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

КЛАСИЧНІ АЛГОРИТМІЧНІ СИСТЕМИ. РОЗРОБКА ЕМУЛЯТОРА МАШИНИ ПОСТА

Текстова частина до курсової роботи
за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи
к.т.н., доц. Франчук О. В.
_____
(підпис)
“____” _____ 2020 р.
Виконала студентка
Шкута А. О.
“____” _____ 2020 р.

Київ 2020

Тема: «Класичні алгоритмічні системи. Розробка емулятора машини Поста»

Календарний план виконання роботи:

	Назва етапу курсової роботи	Термін виконання	Примітка
1.	Отримання завдання	Жовтень 2019	
2.	Огляд літератури за темою роботи	Січень 2020	
3.	Створення практичної частини роботи	Лютий 2020	
4.	Написання текстової частини роботи	Березень 2020	
5.	Надання роботи для попередньої перевірки	Квітень 2020	
6.	Корегування роботи	Квітень 2020	
7.	Захист курсової роботи	Квітень 2020	

Студент _____

Керівник _____

«___» _____ 2020р.

Зміст

Анотація

В теоретичній частині роботи розглядається поняття класичних алгоритмічних систем, історичне підґрунтя для їх виникнення та подальший розвиток. Також більш детально розглядаються деякі конкретні приклади основних типів алгоритмічних систем, такі як машина Тюрінга, рекурсивні функції, нормальні алгоритми Маркова та машина Поста. Для алгоритмів Маркова, машини Тюрінга та машини Поста наводяться ілюстрації, які описуються процес їх роботи та приклади розв’язання деяких задач за допомогою цих систем. В практичній частині роботи створюється емулятор машини Поста з використанням мови програмування C#, а також таких технологій як .Net Framework та WPF.

Вступ

Алгоритмічні системи та їх розвиток мали значний вплив на наукову діяльність. Введення поняття алгоритмічної системи стало підґрунтям для розвитку такої науки як теорія алгоритмів.

Метою теоретичної частини цієї роботи є дослідження класичних алгоритмічних систем, виникнення яких стало значним поштовхом для розвитку математичної науки та інформатики.

Другий, практичний розділ цієї роботи присвячено розробці емулятора машини Поста. Спираючись на зібрані теоретичні відомості про роботу цієї машини буде розроблено та написано алгоритм роботи програми на обраній мові програмування. Також важливою частиною роботи буде розробка зручного, інтуїтивно зрозумілого для користувача вигляду вікна програми.

1 Класичні алгоритмічні системи

1.1 Поняття алгоритмічної системи

Алгоритмічною системою називається загальний спосіб задання алгоритмів. Алгоритмічна система визначає: множину типів об’єктів вхідних і вихідних даних, набір дій користувача необхідних для виконання алгоритму та мову визначення правил, яка повинна бути зрозумілою для користувача і однозначною.

1.2 Історичне підґрунтя

Обчислювальні процеси, які мають алгоритмічний характер, а саме арифметичні дії над цілими числами, знаходження найбільшого спільного дільника двох чисел тощо, відомі людству з найдавніших часів, проте в явному вигляді поняття алгоритму сформувалося лише на початку XX століття.

Неможливість знайти єдиний спосіб розв′язання для деяких задач, пов′язаних передусім з арифметикою та математичною логікою, спровокувала виникнення необхідності уточнення поняття алгоритму.

Також важливою задачею стало доведення того, що алгоритму розв′язання певних задач не існує, а для цього потрібно мати його точне математичне визначення. Тому після визначення поняття алгоритму як окремої сутності, наступним важливим завданням стало створення формальних моделей алгоритму та дослідження їх властивостей. Все це призвело до виникнення теорії алгоритмів як окремого розділу математики.

1.3 Основні типи алгоритмічних моделей

Уточнення поняття алгоритму тісно пов’язане з уточненням алфавіту даних і способом їх подання, розміщенням даних у пам’яті, елементарними кроками алгоритму та механізмами його реалізації. Для уникнення постійного уточнення різноманітних понять, в теорії алгоритмів прийнятний підхід, заснований на конкретній алгоритмічній моделі, усі ці вимоги сформульовані і виконуються.

Існує три основні типи алгоритмічних моделей:

1. Перший тип тлумачить алгоритм як деякий детермінований пристрій, який у кожен момент часу виконує лише конкретну фіксовану множину операцій. Основною теоретичною моделлю такого типу є машина Тюрінга.
2. Другий тип пов’язує поняття алгоритму з процедурами обчислень значень числових функцій. Основною теоретичною моделлю такого типу є рекурсивні функції.
3. Третій тип – перетворення слів у довільних алфавітах, де операції – це заміни одних фрагментів слів іншим словом. Основною теоретичною моделлю такого типу є нормальні алгоритми Маркова.

1.3.1 Машина Тюрінга

Машина Тюрінга – абстрактна обчислювальна машина. Вона була запропонована Аланом Тюрінгом у 1936 році для формалізації поняття алгоритму.

Машина Тюрінга складається з:

1. Нескінченної в обидві сторони стрічки (можливі варіанти машини Тюрінга, які мають декілька нескінченних стрічок), яка розділена на комірки.
2. Керуючий пристрій, який може знаходитися в одному з безлічі станів – автомат (голівка для зчитування або запису, якою керує програма).

