МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ПРИКЛАДНОЇ МАТЕМАТИКИ ТА КІБЕРБЕЗПЕКИ

О. М. Данильчук

МАТЕМАТИЧНА ЛОГІКА ТА ТЕОРІЯ АЛГОРИТМІВ (частина 1)

Методичні рекомендації для самостійної роботи
для здобувачів вищої освіти ОС «Бакалавр»
спеціальності Е7 Математика

Вінниця
2025

УДК 510.6:519.16(075.4)
М 34

Затверджено на засіданні вченої ради
факультету інформаційних і прикладних технологій
Донецького національного університету імені Василя Стуса
(протокол № 2(6) від 19 грудня 2024 р.)

Укладач:
Оксана ДАНИЛЬЧУК, канд. пед. наук, доцент кафедри прикладної математики та кібербезпеки ДонНУ імені Василя Стуса.

Рецензенти:
Тарас ФУРМАН, канд. пед. наук, доцент кафедри маркетингу та бізнес-аналітики ДонНУ імені Василя Стуса;
Леся ВОТЯКОВА, канд. фіз-мат. наук, доцент кафедри алгебри і методики навчання математики ВДПУ імені Михайла Коцюбинського.

М 34 Математична логіка та теорія алгоритмів (частина 1): методичні рекомендації для самостійної роботи для здобувачів вищої освіти ОС «Бакалавр» спеціальності Е7 Математика / уклад. О. М. Данильчук. Вінниця: ДонНУ імені Василя Стуса, 2025. 68 с.

Методичні рекомендації охоплюють матеріал за темами про основні поняття алгоритму, алгоритмічними моделями, а саме машиною Тюрінга, Поста та Маркова. Виклад теоретичного матеріалу ілюстрований прикладами. В кожній темі після теоретичного блоку наведені запитання для самоконтролю та вправи для самостійного розв'язання.

Матеріал методичних рекомендацій можуть використовувати студенти всіх спеціальностей денної та заочної форм навчання, які вивчають математичну логіку та теорію алгоритмів. Він може бути корисним вчителям математики та інформатики.

УДК 510.6:519.16(075.4)

ЗМІСТ

ВСТУП

У сучасному інформаційному суспільстві логічне мислення, здатність до формалізації знань і вміння знаходити ефективні способи розв'язання задач є ключовими компетентностями фахівця будь-якої галузі. Зокрема, у сфері інформаційних технологій, природничих і технічних наук ці навички мають визначальне значення. Саме тому дисципліна **«Математична логіка та теорія алгоритмів»** займає особливе місце у системі математичної та інформатичної освіти. Ця дисципліна становить теоретичний фундамент інформатики, програмування, штучного інтелекту та моделювання складних систем.

Математична логіка є розділом математики, що досліджує закони й форми правильного мислення за допомогою математичних методів. Вона дає змогу формалізувати міркування, будувати точні докази, строгі логічні міркування, що лежать в основі будь-яких наукових досліджень і технічних розробок, виявляти структуру аргументації та перевіряти істинність висловлювань незалежно від їх конкретного змісту. Знання принципів логічного аналізу дає змогу студентам розвивати критичне мислення, точність у формулюваннях і послідовність у міркуваннях як у науковій, так і в практичній діяльності.

Теорія алгоритмів є теоретичною основою інформатики й програмування, яка розглядає методи розв'язання задач у точних, формалізованих формах. Знання принципів побудови та аналізу алгоритмів дає змогу розробляти ефективні комп'ютерні програми, оптимізувати процеси обчислень і прийняття рішень. Вона вивчає способи опису, побудови, аналізу та класифікації алгоритмів – послідовностей дій, спрямованих на досягнення певного результату. Поняття алгоритму є центральним у сучасній науці, оскільки будь-який процес обробки інформації, від обчислень до штучного інтелекту, може бути поданий у вигляді алгоритмічної структури.

Логіка – наука про закони та помилки. Зародження логіки розпочалося у VI ст. до н. е. (Фалес, Парменід). Подальший розвиток вона отримала завдяки Арістотелю, який створив аксіоматичний метод, запропонував першу формальну систему логіки – силогістику, і заклав основи модальної логіки. Після Арістотеля великий крок у розвитку логіки був зроблений аж у XVII ст. – Г. Лейбніцом (1646–1716), який розвинув ідею створення універсального логічного рахунку, що далеко обігнала свій час.

Математична логіка – наука про закони математичної помилки. Поняття та методи математичної логіки дають обґрунтування правильності міркувань і допомагають знаходити способи відокремлення істинного знання від хибного.

Предметом математичної логіки є математичні теорії, що описують побудову та використання логіко-математичних мов. Алгебру логіки запропонував Дж. Буль

(1815–1864). Логіку математичних міркувань і її теоретичні засади було розкрито в роботах Г. Фреге (1848–1925), який уперше системно описав поняття предикатів і кванторів.

Вивчення зазначених дисциплін сприяє формуванню системного підходу до розв'язання задач, уміння моделювати складні процеси, оптимізувати обчислювальні ресурси, а також застосовувати математичні методи до реальних проблем. Саме поєднання логічного та алгоритмічного мислення створює основу для розвитку сучасного програмування, комп'ютерних наук і цифрової освіти.

Метою методичних рекомендацій є *систематизація основних понять, методів і принципів математичної логіки та теорії алгоритмів,* а також формування у студентів цілісного уявлення про структуру формальних систем і процес побудови алгоритмів, а також розвиток умінь практичного застосування теоретичних знань. Матеріали спрямовані на розвиток аналітичного мислення, уміння формалізувати задачі та обґрунтовувати їх розв'язання.

«Математична логіка та теорія алгоритмів» – одна з основних фундаментальних дисциплін у загальнонауковій підготовці студентів за спеціальністю Е7 Математика. Дисципліна «Математична логіка і теорія алгоритмів» є базовою для таких дисциплін: «Мови програмування» (розділ «Теорія алгоритмів»), «Спеціалізовані мови програмування» (розділ «Алгебра предикатів»), «Експертні системи» (розділ «Автоматичне доведення теорем») та інших. Під час вивчення курсу використовуються основні результати дисциплін «Дискретна математика», «Математичний аналіз» та «Лінійна алгебра».

До кожної теми у методичних рекомендаціях подано формулювання означень основних понять, теорем та інших фактів, знання яких необхідне для опанування цієї теми. Наведено розв'язання прикладів із детальним поясненням, що дасть змогу студентам самостійно опрацьовувати матеріал і краще розуміти деякі традиційно складні для студентів теми. До кожної теми розділів підібрані вправи для самостійного розв'язування. Означення й теореми проілюстровано прикладами. Прості твердження та твердження, що можуть бути доведеними за аналогією, запропоновані як вправи для самостійної роботи. Послідовність і стиль подання матеріалу повністю відповідають силабусу дисципліни та задовольняють потреби суміжних і прикладних дисциплін. Їх кількість достатня для роботи на практичних заняттях та виконання поточних домашніх завдань. Кожна тема завершується запитаннями для самоконтролю, що допоможе студентам під час підготовки до модульних та підсумкового контролів.

Методичні матеріали містять завдання для самостійної роботи й питання для контролю знань. Особливу увагу приділено взаємозв'язку між логічними формами мислення та алгоритмічними процесами, що є фундаментом сучасного програмування й штучного інтелекту.

ТЕМА 1
ПОНЯТТЯ АЛГОРИТМУ

1.1. Інтуїтивне поняття алгоритму.
- 1.1.1. Властивості алгоритму.
- 1.1.2. Алфавіти і слова.
- 1.1.3. Кодування та нумерація.
- 1.1.4. Проблема еквівалентності слів. Словникові примітивно рекурсивні функції.

1.1. Інтуїтивне поняття алгоритму

Одним зі значних досягнень науки XX ст. є теорія алгоритмів – нова математична дисципліна. Одним з найбільш істотних факторів, що визначають значення математики та її місце в сучасній науці, є точність і висока ступінь універсальності образотворчих засобів, якими вона володіє. Ці засоби дають змогу будувати уточнені моделі різноманітних за своєю природою фундаментальних наукових понять, зокрема й математичних.

У теорії алгоритмів та інформаційних науках взагалі відсутнє канонічне визначення поняття «алгоритм». Низка авторів використовують різні визначення. Це свідчить про неузгодженість окремих неформальних понять цього терміна. Сьогодні формальне визначення алгоритму є одним із завдань обчислювальної математики та інформатики, над яким працюють теоретики. Якщо алгоритмом називати набір інструкцій для виконання деякого завдання – це не буде фатальною помилкою для програмістів, тим паче, що таке визначення не змінює сутності розуміння алгоритму.

Теорія ЕОМ та практика програмування не можуть обійтися без теорії алгоритмів.

Теорія алгоритмів – це самостійна наука, яка має свій предмет. Предметом теорії алгоритмів є алгоритми.

Слово «алгоритм» походить від імені узбецького математика Хорезмі (арабською – аль-Хорезмі). Цей математик у IX ст. н. е. розробив правила чотирьох дій над числами в десятковій системі числення. Сукупність цих правил у Європі здобула назву «алгоритм». Розробка такого виду уточненої моделі була проведена у другій половині 30-х рр. минулого століття і привела до відкриття одного з найважливіших понять математики – поняття алгоритму (тобто алгоритм, метод, спосіб). Алгоритм розуміють як точне розпорядження про виконання в певному порядку послідовності операцій для вирішення всіх завдань з даного класу задач.

У 30-х рр. XX ст. поняття «алгоритм» стало об’єктом математичного вивчення. Розвиток ЕОМ і методів програмування прискорив розвиток теорії алгоритмів

як необхідного засобу автоматизації. До кінця ще не визначено точне поняття алгоритму, тому перейдемо до інтуїтивного поняття алгоритму з його властивостями, прикладами й недоліками.

Основні вимоги до алгоритмів формулюють так:

1. Кожен алгоритм має справу з даними – вхідними, проміжними, вихідними. Для того, щоб уточнити поняття даних, фіксується кінцевий алфавіт вихідних символів (цифри, букви тощо) і вказуються правила побудови алгоритмічних об'єктів. Типовим використовуваним засобом є індуктивна побудова. Наприклад, визначення ідентифікатора в мові С++: ідентифікатор – це або буква, або ідентифікатор, до якого приписана праворуч або буква, або цифра. Слова скінченної довжини в скінченних алфавітах – найбільш звичний тип алгоритмічних даних, а число символів у слові – природна міра об'єму даних. Інший випадок алгоритмічних об'єктів – формули. Прикладом можуть слугувати формули алгебри предикатів і алгебри висловлювань. У цьому випадку не кожне слово в алфавіті буде формулою.

2. Алгоритм для розміщення даних вимагає пам'яті. Пам'ять зазвичай вважається однорідною і дискретною, тобто вона складається з однакових комірок, причому кожна комірка може містити один символ даних, що дає змогу узгодити одиниці вимірювання обсягу даних і пам'яті.

3. Алгоритм складається з окремих елементарних кроків, причому множина різних кроків, з яких складений алгоритм, скінченна. Типовий приклад множини елементарних кроків – система команд процесора.

4. Послідовність кроків алгоритму детермінована, тобто після кожного кроку вказується, який крок потрібно виконувати далі, або вказується, коли треба роботу алгоритму вважати закінченою.

5. Алгоритм повинен бути результативним, тобто зупинятися після скінченного числа кроків (залежного від вхідних даних) з наданням результату. Цю властивість іноді називають збіжністю алгоритму.

6. Алгоритм передбачає наявність механізму реалізації, який за описом алгоритму породжує процес обчислення на основі вхідних даних. Передбачається, що опис алгоритму та механізм його реалізації скінченні. Можна помітити аналогію з обчислювальними машинами. Вимога 1 відповідає цифровій природі ПК, вимога 2 – пам'яті ПК, вимога 3 – програмі машини, вимога 4 – її логічній природі, вимоги 5, 6 – обчислювальному пристрою і його можливостям.

Функція $f(x_1, x_2, \dots, x_n)$ називається ефективно обчислюваною, якщо для кожного набору аргументів $a_1, a_2, \dots, a_n$ з її області визначення може бути обчислено значення функції $f(a_1, a_2, \dots, a_n)$. Якщо функція ефективно обчислювана, то кажуть, що існує алгоритм її обчислення.

Поняття алгоритму інтуїтивно зрозуміло і часто використовується в математиці. Додавання і множення чисел «у стовпчик», яке відоме ще зі школи, формула знаходження коренів квадратного рівняння, обчислення визначників за «правилом трикутника» чи «правилом Саррюса» є алгоритмами. Безліч прикладів можна продовжити.

Алгоритм можна описати словесно, де кожен пункт описуватиме відповідну дію. Такий спосіб використовують в окремих випадках, які вимагають низки додаткових пояснень. Наприклад, алгоритм пошуку коренів квадратного рівняння можна подати так:

1. Знайти дискримінант D за формулою $D = (-b)^2 - 4ac$.
2. Якщо $D < 0$, то квадратне рівняння не має коренів.
3. Якщо $D = 0$, то рівняння має один корінь.
4. Якщо $D > 0$, то рівняння має два корені.

Після такого опису можна починати створювати код мовою високого рівня і запускати його на комп'ютері.

Однак такий описовий спосіб алгоритмізації задач не набув широкого розповсюдження в основному через відсутність універсальності та труднощі пояснення для різних мов програмування.

Існує також символічний спосіб, який полягає в записі алгоритму за допомогою умовних символів. Такий спосіб подання алгоритму робить запис алгоритму дуже стислим і не наочним. Для розуміння логіки роботи алгоритму, записаного таким способом, необхідні додаткові знання і певна практика роботи з кодами програм та готовими алгоритмічними рішеннями. Зрештою, сам програмний код є способом запису алгоритму, але для його розуміння (навіть якщо він містить коментарі) треба знати мову програмування, на якій написаний цей код, і всі особливості його реалізації з використанням такої мови. Названі способи не є зручними для розуміння логіки роботи того чи іншого алгоритму.

Щоб полегшити розуміння роботи того чи іншого алгоритму, використовують спеціальні графічні побудови, або блоки, що відображають логіку роботи алгоритмів. Використання таких блоків регулюється відповідним міжнародним стандартом, хоча в багатьох випадках практикуючі програмісти цього стандарту можуть строго не дотримуватися. У більшості типових та розповсюджених алгоритмах для пояснення логіки роботи тієї чи іншої програми використовують саме їх. На рис. 1.1–1.3 показані окремі типові блоки для створення алгоритмів, що пов'язані з елементарними математичними та логічними операціями, а також із циклами, взяті із текстового редактора Microsoft Word. Під час кодування цикли в алгоритмах можна організовувати кількома способами з використанням умовних розгалужень (рис. 1.2). Для подання алгоритмів із наведених блоків та інших стандартних графічних побудов створюють блок-схеми.

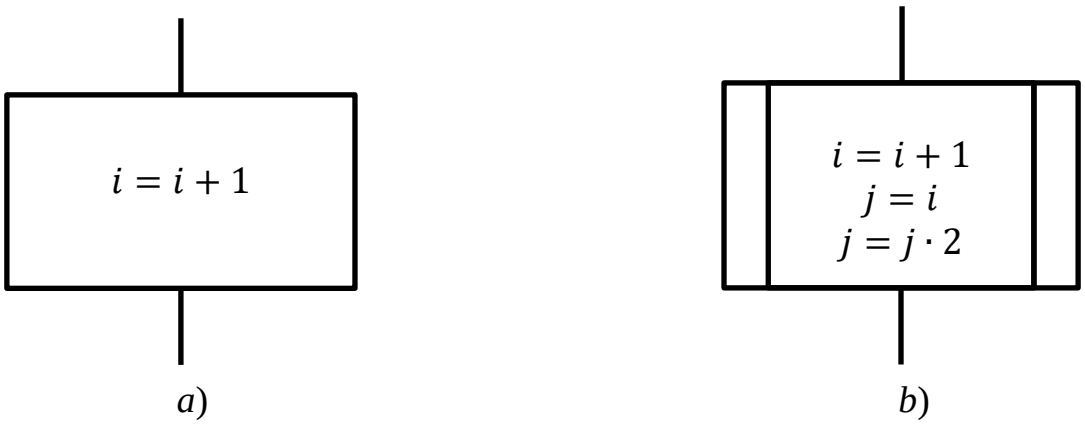


Рис. 1.1. Блоки для позначення виконуваної операції – a та групи операцій – b

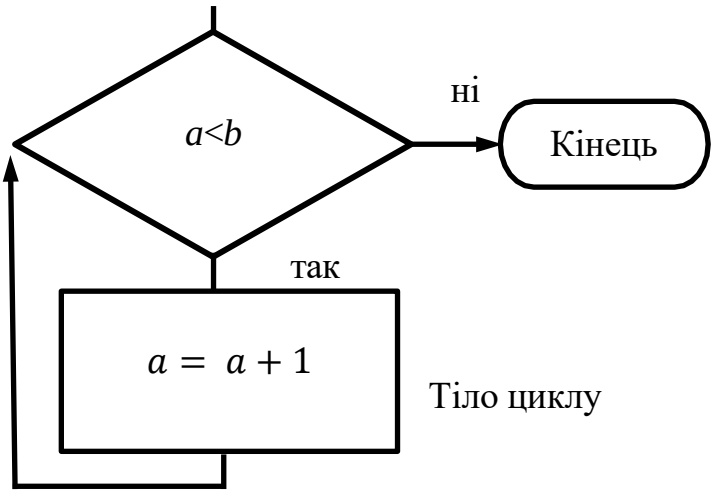


Рис. 1.2. Блок для позначення умовного розгалуження та організація циклу за допомогою нього

Приклад блок-схеми пошуку максимального елементу в одномірному масиві показаний на рис. 1.3.

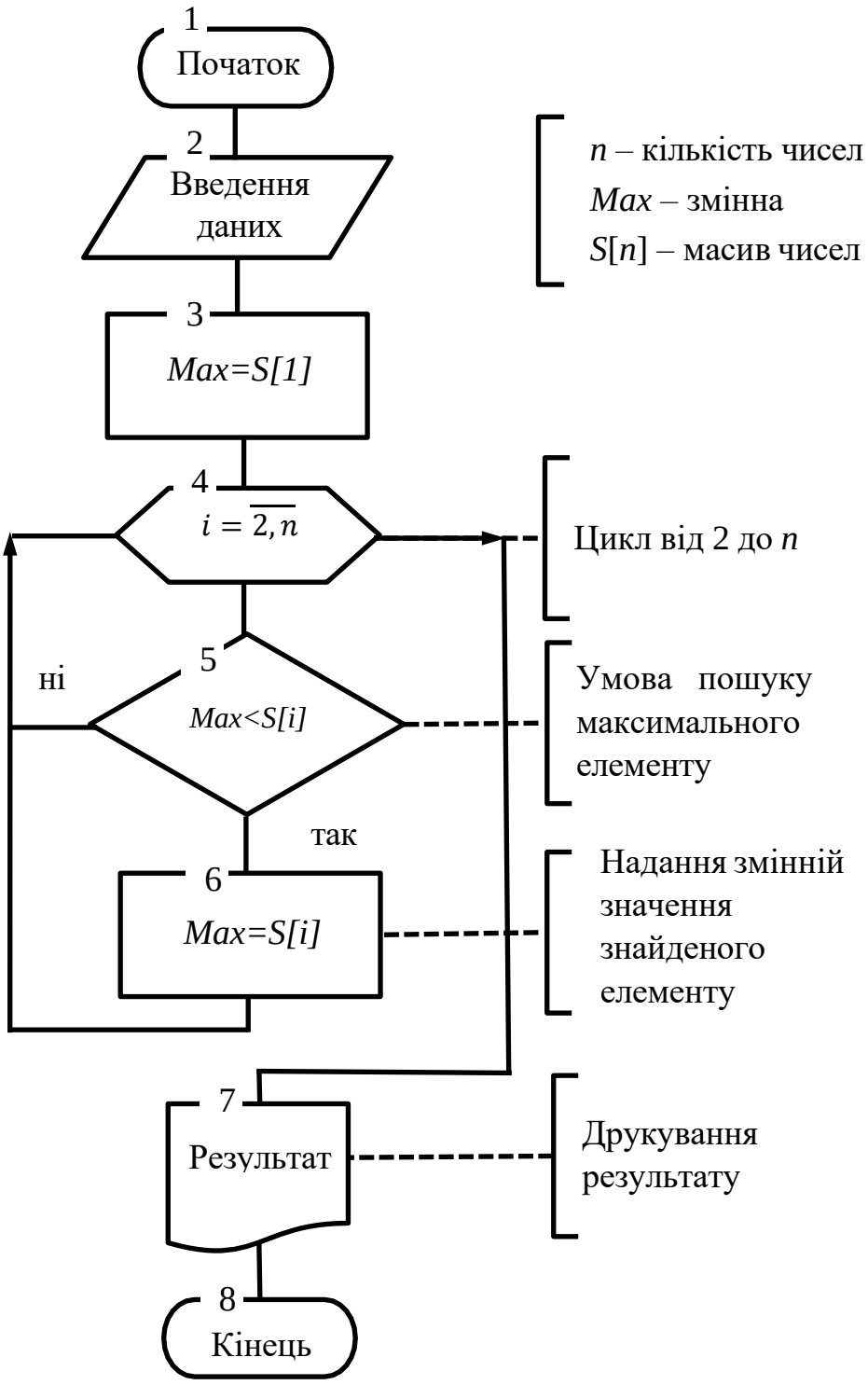


Рис. 1.3. Блок-схема алгоритму пошуку максимального елементу в одномірному масиві

Варто зазначити, що для графічного подання алгоритмів використовуються й інші графічні побудови. Зокрема, діаграми Насі–Шнейдермана (рис. 1.4) та інші. Однак використовуються вони нечасто.

