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 **Пинчук М. С.**
Проверил:
Преподаватель, каф. 806 **Севастьянов В.С.**

Москва 2023

Содержание:

Введение.

Перед тем, как перейти к практической части курсовой работы, необходимо ознакомится во введении с её теоретической частью.

Итак, что же такое алгоритмы и алгоритмические модели? Алгоритм — это точно заданная последовательность правил, указывающая, каким образом можно за конечное число шагов получить выходное сообщение определенного вида, используя заданное исходное сообщение. Всё было бы замечательно с этим определением, если бы не одно «но»: оно не является формальным. Для более строгой трактовки определения необходимы так называемые алгоритмические модели. Среди них выделяют: рекурсивные функции (понятие алгоритма связывается с вычислениями и числовыми функциями), машины Тьюринга (алгоритм представляется как описание процесса работы некоторой машины, способной выполнять лишь небольшое число весьма простых операций), нормальные алгоритмы Маркова (алгоритмы описываются как преобразования слов в произвольных алфавитах). В моей курсовой работе будут фигурировать лишь машины Тьюринга и нормальные алгоритмы Маркова. После такого небольшого вступления можно перейти к описанию целей и задач этой работы.

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

Задачи: (взял цели из лабораторных работ)

1) составить программу машины Тьюринга в четвёрках, 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, обменивающий местами два троичных числа, разделённых знаком "^".

4) обобщить полученную информацию и сделать соответствующий вывод.

Для выполнения поставленных задач мне необходимы:

- Эмулятор машины Тьюринга в четвёрках (ссылка на него: file:///C:/Users/vadim/Downloads/l05-2011_jstu4-2.3/jstu4-2.3/jstu4/jstu4.html?lang=ru)
- Диаграммер для работы с диаграммами Тьюринга («VirtualTuringMachine.exe»)
- эмулятор для нормальных алгоритмов Маркова (file:///C:/Users/vadim/OneDrive/Рабочий%20стол/nam/index.html)
- текстовый редактор Microsoft Word
- Репозиторий на GitHub, куда были выложены все описанные ниже алгоритмы(utyfull/labs_inf (github.com))

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

1. Машина Тьюринга.

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).

Мне это определение и самому было не понятно, пока я не сделал пятую лабораторную работу. В переводе на русский язык можно сказать, что каждая четвёрка 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 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.

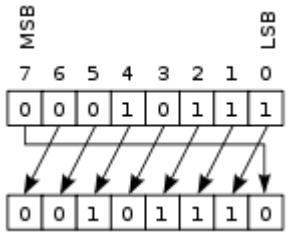
Подводя итоги теоретической части, нужно сказать, что на вот этих «четвёрках» и построено определение алгоритма по Тьюрингу. С их помощью и задаются правила, по которым можно определить за конечное число шагов необходимое нам преобразование сообщения. Определение алгоритмов по Тьюрингу настолько фундаментально, 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 вычислинении двоичного циклического сдвига первого числа вправо на число разрядов, равное втором . Под нормированным переводом подразумевалось начальное положение головки справа от входного сообщения, сохранение исходного сообщения и запись нового числа справа от исходного сообщения через один пробел.

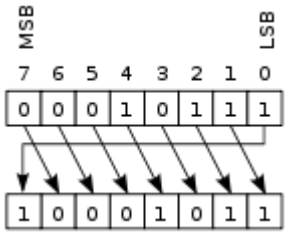
Сначала я постараюсь описать основополагающую информацию для выполнения данной лабораторной работы.

Примером такой информации является описание циклического сдвига.

Циклический сдвиг



Циклический сдвиг влево



Циклический сдвиг вправо

При этом сдвиге уходящий бит появляется на месте появившегося свободного на другом конце числа.

Пример

- Пусть у нас есть число 11111010b (в двоичной системе).
- Если сделать сдвиг влево на 1 бит, то получим число 11110101b.
- Если сделать сдвиг исходного числа вправо на 1 бит, то получим число 01111101b.

Как видно из примера циклического сдвига, достаточно забирать крайнее число справа или слева (в зависимости от вида сдвига) и вставлять его с другой стороны. Как бы сдвигая число на ячейку вправо или влево, и число, уходящее за рамки доступных ячеек вставлять в свободную ячейку с другой стороны.

Вышенаписанное является алгоритмом, свободно реализуемым на Машине Тьюринга и содержит все необходимые шаги для реализации данного задания, используя алгоритмическую модель Тьюринга.

Необходимо заметить, что задание осложняется тем, что необходимо выполнить циклический сдвиг не 1 раз, а столько раз, чему равно число справа. Для этого необходимо немного усложнить алгоритм. Так, необходимо вычитать из числа единицу, и только потом осуществлять сдвиг. Выделим точку выхода: равенство нулю второго числа.

Также не стоит забывать, что машина должна быть нормированной, следовательно необходимо написать машину копирования и использовать ее, чтобы скопировать исходные данные.

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

Входное сообщение	Выходное сообщение
101 1	101 1 110
101 10	101 10 011

Из тестов видно, что стартовые данные не удаляются. Это связано с тем, что чтобы машина была нормированной входные данные не должны быть изменены.

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.

