Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Класичні алгоритмічні системи. Розробка емулятора алгоритмів Маркова

Текстова частина до курсової роботи
за спеціальністю „Інформатика”

Керівник курсової роботи
Доцент Франчук О.В.
(прізвище та ініціали)
_____
(підпис)
“____” _____ 2020 р.

Виконав студент Дайрабеков А.О.
_____
(підпис)
“____” _____ 2020 р.

Київ 2020

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри інформатики,
Проф., к. ф.-м. н. _____С. С. Гороховський
(підпис)
„____”_____2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту ***Дайрабекову Артему*** факультету ***інформатики 4*** курсу
ТЕМА ***Класичні алгоритмічні системи. Розробка емулятора алгоритмів Маркова***
Вихідні дані:
-
-
Зміст ТЧ до курсової роботи:
Індивідуальне завдання
Календарний план
Анотація
Вступ
1 Основна частина
2 Практична частина
Висновки
Список використаних джерел

Дата видачі „___” _____ 2019 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Тема: ***Класичні алгоритмічні системи. Розробка емулятора алгоритмів Маркова***

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.		
2.	Огляд технічної літератури за темою роботи.		
3.	Написання вступу. Дати загальні відомості про принципи розробки		
4.	Розгляд класичних алгоритмічних систем		
5.	Робота з класичними алгоритмічними системами		
6.	Написання пояснювальної записки		
7.	Створення слайдів для доповіді та написання доповіді.		
8.	Завершення роботи над практичною частиною		
9.	Завершення корегування роботи		
10.	Остаточне оформлення пояснювальної записки та слайдів.		
11.	Захист курсової роботи		

Студент _____

Керівник _____

“_____”_____

ЗМІСТ

АНОТАЦІЯ 5
ВСТУП 6
РОЗДІЛ 1. ІСТОРІЯ РОЗРОБКИ КЛАСИЧНИХ АЛГОРИТМІЧНИХ СИСТЕМ 8
1.1 Поняття алгоритму 8
1.2 Поняття алгоритмічної системи 8
РОЗДІЛ 2. ПОРІВНЯННЯ РІЗНИХ КЛАСИЧНИХ АЛГОРИТМІЧНИХ СИСТЕМ 10
2.1 Машина Тьюрінга 10
2.2 Машина Поста 12
2.3 Алгоритм Маркова 13
РОЗДІЛ 3. ОПИС РЕАЛІЗАЦІЇ ЕМУЛЯТОРА АЛГОРИТМІВ МАРКОВА 16
3.1 Приклад створення та виконання алгоритму Маркова 16
3.2 Кілька прикладів різних алгоритмів Маркова 23
ВИСНОВКИ 29
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 30

АНОТАЦІЯ

У даній курсовій роботі розглянуто різні алгоритмічні системи (зокрема, машина Тьюрінга, Поста та алгоритми Маркова). Було створено кілька прикладів виконання однакової задачі на цих системах. Також, було побудовано емулятор алгоритмів Маркова. Для створення емулятора було використано такі засоби: мова Swift та IDE XCode.

ВСТУП

Алгоритми вже давно існують у нашому світі. Ще в 9 сторіччі вчений Ал-Хорезмі у своєму трактаті систематизував та описав арифметику в десятковій системі числення та розв'язок квадратних рівнянь. Через деякий час трактат потрапив у Європу, де його переклали як «Algoritmi de numero Indorum» (Алгоритими про рахунок індійською). Так і з'явився термін «алгоритм».

Проте, тільки у 20 сторіччі алгоритми почали займати центральне місце у сучасній математиці. Це сталось завдяки швидкому розвитку технологій та необхідністю вміти формально описувати задачі для їхнього виконання деякою машиною. Таким чином, з'явилися перші алгоритмічні системи, такі як машина Тьюрінга, алгоритми Маркова і багато інших.

На сьогоднішіній день, будь-яка сучасна мова програмування є Тьюрінг-повною. Тобто, для будь-якої сучасної мови можна створити еквівалентну їй машину Тьюрінга (формальний опис якої Алан Тьюрінг дав ще в далекому 1936 році)

Джерелом дослідження є класичні алгоритмічні системи.

Об`єктом дослідження є різні класичні алгоритмічні системи.

Предметом дослідження є можливості різних алгоритмічних систем виконати одну й ту саму задачу.

Мета роботи: проаналізувати різні алгоритмічні системи, порівняти їх на прикладі виконання одних і тих самих задач.

Мету буде реалізовано шляхом виконання таких пунктів:

1. Проаналізувати різні класичні алгоритмічні системи
2. Побудувати рішення однієї і тої самої задачі на різних системах

Теоретична цінність роботи полягає в тому, що спостереження та висновки, зроблені в роботі, можуть слугувати теоретичною базою для розуміння принципів роботи простих ЕОМ та побудови алгоритмів.

Практичне значення дослідження полягає в нагоді створення емулятора одного з класичних алгоритмічних систем - алгоритмів Маркова.

Робота складається зі змісту, вступу, трьох розділів, висновку, додатків та списку використаних джерел.

