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 **Григорьев Т.А.**

Проверил:

Преподаватель, каф. 806 **Севастьянов В.С.**

Москва 2023

Содержание:

Введение.

Алгоритмы и алгоритмические модели представляют собой важную часть информатики и компьютерных наук. Алгоритм можно определить как точно заданную последовательность правил, которая указывает, каким образом можно получить выходное сообщение определенного вида из заданного исходного сообщения за конечное число шагов. Однако, это определение не является формальным и требует дополнительной трактовки с использованием алгоритмических моделей.

Алгоритмические модели представляют собой различные способы формализации алгоритмов. Среди них выделяют рекурсивные функции, машины Тьюринга и нормальные алгоритмы Маркова. Рекурсивные функции связывают понятие алгоритма с вычислениями и числовыми функциями, машины Тьюринга описывают алгоритм как процесс работы машины, способной выполнять простые операции, а нормальные алгоритмы Маркова описывают алгоритмы как преобразования слов в произвольных алфавитах.

В моей курсовой работе будут использоваться машины Тьюринга и нормальные алгоритмы Маркова для решения определенных задач, а также будут использованы диаграммы Тьюринга.. Эти модели предоставляют удобные инструменты для изучения и анализа алгоритмов.

Таким образом, введение в тему алгоритмов и алгоритмических моделей позволяет лучше понять цели и задачи моей курсовой работы.

Цель: проиллюстрировать определения алгоритма путем построения алгоритмических моделей Тьюринга и Маркова.

Задачи: (взял цели из лабораторных работ)

1) составить программу машины Тьюринга в четверках, выполняющую заданное действие над словами, записанными на ленте;
2) разработать диаграмму Тьюринга решения задачи с использованием стандартных машин (r, l, R, L, K, a) и вспомогательных машин, определяемых задачей.
3) разработать нормальный алгоритм Маркова для вычисления двоичного логического сдвига вправо на число разрядов, равное первому числу
4) обобщить полученную информацию и сделать соответствующие выводы о каждом способе, провести оценку каждого алгоритма.

Для выполнения поставленных задач мне необходимы:

- эмулятор машины Тьюринга в четверках
- диаграммер для работы с диаграммами Тьюринга
- эмулятор для нормальных алгоритмов Маркова
- текстовый редактор Microsoft Word
- мой репозиторий на GitHub, куда были выложены все описанные ниже алгоритмы(https://github.com/timofeez/inf_labs)

В качестве списка литературы могу представить лишь один пункт:

- 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.

На этом моё вводная часть курсовой работы подходит к концу. Пришло время приступить к рассмотрению самих алгоритмических моделей.

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).

Данное определение очень сложное и может быть непонятно тому, кто впервые столкнулся с такой информацией, но поверьте, это до тех пор пока вы не проникнетесь пятой лабораторной работой.. В переводе на русский язык можно сказать, что каждая четверка состоит из:

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 изменить состояние (смысла в такой команде нет, поэтому ее не упоминают).
4) состояния, в которое должна перейти головка.

Небольшая вставка: если в четверке не описано какое-нибудь действие (команда или смена состояние), то это приведет 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 «четверках» 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. Ее задача заключалась в Умножение однозначных чисел в усеченной римской системе счисления.

Сначала я постараюсь описать основополагающую информацию для выполнения данной лабораторной работы.

Примером такой информации является таблица перевода цифр из римской системы в десятичную.

Римская система счисления	Десятичная система счисления
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

Не стоит забывать и про нормированность. Ввиду простоты поставленной задачи с этим проблем не возникло.

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

Входное сообщение	Выходное сообщение
I V	V
X X	C
V V	XXV
C V	D
L V	CCL
L L	MMD
L X	D

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

00, ,<,00 (< означает сдвиг головки на ячейку влево)

Под 00 подразумевается начальное состояние головки машины Тьюринга. Команда 00 переходит сама в себя, то есть передвигает головку

влево до первого непробельного символа. В этом случае она делает шаг влево (от правого числа до пробела между левым и правым) и переходит во множество состояний, определяющих правое число вида 00_X, где X - правое число в десятичной системе счисления:

00,I,<,0_1

00,V,<,0_5

00,X,<,0_10

00,L,<,0_50

00,C,<,0_100

00,D,<,0_500

00,M,<,0_1000

Далее, для каждого из правых чисел, пройдя через пробел между двумя операндами, определяется левое число так, что осуществляется переход в состояние semi_right_X_Y, где X - левое число в десятичной системе счисление, Y - правое.