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. 1.3. Сложностная оценка и выводы к машине Тьюринга

1. Машина копирования для соблюдения нормированности алгоритма

Машина копирования описана в учебнике Зайцева В. Е.:

$$
\begin{array}{llll}
[z_0\lambda|||||(\lambda)\lambda > & \overset{*}{\Rightarrow} [z_0(\lambda)|||||\lambda\lambda > & \Rightarrow [z_0\lambda(|)||||\lambda\lambda > & \Rightarrow [z_0\lambda(\lambda)||||\lambda\lambda > \overset{*}{\Rightarrow} \\
[z_0\lambda\lambda||||(\lambda)\lambda > & \Rightarrow [z_0\lambda\lambda|||\lambda(\lambda)\lambda > & \Rightarrow [z_0\lambda\lambda||||(\lambda)|\lambda > & \overset{*}{\Rightarrow} [z_0\lambda(\lambda)||||\lambda|\lambda > \Rightarrow \\
[z_0\lambda(|)||||\lambda|\lambda > & \Rightarrow [z_0\lambda|(|)|||\lambda|\lambda > & \Rightarrow [z_0\lambda|(\lambda)|||\lambda|\lambda > & \overset{*}{\Rightarrow} [z_0\lambda|\lambda|||(\lambda)|\lambda > \Rightarrow \\
[z_0\lambda|\lambda|||\lambda|(\lambda)\lambda > & \Rightarrow [z_0\lambda|\lambda|||\lambda|(|)\lambda > & \overset{*}{\Rightarrow} [z_0\lambda|(\lambda)|||\lambda||\lambda > & \Rightarrow [z_0\lambda|(|)|||\lambda||\lambda > \Rightarrow \\
[z_0\lambda||(|)||\lambda||\lambda > & \Rightarrow [z_0\lambda||(\lambda)||\lambda||\lambda > & \overset{*}{\Rightarrow} [z_0\lambda|||||\lambda|||||(\lambda)\lambda > &
\end{array}
$$

Программа может быть записана в виде следующей таблицы:

Состояние	**Буква**			
	|		λ	
начало работы	—		l	поиск начала исходного слова
поиск начала исходного слова	l	поиск начала исходного слова	λ	начало исходного слова найдено
начало исходного слова найдено	—		r	копирование очередной буквы
копирование очередной буквы	λ	пометка копируемой буквы	λ	слово скопировано
пометка копируемой буквы	—		λ	перенос буквы в копию исходного слова
перенос буквы в копию исходного слова	—		r	поиск конца исходного слова
поиск конца исходного слова	r	поиск конца исходного слова	λ	конец исходного слова найден
конец исходного слова найден	—		r	поиск конца копии
поиск конца копии	r	поиск конца копии	l	конец копии найден
конец копии найден	—		|	очередная буква скопирована
очередная буква скопирована	l	поиск начала копии слова	—	
поиск начала копии слова	l	поиск начала копии слова	λ	начало копии слова найдено
начало копии слова найдено	—		l	поиск пометки в исходном слове
поиск пометки в исходном слове	l	поиск пометки в исходном слове	λ	пометка найдена
пометка найдена	—		|	копирование следующей буквы
копирование следующей буквы	r	копирование очередной буквы	—	
слово скопировано	—		r	установка головки
установка головки	r	установка головки	λ	конец работы
конец работы	s	конец работы	s	конец работы

Ее реализация в контексте задачи немного усложнена тем, что необходимо скопировать не один набор символов, а два, разделенных пробелом.

Реализация на МТ следующая:

```
00, ,<,01

01,0,<,01  01,1,<,01 01, ,=,02
02, ,<,03

03,0,<,03  03,1,<,03 03, ,=,04
04, ,=,05

05, ,>,06

06,0, ,99 06,1, ,98

99, ,>,07 98, ,>,12

07,0,>,07  07,1,>,07 07, ,=,09  09, ,>,10   12,0,>,12  12,1,>,12 12, ,=,13  13, ,>,14

10,0,>,10  10,1,>,10 10, ,=,11 11, ,=,16    14,0,>,14  14,1,>,14 14, ,=,15 15, ,=,26

16, ,>,17 26, ,>,27

17,0,>,17 17,1,>,17 17, ,0,18

18,0,<,18  18,1,<,18 18, ,=,19
19, ,<,20

20,0,<,20  20,1,<,20 20, ,=,21
21, ,<,22

22,0,<,22  22,1,<,22 22, ,=,23
23, ,0,24



27,0,>,27 27,1,>,27 27, ,1,28

28,0,<,28  28,1,<,28 28, ,=,29
29, ,<,30

30,0,<,30  30,1,<,30 30, ,=,31
31, ,<,32

32,0,<,32  32,1,<,32 32, ,=,33
33, ,1,24


24,0,>,25 24,1,>,25

25,0, ,99 25,1, ,98 25, , ,34

34, ,>,35

35,0, ,36 35,1, ,44

36, ,>,37 44, ,>,45

37,0,>,37  37,1,>,37 37, ,=,38  38, ,>,39    45,0,>,45  45,1,>,45 45, ,=,46  46, ,>,47

39,0,>,39 39,1,>,39 39, ,=,40 40, ,=,41       47,0,>,47 47,1,>,47 47, ,=,48 48, ,=,49

41, ,>,42 49, ,>,50

42,0,>,42 42,1,>,42 42, ,0,51

51,0,<,51  51,1,<,51 51, ,=,52
52, ,<,53

53,0,<,53  53,1,<,53 53, ,=,54
54, ,<,55

55,0,<,55  55,1,<,55 55, ,=,56
56, ,0,56
```

```
50,0,>,50 50,1,>,50 50, ,1,58

58,0,<,58  58,1,<,58 58, ,=,59
59, ,<,60

60,0,<,60  60,1,<,60 60, ,=,61
61, ,<,62

62,0,<,62  62,1,<,62 62, ,=,63
63, ,1,56

56,0,>,64 56,1,>,64

64,0, ,36 64,1, ,44 64, ,=,65
```

После копирования можно приступать к реализации алгоритма. Заметим, что после копирования МТ находится в ячейке после крайнего символа.

