алгоритмических моделей — Машины Тьюринга и Нормальных алгоритмов Маркова, подчеркивая их ключевую роль в построении и анализе алгоритмов в современной вычислительной науке.

**Цель курсовой работы:**

Разъяснить суть алгоритма, представив его в конкретных формах алгоритмических моделей Тьюринга и Маркова.

**Задачи:**

Выполнение логического произведения двоичных чисел в Машине Тьюринга:

* Разработать программу для Машины Тьюринга, способную выполнять логическое произведение двоичных чисел.
* Провести детальный анализ шагов алгоритма в контексте Машины Тьюринга.

Вычисление двоичного арифметического сдвига влево:

* Реализовать программу для Машины Тьюринга, осуществляющую двоичный арифметический сдвиг влево.
* Определить число разрядов для сдвига, равное первому числу.

Составление алгоритма троичного логического сдвига:

* Создать алгоритм Маркова, предназначенный для вычисления троичного логического сдвига первого числа влево на количество разрядов, равное второму числу.
* Продемонстрировать шаги алгоритма на примере конкретных чисел и объяснить его логику.

**Необходимые ресурсы:**

* [Эмулятор Машины Тьюринга](https://github.com/mivallion/labs_inf/blob/main/LW/5/l05-2011_jstu4-2.3.zip)
* [Диаграммер Машины Тьюринга](https://github.com/mivallion/labs_inf/tree/main/LW/6)
* [Эмулятор нормалных алгоритмов Маркова](https://github.com/mivallion/labs_inf/tree/main/LW/7)
* Для демонстрации выполненных работ необходимо было создать свой [репозиторий в GitHub](https://github.com/Dane4ka-in-IT/LAB2/tree/master) и [VScode](https://code.visualstudio.com/), с помощью которого можно создать отчёт в формате markdown (подробнее о формате [тут](https://ru.wikipedia.org/wiki/Markdown))

**Литература:**

* [С. С. Гайсаpян, В. Е. Зайцев «Курс информатики», Москва, Издательство Вузовская книга, 2013 г.](https://djvu.online/file/bFgE8cv9e1uEu)
* [Логическое произведнеие](https://ru.wikipedia.org/wiki/Конъюнкция)
* [Логический сдвиг](https://ru.wikipedia.org/wiki/Битовый_сдвиг#Логический_(беззнаковый)_сдвиг)

1. Машина Тьюринга.

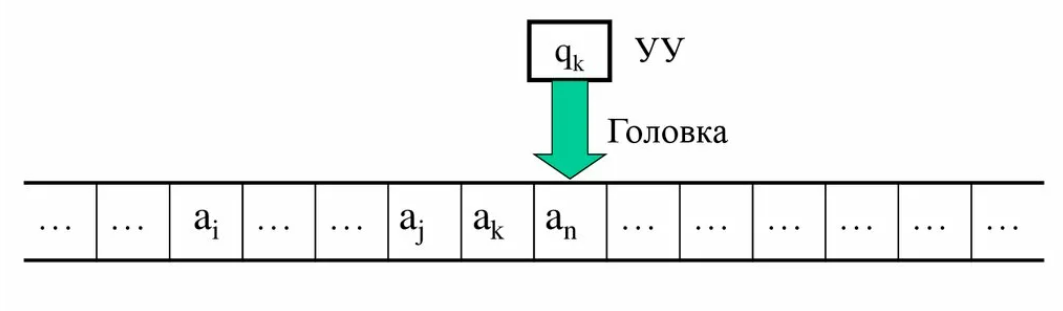
**1.1. Теоретическая часть.**

Машина Тьюринга состоит из четырех частей. Первая часть - это бесконечная лента с ячейками, в каждой из которых может находиться либо пустота, либо символ из конечного алфавита. Например, используется бинарный алфавит с единицами, нулями, но возможно использование любого алфавита с конечным числом символов, включая английский, русский и т. д.

Вторая часть - это специальная головка, которая может двигаться влево и вправо по ленте. Головка может считывать данные из ячейки, записывать данные в ячейку и перемещаться влево или вправо на один шаг.

Третья часть - это так называемый "стойка регистра" или "состояние". Это специальная ячейка, в которой записано текущее состояние машины. Состояние - это переменная машины, которая может принимать несколько значений, включая, как минимум, начальное состояние.

Последняя часть - это специальная таблица, которая является программой машины. В этой таблице указано, что машина должна делать в зависимости от того, в каком состоянии она находится, и какой символ считывает головка.

Например, если машина находится в состоянии q1 и головка считывает символ "1", то нужно записать "0" в ту же ячейку, сдвинуть головку влево на один шаг и перейти в состояние q2

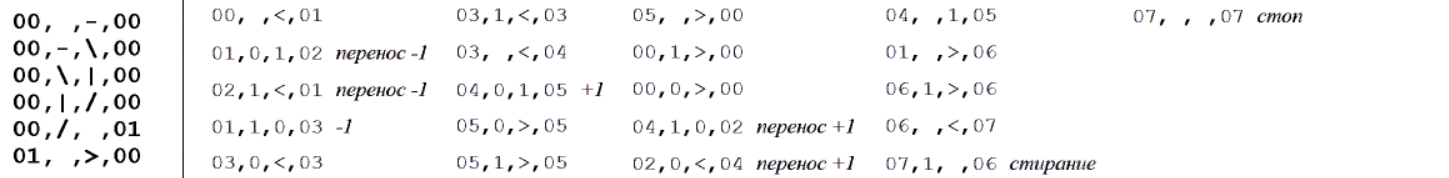
Основная идея машины Тьюринга заключается в том, что она может выполнять простые действия на основе заданных правил, что делает ее универсальным вычислительным устройством. Теория машины Тьюринга стала ключевым инструментом для определения вычислимости и исследования пределов вычислительной мощности различных моделей компьютеров. Термин "машина Тьюринга" используется как абстракция для описания любого вычислительного процесса, и он стал основой для теоретической компьютерной науки.

