Міністерство освіти і науки України

ДВНЗ "Прикарпатський національний університет
імені Василя Стефаника"

Володимир Гаврилків

Елементи теорії алгоритмів

Навчальний посібник

Хрестоматія

Івано-Франківськ

2020

УДК 510.5:004.423.24
ББК 22.123
Г 12

Гаврилків В.М. *Елементи теорії алгоритмів: навчальний посібник, хрестоматія*, Івано-Франківськ, 2020. — 81 с.

У хрестоматії у вигляді курсу з 8 параграфів розглянуто три основні алгоритмічні моделі: машина Тюрінґа, рекурсивні функції та нормальні алгоритми Маркова. Кожен параграф супроводжується питаннями та завданнями для самостійного розв'язування.

© Володимир Гаврилків, 2020

Зміст

Елементи теорії алгоритмів

§ 1. Машина Тюрінґа

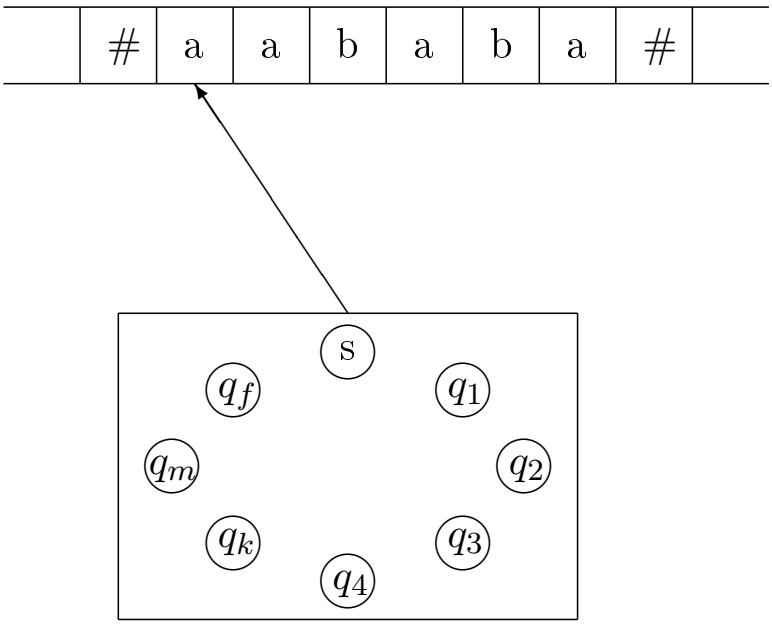
В попередньому розділі ми вивчили два типи обчислювальних моделей: скінченні автомати і автомати з магазинною пам'яттю, які розпізнають регулярні та контекстно-вільні мови відповідно. Зрозуміло, що дані моделі мають обмежені обчислювальні можливості в порівнянні з реальними персональними комп'ютерами. Наприклад, формальна мова $\{a^n b^n c^n \mid n \geq 0\}$ не розпізнається жодним із згаданих вище скінченних автоматів, але може бути легко обчислена з допомогою короткої програми на ПК.

В цьому розділі ми опишемо нову обчислювальну алгоритмічну модель, яка називається машиною Тюрінґа (МТ). Основні відмінності між машиною Тюрінґа та скінченним автоматом полягають у тому, що стрічка МТ є нескінченною і МТ може пересуватись по стрічці (чи зміщувати стрічку) в будь-якому напрямку. Це надає машині потенційно необмежену пам'ять, яку можна використовувати під час обчислень.

МТ складається з керуючого пристрою, що містить головку і зовнішньої пам'яті (стрічки). Головка машини Тюрінґа може рухатися вздовж стрічки праворуч або ліворуч, читати та записувати символи, і, отже, поводиться аналогічно, як і справжній комп'ютер. Машина Тюрінґа має ті ж самі обчислювальні можливості, що і багато інших комп'ютерних моделей.

Розглянемо детальніше основні складові машини Тюрінґа.

Стрічка використовується для збереження інформації. Вона поділена на комірки і є нескінченною в обидва боки. Комірки не нумеруються. Кожна комірка містить єдиний символ із скінченного алфавіту A. Букви алфавіту $A = X \cup Y \cup D$ діляться на три групи. Букви алфавіту X називаються *вхідними буквами або символами* (з них утворюються слова, які записані на стрічці перед початком роботи МТ). Через Y позначається множина букв, які називаються *вихідними буквами або символами* (з них утворюються слова, які записані на стрічці після завершення роботи МТ). D – множина *допоміжних символів*, до яких відноситься порожній символ #. Множини X та Y можуть перетинатися.



Головка машини Тюрінґа може рухатися вздовж стрічки комірка за коміркою праворуч або ліворуч. Крім того, вона може читати букву з комірки, яку аналізує, а також видаляти букву і записувати в комірку нову букву. В кожен момент часу головка розміщується під однією з комірок стрічки і аналізує її вміст. Дана клітинка називається *активною*, а символ, який в ній знаходиться – *активним символом*. Вміст клітинок, які не є активними, головка МТ не аналізує і не знає, які символи в них розміщені.

Головка машини Тюрінґа контролюється керуючим пристроєм, який складається зі скінченної кількості станів. Серед них є два виділені стани: початковий і кінцевий. Початковий стан зазвичай позначається через s або q_0, а кінцевий – через q_f. Множину станів позначимо через Q. В кожен момент часу керуючий пристрій перебуває в одному зі станів, які позначатимемо через $q_1, q_2, \ldots, q_n$. Перебуваючи в певному стані, МТ виконує деяку дію (наприклад, рухається праворуч, міняючи при цьому вміст клітинок на порожній символ #). Після того, як головка прочитає символ зі стрічки, керуючий пристрій визначає, яку дію виконувати (який символ записувати і куди рухатися: праворуч чи ліворуч) і переходить в інший стан. МТ може виконувати тільки три елементарні дії: записувати в активну комірку новий символ, пересуватися на одну клітинку ліворуч або праворуч, переходити в новий стан. Жодних інших дій МТ виконувати не може! Тому більш складні дії повинні

виражатися через елементарні. Ці дії описуються *функцією переходів (програмою)*:

$$\delta : Q \times A \to Q \times A \times \{L, R\}.$$

Якщо для функції δ виконується $\delta(q, a) = (p, b, R)$ ($\delta(q, a) = (p, b, L)$), то це означає, що перебуваючи в стані q, головка аналізує букву a (тобто буква a є активною), міняє її на b і рухається праворуч (відповідно ліворуч), керуючий пристрій при цьому переходить зі стану q у стан p. Описаний процес називається *тактом роботи машини Тюрінґа*. Таким чином, МТ працює тактами, що визначаються функцією δ.

Програма МТ записується у вигляді наступної таблиці:

δ	a_1	a_2	$\ldots$	a_i	$\ldots$	a_n	#
s							
q_1							
$\ldots$							
q_j				$q', a', [L, R]$			
$\ldots$							
q_m							

Зліва записуються всі стани, крім кінцевого q_f, з множини станів

$$Q = \{s, q_1, \ldots, q_j, \ldots, q_m, q_f\},$$

в яких може знаходитися керуючий пристрій МТ. Зверху – всі букви з алфавіту A (серед яких обов'язково є #), які головка МТ може аналізувати на стрічці. На перетинах рядків і стовпців таблиці вказуються ті такти, котрі повинен виконувати керуючий пристрій, знаходячись у відповідному стані й аналізуючи відповідну букву на стрічці.

Підсумовуючи все вищенаписане, машина Тюрінґа може бути визначена як п'ятірка $MT = (Q, A, \delta, s, q_f)$, де Q – скінченна множина станів, A – скінченний алфавіт, δ – функція переходів, $s \in Q$ – початковий стан, $q_f \in Q$ – кінцевий стан.

Вхідне слово – це скінченна послідовність букв з алфавіту $X \subset A$, записаних зліва направо по одній в кожній клітинці стрічки. Всередині вхідного слова порожніх клітинок немає. Всі клітинки ліворуч і праворуч від клітинок, заповнених буквами вхідного слова ω, вважаються порожніми, і в них знаходиться символ #. Якщо

вхідне слово порожнє, то на стрічці всі клітинки заповнені порожнім символом #.

Для того щоб пояснити, як машина Тюрінґа працює з вхідним словом ω, визначимо поняття конфігурації МТ. Припустимо, що на деякому етапі роботи МТ перебуває в стані q, її стрічка містить символи $\ldots\#\#\#x_1x_2\ldots x_m\#\#\#\ldots$, де $x_i \in A$, $i = 1,\ldots,m$, і головка аналізує букву x_k. В випадку, коли $k \le m$, будемо писати:

$$(q, x_1x_2\ldots x_{k-1}\underline{x_k}x_{k+1}\ldots x_m),$$

щоб позначити цю конфігурацію. Якщо ж $k > m$, то позначимо цю конфігурацію наступним чином:

$$(q, x_1x_2\ldots x_m\underbrace{\#\#\ldots\underline{\#}}_{k-m}).$$

Таким чином, конфігурація – це пара $(q,\omega) \in Q \times A^*$. Конфігурація $(q, x_1x_2\ldots x_{k-1}\underline{x_k}x_{k+1}\ldots x_m)$ позначає той факт, що машина перебуває в стані q, її стрічка містить слово $x_1x_2\ldots x_{k-1}x_kx_{k+1}\ldots x_m$ і головка аналізує букву x_k.

Вживаючи ці позначення, результат кожного руху МТ можна описати наступним чином:

- якщо $\delta(q,x_k) = (p,a,R)$, то після одного такту конфігурація $(q, x_1x_2\ldots\underline{x_k}x_{k+1}\ldots x_m)$ змінюється на конфігурацію $(p, x_1x_2\ldots a\underline{x_{k+1}}\ldots x_m)$;
- якщо $\delta(q,x_m) = (p,a,R)$, то після одного такту конфігурація $(q, x_1x_2\ldots\underline{x_m})$ змінюється на $(p, x_1x_2\ldots a\underline{\#})$;
- якщо $\delta(q,x_k) = (p,a,L)$, то після одного такту конфігурація $(q, x_1x_2\ldots x_{k-1}\underline{x_k}\ldots x_m)$ змінюється на конфігурацію $(p, x_1x_2\ldots\underline{x_{k-1}}a\ldots x_m)$;
- якщо $\delta(q,x_1) = (p,a,L)$, то після одного такту конфігурація $(q,\underline{x_1}x_2\ldots x_m)$ змінюється на $(p,\underline{\#}ax_2\ldots x_m)$;
- в конфігурації $(q_f, x_1x_2\ldots\underline{x_k}x_{k+1}\ldots x_m)$ МТ зупиняє свою роботу і на виході дає слово $x_1x_2\ldots x_m$.

Якщо конфігурація $\alpha_1 = (q,\omega_1\underline{a}\omega_2)$ змінюється на конфігурацію $\alpha_2 = (q,\upsilon_1\underline{b}\upsilon_2)$, то писатимемо $\alpha_1 \to \alpha_2$. Якщо ми матимемо послідовність конфігурацій $\alpha_1,\alpha_2,\ldots\alpha_n$, таких, що $\alpha_i \to \alpha_{i+1}$, $i = 1,\ldots n-1$, то писатимемо $\alpha_1 \Rightarrow \alpha_n$.

На початку роботи машина Тюрінґа перебуває в початковому стані s і головка аналізує першу зліва непорожню клітинку. Далі

починається виконання програми роботи МТ. У таблиці обирається клітинка на перетині першого рядка (бо керуючий пристрій знаходиться в початковому стані s) та того стовпчика, який відповідає першій букві вхідного слова, і виконується такт, вказаний в цій клітинці. Таким чином, МТ перейде до нової конфігурації. Потім виконуються такі ж дії, але вже в новій конфігурації: в таблиці знаходимо клітинку, що відповідає стану керуючого пристрою і букві цієї конфігурації, і виконується такт з цієї клітинки і т.д і т.п. Перейшовши в кінцевий стан q_f, МТ зупиняє свою роботу, і на виході одержуємо нове слово, яке записане в комірках стрічки МТ після завершення її роботи. Дане слово називатимемо *вихідним словом*. Кажемо, що *слово $\omega = x_1x_2\dots x_m$ розпізнається машиною Тюрінґа*, якщо МТ, аналізуючи ω, зупиняє свою роботу в стані q_f, тобто $(s, \underline{x_1}x_2\dots x_m) \Rightarrow (q_f, y_1y_2\dots \underline{y_k}\dots y_n)$. В протилежному випадку слово ω не розпізнається МТ.

В момент зупинки всередині вихідного слова не може міститися порожніх клітинок, хоча в процесі виконання програми такі клітинки всередині проміжного слова можуть з'являтися.

Може трапитися так, що в процесі обробки вхідного слова ω, МТ ніколи не перейде в кінцевий стан q_f. У цьому випадку кажемо, що МТ *зациклюється на вхідному слові ω* і слово не розпізнається МТ. Якщо функція δ не визначена на наборі (q, a) і МТ, перейшовши в стан q, аналізує букву a слова ω, то у цьому випадку також будемо вважати, що слово ω не розпізнається МТ.

1.1. Приклад. Нехай машина Тюрінґа задана як п'ятірка $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_1, q_2, q_f\}$, $A = \{a, b, 0, 1, \#\}$, $X = \{a, b\}$, $Y = \{0, 1\}$, $D = \{\#\}$, і програма роботи δ задана у вигляді таблиці:

δ	a	b	$\#$
s	$s, 0, R$	$q_1, 1, R$	$q_2, \#, L$
q_1	$q_2, 0, R$	$q_1, 1, R$	$q_f, 0, L$
q_2	$q_2, 0, R$	$q_2, 1, R$	$q_2, 0, R$

Початкова конфігурація МТ на вхідному слові $aababa$ матиме вигляд $(s, \underline{a}ababa)$. Починаючи з цієї конфігурації, робота МТ на слові $aababa$ може бути описана наступним чином:

$(s, \underline{a}ababa) \to (s, 0\underline{a}baba) \to (s, 00\underline{b}aba) \to (q_1, 001\underline{a}ba) \to$
$(q_2, 0010\underline{b}a) \to (q_2, 00101\underline{a}) \to (q_2, 001010\underline{\#}) \to (q_2, 0010100\underline{\#}) \Rightarrow$

$(q_2, 001010\ldots0\underline{\#}) \to \ldots$

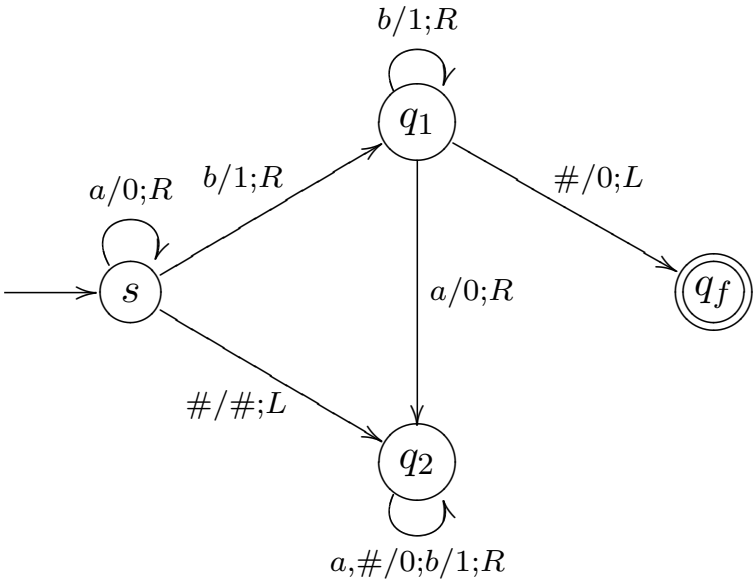
Таким чином, МТ ніколи не вийде зі стану q_2 і її головка нескінченно рухатиметься праворуч, міняючи при цьому порожні символи на 0. Іншими словами, в стані q_2 МТ зациклюється на вхідному слові $aababa$ і дане слово не розпізнається МТ.

На вхідному слові $aaaabb$ МТ працює іншим чином:

$(s, \underline{a}aaabb) \to (s, 0\underline{a}aabb) \to (s, 00\underline{a}abb) \to (s, 000\underline{a}bb) \to (s, 0000\underline{b}b)$ $\to (q_1, 00001\underline{b}) \to (q_1, 000011\underline{\#}) \to (q_f, 0000\underline{1}10)$.

Оскільки МТ перейшла в стан q_f, то вона зупиняє свою роботу. Вхідне слово $aaaabb$ розпізнається МТ. Вихідним словом є 0000110.

Для зручності МТ зображують поміченим орієнтованим графом $G(MT)$. Вершинами даного графа є стани автомата. Кожна стрілка графа має позначку виду „$a/b; R$" або „$a/b; L$". Якщо $\delta(q, a)$ $=(p, b, R)$ $(\delta(q, a) = (p, b, L))$, то граф містить стрілку з позначкою „$a/b; R$" (відповідно „$a/b; L$"), початком якої є вершина q, а кінцем – вершина p. При цьому, якщо стрілка містить дві позначки виду „$a/b; R$" і „$c/b; R$" (відповідно „$a/b; L$" і „$c/b; L$"), то для компактності запису пишемо позначку „$a, c/b; R$" (відповідно „$a, c/b; L$"). Крім того, щоб підкреслити початковий стан s, малюється стрілка без початкової вершини, кінцевою вершиною якої є стан s. Кінцевий стан позначається двома концентричними колами. Наприклад, граф МТ з прикладу 1.1 має вигляд:



Мовою $L_1(MT)$, що розпізнається машиною Тюрінґа, називається множина всіх слів, які розпізнаються МТ, тобто

$$L_1(MT) = \{\omega \in A^* : \omega \text{ розпізнається МТ}\}.$$

Легко бачити, що в прикладі 1.1 мова, що розпізнається машиною Тюрінґа $L_1(MT) = \{a^m b^n : m \geq 0, n \geq 1\}$.

Кажемо, що формальна мова L розпізнається машиною Тюрінґа, якщо існує така МТ, що $L = L(MT)$.

Мовою $L_2(MT)$, що породжується машиною Тюрінґа, називається множина всіх вихідних слів, які утворюються з вхідних слів мови $L_1(MT)$, що розпізнається машиною Тюрінґа. Наприклад, в прикладі 1.1 мова, що породжується машиною Тюрінґа $L_2(MT) = \{0^m 1^n 0 : m \geq 0, n \geq 1\}$.

Рекомендована література : [**1**, с. 24–33], [**8**, с. 178–183], [**11**, с. 317–322], [**17**, с. 312–318], [**20**, с. 3–6], [**24**, с. 76–82].

Питання та вправи до параграфа 1.

1.1. Дайте означення машини Тюрінґа.

1.2. Що ви розумієте під тактом роботи МТ?

1.3. Опишіть процес обробки МТ вхідного слова.

1.4. Як будується граф МТ?

1.5. Нехай МТ задана як п'ятірка: $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_f\}$, $A = \{0, 1, \#\}$, а програма роботи δ задана у вигляді таблиці:

δ	0	1	#
s	$s, 1, R$	$q_f, 0, R$	$s, \#, R$

Виписуючи на кожному такті роботи МТ одержану конфігурацію, визначте, в яке слово перетворює МТ кожне з наступних вхідних слів:

а) 100;
б) 001;
в) 0010;
г) 0000.

1.6. Зобразіть граф МТ з попереднього прикладу та визначте мову, що розпізнається даною МТ.

1.7. В чому полягає робота машини Тюрінґа $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_f\}$, $A = \{0, 1, \#\}$, а програма роботи δ задана у вигляді таблиці:

δ	0	1	#
s	$s, 0, R$	$q_f, 1, R$	$q_f, 1, L$

?

1.8. Нехай МТ задана як п'ятірка: $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q, q_f\}$, $A = \{0, 1, \#\}$, а програма роботи δ задана у вигляді таблиці:

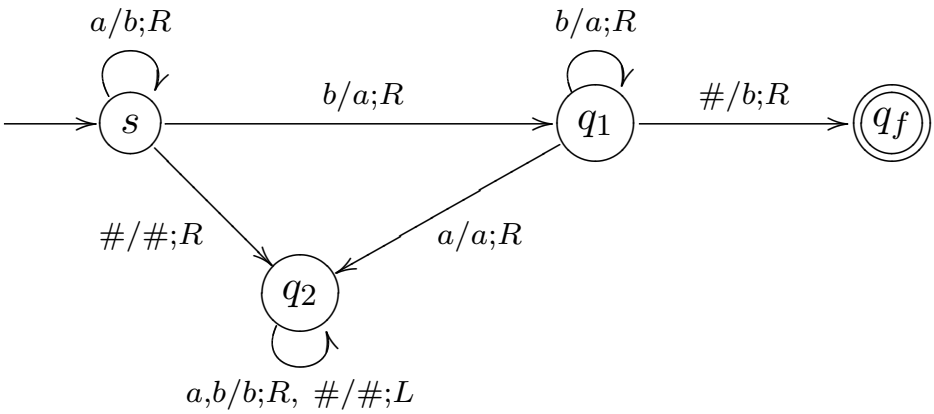
δ	0	1	#
s	$s, 0, R$	$s, 1, R$	$q, \#, L$
q	$q_f, 1, L$	$q, 0, L$	$q_f, 1, L$

Проаналізуйте роботу МТ на наступних вхідних словах:

а) 00;
б) 100;
в) 10011;
г) 111.

Вважаючи вхідні слова записами натуральних чисел в двійковій системі числення, вкажіть загальну закономірність роботи машини на усіх вхідних словах.

1.9. Нехай МТ зображена графом:



Задайте її як п'ятірку $MT = (Q, A, \delta, s, q_f)$ та визначте мову, що розпізнається даною МТ.

1.10. Знайти мову, що породжується машиною Тюрінґа з попереднього прикладу.