РОЗДІЛ 1. ІСТОРІЯ РОЗРОБКИ КЛАСИЧНИХ АЛГОРИТМІЧНИХ СИСТЕМ

1.1 Поняття алгоритму

У 1936-1937 роках А. Тьюрінг та Е. Пост дали визначення поняттю алгоритму майже одночасно з роботами А. Черча та С. Кліні з приводу алгоритмів. Основна ідея визначення алгоритму за Тьюрінгом: алгоритмічні процеси - це процеси, які може виконати спеціально побудована машина. Згідно цьому, шляхом точних математчних термінів були описані класи машин, що можуть імітувати алгоритмічні процеси.

Алгоритм має наступні властивості:

- Дискретність - розв’язок задачі має бути шляхом виконання деяких простих кроків. До того ж, кожний крок має виконуватись у скінченний проміжок часу.
- Детермінованість - у кожний момент часу наступний простий крок має бути однозначно визначений станом системи.
- Скінченність - при правильно заданих початкових даних, алгоритм має надати результат протягом виконання скінченної кількості простих кроків.
- Зрозумілість - алгоритм має виконувати тільки ті команди, які входять до системи команд виконавця алгоритму.
- Універсальність - алгоритм може бути застосованим до різного набору початкових даних.
- Результативність - алгоритм має закнічуватись певними результатами.

1.2 Поняття алгоритмічної системи

Алгоритмічна система - це загальний спосіб задання алгоритму. При створенні алгоритмічної системи використовуються формалізовані засоби, які можна розділити на два типи - алгебраїчні та геометричні.

Алгебраїчна теорія будується таким чином, що алгоритми розглядаються у

вигляді лінейних тестів.

Геометрична теорія будується у вигляді множин між якими вводяться бінарні зв’язки, що носять характер відображення або бінарних вносин.

До алгебраїчних теорій відносять: машини Тьроінга, Поста, частково-рекурсивні функції операторні системи Ван-Хао та багато інших.

До геометричних теорій відносять представлення нормальних алгоритмів Маркова у вигляді блок-схем, запропонованих Л. А. Калужіним, блок-схемний метод алгоритмізації та багато інших.

РОЗДІЛ 2. ПОРІВНЯННЯ РІЗНИХ КЛАСИЧНИХ АЛГОРИТМІЧНИХ СИСТЕМ

2.1 Машина Тьюрінга

Машина Тьюрінга складається з двох частин - стрічки та автомату.

Стрічки (рис.2.1) та автомату (рис.2.2):

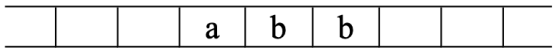

Рис.2.1 Стрічка машини Тьюрінга

Рис.2.2 Автомат машини Тьюрінга

Стрічка використовується для зберігання інформації. Ця стрічка нескінченна в обидві сторони та розбита на клітинки, що не мають нумерації та назви. Те, що зберігається в клітинці, може бути записаним або зчитаним. З самого початку, в стрічці записане деяке слово, решта клітинок є порожніми.

В кожний момент часу, автомат Машина Тьюрінга знаходиться під однією з клітинок, бачить її вміст, а також знаходиться в одному зі станів.

Пара (вміст клітинки, стан) називається конфігурацією.

Виконання операції - це виконання 3 наступних дій, строго у заданому порядку:

1) Записати символ у комірку
2) Зсунути на одну комірку вправо (R), на одну комірку вліво (L), або лишитись на місці (N)
3) Перейти в інший стан

Отже, по суті, одна операція може бути записана у вигляді трійки.

$$a, [L|R|N], q$$

Де a - символ, який ми записуємо, $[L|R|N]$ - один з трьох зсувів автомату та q - новий стан.

Машина Тьюрінга задається набором: *конфігурація -> виконання операції.*

Приклад 1

Машина Тьюрінга, де стрічка складається з довільної кількості одиниць, нам треба дописати справа та повернути автомат на перший символ одиниці зліва (це будемо позаначати як пустий символ, за $q!$ - знак кінця роботи):

$$(q1,1) \to (1, R, q1)$$
$$(q1, e) \to (1, q2, L)$$
$$(q2,1) \to (1, q2, L)$$
$$(q2, e) \to (e, q!, R)$$

Приклад 2

Машина Тьюрінга, де стрічка - це двійкове число (складається з нулів та одиниць), у результаті має бути це число плюс одиниця, автомат має знову бути на першому непустому символі зліва:

Рухаємо автомат вправо до кінця:

$$(q1,1) \to (1, R, q1)$$
$$(q1,0) \to (0, R, q1)$$
$$(q1, e) \to (e, L, q2)$$

Перевіряємо, чи остання цифра нуль. Якщо нуль, тоді просто змінюємо на одиницю та повертаємось назад:

$$(q2,0) \to (1, L, q4)$$
$$(q2,1) \to (0, L, q3)$$

Якщо ж була одиниця, знаходимо перший нуль, попутно змінюючи усі одиниці на нулі.