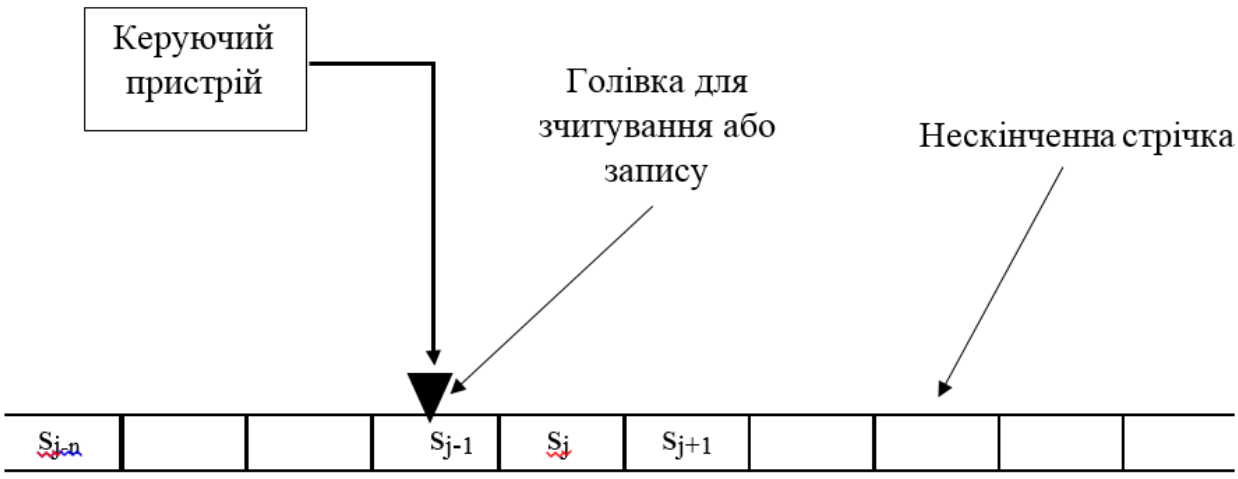

Рис 1. Схема машини Тюрінга

Кількість можливих станів керуючого пристрою точно задано та є скінченним. Керуючий пристрій може рухатися вправо або вліво по стрічці, читати і записувати в комірки стрічки символи деякого скінченного алфавіту.

Щоб задати конкретну машину Тюрінга, потрібно описати для неї такі складові:

1. Зовнішній алфавіт, інакше – алфавіт вхідних символів $A = \{a_0, a_1, \dots, a_m\}$. Скінченна множина (наприклад A), елементи якої називаються буквами (символами). Одна з літер цього алфавіту (наприклад, a_0) повинна бути пустим символом.
2. Внутрішній алфавіт, інакше – алфавіт станів $Q = \{q_0, q_1, \dots, q_p\}$. Скінченна множина станів голівки (автомата). Один із цих станів (наприклад, q_1) повинен бути початковим, тобто таким, перебуваючи у якому машина починає свою роботу. Ще один зі станів алфавіту (наприклад, q_0) кінцевим, тобто таким, після потрапляння в який машина завершує виконання програми.
3. Таблиця переходів. Опис поведінки автомата (голівки) в залежності від стану та зчитаного символу.

В процесі роботи автомат машини Тюрінга може виконувати такі дії:

1. Записувати символ зовнішнього алфавіту у комірку, включаючи пустий символ (фактично – це видалення символу з комірки).
2. Пресуватися на одну комірку вліво або вправо.
3. Змінювати свій внутрішній стан.

Одна команда для машини Тюрінга – якась конкретна комбінація з трьох складових: вказівки який символ потрібно записати у комірку над якою в даний момент часу знаходиться каретка, куди треба її перемістити і в який стан перейти. Важливо також зазначити, що команда може мати не всі складові. Наприклад, не змінювати символ у комірці, не рухатися вліво чи вправо, тобто залишатися на місці або не змінювати внутрішній стан автомату.

Програми для машини Тюрінга записуються у вигляді таблиці (таблиця переходів), де перший стовпчик це усі можливі (в межах заданого внутрішнього

алфавіту) внутрішні стани, а перший рядок – усі можливі символи в межах заданого зовнішнього алфавіту. Вміст таблиці – це команди для машини Тюрінга. Символ в комірці, який зчитує каретка і її внутрішній стан визначають яку команду потрібно виконати. Команда, яку потрібно виконати на наступному кроці визначається перетином символів внутрішнього і зовнішнього алфавітів в таблиці.

Приклад розв'язання задач

Задача: Перенести перший символ непустого слова P в його кінець. Алфавіт A = {a, b, c}.

Таблиця 1. Приклад програми для машини Тюрінга. Таблиця переходів

	a	b	c	Пустий символ
q0	_ > q1	_ > q2	_ > q3	
q1	a > q1	b > q1	c > q1	a _ q0
q2	a > q2	b > q2	c > q2	b _ q0
q3	a > q3	b > q3	c > q3	c _ q0

Примітка до таблиці:

В даній таблиці переходів знак «_», розташований на початку команди позначає пустий символ. Знак «_», розташований на другій позиції в команді (замість позначення кроку вліво чи вправо) означає що каретка залишається на місці.

Пояснення розв'язання задачі:

Якщо перший пустий символ це a, то потрібно перейти у стан q1, у якому автомат рухається вправо, і записує вкінці a. Якщо перший пустий символ це b, то потрібно перейти у стан q2, у якому автомат рухається вправо, і записує вкінці b. Якщо перший пустий символ це c, то потрібно перейти у стан q3, у якому автомат рухається вправо, і записує вкінці c.