1.11. Побудувати граф та проаналізувати роботу машини Тюрінґа $MT = (Q, A, \delta, s, q_f)$ на словах $1^n = \underbrace{1 \dots 1}_{n}$, де $Q = \{s, q_1, q_2, q_f\}$, $A = \{0, 1, \#\}$, а програма роботи δ задана у вигляді таблиці:

δ	0	1	#
s	$q_2, 1, R$	$q_1, \#, R$	$q_2, 1, R$
q_1	$q_2, 1, R$	$q_f, \#, R$	$q_2, \#, R$
q_2	$q_2, 1, R$	$q_2, 1, R$	$q_2, 1, R$

Як працює машина на слові 1000000? Яку мову розпізнає дана МТ?

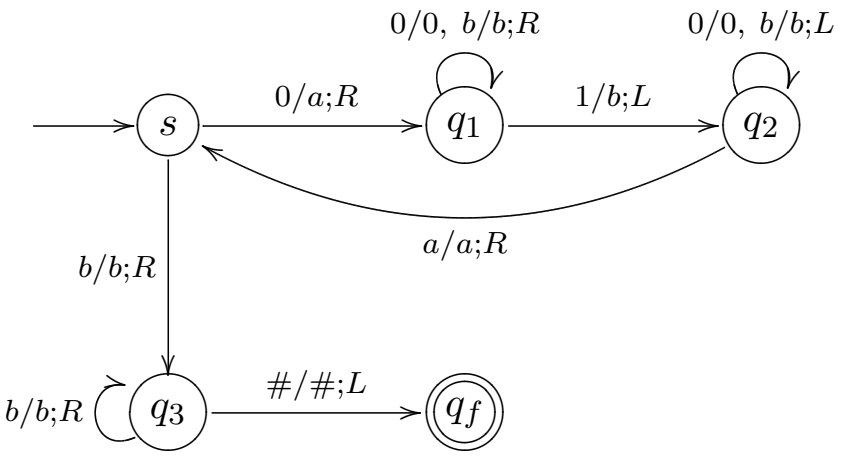
1.12. В чому полягає робота машини Тюрінґа $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_1, q_2, q_3, q_f\}$, $A = \{a, b, \#\}$, а програма роботи δ задана у вигляді таблиці:

δ	a	b	#
s	q_1, a, R	q_2, b, R	
q_1	q_1, a, R	q_2, a, R	$q_3, \#, L$
q_2	q_1, a, R	q_2, b, R	$q_3, \#, L$
q_3	q_f, a, R	$q_f, \#, L$	

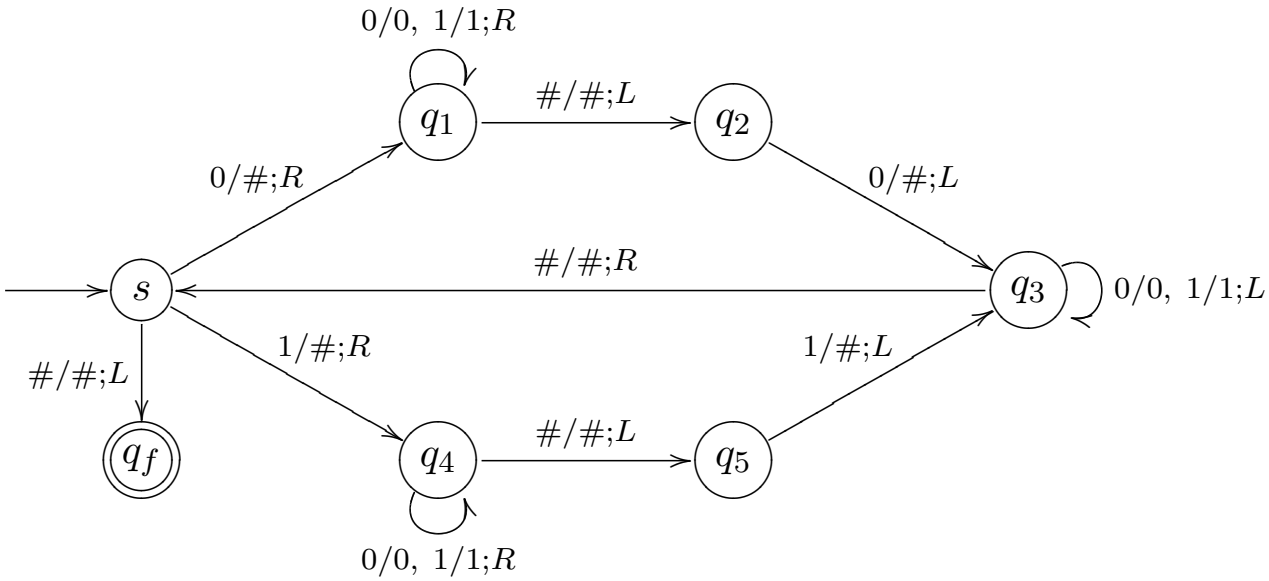
?

1.13. Знайти мову, яку розпізнає МТ, задана графом:

а)



б)

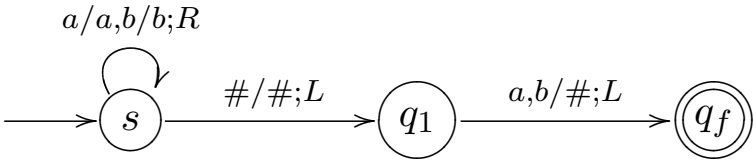


§ 2. Алгоритми синтезу МТ

У цьому параграфі на прикладах пояснюються основні прийоми складання алгоритмів для МТ.

2.1. Приклад. Побудувати машину Тюрінґа, яка розпізнає всі слова в алфавіті $X = \{a, b\}$ і видаляє із вхідного слова його останню букву.

Для розв'язання цієї задачі потрібно перемістити головку МТ під останню букву і видалити її. Нехай s – стан, в якому керуючий пристрій МТ переміщує головку комірка за коміркою праворуч, не змінюючи букви і залишаючись в стані s. Тут виникає невелика проблема: МТ аналізує тільки активну комірку і не може визначити, чи наступна комірка є порожньою. Тому головка МТ спершу має перемістится під першу порожню комірку (в ній записаний символ #), яка розміщена після вхідного слова, а потім керуючий пристрій, перебуваючи в стані s і аналізуючи символ #, має перейти в новий стан q_1, в якому видалить останню букву і зупинить свою роботу (перейде в кінцевий стан q_f). Таким чином, граф буде мати наступний вигляд:



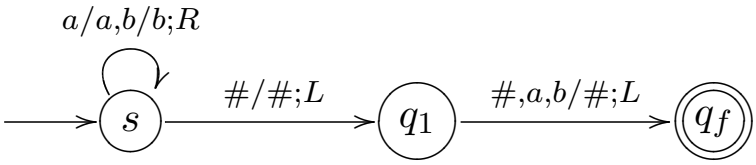
Проаналізуємо роботу даної МТ на вхідних словах abb, a і порожньому слові e.

$(s, \underline{a}bb) \to (s, a\underline{b}b) \to (s, abb\underline{\#}) \to (q_1, abb\underline{\#}) \to (q_f, a\underline{b})$.

$(s, \underline{a}) \to (s, a\underline{\#}) \to (q_1, \underline{a}) \to (q_f, \underline{\#})$.

$(s, \underline{\#}) \to (q_1, \underline{\#})$

Ми бачимо, що на порожньому слові робота МТ невизначена і вона не розпізнає порожнього слова. В зв'язку з цим домовимося, що порожнє слово МТ не змінює. Остаточно, граф матиме вигляд:

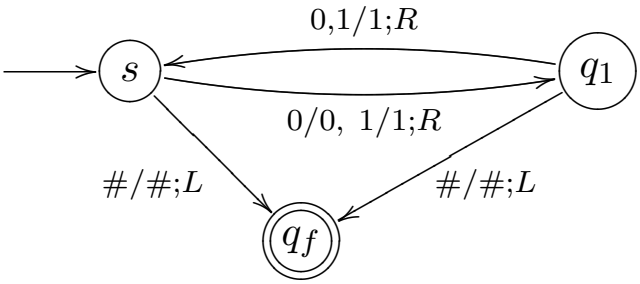


Таким чином, МТ задається як п'ятірка $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_1, q_f\}$, $A = \{a, b, \#\}$, $X = Y = \{a, b\}$, $D = \{\#\}$ і програма роботи δ задана у вигляді таблиці:

δ	a	b	$\#$
s	s, a, R	s, b, R	$q_1, \#, L$
q_1	$q_f, \#, L$	$q_f, \#, L$	$q_f, \#, L$

2.2. Приклад. Побудувати машину Тюрінґа, яка розпізнає всі слова в алфавіті $X = \{0, 1\}$ і замінює на 1 кожну другу букву вхідного слова.

Для розв'язання цієї задачі потрібно навчитися визначати, чи знаходиться головка МТ під буквою, яка міститься на парній позиції вхідного слова. Для цього розглянемо два стани s і q_1, в які керуючий пристрій буде почергово переходити при русі головки МТ праворуч і аналізі слова буква за буквою, починаючи з першої букви. Таким чином, якщо МТ перебуває в стані s, то вона аналізує букву, що знаходиться на непарній позиції у вхідному слові; якщо ж МТ перебуває в стані q_1, то вона аналізує букву, що знаходиться на парній позиції. Домовимося також, що МТ не змінює порожнього слова. Отже, необхідно визначити роботу МТ так, щоб у стані s головка МТ не змінювала букви, а в стані q_1 замінювала 0 і 1 на 1. Підсумовуючи все вищесказане, побудуємо граф МТ наступним чином:



Шукана МТ задається як п'ятірка $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_1, q_f\}$, $A = \{0, 1, \#\}$, $X = Y = \{0, 1\}$, $D = \{\#\}$ і програма роботи δ задана у вигляді таблиці:

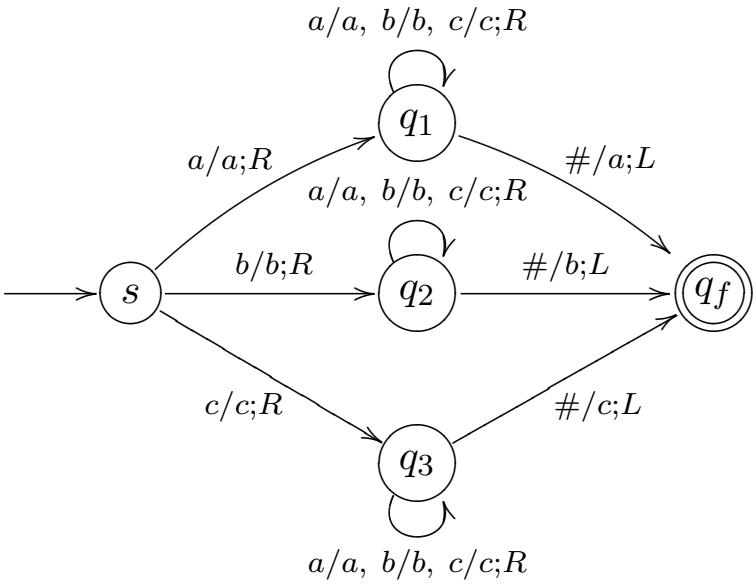
δ	0	1	#
s	$q_1, 0, R$	$q_1, 1, R$	$q_f, \#, L$
q_1	$s, 1, R$	$s, 1, R$	$q_f, \#, L$

2.3. Приклад. Побудувати машину Тюрінґа, яка розпізнає всі непорожні слова в алфавіті $X = \{a, b, c\}$ і дописує вкінці вхідного слова його першу букву.

Для розв'язання даної задачі необхідно виконати наступні дії:

- запам'ятати першу букву;
- перемістити головку МТ праворуч під першу порожню комірку і записати в неї першу букву.

Як переміщуватися праворуч ми вже знаємо з попередніх прикладів. Але як запам'ятати першу букву? Адже в МТ немає іншої пам'яті, крім стрічки, а запам'ятовувати букву в якій-небудь комірці немає сенсу: як тільки головка МТ переміститься ліворуч чи праворуч від цієї клітки, МТ відразу ж забуде дану букву. Як же вийти з цієї ситуації? Стандартним вирішенням цієї проблеми є наступне – для кожної букви треба використовувати окрему множину станів. Якщо першою буквою є a, то потрібно перейти в стан q_1, в якому головка МТ переміщується праворуч і записує вкінці слова букву a. Якщо ж першою буквою є буква b, то треба перейти в стан q_2, в якому виконуються ті ж самі дії, тільки вкінці слова дописується буква b. У випадку, якщо першою була буква c, переходимо в стан q_3, в якому дописуємо вкінці вхідного слова букву c. Цей прийом будемо використовувати і надалі для запам'ятовування різних букв вхідного слова. Підсумовуючи все сказане, граф МТ можна зобразити наступним чином:



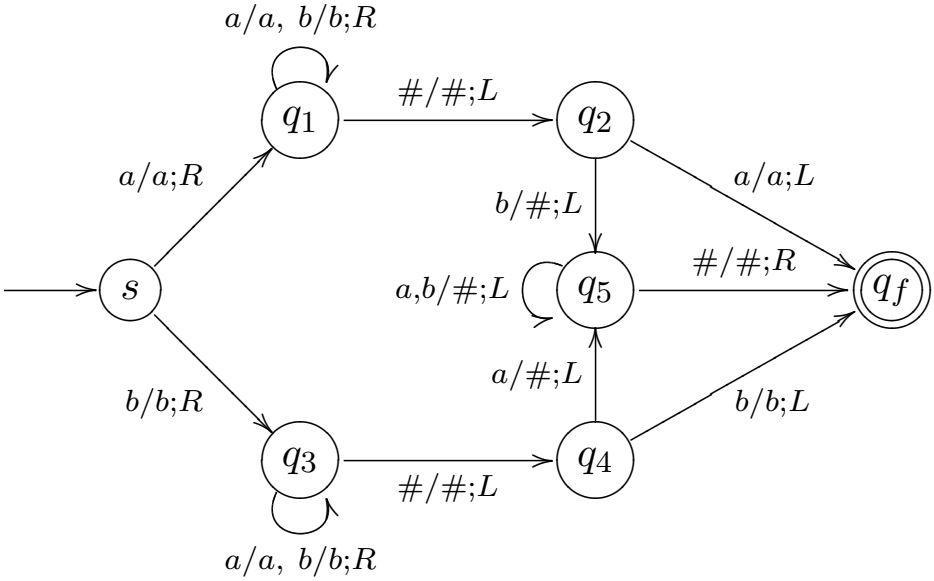
Програма роботи δ машини Тюрінґа $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_1, q_2, q_3, q_f\}$, $A = \{a, b, c, \#\}$, $X = Y = \{a, b, c\}$, $D = \{\#\}$, задається у вигляді таблиці:

δ	a	b	c	$\#$
s	q_1, a, R	q_2, b, R	q_3, c, R	$-$
q_1	q_1, a, R	q_1, b, R	q_1, c, R	q_f, a, L
q_2	q_2, a, R	q_2, b, R	q_2, c, R	q_f, b, L
q_3	q_3, a, R	q_3, b, R	q_3, c, R	q_f, c, L

2.4. Приклад. Побудувати машину Тюрінґа в алфавіті $X = \{a, b\}$, яка працює наступним чином: якщо перша і остання букви непорожнього вхідного слова співпадають, то слово залишається незмінним, в протилежному випадку замінює його на порожнє слово.

Для розв'язання цієї задачі потрібно запам'ятати першу букву вхідного слова, перемістити головку МТ під останню букву і порівняти її з першою. Якщо ці букви однакові, то МТ має перейти в кінцевий стан і зупинити свою роботу, в протилежному випадку – видалити всі букви в слові. З попереднього прикладу ми вже знаємо, що для кожної букви потрібно створити свою множину станів для запам'ятовування даної букви і порівняння її з останньою буквою. Нехай такими станами для букви a будуть q_1, в якому МТ запам'ятовує a, і q_2, в якому перевіряється чи останньою є буква a. Аналогічні стани для букви b позначимо через q_3 і q_4. Якщо ж в стані q_2 головка МТ аналізує букву b або в стані q_4 головка МТ аналізує букву a, то ці букви видаляються (замінюються порожнім

символом #) і МТ переходить в новий стан q_5, в якому головка МТ переміщується ліворуч на початок слова, видаляючи при цьому всі букви. Таким чином, граф МТ буде мати наступний вигляд:



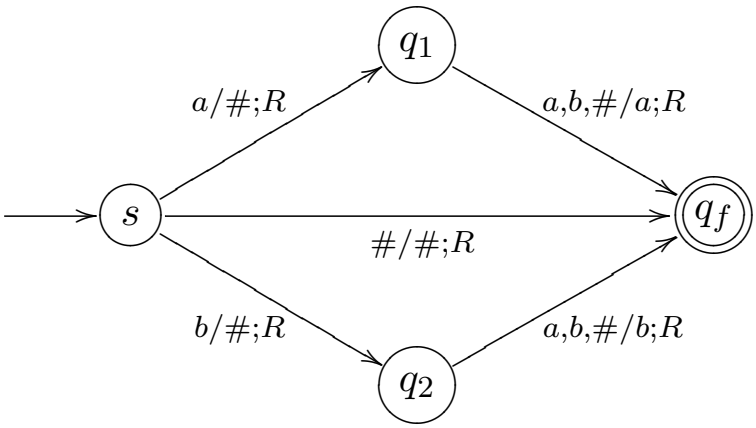
Отже, МТ задається як п'ятірка $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_1, q_f\}$, $A = \{a, b, \#\}$, $X = Y = \{a, b\}$, $D = \{\#\}$ і програма роботи δ задана у вигляді таблиці:

δ	a	b	$\#$
s	q_1, a, R	q_3, b, R	$-$
q_1	q_1, a, R	q_1, b, R	$q_2, \#, L$
q_2	q_f, a, L	$q_5, \#, L$	$-$
q_3	q_3, a, R	q_3, b, R	$q_4, \#, L$
q_4	$q_5, \#, L$	q_f, b, L	$-$
q_5	$q_5, \#, L$	$q_5, \#, L$	$q_f, \#, R$

2.5. Приклад. Побудувати машину Тюрінґа, яка розпізнає всі слова в алфавіті $X = \{a, b\}$ і видаляє із вхідного слова його другу букву.

На перший погляд здається, що таку МТ побудувати надзвичайно просто: достатньо змістити головку МТ на одну букву праворуч, а потім видалити другу букву вхідного слова. Але вкінці роботи МТ вихідне слово не може містити пропусків, тому після видалення другого символу потрібно стиснути слово, тобто перемістити першу букву в комірку праворуч. Проте, переміщуючись праворуч до другої букви, а потім повертаючись назад до першої

букви, ми виконуємо зайву роботу: яка різниця чи переносити першу букву в порожню комірку чи в комірку з якоюсь буквою? Тому запам'ятовуємо першу букву, видаляємо і записуємо її замість другої букви. Крім того, вважатимемо, що МТ не змінює порожнього слова. Отже, граф МТ матиме наступний вигляд:



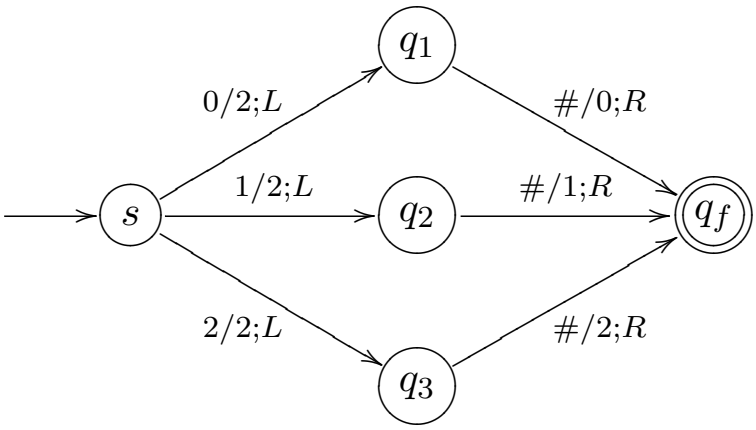
Проаналізуємо роботу МТ на вхідному слові, яке містить тільки одну букву. Аналізуючи дану букву, головка МТ видалить її, переміститься на одну комірку праворуч і запише в порожню комірку дану букву. Таким чином, слово із однієї букви просто зміститься на комірку праворуч. Це допустимо, бо комірки стрічки не нумеруються, тому розташування слова на стрічці ніяк не фіксується і переміщення слова ліворуч чи праворуч неможливо помітити. В зв'язку з цим не вимагається, щоб вихідне слово обов'язково знаходилось на тому ж місці, де було вхідне слово: результат може зміститися ліворуч чи праворуч від початкового місця.

Шукана МТ задається як п'ятірка $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_1, q_2, q_f\}$, $A = \{a, b, \#\}$, $X = Y = \{a, b\}$, $D = \{\#\}$ і програма роботи δ задана у вигляді таблиці:

δ	a	b	$\#$
s	$q_1, \#, R$	$q_2, \#, R$	$q_f, \#, R$
q_1	q_f, a, R	q_f, a, R	q_f, a, R
q_2	q_f, b, R	q_f, b, R	q_f, b, R

2.6. Приклад. Побудувати машину Тюрінґа, яка за першою буквою непорожнього слова в алфавіті $X = \{0, 1, 2\}$ вставляє букву 2.