2. Реализация алгоритма.

В алгоритме можно выделить две основные части:

1. Вычитание единицы из правого числа и проверка на точку выхода.

```
70x,0,<,70x 70x,1,<,70x 70x, ,>,71x

71x,0, ,72x 71x,1,=,73x 71x, ,<,75x

72x, ,>,71x

73x,1,>,73x 73x,0,>,73x 73x, ,<,70

75x, ,<,75x 75x,1,>,xxx 75x,0,>,xxx

xxx, ,#,xxx

70,0,1,71 70,1,0,77 70, ,=,72
71,1,<,70 71,0,<,70

77,1,<,77 77,0,<,77 77, ,=,72

72, ,>,73

73,0,<,74 73,1,<,74
74, ,<,z 74,1,=,74 74,0,=,74

z, ,<,z z,1,=,y z,0,=,y

y,1,<,y y,0,<,y y, ,>,75

75,0,=,76 75,1,=,78
```

Здесь происходит вычитание 1 из двоичного числа и проверка на число сдвигов (если число сдвигов 0, то осуществляется последний сдвиг и происходит выход)

2. Осуществление циклического сдвига 1 раз

Здесь циклический сдвиг осуществляется один раз и переходит к вычитанию единицы (п. 1) или выходу.

```
79x,0,=,78 80x,1,=,76
76,0,>,79 76,1,>,79 78,1,>,80 78,0,>,80
79,1,0,79x 79,0,=,75 79, ,=,81    80,1,=,75 80,0,1,80x 80, ,=,88
81, ,<,83 88, ,<,89
83,1,<,83 83,0,<,83 83, ,>,84
84,1,0,85 84,0,=,85
85,1,>,85 85,0,>,85 85, ,>,86
86, ,>,86 86,1,=,87 86,0,=,86z
87,1,>,87 87,0,>,87 87, ,=,69

89,1,<,89 89,0,<,89 89, ,>,90
90,1,=,91 90,0,1,91
91,1,>,91 91,0,>,91 91, ,>,92
92, ,>,92 92,1,=,93 92,0,=,86z
93,1,>,93 93,0,>,93 93, ,=,69
86z,0, ,87z
87z, ,>,88z
88z, ,=,89z 88z,1,=,93 88z,0,=,93
89z, ,<,89z 89z,1,>,90z 89z,0,>,90z
90z, ,#,90z
```

Сложностная оценка:

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
10 1	9
10 10	14
1 1	7
1 10	11
10 100	22

1000 1	14

Как можно убедиться из примеров, при увеличении длины слов в 2 раза, время выполнения возрастает в 2 раза -> сложностная оценка - O(n).

Данный алгоритм легок в реализации, но не является высокоэффективным. Это связано с тем, что в процессе выполнения осуществляются дополнительные проверки, которые занимают довольно много времени.

2. Диаграмма Тьюринга

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 циклического сдвига второго числа влево на число разрядов, равное первому

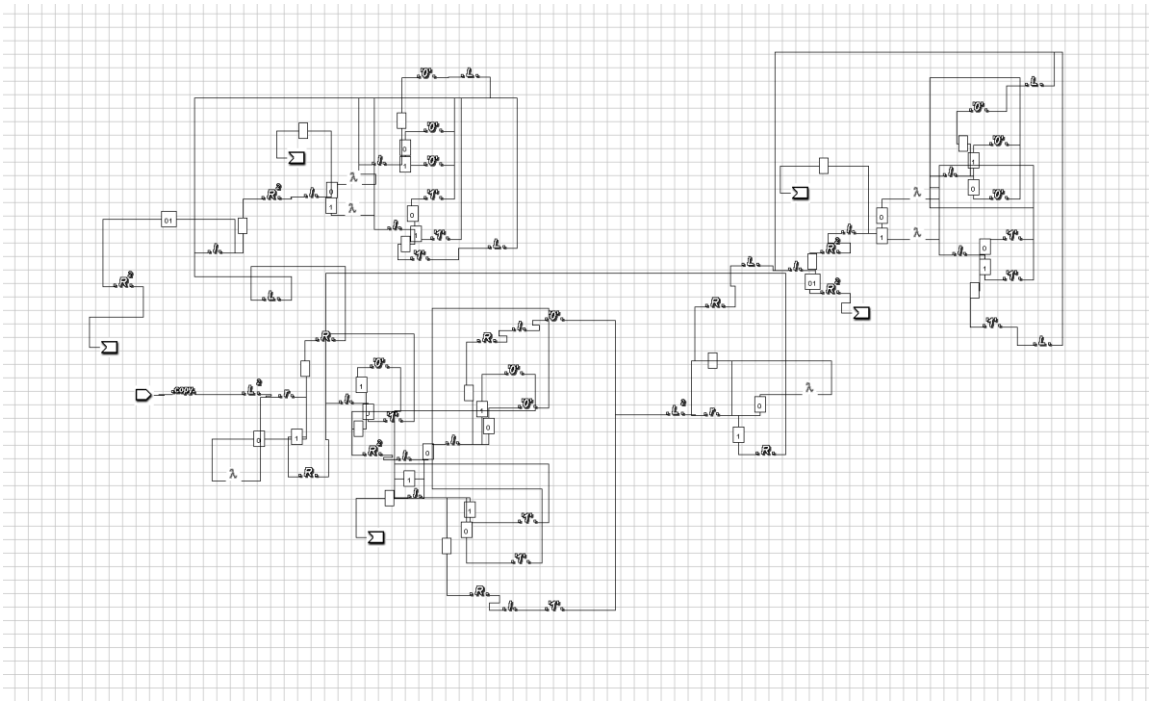
Циклический сдвиг для задания с использованием ДТ аналогичен циклического сдвигу для МТ, с единственным отличием: циклический сдвиг осуществляется в другую сторону и число сдвигов должно быть равно не левому числу, а правому. Это несколько осложняет реализации альгоритма, так как теперь необходимо сдвинуть получившееся число.