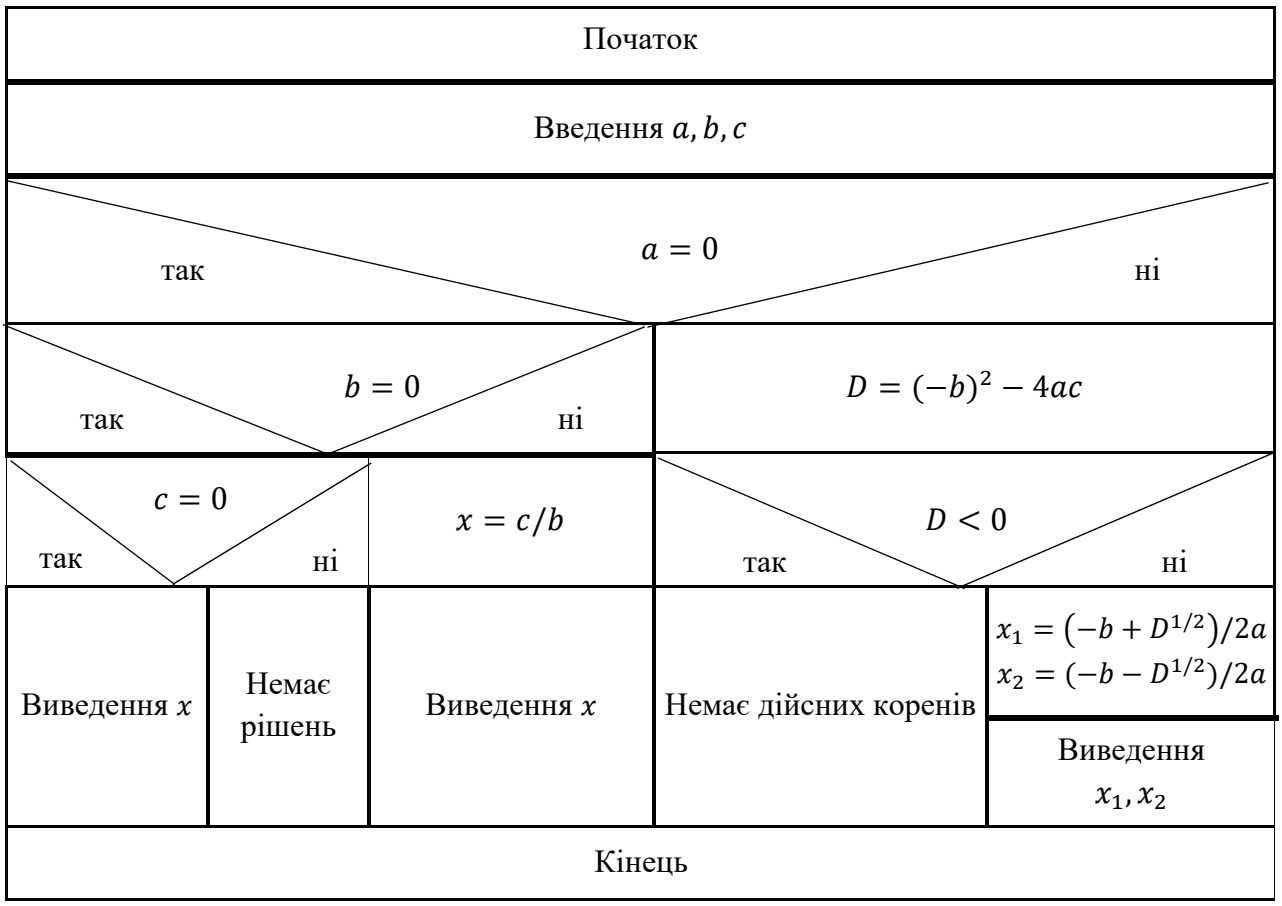


Рис. 1.4. Приклад графічного подання алгоритму обчислення коренів квадратного рівняння за допомогою діаграми Насі–Шнейдермана

Під час вивчення роботи алгоритмів також використовують універсальну мову кодування високого рівня. Ця мова не застосовується в комп'ютерах. Її призначення – пояснення особливостей алгоритмів незалежно від того, яка мова програмування використовується на практиці.

Такий псевдокод не містить спеціальних символів чи обов'язкових правил, як-от для написання змінних з малої чи великої літери або оголошення типів змінних тощо. Мета такого кодування – пояснення, яким способом алгоритм може бути запрограмований мовою високого рівня. Нижче наведений фрагмент такого псевдокоду для алгоритму пошуку максимального елементу в одномірному масиві, блок-схема якого показана на рис. 1.3.

```
MaxS (S,n; Max)//Назва програми, масив та змінні
Max ← S[1]  //Присвоєння змінній значення першого елементу For i ← 2 to n
//Цикл від i=2 до змінної n
if Max < S[i] then Max ← S[i] //Умова відбору end for      //Закінчення циклу
return Max //Повернення знайденого числа End   //Закінчення програми
```

Блок-схеми та псевдокоди суттєво спрощують розуміння роботи алгоритмів у їх практичному виконанні. Використання цих інструментів дає можливість

розібратися в тому, як працює той чи інший алгоритм, не вдаючись до подробиць, пов'язаних зі специфікою використання різних мов програмування.

У сучасній практиці під алгоритмом часто розуміють програму, а критерієм існування алгоритму вважають можливість його запрограмувати. Однак це не зовсім так. Спочатку дамо поняття алгоритму. ***Алгоритм*** – це сукупність правил, які визначають процес переробки допустимих початкових даних у вихідні результати.

Наведене вище поняття алгоритму не є чітким математичним визначенням. Це поняття характеризується набором властивостей, притаманних алгоритму.

Інтуїтивне поняття алгоритму:

Приклади деяких алгоритмів, які ми зустрічаємо в житті:

- у книгах рецептів приготування різних страв;
- у довідниках аптечних рецептів приготування ліків.

Приклади найпростіших алгоритмів із математики:

- правила оперування з десятковими числами (додавання, віднімання, множення, ділення);
- правила оперування з комплексними числами;
- алгоритми розв'язання квадратичних рівнянь, систем рівнянь та ін.

Кожна наукова й технічна дисципліна має довідники. Такі довідники переважно – це збірник алгоритмів, накопичених даною науковою або технічною дисципліною. Особливе значення мають алгоритми, накопичені в математиці, бо надбання математики є багатством усіх наук.

1.1.1. Властивості алгоритму

Поняття алгоритму має такі властивості або загальні риси:

1. Дискретність: алгоритм виконується покроково у дискретному часі. Дискретність інформації: кожен алгоритм має справу з даними – вхідними, проміжними і вихідними. Ці дані подаються у вигляді скінченних слів у деякому алфавіті. Дискретність роботи алгоритму: алгоритм виконується крок за кроком, і водночас на кожному кроці виконується тільки одна операція. Здійсненність операцій: в алгоритмі не повинно бути нездійсненних операцій. Наприклад, не можна в програмі присвоїти змінній значення «нескінченність», – така операція була б нездійсненною. Кожна операція обробляє певну ділянку в оброблюваному слові.

2. Детермінованість: система величин, яка отримана на якомусь кроці алгоритму, однозначно визначається системою величин, отриманих на попередньому кроці алгоритму. Кожен крок алгоритму строго визначений. Після кожного кроку точно вказується, який крок зробити далі, або вказується, що алгоритм повинен закінчити свою роботу на даному етапі. До кожної ділянки слова на кожному кроці може бути застосована тільки одна операція.

3. Скінченність алгоритму означає, що опис алгоритму має бути скінченним. До того ж робота самого алгоритму також повинна бути скінченна, тобто після скінченного числа кроків алгоритм повинен закінчувати свою роботу. Для програм, зокрема, це означає, що алгоритм не повинен «зациклюватися». Робота алгоритму завершується видачею результату, тому скінченність алгоритму тісно пов'язана з його результативністю – здатністю видавати результат.

4. Елементарність: закон одержання системи величин на кожному кроці алгоритму із системи величин на попередніх кроках повинен бути простим.

5. Направленість: ця властивість передбачає, що виконання алгоритмів повинне завершуватись одержанням певних результатів. Якщо ж спосіб одержання наступної величини не дає результату, то треба казати, що вважати результатом алгоритму.

6. Масовість: початкову систему величин можна вибирати з потенційно нескінченної множини величин. Масовість алгоритму означає, що алгоритм повинен вирішувати всі завдання з даного класу задач. Якщо знайдеться хоча б одна задача, для якої алгоритм не застосуємо, то цю послідовність операцій не можна вважати алгоритмом. Так, наприклад, метод резолюцій не є алгоритмом доказу загальнозначущості формул логіки предикатів. Існують завдання, для яких метод резолюцій не може дати однозначної відповіді, наприклад, чи повинен відомий нам цирульник голити самого себе.

Таке поняття алгоритму не строге, не точне, бо воно включає слова «правило», «спосіб», «величина», точний зміст яких не встановлено, хоча інтуїтивно зрозуміло зміст цих слів.

Алгоритмічна система – система, яка дає чітке поняття, визначення алгоритму, задає загальний спосіб побудови алгоритмів.

1.1.2. Алфавіти і слова

Означення 1.1. Будемо називати довільну скінченну множину ***алфавітом A,*** а елементи цієї множини *буквами* алфавіту A. Тоді довільні скінченні послідовності букв називаються *словами* в даному алфавіті.

Будь-який алфавіт містить порожній символ. Будемо позначати його символом Λ. Порожній символ є і порожнім словом.

Множину слів алфавіту A будемо позначати A^*.

Означення 1.2. Кількість букв у слові називається його ***довжиною*** і позначається вертикальними дужками. Наприклад, довжина слова a дорівнює $|aaaa| = 4$, довжина порожнього слова $|\Lambda| = 0$.

Основною операцією над словами є ***конкатенація.***

Означення 1.3. ***Конкатенацію*** двох слів P і Q назвемо слово, отримане дописуванням слова Q після P: PQ.

Наприклад, конкатенацією слів aab і ba є слово $aabba$. Операція конкатенації некомутативна, але асоціативна.

Означення 1.4. Якщо x – деяке слово алфавіту A і якщо x представимо у вигляді конкатенації PQ, то P і Q називають ***підсловами*** слова x.

Наприклад, слово abb містить підслова a, b, ab, bb. Якщо $x \in A^*$ і $X = P_1QP_2QQP_3$, то йдеться про перше, друге тощо ***входження*** слова Q в слово X. Слова, складені з послідовності однакових букв, будемо іноді записувати $aaabb = a^3b^2$, де ступінь позначає кількість входжень літери в слово.

1.1.3. Кодування та нумерація

Означення 1.5. Нехай дано два алфавіти A і B. ***Кодуванням*** слів алфавіту A словами в алфавіті B є функція $\varphi(p)$, яка здійснює відображення множини слів в алфавіті A в множину слів в алфавіті B: $\varphi(p)$: $A^* \to B^*$, де $p \in A^*, \varphi(p) \in B^*$.

Якщо кожному слову в алфавіті A відповідає слово в алфавіті B, то відображення однозначне.

Приклад 1.1. Нехай $A = \{a, b\}, B = \{a, b, c\}$. Відображення $\varphi(p)$: $p \to cpc$ визначає кодування слів в алфавіті A словами в алфавіті B, наприклад, $ab \to cabc$.

Означення 1.6. Кодування називається ***блоковим,*** якщо кожній букві алфавіту A відповідає слово в алфавіті B.

Приклад 1.2. Відображення $\varphi(a)$: $a \to ac$, $\varphi(b)$: $b \to bc$ є блоковим кодуванням. Наприклад, $\varphi(ab) = acbc$.

Будь-яку множину слів у деякому алфавіті можна впорядкувати в лексикографічному порядку: якщо $R_1\alpha R_2, R_1\beta R_2$, і $\alpha < \beta$, то $R_1\alpha R_2 < R_1\beta R_2$.

Приклад лексикографічного впорядкування для слів алфавіту $A = \{a, b\}$: $\Lambda, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb$ тощо.

1.1.4. Проблема еквівалентності слів. Словникові примітивно рекурсивні функції

Вихідні функції для алфавіту A:

Нуль-функція $0(x) = \Lambda$.

Додавання символу ζ: $P\zeta(x) = x\,\zeta$, де $\zeta \in A$.

Проєктуюча функція $U_i^j(x) = x_i$, де j – кількість букв у слові x, x_i – i-та буква слова x.

Схема примітивної рекурсії для алфавіту $A = \{a, b\}$.

Нехай $g(x_1, \dots, x_n)$, $h_1(x_1, \dots, x_{n+2})$, $h_2(x_1, \dots, x_{n+2})$, – вихідні або примітивно рекурсивні функції. Тоді $f(x_1, \dots, x_{n+1})$, – нова примітивно рекурсивна функція, якщо:

$$f(x_1, \dots, \Lambda) = g(x_1, \dots, x_n);$$
$$f(x_1, \dots, a) = h_1\big(x_1, \dots, x_n, a, f(x_1, \dots, x_{n+1})\big);$$
$$f(x_1, \dots, b) = h_1\big(x_1, \dots, x_n, b, f(x_1, \dots, x_{n+1})\big).$$

Позначимо операцію конкатенації $con(x_1, x_2), x_1, x_2 \in A^*, A = \{a, b\}$ і покажемо, що вона є примітивно рекурсивною.

$$con(x_1, \Lambda) = x_1 = U_1^2(x) = x_1;$$
$$con(x_1, x_2, a) = x_1x_2a = P_a(con(x_1x_2));$$
$$con(x_1, x_2, b) = x_1x_2b = P_b\big(con(x_1x_2)\big).$$

У загальному випадку:

$$con(x_1, x_2, \xi) = x_1, x_2\xi = P_\xi\big(con(x_1, x_2)\big).$$

Асоціативні обчислення

Означення 1.7. Підстановка слова P замість Q в слові W означає заміну Q на P в слові W. Підстановка позначається: $P \to Q$.

Приклад 1.3. Нехай дано алфавіт $A = \{a, b\}$ і підстановка $ab \to ba$. Тоді результат виконання підстановки над словом $ababbbab$ буде таким:

1. $\underline{ab}\ abbbab$
2. $ba\ \underline{ab}\ bbab$
3. $b\ \underline{ab}\ abbab$
4. $bba\ \underline{ab}\ bab$
5. $b\ b\ \underline{ab}\ abab$
6. $bbba\ \underline{ab}\ ab$
7. $bbb\ \underline{ab}\ aab$
8. $bbbaa\ \underline{ab}$
9. $bbba\ \underline{ab}\ a$
10. $bbbb\ \underline{ab}\ aa$
11. $bbbbbaaa$

Означення 1.8. Сукупність усіх слів алфавіту A разом зі скінченною системою допустимих підстановок називається ***асоціативним обчисленням.***

Проблема еквівалентності слів

Означення 1.9. Якщо слово R може бути перетворене в слово S за допомогою одноразового застосування допустимої підстановки, то очевидно, що слово S може бути перетворене в слово R за допомогою застосування зворотної підстановки. Такі слова називаються ***суміжними*** словами.

Означення 1.10. Послідовність суміжних слів $R_1, R_2, \ldots, R_n$ називають ***дедуктивним ланцюжком*** від R_1 до R_n.

Означення 1.11. Якщо існує дедуктивний ланцюжок від R до S, то існує і дедуктивний ланцюжок від S до R. Такі два слова називаються ***еквівалентними словами.***

Проблема еквівалентності слів полягає в знаходженні дедуктивного ланцюжка від слова R до слова S. Це означає, що необхідно знайти таку множину допустимих підстановок, щоб, застосовуючи їх до слова R, у кінці дедуктивного ланцюжка отримати слово S.

Надалі ми покажемо, що проблема еквівалентності слів для будь-якого довільно взятого асоціативного обчислення алгоритмічно нерозв'язна.

ПИТАННЯ ДО ТЕМИ 1

1. Яке основне інтуїтивне визначення алгоритму?
2. Сформулюйте основні властивості алгоритму.
3. Скільки способів задання циклу в алгоритмі можна використати, використовуючи блок-схеми?
4. Чому використання псевдокодів є зручним для роботи з алгоритмами?
5. У чому полягають труднощі розуміння роботи коду без графічного представлення алгоритму?
6. Що ми називаємо алфавітом?
7. Дайте визначення конкатенації.
8. Що ми розуміємо під поняттям «кодування»?
9. Що ми розуміємо під поняттям «нумерація»?
10. Які асоціативні обчислення ви можете описати?
11. У чому полягає проблема еквівалентності слів?

ТЕМА 2
РЕКУРСИВНІ ФУНКЦІЇ

2.1. Рекурсивні функції. Теоретичні поняття та визначення.

2.2. Примітивно рекурсивні функції.

2.3. Частково рекурсивні функції.

2.1. Рекурсивні функції. Теоретичні поняття та визначення

Історично першою алгоритмічною системою була система рекурсивних функцій. Ця теорія дає змогу з більш простих функцій будувати складніші функції або алгоритми. Завдяки теорії рекурсивних функцій було отримане уточнення поняття алгоритму.

Ця теорія базувалася на розгляді цілочисельних функцій.

Аргументи цих функцій і самі функції будуть приймати невід'ємні цілі значення $N_0 = \{0\} \cup N$.

Найпростіші функції (елементарні арифметичні функції):

1. Нуль-функції тотожно рівні нулю:

$$O^n(x_1, x_2, \dots, x_n) = 0.$$

2. Тотожні (селекторні) функції, які повторюють значення своїх аргументів:

$$I_i^n(x_1, x_2, \dots, x_i, \dots, x_n) = x_i \ \ (1 \le i \le n; n = 1,2, \dots).$$

3. Функція безпосереднього слідування:

$$S(x) \ = \ x \ + 1 \ .$$

Тепер сформулюємо конструктивні засоби, допустимі операції, які дають змогу з елементарних найпростіших функцій будувати більш складні арифметичні функції.

Операція (оператор) суперпозиції, або підстановки, полягає в підстановці одних арифметичних функцій замість аргументів інших арифметичних функцій. Нехай:

$$\begin{cases} f_1^m(x_1, x_2 \ , \dots, x_m); \\ f_2^m(x_1, x_2 \ , \dots, x_m) \ ; \\ f_n^m(x_1, x_2 \ , \dots, x_m), \end{cases}$$

n функцій, кожна з яких залежить від m змінних, і є функція:

$f^n(x_1, x_2 \ , \dots, x_n)$ від n аргументів, або змінних.