// определение левого числа для правого числа = I

0_1, ,<,0_1

0_1,I,>,semi_right_1_1

0_1,V,>,semi_right_5_1

0_1,X,>,semi_right_10_1

0_1,L,>,semi_right_50_1

0_1,C,>,semi_right_100_1

0_1,D,>,semi_right_500_1

0_1,M,>,semi_right_1000_1

и так далее… для каждого возможного правого числа

semi_right_X_Y, в свою очередь, возвращает каретку обратно к правому числу и производит переход в состояние right_X_Y:

semi_right_1_1, ,>,semi_right_1_1

semi_right_1_1,I,>,right_1_1

right_1_1, ,>,result_1_1

и так далее…

Состояние result уже предназначено непосредственно для вывода ответа. В случае ответа, занимающего больше одной ячейки производит переход в специальные состояние long_result_X_Y_pZ (Z - порядковый номер такого состояния, очень удобно использовать нумерацию для вывода больших ответов):

// вывод результата и переход к финальному состоянию

result_1_1, ,I,final

result_5_1, ,V,final

result_10_1, ,X,final

result_50_1, ,L,final

result_100_1, ,C,final

result_500_1, ,D,final

result_1000_1, ,M,final

result_5_5, ,X,long_result_5_5

result_10_5, ,L,final

result_50_5, ,C,long_result_50_5

result_100_5, ,D,final

result_500_5, ,M,long_result_500_5

result_1000_5, ,M,long_result_1000_5

result_10_10, ,C,final

result_50_10, ,D,final

result_100_10, ,M,final

result_1000_10, ,M,long_result_1000_10

result_500_50, ,M,long_result_500_50

result_1000_50, ,M,long_result_1000_50

Далее идёт переход к конечному состояние в лице команды final и завершение работы программы. (# означает переход к конечному состоянию)

final,I,>,final

final,V,>,final

final,X,>,final

final,L,>,final

final,C,>,final

final,D,>,final

final,M,>,final

final, ,#,final

1.3. Сложностная оценка и выводы к машине Тьюринга

Для проведения сложностной оценки здесь и далее я буду предоставлять таблицы с входными сообщениями разной длины и временем выполнения программы. На их основе сложность алгоритма можно будет определить, не углубляясь далеко в дебри матанализа. Все вычисления времени будут примерными, так как пользоваться я буду обычным секундомером и округлять значения до целого. Такая погрешность компенсируется обильностью тестов.

Входное сообщение	Время выполнения программы, с
I V	1
X D	2,5
M L	20

Как можно убедится из примеров, при увеличении длины выходного слова в 2 раза время выполнения возрастает больше чем в 2, но меньше чем в 4 раза, из чего можно сделать вывод, что сложность алгоритма составляет O(n*log(n)). Учитывая специфику программ на машине Тьюринга, сделать сложность линейной для моей задачи нельзя, равно, как и логарифмическую сложность.

Что касается выбранного мною способа решения, то для меня он оказался наиболее простым в исполнении. Принципиально более продвинутых методов решения даже сам Зайцев В.Е. (с логарифмической сложностью) предлагать не стал, так как посчитал это слишком сложным для нас. Да их, собственно, и нет. Можно было только изменить состояния для перемещения головки, что ничего бы толком не изменило. На этом работа над машиной Тьюринга завершена

2. Диаграмма Тьюринга

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

Стандартные машины Тьюринга имеют ряд существенных недостатков. Из тех, что я обнаружил при выполнение лабораторной работы, могу 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. Её задачей являлось вычисление двоичного арифметического сдвига первого числа вправо на число разрядов, равное второму числу.