Шукана МТ має спочатку вивільнити місце для букви 2. Для цього потрібно „розтиснути" слово. Ми можемо запам'ятати першу букву, видалити, записати її в комірці ліворуч, а потім у порожню комірку записати букву 2. Але, аналогічно як у попередньому прикладі, можна зробити це простіше: запам'ятати першу букву, записати на її місце букву 2, а потім перемістится ліворуч і записати в порожній комірці першу букву вхідного слова. Таким чином, отримуємо наступний граф МТ:



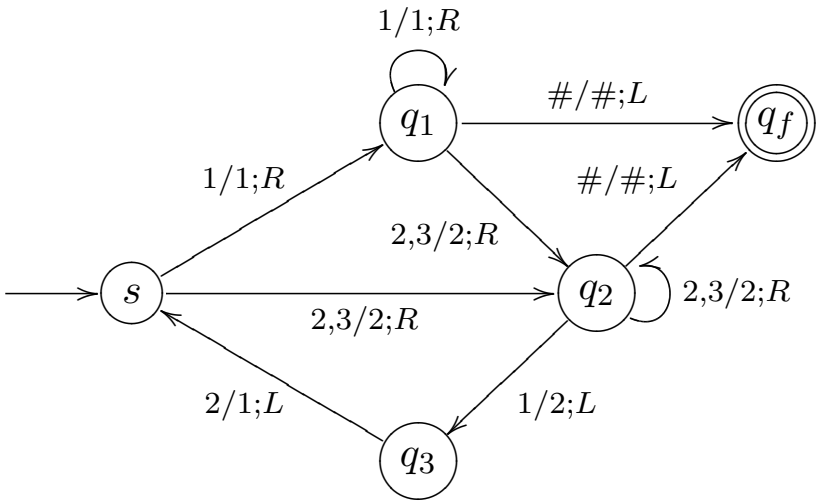
Отже, МТ задається як п'ятірка $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_1, q_2, q_3, q_f\}$, $A = \{0, 1, 2, \#\}$, $X = Y = \{0, 1, 2\}$, $D = \{\#\}$ і програма роботи δ задана у вигляді таблиці:

δ	0	1	2	#
s	$q_1, 2, L$	$q_2, 2, L$	$q_3, 2, L$	–
q_1	–	–	–	$q_f, 0, R$
q_2	–	–	–	$q_f, 1, R$
q_3	–	–	–	$q_f, 2, R$

2.7. Приклад. Побудувати машину Тюрінґа в алфавіті $X = \{1, 2, 3\}$, яка у непорожньому вхідному слові замінює всі букви 3 на букви 2 і впорядковує послідовність букв 1 і 2 так, щоб спочатку були записані букви 1, а потім букви 2.

Шукана МТ має впорядкувати вхідне слово, замінивши при цьому всі букви 3 на букви 2. Принцип її роботи може бути описаний наступним чином. Як завжди, на початку роботи МТ знаходиться в початковому стані s і її головка аналізує першу букву вхідного слова. В стані q_1 головка МТ аналізує всі букви 1 слова, рухаючись праворуч доти, доки не знайде першу букву 2 або 3. Далі керуючий пристрій переходить в стан q_2, в якому головка МТ,

рухаючись праворуч, аналізує букви 2 і 3, міняючи при цьому 3 на 2 доти, доки не побачить наступну 1. Після цього вона міняє місцями в стані q_3 1 з 2, що знаходиться ліворуч, і знову повертається в початковий стан, повторюючи далі аналогічні кроки аж доки вхідне слово не буде впорядковане. Граф МТ має вигляд:



Проаналізуємо роботу даної МТ на вхідному слові 132113.
$(s, \underline{1}32113) \to (q_1, 1\underline{3}2113) \to (q_2, 12\underline{2}113) \to (q_2, 122\underline{1}13) \to$
$\to (q_3, 12\underline{2}213) \to (s, 1\underline{2}1213) \to (q_2, 12\underline{1}213) \to (q_3, 1\underline{2}2213) \to$
$\to (s, \underline{1}12213) \to (q_1, 1\underline{1}2213) \to (q_1, 11\underline{2}213) \to (q_2, 112\underline{2}13) \to$
$\to (q_2, 1122\underline{1}3) \to (q_3, 112\underline{2}23) \to (s, 11\underline{2}123) \to (q_2, 112\underline{1}23) \to$
$\to (q_3, 11\underline{2}223) \to (s, 1\underline{1}1223) \to (q_1, 11\underline{1}223) \to (q_1, 111\underline{2}23) \to$
$\to (q_2, 1112\underline{2}3) \to (q_2, 11122\underline{3}) \to (q_2, 111222\underline{\#}) \to (q_f, 11122\underline{2})$

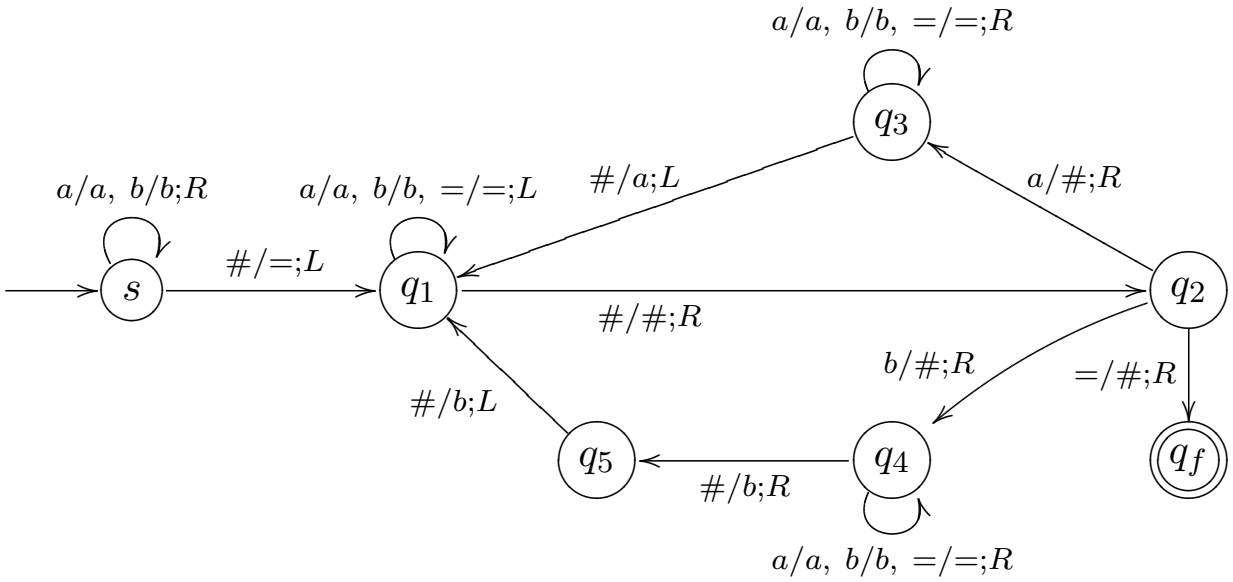
Шукана МТ задається як п'ятірка $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_1, q_2, q_3, q_f\}$, $A = \{1, 2, 3, \#\}$, $X = \{1, 2, 3\}$, $Y = \{1, 2\}$, $D = \{\#\}$ і програма роботи δ задана у вигляді таблиці:

δ	1	2	3	#
s	$q_1, 1, R$	$q_2, 2, R$	$q_2, 2, R$	–
q_1	$q_1, 1, R$	$q_2, 2, R$	$q_2, 2, R$	$q_f, \#, L$
q_2	$q_3, 2, L$	$q_2, 2, R$	$q_2, 2, R$	$q_f, \#, L$
q_3	–	$s, 1, L$	–	–

2.8. Приклад. Побудувати машину Тюрінґа в алфавіті $X = \{a, b\}$, яка подвоює у вхідному слові всі входження букви b.

У попередніх прикладах ми бачили, що вставити чи видалити хоча б одну букву зі слова досить непросто. Тому нове слово будемо

будувати на вільному місці, а саме, праворуч від вхідного, попередньо дописавши вкінці вхідного слова допоміжний символ =. Для цього в початковому стані s переміщуємося праворуч вкінець слова, міняємо перший порожній символ на = і переходимо в новий стан q_1.



Далі нам потрібно переносити букви за знак =, подвоюючи при цьому букву b. Для цього в стані q_1 „біжимо" на початок слова, переходимо в стан q_2 і запам'ятовуємо першу букву: якщо це буква a, то переходимо в стан q_3, якщо b – в q_4. В станах q_3 і q_4 „біжимо" праворуч до першої порожньої комірки і записуємо в неї першу букву, дублюючи її в стані q_5, якщо цією буквою є b. Потім знову повертаємося ліворуч до тієї букви, яка стала першою у вхідному слові, і повторюємо описані вище дії з цією новою першою буквою.

Даний цикл завершиться тоді, коли в стані q_2 головка МТ буде аналізувати символ =. На останньому етапі витираємо допоміжний символ = і переходимо в кінцевий стан q_f.

Програма роботи δ машини Тюрінґа $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_1, q_2, q_3, q_4, q_5, q_f\}$, $A = \{a, b, =, \#\}$, $X = Y = \{a, b\}$, $D = \{\#, =\}$, задається у вигляді таблиці:

δ	a	b	$=$	$\#$
s	s, a, R	s, b, R	$-$	$q_1, =, L$
q_1	q_1, a, L	q_1, b, L	$q_1, =, L$	$q_2, \#, R$
q_2	$q_3, \#, R$	$q_4, \#, R$	$q_f, \#, R$	$-$
q_3	q_3, a, R	q_3, b, R	$q_3, =, R$	q_1, a, L
q_4	q_4, a, R	q_4, b, R	$q_4, =, R$	q_5, b, R
q_5	$-$	$-$	$-$	q_1, b, L

2.9. Приклад. Побудувати машину Тюрінґа в алфавіті $X = \{a, b\}$, яка подвоює слово, поклавши між ним і його копією знак $=$.

Ця задача розв'язується аналогічно до попередньої: праворуч від вхідного слова дописуємо символ $=$, потім повертаємося на початок слова і в циклі копіюємо всі його букви в порожні комірки праворуч.

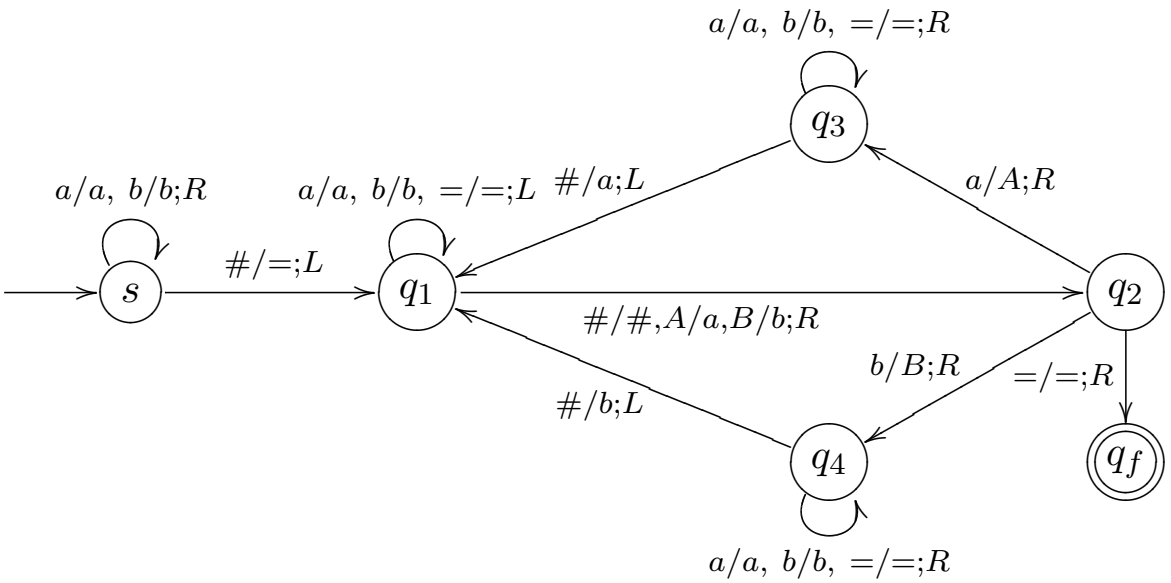
Однак, тут є одна відмінність: букви вхідного слова не видаляються, і це приводить до такої проблеми. Записавши справа копію наступного слова, ми повинні повернутися до вхідного слова в позицію цієї букви і потім переміститися праворуч до наступної букви. Але як дізнатися, в яку позицію вхідного слова потрібно повернутися? В попередній задачі ми завжди поверталися до першої з букв вхідного слова, які залишилися, а тепер ми не видаляємо букви, і тому незрозуміло, які букви ми вже продублювали, а які – ні. Нагадаємо, що в МТ комірки стрічки не номеруються, немає в МТ і лічильників, які б дозволили визначити, скільки букв ми вже продублювали.

В загальному випадку, проблема, з якою ми стикнулись, наступна: як зафіксувати на стрічці деяку позицію, в якій ми вже були і до якої пізніше повинні повернутися? Зазвичай ця проблема вирішується таким чином. Коли ми опиняємося в цій позиції перший раз, то замінюємо активну букву на її двійник – новий допоміжний символ, причому різні символи замінюємо різними двійниками. Після цього виконуємо дії в інших місцях стрічки. Щоб потім повернутися до нашої позиції, потрібно просто відшукати на стрічці ту комірку, де знаходиться буква-двійник. Потім в активній комірці можна відновити попередню букву.

Таким чином, побудуємо МТ за наступним планом:

- спершу записуємо символ $=$ після останньої букви слова;
- потім повертаємося під першу букву вхідного слова;

- далі замінюємо активну букву на її двійник (a на A, b на B), переміщуємося праворуч до першої вільної комірки і записуємо в неї букву, яку запам'ятали;
- після цього повертаємося ліворуч до комірки з двійником, відновлюємо попередній символ і переміщуємося праворуч до наступної букви;
- аналогічним чином в циклі дублюємо решту букв вхідного слова;
- коли ми продублюємо останню букву вхідного слова і повернемося до її двійника, то в комірці праворуч буде знаходитись символ $=$. Це ознака того, що вхідне слово повністю продубльоване, і тому роботу МТ потрібно зупинити.



Таким чином, шукана МТ задається як п'ятірка (Q, A, δ, s, q_f), де $Q = \{s, q_1, q_2, q_3, q_4, q_f\}$, $A = \{a, b, A, B, =, \#\}$, $X = \{a, b\}$, $Y = \{a, b, =\}$, $D = \{A, B, \#\}$ і програма роботи δ задана у вигляді таблиці:

δ	a	b	A	B	$=$	$\#$
s	s, a, R	s, b, R	$-$	$-$	$-$	$q_1, =, L$
q_1	q_1, a, L	q_1, b, L	q_2, a, R	q_2, b, R	$q_1, =, L$	$q_2, \#, R$
q_2	q_3, A, R	q_4, B, R	$-$	$-$	$q_f, =, R$	$-$
q_3	q_3, a, R	q_3, a, R	$-$	$-$	$q_3, =, R$	q_1, a, L
q_4	q_4, a, R	q_4, b, R	$-$	$-$	$q_4, =, R$	q_1, b, L

Рекомендована література : [**8**, с. 183–185], [**20**, с. 7–15], [**22**, с. 163–178].

Питання та вправи до параграфа 2.

В задачах 2.1-2.16 побудувати машину Тюрінґа, яка розпізнає всі слова в алфавіті X і виконує наступну роботу:

2.1. X – український алфавіт. Дописує зліва до вхідного слова ω слово ВУЗ ($\omega \to$ ВУЗω).

2.2. $X = \{$Н, П, У$\}$. Дописує справа до вхідного слова ω слово ПНУ ($\omega \to \omega$ПНУ).

2.3. $X = \{a, b\}$. Залишає в слові тільки першу букву (порожнє слово не міняє).

2.4. $X = \{a, b\}$. Залишає в слові тільки останню букву (порожнє слово не міняє).

2.5. $X = \{a, b, c\}$. Якщо вхідне слово не містить букви c, то замінює всі букви a на b. В іншому випадку – видає слово з однієї букви c.

2.6. $X = \{0, 1, 2, a\}$. З'ясувати, чи є вхідне слово записом числа в трійковій системі числення. Відповідь: 1 – якщо є, 0 – в протилежному випадку.

2.7. $X = \{0, 1, 2\}$. Вважаючи вхідне слово записом числа в трійковій системі числення, видалити з нього всі незначущі нулі.

2.8. $X = \{0, 1, 2\}$. Вважаючи вхідне слово записом числа в трійковій системі числення, з'ясувати, чи ділиться воно на 3. Відповідь: 1 – якщо ділиться, 0 – в протилежному випадку.

2.9. $X = \{a, b\}$. В непорожньому вхідному слові поміняти місцями першу та останню букви.

2.10. $X = \{|\}$. Якщо вхідне слово має непарну довжину, то замінити його на слово $|$, в іншому випадку – на порожнє слово.

2.11. $X = \{a, b, c\}$. Якщо вхідне слово має парну довжину, то залишити в ньому тільки праву половину.

2.12. $X = \{0, 1\}$. Визначити, чи вхідне слово є симетричним (поліндромом), тобто $\omega = \omega^R$. Відповідь: 1 – є, 0 – не є.

2.13. $X = \{a, b\}$. Замінити у вхідному слові кожне входження букви b на ab.

2.14. $X = \{a, b\}$. Видалити із вхідного слова кожне входження підслова ba.

2.15. $X = \{a, b\}$. Подвоїти кожну букву вхідного слова (наприклад, $aba \to aabbaa$).

2.16. $X = \{a, b\}$. Обернути вхідне слово (наприклад, $aab \to baa$).

§ 3. Функції, що обчислюються МТ

В даному параграфі розглядаються функції, для яких існує машина Тюрінґа, що їх обчислює. Дані функції визначаються поведінкою МТ. Аргументи (вхідні слова) з'являються на стрічці до початку обчислення, а значення функції для цього аргументу – слово, яке залишається на стрічці, коли обчислення завершено. Зауважимо, що при аналізі деяких вхідних слів МТ ніколи не переходить у кінцевий стан, і, отже, функція може бути невизначена для деяких вхідних слів-аргументів. В загальному випадку кажемо, що функція $f : (X^*)^n \to Y^*$ є *частково визначеною функцією*, якщо вона не визначена для деяких наборів аргументів $(x_1, \ldots, x_n) \in (X^*)^n$, тобто коли область визначення f є підмножиною $(X^*)^n$. Якщо область визначення f співпадає з $(X^*)^n$, то кажемо, що *функція f повністю визначена на $(X^*)^n$*. Якщо функція f не визначена на наборі $(x_1, \ldots, x_n)$, то писатимемо $f(x_1, \ldots, x_n) = \uparrow$.

Кажемо, що *частково визначена функція $f : (X^*)^n \to Y^*$ обчислюється з допомогою машини Тюрінґа*

$$MT = (Q, A = X \cup Y \cup D, \delta, s, q_f),$$

якщо для кожного набору $(x_1, \ldots, x_n) \in (X^*)^n$, на якому f визначена,

$$(s, x_1 * x_2 * \ldots * x_n) \Rightarrow (q_f, y),$$

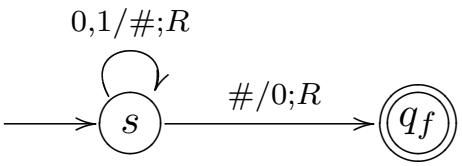
де $y = f(x_1, \ldots, x_n) \in Y^*$, а $* \in D$ – спеціальний допоміжний символ, який розділяє вхідні аргументи; і на кожному наборі

$(x_1, \ldots, x_n) \in (X^*)^n$, для якого функція не визначена, МТ ніколи не перейде в кінцевий стан q_f.

Надалі найчастіше розглядатимемо числові функції $f : (\mathbb{N}_0)^n \to \mathbb{N}_0$, де $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.

3.1. Приклад. Покажемо, що функція $f : \mathbb{N}_0 \to \mathbb{N}_0$, $f(n) = 0$ обчислюється з допомогою деякої машини Тюрінґа.

Вважатимемо, що невід'ємні цілі числа записані в двійковій системі числення. Для побудови МТ, яка правильно обчислює дану функцію, достатньо визначити стани, в яких МТ замінює всі цифри вхідного слова на порожній символ, друкує цифру 0 і зупиняє роботу. Граф шуканої МТ має вигляд:



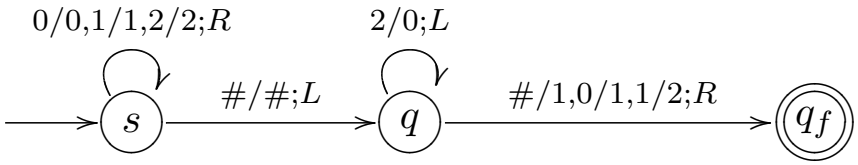
3.2. Приклад. Побудуємо машину Тюрінґа, яка обчислює функцію $f : \mathbb{N} \to \mathbb{N}$, $f(n) = n + 1$ (вважаємо, що натуральні числа задані в трійковій системі числення).

Нехай s – стан, в якому головка МТ рухається в кінець слова. Дійшовши до першої порожньої клітинки, головка МТ переходить ліворуч під останню цифру слова, і керуючий пристрій переходить в стан q. В стані q МТ додає одиницю до тієї цифри, яку вона аналізує в даний момент часу. Спочатку нею є остання цифра числа. Якщо це цифра 0 або 1, то керуючий пристрій замінює її на 1 або 2 відповідно, і МТ зупиняє роботу. Але якщо дана цифра є 2, то керуючий пристрій замінює її на 0, і головка переміщується ліворуч, залишаючись в стані q. Таким чином, тепер МТ додаватиме 1 до попередньої цифри. Якщо нею є цифра 2, то МТ міняє її на 0 і переміщується ліворуч, залишаючись як і до того в стані q, тобто керуючий пристрій повинен виконати ту ж саму дію – збільшити активну цифру на 1. Якщо ж головка перемістилася ліворуч, а в активній комірці немає цифри (а є #), то МТ записує туди 1 і переходить в кінцевий стан q_f.

Зауважимо, що для порожнього вхідного слова наша задача не визначена, тому на цьому слові МТ може поводитися як завгодно.