Операція суперпозиції функцій $f_1, f_{12}, \dots, f_{1n}$ з функцією f (або операція підстановки функцій $f_1, f_{12}, \dots, f_{1n}$) у функцію f дає функцію:

$$g^m(x_1, x_2, \dots, x_m) = f^n(f_1^m(x_1, x_2, \dots, x_m) \dots f_n^m(x_1, x_2, \dots, x_m)).$$

Приклад 2.1. Нехай є функція $O(x) = 0$ і функція $S(x) = x + 1$.

Підставимо першу функцію (нуль-функцію) у функцію безпосереднього слідування.

Отримаємо функцію:

$$g(x) = S(O(x)) = S(0) = 0 + 1 = 1;$$

$g(x) \equiv 1 = const_1(x)$ – функція-константа, тотожно рівна 1.

Приклад 2.2. Нехай $S(x) = x + 1$. Підставимо її замість x, тобто утворимо суперпозицію $S(x)$ з самою собою:

$$g(x) = S(S(x)) = S(x + 1) = x + 2; g(x) = S(S(x)) = x + 2.$$

Приклад 2.3. Нехай є функція $O^n(x_1, x_2, \dots, x_n) = 0$ і $S(x) = x + 1$.

$$S\left(S\left(S\left(O^n(x_1, x_2, \dots, x_n)\right)\right)\right) = S\left(S\left(S(O)\right)\right) = S\left(S(1)\right) = S(2) = 3:$$

$g(x_1, x_2, \dots, x_n) = 3$ – функція-константа.

Приклад 2.4. Нехай $S(z) = z + 1, I_3^3(x, y, z) = Z;$

$$h(x, y, z) = I_3^3(x, y, S(z)) = I_3^3(x, y, z + 1) = z + 1;$$

x, y – фіктивні аргументи.

2.2. Примітивно рекурсивні функції

Під рекурсією будемо розуміти спосіб завдання функції, за якого значення деякої функції будуть виражатися через значення цієї функції для менших значень аргументів.

Почнемо з прикладу. Розглянемо дві функції $I_1^1(x) = x$ – тотожну (селекторну) функцію і $h(x, y, z) = z + 1$ – різновид функції безпосереднього слідування.

Звернемо увагу, що перша функція – унарна, а друга – тернарна.

Створимо третю функцію з першої і другої, щоб вона залежала від двох аргументів $(f(x, y))$.

Побудуємо її так:

$$f(x, 0) = I(x) = x;$$

$$f(x, 1) = h\big(x, 0, f(x, 0)\big) = h(x, 0, x) = x + 1;$$

$$f(x, 2) = h\big(x, 1, f(x, 1)\big) = h(x, 1, x + 1) = x + 2;$$

$$f(x, y - 1) = h\big(x, y - 2, f(x, y - 2)\big) = h(x, y - 2, x + y - 2) = x + y - 1;$$

$$f(x, y) = h(x, y - 1, f(x, y - 1)) = h(x, y - 1, x + y - 1) = x + y.$$

Операція примітивної рекурсії дає змогу визначити значення функції $f(x, y)$ через значення цієї функції для менших значень аргументу.

Так за допомогою операції примітивної рекурсії і двох елементарних функцій ми одержали функцію, що визначає суму двох чисел:

$$f(x, y) = x + y.$$

Тепер можна дати визначення операції примітивної рекурсії в загальному вигляді.

Зауважимо, що операція примітивної рекурсії дає змогу будувати функцію від $n + 1$ аргументу ($n + 1$-арну функцію) з двох заданих функцій, одна з яких n-арна, а друга $n + 2$-арна.

Нехай дано довільні часткові функції: n-арна функція g і $n + 2$-арна функція h. Говорять, що $n + 1$-арна функція f одержана операцією примітивної рекурсії з функцій g, h, якщо для всіх $x_1, x_2, \dots, x_n \in N$ маємо:

$$f(x_1, x_2, \dots, x_n, 0) = g(x_1, x_2, \dots, x_n);$$
$$f(x_1, x_2, \dots, x_n, y + 1) = h(x_1, x_2, \dots, x_n, y, f(x_1, x_2, \dots, x_n, y)).$$

Означення 2.1. ***Примітивно рекурсивними функціями*** (ПРФ) називають такі функції, які можна побудувати з елементарних арифметичних функцій за допомогою операції суперпозиції та примітивної рекурсії, застосованих скінченним числом разів у довільному порядку.

Тобто функції:

$$\begin{cases} f(x, y) = x + y; \\ f(x, y) = xy \\ f(x) = \frac{x}{1}; \end{cases} \in \text{ПРФ};$$

$$\frac{x}{1} = \begin{cases} x - 1, & x \geq 1; \\ 0, & x = 0. \end{cases}$$

Операція мінімізації (операція найменшого кореня)

Ця операція дає змогу будувати нову арифметичну n-арну функцію $f(x_1, x_2, \dots, x_n)$ з заданої $(n + 1)$-арної функції $g(x_1, x_2, \dots, x_n, y)$.

Цю операцію мінімізації, або найменшого кореня, розглянемо спочатку на прикладі.

Приклад 2.5. Нехай задана функція $g(x) = x^2$ – ПРФ. Допустимі значення аргументу x: $0, 1, 2, \dots$. Введемо допоміжний аргумент y і розглянемо рівняння $g(y) = x$, тобто $y^2 = x$. Зафіксуємо допустимі значення основного аргументу x у таблиці 2.1.

Отримане значення кореня рівняння будемо приймати за значення функції, яку ми будуємо.

Таблиця 2.1

x	Рівняння	Корінь рівняння $\mu_y(y^2 = x)$	$f(x)$ побудована функція
0	$y^2 = 0$	0	$0 \to 0$
1	$y^2 = 1$	1	$1 \to 1$
2	$y^2 = 2$	немає	$2 \to$ не визначене
3	$y^2 = 3$	немає	$3 \to$ не визначене
4	$y^2 = 4$	2	$4 \to 2$
5	$y^2 = 5$	немає	$5 \to$ не визначене
6	$y^2 = 6$	…	…
7	$y^2 = 7$	немає	$7 \to$ не визначене
8	$y^2 = 8$	немає	$8 \to$ не визначене
9	$y^2 = 9$	3	$9 \to 3$
…	…	…	…

Це значення функції відповідає зафіксованому значенню x.

Аналітичний вираз побудованої функції $f(x)$:

$$f(x) = \begin{cases} \sqrt{x}, \text{для } x, \text{які є повним квадратом числа } x = k^2, \text{де } k \in N_0; \\ \text{не визначена для інших допустимих значень } x. \end{cases}$$

Формальне визначення операції мінімізації

Нехай $g(x_1, x_2, \dots, x_{n-1}, x_n)n$-арна часткова функція.

Зафіксуємо $a_1, a_2, \dots, a_{n-1}, a_n$ – набір допустимих значень змінних, де $x_1 = a_1, x_2 = a_2, \dots, x_{n-1} = a_{n-1}, x_n = a_n, a_i \in N0(1 \le i \le n)$.

Розглянемо рівняння відносно змінної y:

$$(a_1, a_2, \dots, a_{n-1}, y) = a_n. \qquad (1)$$

Якщо існує розв'язок (корінь) цього рівняння, то позначимо через μ_y – найменший натуральний корінь цього рівняння.

Якщо

$$g(a_1, a_2, \dots, a_{n-1}, 0), g(a_1, a_2, \dots, a_{n-1}, 1)g\big(a_1, a_2, \dots, a_{n-1}, \mu_y - 1\big) \qquad (2)$$

визначені, тоді функція f є результатом виконання операції мінімізації і визначається так:

$$f(a_1, a_2, \dots, a_{n-1}, a_n) = \mu_y.$$

Щоб підкреслити, що функція f отримана з функції g шляхом виконання операції мінімізації, використовується таке позначення:

$$f(x_1, x_2, \dots, x_n) = \mu_y(g(x_1, x_2, \dots, x_n, y) = x_n).$$

Якщо ж рівняння (1) не має кореня, або хоча б одне з значень, яке позначене (2), не визначене, то значення функції $f(a_1, a_2, \dots, a_{n-1}, a_n)$ теж не визначене. Водночас вважається, що сама операція мінімізації закінчується без результату.

2.3. Частково рекурсивні функції

Означення 2.2. Функції, які будуються з елементарних арифметичних функцій за допомогою скінченого числа операцій підстановки, примітивної рекурсії та мінімізації, називаються ***частково рекурсивними функціями*** (ЧРФ).

Тобто ПРФ – це функції, які можна побудувати з елементарних арифметичних функцій за допомогою операції суперпозиції та примітивної рекурсії, застосованих скінченне число разів у довільному порядку.

Чи рівні ці множини функцій між собою: ПРФ ≠ ЧРФ? Яка з цих множин є підмножиною іншої ПРФ ⊂ ЧРФ?

Кожна з примітивно рекурсивних функцій буде частково рекурсивною. А чи справедливе обернене твердження? Кожна з ЧРФ ∈ ПРФ – так чи ні? Існують ЧРФ, які не є ПРФ.

Дійсно, ПРФ-функції всюди визначені на відміну від ЧРФ.

Приклад 2.6. Нехай задана функція:

$$(x) = S(x) = x + 1 \to f(x) = \mu_y(S(y) = x) = \mu_y(y + 1) = x,$$

$S(x)$ всюди визначена.

Частково визначена:

$$\mu_y(S(y) = x) = \begin{cases} x - 1, & \text{при } x > 0; \\ \text{не визначена}, & \text{при } x = 0. \end{cases}$$

Отже, $\mu_y(S(y) = x)$ не визначена при $x = 0$, і вона буде ПРФ-функцією.

Приклад 2.7. Нехай задана функція $g(x) = 5^x, x \in N_0, g(y) = x, 5^y = x$.

Розв'язання представимо у вигляді таблиці 2.2.

Таблиця 2.2

x	**Рівняння** $5^y = x$	**Корінь рівняння** $\mu_y(5^y = x)$	**Функція** $f(x)$
0	$5^y = 0$	немає	0 → не визначена
1	$5^y = 1$	0	0 → 1
2	$5^y = 2$	немає	не визначена
3	…	…	…
4	…	…	…
5	$5^y = 5$	1	5 → 1
6	…	немає	не визначена
…	…	…	…
10	…	…	…
5^2	$5^y = 25$	$25 \to 2$	25→ 2
…	…….	……	……

Аналітичний вираз побудованої функції $f(x)$ має вигляд:

$$f(x) = \begin{cases} Log_5 x, & \text{при } x = 5^n, n \in N_0; \\ \text{не визначена}, & \text{при } x \neq 5^n. \end{cases}$$

$$g(x) = x + 1, f(x) = \mu_y(y + 1 = x) = \begin{cases} x - 1, & \text{якщо } x > 0; \\ \text{не визначена} & \text{при } x = 0. \end{cases}$$

$$g(x) = 7x, f(x) = \mu_y(7y = x) = \begin{cases} \frac{x}{7}, & \text{при } x = 7k, k \in N_0; \\ & \text{не визначена.} \end{cases}$$

Якщо функція $g(x)$, з якої буде будуватися функція операцією мінімізації, одномісна, унарна, то функція $f(x) = \mu_y(g(y) = x)$ буде оберненою до $g(x)$.

Тобто якщо необхідно побудувати ЧРФ з функції $g(x) = x^3$, то:

$$g^{-1}(x) = f(x) = \mu_y(g(y) = x) = \mu_y(y^3 = x) = \begin{cases} \sqrt[3]{x}, & x = k^3, k \in N_0, \\ \text{не визначена}, & x \neq k^3, k \in N_0. \end{cases}$$

Але все одно, хоч і не так просто, як для одномісних функцій, 2-, 3-, *n*-місні частково рекурсивні функції можна будувати за чіткою схемою мінімізації.

Підводячи підсумок класу ЧРФ, можна зробити такі висновки. Поняття ЧРФ є одним із головних понять теорії алгоритмів. ЧРФ являють собою найбільш загальний клас конструктивно створюваних арифметичних функцій. Усі ЧРФ можуть бути обчислені шляхом деякого алгоритму.

Так, алгоритм, який супроводить операцію примітивної рекурсії, ми записали цілком чітко. Алгоритм, який супроводить операцію мінімізації можна коротко сформулювати так.

1. Ввести додатковий аргумент і розглянути рівняння:

$$g(x_1, x_2, \dots, x_{n-1}, y) = x_n.$$

2. Зафіксувати допустимі значення аргументів $x_1 = a_1, x_2 = a_2, \dots, x_{n-1} = = a_{n-1}, x_n = a_n$.

3. Надавати додатковому аргументу послідовні значення, починаючи з 0, доти, поки не виконається рівність:

$$g(x_1, x_2, \dots, x_{n-1}, \mu) = x_n.$$

Отримане значення додаткового аргументу μ прийняти за значення функції *f*, яке відповідає значенням зафіксованих аргументів. Тобто:

$$f(a_1, a_2, \dots, a_{n-1}, a_n) = \mu.$$

У іншому випадку будемо вважати, що операція мінімізації закінчується без результату, а функція не визначена для даних значень аргументів (тобто якщо процедура обчислення значень функції працює нескінченно, не видаючи жодного результату, то функція f – не визначена).

На прикладах алгоритмів, що супроводять операції примітивної рекурсії й мінімізації, ми бачимо, що частково рекурсивні функції обчислюються за деякими алгоритмами, тобто є обчислюваними функціями.

Поняття обчислюваної функції у теорії алгоритмів існує. ***Обчислюваними функціями*** називаються числові функції, значення яких можна обчислити шляхом деякого алгоритму.

Поняття алгоритму у цьому визначенні інтуїтивне, значить і поняття обчислюваної функції є теж інтуїтивним. На відміну від цих інтуїтивних понять, поняття частково рекурсивної функції є точним.

Як же пов'язані поняття алгоритму, обчислюваної функції і частково рекурсивної функції?

Ми знаємо, що теорія рекурсивних функцій з'явилась у зв'язку з необхідністю формалізації поняття алгоритму. Дійсно, ЧРФ відповідає нашому інтуїтивному уявленню про алгоритм. І навпаки? З практики алгоритмізації: які б класи чітко окреслених алгоритмів до сих пір не будувались, виявлялось, що числові функції обчислювані шляхом цих алгоритмів, були частково рекурсивними. Тому у 1936 р. американський математик А. Черч висунув тезу.

Теза Черча. Клас алгоритмічно (або машино) обчислюваних часткових числових функцій співпадає з класом усіх частково рекурсивних функцій.

Теза Черча є гіпотезою. Вона зв'язує неточне інтуїтивне поняття алгоритму, обчислюваної функції з точним математичним поняттям частково рекурсивної функції. Розмитість тези не дає змоги довести її, але на користь її справедливості вказують багато чітко доведених тверджень. Тобто є достатньо вагомих підстав прийняти тезу Черча як основу теорії алгоритмів. З даними тези Черча ми ознайомимося пізніше.

ПИТАННЯ ДО ТЕМИ 2

1. Які операції підстановки рекурсивної функції ви знаєте?
2. Які операції суперпозиції рекурсивної функції ви знаєте?
3. Які операції примітивної рекурсії рекурсивної функції ви знаєте?
4. Що таке примітивно рекурсивні функції?
5. Який оператор називається оператором мінімізації?
6. Що таке частково рекурсивні функції?
7. Що ми розуміємо під поняттям «взяття обмеженої суми», «обмеженого добутку»?

ТЕМА 3
МАШИНА ТЮРІНГА. ЇЇ ЗНАЧЕННЯ ДЛЯ ТЕОРІЇ АЛГОРИТМІВ

3.1. Теоретичні основи.
3.2. Машина Тюрінга.
3.3. Робота машини Тюрінга.
3.4. Способи задання машини Тюрінга.
3.5. Машина Тюрінга, що розпізнає.
3.6. Властивості машини Тюрінга як алгоритму.
3.7. Композиція машин Тюрінга.
3.8. Різновиди машин Тюрінга.

3.1. Теоретичні основи

Незважаючи на те, що поняття алгоритму виникло в математиці ще у глибокому середньовіччі, основні наукові положення, що стосуються концепції застосування алгоритмів як частини машинних обчислень, були розроблені лише у XX ст. У 1930-х рр. американський математик Еміль Пост і британський математик Алан Тюрінг заклали теоретичні основи алгоритмічної реалізації. Обчислювальні моделі, які були створені цими науковцями незалежно один від одного, називають «машинами» через їх схожість з обчислювальними автоматами. Через схожість між собою їх також називають «машиною Поста–Тюрінга».

Алан Матісон Тюрінг (англ. Alan Mathison Turing, 23.06.1912–07.06.1954) – англійський математик, логік, криптограф, винахідник машини Тюрінга. Син британського чиновника з Індії, вчився у Франції, Англії та США. Тоді багато математиків намагалися створити алгоритм для визначення істинності висловлень. Геделю вдалося довести, що будь-яка корисна математична система аксіом неповна в сенсі того, що в ній існує вислів, істинність якого не можна ні спростувати, ні підтвердити. Це спонукало Тюрінга довести, що немає загального методу визначення істинності, а отже, математика завжди буде містити недоведені висловлювання.

У своїй роботі Тюрінг запропонував проєкт простого пристрою, що має всі основні властивості сучасної інформаційної системи: програмне управління, пам’ять і покроковий спосіб дій. Ця уявна машина, що отримала назву «машини Тюрінга», використовується в теорії автоматів або комп’ютерів.

Коли Тюрінг зі США повернувся в Англію, почалася Друга світова війна. Одним з найважливіших озброєнь цієї війни була ЕОМ «Колос» за проєктом «Ультра», що почала в 1943 р. зламувати надскладні шифри німців.

Після війни в 1945 р. Алан Тюрінг очолив проєкт створення комп’ютера «ТУЗ» (ACE, Automatic Computing Engine), а в 1948 р. вчений став працювати з «МАДАМ» (MADAM, Manchester Automatic Digital Machine) – комп’ютером з найбільшою

пам’яттю у світі на той час. Роботи Алана зі спорудження перших ЕОМ і розвитку методів програмування мали неоціненну важливість, давши основу більшості досліджень в області штучного інтелекту. Він вважав, що комп’ютери врешті-решт зможуть мислити, як людина, і запропонував просту перевірку, відому як тест Тюрінга, що оцінює здатність машини мислити: поговоріть з ЕОМ, і нехай вона переконає вас, що вона – людина.

Одна зі щорічних нагород Асоціації обчислювальної техніки називається Премія Тюрінга. Премія заснована Асоціацією обчислювальної техніки в честь Алана Тюрінга, як видатного вченого, який отримав перші глибокі результати щодо обчислюваності задовго до появи перших електронних обчислювальних машин. Премія щорічно вручається одному або кільком фахівцям у галузі інформатики та обчислювальної техніки, чий внесок у цю область справив сильний тривалий вплив на комп’ютерне співтовариство.

Премія може бути присуджена одній людині не більше одного разу. У сфері інформаційних технологій премія Тюрінга має статус, аналогічний Нобелівській премії в академічних науках. Вперше премія Тюрінга була присуджена в 1966 р. Алану Перлісу за розвиток технології створення компіляторів.

У 2000-ні рр. преміальний фонд спонсорувався корпораціями Intel і Google, щорічний розмір премії становив $250 тисяч. Із 2014 р. щорічний призовий фонд збільшено до $1 млн, а компанія Google стала єдиним спонсором премії.

За традицією, лауреат премії Тюрінга під час її вручення виступає з доповіддю, що має назву «Тюрінгівська лекція». У цій «лекції» зазвичай йдеться про ті питання комп’ютерної науки, теорії та практики використання обчислювальної техніки, які лауреат вважає доволі важливими, щоб поділитися своєю думкою про них із якомога більшою кількістю фахівців.

Серед лауреатів премії були:

- Дональд Кнут (аналіз алгоритмів);
- Едсгер Дейкстра (теорія мов програмування);
- Хоар, Чарльз Ентоні Річард (теоретик мов програмування, винахідник алгоритму Quick Sort);
- Ніклаус Вірт (творець мови Паскаль) та багато інших видатних науковців.