Знайшовши перший нуль, просто рухаємось назад:

$$(q3,0) \to (1, L, q4)$$
$$(q3,1) \to (0, L, q3)$$

Якщо не було жодного нуля (наприклад, число 111), просто дописуємо одиницю зліва:

$$(q3, e) \to (1, q!, N)$$

Просто рух назад (до пустого символу):

$$(q4,0) \to (0, L, q4)$$
$$(q4,1) \to (1, L, q4)$$

$$(q4, e) \rightarrow (e, q!, R)$$

2.2 Машина Поста

У 1936 році, одночасно з Аланом Тьюрінгом, американський математик Еміль Пост запропонував свою модель машини.

У ній також є нескінченна стрічка, що розділена на комірки, та автомат. Так само, як і машина Тьюрінга, машина Поста обробляла одну комірку в будь-який момент часу.

Операція - це одне з наступних:

1) Зсунути на одну комірку вправо (R)
2) Зсунути на одну комірку вліво (L)
3) Записати символ

Дія може бути однією з наступних пар:

1) Пара (операція, номер наступної команди) або HLT (кінець роботи)
2) Розгалудження, або набір пар (зчитати символ, номер наступної команди):

$$\left[\frac{|\,(a1,c1)}{\frac{|\,(a2,c2)}{|(am,cm)}}\right]$$

При розгалудженні усі символи *a1..an* мають бути різними, але не обов'язково використовувати весь алфавіт. Якщо символ є в комірці, то виконується команда *c(i)* (при цьому, нічого в клітинку не записується та автомат нікуди не рухається). Якщо ж жодного з символів немає в комірці, виконується так звана безрезультатна зупинка.

Машина Поста задається набором *номер команди, дія.*

Приклад 1. Машина Поста (теж додавання одиниці в кінець та повернення автомату на початок, e - пустий символ):

1. $(R, 2)$
2. $\left[\,|\,\frac{|\,(1,1)}{|\,(e,3)}\right]$
3. $(1,4)$

4. $(L, 5)$

5. $[\ \frac{|\,(1,4)}{|\,(e,6)}\ ]$

6. $(e, 7)$

7. $(R, 8)$

8. HLT

Приклад 2. Машина Поста для додавання одиниці до двійкового числа:

1. $(R, 2)$

2. $[\ \frac{|\,(1,1)}{|\,(e,3)}\ ]$

3. $(L, 4)$

4. $[\ \frac{|\,(0,5)}{|\,(1,9)}\ ]$

Якщо нуль, записуємо одиницю та рухаємось до початку:

5. $(1,6)$

Рух на початок:

6. $[\ \frac{|(0,7)}{\frac{|(1,7)}{|(e,8)}}\ ]$

7. $(L, 6)$

8. $(R, 12)$

Якщо одиниця, записуємо нуль та шукаємо перший нуль зліва, попутно змінюючи всі одиниці на нулі:

9. $(0,10)$

Якщо знайшли нуль, записуємо одиницю та рухаємось на початок (в 5-й стан):

10. $[\ \frac{|(0,5)}{\frac{|(1,9)}{|(e,11)}}\ ]$

11. $(1,12)$

12. HLT

2.3 Алгоритм Маркова

Андрій Андрійович Марков - радянський математик, що займався логікою та конструктивною математикою, створив так звані нормальні алгоритми. Алгоритм

задається алфавітом та кінцевим набором підстановок. Робота алгоритму складається з двох операторів: оператор розпізнання та оператор підстановки.

Оператор розпізнавання знаходить входження лівої частини підстановки в слові, а оператор пісдтановки замінює цю частину правою частиною підстановки

Нормальний алгоритм Маркова виглядає так:

Алфавіт:

$$A = \{a1..an\}$$

Та множина підстановок:

$$s1 \to r1$$

$$s2 \to r2$$

...

$$sn \to rn$$

Тут, усі символи з *s(i)* та *r(i)* мають бути в алфавіті або бути спеціальним символом закінчення *(.)*

Робота алгоритму:

1) Перебираючи усі підстановки по черзі, знайти першу підстановку, ліва частина якої входить до слова.

2) Замінити на праву частину. У випадку, якщо правва чиастина містить символ закінчення, замінити без цього символу.

3) Почати перебирати усі підстановки з самого початку (перейти до пункту 1).

Робота алгоритму закінчується або коли зустрінеться символ закінчення (після виконання останньої підстановки), або якщо жодна ліва частина підстановки не входить до слова.

Ліва частина може бути пустою. У такому випадку просто зліва до слова дописується права частина підстановки.

Права частина також може бути пустою. У цьому випадку, зі слова зтирається ліва частина підстановки.