1.3.2 Рекурсивні функції

Рекурсивний метод програмування, заснований на організації звернень деяких програм (функції) один до одного і розглядається як альтернатива ітераційному методу.

Визначення поняття рекурсивних функцій є індуктивним і складається з класу базових примітивних функцій і операторів суперпозиції, примітивної рекурсії і мінімізації.

Базовими примітивними функціями є:

1. Нульова функція $O^n(x_1, \dots, x_n) = 0$. Не має аргументів і завжди повертає нуль.
2. Функція слідування $S(x) = x + 1$. Будь-яке натуральне число x співставляється до наступне за ним натурального числа x+1.
3. Функція проектування $I_m^n(x_1, \dots, x_n) = x_m$, де $0 < m \leq n$. Будь-який набір натуральних чисел $x_1, \dots, x_n$ співставляється до числа x_m з цього набору чисел.

Приклади використання рекурсивних функцій

Факторіал ($F(n) = n!$).

Загальне визначення цієї функції: $F(n) = n * (n-1) * \dots * 1,\ 0! = 1$

Рекурсивне визначення:

$$F(n) = \begin{cases} 1, n = 0 \\ n * F(n-1), n > 0 \end{cases}$$

Функція додавання двох натуральних чисел ($Sum(a, b) = a + b$).

$$Sum(x, 0) = I_1^1(x)$$

$$Sum(x, y + 1) = F\big(x, y, Sum(x, y)\big)$$

$$F(x, y, z) = S(I_3^3(x, y, z))$$

Застосовується оператор примітивної рекурсії до функцій $I_1^1(x)$ і F. При цьому F отримується шляхом підстановки основної функції I_3^3 в основну функцію S.

1.3.3 Нормальні алгоритми Маркова

Алгоритмічна система, яка основана на відповідності між словами в абстрактному алфавіті. Нормальні алгоритми Маркова еквівалентні за своїми можливостями машині Тюрінга та машині Поста.

Нормальний алгоритм задає метод перетворення рядків за допомогою системи підстановок. Кожна підстановка складається зі слова-зразку і слова-заміни. На кожному кроці заміни підстановки проглядаються по порядку згори до низу, і виконується перша з них, яка підійшла: перше знайдене слово-зразок у рядку який обробляє програма замінюється на слово-підстановку.

Для формалізації поняття алгоритму А. А. Марков запропонував використовувати асоціативні обчислення. Маємо алфавіт – скінченний набір різних символів, які також можна називати літерами алфавіту. Будь-яка скінченна послідовність літер алфавіту називається словом у цьому алфавіті.

В деякому алфавіті задається скінченна система підстановок N - M, K- L,..., де N, M, K, L, … - слова в цьому алфавіті. Будь-яку підстановку N - M можна застосувати до деякого слова S таким чином: якщо в S міститься одне або декілька входжень N, то будь-яке з них може бути замінене словом M і навпаки, якщо в S міститься одне або декілька входжень M, то будь-яке з них може бути замінене словом N.

Наприклад, в алфавіті A={x, y, z} є слова N = xy, M = z, K = xyzxyz. Якщо замінити в слові K слово N на M, то отримаємо zzxyz або xyzzz, і навпаки замінивши в слово M на N, отримаємо xyxyxyz або xyzxyxy.

Підстановка xy - zz недопустима для слова yxxz, тому що xy і zz не входять у це слово. До отриманих за допомогою допустимих підстановок слів модна застосувати інші допустимі підстановки. Сукупність усіх слів у заданому алфавіті разом із системою підстановок називають асоціативним обчисленням.

Також існує вид асоціативного обчислення, у якому підстановки є орієнтованими: N -> M (підстановку дозволено виконувати лише зліва направо, тобто входження слова N замінюється на M).

Приклад 1. Додавання одиниці до числа у десятковій системі числення.

Дано алфавіт A = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, @}. Число на вхід подається у вигляді слова «<число>@». Тобто остання цифра відрізняється тим, що після неї стоїть символ «@».

Першими повинні бути підстановки типу <число>@ -> <число + 1>. Таким чином, якщо остання цифра числа менша від 9, ми просто збільшуємо її на одиницю. У випадку ж якщо остання цифра числа це 9, потрібно її замінити на 0 (за це відповідає підстановка 9@ -> @0) і додати одиницю до старшого розряду (знову використовується перша частина підстановок). Зрештою, якщо найстарший розряд числа дорівнює 9, то перед цим числом потрібно поставити 1 (підстановка @@ -> 1), інакше символ @ потрібно стерти (підстановка @ -> _).

Таким чином отримуємо нормальний алгоритм Маркова для збільшення числа на 1:

0@ -> 1

1@ -> 2

2@ -> 3

3@ -> 4

4@ -> 5

5@ -> 6

6@ -> 7

7@ -> 8

8@ -> 9

9@ -> @0

@@ -> 1

@ -> _

Виконання алгоритму для числа 49: @49@ -> @4@0 -> @50 -> 50.

Виконання алгоритму для числа 999: @999@ -> @99@0 -> @9@00 -> @@000-> 1000.

1.4 Машина Поста

Машина Поста – це абстрактна (неіснуюча у реальності) обчислювальна машина, яку було створено для уточнення (формалізації) поняття алгоритму. Ця

машина – універсальний виконавець, який дозволяє вводити дані та зчитувати результат виконання програми.