В нашій програмі, наприклад, на порожньому вхідному слові МТ зупиняється і видає 1.

Граф МТ матиме вигляд:



Під унарною системою числення розумітимемо запис невід'ємного цілого числа з допомогою „паличок": має бути виписано стільки паличок, якою є величина числа, наприклад, $2 \to ||$, $5 \to |||||$, $0 \to$ <порожній символ>.

3.3. Приклад. Побудуємо машину Тюрінґа, яка обчислює функцію $f : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$, $f(n, m) = n + m$ (вважаємо, що натуральні числа задані в унарній системі числення).

МТ починає аналізувати вхідне слово в конфігурації

$(s, \underbrace{||\ldots|}_{n} * \underbrace{||\ldots|}_{m})$, або в конфігурації $(s, \underline{*}\underbrace{||\ldots|}_{m})$, якщо $n = 0$.

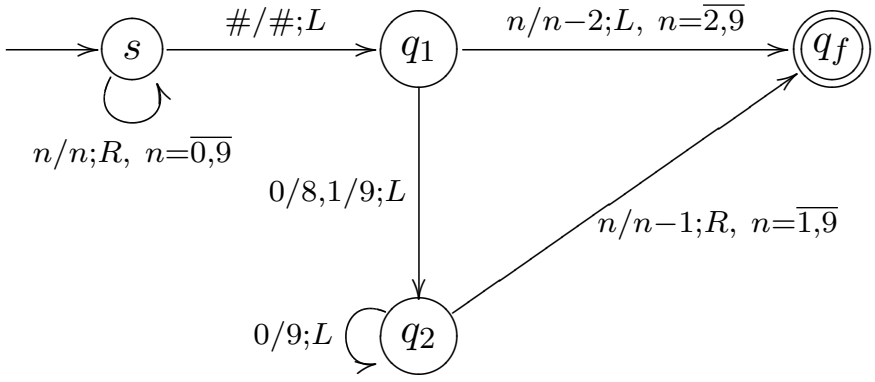
Шукана машина Тюрінґа, яка правильно обчислює дану функцію двох змінних, може працювати, наприклад, таким чином. Якщо $n = 0$, то вона видаляє $*$ і зупиняє роботу. В іншому випадку – видаляє першу паличку $|$, переміщується праворуч до $*$ і на її місці записує $|$. Таким чином $MT = (Q, A = X \cup Y \cup D, \delta, s, q_f)$, де $Q = \{s, q, q_f\}$, $X = Y = \{|\}$, $D = \{*\}$, а програма роботи δ задана у вигляді таблиці:

δ	$\|$	$*$	$\#$
s	$q, \#, R$	$q_f, \#, R$	$-$
q	$q, \|, R$	$q_f, \|, R$	$-$

3.4. Приклад. Побудуємо машину Тюрінґа, яка обчислює частково визначену функцію $f : \mathbb{N}_0 \to \mathbb{N}_0$, $f(x) = x - 2$.

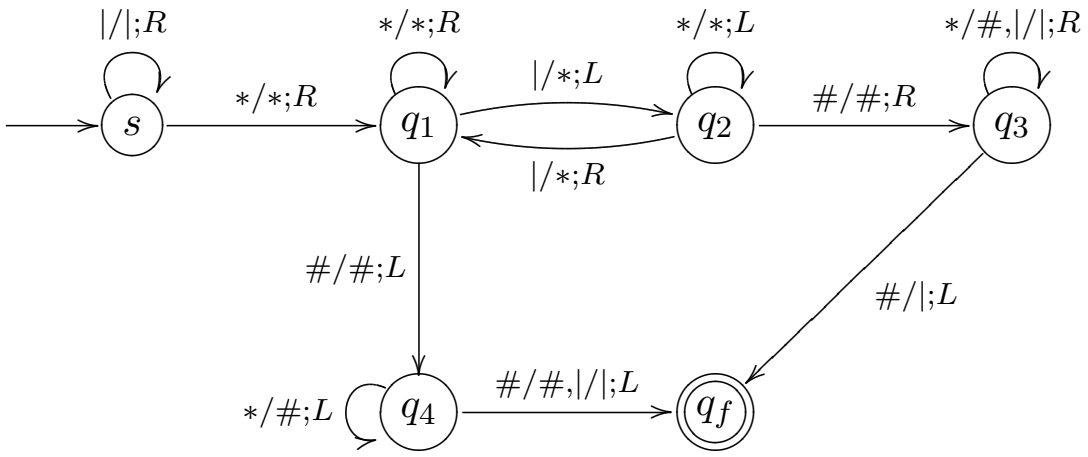
Ідея розв'язання задачі така ж як і в прикладі 3.2. Нехай s – стан, в якому головка МТ рухається вкінець слова. Дійшовши до першої порожньої клітинки головка МТ переходить ліворуч під останню цифру слова, і керуючий пристрій переходить в стан q_1.

Якщо в стані q_1, МТ аналізує цифри $2, 3, \ldots, 9$, то вони замінюються на $0, 1, \ldots, 7$ відповідно, і МТ зупиняє роботу. В іншому випадку, якщо це цифри 0 або 1, то керуючий пристрій замінює їх на 8 або 9 відповідно, і головка переміщується ліворуч, керуючий пристрій при цьому переходить в стан q_2, в якому МТ віднімає 1 від попередньої цифри. Якщо ця цифра дорівнює $1, 2, \ldots, 9$, то МТ міняє її на $0, 1, \ldots, 8$ і зупиняє роботу; інакше – замінює 0 на 9 і переміщується ліворуч, залишаючись як і до того в стані q_2, тобто керуючий пристрій повинен виконати ту ж саму дію – зменшити активну цифру на 1 і т.д. Якщо ж головка, перемістившись ліворуч, в стані q_2 аналізує # (це має місце тільки для чисел 0 та 1), то подальші дії МТ не визначені і до таких чисел МТ не застосовна. Побудуємо граф даної МТ:



3.5. Приклад. Побудуємо машину Тюрінґа, яка обчислює функцію $f : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$, $f(n, m) = |n - m|$ (вважаємо, що натуральні числа задані в унарній системі числення).

Нехай початковою конфігурацією є $(s, \underbrace{|\ldots|}_{n} * \underbrace{|\ldots|}_{m})$. Ідея розв'язання задачі така. Спершу МТ переміщується під першу паличку другого числа. Далі порівнює числа наступним чином: у станах q_1 та q_2 МТ відмічає почергово по одній палочці (замінюючи | на $*$) в обох послідовностях | і, вичерпавши одну із послідовностей, робить висновок, яка з них складається із більшої кількості паличок. Тоді в станах q_3 або q_4 видаляє всі $*$ і на стрічці залишається $n - m$ паличок, якщо $n \geq m$, або $m - n$ паличок, якщо $n < m$. Граф МТ має вигляд:



Проілюструємо роботу МТ на прикладі. Нехай на вході маємо слово $|*|||$. Тоді:

$(s, \underline{|}*|||) \to (s, |\underline{*}|||) \to (q_1, |*\underline{|}||) \to (q_2, |\underline{*}*||) \to (q_2, \underline{|}**||) \to (q_1, *\underline{*}*||) \Rightarrow (q_1, ***\underline{|}|) \to (q_2, ***\underline{*}|) \Rightarrow (q_2, \underline{\#}****|) \to (q_3, \underline{*}***|) \Rightarrow (q_3, |\underline{\#}) \to (q_f, \underline{|}|)$.

3.6. Приклад. Побудуємо МТ, яка обчислює функцію $f : \mathbb{N}_0 \times \mathbb{N} \to \mathbb{N}$, $f(n, m) =$ НСД(n, m).

Нехай цілі невід'ємні числа n та m задані в унарній системі числення. Знайдемо їх найбільший спільний натуральний дільник, користуючись алгоритмом Евкліда. Даний алгоритм базується на таких рекурентних формулах для знаходження НСД:

$$\text{НСД}(n, m) = \begin{cases} \text{НСД}(n - m, m), & n > m; \\ \text{НСД}(n, m - n), & n < m; \\ n, & n = m. \end{cases}$$

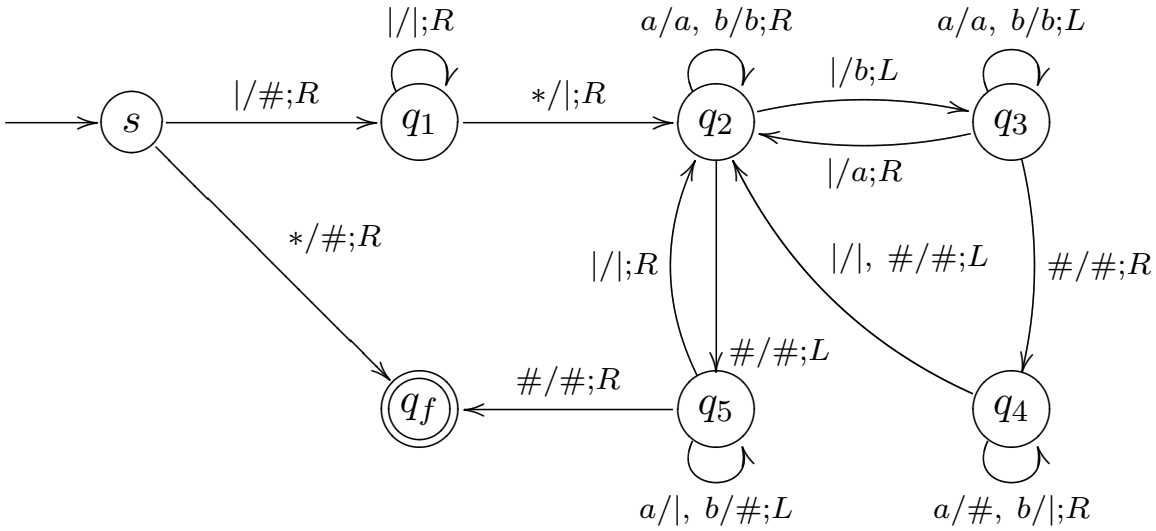
Побудуємо МТ для відшукання НСД, процес обчислення на якій базуватиметься на почерговому використанні циклів порівняння і віднімання.

Процес побудови МТ будемо супроводжувати ілюстрацією конфігурацій для обчислення НСД$(4, 6)$. Початкова конфігурація матиме вигляд $(s, \underline{|}|||*||||||)$. Спершу в станах s та q_1 перенесемо першу паличку на місце $*$. В результаті цього отримаємо конфігурацію $(s, ||||\underline{|}||||||)$. Далі, використовуючи допоміжні мітки a та b в станах q_2 та q_3, будемо порівнювати числа n і m. При цьому МТ працюватиме так, як би працювала людина, яка порівнює дві довгі

послідовності з |, які відразу важко повністю переглянути. Людина відмічала б почергово по одній палочці в обох послідовностях і, вичерпавши одну із послідовностей, зробила б висновок, яка з них складається з більшої кількості паличок. Машина ж замінює першу паличку другого числа на b, потім замінює першу паличку першого числа буквою a, далі знову повертається до паличок другого числа і т.д. Після декількох тактів на стрічці виникає конфігурація $(q_3, \underline{\#}aaaabbbbb|)$, в якій перше число вже вичерпане, а друге – ще ні. Далі МТ переходить в стан q_4, в якому її головка переміщується праворуч, витираючи мітки a та замінюючи мітки b на |. Після цього вона переходить в стан q_2 для порівняння чисел $n = 4$ і $m - n = 2$, знаходячись при цьому в конфігурації $(q_2, |||||\underline{|}|)$ під першою паличкою числа $m - n = 2$. Таким чином, після циклу порівняння виконано цикл віднімання від другого числа першого, в результаті якого менше число n витерто, а більше число m розбито на n і $m - n$.

Далі знову проходить цикл порівняння, який на цей раз закінчується „вичерпанням” другого числа (що знаходиться праворуч), яке в цьому випадку є меншим. При цьому одержимо конфігурацію $(q_2, ||aabb\underline{\#})$. Наступний такт породжує конфігурацію $(q_5, ||aab\underline{b})$, і починається цикл віднімання від першого числа другого, тобто стирання всіх букв b і заміна букв a палочками. Після цього МТ знову переходить в стан q_2 у конфігурації $(q_2, ||\underline{|}|)$. Цей процес триває доти, доки задачу не буде зведено до випадку двох рівних між собою чисел (в нашому випадку ми вже досягли цього). Насамкінець починається останній цикл порівняння, в якому одне з чисел видаляється, МТ переходить у кінцевий стан q_f, і на стрічці залишається число, записане в унарній системі числення, яке згідно алгоритму Евкліда дорівнює НСД(n, m).

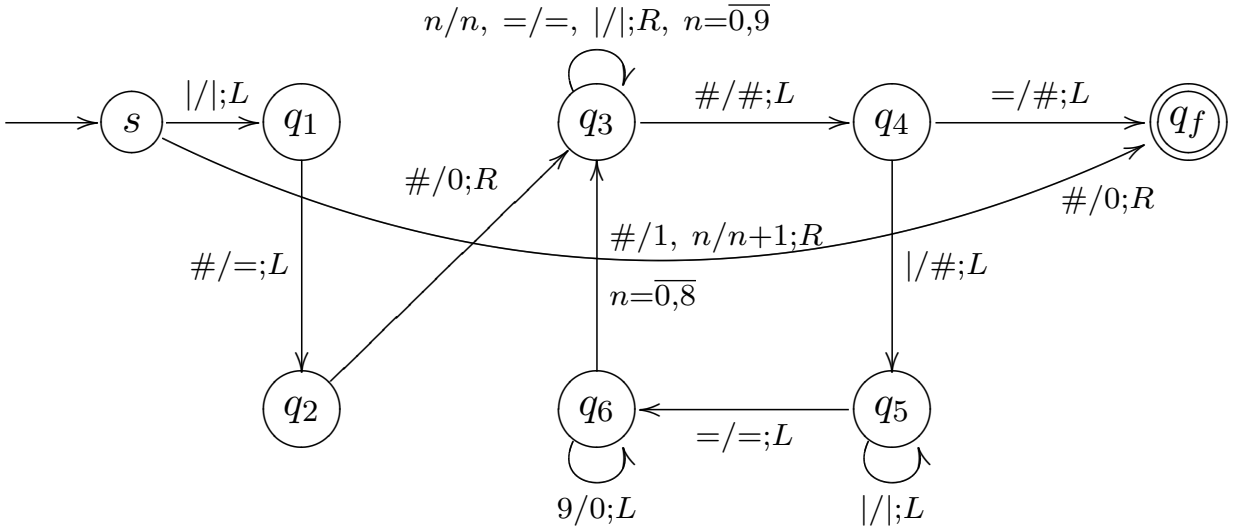
Граф шуканої МТ має вигляд:



Зауважимо, що якщо перше число $n = 0$, то, перебуваючи в конфігурації $(s, \underline{*}|\dots|)$, МТ переходить у кінцевий стан і на виході дає число $m = \text{НСД}(0, m)$, $m \in \mathbb{N}$.

3.7. Приклад. Побудуємо машину Тюрінґа, яка переводить натуральні числа з унарної системи числення в десяткову.

Граф МТ матиме вигляд:



Ідея розв'язання задачі така. Спершу зліва від унарного запису числа, перебуваючи в конфігурації $(s, \underline{|}\dots|)$, в станах q_1 і q_2 дописуємо допоміжний знак $=$ і цифру $\overline{0}$ та переходимо в стан q_3. В результаті цього отримуємо конфігурацію $(q_3, 0\underline{=}|\dots|)$. Далі від унарного числа, розміщеного праворуч, віднімаємо одиницю (видаляємо одну паличку) в стані q_4, переміщуємося ліворуч та в стані

q_6 до десяткового числа додаємо одиницю, діючи за правилами десяткової арифметики. Після цього знову повертаємося до унарного числа і повторюємо циклічно всі операції доти, доки всі палички не будуть витерті. Насамкінець зсуваємося праворуч, в стані q_4 витираємо знак $=$ і зупиняємося.

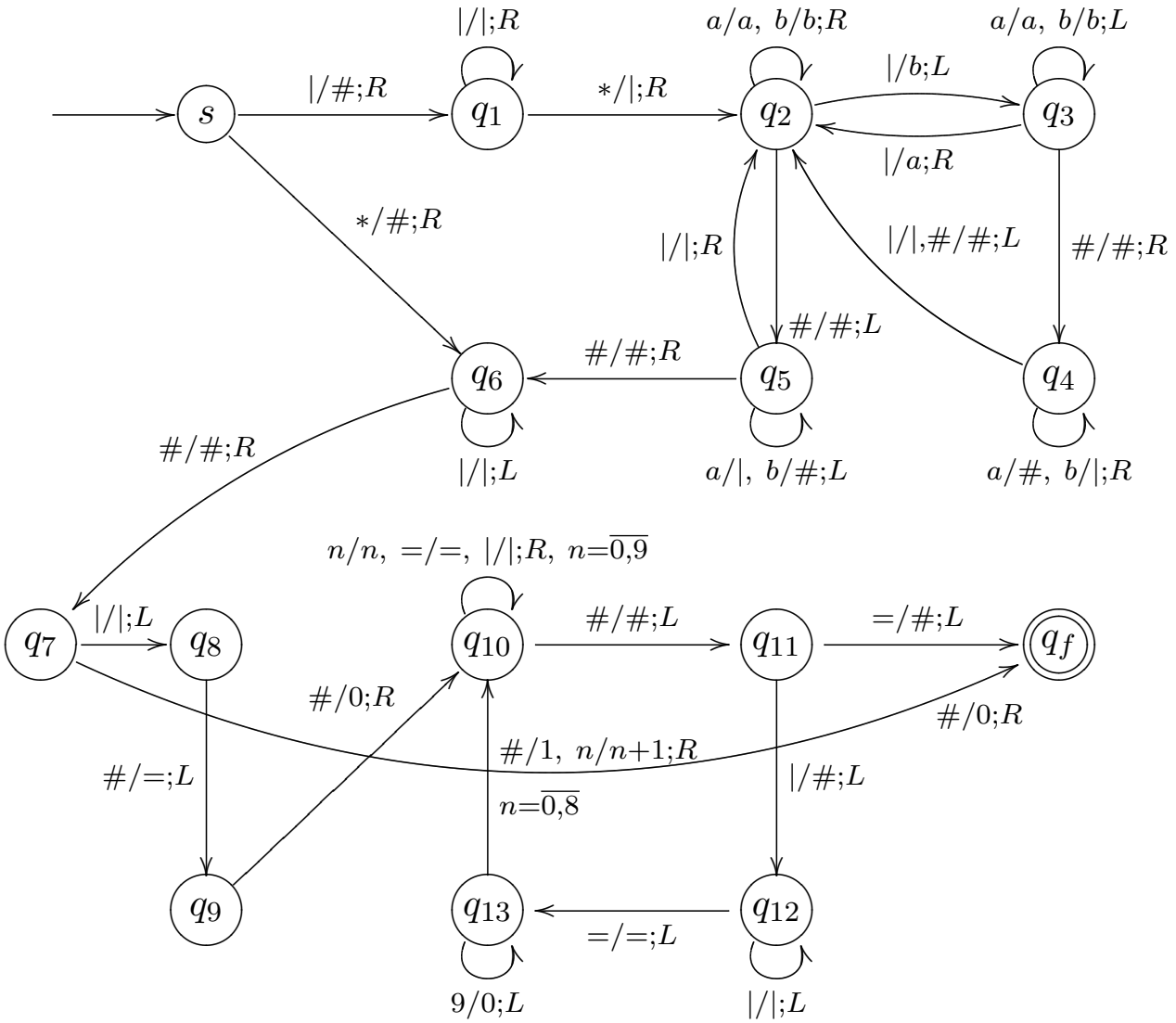
Нехай маємо дві машини Тюрінґа $MT_1 = (Q_1, A_1 = X_1 \cup Y_1 \cup D_1, \delta_1, s_0, s_n)$ і $MT_2 = (Q_2, A_2 = X_2 \cup Y_2 \cup D_2, \delta_2, q_0, q_m)$, причому вважаємо, що вихідний алфавіт першої машини співпадає з вхідним алфавітом другої, тобто $Y_1 = X_2$. Нехай ці МТ обчислюють словесні функції $f_1 : (X_1)^* \to (Y_1)^*$ та $f_2 : (X_2)^* \to (Y_2)^*$ відповідно. Побудуємо МТ, яка обчислює їх композицію $f_2 \circ f_1$. Нагадаємо, що композицією функцій f_1 та f_2 називають таку функцію $f = f_2 \circ f_1$, що для кожного x з області визначення функції f_1 значення $f(x) = f_2(f_1(x))$. При цьому вимагається, щоб функція f_2 була визначена на значенні $f_1(x)$.

Машина Тюрінґа $MT = (Q, A = X \cup Y \cup D, \delta, s, q_f)$ для функції $f = f_2 \circ f_1$ будується наступним чином. У разі потреби стани машини MT_2 перепозначаємо так, щоб вони відрізнялися від станів машини MT_1. Вхідний алфавіт X співпадає із вхідним алфавітом X_1 машини MT_1, вихідний алфавіт Y – з вихідним алфавітом Y_2 машини MT_2, а допоміжний алфавіт $D = X_2 \cup D_1 \cup D_2$. Початковим станом MT є початковий стан s_0 машини MT_1, а кінцевим – кінцевий q_m машини MT_2. Програми роботи δ_1 і δ_2 об'єднуємо і записуємо в одну таблицю, доозначивши, що в стані s_n шукана МТ переміщується ліворуч до початку слова і потім переходить в стан q_0.