Приклад 1. Алгоритм Маркова для додавання 1 (теж додавання одиниці в кінець стрічки з одиниць):

$$1 * \rightarrow * 1$$

$$* \rightarrow .1$$

$$\rightarrow .*$$

Приклад 2. Алгоритм Маркова для додавання одиниці до двійкового числа:

$$A = \{10 * +\}$$

Зміщуємо символ * в кінець (щоб знайти останню цифру):

$$* 1 \rightarrow 1 *$$

$$* 0 \rightarrow 0 *$$

Замінюємо * на + щоб почати зміщувати його вліво для знаходження першого нуля:

$$* \rightarrow +$$

$$1+ \rightarrow +0$$

$$0+ \rightarrow .1$$

Якщо не було жодного нуля (наприклад, число 111), додати одиницю зліва:

$$+0 \rightarrow .10$$

$$\rightarrow *$$

РОЗДІЛ 3. ОПИС РЕАЛІЗАЦІЇ ЕМУЛЯТОРА АЛГОРИТМІВ МАРКОВА

При розробці реалізації емулятора алгоритмів Маркова спочатку було майже до кінця реалізовано побудову емулятора на мові python, середовищем відображення був термінал. Втім, візуалізація залишала очікувати кращого. Тому, було вирішено розробити на мові Swift, середовищем відображення був би будь-який пристрій з операційною системою iOS. Таким чином, під час розробки, я ознайомився з новим для себе способом розробки - написанням мобільного застосунку та з новою мовою - Swift

3.1 Приклад створення та виконання алгоритму Маркова

Таким чином виглядає початковий екран (рис. 3.1). Тут будуть усі створені алгоритми Маркова.

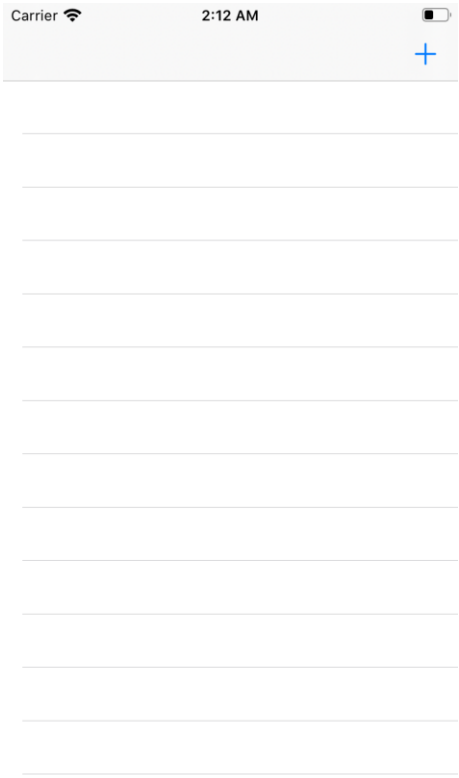

Рис. 3.1 Початковий екран

Далі буде продемонстровано, як можна додати новий алгоритм для додавання одиниці в кінець слова.

Щоб додати новий алгоритм, необхідно натиснути на +.Тут (рис.3.2) можна ввести назву машини та алфавіт, яким має задовольняти правила підстановки (символ . означає кінцевий стан).

Ввівши назву та алфавіт, треба натиснути на кнопку Next:

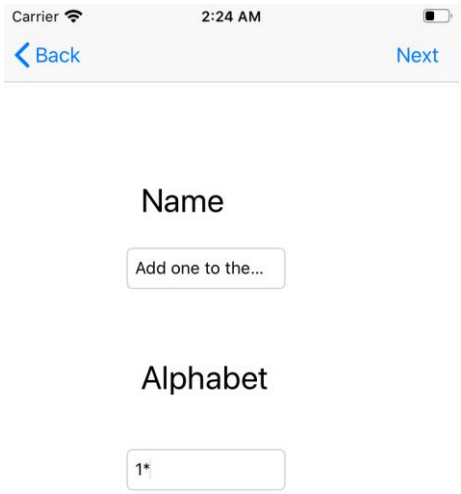

Рис. 3.2 Додавання нового алгоритму

На цьому екрані (рис. 3.3) можна додати нові правила для алгоритму, переглянути правила, які вже є, а також запустити його алгоритм на вконання зі словом Щоб додати правило, треба натиснути Add statement.

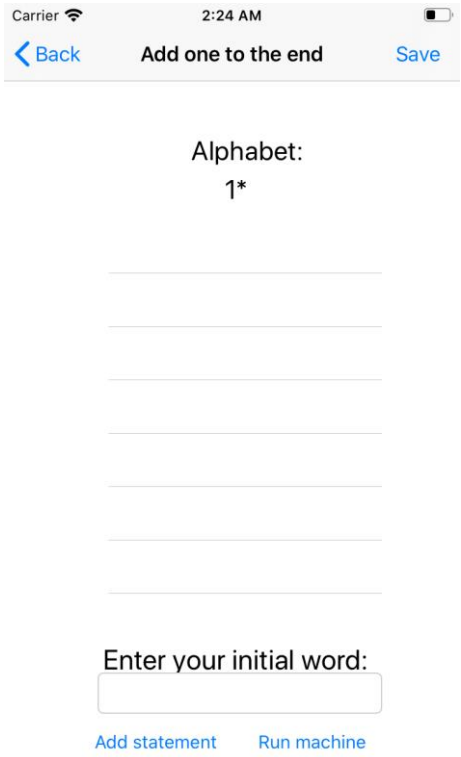

Рис. 3.3 Екран правил алгоритму

Тут (рис.3.4) можна додати правило, обравши початкове та кінцеве слова підстановки. Обидва слова мають задовольняти алфавіт.