У 1936 році американський математик Еміль Пост описав систему, яка має алгоритмічну простоту та може визначити, чи існує алгоритмічне рішення для задачі. Якщо алгоритмічне рішення для задачі існує, то його можна представити у формі команд для машини Поста.

Пост сприймає загальну велику проблему у вигляді великої кількості менших конкретних проблем. При цьому, рішення яке розв′язує велику загальну проблему, є рішенням і для кожної меншої конкретної проблеми.

Наприклад, розв′язання рівняння 3+4*x=0 це одна з конкретних, менших проблем, а рішення рівняння a+b*x=0 – це загальна проблема.

Машина Поста складається з:

1. Нескінченної стрічки, яка поділена на безліч однакових комірок (секцій). Секція може бути пустою (0 або нічого), або містити мітку (1 або будь-який інший символ).
2. Голівки (каретки), яка може пересуватися по стрічці на одну комірку вправо, або вліво. Також ця каретка може перевірити, чи записана у комірці мітка, чи вона пуста, а також може поставити мітку, або стерти її.

Поточний стан машини Поста визначається станом стрічки та положенням каретки. Стан стрічки – інформація про те, які комірки на стрічці пусті, а які відмічені. Крок – це рух каретки на одну секцію вправо або вліво. Стан стрічки може змінюватися в процесі виконання програми. Число команд програми називається довжиною програми.

Будь-який алфавіт, може бути закодований за допомогою двох знаків, тому для роботи машини Поста достатньо лише двох різних символів (є мітка, немає мітки). В залежності від алфавіту збільшуватися може тільки кількість двійкових символів в літері алфавіту.

Діями каретки керує програма, яка складається зі стрічок команд. Синтаксис кожної команди: i, K j, де i – номер команди, K – дія каретки, j – номер наступної команди (посилання, перехід на команду стрічку під номером j).

Всього для машини Поста існує шість типів команд:

1. 1 j – поставити мітку, перейти до j-го рядку програми.
2. 0 j – стерти мітку, перейти до j-го рядку програми.
3. < j – пересунути каретку на одну комірку вліво, перейти до j-го рядку програми.
4. > j – пересунути каретку на одну комірку вправо, перейти до j-го рядку програми.
5. ? j_1; j_2 – якщо в комірці немає мітки, то перейти до j_1-го рядку програми, інакше перейти до j_2-го рядку програми.
6. ! – кінець програми (стоп). Команда стоп не має посилання на інший рядок програми.

Стан стрічки до та після виконання команд:

1. Поставити мітку:

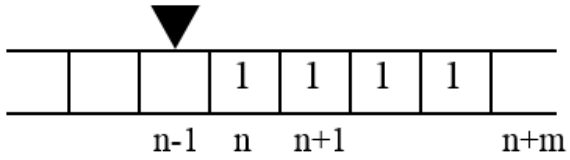

Рис 2. Поставити мітку. Стан стрічки до виконання команди

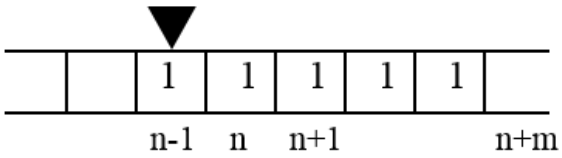

Рис 3. Поставити мітку. Стан стрічки після виконання команди

2. Стерти мітку:

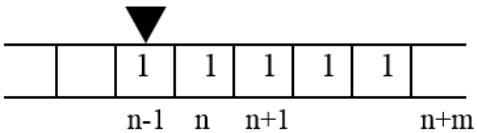

Рис 4. Стерти мітку. Стан стрічки до виконання команди

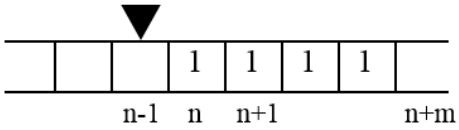

Рис 5. Стерти мітку. Стан стрічки після виконання команди

3. Пересунути каретку на одну комірку вправо:

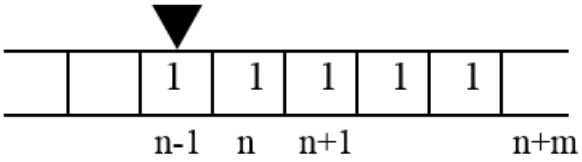

Рис 6. Крок вправо. Стан стрічки до виконання команди

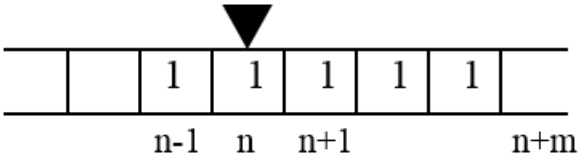

Рис 7. Крок вправо. Стан стрічки після виконання команди

4. Пересунути каретку на одну комірку вліво:

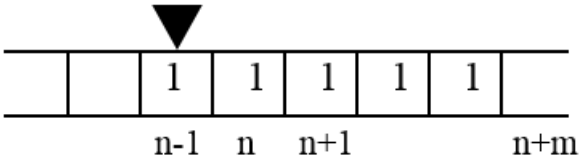

Рис 8. Крок вліво. Стан стрічки до виконання команди

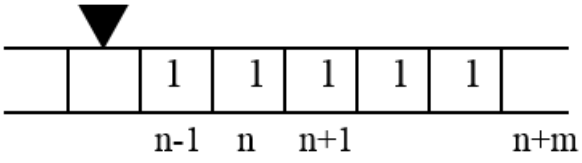

Рис 9. Крок вліво. Стан стрічки після виконання команди

Варіанти закінчення виконання програми на машині Поста:

1. Команда «стоп» - коректна зупинка виконання програми. Така ситуація виникає в результаті виконання правильно написаного алгоритму.
2. Виконання недопустимої команди – безрезультатна зупинка. Випадки коли каретка повинна записати мітку там, де вона вже є, або стерти мітку

там, де її немає вважаються недопустимими. В такому разі програма закінчує своє виконання аварійно.