Нехай $\omega \in (X_1)^*$. Розглянемо початкову конфігурацію (s, ω). Оскільки $s = s_0$, то на початку роботи MT працює так, як MT_1, і якщо MT_1 застосовна до слова ω, то на деякому кроці буде одержано конфігурацію $(s_n, f_1(\omega))$. Далі в стані s_n головка машини MT переміститься ліворуч під першу букву слова $f_1(\omega)$ і керуючий пристрій перейде в стан q_0, тобто одержимо конфігурацію $(q_0, f_1(\omega))$. Якщо MT_2 застосовна до слова $f_1(\omega)$, то на деякому кроці отримаємо конфігурацію $(q_m, f_2(f_1(\omega)))$, яка є заключною для MT, бо $q_f = q_m$. Таким чином, якщо MT_1 застосовна до ω і MT_2 застосовна до $f_1(\omega)$, то і MT застосовна до ω. Машина MT називається композицією машин MT_1 та MT_2 і позначається через $MT_2 \circ MT_1$.

Означення композиції залишається без зміни, якщо MT_1 та MT_2 обчислюють функції кількох змінних. Важливо лише, щоб дані для MT_2 були у відповідному вигляді підготовлені машиною MT_1.

3.8. Приклад. Розглянемо МТ, задану графом:



Дана МТ обчислює функцію $f : \mathbb{N}_0 \times \mathbb{N} \to \mathbb{N}$, $f(n, m) =$ НСД(n, m) і видає результат в десятковій системі числення. Вона є композицією МТ, побудованих в прикладах 3.6 та 3.7.

Аналізуючи попередні приклади, у читача складається враження, що процес, який відбувається на МТ, є сповільненим переглядом процесу обчислення, який виконується людиною відповідно до деякого алгоритму. Разом з тим розглянуті приклади наштовхують на думку, що з допомогою МТ можна задавати й інші відомі нам алгоритми. Цілком природно виникає питання: чи спосіб задання

алгоритмів з допомогою МТ є універсальним в тому сенсі, що будь-який алгоритм можна подати таким чином? На це питання сучасна теорія алгоритмів дає відповідь з допомогою наступної гіпотези:

Теза Тюрінґа-Черча. *Будь-який алгоритм можна реалізувати на відповідній машині Тюрінґа.*

Перш за все звернемо увагу на наступну характерну особливість даної гіпотези. В її формулюванні мова йде, з одного боку, про загальне поняття алгоритму, яке не є точним математичним поняттям; з другого боку, в цьому ж формулюванні говориться про таке точне математичне поняття як машина Тюрінґа. Тому не може йти і мови про доведення даної гіпотези подібно до того, як доводяться теореми в математиці. Значення гіпотези полягає саме в тому, що вона уточнює загальне, але розпливчасте поняття „будь-якого алгоритму" з допомогою цілком точного математичного поняття МТ. Таким чином, теорія алгоритмів оголошує об'єктом своїх досліджень машини Тюрінґа. Тепер вже стають коректними питання можливості чи неможливості побудови алгоритму для задачі того чи іншого типу.

Справедливість гіпотези підтверджується практикою, яка, як відомо з філософії, є єдиним критерієм істини. Всі відомі алгоритми, які були придумані впродовж багатьох тисячоліть історії математики, можуть бути задані з допомогою МТ.

Рекомендована література : [**1**, с. 33–45], [**8**, с. 186–194], [**11**, с. 324–333], [**17**, с. 319–323], [**24**, с. 83–86].

Питання та вправи до параграфа 3.

3.1. Дайте означення частково визначеної функції, що обчислюється МТ.

3.2. Що ви розумієте під композицією МТ?

3.3. Побудувати машину Тюрінґа, що обчислює числову функцію $f : \mathbb{N}_0 \to \mathbb{N}_0$, аргументи якої задані в трійковій системі числення.

б) $f(x) = 0$;

а) $f(x) = \begin{cases} 0, & x = 0 \\ 1, & x \neq 0; \end{cases}$

б) $f(x) = x + 2$;

в) $f(x) = x - 2$;
г) $f(x) = 2 \cdot x + 1$;
д) $f(x) = 9_{10} \cdot x + 2$.

3.4. Побудувати машину Тюрінґа, яка обчислює числову функцію $P_i^4 : (N_0)^4 \to \mathbb{N}_0$, $P_i^4(x_1, x_2, x_3, x_4) = x_i$, де $i = 1, 2, 3, 4$ (вважаємо, що аргументи x_k задані в унарній системі числення).

3.5. Побудувати машину Тюрінґа, що обчислює частково визначену числову функцію $f : (\mathbb{N}_0)^2 \to \mathbb{N}_0$, аргументи якої задані в унарній системі числення.

а) $f(x, y) = \begin{cases} x - y, & x > y \\ 0, & x \leq y; \end{cases}$

б) $f(x, y) = \begin{cases} x - y, & x \geq y \\ \uparrow, & x < y; \end{cases}$

в) $f(x, y) = xy$;
г) $f(x, y) = \min\{x, y\}$;
д) $f(x, y) = \max\{x, y\}$;
е) $f(x, y) = [\frac{x}{y}]$;
є) $f(x, y) = x - y[\frac{x}{y}]$.

3.6. Побудувати машину Тюрінґа, що обчислює частково визначену числову функцію $f : \mathbb{N}_0 \to \mathbb{N}_0$, аргументи якої задані в унарній системі числення.

а) $f(x) = \begin{cases} \frac{x}{2}, & x - \text{ парне} \\ \uparrow, & x - \text{ непарне}; \end{cases}$

б) $f(x) = [\frac{x}{2}]$.

3.7. Побудувати машину Тюрінґа, яка обчислює частково визначену числову функцію $f : (\mathbb{N}_0)^2 \to \mathbb{N}_0$.

а) $f(x, y) = x + y$;
б) $f(x, y) = x - y$.

Вважаємо, що аргумент x заданий в четвірковій системі числення, а y – в трійковій системі. Результат отримати в четвірковій системі числення.

3.8. Доведіть, що функція

$$f_p(x) = \begin{cases} 1, & \text{якщо } x \text{ ділиться на } p \\ 0, & \text{якщо } x \text{ не ділиться на } p \end{cases}$$

обчислюється деякою МТ.

3.9. Побудувати машину Тюрінґа, яка обчислює словесну функцію $f : \{a, b\}^* \to \{a, b\}^*$.

а) $f(\omega) = \omega ab$;
б) $f(\omega) = \omega^R$;
в) $f(\omega) = \omega\omega$;
г) $f(\omega) = \omega\omega^R$.

3.10. Побудувати машину Тюрінґа, яка переводить натуральні числа з трійкової системи числення в унарну.

3.11. Побудувати машину Тюрінґа, яка переводить натуральні числа з четвіркової системи числення в двійкову.

3.12. Побудувати машину Тюрінґа, яка переводить натуральні числа з двійкової системи числення в четвіркову (підказка: врахуйте, що в двійковому числі може бути непарна кількість букв).

3.13. Скориставшись композицією МТ, побудуйте машину Тюрінґа, яка за записом числа в унарній системі числення визначає, чи є це число степенем 3 $(1, 3, 9, 27, \ldots)$. Відповідь: 1 якщо так, 0 – в протилежному випадку.

§ 4. Рекурсивні функції

Будь-який алгоритм співставляє допустимим вхідним даним результат. Це означає, що з кожним алгоритмом однозначно зв'язана функція, яку він обчислює. В цьому параграфі розглядатимуться алгоритмічно обчислювані функції. Ми побудуємо клас всіх функцій такого виду з допомогою тільки трьох найпростіших функцій і трьох операцій над такими функціями. Даний підхід до формалізації поняття алгоритму належить Геделю. Основна ідея Геделя полягала в тому, щоб одержати всі обчислювані функції із невеликої кількості базових функцій з допомогою найпростіших алгоритмічних засобів.

Назвемо найпростішими такі функції:

- нуль-функцію: $O(x) = 0$, $x \in \mathbb{N}_0 = \mathbb{N} \cup \{0\}$;
- функцію наступності: $S(x) = x + 1$, $x \in \mathbb{N}_0$;
- функції проектування: $P_i^n(x_1, x_2, \ldots, x_i, \ldots x_n) = x_i$, $x_i \in \mathbb{N}_0$.

До даних функцій будемо застосовувати наступні операції.

Композиція. Нехай $m, k \in \mathbb{N}$ і задано числові функції $g : \mathbb{N}_0^m \to \mathbb{N}_0$ і $h_i : \mathbb{N}_0^k \to \mathbb{N}_0$, де $i = 1, 2, \ldots, m$. Визначимо функцію $f : \mathbb{N}_0^k \to \mathbb{N}_0$ за правилом

$$f(n_1, \ldots, n_k) = g(h_1(n_1, \ldots, n_k), \ldots, h_m(n_1, \ldots, n_k)).$$

Кажемо, що *функція f одержана з допомогою операції композиції з функцій g і $h_1, \ldots, h_m$*. Функцію f позначимо через $g \circ (h_1, \ldots, h_m)$.

Примітивна рекурсія. Нехай $k \in \mathbb{N}_0$ і задано числові функції $g : \mathbb{N}_0^k \to \mathbb{N}_0$ і $h : \mathbb{N}_0^{k+2} \to \mathbb{N}_0$ (у випадку, коли $k = 0$, функція g ототожнюється з числом із множини $\mathbb{N}_0$). Визначимо функцію $f : \mathbb{N}_0^{k+1} \to \mathbb{N}_0$ за правилом

$$f(n_1, \ldots, n_k, 0) = g(n_1, \ldots, n_k),$$

$$f(n_1, \ldots, n_k, m + 1) = h(n_1, \ldots, n_k, m, f(n_1, \ldots, n_k, m)), m \geq 0.$$

В цьому випадку кажемо, що *функція f отримана з функцій g і h з допомогою операції примітивної рекурсії*.

Клас примітивно рекурсивних функцій визначається наступним чином:

- найпростіші функції O, S і P_i^n є примітивно рекурсивними функціями;
- якщо $g : \mathbb{N}_0^m \to \mathbb{N}_0$ і $h_i : \mathbb{N}_0^k \to \mathbb{N}_0$, де $i = 1, 2, \ldots, m$, є примітивно рекурсивними функціями, то $g \circ (h_1, \ldots, h_m)$ також є примітивно рекурсивною функцією;
- якщо $g : \mathbb{N}_0^k \to \mathbb{N}_0$ і $h : \mathbb{N}_0^{k+2} \to \mathbb{N}_0$ є примітивно рекурсивними функціями, то функція f, одержана з g і h за допомогою операції примітивної рекурсії, також є примітивно рекурсивною;
- інших примітивно рекурсивних функцій не існує.

Іншими словами, функцію називають *примітивно рекурсивною*, якщо її можна одержати з найпростіших функцій за допомогою

скінченної кількості застосувань операцій композиції та примітивної рекурсії.

4.1. Приклад. Довести, що функція $O_k(n_1, n_2, \ldots, n_k) = 0$ є примітивно рекурсивною.

Дану функцію можна зобразити у вигляді композиції двох найпростіших примітивно рекурсивних функцій. Дійсно,

$$O_k(n_1, n_2, \ldots, n_k) = O(P_1^k(n_1, n_2, \ldots, n_k)).$$

Таким чином, функція $O_k = O \circ P_1^k$ є примітивно рекурсивною.

4.2. Приклад. Показати, що функції $S_i^k(n_1, \ldots, n_i, \ldots, n_k) = n_i + 1$, де $i = 1, \ldots, k$, є примітивно рекурсивними.

Функції S_i^k можна визначити як

$$S_i^k(n_1, n_2, \ldots, n_k) = S(P_i^k(n_1, \ldots, n_i, \ldots, n_k)).$$

Отже, функція $S_i^k = S \circ P_i^k$ є примітивно рекурсивною, бо вона одержана з примітивно рекурсивних функцій з допомогою операції композиції.

4.3. Приклад. Довести, що сталі функції $C_a^k(n_1, n_2, \ldots, n_k) = a$, де $k, a \in \mathbb{N}_0$, є примітивно рекурсивними.

Дані функції можна зобразити у вигляді скінченної кількості композицій примітивно рекурсивних функцій

$$C_a^k(n_1, n_2, \ldots, n_k) = \underbrace{S(S \ldots (S}_{a}(O_k(n_1, n_2, \ldots, n_k))) \ldots).$$

4.4. Приклад. Показати, що функція $Suma : \mathbb{N}_0^2 \to \mathbb{N}_0$, $Suma(n, m) = n + m$ є примітивно рекурсивною.

Функцію $Suma$ можна визначити наступним чином.

$$Suma(n, 0) = n = P_1^1(n),$$

$$Suma(n, m+1) = n + (m+1) = (n+m) + 1 = S(P_3^3(n, m, Suma(n, m))).$$

Отже, $Suma$ є примітивно рекурсивною функцією, оскільки її одержано з примітивно рекурсивних функцій $P_1^1 : \mathbb{N}_0 \to \mathbb{N}_0$ і $S \circ P_3^3 : \mathbb{N}_0^3 \to \mathbb{N}_0$ з допомогою операції примітивної рекурсії.

4.5. Приклад. Довести, що функція $Prod : \mathbb{N}_0^2 \to \mathbb{N}_0$, $Prod(n, m) = nm$ є примітивно рекурсивною.

Функцію $Prod$ можна одержати з примітивно рекурсивних функцій O і $Suma \circ (P_3^3, P_1^3)$ з допомогою операції примітивної рекурсії наступним чином.

$$Prod(n,0) = 0 = O(n),$$
$$Prod(n,m+1) = n(m+1) = nm + n =$$
$$= Suma(P_3^3(n,m,Prod(n,m)), P_1^3(n,m,Prod(n,m))).$$

4.6. Приклад. Показати, що функція $Exp(n,m) = n^m$ є примітивно рекурсивною.

Функцію Exp можна визначити як

$$Exp(n,0) = 1 = S(O(n)),$$
$$Exp(n,m+1) = n^{m+1} = n^m n =$$
$$= Prod(P_3^3(n,m,Exp(n,m)), P_1^3(n,m,Exp(n,m))).$$

4.7. Приклад. Довести, що функції

а) $Prec : \mathbb{N}_0 \to \mathbb{N}_0, \quad Prec(n) = n \div 1 = \begin{cases} 0, & n = 0 \\ n-1, & n > 0 \end{cases}$

б) $Minus : \mathbb{N}_0^2 \to \mathbb{N}_0, \quad Minus(n,m) = n \div m = \begin{cases} 0, & n \le m \\ n-m, & n > m \end{cases}$

в) $Abs(n,m) = |n-m|$ є примітивно рекурсивними.

а) Функцію $Prec$ можна одержати з константи і проектування $P_1^2 : \mathbb{N}_0^2 \to \mathbb{N}_0$ з допомогою операції примітивної рекурсії.

$$Prec(0) = 0,$$
$$Prec(m+1) = m = P_1^2(m, Prec(m)).$$

б) Функцію $Minus$ можна одержати з допомогою функції $Prec$ наступним чином.

$$Minus(n,0) = n = P_1^1(n),$$
$$Minus(n,m+1) = n \div (m+1) = (n \div m) \div 1 =$$
$$= Prec(P_3^3(n,m,Minus(n,m))).$$

в) $Abs(n,m) = |n-m| = (n \div m) + (m \div n) =$
$$= Suma(Minus(n,m), Minus(m,n)).$$

Оскільки метою цього розділу є показати, що клас примітивно рекурсивних функцій є досить широким, то слід довести, що булеві функції, які використовуються в мовах програмування високого рівня, є примітивно рекурсивними. В наступних вправах через 1 позначаємо значення „істина”, через 0 – „хибність”.

4.8. Приклад. Показати, що наступні функції є примітивно рекурсивними.

а) $Neg(n) = \begin{cases} 0, n \geq 1 \\ 1, n = 0 \end{cases}$

б) $And(n,m) = \begin{cases} 1, n \geq 1 \text{ і } m \geq 1 \\ 0, \text{ в протилежному випадку} \end{cases}$

в) $Or(n,m) = \begin{cases} 1, n \geq 1 \text{ або } m \geq 1 \\ 0, \text{ в протилежному випадку} \end{cases}$

г) $If\text{-}then\text{-}else(n,m,k) = \begin{cases} m, n \geq 1 \\ k, \text{ в протилежному випадку.} \end{cases}$

Дійсно, дані функції утворюються з примітивно рекурсивних функцій наступним чином:

а) $Neg(n) = Minus(1,n)$;
б) $And(n,m) = Neg(Neg(Prod(n,m)))$;
в) $Or(n,m) = Neg(And(Neg(n), Neg(m)))$;
г) $If\text{-}then\text{-}else(n,m,k) =$

$$= Suma(Prod(Neg(Neg(n)),m), Prod(Neg(n),k))).$$

Надалі писатимемо „x *and* y”, „x *or* y” та „if x then y else z” замість $And(x,y)$, $Or(x,y)$ та $If\text{-}then\text{-}else(x,y,z)$ відповідно.

4.9. Приклад. Довести, що наступні функції є примітивно рекурсивними.

а) $Eq(n,m) = \begin{cases} 1, n = m \\ 0, n \neq m \end{cases}$

б) $Gr(n,m) = \begin{cases} 1, n > m \\ 0, n \leq m \end{cases}$

в) $Geq(n,m) = \begin{cases} 1, n \geq m \\ 0, n < m \end{cases}$

г) $Ls(n,m) = \begin{cases} 1, n < m \\ 0, n \geq m \end{cases}$

д) $Leq(n,m) = \begin{cases} 1, n \leq m \\ 0, n > m \end{cases}$

Дані функції утворюються з примітивно рекурсивних функцій так:

а) $Eq(n,m) = Neg(Add(Minus(n,m), Minus(m,n)))$;
б) $Gr(n,m) = Neg(Neg(Minus(n,m)))$;
в) $Geq(n,m) = Gr(n,m)$ or $Eq(n,m)$;
г) $Ls(n,m) = Neg(Geq(n,m))$;
д) $Leq(n,m) = Neg(Gr(n,m))$.

4.10. Приклад. Довести, що для кожного $k \in \mathbb{N}$ функція

$$Max^k(n_1, n_2, \ldots, n_k) = \max\{n_1, n_2, \ldots, n_k\}$$

є примітивно рекурсивною.

Доведемо дане твердження індукцією по k. Для $k = 1$ твердження вірне, бо $Max^1(n_1) = P_1^1(n_1)$. Припустимо, що твердження виконується для $k-1$ і доведемо для k:
$Max^k(n_1, \ldots, n_k) =$
$=$ if $n_k >$ $Max^{k-1}(n_1, \ldots, n_{k-1})$ then n_k else $Max^{k-1}(n_1, \ldots, n_{k-1})$.

4.11. Приклад. Застосувати операцію примітивної рекурсії до примітивно рекурсивних функцій $g : \mathbb{N}_0 \to \mathbb{N}_0$ і $h : \mathbb{N}_0^3 \to \mathbb{N}_0$.

а) $g(n) = 3^n$, $h(n_1, n_2, n_3) = 4n_2 + n_3$;
б) $g(n) = 2^n$, $h(n_1, n_2, n_3) = 3^{n_1} n_3^{n_1}$ (вважаємо, що $0^0 = 1$).

а) $f(n,0) = g(n) = 3^n$
$f(n, m+1) = h(n, m, f(n,m)) = 4m + f(n,m)$
$f(n,1) = 4 \cdot 0 + f(n,0) = 3^n$
$f(n,2) = 4 \cdot 1 + f(n,1) = 4 + 3^n$
$f(n,3) = 4 \cdot 2 + f(n,2) = 4 \cdot 2 + 4 \cdot 1 + 3^n$
Доведемо індукцією по m, що
$f(n,m) = 4((m-1) + \ldots + 2 + 1) + 3^n = 4\frac{(m-1)+1}{2}(m-1) + 3^n =$
$= 2m(m-1) + 3^n$.

Припустимо, що твердження виконується для $m = k$, тобто $f(n,k) = 2k(k-1) + 3^n$.

Для $m = k + 1$ маємо $f(n, k + 1) = 4k + f(n, k) = 4k + 2k(k - 1) + 3^n = 2k(2 + k - 1) + 3^n = 2(k + 1)k + 3^n$.

Таким чином, для будь-яких $n, m \in \mathbb{N}_0$

$$f(n, m) = 2m(m - 1) + 3^n$$

Функція f отримана з примітивно рекурсивних функцій g і h з допомогою операції примітивної рекурсії і, отже, є примітивно рекурсивною.

б) $f(n, 0) = g(n) = 2^n$

$f(n, m + 1) = h(n, m, f(n, m)) = 3^n(f(n, m))^n$

$f(n, 1) = 3^n(f(n, 0))^n = 3^n 2^{n^2}$

$f(n, 2) = 3^n(3^n 2^{n^2})^n = 3^n 3^{n^2} 2^{n^3} = 3^{n+n^2} 2^{n^3}$

$f(n, 3) = 3^n(3^{n+n^2} 2^{n^3})^n = 3^{n+n^2+n^3} 2^{n^4}$

Доведемо індукцією по m, що

$f(n, m) = 3^{n+n^2+\ldots n^m} 2^{n^{m+1}} = 3^{\frac{n(n^m-1)}{n-1}} 2^{n^{m+1}}$.

Припустимо, що твердження виконується для $m = k$, тобто $f(n, k) = 3^{\frac{n(n^k-1)}{n-1}} 2^{n^{k+1}}$.

Для $m = k + 1$ маємо

$f(n, k+1) = 3^n(f(n, k))^n = 3^n(3^{\frac{n(n^k-1)}{n-1}} 2^{n^{k+1}})^n = 3^n 3^{\frac{n^2(n^k-1)}{n-1}} 2^{n^{k+2}}$
$= 3^{n+\frac{n^2(n^k-1)}{n-1}} 2^{n^{k+2}} = 3^{\frac{n^2-n+n^{k+2}-n^2}{n-1}} 2^{n^{k+2}} = = 3^{\frac{n(n^{k+1}-1)}{n-1}} 2^{n^{k+2}}$.