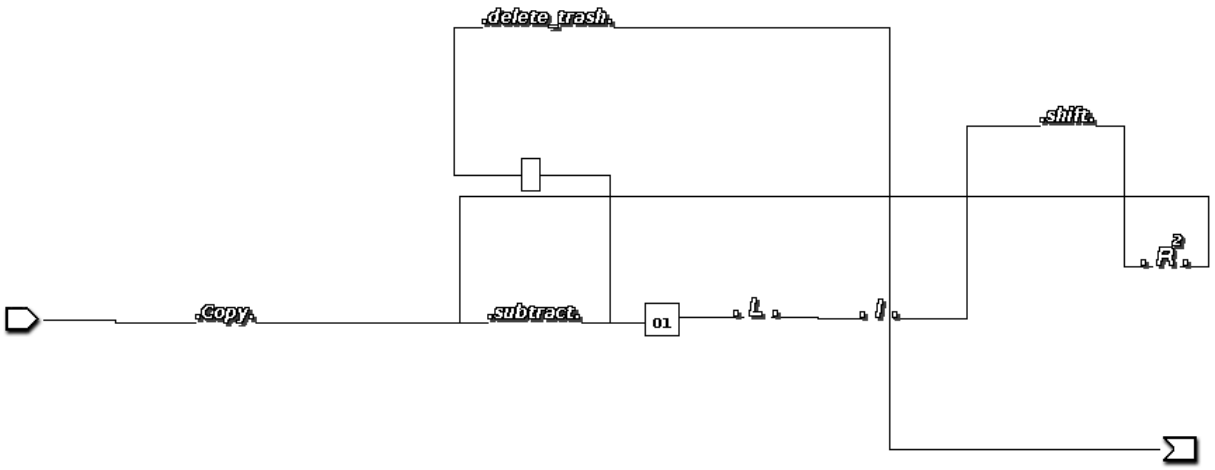
А пока стоит ещё остановиться на тестах.

Входные данные	Выходные данные
1010 10	1010
10111 11	10
0 000	0

2.2. Диаграммы

Диаграмма Тьюринга в моем исполнении состоит из 5 составных частей, то есть 4 подмашин и главной машины. Для удобства, а также для прямой, простой доступности я буду последовательно пояснять каждую диаграмму, одновременно склеивая все части в одну целую картину.

Итак, начнем с самого начала. Мы хотя бы можем увидеть, как мы начинаем работу программы с левой стороны, перемещаясь вправо, к окончанию программы. Первая подмашина **copy**, обеспечивает копирование слова, это необходимое условия для построения диаграмм Тьюринга, к счастью в java версии копирование происходит намного проще чем в windows версии, далее идет машина **subtract,** а также машина **delete_trash** и **shift,** после чего работа машины завершается, теперь перейдем непосредственно к каждой подмашине.