3. Нескінченне виконання програми, виникнення нескінченного циклу. В результаті виконання алгоритму програми машина Поста може взагалі не зупинитися. Тобто ніколи не дійти до команди «стоп» і ніколи не завершити своє виконання аварійною ситуацією.

Елементарні дії (виконання команд) машини Поста простіші ніж команди машини Тюрінга. Тому програми для машини Поста мають більшу кількість команд, ніж аналогічні до них, програми для машини Тюрінга.

Початковим станом каретки зазвичай вважають її становище проти порожньої комірки зліва від найбільш лівої комірки, в якій стоїть мітка.

Число k представляється на стрічці машини Поста як k+1 число міток, що розташовані підряд. Одна мітка означає число «0». Зазвичай між двома числами робиться інтервал в як мінімум одну порожню комірку.

Наприклад, запис чисел 2 і 3 на стрічці машини Поста буде виглядати наступним чином:

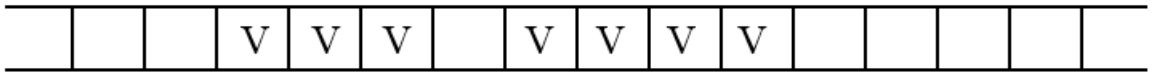

Рис 10. Запис чисел 2 і 3 на стрічці машини Поста

Приклад розв’язання задач

Задача 1. збільшити число 4 на одиницю (змінити значення в пам’яті з 4 на 5). Ціле додатне число на стрічці машини Поста представлено мітками, які розташовані одна за одною в комірках.

Припустимо, що знаходиться зліва від міток на пустій комірці. Тоді програма збільшення числа на одиницю може мати наступний вигляд:

1. > 2
2. ? 1,3
3. < 4
4. 1 5
5. !

Процес виконання (число, до якого треба додати одиицю представлене у вигляді P + 1 одиниць, початкове положення каретки позначено символом «v».):

v

0011111000 (виконується команда 1 – крок вправо)

v

0011111000 (виконується команда 2 – перевірка, чи комірка пуста; пуста – отже виконується перехід до команди 1)

v

0011111000 (виконується команда 1 – крок вправо)

v

0011111000 (виконується команда 2 – перевірка, чи комірка пуста; непуста – отже виконується перехід до команди 3)

v

0011111000 (виконується команда 3 – крок вліво і перехід до команди 4)

v

0111111000 (виконується командра 4 – запис мітки, перехід до команди 5 – зупинка виконання програми)

Задача 2. Віднімання натуральних чисел P – Q

Далі для демонстрації поточного стану стрічки для відображення цілого натурального числа P використовуємо набір з P + 1 одиниць. Числа на стрічці розділяються нулем. Початкове положення каретки позначено символом «v».

v

001111011000

P Q

Програма віднімання складається з послідовного видалення крайніх лівих міток у числі Q і правих у числі P:

1. 0 2 - видаляємо лівий символ у Q
2. > 3 - крок вправо
3. ? 4,5 - якщо в комірці, на якій знаходиться каретка немає мітки – переходимо до пункту 4, інакше до пункту 5

4. ! - стоп, якщо видалили Q (Q=0)

5. < 6 - крок вліво

6. ? 5, 7 - цикл пошуку P

7. 0 7 - витираємо правий символ у P

8. > 9 - крок вправо

9. ? 8, 1 - пошук Q

Для коректного виконання даної програми вхідні числа повинні задовольняти умову P > Q.

2 Розробка емулятора машини Поста

Метою практичної частини курсової роботи була розробка програми, яка б симулювала роботу машини Поста.

Також важливим завданням було створити зручний користувацький інтерфейс, і особливу увагу приділити валідації вхідних даних.

Перелік використаних засобів і технологій: мова програмування C#, .Net Framework, WPF.

2.1 Користувацький інтерфейс і функціонал програми

Для відпрацювання програми користувачу необхідно ввести за бажанням умову задачі, стан стрічки і безпосередньо кроки алгоритму.

Умову задачі можна вводити у довільному форматі. Це такі собі коментарі для користувача, якими він міг би користуватися при повторному використанні написаної програми.

Для уникнення помилок виконання стан стрічки слід записувати у вигляді послідовних цілих чисел розділених пробілом, або знаками «,» чи «;».

Програму слід записувати такому форматі:

[номер рядку програми]. [команда] [номер наступного рядку команди]

Кожен рядок у програмі повинен мати номер [номер рядку програми]. При вводі даних бажано виділити цей номер крапкою, як показано у зразку. Це номер, за яким програма знаходить наступний номер ([номер наступного рядку команди]), який потрібно виконати далі.