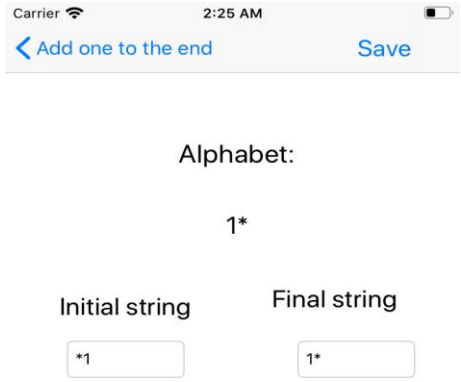

Рис. 3.4 Додавання правила

Якщо слово не задовольнятиме алфавіт, буде попередження про це (рис. 3.5).

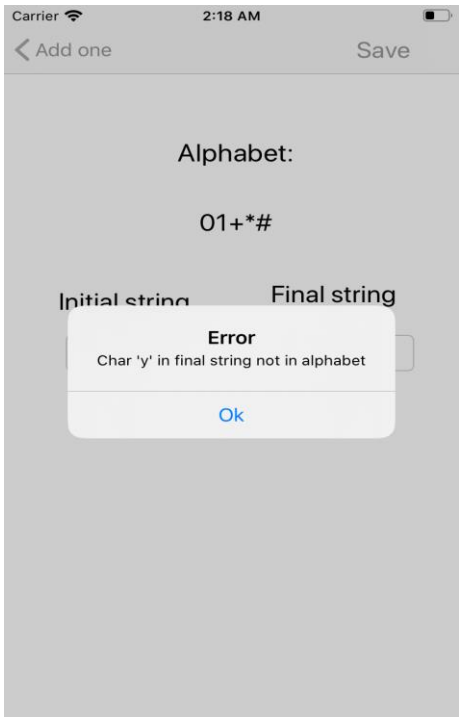

Рис. 3.5 Попередження про незадовільнення алфавіту

Ввівши необхідне правило, треба натиснути на кнопку Save. Правило буде додане до нинішнього алгоритму Маркова. Так (рис.3.6) виглядає наш алгоритм після додавання усіх правил підстановки.

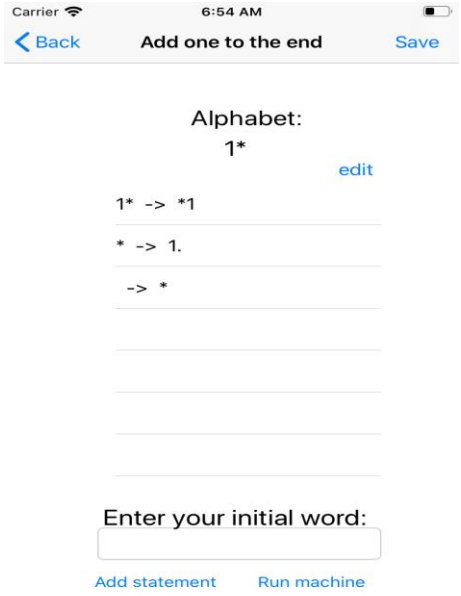

Рис. 3.6 Вигляд алгоритму після додавання усіх правил підстановки

При необхідності, можна змінити порядок правил, або видалити одне з правил, натиснувши на кнопку edit:

Тепер можна запустити його, ввівши початкове слово та натиснувши на кнопку Run Machine (рис.3.7).

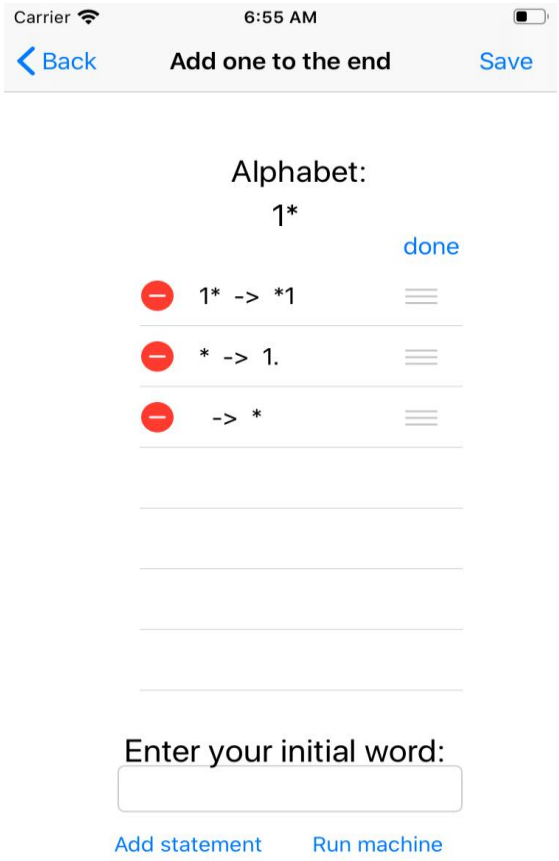

Рис. 3.7 Редагування правил

Тут (рис. 3.8) зверху показаний наш алфавіт, початкове слово та слово, яке зараз змінюється за допомогою правил підстановки. Також, тут показані усі правила підстановки для поточного алгоритму Маркова.