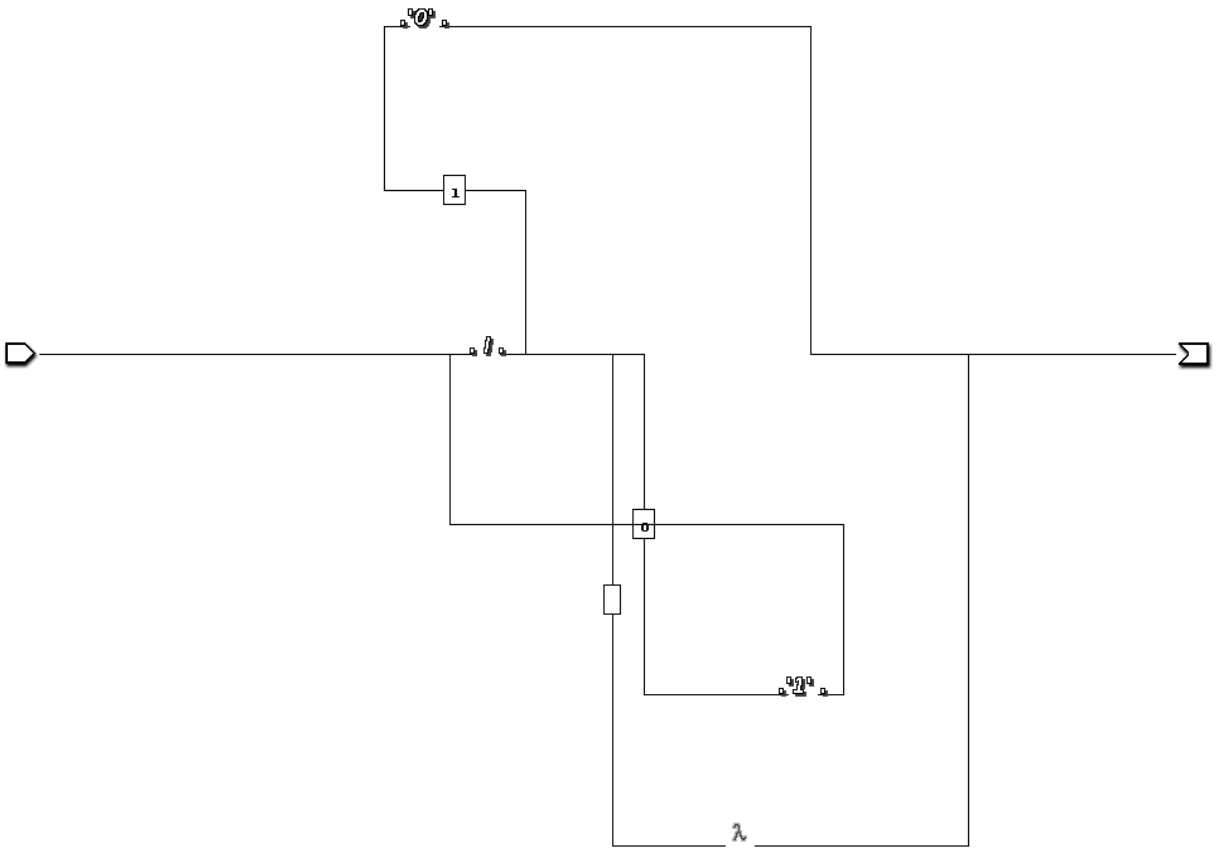


Подмашина **copy.**

Подмашина copy является самой простой подмашиной в моей диаграмме, она состоит из двух пар последовательных элементов.

Начинается все с элемента L, он переносит нас к ближайшему левому слову, до его конца, происходит копирование этого слова, далее действия повторяются со вторым словом, после проделанных элементарных действий, процесс завершается, можно переходить к следующим схемам.

Подмашина **subtract**



Вычитание в двоичной системе счисление просиходит не так просто как может показаться на первый взгляд, на этой теме стоит чуть остановиться и рассказать о ней поподробней.