Команди ([команда]) задаються відповідно до команд, перелічених у пункті 1.4 «Типи команд». Кожній команді відповідає декілька символів:

1. Крок каретки вліво – “l”, “L”, ”left”, “LEFT”, “>”.
2. Крок каретки вправо – “r”, “R”, “right”, “RIGHT”, ”>”.
3. Поставити мітку – “1”.
4. Видалити мітку – «0».
5. Перевірка (вибір наступної команди в залежності від значення у комірці) – ?.
6. Зупинка виконання програми – !.

Кожен рядок програми записується на новому рядку у полі вводу.

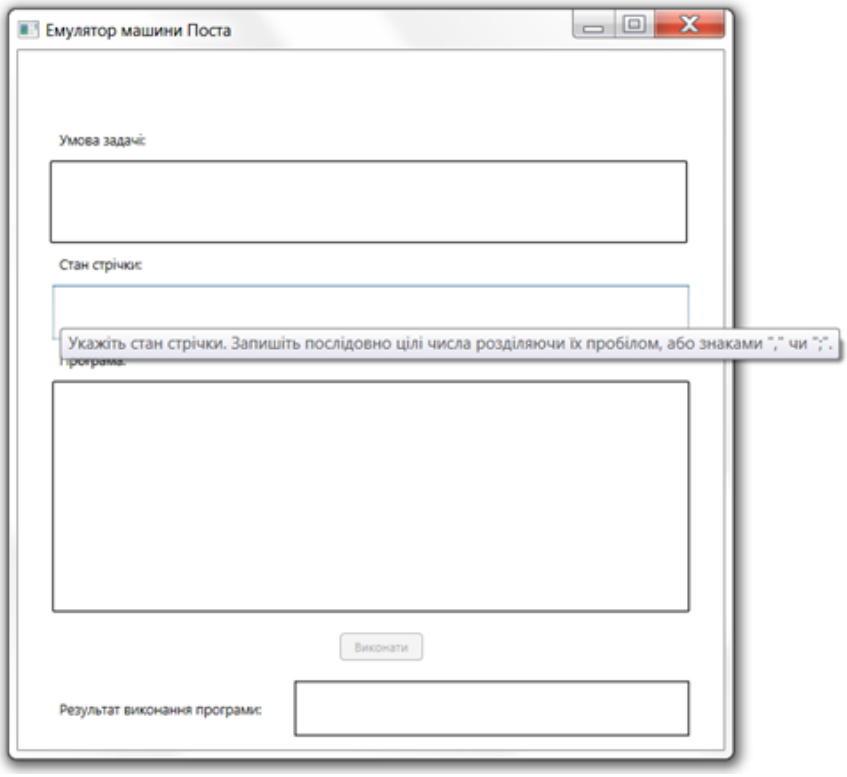

Рис 11. Користувацький інтерфейс. Стан стрічки. Підказка

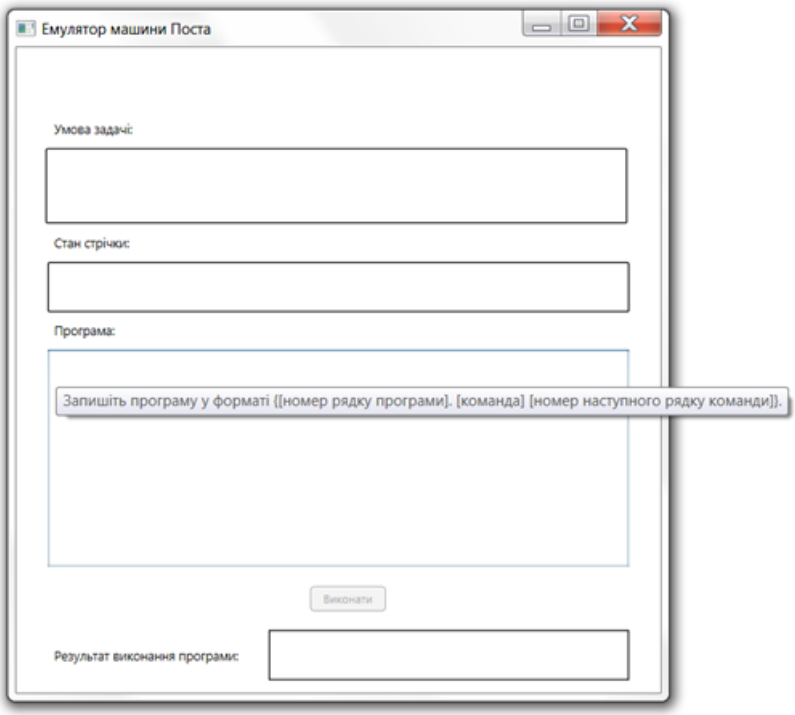

Рис 12. Користувацький інтерфейс. Програма. Підказка

Кнопка виконати буде доступною для натискання коли користувач введе усі дані, необхідні для виконання програми. А саме, стан стрічки та безпосередньо команди програми. Коментарі вводити необов’язково.

У випадку успішного виконання програми, у віконці «Результат виконання програми» з’являться послідовні цілі числа розділені пробілом. Кожне таке ціле число це P-1 послідовно розташованих одиниць розділених комірками з пустими мітками на стрічці «машини Поста».

У випадку виникнення помилки, у віконці «Результат виконання програми» з’являться повідомлення про помилку. Повідомлення можуть бути таких видів:

1. <invalid line input> - помилка вводу стану стрічки
2. <command number parse error> - помилка вводу номеру рядку програми
3. <command transition parse error> - помилка вводу переходу на наступний рядок команди
4. <command value parse error> - помилка ввозу значення команди ([команда])

Усі ці помилки пов’язані з неправильним форматом вводу даних.