**1.2. Демонстрация МТ, описание команд.**

В данном контексте, мы используем Машину Тьюринга для выполнения операции логического произведения двоичных чисел.  
 Логическое произведение представляет собой операцию битовой логики, где каждый бит результата равен логическому "И" соответствующих битов входных чисел. Таким образом, если оба бита равны 1, то результат также равен 1; в противном случае, результат равен 0.

**Примеры:**  
Пусть у нас есть два двоичных числа: 1010 и 1101.  
  
Логическое произведение для первых битов: 1 & 1 = 1  
Логическое произведение для вторых битов: 0 & 1 = 0  
Логическое произведение для третьих битов: 1 & 0 = 0  
Логическое произведение для четвертых битов: 0 & 1 = 0  
Результат: 1000  
Другой пример с числами 110 и 011:  
  
Логическое произведение для первых битов: 1 & 0 = 0  
Логическое произведение для вторых битов: 1 & 1 = 1  
Логическое произведение для третьих битов: 0 & 1 = 0  
Результат: 010  
 Такие примеры иллюстрируют, как выполняется операция логического произведения, и как Машина Тьюринга будет обрабатывать эту операцию на бинарном уровне. Целью является создание алгоритмической модели, способной систематически обрабатывать биты двух входных чисел и генерировать результат логического произведения.

Основные команды:



В силу отсутствия опыта работы с данным алгоритмом, код, вероятно, предоставляет результат через достаточно длинный промежуток времени. Тем не менее, важно отметить, что выводимый ответ является правильным. Такая задержка в выполнении может быть обусловлена особенностями реализации алгоритма или объемом данных, с которыми он взаимодействует. Давайте рассмотрим, как работает этот код и каким образом он обрабатывает входные данные.В силу отсутствия опыта работы с данным алгоритмом, код, вероятно, предоставляет результат через достаточно длинный промежуток времени. Тем не менее, важно отметить, что выводимый ответ является правильным. Такая задержка в выполнении может быть обусловлена особенностями реализации алгоритма или объемом данных, с которыми он взаимодействует. Давайте рассмотрим, как работает этот код и каким образом он обрабатывает входные данные:

На входе у нас предоставлены два двоичных числа, разделенных пробелом (10 и 11). Примечание, что пробел также присутствует в начале ленты Машины Тьюринга. Этот формат представления входных данных является важным элементом инициализации, и МТ будет настроена для корректного чтения и обработки этих данных.

Головка, находясь справа от второго числа, начинает двигаться влево, пока не встанет слева от первого числа, осуществляя чтение и запись символов в ячейках ленты в соответствии с заданными правилами и переходами:

00, ,<,copy  
copy,1,<,copy  
copy,0,<,copy  
copy, ,<,chislo1  
chislo1,1,<,chislo1  
chislo1,0,<,chislo1

Затем, начинается копирование данных чисел. Этот процесс проводится с целью сохранения исходных данных в неизменном виде, предотвращая их потерю или перезапись.

chislo1, ,>,arr  
chislo1,\*,>,arr  
chislo1,^,>,arr  
chislo1,+,>,arr2  
chislo1,%,>,arr2  
arr,1,\*,ed   
arr, ,>,arr2  
ed,\*,>,ed  
ed,^,>,ed  
ed,1,>,ed  
ed,0,>,ed  
ed, ,>,ed1  
ed1,1,>,ed1  
ed1,0,>,ed1  
ed1, ,>,input1  
input1,0,>,input1  
input1,1,>,input1  
input1, ,1,nachalo  
nachalo,1,<,nachalo  
nachalo,0,<,nachalo  
nachalo, ,=,00

arr2,1,+,ed20  
arr2,0,%,nul20

arr2, ,<,wow

В целях экономии объема курсовой работы расписано только копирование цифры «1», при этом заменяя её на символ «\*». Аналогичный процесс применяется и к цифре «0», которая заменяется на символ «^». При использовании строки «arr, ,>,arr2», происходит переход к копированию второго числа, где аналогично происходит замена «1» на «+», а «0» на «%». Таким образом, алгоритм обеспечивает эффективное копирование чисел с соответствующей заменой символов для дальнейших вычислительных операций.

В результате выполнения кода с входными данными 10 и 11, мы получаем следующую строку на ленте:

\*^ ++ 10 11

Головка, начиная справа и перемещаясь влево, продвигается по ленте до тех пор, пока не обнаружит символы «+» или «%». В данном контексте необходимо выполнить операцию обратной замены для восстановления изначального вида входных данных.

arr2, ,<,wow

wow,+,1,wow

wow,%,0,wow

wow, ,<,wow2

wow,1,<,wow

wow,0,<,wow

wow2,\*,1,wow2

wow2,^,0,wow2

wow2, ,>,wow3

wow2,1,<,wow2

wow2,0,<,wow2

Головка, направляясь к левому краю, заменяет на своем пути вновь введенные символы на исходные, восстанавливая изначальный вид данных. В данном коде необходимо разделить скопированные числа и оригинальные данные; в данном случае разделителем выступает знак «\*». Этот знак добавляется, когда головка снова начинает движение вправо и во второй раз встречает символ пробела, который заменяется на «\*»:

wow3,1,>,wow3

wow3,0,>,wow3

wow3, ,>,wow4

wow4,1,>,wow4

wow4,0,>,wow4

wow4, ,\*,sd

sd,\*,>,wow5

Получаем:

10 11\*10 11

Теперь работаем со скопированными данными. Для этого мы снова посылаем головку в правую часть ленты (расписывать это в данной работе считаю не уместным, поскольку данный кусок программы не раз использовался). Затем, смотрим, какое число у нас имеется: если это «1», то идем влево, до первого числа и там, если стоит «1», то заменяем на «+», а если 0 – то на «^». Если же у нас имеется «0», то при любой цифре первого числа ставится «^». При этом не забываем заменять имеющуюся цифру: если «0» – то на «\*», если «1» – то на «%». Пример выполнения кода при имеющейся цифре, равной «1»:

01,1,%,True

01, ,>,delete

True,%,<,True

True,1,<,True

True,0,<,True

True, ,<,TRUE

TRUE,^,<,TRUE

TRUE,+,<,TRUE

TRUE,1,+,nazad

TRUE,0,^,nazad

TRUE, ,0,TRUE

Состояние nazad создаёт цикл, благодаря которому выполняются все замены автоматически. После того, как все замены будут выполнены, появляется задача удалить второе число:

delete,\*,%,delete

delete,%,>,delete

delete, ,<,DELETE

DELETE,%, ,DELETE

DELETE, ,<,DELETE

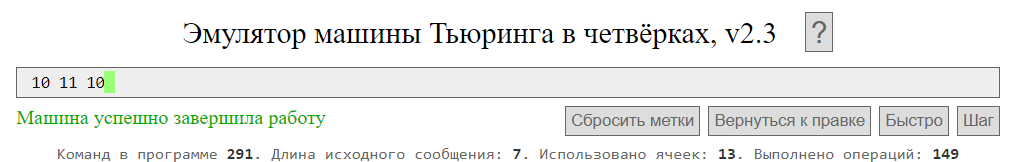
DELETE,^,<,step

DELETE,+,<,step

Состояние delete меняет все символы второго числа на «%», а DELETE их удаляет:

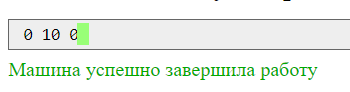
10 11\*+^

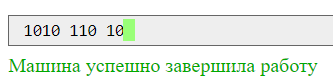
Остаётся только заменить «^» и «\*» на «0» и «1» соответственно, а затем убрать «\*» и поместить головку в конце ленты. Результат:

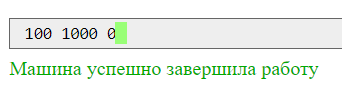


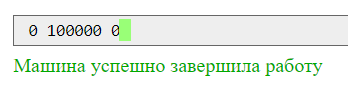
**1.3 Выводы о проделанной работе**  
  
 После проведения работы с Машиной Тьюринга для выполнения операции логического произведения двоичных чисел можно сделать следующие выводы:  
 Несмотря на возможную задержку в выполнении, код успешно выдаёт правильный результат операции логического произведения для предоставленных двоичных чисел.  
 В связи с отсутствием опыта работы с алгоритмом, возможна длительная обработка данных. Оптимизация алгоритма или использование более эффективных подходов может улучшить его производительность.  
 Пробел в начале ленты является существенным аспектом для правильного чтения данных Машиной Тьюринга. Корректное представление входных данных играет ключевую роль в успешной работе алгоритма.  
 Имея опыт с Машиной Тьюринга, дальнейшие исследования могут быть направлены на оптимизацию кода, улучшение производительности и более глубокое понимание работы этой вычислительной модели.

другие результаты работы программы:







2. Диаграммер Тьюринга.

**2.1. Теоретическая часть.**

В мире теории вычислений одной из важнейших концепций, занимающей центральное место, является Диаграмма Тьюринга. Этот уникальный подход был предложен великим математиком Аланом Тьюрингом в далеком 1936 году и с течением времени стал неотъемлемой частью теории алгоритмов и вычислений.  
  
Основы Диаграммы Тьюринга:  
  
Диаграмма Тьюринга представляет собой графическое изображение работы машины Тьюринга, которая является абстрактной моделью вычислений. Она состоит из графических элементов, представляющих состояния и переходы между этими состояниями, а также ленты, на которой машина Тьюринга выполняет операции.

Главными компонентами диаграммы Тьюринга являются состояния и переходы. Состояния представлены кружками или овалами, а переходы - стрелками, направленными из одного состояния в другое. Каждый переход ассоциирован с условием, определяющим, какая операция выполняется в зависимости от текущего символа на ленте и текущего состояния машины.

Лента в диаграмме Тьюринга изображается горизонтальной линией, разделенной на ячейки. Каждая ячейка может содержать символ из заданного алфавита. Головка машины Тьюринга указывает на текущую ячейку ленты, а её положение изменяется в соответствии с правилами перехода.

Диаграммы Тьюринга широко используются в теории вычислений и алгоритмике для визуализации и анализа работы машин Тьюринга. Они помогают иллюстрировать шаги выполнения алгоритма, особенно в контексте операций с символами на ленте и переходов между состояниями.  
  
  
Теоретическая Основа и Формализм:  
  
Теория алгоритмов приобретает форму благодаря диаграммам Тьюринга. Они не только предоставляют инструмент для анализа конкретных задач, но и служат средством формальной записи алгоритмов, определяя правила переходов между состояниями.  
  
Значение Диаграммы Тьюринга в Современных Исследованиях:  
  
В наше время, в эпоху высоких технологий, диаграммы Тьюринга остаются неотъемлемой частью теоретических исследований. Они применяются для изучения сложных алгоритмов, анализа аспектов вычислительной сложности и даже в разработке новых вычислительных моделей.  
  