Таким чином, для будь-яких $n, m \in \mathbb{N}_0$

$$f(n, m) = 3^{\frac{n(n^m-1)}{n-1}} 2^{n^{m+1}}.$$

Дана функція f отримана з примітивно рекурсивних функцій g та h, і тому є примітивно рекурсивною.

Мінімізація. Розглянемо частково визначену функцію n змінних $f(x_1, \ldots, x_n)$. Визначимо функцію $g(x_1, \ldots, x_n)$ наступним чином. Зафіксуємо певні $n-1$ значення аргументів $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots x_n$ функції f та розглянемо рівняння $f(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n) = x_i$. Обчислюватимемо значення $f(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n)$ для $y = 0, 1, 2, \ldots$.

Розглянемо наступні випадки:

- значення $f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$ не визначено;

- значення $f(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n)$ для $y = 0, 1, \ldots, a-1$ визначені, але вони відмінні від x_i, а значення $f(x_1, \ldots, x_{i-1}, a, x_{i+1}, \ldots, x_n)$ не визначено;
- значення $f(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n)$ визначені для всіх $y \in \mathbb{N}_0$, але вони відмінні від x_i.

В усіх цих випадках значення функції $g(x_1, \ldots, x_n)$ вважають не визначеним. Інакше, нехай

$$a = \min\{y \in \mathbb{N}_0 \mid f(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n) = x_i\}.$$

У цьому випадку зазначений процес зупиняється і дає найменший розв'язок $y = a$ рівняння $f(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n) = x_i$, який є значенням функції $g(x_1, \ldots, x_n) = a$. Значення функції g для заданої функції f залежить від значень параметрів $x_1, \ldots, x_n$, тому функція g є частково визначеною функцією змінних $x_1, \ldots, x_n$. Кажуть, що функція g отримана з функції f з допомогою *операції мінімізації за змінною* x_i.

Функцію f називають *частково рекурсивною*, якщо вона може бути утворена з найпростіших функцій за допомогою скінченної кількості застосувань операцій композиції, примітивної рекурсії та мінімізації. Всюди визначені частково рекурсивні функції називаються *загальнорекурсивними*.

4.12. Приклад. Розглянемо функцію $f : \mathbb{N}_0 \to \mathbb{N}_0$, $f(x) = x + 2$. За допомогою операції мінімізації визначимо функцію $g : \mathbb{N}_0 \to \mathbb{N}_0$. Для цього розглянемо рівняння $y + 2 = x$. Бачимо, що функція g не визначена, якщо $x = 0$ або $x = 1$, і $g(x) = x - 2$, якщо $x \geq 2$.

4.13. Приклад. Доведемо, що частково визначена функція $g : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$,

$$g(x_1, x_2) = \frac{x_2}{1 - x_1 x_2^{2012}}$$

є частково рекурсивною. Застосуємо операцію мінімізації за змінною x_2 до примітивно рекурсивної функції $f(x_1, x_2) = (1 \div x_1)x_2$. В результаті отримаємо рівняння $(1 \div x_1)y = x_2$. Отже,

$$g(x_1, x_2) = \begin{cases} x_2, & x_1 = 0, \\ 0, & x_2 = 0, \\ \uparrow, & x_1 \neq 0 \text{ і } x_2 \neq 0, \end{cases} \qquad = \frac{x_2}{1 - x_1 x_2^{2012}}.$$

В даному параграфі ми показали, що клас частково рекурсивних функцій є досить широким. Поняття частково рекурсивної функції – одне з основних понять теорії алгоритмів. Його значення є таким. З одного боку, кожна стандартно задана частково рекурсивна функція є обчислювана за певною процедурою, яка відповідає інтуїтивному уявленню алгоритму, а з іншого – які б досі не будувалися класи точно визначених алгоритмів, завжди з'ясовувалося, що числові функції, які обчислювалися за алгоритмами цих класів, були частково рекурсивними. Тому загальноприйнятою є така наукова гіпотеза:

Теза Черча. *Клас алгоритмічно обчислюваних частково визначених числових функцій збігається з класом усіх частково рекурсивних функцій.*

У формулювання цієї тези входить інтуїтивне поняття обчислюваності, тому її не можна ні довести, ні спростувати в загальноматематичному значенні. Це факт, на користь якого свідчить багаторічна математична практика. Проте справедливою є наступна теорема, доведення якої читач може знайти в [**11**]:

4.14. Теорема. *Функція є обчислюваною за Тюрінґом тоді і тільки тоді, коли вона є частково рекурсивною.*

Рекомендована література : [**8**, с. 195–202], [**11**, с. 333–354], [**14**], [**17**, с. 323–327], [**21**, с. 194–207].

Питання та вправи до параграфа 4.

4.1. Дайте означення примітивно рекурсивної функції.

4.2. Опишіть операцію мінімізації частково рекурсивної функції.

4.3. Як пов'язнані частково рекурсивні функції і машини Тюрінґа?

4.4. Довести, що наступні функції однієї змінної є примітивно рекурсивними:

а) $f(n) = 3n + 4$;
б) $f(n) = 2^n + n^2$;
в) $f(n) = n!$;

г) $sg(n) = \begin{cases} 0, & n = 0 \\ 1, & n > 0. \end{cases}$

4.5. Довести, що наступні функції $f : \mathbb{N}_0^2 \to \mathbb{N}_0$ є примітивно рекурсивними:

а) $f(n, m) = \min\{n, m\}$;
б) $f(n, m) = \max\{n, m\}$.

4.6. Застосувати операцію примітивної рекурсії до функцій $g : \mathbb{N}_0 \to \mathbb{N}_0$ і $h : \mathbb{N}_0^3 \to \mathbb{N}_0$. Записати результуючу функцію $f : \mathbb{N}_0^2 \to \mathbb{N}_0$ аналітично.

а) $g(n) = 2^n$, $h(n_1, n_2, n_3) = 3n_2 + n_3$;
б) $g(n) = 2012n^{2012}$, $h(n_1, n_2, n_3) = 5n_2 + n_3$;
в) $g(n) = 3^n$, $h(n_1, n_2, n_3) = 2n_1 + 5n_3$;
г) $g(n) = 2012^n$, $h(n_1, n_2, n_3) = 5^{n_1} n_3^{n_1}$ (вважаємо, що $0^0 = 1$);
д) $g(n) = 2^n + n^2$, $h(n_1, n_2, n_3) = n_1 + 3n_3$;
е) $g(n) = 1$, $h(n_1, n_2, n_3) = n_3(sg|n_1 + 2 - 2n_3|)$.

4.7. Застосувати операцію мінімізації до функції f за всіма її змінними. Подати вихідні функції g аналітично.

а) $f(n) = 2$;
б) $f(n) = [\frac{n}{3}]$;
в) $f(n, m) = n + m$;
г) $f(n, m) = 2 - n - m$;
д) $f(n, m) = P_1^2(n, m)$;
е) $f(n, m) = \max\{n, m\}$;
є) $f(n, m) = \min\{n, m\}$;
ж) $f(n, m) = n \div m$.

4.8. Застосувавши операцію мінімізації до відповідної примітивно рекурсивної функції f, довести, що функція g є частково рекурсивною.

а) $g(n) = 3 - n$;
б) $g(n) = \frac{n}{3}$;
в) $g(n, m) = n - 2m$;
г) $g(n) = \frac{1}{n+1}$;
д) $g(n, m) = \frac{n}{m+2}$.

4.9. Довести, що застосувавши один раз операцію мінімізації до всюди визначеної числової функції, отримаємо функцію, яка визначена принаймні в одній точці.

4.10. Навести приклад функції $f : \mathbb{N}_0 \to \mathbb{N}_0$, застосувавши до якої операцію мінімізації двічі, отримаємо ніде не визначену функцію.

4.11. Сформулювати необхідну і достатню умову того, що функція g, отримана з функції f з допомогою операції мінімізації, є

а) всюдивизначеною;
б) ніде не визначеною.

§ 5. Нормальні алгоритми Маркова

В даному параграфі розглядається ще один важливий підхід до уточнення поняття алгоритму, розроблений А.А. Марковим. Нормальні алгоритми Маркова (НАМ) базуються на перетворенні слів в деякому скінченному алфавіті в відповідності з певним чином визначеними правилами. В НАМ використовується тільки одна елементарна дія – підстановка, яка визначається наступними чином. Нехай $A = \{a_1, a_2, \ldots, a_n\}$ – алфавіт. Якщо α і β – слова в алфавіті A, то вирази $\alpha \to \beta$ і $\alpha \mapsto \beta$ називаються *формулами підстановки в алфавіті A* (вважаємо, що символи $\to$ та $\mapsto$ не належать алфавіту A). При цьому формула $\alpha \to \beta$ називається *звичайною формулою підстановки*, а формула $\alpha \mapsto \beta$ – *заключною формулою підстановки*. Сама підстановка (як дія) задається формулою підстановки і застосовується до слів ω в алфавіті A. Суть операції $\alpha \to \beta$ ($\alpha \mapsto \beta$) полягає в відшуканні в слові ω найлівішого входження підслова α і заміні його на слово β. При цьому інші частини слова ω залишаються незмінними. Якщо ліва частина формули підстановки входить у слово ω, то кажуть, що дана *формула є застосовною до слова ω*. В іншому випадку кажемо, що *формула підстановки є незастосовною до слова ω*, і в цьому випадку підстановка не виконується. Якщо права частина формули підстановки – порожнє слово (яке в НАМ не познається ніяким символом), то підстановка $\alpha \to$ зводиться до видалення підслова α в слові ω. Якщо в лівій частині формули підстановки є порожнє слово, то підстановка $\to \beta$ полягає в дописуванні зліва до слова ω слова β. Зауважимо, що згідно означення

формула з порожньою лівою частиною застосовна до будь-якого слова.

5.1. Приклад. Нехай для слів в алфавіті $A = \{a, b, c\}$ задані наступні підстановки:

а) $ab \to ac$;
б) $abc \to ba$;
в) $ba \to$;
г) $b \mapsto c$.

Застосуємо кожну з них до слова $abbacba$.

Для застосування марківської підстановки до слова ω необхідно знайти найлівіше входження в ω лівої частини підстановки. У випадку а) слово ab тільки раз входить у слово $abbacba$, тому в результаті підстановки одержуємо слово $acbacba$. Оскільки слово abc не є підсловом слова $abbacba$, то підстановка з пункту б) є незастосовною до даного слова. У пунктах в) та г) ліві частини підстановок входять декілька раз у слово $abbacba$, а тому застосовуємо підстановки до першого входження їх у вхідне слово. Отримаємо слова $abcba$ і $acbacba$ відповідно.

5.2. Приклад. Застосувати кожну з підстановок із попереднього прикладу до слова $abbacba$ максимальну кількість разів.

Застосувавши один раз підстановку $ab \to ac$ до слова $abbacba$, одержимо слово $acbacba$, яке вже не містить підслова ab, а тому дана підстановка далі є незастосовною. Підстановка з пункту б) є незастосовною до слова $abbacba$. Після першого застосування підстановки $ba \to$ до слова $abbacba$ отримуємо слово $abcba$, до якого ще раз можна застосувати дану підстановку. Остаточно одержуємо слово abc, до якого вже незастосовна підстановка з пункту в). Оскільки підстановка $b \mapsto c$ є заключною, то вона застосовується тільки раз до вхідного слова і її результатом є слово $acbacba$.

Нормальним алгоритом Маркова називається пара (A, P), де A – алфавіт, а P – скінченна впорядкована послідовність формул

підстановки:

$$\begin{cases} \alpha_1 \to \beta_1 \\ \alpha_2 \to \beta_2 \\ \dots \\ \alpha_k \to \beta_k \end{cases},$$

яка називається *схемою підстановок*. При цьому вважається, що в P виділено деяку підмножину $P_f \subset P$ заключних формул підстановки.

Роботу НАМ можна описати наступним чином. Нехай задано деяке вхідне слово ω (у НАМ не важливо, де саме воно записано). Далі всі формули схеми підстановок проглядаються зверху вниз, обирається перша з формул, яка застосовна до слова ω, і виконується підстановка відповідно до знайденої формули підстановки. В результаті одержується нове слово ν. Потім ті ж самі дії виконуються з словом ν і т.д. При цьому зауважимо, що на кожному кроці формули в P завжди проглядаються зверху вниз, починаючи з першої формули! Якщо на черговому кроці була застосована звичайна формула підстановки, то робота НАМ продовжується, якщо ж – заключна формула, то після її застосування робота НАМ зупиняється і одержане слово називається *вихідним словом* або *результатом застосування НАМ до вхідного слова* ω. У випадку, якщо на деякому кроці кожна з формул схеми P є незастосовною до проміжного слова, то робота НАМ також зупиняється і вихідним словом вважається дане проміжне слово. В обох розглянутих випадках кажуть, що *НАМ застосовний до вхідного слова* ω.

5.3. Приклад. Застосувати кожен з НАМ

а) $\begin{cases} ab \to ba \\ ba \to \end{cases}$ б) $\begin{cases} ba \to \\ ab \mapsto ba \end{cases}$

в алфавіті $A = \{a, b\}$ до слова $abbbbaa$.

а) Спершу застосовуємо першу підстановку максимальну можливу кількість разів: $\underline{ab}bbbaa \Rightarrow b\underline{ab}bbaa \Rightarrow bb\underline{ab}baa \Rightarrow bbb\underline{ab}aa \Rightarrow bbbbaaa$. Потім застосуємо до слова $bbbbaaa$ підстановку $ba \to$: $bbbbaaa \Rightarrow bbbaa \Rightarrow bba \Rightarrow b$. В результаті одержуємо слово b.

Оскільки кожна з формул схеми а) є незастосовною до слова b, то НАМ застосовний до слова $abbbbaa$.

б) Застосуємо НАМ до слова $abbbbaa$: $abbb\underline{ba}a \Rightarrow abb\underline{ba} \Rightarrow \underline{ab}b \Rightarrow bab$. Оскільки ми використали заключну підстановку $ab \mapsto ba$, то робота НАМ зупиняється і алгоритм з пункту б) застосовний до слова $abbbbaa$.

Проте може трапитися так, що НАМ ніколи не зупиниться, тобто на кожному кроці до проміжного слова застосовна звичайна формула. В цьому випадку кажуть, що *НАМ є незастосовним до вхідного слова ω* або *зациклюється на вхідному слові ω*. *Область застосування НАМ* – це множина всіх слів, до яких застосовний даний НАМ.

5.4. Приклад. Визначимо область застосування НАМ відносно алфавіту $A = \{a, b\}$:

$$\begin{cases} a \to a \\ b \mapsto a \end{cases}$$

Перша формула застосовна до будь-якого вхідного слова, яке міститить хоча б одну букву a, причому вона не змінює даного слова. Тому на таких словах НАМ зациклюється. Якщо ж у вхідному слові немає букв a, але є хоча б одна буква b, тоді перша формула не буде застосовуватися, а відразу виконується друга підстановка і НАМ зупинить свою роботу. До порожнього вхідного слова кожна формула підстановоки є незастосовною, а тому НАМ до нього застосовний. Отже, область застосування даного НАМ – всі слова b^n $(n \geq 0)$.

5.5. Приклад. Побудуємо НАМ, який застосовний до всіх слів в алфавіті $A = \{a, b\}$, крім слів aa та ab.

З умови задачі випливає, що НАМ має зациклюватися на словах aa та ab. Однак алгоритм

$$\begin{cases} aa \to aa \\ ab \to ab \end{cases}$$

не є розв'язком даної задачі, оскільки він зациклюється не тільки на даних словах, але й на будь-яких інших словах, які містять підслово aa або ab. Зазвичай задачі такого типу розв'язують наступним

чином: початок і кінець вхідного слова ω відмічають спеціальними мітками (наприклад, $*\omega+$), а потім використовують формули підстановки, в лівій частині яких вказуються потрібні слова і ці мітки. Для зупинки алгоритму на інших словах можна використати, наприклад, формулу $* \mapsto$. Таким чином, шуканий НАМ має вигляд:

$$\begin{cases} +a \to a+ \\ +b \to b+ \\ *aa+ \to *aa+ \\ *ab+ \to *ab+ \\ * \mapsto \\ \mapsto *+ \end{cases}$$

Два НАМ над одним і тим же алфавітом A називаються *рівносильними*, якщо їх області застосувань співпадають і на однакових вхідних словах вони видають однакові результати.

5.6. Приклад. Визначити, чи є рівносильними наступні пари НАМ відносно алфавіту $A = \{a, b\}$:

а) $P_1 : \begin{cases} bb \to ab \end{cases}$ $P_2 : \begin{cases} bb \to ba \end{cases}$

б) $P_3 : \begin{cases} a \to ab \\ b \to aab \end{cases}$ $P_4 : \begin{cases} a \to b \\ b \to ba \end{cases}$

в) $P_5 : \begin{cases} b \mapsto ba \end{cases}$ $P_6 : \begin{cases} +a \to a+ \\ +b \mapsto ba \\ \to + \end{cases}$

Розглянемо схеми P_1 і P_2 НАМ з пункту а). У них одна і та ж область застосування – множина всіх слів в алфавіті $A = \{a, b\}$. Однак друга умова рівносильності (однакові результати на однакових вхідних словах) не виконується. Дійсно,

$$P_1 : \ a\underline{bb}b \Rightarrow aa\underline{bb} \Rightarrow aaab, \quad P_2 : \ a\underline{bb}b \Rightarrow abab.$$

Отже, НАМ з пункту а) не рівносильні.

У НАМ P_3 і P_4 області застосування співпадають і містять тільки єдине слово – порожнє. На непорожніх словах дані алгоритми зациклюються, причому по-різному. Однак така різна поведінка при зациклюванні не відіграє жодної ролі в означенні рівносильності алгоритмів. Важливо тільки, щоб у випадку зупинки алгоритми видавали однакові вихідні слова. А для пари з пункту б) ця умова виконується: на єдиному слові (порожньому), до якого вони застосовні, НАМ видають одну і ту ж відповідь – порожнє слово. Таким чином, дані алгоритми рівносильні.

В алгоритмів з пункту в), які замінюють перше входження букви b на ba, не співпадають області застосування: P_5 застосовний до всіх слів в алфавіті A, а P_6 зациклюється на словах, які не містять букви b. Отже, алгоритми з пункту в) не є рівносильними.

Рекомендована література : [**1**, с. 45–53], [**5**, с. 362–365], [**11**, с. 354–356], [**15**, с. 138–146], [**20**, с. 18–21].

Питання та вправи до параграфа 5.

5.1. Дайте означення формули підстановки в алфавіті A. Яка формула підстановки називається заключною?

5.2. Які НАМ називаються рівносильними?

5.3. Нормальний алгоритм Маркова $(\{a, b\}, P)$ задається схемою

$$P: \begin{cases} ba \to a \\ bb \to b \\ ab \to \\ \mapsto b \end{cases}.$$

Застосуйте його до слів $bbab$, $aaaab$, $bbaabba$.

5.4. В чому полягає робота НАМ (A, P), де $A = \{a, b\}$, а схема P має вигляд:

$$P: \begin{cases} a \to \\ b \to \\ \mapsto baba \end{cases}$$

5.5. Визначити область застосування НАМ з алфавітом $A = \{0, 1, 2\}$, якщо:

а) $P_1 : \begin{cases} 1 \to \\ 2 \to 2 \\ 0 \to \end{cases}$ б) $P_2 : \begin{cases} 1 \to \\ 22 \to 2 \\ 000 \to 00 \end{cases}$

в) $P_3 : \begin{cases} 1 \to 0 \\ 2 \to 1 \\ 00 \to \\ 0 \to 0 \end{cases}$ г) $P_4 : \begin{cases} 2 \to 1 \\ 1 \mapsto 0 \\ 00 \to \\ 0 \to 0 \end{cases}$

5.6. Знайти область застосування та визначити, в чому полягає робота НАМ $(\{a, b\}, P)$, де

$$P : \begin{cases} a \to aa \\ bb \to \\ b \mapsto ab \end{cases}$$

5.7. Побудувати НАМ в алфавіті $A = \{0, 1\}$, який застосовний тільки до порожнього слова та слова 1000.

5.8. Зі схеми

$$P : \begin{cases} 1 \to 0 \\ 01 \to 111 \\ 0 \to 1 \end{cases}$$

викреслити рівно одну формулу підстановки, щоб одержати НАМ, який застосовний до всіх слів в алфавіті $A = \{0, 1\}$.

5.9. Для кожної пари НАМ визначити, чи рівносильні вони відносно алфавіту $\{0, 1\}$:

а) $P_1 : \{1 \to 1$ $P_2 : \begin{cases} +1 \to 11 \\ 1 \to +1 \end{cases}$

б) $P_3 : \begin{cases} 10 \to 01 \\ 01 \to \end{cases}$ $P_4 : \begin{cases} 01 \to \\ 10 \to \end{cases}$

в) $P_5:\begin{cases}+1 \to 1\star\\ +0 \to 0+\\ \star 1 \mapsto\\ \star 0 \to 0+\\ \star \mapsto\\ \to +\end{cases}$ $P_6:\begin{cases}+0 \to 0+\\ +1 \mapsto\\ + \mapsto\\ 1 \to 1+\end{cases}$

5.10. З НАМ в алфавіті $\{0,1\}$, заданого схемою

$$P:\begin{cases}010 \to 001\\ 10 \to 01\\ 0101 \to 0011\\ 01 \to\end{cases}$$

викреслити якомога більше формул підстановки так, щоб одержаний алгоритм був рівносильний даному.