А пока стоит ещё остановиться на тестах.

Входные данные	Выходные данные
000 10	000 10 10
001 10	001 10 01

Для ДТ применяются такие же критерии нормированности -> входные данные должны остаться без изменений.

Диаграмма main machine



Так выглядит диаграмма целиком. Для того чтобы лучше понять, как осуществляется алгоритм, необходимо остановиться на конкретных частях.

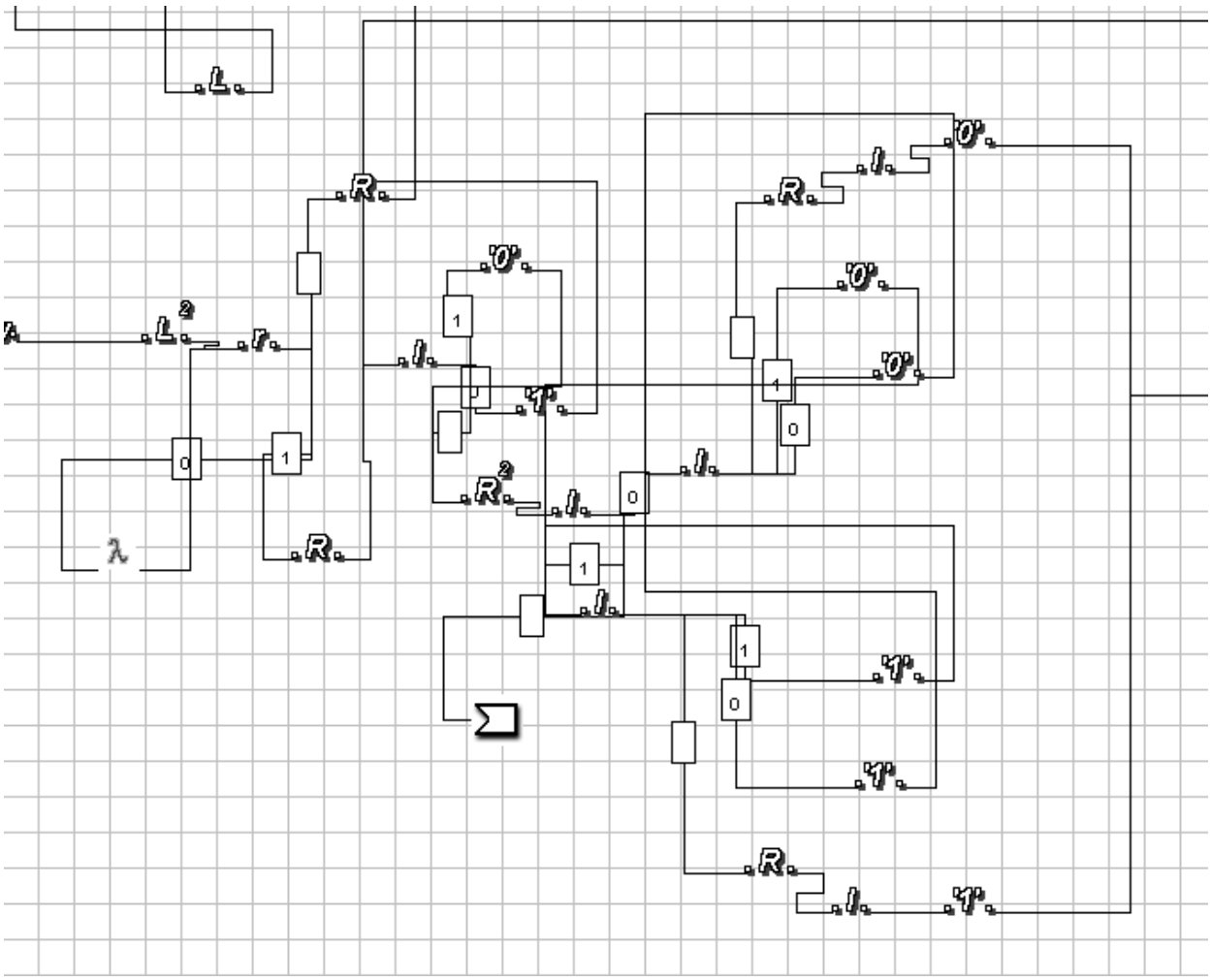
1. Копирование

Копирование представлено подмашиной copy и выглядит следующим образом:

2. Вычитание единицы

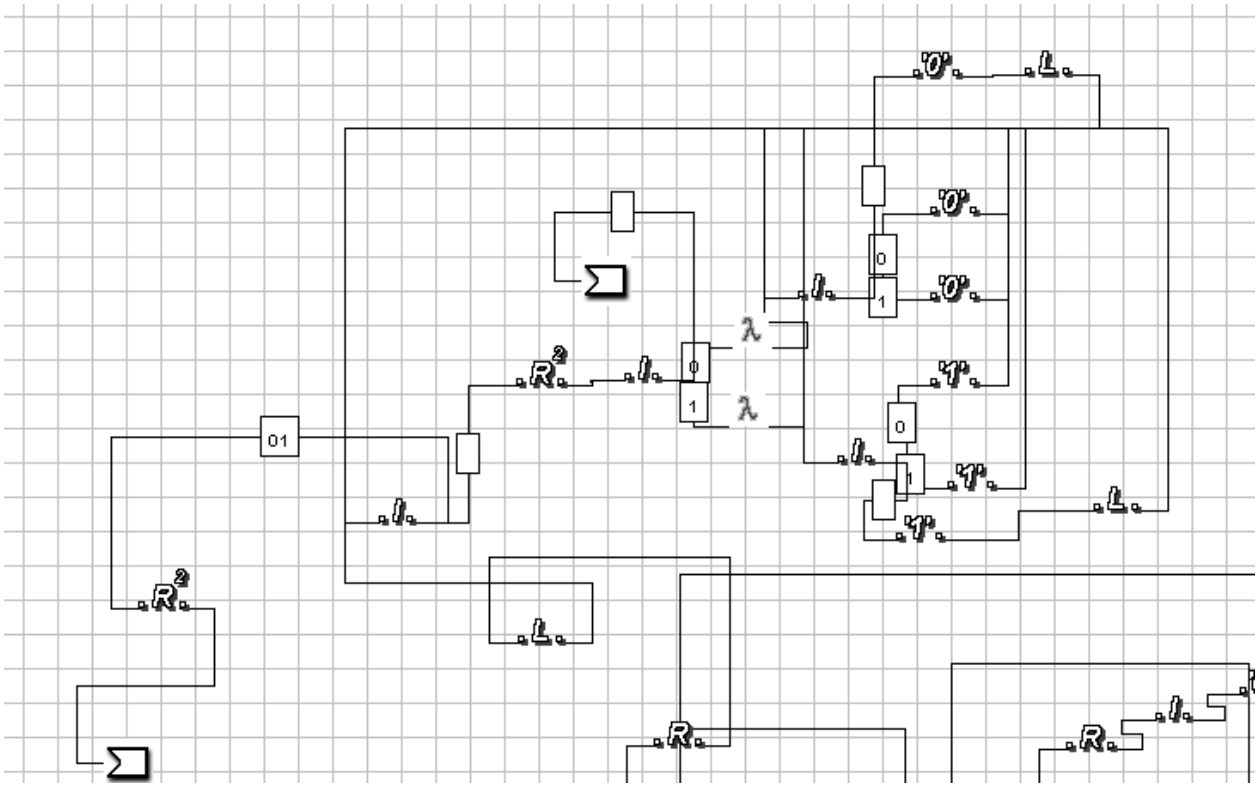
Достаточно сложно визуально осознать, что происходит в этой части ДТ.

Здесь происходит вычитание 1 и переход к осуществлению сдвига. Здесь также находится одна из точек выхода, которая проверяет на количество оставшихся сдвигов.



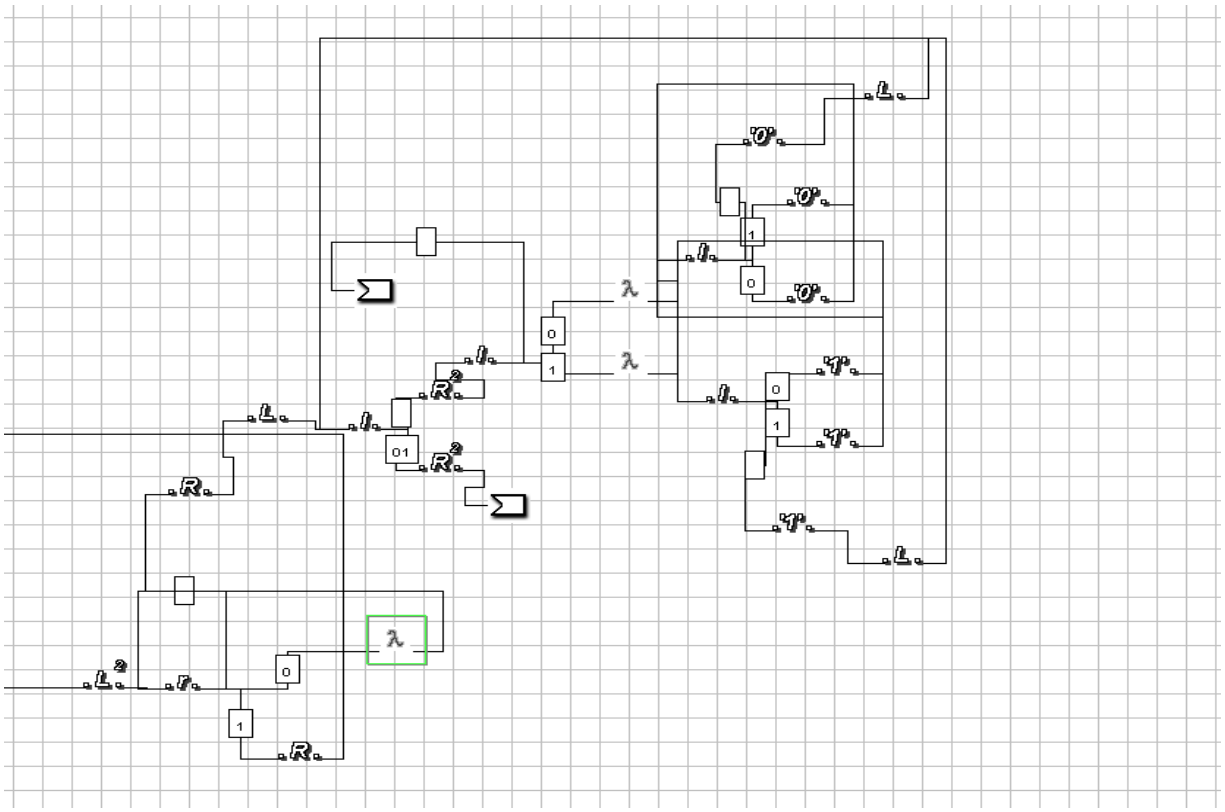
2. 2. Точка выхода и осуществлении сдвига последний раз

На диаграмме видно, что переход к этой части диаграммы осуществляется после проверки в предыдущем разделе. В этой части осуществляется сдвиг последний раз и машина завершает свою работу.