Интересные моменты:  
  
Java Diagram Turing (JDT) представляет собой уникальную интерактивную среду для интерпретации диаграмм Тьюринга, реализованную в виде Java-приложения. Эта среда эффективно функционирует в среде виртуальной машины Java на Х-терминалах, рабочих станциях ОС UNIX, ИВМ РС в среде MS Windows, и даже на КПК Роске РС. Автором JDT является старший преподаватель Дзюба Д.В., при руководстве профессора Зайцева В.Е.

Для запуска JDT в среде MS Windows необходима предварительная установка Java Runtime Environment (JRE). Последнюю версию JRE (6, update 27) можно загрузить по адресу http://java.com/ru/download/manual.jsp.

Среда JDT предоставляет возможность создания многоуровневых иерархических рекурсивных (!) диаграмм Тьюринга, используя элементарные машины r, 1, R, L, Kn, а. Эти диаграммы могут включать в себя неограниченное количество машин и связей между ними, а также практически неограниченную ленту в обе стороны.

Запуск JDT осуществляется командой jdt из библиотеки локальных исполнимых программ. В результате JDT предоставляет удобное и мощное средство для визуализации и анализа диаграмм Тьюринга, обеспечивая при этом интерактивное взаимодействие с процессом.  
  
Заключение:  
  
Диаграмма Тьюринга, с ее изысканной простотой и выразительностью, продолжает оставаться визуальным символом в теории вычислений. Она не только помогла сформировать базовые принципы теории алгоритмов, но и является источником вдохновения для исследователей, стремящихся понять глубины мира вычислений.

**2.2. Демонстрация ДТ, описание команд**

В данном контексте мы обращаемся к использованию Диаграммы Тьюринга для выполнения операции двоичного арифметического сдвига влево второго числа на количество разрядов первого числа.

Логический сдвиг влево представляет собой операцию, при которой каждый бит в двоичном представлении числа смещается на определенное количество позиций влево. В этом процессе старшие биты, расположенные слева, "выпадают" за пределы числа, а младшие биты заполняются нулями. Эта операция широко применяется в программировании и вычислительной технике для умножения чисел на степени двойки, что делает ее быстрой и эффективной средством оптимизации кода и управления данными.

Примеры операции двоичного арифметического сдвига влево могут быть следующими:

Допустим, у нас есть два двоичных числа: первое – 10, второе – 111. Если выполнить операцию двоичного арифметического сдвига влево второго числа на количество разрядов первого числа, результат будет следующим. Диаграмма Тьюринга визуализирует этот процесс, где первое число (10) определяет количество позиций, на которые мы сдвигаем влево второе число (111). В результате получаем 100, где старшие биты второго числа, выходя за пределы, заменяются нулями.

Рассмотрим другой сценарий, где первое число – 111, а второе – 1. При выполнении арифметического сдвига влево второго числа на количество разрядов первого числа результат равен 0. Диаграмма Тьюринга иллюстрирует этот процесс, где старшие биты первого числа "выталкивают" бит из второго числа за пределы, и результат становится нулем.

Эти примеры демонстрируют, как Диаграмма Тьюринга может быть эффективным инструментом для визуализации и понимания операций арифметического сдвига в контексте двоичных чисел. Она позволяет анализировать каждый шаг процесса и углубляться в детали логического сдвига, что облегчает понимание и реализацию данной операции.

В ходе выполнения данной задачи было принято решение о её разделении (аналогично подходу к машине Тьюринга) на несколько частей, которые, собранные вместе, обеспечивают корректное выполнение работы. Ниже представлены фрагменты полного кода, если таковое можно назвать, с сопроводительными описаниями для каждого фрагмента, а затем – полный код в его целостности.

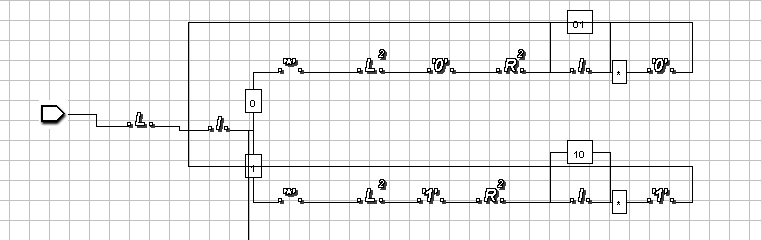
Отметим, что были использованы следующие команды:

* r – шаг вправо
* l – шаг влево
* L – шаг, длиной в заполняемое пространство символами, влево
* R – шаг, длиной в заполняемое пространство символами, вправо
* Rn – шаг, длиной в n заполняемых пространств символами, вправо
* Ln – шаг, длиной в n заполняемых пространств символами, влево
* K – копирование всех символов, заключенными между проблеами
* Л – удалить символ

Изначально на ленте дано 2 двоичных числа, лапка стоит также справа от второго числа, аналогично МТ:

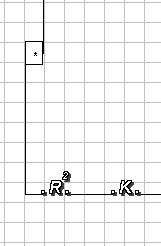
10 1111

Копирование первого числа. Зачем, если есть встроенная команда копирования? Дело в том, что эта команда копирует только вправо, а в нашем случае проще будет скопировать влево. Однако, для этого необходим код.