5.11. Побудувати машину Тюрінґа, яка рівносильна вказаному НАМ в алфавіті $\{0,1\}$:

а) $P:\begin{cases}+1 \to 0+\\ +0 \mapsto 0\\ \to +\end{cases}$ б) $P:\begin{cases}01 \mapsto 01\\ 10 \mapsto 10\\ \to\end{cases}$

5.12. Проаналізувати роботу нормального алгоритму Маркова, заданого над алфавітом $A = \{a_1, a_2, \ldots, a_m\}$ зі схемою

$$P:\begin{cases}** \to \#\\ \#a \to a\#, \text{ де } a \in A\\ \#* \to \#\\ \# \mapsto\\ *ab \to b*a, \text{ де } a,b \in A\\ \to *\end{cases}$$

§ 6. Синтез нормальних алгоритмів Маркова

У цьому параграфі на прикладах пояснюються основні прийоми побудови нормальних алгоритмів Маркова.

6.1. Приклад. Побудувати НАМ, який застосовний до всіх слів в алфавіті $A = \{a, b, c\}$ і видаляє із вхідного слова перше входження букви a (якщо таке є), а букви b замінює на c.

Перш за все зауважимо, що в НАМ, на відміну від машини Тюрінґа, легко реалізуються вставки, заміни та видалення букв. Наприклад, заміна букви b на букву c здійснюється за допомогою формули підстановки $b \to c$, а видалення букви a реалізується формулою $a \to$. При цьому проміжне слово розтягується або стискається автоматично. Шуканий алгоритм спершу має всі букви b замінювати на c, тому першою формулою має бути $b \to c$. Якщо букв b у слові вже не залишиться, то потрібно видалити перше входження букви a, використавши заключну підстановку $a \mapsto$. Таким чином, НАМ матиме вигляд:

$$\begin{cases} b \to c \\ a \mapsto \end{cases}$$

6.2. Приклад. Побудувати НАМ, який „сортує” непорожнє слово в алфавіті $A = \{0, 1\}$ (послідовність з цифр 0 і 1) в порядку незростання.

Дана задача розв'язується з допомогою НАМ, який містить єдину формулу підстановки $01 \to 10$. Дана формула міняє місцями 0 з кожною 1 доти, доки у слові праворуч від хоча б однієї цифри 0 є цифра 1. Формула стає незастосовною, коли праворуч від кожного 0 нема жодної 1, тобто коли вхідне слово відсортоване по незростанню. Наприклад,

$$\underline{01}10 \to 1\underline{01}0 \to 1100.$$

6.3. Приклад. Побудувати НАМ, який застосовний до всіх слів в алфавіті $A = \{a, b\}$ і видаляє останню букву непорожнього вхідного слова.

Для розв'язання задачі потрібно помітити останню букву, поставивши після неї новий спецзнак *. Але як помістити цей знак в

кінець слова? Робиться це наступним чином: спочатку дописуємо ліворуч до вхідного слова *, а потім „переганяємо" її в його кінець. Неважко помітити, що „перескакування" зірочки через букву – це заміна пари $*x$ на пару $x*$, яка здійснюється з допомогою формули підстановки $*x \to x*$. Після того як * опиниться вкінці слова, потрібно видалити останню букву і * та зупинити роботу алгоритму. Таким чином, одержуємо наступний НАМ:

$$\begin{cases} \to * \\ *a \to a* \\ *b \to b* \\ a* \mapsto \\ b* \mapsto \end{cases}$$

Проаналізуємо його роботу на слові abb:

$$abb \to *abb \to **abb \to ***abb \to \dots$$

Бачимо, що цей алгоритм постійно дописує зліва зірочки. Чому? Нагадаємо, що формула підстановки з порожньою лівою частиною застосовна завжди, тому перша формула буде застосовуватися нескінченно, блокуючи доступ до наступних формул. Звідси випливає дуже важливе правило: *якщо в НАМ є формула з порожньою лівою частиною, то вона має міститися вкінці НАМ*. З урахуванням цього алгоритм перепишеться так:

$$\begin{cases} *a \to a* \\ *b \to b* \\ a* \mapsto \\ b* \mapsto \\ \to * \end{cases}$$

Однак це ще не все: алгоритм зациклюється на порожньому вхідному слові, бо остання формула завжди буде застосовна. В чому причина помилки? Річ у тім, що ми ввели знак * для того, щоб помітити останню букву вхідного слова, а потім видалити * і цю букву. Щоб врахувати випадок порожнього слова, треба перед останньою формулою записати ще одну формулу, яка видаляє „одиноку" зірочку і зупиняє алгоритм. Таким чином, остаточно НАМ матиме вигляд:

$$\begin{cases} *a \to a* \\ *b \to b* \\ a* \mapsto \\ b* \mapsto \\ * \mapsto \\ \to * \end{cases}$$

6.4. Приклад. Побудувати НАМ над алфавітом $A = \{a, b\}$, який подвоює кожне входження букви b та дописує вкінці слова суфікс $baba$.

На перший погляд, щоб подвоїти кожну букву b, досить застосувати формулу $b \to bb$. Щоб переконатися, що це не так, розглянемо приклад:

$$\underline{b}ab \to \underline{b}bab \to \underline{b}bbab \to \dots$$

Помилка тут у тому, що після заміни першого входження букви b на bb ми не можемо відрізнити вже замінені букви b від тих, які ще не мінялися. Для вирішення цієї проблеми можна помітити зліва спецзнаком * ту букву b, яку потрібно в даний момент замінити, а після того як така заміна вже здійснена, спецзнак треба перемістити до наступної букви.

Таким чином, спершу слід розмістити ліворуч від вхідного слова спецзнак *, а потім „перескакувати" через кожну наступну букву, не змінюючи її, якщо ця буква a, або подвоюючи, якщо нею є b. Якщо праворуч від * вже не виявиться жодної букви, то спецзнак потрібно замінити на слово $baba$.

Отже, отримуємо наступний НАМ:

$$\begin{cases} *a \to a* \\ *b \to bb* \\ * \mapsto baba \\ \to * \end{cases}$$

Перевіримо його на вхідному слові bab:

$$bab \to *bab \to bb*ab \to bba*b \to bbabb* \to bbabbbaba.$$

6.5. Приклад. Побудувати НАМ над алфавітом $A = \{a, b\}$, який видаляє останнє входження букви b (якщо таке є).

Для того щоб помістити * поруч з останнім входженням букви b, можна спершу перемістити * в кінець слова, а потім перенести * справа наліво через букви a до найближчої букви b. При цьому потрібно врахувати, що вхідне слово може не містити букви b: якщо зірочка знову опиниться на початку слова, то її потрібно видалити і зупинитися. Реалізуємо дану ідею з допомогою наступного НАМ:

$$\begin{cases} *a \to a* \\ *b \to b* \\ a* \to *a \\ b* \mapsto \\ * \mapsto \\ \to * \end{cases}$$

Перевіримо роботу даного алгоритму на вхідному слові aba:

$$aba \to *aba \Rightarrow ab\underline{a*} \to ab\underline{*a} \to ab\underline{a*} \to \dots$$

Як бачимо, замість того, щоб рухатися справа наліво до найближчої букви b, зірочка почала „кружляти" навколо останньої букви слова. Чому? Річ у тім, що перші дві формули, які переміщують * праворуч, заважають третій формулі. Зауважимо, що перестановка цих формул не приведе до потрібного результату, бо тоді * почне „бігати" навколо першої букви вхідного слова. Помилка полягає в тому, що ми використовуємо спецзнак * як для руху ліворуч, так і для руху праворуч. Щоб виправити цю помилку, потрібно просто ввести ще один спецзнак, наприклад #, поділивши між цими спецзнаками обов'язки: нехай * переміщується праворуч, а # – ліворуч. З'явитися ж спецзнак # має тоді, коли * дійде до кінця слова, тобто коли справа від * не виявиться інших букв. Остаточно отримаємо такий НАМ:

$$\begin{cases} *a \to a* \\ *b \to b* \\ * \to \# \\ a\# \to \#a \\ b\# \mapsto \\ \# \mapsto \\ \to * \end{cases}$$

6.6. Приклад. Побудувати НАМ, який застосовний до всіх слів в алфавіті $A = \{a, b, c\}$ і визначає кількість різних букв у вхідному слові. Відповідь одержати в унарній системі числення (наприклад: $aabbab \to ||$).

Для розв'язання цієї задачі спочатку відсортуємо вхідне слово таким чином, щоб спершу були всі букви a, потім – b, а вкінці – c. Далі видалимо зайві букви так, щоб залишилась максимальна кількість попарно різних букв. Для цього скористаємось формулами підстановки виду $xx \to x$. Насамкінець замінимо всі букви на палички. Шуканий НАМ матиме вигляд:

$$\begin{cases} ba \to ab \\ ca \to ac \\ cb \to bc \\ aa \to a \\ bb \to b \\ cc \to c \\ a \to | \\ b \to | \\ c \to | \end{cases}$$

Наприклад,

$$aab\underline{ba}b \to aa\underline{ba}bb \to aaabbb \to aabbb \to abbb \to abb \to ab \to |b \to ||.$$

6.7. Приклад. Побудувати НАМ над алфавітом $A = \{0, 1, 2\}$, який дописує праворуч до непорожнього вхідного слова (трійкового числа) знак „= ” і стільки паличок, скільки цифр містить трійкове число.

Розв'яжемо дану задачу наступними чином. Спершу за кожною цифрою вхідного слова вставимо паличку $|$. Для цього допишемо зліва до вхідного слова спецзнак $*$, а потім перенесемо його через кожну цифру так, щоб зліва від нього залишалася ця цифра і відповідна їй паличка. Наприклад,

$$10 \to *10 \to 1|*0 \to 1|0|*$$

В одержаному слові переставимо цифри і палички так, щоб зліва опинилися всі букви, а справа – палички, зберігаючи при цьому вихідний взаємний порядок як букв, так і паличок:

$$1|0|* \to 10||*$$

Вкінці слова замінимо $*$ на новий спецзнак $=$ і перемістимо його ліворуч до першої цифри:

$$10||* \to 10|| = \to 10| = | \to 10 = ||$$

Всі вказані дії описуються наступним алгоритмом Маркова:

$$\begin{cases} *n \to n|*, \ n \in A \\ |n \to n|, \ n \in A \\ * \to = \\ | = \to = | \\ n = \mapsto n =, \ n \in A \\ \to * \end{cases}$$

Рекомендована література : [**11**, с. 354–356], [**15**, с. 146–180], [**20**, с. 21–32], [**22**, с. 178–189].

Питання та вправи до параграфа 6.

У задачах 6.1-6.21 побудувати НАМ, який застосовний до всіх слів в алфавіті A і виконує наступну роботу:

6.1. $A = \{$Н, П, У$\}$. Дописує справа до вхідного слова ω слово ПНУ ($\omega \to \omega$ПНУ).

6.2. $A = \{a, b\}$. Залишає у слові тільки першу букву (порожнє слово не міняє).

6.3. $A = \{a, b\}$. Залишає у слові тільки останню букву (порожнє слово не міняє).

6.4. $A = \{a, b, c\}$. За першою буквою непорожнього вхідного слова вставляє букву c.

6.5. $A = \{a, b\}$. Якщо у вхідному слові є принаймні дві букви, то міняє місцями перші дві букви.

6.6. $A = \{a, b\}$. Якщо вхідне слово містить більше букв a, ніж букв b, то видає вихідне слово з однієї букви a; якщо однакова кількість букв a і b, то видає порожнє слово; інакше – слово b.

6.7. $A = \{a, b\}$. У вхідному слові всі букви a замінює на b, а всі (попередні) букви b – на a.

6.8. $A = \{a, b, c\}$. Подвоює кожну букву у вхідному слові.

6.9. $A = \{a, b\}$. Дописує праворуч до вхідного слова стільки паличок, зі скількох підряд записаних букв b починається це слово.

6.10. $A = \{a, b\}$. Зі всіх входжень букви a у вхідне слово залишає тільки останнє входження, якщо таке є.

6.11. $A = \{a, b\}$. Якщо вхідне слово містить одночасно букви a і b, то замінює його на порожнє слово.

6.12. $A = \{a, b\}$. Якщо вхідне слово не є словом $aabba$, то замінює його на порожнє слово.

6.13. $A = \{a, b\}$. Визначає, чи входить перша буква непорожнього вхідного слова ще раз у це слово. Відповідь: слово a, якщо входить, або порожнє слово в протилежному випадку.

6.14. $A = \{a, b\}$. Переносить останню букву непорожнього вхідного слова на його початок.

6.15. $A = \{a, b\}$. В непорожньому вхідному слові переставляє місцями першу та останню букви.

6.16. $A = \{a, b\}$. Якщо в непорожньому вхідному слові співпадають перша та остання букви, то видаляє ці букви, в іншому випадку слово не змінює.

6.17. $A = \{a, b, c\}$. Якщо вхідне слово не містить букви c, то замінює всі букви a на b. В іншому випадку видає слово з однієї букви c.

6.18. $A = \{a, b\}$. Подвоює вхідне слово (дописує справа його копію).

6.19. $A = \{a, b\}$. Обертає вхідне слово (наприклад, $aab \to baa$).

6.20. $A = \{a, b\}$. Визначає, чи є слово поліндромом. Відповідь: слово a, якщо є, або порожнє слово в протилежному випадку.

6.21. $A = \{a, b\}$. Нехай вхідне слово має непарну довжину. Видаляє з нього середню букву.

§ 7. Нормально обчислювані функції

В даному параграфі розглядаються функції, для яких існує нормальний алгоритм Маркова, що їх обчислює. Зауважимо, що при аналізі деяких вхідних слів алгоритм Маркова може ніколи не зупинитися (зациклюється), і тому функція може бути не визначена для деяких вхідних слів-аргументів. У загальному випадку кажемо, що функція $f : (X^*)^n \to Y^*$ є *частково визначеною функцією*, якщо вона не визначена для деяких наборів аргументів $(x_1, \ldots, x_n) \in (X^*)^n$, тобто коли область визначення f є підмножиною $(X^*)^n$. Якщо область визначення f співпадає з $(X^*)^n$, то кажемо, що *функція f всюди визначена на $(X^*)^n$*. Якщо функція f не визначена на наборі $(x_1, \ldots, x_n)$, то писатимемо $f(x_1, \ldots, x_n) = \uparrow$.

Частково визначена функція $f : (X^*)^n \to Y^*$ називається *нормально обчислюваною*, якщо існує такий нормальний алгоритм Маркова, що для кожного набору $(x_1, \ldots, x_n) \in (X^*)^n$, на якому f визначена, НАМ перетворює вхідне слово $x_1 * x_2 * \ldots * x_n$ у вихідне слово y, де $y = f(x_1, \ldots, x_n) \in Y^*$, а $*$ – спеціальний допоміжний символ, який розділяє вхідні аргументи; і на кожному наборі $(x_1, \ldots, x_n) \in (X^*)^n$, для якого функція не визначена, НАМ зациклюється.

Надалі найчастіше розглядатимемо числові функції $f : (\mathbb{N}_0)^n \to \mathbb{N}_0$, де $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.

7.1. Приклад. Покажемо, що функція $f : \mathbb{N}_0 \to \mathbb{N}_0$, $f(n) = 0$ обчислюється з допомогою деякого НАМ.

Вважатимемо, що числа записані в двійковій системі числення. Для побудови НАМ, який обчислює дану функцію, достатньо в його схему підстановок включити формули для видалення всіх букв (цифр 0 і 1) вхідного слова і заключну формулу для заміни порожнього слова нулем. Таким чином, схема НАМ матиме вигляд:

$$\begin{cases} 0 \to \\ 1 \to \\ \mapsto 0 \end{cases}$$

7.2. Приклад. Покажемо, що функція $f : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$, $f(n, m) = |n - m|$ є нормально обчислюваною (вважаємо, що невід'ємні цілі числа записані в унарній системі числення).

Нагадаємо, що

$$|n - m| = \begin{cases} n - m, \ n \geq m \\ m - n, \ n < m \end{cases}$$

На початку роботи вхідне слово має вигляд $\underbrace{||\ldots|}_{n} * \underbrace{||\ldots|}_{m}$. Оскільки $|n - m| = |(n - 1) - (m - 1)|$, то для розв'язання задачі досить одночасно видаляти по одній паличці зліва і справа доти, доки в деякій частині не залишиться жодної палички. Отже, НАМ задається схемою:

$$\begin{cases} |*| \to * \\ * \mapsto \end{cases}$$

Наприклад, $||*|||| \to |*||| \to *|| \to ||$ і $f(2, 4) = |2 - 4| = 2$.

7.3. Приклад. Покажемо, що функція $f : \mathbb{N}_0 \to \mathbb{N}_0$, $f(n) = n \bmod 4$, яка обчислює остачу від ділення натурального числа на 4, є нормально обчислюваною (вважаємо, що невід'ємні цілі числа записані в унарній системі числення).

Оскільки остачі від ділення чисел n і $n - 4$ на 4 однакові, то для знаходження остачі досить від числа віднімати 4 доти, доки

не отримаємо невід'ємне число менше 4 – воно і буде шуканою остачею. Таким чином, схема НАМ буде містити єдину формулу підстановки:

$$\Big\{|||| \to$$

Наприклад, $||||||||| \to ||||| \to |$ і $f(9) = 1$.

7.4. Приклад. Покажемо, що функція $f : \mathbb{N}_0 \to \mathbb{N}_0$, $f(n) = [n/4]$, є нормально обчислюваною.

Для знаходження цілої частини від ділення числа n на 4, досить спершу підрахувати, скільки четвірок паличок містить це унарне число, а потім витерти палички, які відповідають остачі. Дані міркування реалізуються такою схемою НАМ:

$$\begin{cases} *|||| \to |* \\ *| \to * \\ * \mapsto \\ \to * \end{cases}$$

Наприклад, $f(9) = 1$ і $||||||||| \to *|||||||| \to |*|||| \to ||*| \to ||* \to ||$.

7.5. Приклад. Комбінуючи попередні два приклади, покажемо, що функція $f : \mathbb{N}_0 \to \mathbb{N}_0 \times \mathbb{N}_0$, $f(n) = ([n/4],\ n \mod 4)$, є нормально обчислюваною.

Легко бачити, що дана функція обчислюється НАМ з схемою:

$$\begin{cases} *|||| \to |* \\ * \mapsto * \\ \to * \end{cases}$$

7.6. Приклад. Побудуємо НАМ, який обчислює функцію $f : \mathbb{N} \to \mathbb{N}$, $f(n) = n + 2$ (вважаємо, що натуральні числа задані в трійковій системі числення).

Для розв'язання задачі спершу введемо спецзнак * і перемістимо його вкінець слова, щоб помітити його останню букву (трійкову цифру). Далі додамо 2 за правилами трійкової арифметики. Якщо останньою цифрою є 0, то замінимо її на 2 і зупинимо роботу, інакше міняємо 1 на 0, 2 на 1 і переміщуємося ліворуч для додавання 1 до попередньої цифри з допомогою нового спецзнаку # (при цьому

враховуємо, що цією цифрою може бути 2, яку треба замінити на 0 і перемістити ліворуч для додавання 1).

Всі розглянуті дії описуються таким НАМ:

$$\begin{cases} *0 \to 0* \\ *1 \to 1* \\ *2 \to 2* \\ 0* \mapsto 2 \\ 1* \to \#0 \\ 2* \to \#1 \\ 0\# \mapsto 1 \\ 1\# \mapsto 2 \\ 2\# \to \#0 \\ \# \mapsto 1 \\ \to * \end{cases}$$

7.7. Приклад. Нехай вхідний алфавіт $A = \{0, 1, 2, 3\}$. Вважаючи вхідне слово записом числа в четвірковій системі числення, перевести це число в двійкову систему числення.

Як відомо, для переведення числа з четвіркової системи числення в двійкову, потрібно кожну четвіркову цифру замінити на пару відповідних їй двійкових цифр: $0 \to 00$, $1 \to 01$, $2 \to 10$, $3 \to 11$, див. § **??** першого розділу. Користуючись цим фактом, схему НАМ запишемо так:

$$\begin{cases} *0 \to 00* \\ *1 \to 01* \\ *2 \to 10* \\ *3 \mapsto 11* \\ * \mapsto \\ \to * \end{cases}$$

Нехай задано два нормальні алгоритми Маркова НАМ_1 і НАМ_2 в одному і тому ж алфавіті A, які обчислюють словесні функції $f_1 : A^* \to A^*$ та $f_2 : A^* \to A^*$ відповідно. Нас цікавитимуть НАМ,

які обчислюють їх композицію $f_2 \circ f_1$. Нагадаємо, що композицією функцій f_1 та f_2 називають таку функцію $f = f_2 \circ f_1$, що для кожного x з області визначення функції f_1 значення $f(x) = f_2(f_1(x))$. При цьому вимагається, щоб функція f_2 була визначена на значенні $f_1(x)$. Вихідне слово НАМ_1 можна подати на вхід НАМ_2. Таке послідовне виконання алгоритмів називається композицією НАМ_1 та НАМ_2 і позначається $\text{НАМ}_2(\text{НАМ}_1)$. Якщо будь-який з алгоритмів зациклюється, то зациклюватися має і їх композиція.