Рис 13. Помилка виконання програми

Крім того, також можливе безрезультатне завершення програми у випадку, якщо виникне ситуація, коли потрібно буде поставити мітку туди, де вона вже стоїть, або ж видалити з комірки, у якій мітки немає.

Стрічка машини Поста у програмі представлена у вигляді масиву даних типу bool. Його розмір визначається в залежності від введених даних. Якщо користувач введе у «Стан стрічки» числа 25 і 13, то розмір масиву буде дорівнювати (25+13)*3=114 елементів. У випадку, якщо це значення буде меншим за сто, розмір масиву дорівнюватиме 100. Числа введені у «Стан стрічки» записуються послідовно через 1 елемент масиву у вигляді послідовності значень «true» починаючи від початку масиву.

Положення каретки зберігається у окремій змінній типу int. На початку виконання програми вона знаходиться на першому елементі масиву. У випадку, якщо знаходячись над першим елементом масиву, надійде команда рухатися вліво, каретка переміститься на останній елемент масиву, і навпаки, отримавши команду рухатися вправо, знаходячись на останньому елементу масиву, каретка переміститься на перший елемент масиву. Таким чином масив даних типу bool утворює умовно нескінченну стрічку. Крім того, якщо ціле число у результаті виявиться розбитим на частини – одна частина знаходитиметься на початку масиву, а інша у кінці, то це число буде з’єднано перед виводом результатів роботи алгоритму.

Приклад виконання 1. Необхідно додати одиницю до числа. Програма для машини Поста записана у віконці «Програма» на Рис. 14. Перевірку роботи алгоритму виконуємо на числі 7.

Рис 14. Додавання одиниці до числа

Приклад виконання 2. Віднімання двох цілих чисел. Перевірку роботи алгоритму виконано на числах 21 і 5.

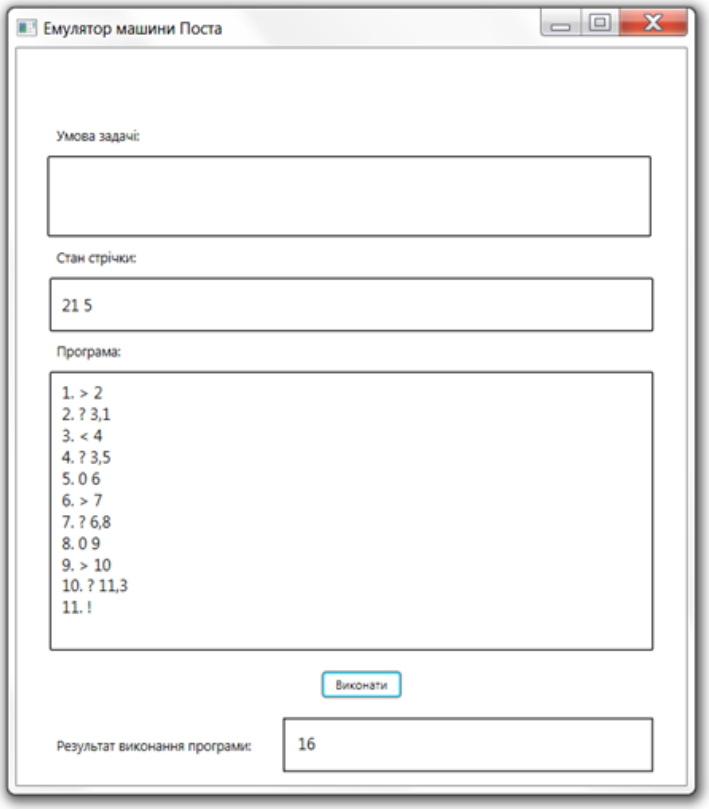

Рис 15. Віднімання двох чисел

Висновки

В результаті роботи над практичною частиною курсової роботи було досліджено поняття класичних алгоритмічних систем та більш детально розглянуто деякі типи алгоритмічних моделей, таких як машина Тюрінга, рекурсивні функції, нормальні алгоритми Маркова та машину Поста.

Під час роботи над практичною частиною, було розроблено застосунок, що емулює роботу машини Поста, було розроблено алгоритм програми та враховано всі можливі випадки закінчення її роботи, а також обмеження, які накладаються на дані, введені користувачем застосунку.

Література

1. Ахо, А. Структури даних і алгоритми / Ахо А., Хопкрофт Дж., Ульман Дж.

2. Вирт Н. Алгоритми і структури даних / Н. Вирт

3. Г. Ч. Курінний, «Обчислюваність: машини Тьюрінга та рекурсивні функції»

4. В. Г. Абрамов, «Машина Тьюринга и алгоритмы Маркова. Решение задач»

5. С. М. Левицька, «Теорія алгоритмів»