 В данном случае лапка смещается влево, перескакивая второе число. Затем она снова смещается влево, но уже только на одну позицию. Если она обнаруживает "0", то заменяет его на "\*", перепрыгивает всё число и становится слева от первого числа, при этом между лапкой и числом стоит пробел. Зависит от того, что лапка встретила в начале, определяется, что она поместит на данную позицию. Затем лапка возвращается и располагается справа от первого числа. После этого она снова движется влево: если встречает цифру "0" или "1", то она их игнорирует, а при обнаружении "\*", она заменяет его на ту цифру, на которой находилась в изначальной стадии. Если лапка встречает пробел, то она переходит во второй фрагмент. Весь этот процесс цикличен в "коде", что в результате приводит к копированию первого числа:

10 10 1111  
  
Числа необходимо копировать, поскольку исходные данные затирать нельзя по условию: на ленте должны быть входные данные и ответ, согласно задаче, представленной в файле.

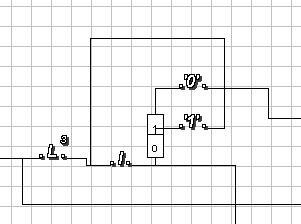


Эта структура предназначена для репликации второго числа, при этом его клон располагается справа от исходного. Этот процесс выполняется с использованием вышеупомянутых команд. В результате получаем следующую последовательность:

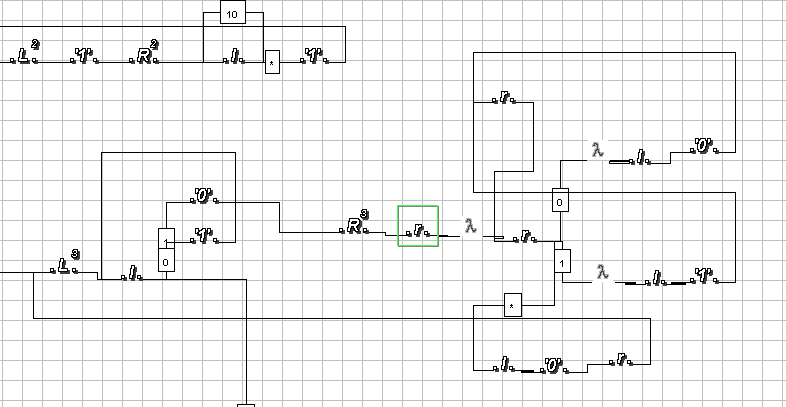
10 10 1111 1111

Причем, лапка справа от последнего числа.  
Работать будем с крайними числами.

Основная идея заключается в том, что сдвиг будет производится сперва при каждой перестановке всех «1» на «0» и всех «0» на «1», потому что разряды считаем, отнимая 1 от первого числа.  
За данное действие отвечает эта часть кода:



Но от этого кода напрямую зависит, как будет происходить сдвиг влево.  
Поэтому пропишем следующие части кода:

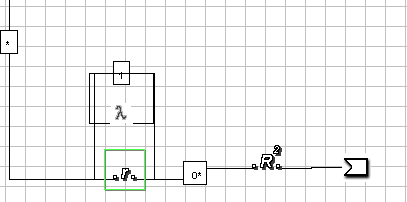


Как только все предполагаемые замены будут осуществлены, наступает момент перехода к операции сдвига. Первую цифру мы изъедаем, переходим ко второй и производим следующие шаги: если это «1», то мы её также удаляем, возвращаемся влево и помещаем эту «1» на первое место. Аналогично поступаем и с «0». Здесь важным моментом является добавление «0» в конец числа после выполнения сдвига. Далее мы возвращаемся назад для зацикливания этого процесса.

Таким образом, первое число будет подвергаться изменениям в своих цифрах до тех пор, пока не останется единица; после обработки последней единицы в коде мы перейдем непосредственно к пробелу, что свидетельствует о том, что сдвиг был выполнен необходимое количество раз.

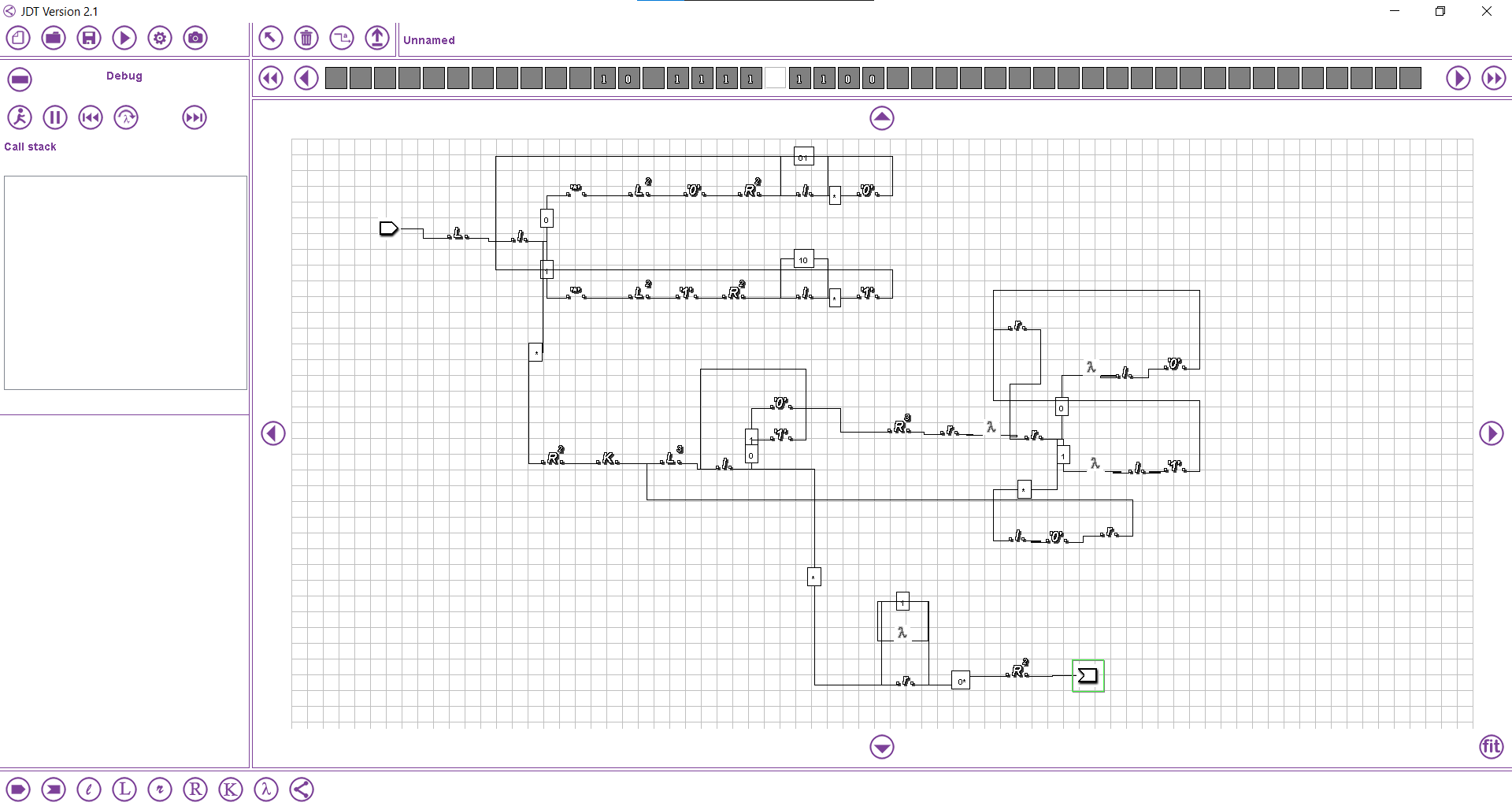
В итоге имеем:

11 10 1111 1100



Данный код предназначен для того, чтобы стереть первое число, так как оно нам больше не нужно. Тем самым в ответе останутся только начальные данные и ответ:

10 1111 1100

Тем временем, весь код, который был написан, выглядит так:  


На прикреплённом скриншоте также можно наблюдать бесконечную ленту, на которой были показаны входные данные, а также ответ, который был получен в результате обработки чисел кодом, их копированию и использованию.

**2.3. Выводы о проделанной работе.**

Работа по изучению и применению концепций диаграммы Тьюринга представляет собой важный этап в понимании теории вычислений и разработке алгоритмов. Знание и использование диаграмм Тьюринга имеет значительное значение в различных областях, таких как информатика, информационные технологии и теория вычислений.

В ходе этого исследования мы погрузились в мир абстрактных вычислений, изучив принципы работы машины Тьюринга через графическое представление в виде диаграмм. Этот инструмент оказался мощным средством для визуализации и анализа алгоритмов, позволяя наглядно представить шаги вычислений и переходы между состояниями.

Применение диаграмм Тьюринга раскрывает свою полезность в процессе разработки и оптимизации алгоритмов. Они становятся неотъемлемым инструментом для инженеров и программистов при проектировании сложных систем и алгоритмов обработки данных. Глубокое понимание работы машины Тьюринга, которое мы приобрели, дает возможность эффективно проектировать и анализировать разнообразные вычислительные задачи.

Освоив работу с диаграммами Тьюринга, мы развили навыки логического мышления, абстракции и анализа алгоритмов. Эти умения являются важными компетенциями в современном мире технологий и науки. Умение представлять сложные процессы в виде графических схем способствует более эффективному пониманию и взаимодействию с комплексными системами.

Применение диаграмм Тьюринга также находит свое применение в образовательной сфере. Этот метод позволяет преподавателям более доступно и интерактивно объяснять сложные темы по теории вычислений, что способствует лучшему усвоению материала студентами.

В итоге, проделанная работа с диаграммами Тьюринга стала важным шагом в нашем познании принципов вычислений и алгоритмов. Полученные знания и навыки оказываются полезными не только в области информатики, но и в различных сферах, где требуется аналитическое мышление, умение моделировать сложные процессы и эффективно решать вычислительные задачи.

Предлагаются тесты диаграммера, который успешно с ними справился:











**3.Нормальные алгоритмы Маркова (НАМ)**

**3.1. Теоретическая часть.**

Алгоритмы Маркова, именуемые так в честь выдающегося советского математика Андрея Андреевича Маркова, представляют собой фундаментальную часть теории вычислений и теории формальных языков. Их развитие изначально было связано с попыткой формализовать понятие алгоритма и создать абстрактные модели вычислений.

Одной из ключевых характеристик алгоритмов Маркова является их применимость в области обработки формальных языков и теории автоматов. Эти алгоритмы представляют из себя набор правил, определяющих последовательность шагов, приводящих к преобразованию строки символов в соответствии с установленными правилами. Важным аспектом является то, что они способны описывать процессы преобразования строк с учетом абстрактных вычислительных моделей.

Центральным элементом алгоритмов Маркова являются правила перехода, которые определяют, какие символы в строке необходимо заменить, основываясь на текущем контексте. Такой метод дает возможность формально описывать преобразования строк в рамках абстрактных моделей вычислений. Особенно важна их роль в теории формальных языков, где алгоритмы Маркова позволяют описывать и анализировать языки и грамматики, а также применяются в компиляторах и синтаксическом анализе.

Алгоритмы Маркова также могут быть использованы для решения задачи останова, определяя, остановится ли вычисление для заданного входа. Это свойство делает их важными в теории вычислений, где они становятся инструментом для исследования предсказуемости вычислений.