Машина Поста і машина Тюрінга схожі між собою і містять низку спільних елементів:

– стрічка (теоретично нескінченна) з набором комірок, у яких можуть розташовуватися символи;

– каретка, що може рухатися над стрічкою і здатна виконувати дії над символами;

– дії над символами стрічки можна задавати (в сучасному розумінні – програмувати).

Однак існують і певні відмінності. Більш детально роботу цих машин ми розглянемо нижче.

3.2. Машина Тюрінга

Машина Тюрінга (МТ) була першою алгоритмічної схемою, запропонованою в якості математичного визначення алгоритму в 1937 р.

МТ є гіпотетичною машиною: Тюрінг не ставив завдання сконструювати цей пристрій. Концепція обчислювальної машини належить Джону фон Нейману, але її основою послужила машина Тюрінга. МТ є знакопереробним пристроєм. Завдання МТ – обробляти вхідне слово $W \in A^*$ в якесь вихідне слово $W^* \in A^*$, де A – зовнішній алфавіт.

Приклад 3.1. Нехай є нескінченна з двох боків паперова стрічка, на якій можна писати букви в клітинках. Є головка, яка: може знаходитись в одному із двох різних станів, читає і записує букви в клітинки стрічки; може рухатися вліво і вправо (п) вздовж стрічки та стояти (с).

Складемо таблицю, яка описує МТ для обчислення функції $S(x) = x + 1$ в алфавіті $A = \{I\}$. # – спеціальний пустий символ.

Символ {!} означає зупинку МТ.

Таблиця 3.1

	$\boldsymbol{q_0}$	$\boldsymbol{q_1}$
#	Iq_1c	
I	Iq_0п	!

Таблиця 3.2

#	I	I	#	
	q_0			
#	I	I	#	
			q_0	
#	I	I	#	
			q_0	
#	I	I	I	#
			q_1	

$x = $ II — початкове число
$Iq_0 \rightarrow Iq_0$п
II$q_0 \rightarrow$ Iq_0п
□$q_0 \rightarrow$ Iq_1с
I$q_1 \rightarrow$ I
$S = $ III — результат

Можна зробити висновок, що яке б число не було записане на стрічці МТ, головка буде рухатися вправо до пустої клітинки, запише в пусту клітинку I і зупиниться. На стрічці буде записано число $x + 1$ замість x. Тобто буде обчислена функція $S(x) = x + 1$.

Алгоритмічну систему, в якій правила, що визначають дію алгоритму, побудовані за командно-адресним принципом для деякої машини з необмеженою пам'яттю, називають МТ.

1. У цій системі виконується запис інформації на нескінченній в обидва боки стрічці, розбитій на клітинки. У клітинку можна записати тільки одну букву скінченого алфавіту $A = \{a1, \dots an\}$.

2. Цю букву можна оглядати (читати, записувати) спеціальним чутливим елементом – так званою головкою МТ.

Головка МТ може знаходитись в одному зі скінченої кількості різних станів:

$$Q = \{q_0, q_1, \dots q_m\}.$$

Головка може записувати в клітинці, яку оглядає, будь-яку букву з алфавіту A, і залишатись на місці або переміщуватись праворуч чи ліворуч уздовж інформаційної стрічки на одну клітинку.

Задати МТ означає задати таку п'ятірку об'єктів:

$$A, Q, q_0, \#, P,$$

де A – алфавіт машини (зовнішній алфавіт);

Q – скінчена непуста множина станів машини, ($A \cap Q = \emptyset$);

q_0 – початковий стан машини Тюрінга;

$\#$ – спеціальний пустий символ ($\# \notin A \cup Q$);

P – програма роботи машини Тюрінга.

Машина Тюрінга виконує команди.

Команда має вигляд $aq \to bq'd (aqbq'd)$;

$$a, b, \in A;$$
$$q', q' \in Q;$$
$$d = \{S, L, R\}.$$

Програма МТ – це скінчена множина команд, причому не існує таких двох команд, які б мали однакові перші два символи. Як виконується команда $aq \to bq'd$?

Головка МТ:

- знаходиться в стані q;
- читає записану в клітинці букву a;
- записує в цю клітинку нову букву b (яка може збігатися з буквою a);
- переходить у стан q' (який може збігатися зі станом q) і переміщується вздовж стрічки на величину d.

Якщо $d = S$, то головка залишається на місці;

……. $d = R$, то відбувається зсув головка вправо на 1 клітинку;

……. $d = L$, то відбувається зсув головки вліво на 1 клітинку.

МТ зупиняється тоді і тільки тоді, коли жодна з команд програми не може застосовуватись або вказується символ {!}, що означає зупинку МТ.

Результатом роботи машини Тюрінга після її зупинки є слово, записане на стрічці. Це слово називається заключним.

У цьому випадку МТ вважається застосовною до початкового слова.

Якщо МТ обробляє початкове слово нескінченно довго, то машина Тюрінга вважається не застосовною до початкового слова.

Функціонування МТ можна описати за допомогою протоколу роботи над заданим початковим словом.

Нехай:

1) $\#a_1, \dots a_k a_k + 1, \dots a_k\#$ – слово, що виникає на стрічці в процесі роботи МТ.

2) МТ знаходиться в стані q.

3) Головка стоїть проти k-го символу слова.

Слово $\#a_1 \dots a_k q a_{k+1} \dots a_m\#$ ($\#a_1 \dots a_k a_{k+1} \dots a_m\#$) називається ***конфігурацією.***

Послідовність конфігурацій під час роботи МТ називається ***протоколом машини Тюрінга.***

Розглянемо приклад.

Приклад 3.2. Створити МТ, яка будь-яке натуральне число n, написане у двійковій системі перетворює на число $n + 1$ теж записане у двійковій системі, тобто реалізує $S(x) = x + 1$ – обчислення функції безпосереднього слідування.

Програму можна записувати у вигляді послідовності команд, тобто у вигляді спеціальної таблиці (табл. 3.3).

Таблиця 3.3

		Стан машини Тюрінга	
Символ	q_0	q_1	q_2
#	$\#q_1L$	$1q_2L$	
0	$0q_0R$	$1q_2L$	
1	$1q_0R$	$0q_2L$	!

Протокол МТ

1) # 1 0 #← початкова конфігурація
q_0
2) # 1 0 #
q_0
3) # 1 0 #
q_0
4) # 1 0 #
q_1
5) # 1 0 # ← заключна конфігурація
q_2

Початкове слово $w = 10$
Результат $w_1 = 11.$

Із кожною МТ пов'язана часткова функція (словесна функція):

$f_{\text{мт}} : S(A) \to S(A)$, де $S(A)$ – множина слів в алфавіті A.

1) Якщо МТ зупиняється під час роботи над початковим словом w, то значення функції визначене, і $f_{\text{мт}}(w) = w'$, де w' – заключне слово на стрічці МТ.

2) Якщо МТ не зупиняється, значення функції не визначене на слові w.

Нехай задана часткова функція f на множині $S(A)$. Ця функція називається ***обчислюваною за Тюрінгом,*** якщо існує МТ така, що:

1) Зовнішній алфавіт МТ співпадає з A.

2) $f(w) \equiv f\text{мт}(w), \forall(w) \in S(A)$.

Приклад 3.3. Наведемо приклад МТ, яка незастосовна до будь-якого непустого слова в алфавіті $A = \{I\}$.

Протокол МТ

I
1) # I
q_0
2) # I #
q_0
3) # I I #
q_0
4) # I I I #
q_0 Протокол МТ нескінченний

Таблиця 3.4

	Стан
Символ	q_0
#	$I\ q_0 L$
I	$I\ q_0 L$

Дія цієї МТ полягає в тому, що, почавши роботу з будь-якого непустого слова w, до цього слова приписується зліва символ I. Машина не зупиняється. Результат роботи не визначений, оскільки машина не застосовна до будь-якого непустого слова $w \in S(A)$.

Приклад 3.4. МТ, яка реалізує алгоритм додавання двох натуральних чисел в алфавіті $\mathrm{A} = \{I\}, B = \{I, +\}$.

Ідея алгоритму:

#	I	+	I	#

1) Кожну I першого доданку стерти й записати її в кінці другого доданку.

2) Стерти знак +.

Таблиця 3.5

	Стан машини Тюрінга			
Символ	q_0	q_1	q_2	q_3
#	$\#\ q_0\ R$	$I\ \ q_2 S$	$\#\ q_0\ S$	!
I	$\#\ q_1\ R$	$I\ q_1 R$	$I\ q_2 L$	
+	$\#\ q_3\ S$	$+\ \ q_1\ R$	$+\ \ q_2\ L$	

Протокол МТ

$P = \mathrm{I} + \mathrm{I}$

1) # I + I #
q_0

2) # # + I #
q_1

3) # + I #
q_1

4) # + I #
q_1

5) # + I I #
q_2

6) # + I I #
q_2

7) # + I I #
q_2

8) # + I I #
q_2

9) # + I I #
q_0

10) # + I I #
q_0

11) # I I #
q_3

12) # I I #
q_3

$f_{МТ}(I + I) = \mathrm{II}$ $\qquad$ $P_1 = \mathrm{I\,I}$

МТ складається з трьох частин.

I. Нескінченна в обидві сторони стрічка, розбита на клітинки, в кожній з яких може перебувати один (і тільки один) символ зовнішнього алфавіту $A = \{\alpha, \beta, \gamma, \dots\}$. Інформація записується на стрічці у вигляді слова в алфавіті A.

II. Голівка, що зчитує, в один дискретний момент часу оглядає одну клітинку і може перебувати в одному з внутрішніх станів $q_i \in Q$, де Q – алфавіт внутрішніх

станів. Водночас вона розпізнає записаний у клітинці символ зовнішнього алфавіту, записує замість нього інший символ (можливо, той самий), переходить в інший стан (можливо – залишається в поточному), після чого зсувається вправо, вліво або залишається на місці (П, Л, Н) (рис. 3.1).

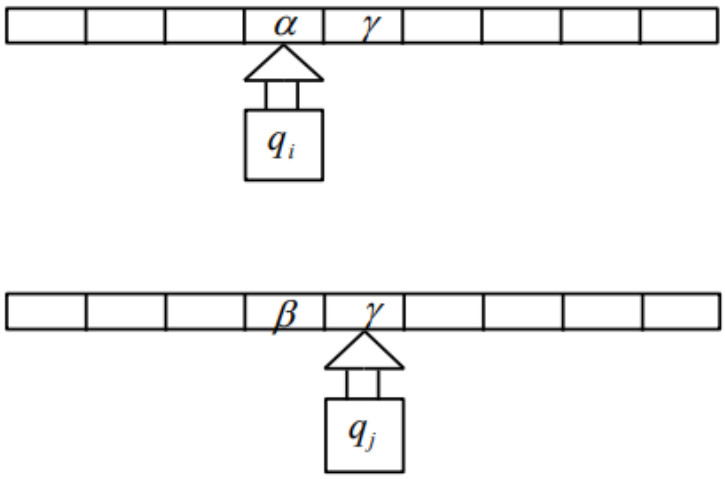


Рис. 3.1. Схематичне зображення машини Тюрінга

Отже, множина внутрішніх станів Q являє собою пам'ять МТ: коли машина в різних станах бачить один і той же самий символ, вона може виконати різні дії.

III. Наступна частина МТ – це програма, що складається з команд виду:

$$\alpha q_i \to \beta q_j \xi, \xi \in \{\text{П}, \text{Л}, \text{Н}\}, q_i q_j \in Q, \alpha, \beta \in A.$$

Внаслідок виконання цієї команди символ α на стрічці буде замінений символом β, головка змінить своє положення залежно від ξ, і внутрішній стан зміниться з q_i на q_j.

Послідовність команд задає алгоритм обробки слова. На вхід МТ подається вихідне слово W в алфавіті A і задається початковий стан q_0: початкова конфігурація. Робота МТ відбувається по *тактах,* або по *кроках*. Під час роботи МТ можливе розширення зовнішнього алфавіту A допоміжними символами, які після обробки слова замінюються символами алфавіту A або стираються. У число символів алфавіту A обов'язково входить порожній символ Λ. Вважається, що спочатку всі комірки стрічки заповнені порожніми символами.

У процесі роботи можливі такі ситуації:

1. МТ обробляє вихідне слово P в Q і зупиняється; і можна сказати, що ця машина може бути застосована до слова P.

2. МТ, починаючи роботу з слова P, ніколи не зупиниться; тоді ця МТ не може бути застосована до слова P.

Приклад 3.5. Нехай заданий алфавіт $A = \{|,*\}$, вихідне слово в якому може мати вигляд: $P = ||| * | * ||$. У початковому стані q_0 машина оглядає крайній лівий символ. МТ повинна перетворити це слово в порожнє слово, тобто повинна реалізовувати алгоритм обчислення нуль-функції: $0(x) = \Lambda$. Для цього вона повинна, рухаючись зліва направо, замінити кожен символ на стрічці на порожній символ і зупинитися, як тільки побачить крайній правий символ.

Розв'язання. Програму для МТ зазвичай записують у вигляді таблиці.

Таблиця 3.6

A/Q	q_0
|	$\Lambda q_0 П$
*	$\Lambda q_0 П$
Λ	!

Це дуже простий алгоритм. Розглянемо програму для алгоритму додавання символа | до слова в тому ж алфавіті. Нехай вихідне слово має вигляд: $P = |||$. Це слово можна розглядати як число 3, а додавання символа | – як обчислення функції $f(x) = x + 1$ в унарній системі числення. Програма для МТ насправді дуже проста: головка машини пропускає всі символи |, рухаючись зліва направо, і, дійшовши до крайнього правого порожнього символа Λ, дописує туди ще один символ |, після чого зупиняється в своєму заключному стані.

Таблиця 3.7

A/Q	q_0
|	$q_0 П$
Λ	$! H$

Очевидно, що так само можна скласти програму для додавання будь-якого символу алфавіту A в кінець слова, тобто такий алгоритм реалізовує вихідну рекурсивну функцію $P_\zeta(x) = x\zeta$, де $\zeta \in A$.

Третя вихідна функція: функція проєкції $U_i^j(x) = x_i$ є не що інше, як елементарна дія МТ – розпізнавання символу.

Розглянемо тепер обчислення більш складної функції: $F(x, y) = x + y$, де x, y – слова, задані в унарній системі числення. Нехай вихідна конфігурація на стрічці задана так, що в початковому стані головка оглядає крайній лівий символ | (див. рис. 3.2). Словом, на стрічці є два числа, розділені символом *.

Складемо МТ, що працює за таким алгоритмом. У початковому стані q_0 машина бачить крайню ліву паличку, стирає її і, перейшовши в стан q_1, рухається вправо, поки не побачить крайній праворуч порожній символ. Тоді на місці порожнього символу машина записує паличку, переходить у новий стан q_2 і рухається вліво в тому ж стані q_2, пропускаючи всі символи, поки не дійде до крайнього

зліва порожнього символу. Тоді машина залишає його на місці, зсувається вправо і переходить у стан q_0. Після цього процес повторюється доти, поки в стані q_0 машина не побачить символ *. Це означає, що вже всі палички перенесені зліва вправо, машина може стерти символ * і зупинитися, тобто перейти до заключного стану.

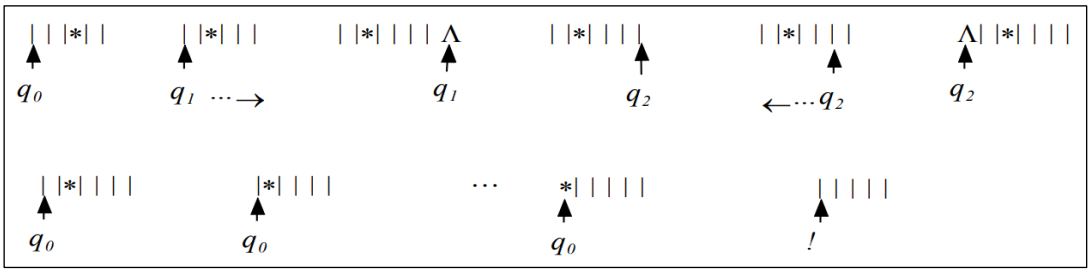


Рис. 3.2. Схема роботи алгоритму Тюрінга

Таблиця 3.8

A/Q	q_0	q_1	q_2
|	$Лq_1П$	$\| q_1П$	$\| q_2Л$
*	$Л\,!\,Н$	$*\ q_1П$	$*\ q_2Л$
$Л$	$Л\,!\,Н$	$\| q_2П$	$Лq_0П$

Ми бачимо, що процес обчислення суми рекурсивний: він заснований на попередньому алгоритмі, коли до одного з доданків (у цьому випадку до правого слова) додається стільки одиниць, скільки їх міститься в іншому доданку. Для порівняння відтворимо процес обчислення суми двох доданків за допомогою рекурсивної функції:

$$f(x,0) = x;$$

$$f(x,y+1) = f(x,y) + 1;$$

$$f(2,3) = f(2,2) + 1 = (f(2,1) + 1) + 1 = \big((f(2,0) + 1) + 1\big) + 1.$$

На цьому завершується прямий перебіг рекурсії – складена рекурсивна схема обчислення функції. Тепер зворотний перебіг рекурсії обчислює її значення:

$$\big((2+1)+1\big)+1 = (3+1)+1 = 4+1 = 5.$$

Таблиця 3.9

A/Q	q_0	q_1	
|	$Лq_1П$	$\| q_1П$	
*	$Л\,Н!$	$\|Н!$	

Можна написати програму для МТ яка просто стирає першу паличку і записує її на місце *. Але така програма, хоч на перший погляд і ефективна, все ж не буде рекурсивною і не буде містити уніфіковані дані.

3.3. Робота машини Тюрінга

У початковий момент часу на стрічці записана скінченна послідовність символів (вхідне слово), машина знаходиться в початковому стані q_0 та, якщо не визначено інше, курсор зчитує перший символ вхідного слова. Далі, відповідно до списку команд, курсор рухається по стрічці і слово перетворюється. МТ закінчує роботу у двох випадках:

- здійснено перехід у заключний стан (нормальне завершення роботи);
- відсутня команда з відповідною лівою частиною (ненормальне або аварійне закінчення роботи).

У МТ використовується не унарна система позначок у комірках стрічки, а алфавіт знаків, які задає користувач. Також у цій машині використовується так званий «алфавіт станів». Зміну станів, яку задає користувач у таблиці переходів, умовно можна назвати програмою.

Ця таблиця визначає поведінку каретки – перехід із клітинки в клітинку, додавання (видалення) символу алфавіту, зупинку. МТ є моделлю, так би мовити, «ближчою» до комп'ютера, і не обмежується однотипним набором інструкцій, а дає змогу створювати послідовність станів, яку умовно можна назвати програмою.

Зауваження 3.1. Наприкінці роботи машини правилом гарного тону вважається повернути курсор на початок слова.

Приклад 3.6. До слова, яке складається зі скінченної кількості паличок, дописати у кінці зірочку.

Розв'язання

1) $q_0|\rightarrow q_0|R;$

2) $q_0\Lambda \rightarrow q_1 * L;$

3) $q_1|\rightarrow q_1|L;$

4) $q_1\Lambda \rightarrow !\,\Lambda\, R.$

Пояснення. У стані q_0 ми проходимо все слово, і в кінці (як тільки зустрінемо Λ) ставимо зірочку та переходимо в стан q_1. У стані q_1 ми проходимо слово справа наліво і зупиняємося на першому символі (крайня ліва паличка) і переходимо в заключний стан !.

Розберемо цей приклад покроково (табл. 3.10):

Таблиця 3.10

Покроковий протокол роботи МТ для прикладу 3.6

№ кроку	команда	стан	стрічка
0	–	q_0	$\Lambda \underline{1}11\Lambda$
1	1)	q_0	$\Lambda 1\underline{1}1\Lambda$
2	1)	q_0	$\Lambda 11\underline{1}\Lambda$
3	1)	q_0	$\Lambda 111\ \underline{\Lambda}$
4	2)	q_1	$\Lambda 11\underline{1} * \Lambda$
5	3)	q_1	$\Lambda 1\underline{1}1 * \Lambda$
6	3)	q_1	$\Lambda \underline{1}11 * \Lambda$
7	3)	q_1	$\underline{\Lambda}\ 111 * \Lambda$
8	4)	!	$\Lambda \underline{1}11 * \Lambda$

Підкреслений символ на стрічці означає положення курсора.