Рис. 3.8 Екран готового алгоритму

Можна обрати один з двох варантів:

1) Покроково його виконувати (натискаючи кнопку next step). Тоді слово, яке зараз змінюється, буде змінюватись кожного разу, та правило, за яким слово було змінено, буде підсвічено (рис. 3.9).

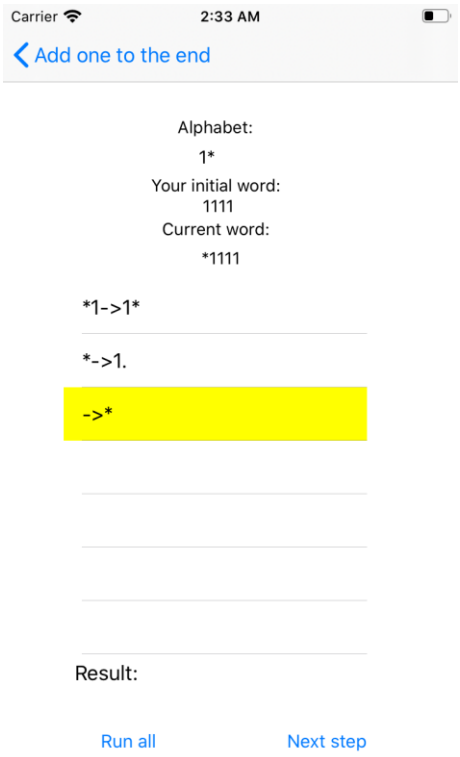

Рис. 3.9 Підсвічене змінене слово

Після останнього правила, буду виведено результат біля поля Result та з’явиться вікно з результатом (рис. 3.10).

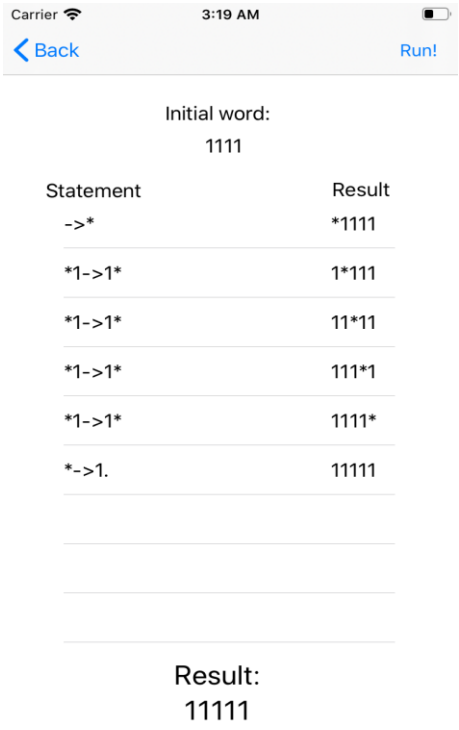

Рис. 3.10 Вікно з результатом

2) Можна також запустити одразу виконання всього алгоритму. Тоді будуть виведені одразу усі правила, за яким слово було змінено та слово, яке вийшло в результаті виконання цього правила. Кінцевий результат буде під полем Result.

Натиснувши Next step після завершення програми, отримаємо вікно, яке каже, що програма вже закінчилась (рис.3.11).

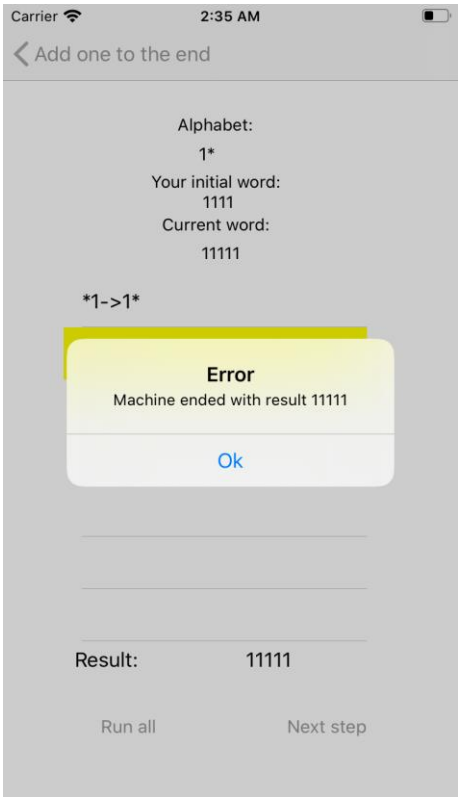

Рис. 3.11 Завершення програми

Усі алгоритми, що були створені, зберігатимуться в локальній пам’яті телефона, тому навіть після закриття та відкриття програми, усі алгоритми будуть збережені.