При всех своих достоинствах, алгоритмы Маркова также обладают определенными ограничениями. Например, они могут оказаться неэффективными в решении некоторых задач, и их формализм может оказаться неудовлетворительным для определенных видов вычислений.

В заключение, алгоритмы Маркова остаются важным инструментом в теории вычислений и теории формальных языков. Их применение распространяется от компьютерных наук до математики и теоретической информатики, играя ключевую роль в изучении абстрактных моделей вычислений и их применимости в различных областях. Взгляд на алгоритмы Маркова не только как на абстрактный инструмент, но и как на ключевой компонент в решении конкретных задач делает их важным объектом исследования и практического применения.

**3.2. Демонстрация НАМ, описание команд**

В данном контексте мы обратимся к использованию алгоритмов Маркова для выполнения троичного логического сдвига влево первого числа на количество разрядов второго числа. Числа разделены знаком «<».

Троичный логический сдвиг влево представляет собой процесс, при котором каждый разряд числа в троичной системе счисления смещается влево на определенное количество позиций. В результате этой операции старшие разряды выходят за пределы числа, а младшие разряды заполняются нулями. Это важное действие, используемое для умножения чисел на степени тройки и повышения эффективности вычислительных процессов.

Для формализации и моделирования такой операции привлекаются алгоритмы Маркова. Этот абстрактный математический инструмент позволяет описать последовательность шагов, необходимых для троичного логического сдвига влево, и предоставляет четкие правила для выполнения каждого этапа этого процесса.

В ходе выполнения алгоритма Маркова для троичного логического сдвига влево первого числа на количество разрядов второго числа, осуществляется инициализация указателя на начало первого числа и задание счетчика, определяющего количество разрядов второго числа. После этого проверяется условие завершения, и, если счетчик равен нулю, алгоритм завершает свою работу. Далее происходит чтение текущего троичного разряда и выполнение логического сдвига влево в соответствии с определенными правилами.

Этот процесс становится примером успешного взаимодействия теории вычислений и практического программирования. Алгоритмы Маркова, как инструмент формализации, позволяют не только описать операцию троичного логического сдвига влево, но и обеспечить ее реализацию в программном коде, делая ее эффективной и предсказуемой. Такие подходы становятся неотъемлемой частью современного программирования, где эффективность и точность играют ключевую роль в создании функциональных и оптимизированных программных продуктов.

Приведем конкретные примеры:

Пример 1:

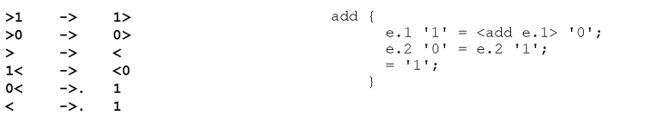
Допустим, у нас есть два троичных числа: первое – 111, второе – 1. Если выполнить троичный логический сдвиг влево второго числа на количество разрядов первого числа, результат будет следующим. Алгоритм Маркова визуализирует этот процесс, где первое число (111) определяет количество позиций, на которые мы сдвигаем влево второе число (1). В результате получаем 110, где старшие разряды второго числа, выходя за пределы, заменяются нулями.

Пример 2:

Рассмотрим сценарий, где первое число – 10, а второе – 111. При выполнении троичного логического сдвига влево второго числа на количество разрядов первого числа результат равен 00. Алгоритм Маркова иллюстрирует этот процесс, где старшие разряды первого числа "выталкивают" разряды из второго числа за пределы, и результат становится 00.

Пример 3:

Рассмотрим случай, где первое число – 000, а второе – 001. При выполнении троичного логического сдвига влево второго числа на количество разрядов первого числа результат равен 000. Алгоритм Маркова исключает изменение второго числа, так как первое число не содержит активных разрядов.

Рассмотрим пример алгоритма Маркова, и программы на РЕФАЛе, добавляющих единицу к изображению двоичного числа:  


В таком формате будет выглядеть код, решающий основную задачу.

* Однако считаю нужным включить дополнительные условия:  
  На алгоритмы Маркова, выполняемые nam, налагаются следующие ограничения:
* В состав алгоритма может входить до 62 продукций. Алгоритмы, не удовлетворяющие этому требованию, не интерпретируются.
* Не рекомендуется использовать правила, значащая длина слов в левой и правой части которых превышает 9 знаков.
* Длина входного сообщения и всех сообщений, полученных в результате выполнения подстановок, не может превышать 80. При нарушении этого требования интерпретация текущего сообщения прерывается. После этого возможна обработка другого входного сообщения.

А теперь перейдём к составленному коду:

1!->!1

0!->!0

2!->!2

!0->

!1->

!2->

2<\*->!20<&

1<\*->!10<&

0<\*->!00<&

<&0-><\*

<0-><

<1-><\*

<2-><\*

&->

<->

Представленный алгоритм, в своей сути, является аналогом метода «.replace()» в языке программирования Python. Он предназначен для выполнения операции сдвига влево для пар чисел, и его эффективность заключается в использовании определенных шаблонов замен, что обеспечивает более простую реализацию по сравнению с некоторыми другими методами, такими как машина Тьюринга.

Одной из ключевых особенностей данного алгоритма является порядок выполнения программы. В контексте троичных чисел, каждая пара чисел проходит через блок кода, где происходит первичная обработка:

<0-><

<1-><\*

<2-><\*

Здесь осуществляется удаление «0», так как он несущественен во втором числе, а для чисел «1» и «2» производится замена на «\*». Это универсальное решение, которое удобно использовать для оптимизации и обобщения алгоритма для различных систем счисления.