Приклад 3.7. Обчислити $x + 1$ у двійковій системі числення.

1) $q_0\xi \to q_0\xi R$;
2) $q_0\Lambda \to q_1\Lambda L$;
3) $q_1 0 \to q_2 1L$;
4) $q_1 1 \to q_1 0L$;
5) $q_1\Lambda \to !\,1S$;
6) $q_2\xi \to q_2\xi L$;
7) $q_2\Lambda \to !\,\Lambda R$.

$\xi \in \{0; 1\}$ використовується для скорочення запису. Тобто команда $q_0\xi \to q_0\xi R$ означає дві команди: $q_0 0 \to q_0 0R$ та $q_0 1 \to q_0 1R$.

Пояснення. У стані q_0 ми проходимо все слово і в кінці (як тільки зустрінемо Λ) ставимо зірочку та переходимо в стан q_1 і стаємо на останню цифру. Якщо ця цифра – «1», то замість неї пишемо «0» і стан не змінюємо, бо з усіма наступними одиницями виконуємо цю саму дію. Але якщо зустрічаємо «0», то замість нього пишемо «1» і змінюємо стан на q_2. У стані q_2 ми проходимо число справа наліво і зупиняємося на першій цифрі отриманого числа. Окремий випадок, коли ми зустрінемо Λ у стані q_1. Цей випадок означає, що ми маємо справу з числом вигляду 11 … 1. Тут всі «1» замінилися на «0», і тоді залишається замість Λ поставити «1» і перейти в заключний стан.

3.4. Способи задання машини Тюрінга

МТ можна представити:

1) списком команд (представлено раніше);
2) табличним способом;
3) графічним способом.

Табличний спосіб задання МТ має вигляд (табл. 3.11):

Таблиця 3.11

Табличний спосіб задання машини Тюрінга

	q_0	q_1	…	q_n	…	q_N	!
α_1			…		…		
α_2			…		…		
…	…	…	…	…	…	…	
α_i			…	$q_m\alpha_j R$	…		
…	…		…		…		
α_M			…		…		
Λ			…		…		

Тут $Q = \{q_0,\ q_1,\ q_2, \dots ,\ q_N,\ \},\ A = \{\alpha_1,\ \alpha_2, \dots ,\ \alpha_M\}$.

Команда $q_n\alpha_i \to q_m\alpha_j R$ відображена в табл. 3.10. У відповідну клітинку записується права частина команди.

Графічний спосіб МТ зображується у вигляді орієнтованого міченого мультиграфу, де множина вершин співпадає з множиною станів, а кожна команда зображується у вигляді ребра. Наприклад, команда $q_n\alpha_i \to q_m\alpha_j R$ зображується так, як представлено на (рис. 3.3):

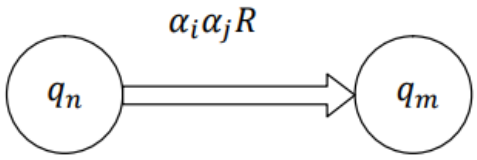


Рис. 3.3. Зображення команди машини Тюрінга

Продемонструємо табличний та графічний спосіб для розв'язання прикладу 3.7 (табл. 3.12 та рис. 3.4):

Таблиця 3.12

Табличний спосіб задання машини Тюрінга для прикладу 3.7

	q_0	q_1	q_2	!
0	$q_0 0R$	$q_2 1\text{L}$	$q_2 0\text{L}$	
1	$q_0 1R$	$q_1 0\text{L}$	$q_2 1\text{L}$	
Λ	$q_1 \Lambda\text{L}$	$1!\,S$	$\Lambda!\,R$	

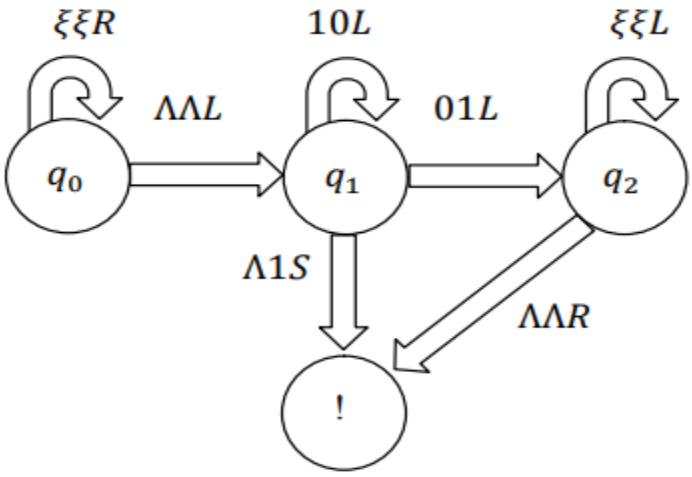


Рис. 3.4. Графічний спосіб задання машини Тюрінга для прикладу 3.7

Приклад 3.8. Скопіювати слово в алфавіті $\{a, b\}$. Вхідне слово: w. Вихідне слово: $w = w$.

Побудуємо граф МТ для розв'язання цього прикладу (рис. 3.5). Позначимо $\xi \in \{a, b\}, \tilde{\xi} \in \{\tilde{a}, \tilde{b}\}$.

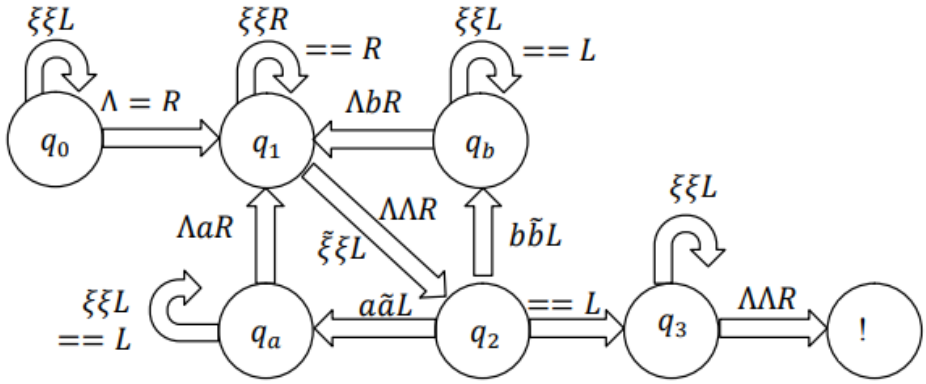


Рис. 3.5. Граф для прикладу 3.8

Пояснення. Слово будемо копіювати справа наліво. У стані q_0 ставимо знак «=» перед словом і більше цей стан використовувати не будемо. Переходимо в стан q_1. У цьому стані ми проходимо слова зліва направо до крайнього нескопійованого символу. Перший раз це буде останній символ слова, а в наступні рази – символ, який стоїть лівіше від відміченого (ξ). Знімаємо мітку і переходимо в стан q_2. У стані q_2 запам'ятовуємо символ, який будемо копіювати, переходячи в стан q_a чи q_b, а також відмічаємо цей символ (ставимо хвильку). У випадку, коли всі символи вже скопійовані (у стані q_2 потрапили на символ «=», переходимо в стан q_3 і рухаємося вліво на початок результуючого слова. У стані $q_a(q_b)$ проходимо слово вліво і на вільне місце записуємо літеру $a(b)$, повертаємося в

стан q_1 Продовжуємо процедуру далі, поки не скопіюємо все слово w. Алгоритм буде також працювати у випадку порожнього слова. Вихідним словом буде «=».

Приклад 3.9. Побудувати МТ, яка:

1) слово в алфавіті $\{a, b\}$ записує в дужках. Вхідне слово w. Вихідне слово (w);
2) обчислює $x - 1$ у двійковій системі $(x > 0)$;
3) переводить число з унарної у двійкову систему;
4) переводить число з двійкової в унарну систему;
5) перевіряє, чи є число в алфавіті $\{a, b\}$ паліндромом (читається однаково зліва направо чи навпаки, справа наліво).

3.5. Машина Тюрінга, що розпізнає

Машина Тюрінга, що розпізнає – це машина, яка обчислює предикат $P(W)$ таким, що якщо $P(W) = T$, то машина зупиняється в стані *так*!, а якщо $P(W) = F$, то в стані *ні*!.

Наприклад, машина, що розпізнає парність числа, представленого в унарній системі числення, наведена нижче в таблиці 3.13. Рухаючись зліва направо, машина зупиняється на порожньому символі в стані *так*!, якщо число парне, і в стані *ні*!, якщо непарне.

Таблиця 3.13

Таблиця схеми запису розпізнавання числа

$A \backslash Q$	q_0	q_1
|	| q_1 П	| q_0 П
Λ	Λ *так*!	Λ *ні*!

3.6. Властивості машини Тюрінга як алгоритму

На прикладі МТ добре простежуються властивості алгоритмів.

Зрозумілість. Кожен крок вивірений. Тобто на кожному кроці в комірку пишеться символ з алфавіту, автомат робить один рух, і МТ переходить в один з визначених станів.

Дискретність. Виконання алгоритмів, як свідчить робота МТ, здійснюється покроково. Перехід до наступного кроку можливий лише після виконання попереднього. Попередній крок визначає те, яким буде наступний.

Детермінованість. Для кожної клітинки описаний лише один варіант дії. Тобто послідовність операцій переходу з комірки в комірку і дії над ними здійснюється не випадковим способом, а відповідно до чітко визначеного правила. Послідовність кроків під час виконання задачі визначена однозначно. Варто зазначити, що існує недетермінована МТ, в якій допускаються варіанти переходу з однієї комірки до інших.

Результативність. Результат виконання задачі на МТ буде досягнутий за фіксоване число кроків.

Масовість. МТ можна застосовувати для виконання конкретних задач зі всіма можливими словами алфавіту. Тобто кожна МТ визначена над усіма допустимими словами з алфавіту.

3.7. Композиція машин Тюрінга

Для розв'язання складних задач за допомогою МТ доцільно будувати машини для окремих підзадач, а потім їх зв'язувати в одну машину.

Означення 3.1. Нехай T_1 та T_2 – дві МТ, у яких множини внутрішніх станів не перетинаються ($Q_1 \cap Q_2 = \emptyset$). ***Композицією машин T_1 та T_2*** називається машина $T = T_2 \circ T_1$, робота якої полягає в тому, що спочатку працює T_1, а потім T_2 (вихідне слово T_1 є вхідним для T_2). $T(w) = T_2\big(T_1(w)\big) = (T_2 \circ T_1)(w)$.

Для реалізації композиції машин Тюрінга T_1 та T_2 достатньо ототожнити кінцевий стан T_1 та початковий T_2 (рис. 3.6 а)).

Композицію машин Тюрінга можна узагальнити для скінченної кількості машин: $T = T_n \circ T_{n-1} \circ \ldots \circ T_2 \circ T_1$.

У більш складних випадках (рис. 3.6 б) і в)) необхідно визначити для машини T_1 два (можливо, і більше) заключні стани, у кожен з яких переходимо у випадку виконання певної умови.

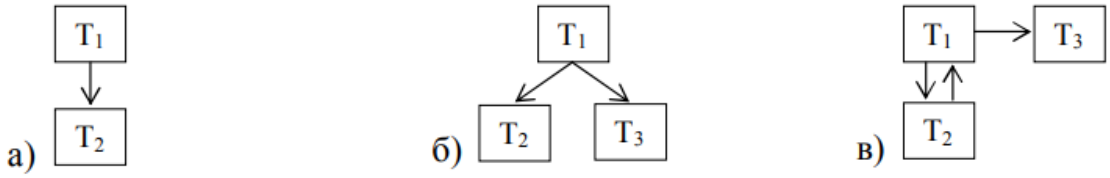


Рис. 3.6. Композиція машин Тюрінга

Можна виділити деякий набір елементарних алгоритмів, з яких можна отримувати більш складні алгоритми за правилами композиції МТ. Ми вже показали, що елементарні функції:

- нуль-функція $0(x) = \Lambda$;
- додавання символу ζ: $P_\zeta(x) = x\zeta$, де $\zeta \in A$;
- проєктуюча функція $U_i^j(x) = x_i$;
- тотожне перетворення $T(x) = x$;
- алгоритм копіювання слова $Copy\ (x) = x * x$;
- замінюючий алгоритм – $Rep_{x_j}^{x_i}(x) = x_j$ де $x_i, x_j \in A \cup B$, де B – допоміжний алфавіт.

3.8. Різновиди машин Тюрінга

Модель МТ допускає різноманітні узагальнення. Розглядаються МТ з довільною кількістю стрічок та багатовимірними стрічками з різними обмеженнями. Також розглядається напівнескінченна стрічка (необмежена в один бік). Недетермінована МТ може перебувати в кількох станах одночасно. Всі ці машини еквівалентні звичайній МТ.

Крім МТ, існують інші алгоритми, еквівалентні їй. Із цих елементарних алгоритмів можна утворювати більш складні за допомогою композиції МТ.

- ***Послідовна композиція***

Нехай $M1$ і $M2$ – дві МТ. Тоді, ототожнивши заключний стан машини $M1$ з початковим станом машини $M2$ і, за необхідності, перенумерувавши внутрішні стани машини $M2$, отримаємо нову МТ, яка, починаючи роботу зі слова W, спочатку буде виконувати над ним перетворення, що виконуються машиною $M1$, а потім буде працювати як машина $M2$. Композицію МТ можна позначити: $M1°M2 = M2(M1(W))$. Отже, послідовна композиція МТ здійснює суперпозицію функцій.

- ***Паралельна композиція***

Якщо на стрічці записано слово W, яке представимо як конкатенацію двох слів P$||Q$, можна скласти таку МТ, яка буде працювати як машина $M1$ над підсловом P і як машина $M2$ над підсловом Q, а потім здійснить конкатенацію результатів: $M1(P)||M2(Q)$.

- ***Машина Тюрінга, що розпізнає***

Якщо існують МТ $M1$ і $M2$ і МТ P, що розпізнає, то можна скласти таку МТ M, що, починаючи роботу зі слова W, МТ працює спочатку як така, що розпізнає – $P(W)$. Якщо вона закінчує обробку вихідного слова в стані «так», то далі працює машина $M1(W)$, а якщо в стані «ні», то машина $M2(W)$ (рис. 3.7).

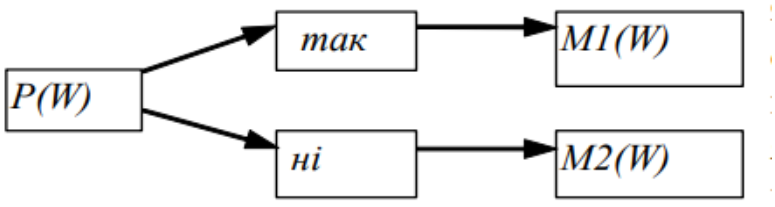


Рис. 3.7. Схема машини Тюрінга

- ***Циклічна Машина Тюрінга***

Можна побудувати таку МТ M, яка, починаючи роботу зі слова W_0, спочатку працює як розпізнавальна машина P, що обчислює предикат $P(W_0)$. Якщо вона завершує свою роботу в стані *так!*, то далі вона працює як МТ $M1$ над словом W_0 і завершує свою роботу з вихідним словом W_1. Далі управління передається розпізнавальній машині P, що обчислює предикат $P(W_1)$, і так далі, доти, поки розпізнавальна МТ P не зупиниться в стані *ні*! на слові W_k. Тоді МТ M зупиняється у своєму заключному стані (рис. 3.8).

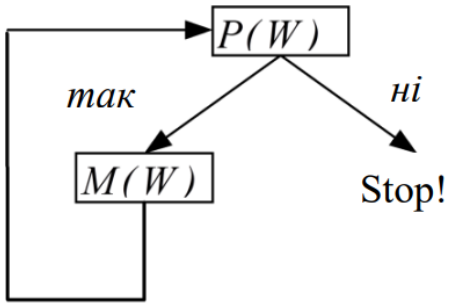


Рис. 3.8. Схема циклічності машини Тюрінга

Було математично доведено, що для реалізації будь-яких алгоритмів достатньо цих чотирьох структур. Композиція МТ містить у собі основну ідею структурного програмування, і саме Тюрінг є автором доведення повноти цих чотирьох базових алгоритмів для реалізації алгоритму будь-якої складності.

Приклад 3.10. Побудувати МТ

1) Розпізнає, чи слово є паліндромом, в алфавіті $\{a, b\}$: $aabbaa$ – так; $aabaa$ – так; $abbaa$ – ні (табл. 3.4).

Таблиця 3.14

$A\backslash Q$	q_0	q_1	q_2	q_3	q_4	q_5
a	$\Lambda q_1 П$	$a q_1 П$	$a q_2 П$	$\Lambda q_5 Л$	ні!	$a q_3 Л$
b	$\Lambda q_2 П$	$b q_1 П$	$b q_2 П$	ні!	$\Lambda q_5 Л$	$b q_5 Л$
Λ	*так*!	$\Lambda q_3 Л$	$\Lambda q_4 Л$	*так*!	*так*!	$\Lambda q_0 П$

2) Копіювання слова в алфавіті $\{0, 1\}$: $001101 \rightarrow 001101 * 001101$ (табл. 3.15).

Таблиця 3.15

$A\backslash Q$	q_0	q_1	q_2	q_3	q_4
0	$0q_0П$	$0q_1Л$	$aq_3П$	$0q_3П$	$0q_4П$
1	$1q_0П$	$1q_1Л$	$bq_4П$	$1q_3П$	$1q_4П$
a		$0q_2П$			
b		$1q_2П$			
*		$*\ q_1Л$	!	$*\ q_3П$	$*\ q_4П$
Λ	$*\ q_1Л$	$Λq_2П$		$0q_1Л$	$1q_1Л$

ПИТАННЯ ДО ТЕМИ 3

1. Що ми називаємо машиною Тюрінга?
2. Що називають машинним словом (або конфігурацією)?
3. Сформулюйте гіпотезу Тюрінга.
4. Які основні етапи машини Тюрінга?
5. Які різновиди машини Тюрінга ви можете сформулювати?
6. Які способи задання машини Тюрінга ви знаєте?

ТЕМА 4
АЛГОРИТМІЧНІ МОДЕЛІ МАШИНИ ТЮРІНГА

4.1. Обчислювальні функції.

4.2. Алгоритмічні моделі на основі детермінованих пристроїв. Машина Тюрінга.

4.1. Обчислювальні функції

З метою формалізації поняття алгоритму, починаючи з 30-х рр. XX ст., було запропоновано кілька алгоритмічних моделей.

Історично першою формалізацією поняття алгоритму були обчислювальні функції. Ґрунтується модель на положенні, що будь-яку алгоритмічну проблему можна звести до обчислення значень цілочисельної функції цілочисельних аргументів.

Під ***числовою*** ми будемо розуміти функцію скінченного числа змінних, яка задана на множині $N \cup \{0\}$ (або на деякій її підмножині) і набуває значень в $N \cup \{0\}$. Через f^n будемо позначати n-місну числову функцію (тобто функцію n змінних).

Числова функція $f^n = (x_1, x_2, \dots, x_n)$ називається ***обчислюваною,*** якщо існує алгоритм, за допомогою якого можна обчислити значення цієї функції для будь-якого набору значень аргументів із області визначення функції. Зазначимо, що розглядатимемо так звані часткові функції, тобто визначені загалом не для всіх значень аргументів.

Базисні функції:

1. $S(x) = x + 1$ (функція слідування);
2. $O(x) = 0$ (нуль-функція);
3. $I_m^n(x_1, x_2, \dots, x_n) = x_m$ (функції-проєктори $1 \le m \le n$).