3.2 Кілька прикладів різних алгоритмів Маркова

Алгоритм, що дозволяє додати одиницю до двійкового числа, виглядає наступним чином (рис. 3.12):

$$A = \{10 * +\}$$

$$+1 \rightarrow 1 *$$

$$* 0 \rightarrow 0 *$$

$$* \rightarrow +$$

$$0+ \rightarrow .1$$

$$1+ \rightarrow +0$$

$$+0 \rightarrow .10$$

$$\rightarrow *$$

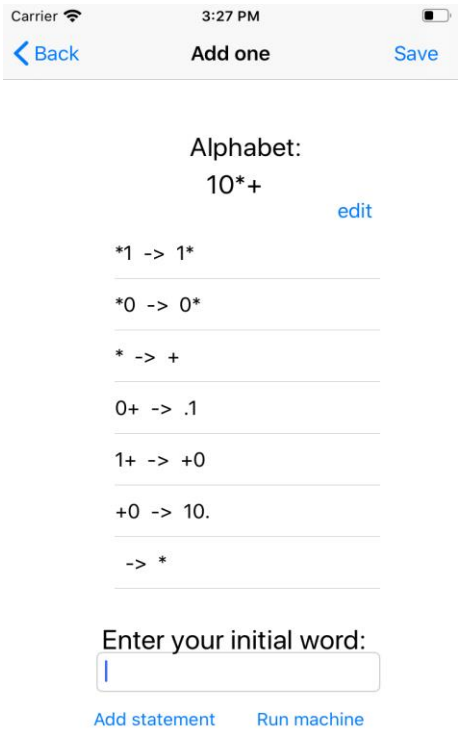

Рис. 3.12 Алгоритм додавання одиниці до двійкового числа

Якщо виконати все одразу зі словом 11011, отримаємо такий результат (рис. 3.13).

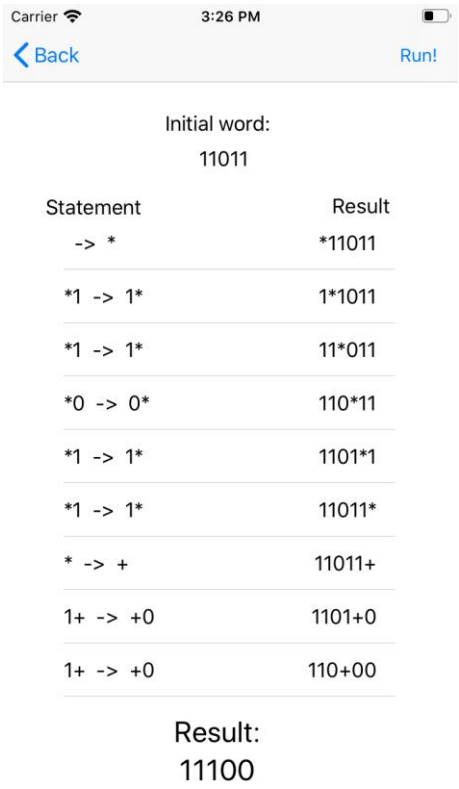

Рис. 3.13 Результат слова 11011

Алгоритм, що віднімає одиницю від двійкового числа (рис. 3.14). Тут символ # необхідний для того, щоб якщо число має вигляд 1 і далі скільки завгодно 0, результатом було *1…1*, а не *01…1*:

$$A = \{10 * \#-\}$$

$$* 1 \rightarrow 1 *$$

$$* 0 \rightarrow 0 *$$

$$*\rightarrow -$$

$$1- \rightarrow \#0$$

$$0- \rightarrow -1$$

$$1\# \rightarrow .1$$

$$0\# \rightarrow .0$$

$$\#0 \rightarrow .$$

$$\rightarrow *$$

Рис. 3.14 Алгоритм, що віднімає одиницю від двійкового числа

Якщо виконати все одразу зі словом 1000, то буде такий результат (рис.3.15).

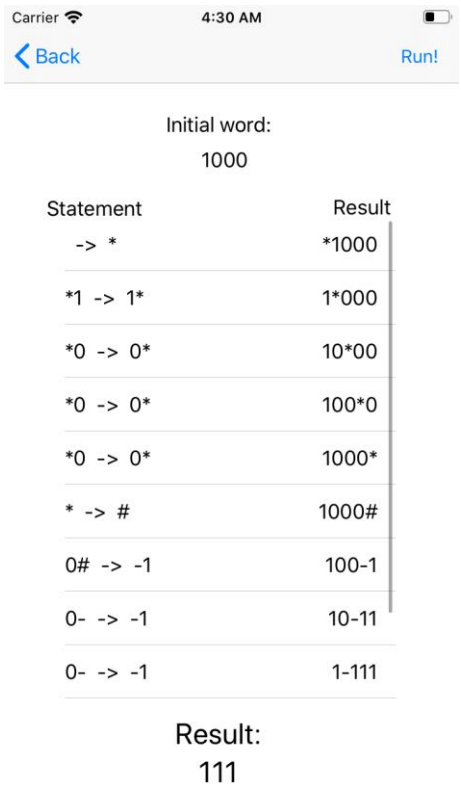

Рис. 3.15 Виконання алгоритму зі словом 1000

Якщо виконати все одразу зі словом 11100, то буде такий результат (рис.3.16).