3. Осуществление циклического сдвига

В этой части диаграммы осуществляется циклический сдвиг 1 раз по аналогии с МТ.



Сложностная оценка диаграммы Тьюринга и выводы

Как и в случае с машиной Тьюринга сложность я буду оценивать на основе данных измерений зависимости скорости выполнения программы от длины строки.

Входные данные	Время выполнения, с
1 10	9
10 10	14
1 1	7
1 10	11
10 100	20
1 1000	14

По данным из таблицы видно, что сложность алгоритмов O(n) и время выполнения примерно совпадает. Заметим, что время исполнения ДТ было измерено не программным методом. Для измерения времени выполнения была взята скорость выполнения одной команды, и команды ДТ выполнялись с этой же скоростью, не смотря на возможности ДТ выполнять программу быстрее. Это лишь подтверждает сложность алгоритма O(n).

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 оаправдали.

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

3. Нормальные алгоритмы Маркова (НАМ)

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 алгоритма вычисления троичного числа - троичного логарифма троичного числа.

Как обычно, перед осмотром программы нужно прописать пару тестов.

Входные данные	Выходные данные
000	0
001	1

Демонстрация сделанного НАМ

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

Но сначала хотелось бы объяснить вкратце, что этот алгоритм делает:

1. Удаление ведущих нулей числа, так как они могут помешать в процессе вычисления.

2. Замена всех символов на нули и удаление последнего символа.

3. Удаление одного нуля и прибавление единицы к троичному числу, образовывающемуся слева.

Эти шаги основаны на важном наблюдении: округленный вниз троичный логарифм троичного числа равен количеству цифр числа, не считая последнего (1 из цифр числа)

Объяснение первой части кода:

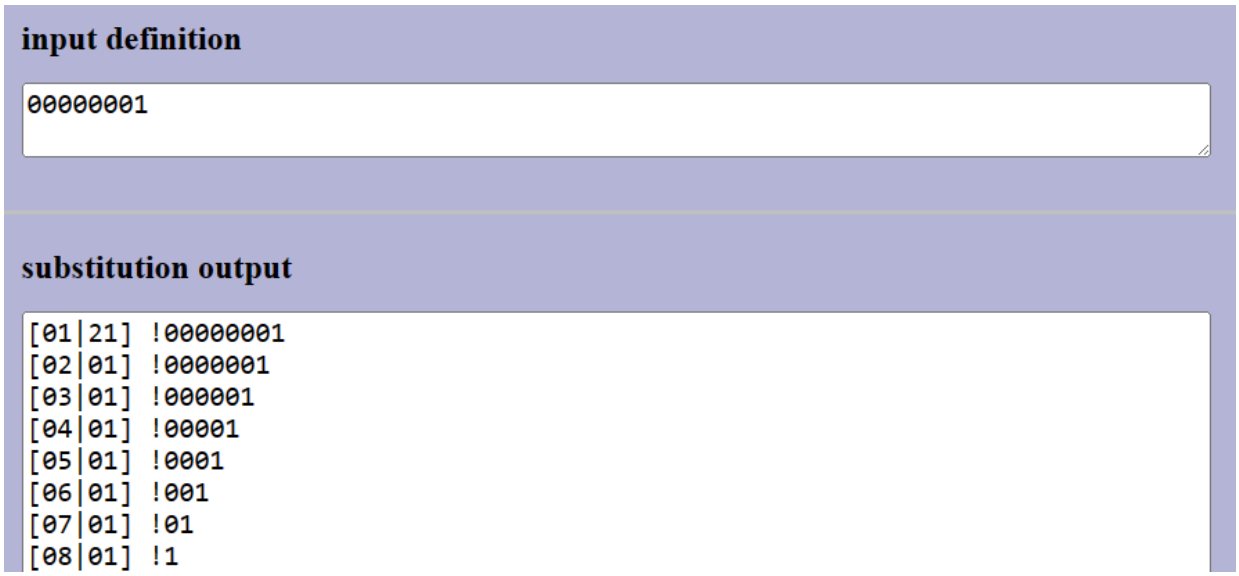
Удаление ведущих нулей осуществляется с помощью двух строчек:

!0->!

->!

Это происходит следующим образом: В момент запуска не выполняется ни одно из условий, кроме последнего, которое выполняется всегда. Оно создает восклицательный знак в начале слова из ничего. После этого появляется условие 1, которое удаляет нули.

Пример:



Как видно из примера, удаляются все ведущие нули, но остается восклицательный знак.

Это помогает в реализации следующей части программы.