Правила утворення нових функцій із базисних:

1) Оператор регулярної суперпозиції

Нехай $k \in \mathbb{N} \cup \{0\}, n \in \mathbb{N}, f^n, g_1^m, g_2^m, \dots, g_n^m$ – числові функції. Будемо говорити, що функція h^m отримана за допомогою *оператора регулярної суперпозиції* із функцій $f, g_1, g_2, \dots, g_n$, якщо для всіх $x_1, x_2, \dots, x_n$ справедлива рівність:

$$h(x_1, x_2, \dots, x_m) = f(g_1(x_1, x_2, \dots, x_m), g_2(x_1, x_2, \dots, x_m), \dots, g_n(x_1, x_2, \dots, x_m).$$

Наприклад, функція $h(x, y) = 2xy + 20x - 10y + 1$ – отримана із функцій $f(x, y, z) = 2x + 3y - z, g_1(x, y) = 2xy, g_2(x, y) = 7x - y, g_3(x, y) = 7y + x - 1$ за допомогою оператора регулярної суперпозиції, оскільки:

$$f\big(g_1(x, y), g_2(x, y), g_3(x, y)\big) = 2g_1(x, y) + 3g_2(x, y) - g_3(x, y) =$$
$$= 2xy + 3(7x - y) - (7y + x - 1) = 2xy + 21x - 3y - 7y - x + 1 =$$
$$= 2xy + 20x - 10y + 1 = h(x, y).$$

2) Оператор примітивної рекурсії

Нехай $n \in \mathbb{N}, g^n, f^{n+2}$ – числові функції. Говорять, що функція h^{n+1} – утворена із функцій g і f за допомогою *оператора примітивної рекурсії*, якщо для всіх $x_1, x_2, \dots, x_n$ справедлива рівність:

$$h(x_1, x_2, \dots, x_n, 0) = g(x_1, x_2, \dots, x_n);$$

$$h(x_1, x_2, \dots, x_n, y+1) = f(x_1, x_2, \dots, x_n, y, h(x_1, x_2, \dots, x_n, y)).$$

Щоб знайти функцію, отриману із функцій за допомогою оператора примітивної рекурсії, розглядаємо послідовно:

$$h(x, 0) = g(x) = x;$$

$$h(x, 1) = f\big(x, y, h(x, 0)\big) = x \cdot h(x, 0) = x \cdot x = x^2;$$

$$h(x, 2) = f\big(x, y, h(x, 1)\big) = x \cdot h(x, 1) = x \cdot x^2 = x^3;$$

$$h(x, 3) = f\big(x, y, h(x, 2)\big) = x \cdot h(x, 2) = x \cdot x^3 = x^4.$$

Помічаємо, що $h(x, y) = x^{y+1}$. Залишається використати метод математичної індукції.

3) Оператор мінімізації

Нехай $n \in \mathbb{N} \cup \{0\}, f^{n+1}$ – числові функції. Говорять, що функція h^n – утворена із функції f за допомогою *оператора мінімізації*, у випадку, коли виконуються умови:

а) $h(x_1, x_2, \dots, x_n) = 0$, якщо $f(x_1, x_2, \dots, x_n, 0) = 0$;

б) $h(x_1, x_2, \dots, x_n) = k$, якщо $f(x_1, x_2, \dots, x_n, i)$ для всіх $i \in \overline{1, k-1}$ – визначені і не дорівнюють 0, а $f(x_1, x_2, \dots, x_n, k) = 0$.

Інакше кажучи, якщо $h(x_1, x_2, \dots, x_n)$ дорівнює найменшому значенню останнього аргументу функції, за якого $f(x_1, x_2, \dots, x_n, k) = 0$, а для всіх попередніх значень визначена і відмінна від 0.

Так, наприклад, функція $h(x, y) = x + y$ – утворена із функції $f(x, y, z) = x + y - z$ – за допомогою оператора мінімізації, оскільки найменше значення z, за якого для заданих x і y значення функції $h(x, y)$ дорівнює 0, є сума $x + y$, а для всіх значень z, менших за $x + y$, функція $h(x, y)$ – визначена, і її значення відмінні від 0.

Функція називається *частково рекурсивною*, якщо вона або є базисною, або її можна отримати із базисних за допомогою скінченного числа операторів суперпозиції, примітивної рекурсії і мінімізації. Функція називається *примітивно рекурсивною,* якщо вона або є базисною, або її можна отримати із базисних за допомогою скінченного числа операторів суперпозиції і примітивної рекурсії.

Кожна примітивно рекурсивна функція є частково рекурсивною. Кожна частково рекурсивна функція є обчислювальною.

Теза (гіпотеза) Черча (перша гіпотеза про розв'язки між класами інтуїтивних і точних алгоритмів): Клас частково рекурсивних функцій збігається із класом обчислювальних функцій.

4.2. Алгоритмічні моделі на основі детермінованих пристроїв. Машина Тюрінга

Другий підхід до уточнення поняття алгоритму базується на уявленні про алгоритм як про деякий детермінований пристрій, що здатен виконувати в кожний конкретний момент лише чітко фіксовані елементарні дії. Основними теоретичними моделями цього типу є фінітний процес Поста і машина Тюрінга. Введення поняття МТ (1936) стало однією із найвдаліших спроб дати точний математичний еквівалент інтуїтивного уявлення про алгоритм.

Машина Тюрінга – це математична модель ідеалізованої цифрової машини (а не фізична машина). Для кращого розуміння роботи МТ її можна уявляти у вигляді автоматично працюючого пристрою, який складається із головки (що зчитує і записує символи) і стрічки, необмеженої в обидва боки і розбитої на комірки (клітинки), в кожній із яких може бути записаний певний символ.

Зафіксуємо два ***алфавіти*:**

- зовнішній алфавіт $A = \{a_0, a_1, \dots, a_n\}, n \geq 1$;
- внутрішній алфавіт $Q = \{q_0, q_1, \dots, q_m\}, m \geq 1$;

і додатково вимагатимемо, щоб $A \cap Q = \emptyset$, а символи $R, L, \rightarrow$ не належали до $A \cup Q$.

У кожен конкретний момент часу, перебуваючи в певному стані q_i $(i \neq 0)$, головка оглядає певну комірку; в кожній із комірок записана рівно одна буква алфавіту A (a_0 – символ порожньої комірки) (рис. 4.1):

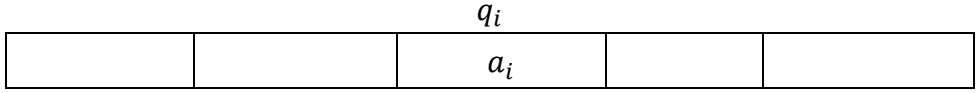


Рис. 4.1. Схема роботи МТ

Залежно від того, в якому стані q_i перебуває машина і який символ a_j міститься в комірці, вона виконує елементарну дію, що має три складові:

I. переходить в інший стан q_k;

II. змінює або не змінює вміст комірки (на букву a_i);

III. і рухається або вправо на одну комірку (тобто починає оглядати найближчу справа комірку), або вліво, або не рухається.

Робота машини полягає в послідовному виконанні елементарних дій, тому її можна повністю описати сукупністю команд, які для кожної пари (q_i, a_j) вказують, яку саме елементарну дію повинна в даний момент виконувати машина.

Розглянемо об'єднаний алфавіт $B = A \cup Q \cup \{R, L, \rightarrow\}$. Серед слів алфавіту B виділимо *команди* (слова одного з наступних трьох типів):

1) $q_i a_j \rightarrow q_k a_i$;

2) $q_i a_j \rightarrow q_k a_i R$;

3) $q_i a_j \rightarrow q_k a_i L$.

Команди інтерпретуються так: якщо машина знаходиться в стані q_i і оглядає комірку, в який записана буква a_j, то вона повинна послідовно виконувати наступне (в залежності від типу команди) (табл. 4.1):

Таблиця 4.1

$q_i a_j \rightarrow q_k a_i$	$q_i a_j \rightarrow q_k a_i R$	$q_i a_j \rightarrow q_k a_i L$
I. перейти в стан q_k		
II. замінити букву a_j в комірці на a_i		
III. продовжувати оглядати ту ж саму комірку	III. зміститись на одну комірку вправо	III. зміститись на одну комірку вліво

Сукупність команд називається *програмою роботи* МТ (функціональною схемою).

Домовимось стан q_0 називати *заключним* станом машини і вважати, що опинившись в цьому стані, машина зупиняється. Стан q_1 називатимемо *початковим*.

Конфігурацією називається кожне слово в алфавіті $A \cup Q$, в яке входить лише один символ $q_i \in Q$, причому лише один раз і не на останньому місці. Так, наприклад, в алфавіті $\{a_0, 1\} \cup \{q_1, q_2\}$ $a_0 1 a_0 a_0 q_1 a_0, a_0 1 q_2 1$ – конфігурації, $a_0 111 q_1,\ a_0 1 q_1 q_1 a_0$ – не конфігурації. У конфігурації $a_{j_1} \dots a_{j_{s-1}} q_i a_{j_s} \dots a_{j_n}$ буква внутрішнього алфавіту q_i задає стан МТ; буква a_{j_s}, яка слідує за нею, – це буква з комірки, яку в даний момент оглядає машина. Наприклад, конфігурація, яка відповідає внутрішньому стану q_5, слову на стрічці $a_1 a_3 a_2 a_2$ та огляду комірки із буквою a_3, записується як $a_1 q_5 a_3 a_2 a_2$. Конфігурація називається *початковою* (відповідно *заключною*), якщо стан, у якому водночас знаходиться машина, – початковий q_1 (відповідно заключний q_0).

Початкову конфігурацію вважатимемо стандартною, якщо машина оглядає комірку з першою (тобто крайньою зліва) буквою заданого слова (тобто якщо має місце конфігурація $q_i a_{j_1} a_{j_2} \dots a_{j_m}$).

Якщо машина Тюрінга, почавши роботу із деяким словом $a_{j_1} \dots a_{j_{s-1}} q_i a_{j_s} \dots a_{j_n}$, записаним на стрічці, прийде в заключний стан q_0, то вона застосовна до даного слова. Результатом її роботи в такому випадку вважається слово, яке буде написане

на стрічці в заключній конфігурації. Якщо ж машина в жоден момент не перейде в заключний стан, то вона вважається незастосовною до заданого слова і результат її роботи – невизначений.

Функція називається *обчислюваною за Тюрінгом*, якщо існує машина Тюрінга, яка обчислює значення цієї функції для будь-якого набору значень аргументів із області визначення функції і не застосовна до наборів значень аргументів, що не входять до області визначення цієї функції.

Теза (гіпотеза) Тюрінга: клас функцій, обчислюваних за Тюрінгом, збігається із класом обчислюваних функцій.

Як і тезу Черча, довести це твердження неможливо, а для спростування потрібно вказати обчислювану функцію, не обчислювану за Тюрінгом.

ПИТАННЯ ДО ТЕМИ 4

1. Яка функція називається обчислюваною за Тюрінгом?
2. Які є види функцій?
3. Які ви знаєте правила утворення нових функцій із базисних?
4. Який оператор називається оператором регулярної суперпозиції?
5. Який оператор називається оператором мінімізації?
6. Який оператор називається оператором примітивної рекурсії?
7. Яка функція називається частково рекурсивною?

ТЕМА 5
НОРМАЛЬНІ АЛГОРИТМИ МАРКОВА

5.1. Опис нормального алгоритму Маркова.
5.2. Робота нормального алгоритму Маркова.
5.3. Композиція нормальних алгоритмів Маркова.
5.4. Еквівалентність машини Тюрінга та нормальних алгоритмів Маркова.
5.5. Нерозв'язні алгоритмічні проблеми.

5.1. Опис нормального алгоритму Маркова

Нормальний алгоритм Маркова (Марков Андрій Андрійович (молодший, 1903–1979) – радянський математик, основоположник радянської школи конструктивної математики, автор поняття нормального алгоритму) (НАМ) задається так.

Нехай $\mathcal{A}$ – алфавіт (непорожня скінченна множина символів), $\mathcal{A} \neq \emptyset, |\mathcal{A}| < \infty$. НАМ записується у вигляді лінійно впорядкованої множини підстановок $P(P \neq \emptyset,\ |P| < \infty)$:

$$P = \{\alpha_i \to \beta_i | \alpha_i, \beta_i \in \mathcal{A}^*\ i = \overline{1,n}\}.$$

Тут $\mathcal{A}^*$ – множина всіх слів над алфавітом $\mathcal{A}$. Слово – скінченна послідовність символів з даного алфавіту. $\mathcal{A}^*$ містить також порожнє слово (яке не містить жодного символу), яке будемо позначати через ε. У множині P обов'язкова наявність хоча б однієї заключної підстановки, яку будемо позначати через $\alpha_k \to \beta_k$.

Окремо взята підстановка $\alpha_i \to \beta_i$ застосовується у випадку, коли поточне слово містить α_i як підслово. Застосування підстановки полягає в тому, що перше входження α_i замінюється на β_i. Порожнє підслово ε міститься між будь-якими літерами даного слова і перше його входження міститься перед першою літерою слова.

Алфавітом будемо називати непорожню скінченну множину Σ деяких символів. *Елементи алфавіту* називатимемо також буквами. *Слово в алфавіті* Σ – це скінченна або порожня послідовність його букв. Множину всіх слів в алфавіт Σ позначатимемо Σ^*.

Основною операцією на множині слів є їх приписування одне до одного: якщо є слова $\alpha = a_0 a_1 \dots a_m, \beta = b_0 b_1 \dots b_n$, то можна утворити слова $\alpha\beta = a_0 a_1 \dots a_m b_0 b_1 \dots b_n$ і $\beta\alpha = b_0 b_1 \dots b_n a_0 a_1 \dots a_m$. Загалом слова $\alpha\beta$ і $\beta\alpha$ – різні. Якщо $\alpha \epsilon \Sigma^*, \beta \epsilon \Sigma^*, \gamma \epsilon \Sigma^*$, то $(\alpha\beta)\gamma = \alpha(\beta\gamma)$. Порожнє слово позначатимемо Λ. Зрозуміло, що $\Lambda\alpha = \alpha\Lambda = \alpha$.

Слово β називається *підсловом* слова α, якщо $\alpha = \gamma\beta\delta$ для деяких слів γ і δ. Якщо $\alpha = \gamma\beta\delta$ і $\alpha = \gamma_1\beta\delta_1$, причому $\gamma \neq \gamma_1$, то говорять про різні входження підслова β в слово α. Важливим є перше входження слова в інше.

Якщо Σ і Σ_1 – два алфавіти, причому $\Sigma \subset \Sigma_1$, то Σ_1 називається *розширенням* алфавіту Σ.

Алгоритм Маркова (або алгорифми) являють собою деякі правила перетворення слів у деякому алфавіті.

Схемою S в алфавіті Σ називається впорядкований набір трійок $((\alpha_1, \beta_1, \omega_1), (\alpha_2, \beta_2, \omega_2), \dots, (\alpha_n, \beta_n, \omega_n))$, у яких $\alpha_i \in \Sigma^*, \beta_i \in \Sigma^*, \omega_i \epsilon \{0,1\}$ $(i = 1,2, \dots, n)$.

Нормальним алгоритмом в алфавіті Σ називається пара (Σ, S), яка складається з алфавіту Σ і схеми S у цьому алфавіті.

Нехай $\alpha \in \Sigma^*$ – нормальний алгоритм зі схемою $S = ((\alpha_1, \beta_1, \omega_1),\ (\alpha_2, \beta_2, \omega_2), \dots, (\alpha_n, \beta_n, \omega_n))$. Якщо кожне зі слів $\alpha_1, \alpha_2, \dots \alpha_n$ не є підсловом слова α, то будемо говорити, що слово α не піддається алфавіту A. Якщо k – найменший номер, для якого α_k є підсловом слова α і β, – результат заміни першого входження слова α_k в слово α на слово β_k, то будемо говорити, що *A просто переводить α* в β, якщо $\delta_k = 0$ (позначатимемо $A\colon \alpha \to \beta$) і *заключно переводить α* в β, якщо $\delta_k = 1$ (позначатимемо $A\colon \alpha \to\cdot \beta$). Символи $\to$ і $\cdot$ не повинні належати алфавіту Σ. Якщо нормальний алгоритм A просто або заключно переводить слово α в слово β, то говоритимемо, що A переводить α в β.

Нормальний алгоритм A перетворює слово α в слово β (позначатимемо $A(\alpha) = \beta$), якщо існує така послідовність $\gamma_1, \gamma_2, \dots \gamma_m$ слів алфавіту Σ, якщо виконуються умови:

1) $\gamma_0 = \alpha, \gamma_m = \beta$;

2) якщо $m = 0$, то α не піддається алгоритму A;

3) якщо A просто переводить γ_i в γ_{i+1} для всіх $i < m$;

4) якщо $m > 0$ і не має місця $A\colon \gamma_{m-1} \to\cdot \gamma_m$, то $A\colon \gamma_{m-1} \to \gamma_m$ і γ_m не піддається алгоритму A.

Зрозуміло, що якщо алгоритм A перетворює слово α в слово β, то β однозначно визначається за α і A. Якщо послідовність слів $\gamma_0, \gamma_1, \dots \gamma_m$, для якої $A\colon \gamma_{i-1} \to \gamma_i$ і $\gamma_0 = \alpha$, є нескінченною, то говорять, що алгоритм A – незастосовний до слова α.

Трійку $(\alpha_i,\ \beta_i, \omega_i)$ зі схеми S зручно записувати в такому вигляді: $\alpha_i \to \beta_i$, якщо $\omega_i = 0$ і $\alpha_i \to\cdot \beta_i$, якщо $\omega_i = 1$. У такій схемі трійки (підстановки) будемо нумерувати.

Прикладом НАМ є алгоритм $A = (\Sigma, S)$, де $\Sigma = a_0, a_1$, із схемою:

1) $a_0 \to \wedge\Lambda$;

2) $a_1 \to a_1$,

який кожне слово, що має хоча б одне входження букви a_0, перетворює в слово, яке одержуємо із даного викреслювання самого лівого входження букви a_0.

Порожнє слово він перетворює в порожнє слово, а до непорожніх слів, у записі яких немає букви a_0, алгоритм незастосовний.

5.2. Робота нормального алгоритму Маркова

Нормальні алгоритми Маркова (НАМ) – формалізація поняття алгоритму, що є системою послідовних застосувань підстановок до слів певного алфавіту, введена математиком А. А. Марковим у 1956 р.

Означення 5.1. ***Нормальний алгоритм Маркова*** (НАМ) – це кінцевий впорядкований набір підстановок виду $P_i \to (.)\ Q_i,\ i = 1,2, \dots, n.\ P_i \to Q_i$ – звичайна підстановка, яка означає, що крайнє ліве входження підслова P_i в W замінюється на Q_i, $P_i \to .\ Q_i$ – кінцева підстановка, тобто після її виконання робота алгоритму завершується.

Початкове слово $W \in \mathcal{A}^*$ переробляється за допомогою алгоритму так. Із першої підстановки алгоритм шукає перше ліве входження підслова P_i в слові W. Якщо воно знайдене, то воно замінюється на Q_i, після чого список підстановок проглядається з початку. Якщо ж входження P_i не було знайдене, то вибирається наступна підстановка. Якщо була виконана заключна підстановка (з крапкою), то процес переробки слова завершується, і тоді говорять, що алгоритм є застосовним до даного слова. Може виявитися, що процес не завершується, тобто жодна із заключних підстановок не застосовувалася в процесі переробки слова. У цьому випадку говорять, що алгоритм не застосовується до даного слова.

Роботу НАМ A можна описати так. Нехай задано слово α. Знаходимо першу в схемі S алгоритму A формулу підстановки $\alpha_i \to \beta$ (або $\alpha_i \to\cdot \beta$), таку, що α_i є підсловом слова α. Підставляємо в слово α слово β_i замість першого входження α_i в α. Нехай γ_i – результат цієї підстановки. Якщо дана підстановка виявилась заключною формулою підстановки (тобто $\alpha_i \to \beta$), то робота алгоритму закінчується і $A\ (\alpha) = \beta$. Якщо формула підстановки виявилась простою (тобто $\alpha_i \to \beta$), то до слова γ_i застосовуємо той же пошук, який здійснювали до слова α тощо. Якщо в результаті одержимо таке слово γ_i, що жодне зі слів $\alpha_1, \alpha_2, \dots, \alpha_n$ не входить в γ_i як підслово, то робота алгоритму закінчується і $A\ (\alpha) = \beta$. Якщо описаний процес не закінчується ніколи, то робимо висновок, що алгоритм A – незастосовний до слова α.