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

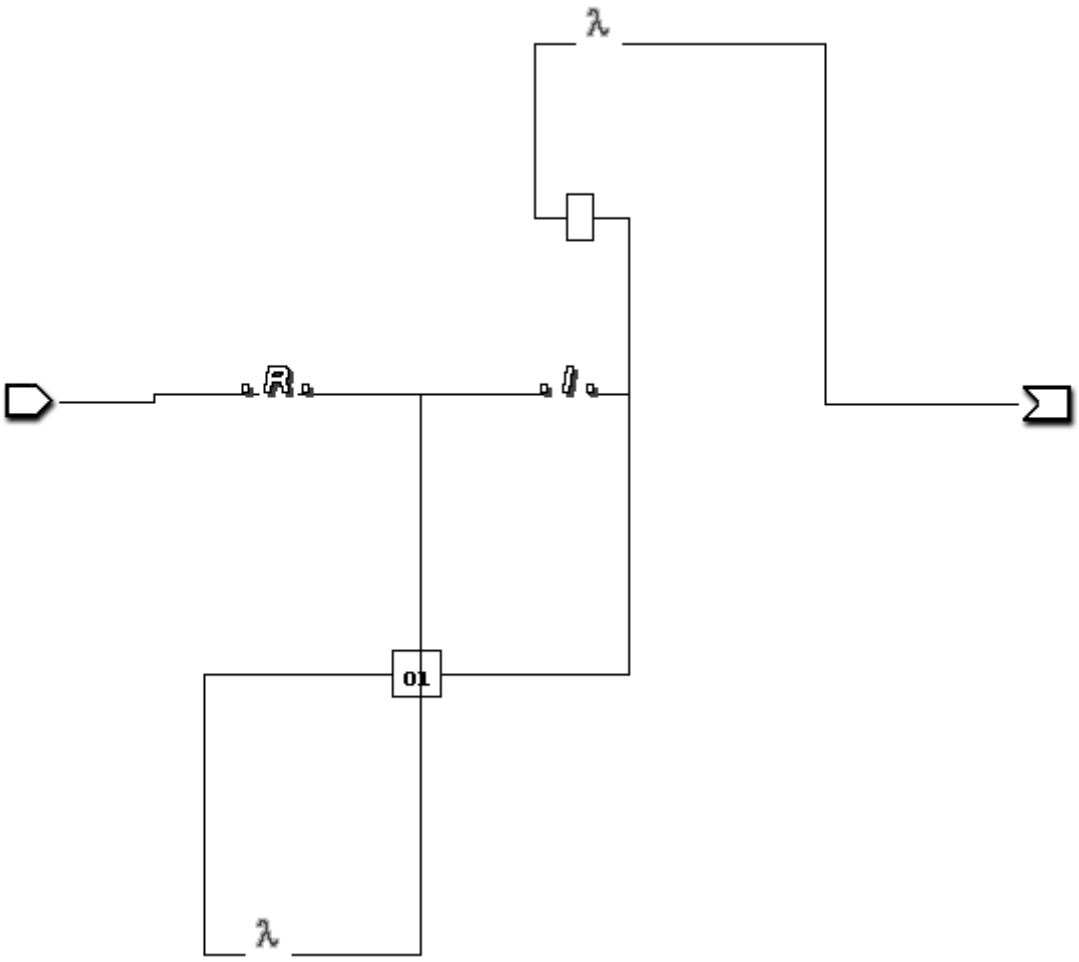
Что такое логический сдвиг? Логический сдвиг – это операция, при которой все биты в битовом значении сдвигаются на определенное количество позиций влево или вправо.

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. Если мы будем видеть 0, то заменим его на единицу, если же мы будем видеть единицу, то мы просто заменим ее на ноль; процесс вычитания завершен.

Подмашина **delete_trash**



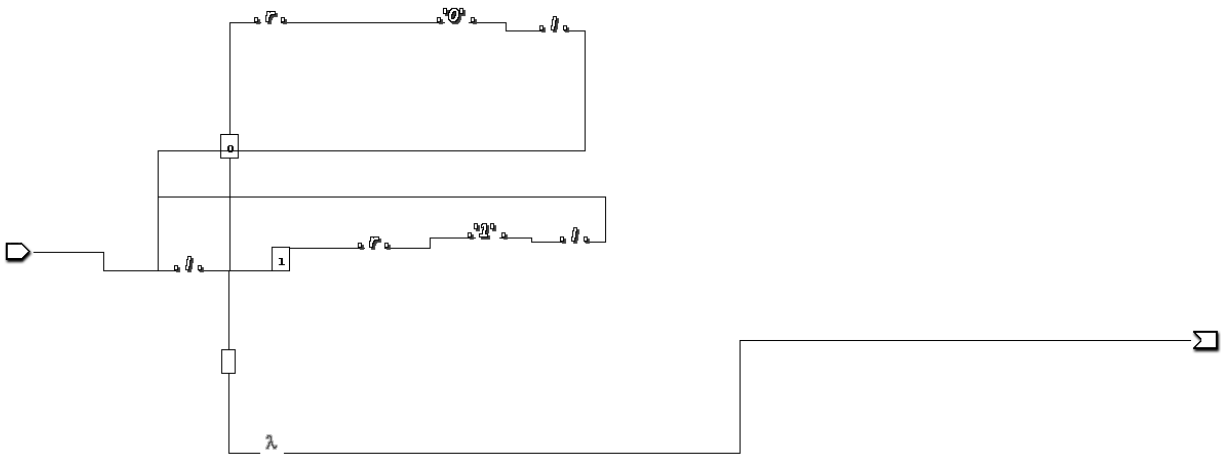
Данная подмашина имеет простую функцию, удаляет ненужные символы, затирая их на ленте, состоит она из нескольких этапов.