Следующий шаг - замена всех цифр на нули, кроме последней:

!1->0#

!2->0#

#1->0#

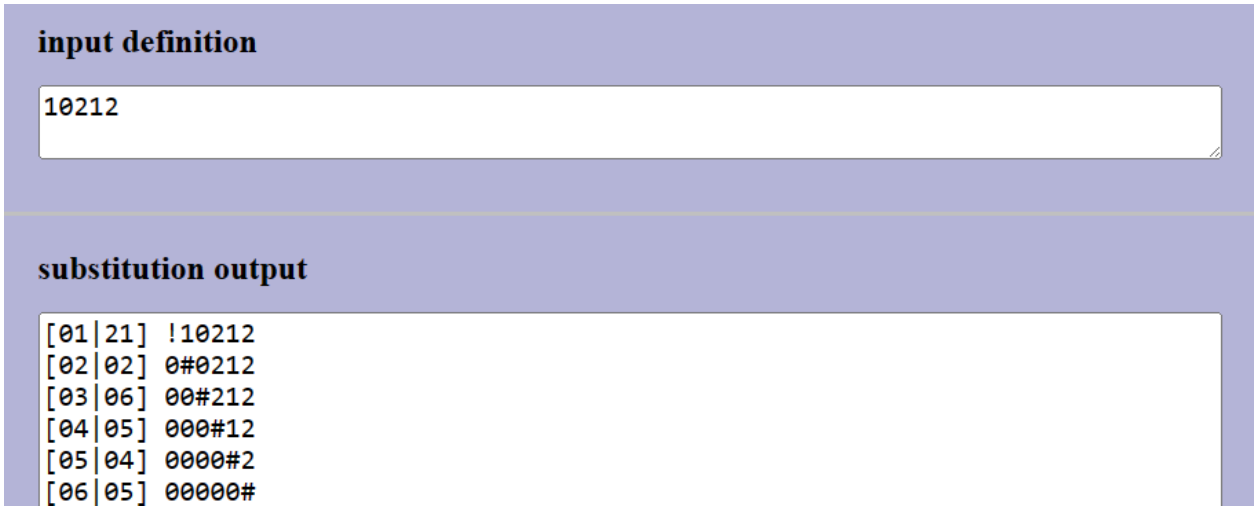
#2->0#

#0->0#

Как видно из кода выше, добавляется новый специальный символ - #.

Он нужен для того, чтобы заменить цифры на 0. Первые две строчки избавляются от восклицательного знака и создают новый специальный символ. Затем мы двигаем специальный символ, одновременно заменяя попавшиеся цифры на 0.

Пример:



Следующий шаг - подсчет нулей, создание и накопление слева троичного числа.

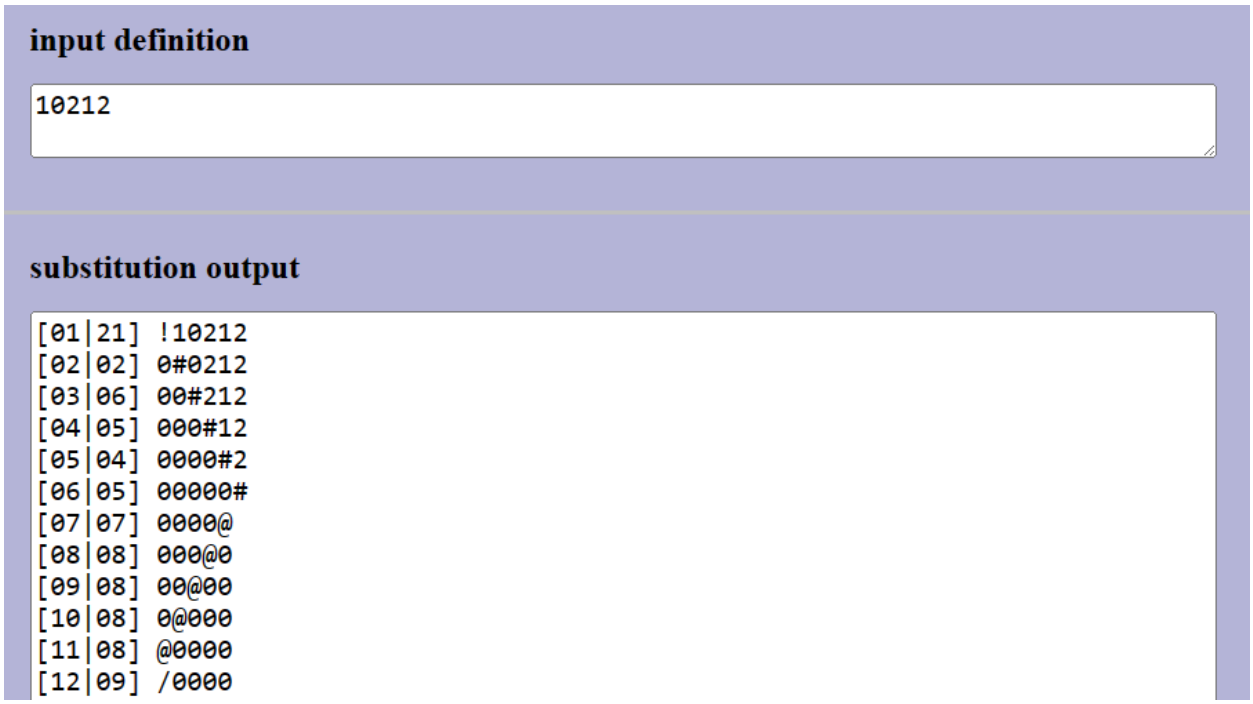
Для этого вводится новый специальный символ, который удаляет последнюю цифру и двигается в начало слова для создания области подсчета троичного логарифма троичного числа.