Функція називається *нормально обчислюваною,* якщо існує такий нормальний алгоритм, який обчислює значення цієї функції для будь-якого набору значень аргументів із області визначення функції і незастосовний до наборів значень аргументів, що не входять в область визначення даної функції.

Принцип нормалізації Маркова: клас нормально обчислюваних функцій збігається із класом обчислюваних функцій. Дане твердження – гіпотеза, яку, як і гіпотези Черча і Тюрінга, довести неможливо.

Теорема 5.1. (про зв'язок між класами числових функцій). **Такі класи числових функцій збігаються:**

1) клас всіх частково рекурсивних функцій;
2) клас всіх функцій, обчислювальних за Тюрінгом;
3) клас всіх нормально обчислювальних функцій.

Дана теорема має строге доведення. Вона означає, що теорії частково рекурсивних функцій, машин Тюрінга і НАМ – рівносильні між собою, що непрямо підтверджує справджуваність гіпотез Черча, Тюрінга і Маркова.

Існують також і інші уточнення поняття алгоритму, для яких доведена рівносильність їх розглянутим поняттям.

На кожному кроці до поточного слова використовується завжди перша підстановка, яку можна застосувати.

Алгоритм закінчує роботу у двох випадках:

а) використали одну із заключних підстановок (нормальне завершення роботи);

б) жодну з підстановок не можна застосувати до поточного слова (ненормальне, або аварійне, завершення роботи).

Приклад 5.1. Анулювати слово в алфавіті $\{a, b\}$.

Пояснення

1) $a \to \varepsilon$;

2) $b \to \varepsilon$;

3) $\varepsilon \to \varepsilon$.

Спочатку застосовується підстановка 1). Видаляються всі літери a. Потім підстановка 2). Видаляються всі літери b. І після цього підстановка 3) (заключна).

Розберемо цей приклад покроково для слова $aaba$ (табл. 5.1).

Таблиця 5.1

Покроковий протокол роботи НАМ для прикладу 5.1

№ кроку	Підстановка	Слово
0	–	$\underline{a}aba$
1	1)	$\underline{a}ba$
2	1)	$b\underline{a}$
3	1)	$\underline{b}$
4	2)	$\underline{\varepsilon}$
5	3)	ε

У цій та наступних таблицях для протоколів роботи НАМ підкреслений символ – перше входження підслова для виконання наступного кроку.

Приклад 5.2. Інвертувати слово в алфавіті $\{a, b\}$. Замість a пишеться b, а замість b пишеться a.

Пояснення

1) $* a \to b *$;

2) $* b \to a *$;

3) $* \to \varepsilon$;

4) $\varepsilon \to *$.

У вхідному слові зірочки немає, тому використовується підстановка 4). Ставимо зірочку на початку слова. Потім, використовуючи підстановки 1) та 2), рухаємо зірочку вправо і замінюємо на шляху кожну літеру a на b і літеру b на a. Якщо після зірочки немає жодної літери, застосовуємо 3). Це заключна підстановка, тому закінчуємо роботу алгоритму.

Розберемо цей приклад покроково для слова $abaa$ (табл. 5.2).

Таблиця 5.2

Покроковий протокол роботи НАМ для прикладу 5.2

№ кроку	Підстановка	Слово
0	–	$\underline{\varepsilon}abaa$
1	4)	$\underline{* a}baa$
2	1)	$b * \underline{ba}a$
3	2)	$ba * \underline{aa}$
4	1)	$bab * \underline{a}$
5	1)	$babb \underline{*}$
6	3)	$babb$

Приклад 5.3. Обернення слова в алфавіті $\{a, b\}$. Якщо вхідне слово $w = \alpha_1\alpha_2 \ldots \alpha_k, k \geq 0$, то оберненим до нього буде слово $w^R = \alpha_k\alpha_{k-1} \cdots \alpha_1$. Для скорочення запису введемо $\xi, \eta \in \{a, b\}$ і $\tilde{\eta} \in \{\tilde{a},\ \tilde{b}\}$.

Пояснення

1) $\xi\tilde{\eta} \to \tilde{\eta}\xi$;

2) $\tilde{\eta} \to \eta$;

3) $* a \to \tilde{a} *$;

4) $* b \to \tilde{b} *$;

5) $* \to \varepsilon$;

6) $\varepsilon \to *$.

Основний принцип цього алгоритму полягає в тому, що, рухаючи зірочку вправо по слову, ми позначаємо літеру хвилькою, яку проштовхуємо на початок слова. Тобто якщо в початковому слові є літера з хвилькою та зірочка, то спочатку літера з хвилькою просувається вперед (підстановка 1)) і потім хвилька зникає (підстановка 2)). Якщо літери з хвилькою немає, але є зірочка, то намагаємося застосувати підстановки 3) або 4). У позитивному випадку зірочка просувається на один символ вправо і цей символ позначається хвилькою. Далі за допомогою підстановок 1) та 2) літера з хвилькою просувається вперед і хвилька стирається.

Якщо ж підстановки 3) та 4) не можна використати, то виконуємо заключну підстановку 5), стираємо зірочку та закінчуємо роботу. Якщо немає зірочки чи літери з хвилькою, виконуємо підстановку 6).

Розберемо цей приклад покроково для слова abb (табл. 5.3).

Таблиця 5.3

Покроковий протокол роботи НАМ для прикладу 5.3

№ кроку	Підстановка	Слово
0	–	$\underline{\varepsilon}abb$
1	6)	$\underline{*\,}abb$
2	3)	$\underline{\tilde{a}} * bb$
3	2)	$a * \underline{bb}$
4	4)	$\underline{a\tilde{b}} * b$
5	1)	$\underline{\tilde{b}a} * b$
6	2)	$ba * \underline{b}$
7	4)	$b\underline{a\tilde{b}} *$
8	1)	$\underline{b\tilde{b}}a *$
9	1)	$\underline{\tilde{b}}ba *$
10	2)	$bba\,\underline{*}$
11	5)	bba

Приклад 5.4. НАМ для складання двох чисел в унарній системі, тобто алгоритм обчислює функцію $f(x, y) = x + y$ в алфавіті $A = \{|,\ *\}$. Список підстановок складається з двох (це приклад цілком рекурсивного алгоритму).

Пояснення

1) $* \,|\rightarrow|\, *$;

2) $*\rightarrow .\,\Lambda$.

[не рекурсивний НАМ 1) $|*| \rightarrow .\,||$]

Процес обробки слова полягає в наступному (в дужках вказано номер застосовуваної підстановки).

Таблиця 5.4

Покроковий протокол роботи НАМ для прикладу 5.4

№ кроку	Підстановка	Слово
0	–	$*\,\|\|\|\|\|\|\|$
1	1)	$\|\|\|\,*\,\|\|\|\|$
2	1)	$\|\|\|\|\,*\,\|\|\|$
3	1)	$\|\|\|\|\|\,*\,\|\|$
4	1)	$\|\|\|\|\|\|\,*\,\|$
5	2)	$\|\|\|\|\|\|\|\,*$
6	2)	$\|\|\|\|\|\|\|$

Приклад 5.5. Даний алгоритм перетворює двійкові числа в «унарні» (в яких записом цілого невід'ємного числа N є рядок з N паличок). Наприклад, двійкове число 101 перетвориться в 5 паличок: |||||.

Пояснення

Алфавіт $\{0, 1, |, \Lambda\}$.

Правила:

1. $1 \to 0|$;
2. $|0 \to 0||$;
3. $0 \to .\Lambda$.

Покрокове виконання алгоритму у разі застосування алгоритму за наведеною вище схемою до слова 101 буде отримувати слова:

1. $0|01$
2. $0|00|$
3. $00|0|$
4. $00|0|||$
5. $000|||||$
6. $00|||||$
7. $0|||||$
8. $|||||$

Завдання 1. Побудувати НАМ, який:

1) визначає останній символ слова в алфавіті $\{a, b\}$;
2) порівнює кількість входжень літер a та b у слові $\{a, b\}$;
3) переводить число з унарної у двійкову систему;
4) переводить число з двійкової в унарну систему;
5) обчислює $x + 1$ у двійковій системі.

5.3. Композиція нормальних алгоритмів Маркова

Як і для машин Тюрінга, для алгоритмів Маркова також реалізується композиція, але трохи складніше. Тут для різних НАМ потрібно використовувати алфавіти, які не перетинаються, і перехід від одного алгоритму до іншого відбувається перейменуванням символів поточного слова.

5.4. Еквівалентність МТ та нормальних алгоритмів Маркова

Під еквівалентністю двох або більше алгоритмів варто розуміти те, що кожна задача, яка розв'язується одним з алгоритмів, буде розв'язуватися всіма іншими.

Теорема 5.2. За класом задач, які розв'язуються, НАМ та машина Тюрінга еквівалентні.

Доведення. За даною МТ будуємо НАМ. Алфавіт НАМ: $A_1 = A \cup \{*_i \mid q_i \in Qi = \overline{1,n}\}, A$ – алфавіт МТ. За кожною командою МТ будуємо підстановку для НАМ (табл. 5.5).

Таблиця 5.5

Перехід від МТ до НАМ

Команда	Підстановка
$q_1a \to q_2bR$	$*_1\, a \to b\, *_2$
$q_1a \to q_2bL$	$j\, *_1\, a \to b\, *_2\, jb$
$q_1a \to q_2bS$	$*_1\, a \to b\, *_2\, b$
$q_1a \to !\, b \begin{bmatrix} R \\ L \\ S \end{bmatrix}$	$*_1\, a \to b$

Приклад 5.6. Продумати ситуацію на кінцях поточного слова.

Пояснення. За заданим НАМ будуємо МТ. Для кожної підстановки НАМ будуємо МТ, яка перевіряє, чи можна її використати. У позитивному випадку підстановка використовується. Далі застосовуємо техніку композицій машин Тюрінга за такою схемою (рис. 5.1).

Нехай $P = \{\alpha_i \to \beta_i \mid i = \overline{1,n}\}, \alpha_k \to \beta_k \in P$.

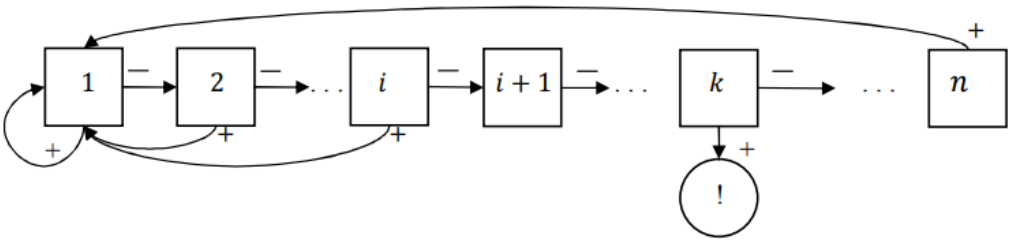


Рис. 5.1. Схема побудови МТ за заданим НАМ

5.5. Нерозв'язні алгоритмічні проблеми

Алгоритмічна (або масова) *проблема* – це проблема, в якій необхідно знайти єдиний метод (алгоритм) розв'язування всіх задач із деякого нескінченного класу задач. Задачі даного класу формулюють у вигляді питання, яке вимагає відповіді або «так», або «ні».

В історії математики відомо багато проблем, для яких не вдавалось підібрати загального алгоритму розв'язання. У зв'язку з цим виникла потреба в уточненні поняття алгоритму, і лише тоді з'явилась можливість встановлювати наявність алгоритмічно нерозв'язаних проблем.

Наведемо приклади алгоритмічно нерозв'язаних проблем.

1. Проблема обчислення значень функції, не обчислюваної за Тюрінгом (детальніше розглядали раніше).

2. Проблема розпізнавання самозастосовності МТ. Припустимо, що на стрічку МТ записано її власну програму в алфавіті машини. Якщо машина застосовна до такої конфігурації, то будемо називати її самозастосовною, в іншому випадку – несамозастосовною. Виникає масова проблема розпізнавання самозастосовних МТ: за заданою програмою МТ визначити, до якого класу машин (самозастосовних чи несамозастосовних) вона належить.

3. Проблема вирішення в логіці предикатів.

4. Десята проблема Гільберта (задача про можливість розв'язання діофантових рівнянь) (1901 р., міжнародний математичний конгрес): нехай дано алгебраїчне рівняння $f(x_1, x_2, \dots, x_n) = 0$ з цілими коефіцієнтами. Необхідно вказати метод, згідно з яким можна було б за скінченну кількість кроків встановити, чи має дане рівняння розв'язки в цілих числах. У 1970 р. радянський математик Ю. В. Матіясевич показав, що такого алгоритму не існує.

5. Проблема тотожності груп (зі скінченним числом твірних елементів і скінченним числом визначальних співвідношень): необхідно знайти алгоритм, який давав би змогу для будь-якої групи зі скінченним числом твірних елементів і скінченним числом визначальних співвідношень за скінченне число кроків дати відповідь на питання про рівність двох слів, записаних твірними елементами даної групи. Сформульована в 1912 р., розв'язана негативно лише в 1952 р. П. С. Новіковим. Аналогічну проблему для півгруп у 1947 р. розв'язали негативно А. А. Марков і Е. Пост, незалежно один від одного.

6. Проблема подання для матриць. Будемо говорити, що деяка матриця A подана через матриці $\Sigma = \{B_1, B_2, \dots, B_n\}$, якщо $A = B_{i_1}, B_{i_2}, \dots, B_{i_n}$, для деяких матриць (не обов'язково різних) із даної множини матриць. Необхідно знайти алгоритм, за допомогою якого за скінченне число кроків для будь-якої матриці A і будь-якої множини матриць Σ дати відповідь на питання: «Чи можна подати матрицю A через матриці множини Σ?».

Алгоритмічна нерозв'язність задач певного класу зовсім не означає, що для деяких конкретних задач цього класу не існує алгоритму розв'язування. Йдеться про неможливість розв'язання всіх задач одним і тим самим методом. Водночас кожна окрема задача може мати свій індивідуальний спосіб розв'язування. Навіть більше, може трапитись так, що не лише кожна окрема задача, а й цілі підкласи задач можна розв'язати своїм індивідуальним способом. Тому якщо проблема – нерозв'язна в загальному випадку, необхідно шукати її розв'язні часткові випадки. Наприклад, якщо проблема тотожності – нерозв'язна для класу всіх скінченнопороджених груп, то цікаво було б дослідити, чи розв'язна вона для тих чи інших

підкласів груп (скінченнопороджених абелевих, нільпотентних (алгебра Лі (векторний простір, на якому визначена операція комутації) називається *нільпотентною,* якщо її нижній центральний ряд зрештою стає рівним нулю), розв'язних, локально розв'язних тощо). Важливо знайти той найбільш широкий клас груп, для якого проблема розв'язується позитивно. Саме пошук максимально широких класів задач, для яких алгоритмічні проблеми розв'язуються позитивно, і є єдиним із важливих чинників розвитку сучасної математики.

ПИТАННЯ ДО ТЕМИ 5

1. У якій послідовності можна представити алгоритм Маркова?
2. Що називають схемою в алфавіті Σ?
3. Які основні етапи роботи побудови НАМ?
4. Що називають нормальним алгоритмом?
5. Що ми розуміємо під поняттям еквівалентності МТ та НАМ?
6. У якому випадку говорять, що нормальний алгоритм A перетворює слово α в слово β?
7. У якому випадку говорять, що нормальний алгоритм A – незастосовний до слова α в слово β? А в якому випадку, що слово не піддається алгоритму A?
8. Яка функція називається нормально обчислюваною?
9. Сформулюйте принцип нормалізації Маркова.
10. Які серед вказаних класів числових функцій збігаються:
- клас всіх обчислювальних функцій;
- клас всіх частково-рекурсивних функцій;
- клас всіх нормально обчислювальних функцій;
- клас всіх функцій, обчислювальних за Тюрінгом?

11. Наведіть приклади алгоритмічно нерозв'язних проблем.

ТЕМА 6
БЛОК-СХЕМА МАШИНИ ПОСТА

6.1. Опис блок-схеми Поста.

6.2. Робота блок-схеми Поста.

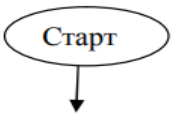
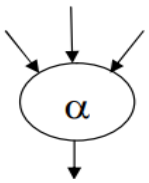
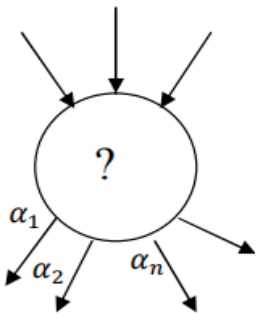
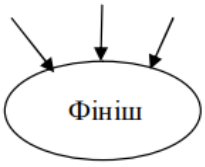
6.3. Машина Поста.

6.4. Еквівалентність МТ, нормального алгоритму Маркова та блок-схеми Поста.

6.1. Опис блок-схеми Поста

Блок-схема Поста (БСП) задається так. Нехай A – алфавіт БСП – алгоритмічна машина, яка задається у вигляді міченого орієнтованого мультиграфу, який може містити 4 типи вершин. На кожному кроці керування знаходиться на деякій вершині (див. табл. 6.1).

Таблиця 6.1

№	Назва	Позначення	Дія	Примітка
1	Стартова		Жодна дія не виконується, означає початок роботи алгоритму	Одна вершина для всього алгоритму, вхідних ребер немає, вихідне – одне
2	Породжуюча (генеруюча) вершина		Записуємо $\alpha \in A$ символ кінець поточного слова	Вхідних ребер може бути багато, але мінімум одне, вихідне ребро – одне
3	Тестова вершина		Видаляємо перший символ. Керування передаємо по ребру, що відповідає цьому символу	$?\notin A, A = \{a_1, a_2, \dots, a_n\}$. Вхідних ребер не менше одного, вихідних не більше $n+1$, де $n = \lvert A \rvert$. Ще одне ребро для випадку порожнього слова. Якщо деякий символ не зустрічається під час керування даної вершини, то відповідне ребро можна не малювати
4	Заключна вершина		Жодна дія не виконується, означає закінчення роботи алгоритму	Вхідних ребер – не менше одного, вихідних – немає. Можлива наявність кількох заключних вершин

6.2. Робота блок-схеми Поста

На вхід задається вхідне слово скінченної довжини із заданого алфавіту A блок-схеми Поста працює покроково. На кожному кроці рівно одна вершина є поточною. В початковий момент часу поточною є стартова вершина. Під час

переходу до наступного кроку над словом проводиться деяка дія відповідно до поточної вершини. Далі керування передається наступній вершині (відповідно вихідного ребра), наявність петель допускається. Алгоритм закінчує роботу, коли поточною є одна з заключних вершин.

Приклад 6.1. Збільшити кількість паличок удвічі (рис. 6.1).

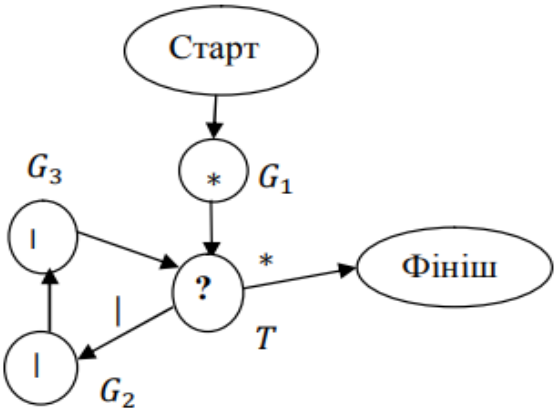


Рис. 6.1. Блок-схема Поста для прикладу 6.1

Пояснення. Ставимо у кінці слова зірочку. Далі видаляємо першу паличку у поточному слові і в кінець записуємо дві палички. Алгоритм закінчує роботу, коли під час керування тестової вершини зустрінеться зірочка.

Представимо даний приклад покроково для слова ||| (табл. 6.2).