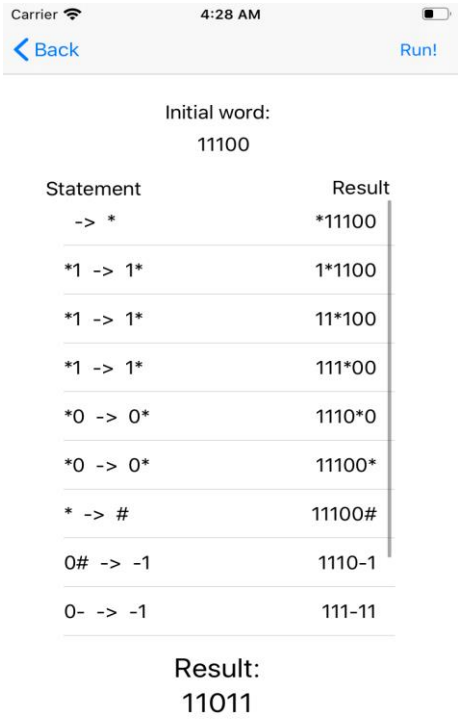

Рис. 3.16 Виконання алгоритму зі словом 11100

А ось приклад (рис.3.17) додавання двох двійкових чисел методом віднімання одиниці від правого та додавання одиниці до лівого, доки справа не буде нуля (числа мають бути записані у форматі *число1*+*число2*):

$$A = \{10MDI+\}$$

$$M0 \to 0M$$

$$M1 \to 1M$$

$$M \to D$$

$$0I \to 1$$

$$1I \to I0$$

$$I \to 1$$

$$0D \to D1$$

$$1D \to 0$$

$$+0 \to +$$

$$+1 \rightarrow I + 1M$$

$$+ \rightarrow$$

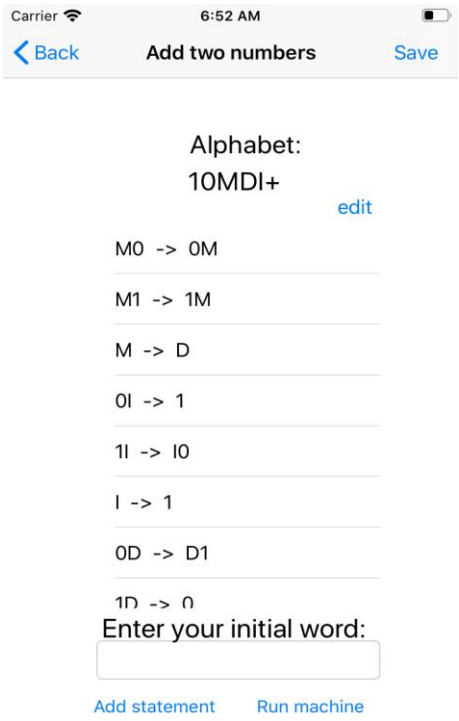

Рис. 3.17 Приклад додавання двох двійкових чисел

Отримаємо такий результат (рис. 3.18), якщо виконати цей алгоритм зі словом *11001+1001*.

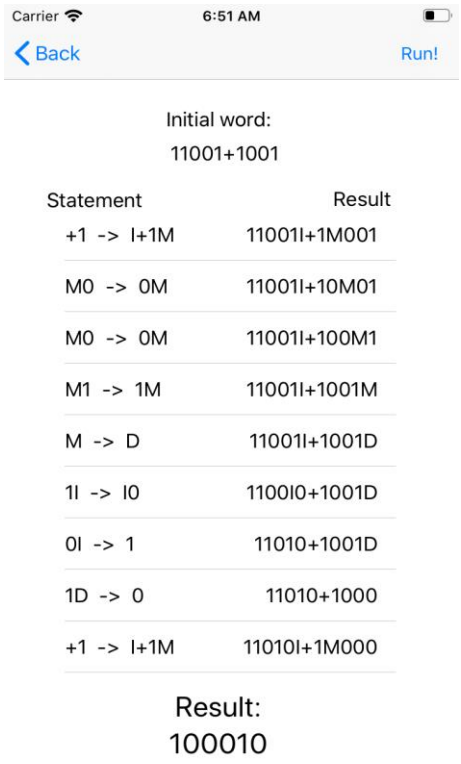

Рис. 3.18 Результат алгоритму зі словом 11001+1001

ВИСНОВКИ

У даній курсовій роботі розглянуті різні класичні алгоритмічні системи (машина Тьюрінга, Поста, алогритми Маркова). Було побудовано рішення для однієї і тої самої задачі в трьох різних системах. Також, було створено емулятор для кожної з них - алгоритмів Маркова

У процесі розробки емулятора було вивчено нову могу програмування Swift.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. https://studfile.net/preview/7423495/page:8/
2. http://mathhelpplanet.com/static.php?p=normalnyye-algoritmy-markova
3. https://ru.wikipedia.org/wiki/Нормальный_алгоритм
4. https://developer.apple.com/documentation/swift
5. https://www.intuit.ru/studies/courses/51/51/lecture/1522
6. http://cmcmsu.info/download/cmc.mt.markov.tasks.pdf
7. https://ru.wikipedia.org/wiki/Алгоритм