Додаток А. Програмний код

Програма знаходження результату роботи програми для машини Поста

```
class PostMachine
   {

      #region Fields

      private Dictionary<int, CommandStep> _program;
      private Boolean[] _line;

      private int[] _results;
      private string _errorMessage;

      #endregion



      #region Properties

      public Dictionary<int, CommandStep> Program
      {
         get { return this._program; }
         set { this._program = value; }
      }

      public Boolean[] Line
      {
         get { return this._line; }
         set { this._line = value; }
      }

      public int[] Results
```

```
    {
        get { return this._results; }
        set { this._results = value; }
    }

    public string ErrorMessage
    {
        get { return this._errorMessage; }
        set { this._errorMessage = value; }
    }

    #endregion

    public PostMachine(int lineLength, int[] lineState, Dictionary<int,
CommandStep> program) //List<Tuple<int, CommandStep>> program)
    {
        //setup line start state
        Line = new Boolean[lineLength];
        int curr = 0;
        foreach(int i in lineState)
        {
            for(int j=0; j<i; j++)
            {
                Line[curr] = true;
                curr++;
            }
            Line[curr] = true;
            curr+=2;
        }
```

```
    Program = program;
}

public bool ExecuteProgram()
{
    //start with the first command in program
    int currCommandNumber = Program.ElementAt(0).Key;
    CommandStep currCommand = Program.ElementAt(0).Value;
    int currPointerPosition = 0;
    int counter = 1;
    while (true)
    {
        Console.WriteLine("step " + counter);
        counter++;
        switch (currCommand.CommandValue)
        {
            case CommandStep.Command.RIGHT:
                if(currPointerPosition + 1 < Line.Length)
                {
                    //just move one step to the right
                    currPointerPosition++;
                }else
                {
                    //move on the first element of Line array
                    currPointerPosition = 0;
                }
                if (!Program.Keys.Contains(currCommand.Transition1))
                {
                    _errorMessage = "Перехід на неусnуючий номер команди";
                    return false;
```

```
                }
                Program.TryGetValue(currCommand.Transition1, out currCommand);
                break;
            case CommandStep.Command.LEFT:
                if (currPointerPosition != 0)
                {
                    //just move one step to the left
                    currPointerPosition--;
                }
                else
                {
                    //move on the last element of Line array
                    currPointerPosition = Line.Length - 1;
                }
                if (!Program.Keys.Contains(currCommand.Transition1))
                {
                    _errorMessage = "Перехід на неіснуючий номер команди";
                    return false;
                }
                Program.TryGetValue(currCommand.Transition1, out currCommand);
                break;
            case CommandStep.Command.PUT_MARK:
                if (Line[currPointerPosition])
                {
                    //mark is already exist -> throw exception
                    ErrorMessage = "Помилка! Намагаетеся додати мітку в комірку,
яка вже відмічена.";
                    return false;
                }else
                {
```

```
                Line[currPointerPosition] = true;
            }
            if (!Program.Keys.Contains(currCommand.Transition1))
            {
                _errorMessage = "Перехід на неіснуючий номер команди";
                return false;
            }
            Program.TryGetValue(currCommand.Transition1, out currCommand);
            break;
        case CommandStep.Command.DELETE_MARK:
            if (!Line[currPointerPosition])
            {
                //mark is already deleted -> throw exception
                ErrorMessage = "Помилка! Намагаетеся видалити мітку в пустій
комірці.";
                return false;
            }
            else
            {
                Line[currPointerPosition] = false;
            }
            if (!Program.Keys.Contains(currCommand.Transition1))
            {
                _errorMessage = "Перехід на неіснуючий номер команди";
                return false;
            }
            Program.TryGetValue(currCommand.Transition1, out currCommand);
            break;
        case CommandStep.Command.QUESTION:
```

```
                if (!Program.Keys.Contains(currCommand.Transition1) ||
!Program.Keys.Contains(currCommand.Transition2))
                {
                    _errorMessage = "Перехід на неіснуючий номер команди";
                    return false;
                }
                if (Line[currPointerPosition])
                {
                    Program.TryGetValue(currCommand.Transition2, out
currCommand);
                }else
                {
                    Program.TryGetValue(currCommand.Transition1, out
currCommand);
                }

                break;
            case CommandStep.Command.STOP:
                ParseResults();
                return true;
            default:
                //throw exc
                ErrorMessage = "Помилка! Щось пішло не так. Перевірте формат
вводу команд.";
                break;
        }
    }
}
```

```
private void ParseResults()
{
    List<int> results = new List<int>();
    int currNumber = 0;
    foreach(Boolean mark in Line)
    {
        if (mark)
        {
            currNumber++;
        }else
        {
            if(currNumber >= 1)
            {
                results.Add(currNumber);
                currNumber = 0;
            }
        }
    }
    if(currNumber > 0)
    {
        results.Add(currNumber);
    }

    if(Line[0] && Line[Line.Length - 1])
    {
        Results = new int[results.Count - 1];
        Results[0] = (results.First() + results.Last()) - 1;
    }else{
        Results = new int[results.Count];
        Results[0] = results.First() - 1;
```

```
            }
            for(int i=1; i < Results.Length; i++){
                Results[i] = results.ElementAt(i) - 1;
            }
        }

    }
```

Клас, який описує об’єктне представлення однієї стрічки програми

```
class CommandStep
    {
        public enum Command
        {
            LEFT,
            RIGHT,
            PUT_MARK,
            DELETE_MARK,
            QUESTION,
            STOP
        }

        #region Fields

        private Command _commandValue;
        private int _transition1;
        private int _transition2;

        private string _comment;

        #endregion
```

```
#region Properties

internal Command CommandValue
{
   get { return this._commandValue; }
   set { this._commandValue = value; }
}
internal int Transition1
{
   get { return this._transition1; }
   set { this._transition1 = value; }
}
internal int Transition2
{
   get { return this._transition2; }
   set { this._transition2 = value; }
}

internal string Comment
{
   get { return this._comment; }
   set { this._comment = value; }
}

#endregion
  }
```