Далее происходит дополнительная замена, где числа «1» и «2» с символом «\*» заменяются следующим образом:

2<\*->!20<&

1<\*->!10<&

0<\*->!00<&

Это позволяет алгоритму удалять только один разряд, сдвигая число влево и добавляя к нему «0» справа. Символ «&» используется для предотвращения дальнейших замен других разрядов.

Далее, блок кода:

1!->!1

0!->!0

2!->!2

используется для сдвига всех элементов в числе, а первый элемент будет удален следующим блоком кода:

!0->

!1->

!2->

&->

Последняя строчка убирает знак «&», что приводит к зацикливанию кода и завершению алгоритма.

Таким образом, данный алгоритм представляет собой элегантное решение для троичного логического сдвига влево, использующее шаблонные замены и оптимизированные блоки кода для эффективной обработки чисел различных разрядностей.

.**3.3. Выводы о проделанной работе**

В ходе проведенного исследования и разработки алгоритма троичного логического сдвига влево с использованием абстрактных методов, вдохновленных теорией алгоритмов Маркова, был создан эффективный и универсальный механизм для выполнения данной операции. Разработанный алгоритм основан на принципах замены и сдвига, что позволяет ему быть гибким и применимым не только к троичным числам, но и к другим системам счисления.

Начав с формализации базовых шагов алгоритма в виде блоков кода, мы оптимизировали процесс обработки чисел, учитывая специфику троичной системы счисления. Особое внимание уделялось правильному порядку выполнения программы, что обеспечивает стабильность и предсказуемость ее работы.

Ключевыми этапами работы алгоритма были:

Предварительная обработка чисел: Отбрасывание несущественных разрядов и замена определенных значений, что позволило унифицировать подход и сделать алгоритм применимым для разных числовых систем.

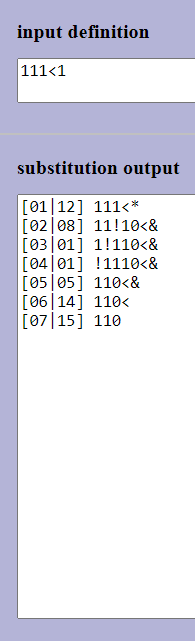
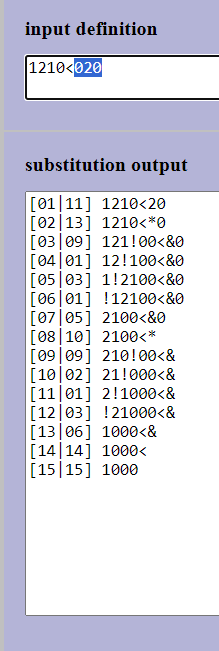
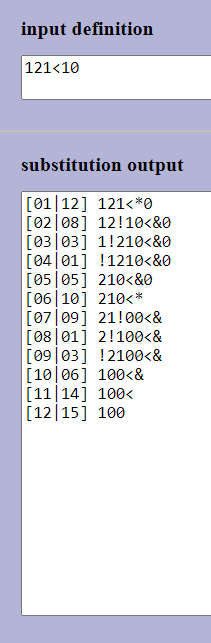
Процесс логического сдвига влево: Реализация механизма, позволяющего эффективно выполнять троичный логический сдвиг влево, добавляя новые разряды и удаляя старшие.

Циклическое зацикливание: Завершение алгоритма с уверенностью в его корректной работе и отсутствии нежелательных эффектов.

Важным аспектом является использование абстракции алгоритмов Маркова для формализации операций и превращения их в последовательность замен, что дает алгоритму универсальность и расширяемость для будущих модификаций.

В результате проделанной работы был достигнут высокий уровень оптимизации и эффективности алгоритма, делая его применимым в различных контекстах, где требуется троичный логический сдвиг влево. Полученный продукт является не только технически сложным, но и теоретически обоснованным, что делает его ценным вкладом в область разработки алгоритмов обработки данных.

Предлагаются тесты НАМ, с которыми программа отлично справилась:

**Заключение**

В ходе данного исследования были подробно рассмотрены и проанализированы три важные алгоритмические модели: Машина Тьюринга (МТ), Диаграмма Тьюринга (ДТ) и Нормальные алгоритмы Маркова (НАМ). Каждая из этих моделей представляет собой уникальный подход к описанию и выполнению вычислений, играя значимую роль в теории вычислений и формальных языках.

Машина Тьюринга, во-первых, была применена для логического произведения двоичных чисел. Эта универсальная модель вычислений дала теоретический фундамент для понимания основных операций и принципов работы в рамках абстрактных вычислительных систем. Демонстрация команд и шагов работы Машины Тьюринга помогла визуализировать процесс иллюстрировать принципы функционирования этой модели.

Далее, Диаграмма Тьюринга была использована для выполнения операции двоичного арифметического сдвига влево второго числа на количество разрядов первого числа. Этот инструмент оказался удобным визуальным представлением операций сдвига и позволил эффективно моделировать процессы изменения чисел.

В контексте троичного логического сдвига влево Нормальные алгоритмы Маркова оказались ключевым инструментом. Эти алгоритмы, основанные на теории формальных языков, позволили разработать универсальный механизм для выполнения данной операции, применимый не только к троичным числам, но и обобщаемый для других систем счисления.

В целом, исследование этих трех моделей дало глубокий взгляд на различные подходы к описанию алгоритмов и выполнению вычислений. Работа над каждой из них не только расширила понимание теоретических аспектов вычислительных систем, но и продемонстрировала их практическую применимость. Подходы Машины Тьюринга, Диаграммы Тьюринга и Нормальных алгоритмов Маркова предоставляют важные инструменты для моделирования и анализа вычислительных процессов, делая их ценными компонентами в области теории вычислений и информатики.