Таблиця 6.2

Протокол роботи блок-схеми Поста для прикладу 6.1

№ кроку	Поточна вершина	Поточне слово
0	*Старт*	|||
1	G_1	||| *
2	T	|| *
3	G_2	|| *	
4	G_3	|| * ||
5	T		* ||
6	G_2		* |||
7	G_3		* ||||
8	T	* ||||
9	G_2	* |||||
10	G_3	* ||||||
11	T	||||||
12	*Фініш*	||||||

Приклад 6.2. З'ясувати, чи є слово в алфавіті $\{a, b\}$ поліномом ($a^n b^m$, $n \geq 0, m \geq 0$). Тобто поліном є слово, у якому після літери b жодного разу не зустрічається літера a (рис. 6.2).

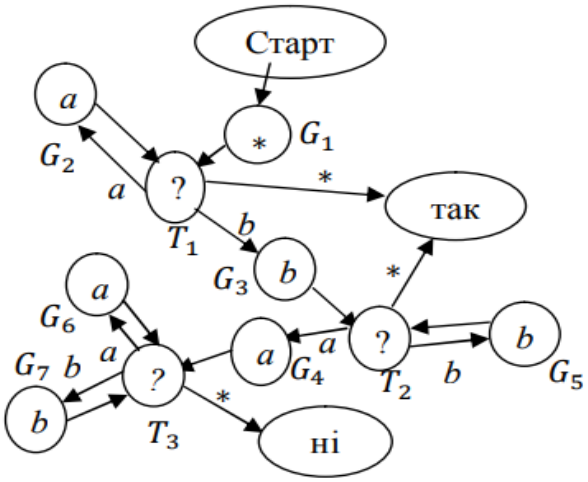


Рис. 6.2. Блок-схема Поста для прикладу 6.2

Пояснення. Ставимо в кінець слова зірочку G_1 і переходимо в T_1. Промотуємо літери a (T_1 та G_2), стираємо з початку слова та записуємо в кінець. Якщо в T_1 потрапляємо на зірочку, то вхідне слово має вигляд $a^n, n \geq 0$ і воно є поліномом. Якщо ж у T_1 потрапили на b, то дописуємо b та переходимо в T_2.

Далі промотуємо всі літери $b((T_2,\ G_3)$ і у випадку потрапляння на зірочку в T_2 вхідне слово має вигляд $a^n b^m, n \geq 0, m \geq 1$ – поліном.

Якщо ж у T_2 потрапили на літеру a, то переходимо в G_4, відновлюємо a, робимо прокрутку залишку слова ($T_3,\ G_6,\ G_7$) і у випадку потрапляння на зірочку закінчуємо роботу. Відповідь – «ні». Після літери b є літера a.

6.3. Машина Поста

Створення машини Поста дало змогу поглянути на алгоритм як на універсальний засіб для вирішення будь-якої проблеми, що може бути математично формалізована і переведена на мову однозначних інструкцій для виконання.

Тобто якщо конкретну проблему вдалося сформулювати, значить, її можливо вирішити, використовуючи набір інструкцій для обчислювального пристрою (рис. 6.3).

Постом був запропонований стандартний набір інструкцій, що містить шість пунктів.

1. Помітити комірку, якщо вона порожня.
2. Стерти мітку, якщо вона є.
3. Перемістится вліво на 1 комірку.

4. Перемістится вправо на 1 комірку.

5. Визначити, чи має комірка позначку, чи ні, і за результатом перейти на одну з двох зазначених інструкцій.

6. Зупинитися.

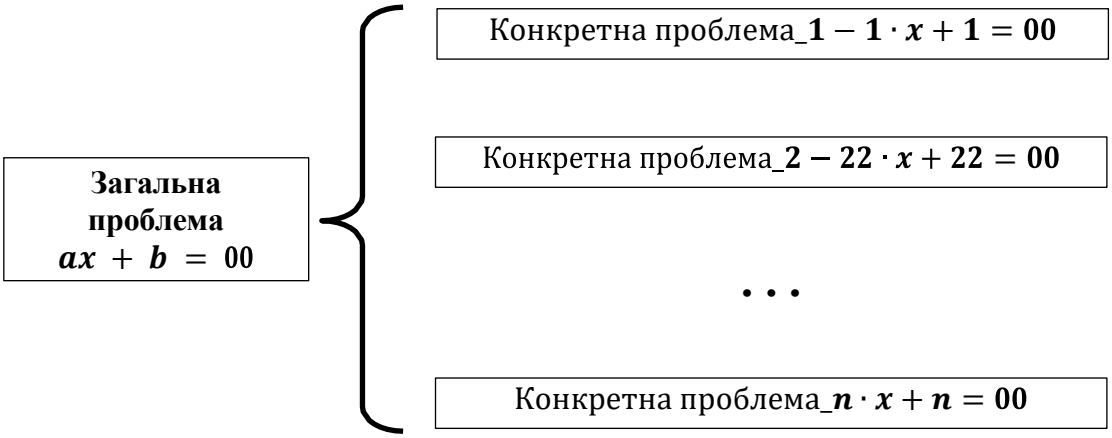


Рис. 6.3. Загальна проблема за Постом (приклад)

Цішість інструкцій дають змогу вирішувати будь-яку проблему за наявності введених Постом понять, до яких належать:

- набір інструкцій застосовується до загальної проблеми, якщо для кожної конкретної проблеми не виникає колізій в інструкціях 1 і 2, тобто ніколи програма не стирає мітку в порожній комірці і не встановлює мітку в позначеній комірці;
- набір інструкцій закінчується (за скінченну кількість інструкцій), якщо виконується інструкція (6);
- набір інструкцій задає фінітний 1 – процес, якщо набір може бути використаним, і закінчується для кожної конкретної проблеми;
- фінітний 1 – процес для загальної проблеми є 1 – рішення, якщо відповідь для кожної конкретної проблеми є правильною (це визначається зовнішньою силою, в сучасних термінах – програмістом).

Представимо приклад машини Поста. Машина Поста працює в унарній системі числення (використовує лише однакові символи). Існує два числа в унарній системі числення $p = **, q = *$. Виконати віднімання $p - q$. Послідовність виконання цієї конкретної проблеми відповідно до інструкції табл. 6.3 показана на рис. 6.4.

Таблиця 6.3

Набір інструкцій для прикладу роботи машини Поста

Крок	Інструкція	Пояснення
1	←	крок ліворуч
2	? 1;3	якщо в комірці пусто, перейти до кроку 1, якщо ні – до кроку 3
3	X	видалити мітку
4	→	крок праворуч
5	? 4;6	якщо в комірці пусто, перейти до кроку 4, якщо ні – до кроку 6
6	X	видалити мітку
7	→	крок праворуч
8	? 4;6	якщо в комірці пусто, перейти до кроку 9, якщо ні – до кроку 1
9	!	кінець

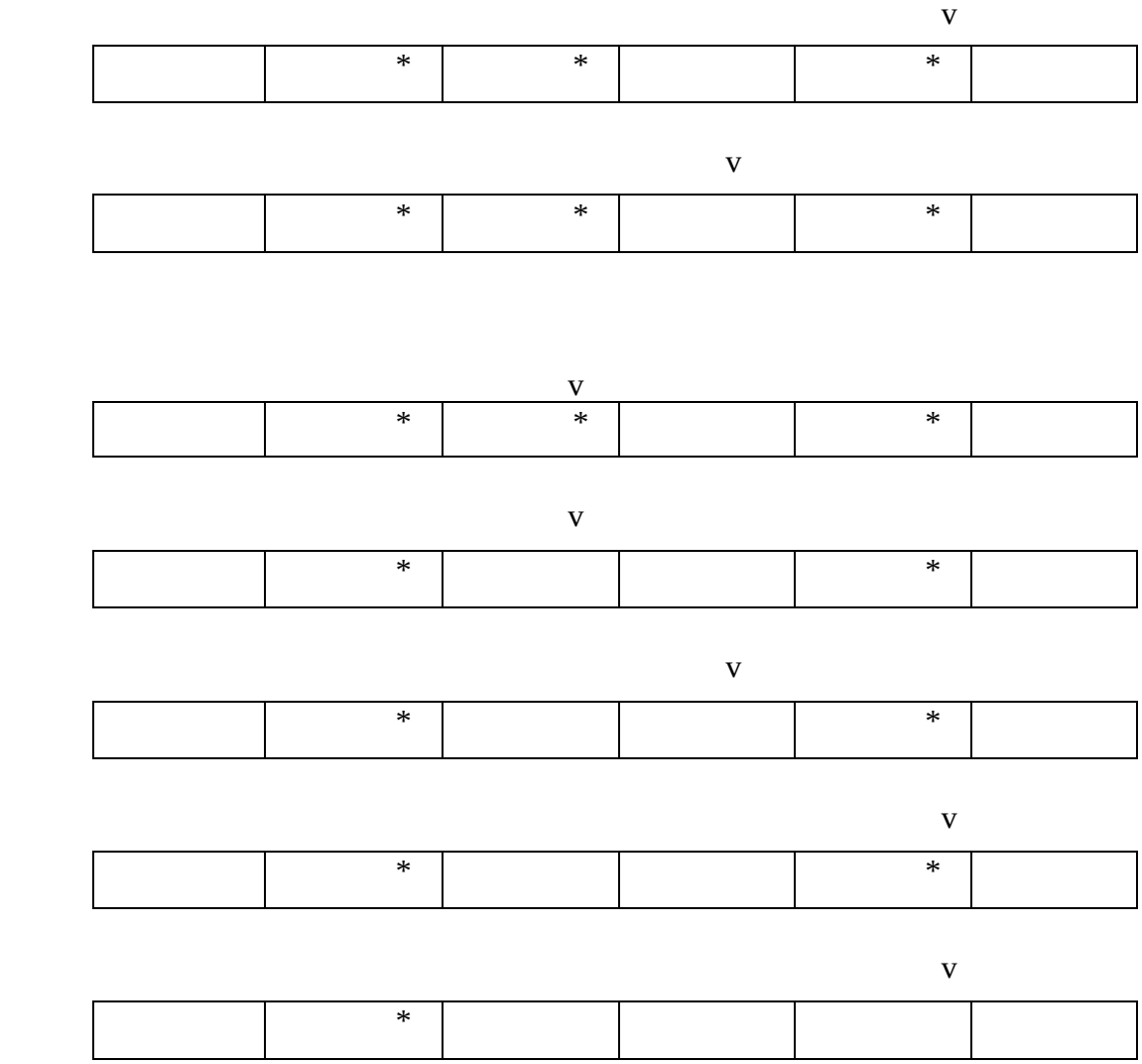


Рис. 6.4. Послідовність виконання набору інструкцій машиною Поста для вирішення конкретної проблеми віднімання

6.4. Еквівалентність МТ, нормального алгоритму Маркова та блок-схеми Поста

Теорема 6.1. За класом задач, які розв'язуються, МТ, нормальні алгоритми Маркова та блок-схема Поста еквівалентні.

Схема доведення

I. За БСП легко будується МТ. Дія кожної вершини БСП реалізується за допомогою відповідної МТ. А потім використовується техніка композиції.

II. Зрозуміло, що рух по слову зліва направо за допомогою БСП реалізується просто. Видаляємо символ з початку слова та пишемо в кінець. Щоб забезпечити рух по слову справа наліво, потрібно вміти розпізнавати останній символ даного слова.

Кінець схеми доведення.

Приклад 6.3. Продублювати останній символ слова в алфавіті $\{a, b\}$. Тобто якщо вхідне слово $w = \alpha_1\alpha_2 \dots \alpha_n$, то вихідне слово буде таким: $\alpha_1\alpha_2 \dots \alpha_n * \alpha_n$ (рис. 6.5).

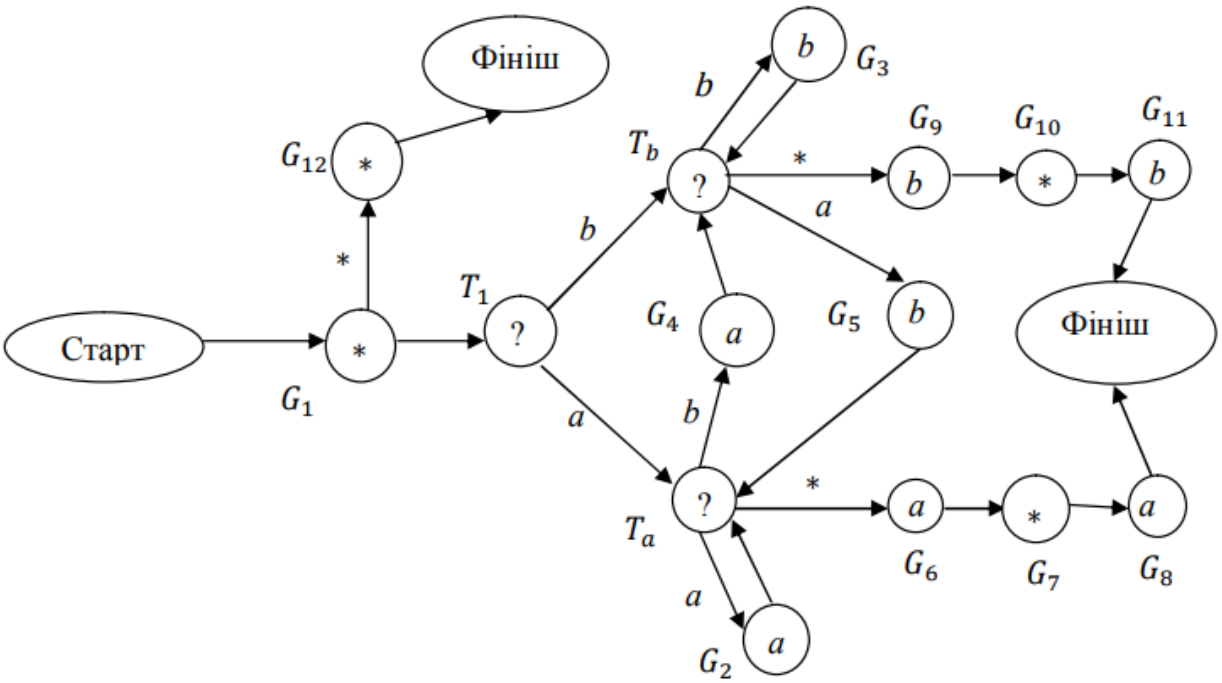


Рис. 6.5. БПС для прикладу 6.3

Пояснення. Пишемо в кінець слова зірочку (G_1) і перевіряємо перший символ слова (T_1). Якщо це зірочка, то вхідне слово було порожнім. Ставимо зірочку (G_{12}) і закінчуємо роботу.

У нетривіальних випадках переходимо в T_a, або T_b, які означатимуть, що остання прочитана літера була a чи b відповідно. У T_a і T_b видаляється наступна

літера, дописується попередня літера ($G_2, G_3,\ G_4,\ G_5$) та залежно від щойно прочитаної літери ми залишаємося в T_a (T_b) або переходимо в іншу тестову вершину T_b (T_a) відповідно. Тобто слово ми переписуємо із затримкою в один крок і запам'ятовуємо останній прочитаний символ. І нарешті, якщо в T_a (T_b) потрапили на зірочку, то останній символ буде a (b). Тому в першому випадку дописуємо $a * a$ ($G_6, G_7,\ G_8$) , а у другому – $b * b$ ($G_9, G_{10},\ G_{11}$) і закінчуємо роботу алгоритму.

Теорема 6.2. Нехай M – МТ із зовнішнім алфавітом A і внутрішнім алфавітом Q. Тоді існує нормальний алгоритм Маркова з алфавітом $A \cup Q$, еквівалентний даній МТ.

Доведення. Нехай дана машина Тюрінга M із зовнішнім алфавітом $\mathrm{A} = \{x_i\}$, де $i = 1, 2, \ldots, n$, і внутрішнім алфавітом $Q = \{q_i\},\ j = 0, 1, \ldots, n$. Двовимірну конфігурацію на стрічці можна записати як $x_1 x_2 \ldots q_j x_i x_{i+1} \ldots x_n$, де символ поточного стану q_j стоїть перед символом, який оглядає головка МТ в даний момент часу. Ми отримаємо слово в алфавіті $A \cup Q$. Тоді можна замінити кожну команду МТ на послідовність підстановок так:

Завдання 1. Побудувати БСП, яка:

1) анулює слово в алфавіті $\{a, b\}$;

2) видаляє всі літери a, подвоює всі літери b та потроює всі літери c у слові в алфавіті $\{a, b, c\}$;

3) обчислює $x + 1$ у двійковій системі;

4) копіює слово в алфавіті $\{a, b\}$;

5) переводить число з унарної у двійкову систему.

ПИТАННЯ ДО ТЕМИ 6

1. Які основні елементи машини Поста?

2. Що ми розуміємо під блок-схемою машини Поста?

3. Чи можливо в машині Поста використати іншу систему числення, крім унарної?

4. Які основні інструкції у машині Поста?

5. На чому ґрунтується еквівалентність МТ, нормального алгоритму Маркова та блок-схеми Поста?

СПИСОК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ

Основна література

1. Авдеюк П. І., Зеленський О. В. Елементи математичної логіки та теорії алгоритмів. Кам'янець-Подільський: КПНУ, 2019.180 с.

2. Зубенко В. В., Шкільня С. С. Основи математичної логіка: навчальний посібник. Київ: НУБіП України, 2020. 102 с.

3. Креневич А. П. Алгоритми і структури даних: підручник. Київ: ВПЦ Київський Університет, 2021. 200 с.

4. Шаховська Н. Б., Голощук Р. О. Алгоритми та структури даних. 2021. 216 с

5. Матвієнко М. П., Шаповалов С. П. Математична логіка та теорія алгоритмів: навчальний посібник. Київ: видавництво Ліра-К, 2021. 212 с.

Допоміжна література

1. Лиман Ф. М. Математична логіка і теорія алгоритмів: навчальний посібник. Суми: Видавництво «МакДен», 2014. 176 с.

2. Нікітченко М. С., Шкільняк С. С. Математична логіка та теорія алгоритмів. Київський національний університет імені Тараса Шевченка, 2008.

3. Комарницький М. Я., Андрійчук В. І., Мельник І. О. Елементи математичної логіки та теорії рекурсії. Львів: ЛНУ ім. Івана Франка, 2013.

4. Матвієнко М. П., Шаповалов С. П. Математична логіка та теорія алгоритмів: навчальний посібник. Київ: Видавництво Ліра-К, 2017. 212 с.

5. Шкільняк С. С. Теорія алгоритмів. Приклади й задачі. Київський національний університет імені Тараса Шевченка, 2012. 146 с.

Інформаційні ресурси в мережі Інтернет

1. CAS Data & Machine Learning. *Eidgenössische Technische Hochschule Zürich.* URL: surl.lu/vmhtjt

2. Knuth D. The Art of Computer Programming. Vol. 4, Fasc. 5: Mathematical Preliminaries Redux; Introduction to Backtracking. Boston: Pearson Education (US). 2020. URL: https://www.tug.org/TUGboat/tb41-1/tb127reviews-knuth-fascicle5.pdf

3. Klein Sh. T. Basic Concepts In Algorithms. Singapore: World Scientific Publishing Co Pte Ltd. 2021. URL: https://dokumen.pub/basic-concepts-in-algorithms-9811237581-9789811237584.html

ДЛЯ НОТАТОК

Навчальне видання

Данильчук Оксана Миколаївна

МАТЕМАТИЧНА ЛОГІКА ТА ТЕОРІЯ АЛГОРИТМІВ (частина 1)

Методичні рекомендації для самостійної роботи
для здобувачів вищої освіти ОС «Бакалавр»
спеціальності Е7 Математика

Редактор О. А. Солдатова
Технічний редактор Т. О. Важеніна-Гопрак

Підписано до друку 15.12.2025.
Формат 60×84/16. Папір офсетний.
Друк – цифровий. Умовн. друк. арк. 3,95.
Тираж 30. Зам. 91.

Донецький національний університет імені Василя Стуса
21021, м. Вінниця, 600-річчя, 21.
Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру
серія ДК № 5945 від 15.01.2018