0#->@

0@->@0

@->/

Пример:



Слева от специального символа / будет вестись подсчет троичного числа

Особенности прибавления единицы к троичному числу:

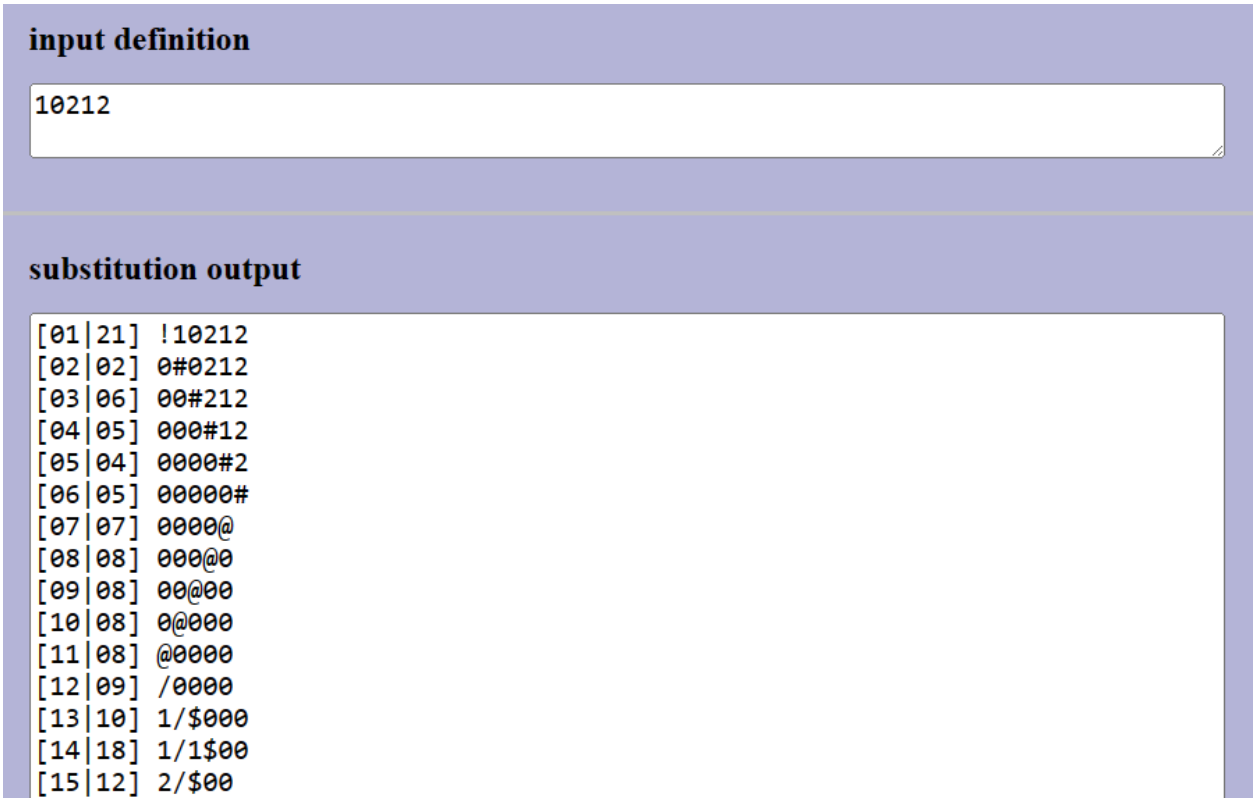
Для реализации прибавления единицы к троичному числу вводится специальный символ - $. Он нужен для ‘съедания нуля’ и привращение его в единицу для троичного числа.

/0->1/$

0/1->1/

1/1->2/

Пример:



Из изображения понятно, что символ $ нужен лишь для удобства вычисления: он превращает 0 позади себя в единицу, которая впоследствии используется для прибавления к троичному числу слева.

В какой-то момент может возникнуть необходимость прибавить 1 к числу 2. Для этого нужно создать новый разряд. Это реализуется с помощью специального символа ^:

2/1->^0/

0^->1

1^->2

2^->^0

^0->10

Пример:

input definition

```
10212
```

substitution output

```
[01|21] !10212
[02|02] 0#0212
[03|06] 00#212
[04|05] 000#12
[05|04] 0000#2
[06|05] 00000#
[07|07] 0000@
[08|08] 000@0
[09|08] 00@00
[10|08] 0@000
[11|08] @0000
[12|09] /0000
[13|10] 1/$000
[14|18] 1/1$00
[15|12] 2/$00
[16|18] 2/1$0
[17|13] ^0/$0
[18|17] 10/$0
[19|18] 10/1$
[20|11] 11/$
```

В данный момент программа почти готова. Осталось только избавится от специальных символов. Для этого используется следующий код:

$0->1$

/$->.

!!->.0

Этот код останавливает работу программы.

Таким образом программа готова к работе.

Пример:

input definition

```
20011
```

substitution output

```
[01|21] !20011
[02|03] 0#0011
[03|06] 00#011
[04|06] 000#11
[05|04] 0000#1
[06|04] 00000#
[07|07] 0000@
[08|08] 000@0
[09|08] 00@00
[10|08] 0@000
[11|08] @0000
[12|09] /0000
[13|10] 1/$000
[14|18] 1/1$00
[15|12] 2/$00
[16|18] 2/1$0
[17|13] ^0/$0
[18|17] 10/$0
[19|18] 10/1$
[20|11] 11/$
[21|19] 11
```

Сложностная оценка сделанного НАМ и выводы

Входные данные	Время выполнения, с
1212	1
10201	1.2
21102	1.2
112211	1.4
1002122	1.6
2012021	1.8
11101112	2

По таблице видно, что алгоритм имеет сложность O(n). Важное замечание:

Алгоритм не зависит от значения числа напрямую, он зависит от количества цифр в числе. Из-за этого можно четко проследить время выполнения.

Также стоит заметить, что скорость выполнения алгоритма очень велика. Это связано с тем, что он вычисляет значение, округляя вниз.

Например, при вводе числа 1_000_000_000 в десятичной системе счисления, результат выводится всего за 5 секунд.

На этом демонстрация определения через НАМ завершена. Пришло время делать окончательные выводы в этой курсовой работе.

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.