Первый символ, R, перемещает нас в конец правого слова, далее перемещаемся на один символ влево, и видя какой то символ, затираем его, ситуация противоположна с значимыми элементами.

Почему кстати 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 дополнить до вида: 3n + 1, где n – натуральное число. 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. Диаграмма **shift.**



В этой диаграмме реализуется сам логический сдвиг, который и есть одна из главных частей в этой задаче.

Первый символ, который нас встречает l, который перемещает головку на ячейку влево, далее идет ветвление на три части, если мы встречаем пустую ячейку, то мы просто ставим лямду, и завершаем программу

Далее рассмотрим ветку диаграммы, где будет 0, там мы перемещаемся вправо на один символ, вставляем 0, после влево, и связываем это с первым символом программы, создавая цикл.

После рассмотрим ветку диаграммы, где будет 1, там мы перемещаемся вправо на один символ, после ставим 1, и после влево.

На этом действие этой подмашины заканчивается.

2.6. Критика диаграммы Тьюринга и выводы

Диаграммы Тьюринга, как и любой другой метод визуализации алгоритмов, имеют свои преимущества и недостатки. Вот несколько аспектов, которые можно рассмотреть при критике диаграмм Тьюринга:

1. Преимущества:

- Визуальное представление: Диаграммы Тьюринга позволяют визуально представить шаги выполнения алгоритма, что может помочь в понимании его работы.

- Универсальность: Машины Тьюринга могут быть использованы для моделирования широкого спектра вычислений, что делает их универсальным инструментом.

- Абстракция: Диаграммы Тьюринга позволяют абстрагироваться от конкретных деталей аппаратного обеспечения и сосредоточиться на логической структуре алгоритма.

2. Недостатки:

- Сложность: Диаграммы Тьюринга могут быть сложными для понимания, особенно для сложных алгоритмов. Некоторые диаграммы могут быть запутанными и труднопонимаемыми.

- Ограничения визуализации: Некоторые аспекты работы машины Тьюринга могут быть трудно или невозможно визуализировать на диаграммах, особенно если они связаны с большим объемом данных или сложными условиями.

- Ограничения модели: Модель машины Тьюринга не всегда может точно отразить работу реальных компьютеров или вычислительных систем.

3. Рекомендации:

- Упрощение диаграмм: Важно упростить диаграммы Тьюринга, чтобы сделать их более понятными для аудитории.

- Использование дополнительных инструментов: Комбинирование диаграмм Тьюринга с текстовыми описаниями и примерами может помочь улучшить понимание алгоритма.

- Тестирование на аудитории: Проведение тестирования диаграмм на целевой аудитории может помочь выявить слабые места и улучшить их.

В целом, диаграммы Тьюринга являются мощным инструментом для визуализации и изучения алгоритмов, но требуют внимательного подхода к их созданию и использованию.

Стоит отметить, что работа с диаграммами очень сильно ускорило создание алгоритма, хотя он и сложнее предыдущего. Я потратил на целый день меньше, так что слова о «человеческом лице» диаграмм Тьюринга себя оправдали.

На этом демонстрация определения алгоритмов через алгоритмические модели диаграмм Тьюринга завершено.

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 не применимыми, т. е . в обрабатываемом слове нет вхождений левой части ни одной формулы подстановки; либо только что применялась 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 не пустые, 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 означало 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, разделенных знаком “^”.

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.

Сам алгоритм выглядит так:

1a->0b

0a->a1

b0->0b

b1->1b

b>->>c

c0->0c

c1->1c

c->d

00d->0d0

01d->0d0

10d->1d1

11d->1d1

>0d->>0

>1d->>0

a->e

e0->e

e1->e

e>->.

>->a>

А теперь давайте его разберем подробнее этот алгоритм

1. 1a->0b: Если встречается символ "1" после символа "a", он заменяется на "0" и добавляется символ "b" после "a".

2. 0a->a1: Если встречается символ "0" после символа "a", он заменяется на "a" и добавляется символ "1" после "a".

3. b0->0bb: Если встречается символ "b" после символа "0", он заменяется на "0" и добавляется два символа "b" после "0".

4. 1b->1b: Если встречается символ "1" после символа "b", он остается без изменений.

5. b>->>cc: Если встречается символ "b" после символа ">", он заменяется на два символа "c" и добавляется два символа ">".

6. 0->0c: Если встречается символ "0", он заменяется на "0" и добавляется символ "c".

7. c1->1cc: Если встречается символ "c" после символа "1", он заменяется на "1" и добавляется два символа "c".

8. cc->d: Если встречаются два символа "c", они заменяются на символ "d".

9. 00d->0d: Если встречаются два символа "0" после символа "d", они заменяются на один символ "0".

10. 001d->0d0: Если встречаются символы "001" после символа "d", они заменяются на "0d0".

11. 10d->1d: Если встречаются символы "10" после символа "d", они заменяются на "1d".

12. 111d->1d1: Если встречаются символы "111" после символа "d", они заменяются на "1d1".

13. >0d->>0: Если встречается символ ">" после символа "0", он заменяется на ">0".

14. >1d->>0: Если встречается символ ">" после символа "1", он заменяется на ">0".

15. a->e: Если встречается символ "a", он заменяется на "e".

16. e0->ee: Если встречается символ "e" после символа "0", он заменяется на два символа "e".

17. e1->e: Если встречается символ "e" после символа "1", он остается без изменений.

18. e>->.: Если встречается символ ">" после символа "e", он заменяется на ".".

19. .>->a: Если встречается символ ".", он заменяется на ">" и добавляется символ "a".

Это пример машины Тьюринга, которая выполняет последовательность преобразований над строкой, используя заданный набор правил.

3.3. Сложностная оценка сделанного НАМ и выводы

Критика алгоритма Маркова включает в себя следующие аспекты:

1. Недостаток универсальности: Алгоритм Маркова не является универсальным, то есть не способен решать всевозможные задачи. Он предназначен для решения конкретных задач и может быть неэффективен или даже бесполезен для других задач.

2. Ограничения на сложность задач: Алгоритм Маркова может столкнуться с ограничениями при попытке решения сложных задач, таких как обработка больших объемов данных или выполнение высокоуровневых алгоритмов.

3. Неоптимальность: В некоторых случаях алгоритм Маркова может быть неоптимален, то есть существуют более эффективные способы решения задачи. Это может происходить из-за ограничений самого алгоритма или из-за специфики конкретной задачи.

4. Сложность разработки: Создание и настройка алгоритма Маркова для конкретной задачи может потребовать значительных усилий и времени. Это может быть вызвано необходимостью определения правил преобразования, тестирования и отладки алгоритма.

5. Трудность поддержки: После создания алгоритма Маркова могут возникнуть сложности с его поддержкой и модификацией. Изменение правил преобразования или добавление новых функций может потребовать значительного времени и усилий.

6. Необходимость в специализированных знаниях: Работа с алгоритмом Маркова требует специализированных знаний в области теории автоматов, формальных языков и вычислительной логики, что может быть препятствием для широкого круга специалистов.

7. Ограниченные возможности параллелизации: В зависимости от структуры задачи алгоритм Маркова может оказаться сложно параллелить, что снижает его эффективность на многоядерных системах.

В целом, алгоритм Маркова имеет свои преимущества, но также подвержен ряду ограничений и недостатков, которые следует учитывать при его использовании.

Заключение

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 я пришёл во время выполнения задач курсовой работы. Все они были мною успешно проделаны. На этом курсовая работа закончена.