Доведена наступна теорема: *для будь-яких двох нормальних алгоритмів в алфавіті A, які обчислюють функції $f_1 : A^* \to A^*$ та $f_2 : A^* \to A^*$ відповідно, існує НАМ, який обчислює композицію $f_2 \circ f_1$*, див. [**15**]

7.8. Приклад. Побудуємо нормальний алгоритм, який є композицією наступних двох алгоритмів НАМ_1 і НАМ_2 з схемами P_1 і P_2 відповідно відносно алфавіту $\{a, b\}$:

$$P_1 : \begin{cases} *a \to a* \\ *b \to ab* \\ * \mapsto \\ \to * \end{cases} \qquad P_2 : \begin{cases} b \to \end{cases}$$

Спершу зауважимо, що в загальному випадку не можна будувати композицію простим виписуванням одна за одною формул підстановки з НАМ_1 і НАМ_2. Наприклад, якщо спочатку виписати формули з НАМ_1, а потім з НАМ_2, то отримаємо алгоритм НАМ_{12}, а якщо спершу виписати формули з НАМ_2, а потім з НАМ_1, то отримаємо алгоритм НАМ_{21}, які задані схемами:

$$P_{12} : \begin{cases} *a \to a* \\ *b \to ab* \\ * \mapsto \\ \to * \\ b \to \end{cases} \qquad P_{21} : \begin{cases} b \to \\ *a \to a* \\ *b \to ab* \\ * \mapsto \\ \to * \end{cases}$$

Але ні НАМ_{12}, ні НАМ_{21} не є композицією алгоритмів НАМ_1 і НАМ_2. Наприклад, для вхідного слова aaa маємо:

$$\text{НАМ}_{12}: \ aba \to *aba \to a*ba \to aab*a \to aaba* \to aaba$$

$$\text{НАМ}_{21}: \ aba \to aa \to *aa \to a*a \to aa* \to aa,$$

тоді як $\text{НАМ}_2(\text{НАМ}_1(aba))=\text{НАМ}_2(aaba) = aaa$.

Зауважимо, що існує загальний метод побудови композиції НАМ, див. [**15**, с. 198]. Однак цей спосіб досить громіздкий і для простіших НАМ краще використовувати інший підхід: перш за все треба зрозуміти, яку задачу розв'язує кожен з алгоритмів НАМ_1 і НАМ_2, а потім послідовність цих задач об'єднати в спільну задачу і побудувати алгоритм для цієї загальної задачі.

В нашому прикладі алгоритм НАМ_1 замінює кожну букву b словом ab, а потім алгоритм НАМ_2 видаляє букви b. Зрозуміло, що послідовне застосування спершу НАМ_1, а потім НАМ_2 замінює кожну букву b вхідного слова буквою a. Шукана композиція НАМ_1 і НАМ_2 задається простою схемою НАМ:

$$\begin{cases} b \to a \end{cases}$$

Ми показали, що клас нормально обчислюваних функцій є досить широким. Поняття нормально обчислюваної функції – одне з основних понять теорії алгоритмів. Зазначимо, що з одного боку, кожна нормально обчислювана функція є обчислюваною за певною процедурою, яка відповідає інтуїтивному уявленню алгоритму, а з іншого – які б досі не будувалися класи точно визначених алгоритмів, завжди з'ясовувалося, що числові функції, які обчислюються за алгоритмами цих класів, є нормально обчислюваними. Тому загальноприйнятою є така наукова гіпотеза:

Теза Маркова-Черча. *Будь-який алгоритм можна реалізувати з допомогою НАМ.*

У формулювання цієї тези входить інтуїтивне поняття алгоритму, а тому її не можна ні довести, ні спростувати в загальноматематичному значенні. Справедливість гіпотези підтверджується

практикою: всі відомі алгоритми, які були придумані впродовж багатьох тисячоліть історії математики, можуть бути задані з допомогою НАМ. Проте справедливою є наступна теорема, доведення якої читач може знайти в [**11**]:

7.9. Теорема. *Для функції f наступні умови рівносильні:*

- *функція f є обчислюваною за Тюрінґом;*
- *функція f частково рекурсивна;*
- *функція f є нормально обчислюваною.*

Рекомендована література : [**11**, с. 356–361], [**15**, с. 180–186, 198–219].

Питання та вправи до параграфа 7.

7.1. Що ви розумієте під композицією НАМ?

7.2. Побудувати НАМ, який обчислює числову функцію $f : \mathbb{N}_0 \to \mathbb{N}_0$, $f(n) = n \mod 2$, якщо аргументи задані:

а) в унарній системі числення;
б) в двійковій системі числення;
в) в трійковій системі числення.

7.3. Побудувати НАМ, який обчислює числову функцію $f : \mathbb{N}_0 \to \mathbb{N}_0$, аргументи якої задані в двійковій системі числення.

а) $f(n) = 4_{10} \cdot n$;
б) $f(n) = [\frac{n}{3}]$;
в) $f(n) = [\frac{n+3}{4}] + 2$.

7.4. Побудувати НАМ, який обчислює числову функцію $f : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$, аргументи якої задані в унарній системі числення.

а) $f(n, m) = n + m$;
б) $f(n, m) = \max\{n, m\}$;
в) $f(n, m) = \min\{n, m\}$.

7.5. Показати, що наступні частково визначені числові функції $f : \mathbb{N}_0 \to \mathbb{N}_0$, аргументи яких задані в четвірковій системі числення, є нормально обчислювані:

а) $f(n) = n + 1$;
б) $f(n) = n + 2$;
в) $f(n) = n - 2$.

7.6. Показати, що наступний НАМ обчислює функцію $f : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$, $f(n,m) =$ НСД(n,m), аргументи якої задані в унарній системі числення:

$$\begin{cases} |a \to a| \\ |*| \to a* \\ |* \to *b \\ b \to | \\ a \to c \\ c \to | \\ * \to \end{cases}$$

7.7. $A = \{0,1,2,a\}$. З'ясувати, чи є вхідне слово записом числа в трійковій системі числення. Відповідь: 1 – якщо є, 0 – в протилежному випадку.

7.8. $A = \{0,1,2\}$. Вважаючи вхідне слово записом числа в трійковій системі числення, видалити з нього всі незначущі нулі.

7.9. $A = \{0,1\}$. Вважаючи непорожнє слово записом двійкового числа, визначити, чи є воно степенем двійки. Відповідь: 1 – якщо є, 0 – в протилежному випадку.

7.10. $A = \{0,1\}$. Вважаючи непорожнє слово записом двійкового числа, перевести його в четвіркову систему числення. (Зауваження: врахувати, що в двійковому числі може бути непарна кількість цифр.)

7.11. $A = \{|\}$. Перевести число з унарної системи числення в трійкову. (Рекомендація: можна у циклі видаляти з унарного числа по паличці і кожен раз додавати 1 до трійкового числа, яке на початку покласти рівним 0.)

7.12. Для кожної пари НАМ побудувати їх композиції відносно алфавіту $\{0,1\}$:

а) $P_1 : \begin{cases} 01 \to 10 \end{cases}$ $\qquad$ $P_2 : \begin{cases} 1 \to 0 \end{cases}$

б) $P_3:\begin{cases}*0 \to * \\ *1 \to 1* \\ * \mapsto \\ \to *\end{cases}$ $P_4:\begin{cases}1 \to 0\end{cases}$

в) $P_5:\begin{cases}*0 \to 1* \\ *1 \to 0* \\ * \mapsto \\ \to *\end{cases}$ $P_6:\begin{cases}01 \to\end{cases}$

7.13. Чи є НАМ зі схемою P_3 композицією нормальних алгоритмів зі схемами P_1 і P_2 відносно алфавіту $\{0,1,2\}$?

$P_1:\begin{cases}01 \to 10\end{cases}$ $P_2:\begin{cases}02 \to 20\end{cases}$

$P_3:\begin{cases}01 \to 10 \\ 02 \to 20\end{cases}$

§ 8. Складність алгоритмів

В загальній теорії алгоритмів вивчається лише теоретична можливість розв'язання задач. При розгляді конкретних задач не звертається увага на ресурси часу і пам'яті для відповідних алгоритмів-розв'язків. Але теоретична можливість розв'язання задачі не гарантує її практичної реалізації.

Введемо необхідні означення, відштовхуючись від машин Тюрінґа. Нехай машина Тюрінґа обчислює словесну функцію $f(\omega)$. Визначимо функцію $t_T(\omega)$, значення якої для слова ω дорівнює кількості тактів машини Тюрінґа T, які виконуються при обчисленні $f(\omega)$, якщо $f(\omega)$ визначене. Якщо $f(\omega)$ не визначене, то значення функції $t_T(\omega)$ теж вважається не визначеним. Функція $t_T(\omega)$ називається *часовою складністю машини Тюрінґа T*.

Активною зоною при роботі машини Тюрінґа на слові ω називається множина всіх комірок стрічки, які містять непорожні букви або відвідувались головкою машини Тюрінґа в процесі обчислення. Введемо функцію $s_T(\omega)$, значення якої дорівнює довжині активної

зони при роботі машини Тюрінґа T на слові ω, якщо $f(\omega)$ визначене. В іншому випадку значення функції $s_T(\omega)$ не визначене. Функція $s_T(\omega)$ називається *ємнісною складністю машини Тюрінґа T*.

Введені функції $t_T(\omega)$ і $s_T(\omega)$ є словесними. Зручно ввести для розгляду функції натурального аргументу, поклавши:

$$t_T(n) = \max_{|\omega|=n}\{t_T(\omega)\} \quad \text{і} \quad s_T(n) = \max_{|\omega|=n}\{s_T(\omega)\}.$$

Ці функції теж називаються функціями часової і ємнісної складності (в найгіршому випадку) машини Тюрінґа T.

Визначимо функції часової та ємнісної складності для алгоритмів Маркова. Нехай алгоритм Маркова M обчислює словесну функцію $f(\omega)$. Визначимо функцію $t_M(\omega)$, значення якої для вхідного слова ω дорівнює кількості підстановок, які виконуються в процесі обчислення вихідного слова $f(\omega)$ з вхідного слова ω, якщо алгоритм застосовний до вхідного слова ω. Якщо ж алгоритм зациклюється на вхідному слові ω, то значення функції $t_M(\omega)$ вважається не визначеним. Функція $t_M(\omega)$ називається *часовою складністю алгоритму Маркова M*.

Введемо функцію $s_M(\omega)$, значення якої дорівнює максимальній з довжин слів, які виникають при обчисленні з вхідного слова ω вихідного слова $f(\omega)$, якщо $f(\omega)$ визначене. Якщо $f(\omega)$ не визначене, то значення функції $s_M(\omega)$ теж не визначене. Функція $s_M(\omega)$ називається *ємнісною складністю алгоритму Маркова M*.

Аналогічно, як і для машини Тюрінґа, зручно ввести функції натурального аргументу, поклавши:

$$t_M(n) = \max_{|\omega|=n}\{t_M(\omega)\} \quad \text{і} \quad s_M(n) = \max_{|\omega|=n}\{s_M(\omega)\}.$$

Ці функції теж називаються функціями часової і ємнісної складності (в найгіршому випадку) алгоритму Маркова M.

Гранична поведінка функцій t_T і t_M (відповідно s_T і s_M) при збільшенні розміру n задачі називається *асимптотичною часовою* (відповідно *ємнісною*) *складністю*. Для конкретних задач розглядаються, як правило, асимптотичні функції складності.

Нехай f і g – дві функції натурального аргументу. Кажуть, що $f(n) = O(g(n))$ (читається: „еф від ен є о велике від же від ен"), якщо існує така константа $c > 0$, $c \in \mathbb{R}$ і таке число $n_0 \in \mathbb{N}_0$, що $0 \le f(n) \le cg(n)$ для всіх $n \ge n_0$. Запис $f(n) = \Theta(g(n))$ означає,

що існують такі константи $c_1, c_2 > 0$, $c_1, c_2 \in \mathbb{R}$ і таке $n_0 \in \mathbb{N}_0$, що $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ для всіх $n \geq n_0$.

Кажуть, що машина Тюрінґа (алгоритм Маркова) розв'язує задачу за *поліноміальний час*, якщо $t_T(n) = O(p(n))$ ($t_M(n) = O(p(n))$) для деякого многочлена p. В протилежному випадку кажуть, що машина Тюрінґа (алгоритм Маркова) розв'язує задачу за *експоненціальний час*.

Часова складність алгоритму відображає потрібні для його роботи витрати часу. Аналогічно, як для машини Тюрінґа і алгоритмів Маркова, для будь-яких алгоритмічних моделей часова складність визначається як функція, яка кожній послідовності вхідних даних довжини n ставить у відповідність максимальний (за всіма індивідуальними задачами довжиною n) час $t(n)$, який витрачає алгоритм для розв'язання індивідуальних задач із цією довжиною. Про конкретну задачу кажуть, що вона розв'язується за поліноміальний час, якщо існує алгоритм (наприклад, машина Тюрінґа чи алгоритм Маркова), який розв'язує цю задачу за поліноміальний час. Через $\mathbb{P}$ позначається клас всіх задач, які розв'язуються за поліноміальний час. В іншому випадку кажуть, що задача розв'язується за експоненціальний час. Задачу називають *важкорозв'язною*, якщо немає поліноміального алгоритму для її розв'язання.

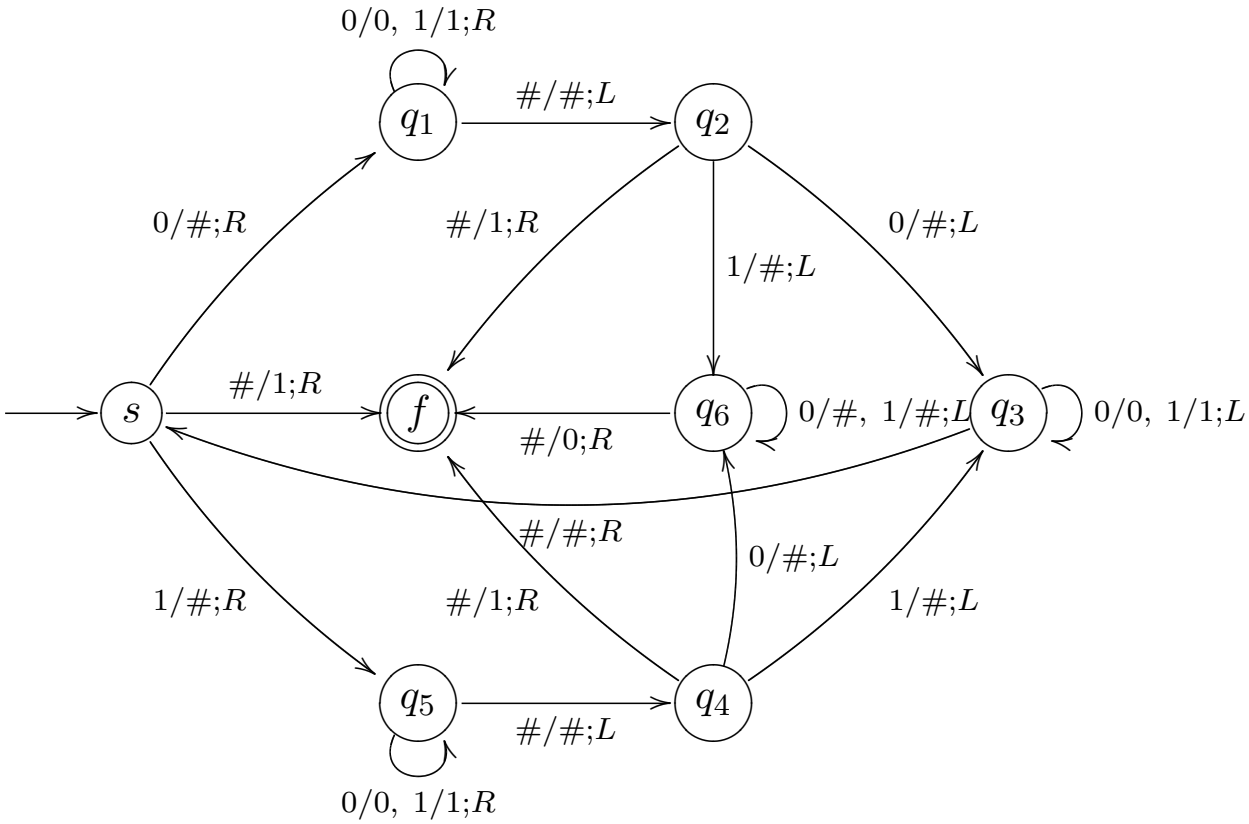
Вважається, що поліноміальні алгоритми відповідають швидким, ефективним на практиці алгоритмам, а поліноміально розв'язні задачі відповідають легким задачам, які можуть бути розв'язаними за прийнятний час на входах довжини, що має практичний інтерес. Часто поліноміальний алгоритм справді задовольняє всі практичні потреби. В гіршому ж випадку його можна вважати швидким лише асимптотично, а на відносно невеликих входах алгоритм може працювати довго. Слід підкреслити, що клас поліноміально розв'язних задач не залежить від того, яка саме з багатьох можливих формалізацій алгоритму вибрана. Цей факт можна строго довести як теорему для будь-яких означень алгоритму.

Слід зазначити, що є експоненціальні алгоритми, які добре зарекомендували себе на практиці. Річ у тім, що часову складність означено як міру поведінки алгоритму в найгіршому випадку. Насправді може виявитись, що для розв'язання більшості конкретних

задач потрібно значно менше часу, і це дійсно так для деяких добре відомих алгоритмів. Наприклад, симплекс-метод для розв'язування задач лінійного програмування має експоненціальну часову складність, але дуже добре працює на практиці.

8.1. Приклад. Побудувати машину Тюрінґа, яка в алфавіті $A = \{0, 1\}$ для будь-якого слова $P = p_1 p_2 \dots p_n$ визначає, чи є воно симетричним, тобто $P = p_1 p_2 \dots p_n = p_n p_{n-1} \dots p_1 = P'$. Відповідь: слово 1, якщо слово P симетричне, 0 – в протилежному випадку. Оцінити часову та ємнісну складність побудованої машини Тюрінґа.

Граф шуканої машини Тюрінґа має вигляд:



Робота машини Тюрінґа здійснюється циклами. Впродовж першого циклу машина перевіряє, чи виконується рівність $p_1 = p_n$. Для цього в стані s запам'ятовується буква p_1 і головка машини Тюрінґа в станах q_1 або q_5 зміщується праворуч доти, доки не знаходить букву p_n і порівнює її з p_1 в станах q_2 або q_4. Якщо ці букви різні, то машина переходить в стан q_6, видаляє слово на стрічці і друкує букву 0, в протилежному випадку машина здійснює другий цикл, впродовж якого порівнює букви p_2 та p_{n-1} і т. д. Зрозуміло, що найбільшу кількість тактів машина Тюрінґа виконує на симетричних

словах довжини n. Якщо симетричне слово P має парну довжину, то буде проведено $n/2$ повних циклів порівнянь букв, впродовж яких буде здійснено число тактів, яке дорівнює

$$(2n+1)+(2(n-2)+1)+\ldots+5=\frac{(2n+1)+5}{2}\cdot\frac{n}{2}=\frac{(n+3)n}{2}.$$

Після цього машина здійснює ще один такт і зупиняє роботу.

Якщо ж симетричне слово P має непарну довжину, то буде проведено $(n-1)/2$ повних циклів порівнянь букв, впродовж яких буде здійснено число тактів, яке дорівнює

$$(2n+1)+(2(n-2)+1)+\ldots+7=\frac{(2n+1)+7}{2}\cdot\frac{n-1}{2}=\frac{(n+4)(n-1)}{2}.$$

Далі МТ здійснює ще три такти і переходить у кінцевий стан.

З проведених вище міркувань і аналізу графа машини Тюрінґа випливає, що $s_T(n)=n+1\le 2n$, $s_T(n)=O(n)$, а функція часової складності має вигляд

$$t_T(n)=\begin{cases}\frac{(n+3)n}{2}+1, n=2k, k\in\mathbb{N}\\ \frac{(n+4)(n-1)}{2}+3, n=2k-1, k\in\mathbb{N}\end{cases}=\frac{n^2+3n+2}{2}$$

Оскільки $\frac{n^2+3n+2}{2}\le\frac{n^2+3n^2+2n^2}{2}=3n^2$, то $t_T(n)=O(n^2)$ і задача розв'язується за поліноміальний час, тобто належить класу $\mathbb{P}$.

8.2. Приклад. Проаналізувати роботу алгоритму Маркова, заданого над алфавітом $A=\{a_1,a_2,\ldots,a_m\}$, та оцінити його складність:

$$\begin{cases}** \to \#\\ \#a\to a\#,\ \text{де } a\in A\\ \#*\to\#\\ \#\mapsto\\ *ab\to b*a,\ \text{де } a,b\in A\\ \to *\end{cases}$$

Якщо $P=b_1b_2\ldots b_n$, то оберненням слова P називається слово $P'=b_nb_{n-1}\ldots b_1$. Покажемо, що алгоритм здійснює обернення слів в алфавіті A. Дійсно, нехай маємо деяке слово P в алфавіті A. Тоді:

$P\to *P\to b_2*b_1\ldots b_n\to b_2b_3*b_1b_4\ldots b_n\to\ldots\to b_2b_3\ldots b_n*b_1\to *b_2b_3\ldots b_n*b_1\to\ldots\to b_3b_4\ldots b_n*b_2*b_1\to\ldots\to *b_n*b_{n-1}\ldots*$

$b_2*b_1 \to **b_n*b_{n-1}\ldots*b_2*b_1 \to \#b_n*b_{n-1}\ldots*b_2*b_1 \to b_n\#*b_{n-1}\ldots*$
$b_2*b_1 \to b_n\#b_{n-1}\ldots*b_2*b_1 \to \ldots \to b_nb_{n-1}\ldots b_1\# \mapsto b_nb_{n-1}\ldots b_1$

Оцінимо часову та ємнісну складність цього алгоритму. Оскільки кількість підстановок, які виконуються при обчисленні вихідного слова P' з вхідного слова P, залежить тільки від довжини слова, то функція часової складності $t_M(n) = t_M(\omega)$ для будь-якого слова ω довжини n. Знайдемо кількість підстановок при аналізі слова $P = b_1b_2\ldots b_n$ довжини n. На першому етапі алгоритм дописує зліва до слова $*$ і переносить її разом з першою буквою вкінець слова, використовуючи при цьому n підстановок. Далі алгоритм дописує зліва до одержаного слова $*$ і переносить її з другою буквою початкового слова ω до попередньої $*$, використовуючи $n-1$ підстановок і т.д., доки не отримаємо слово вигляду $*b_n * b_{n-1}\ldots * b_2 * b_1$. При цьому кількість підстановок дорівнює $n + (n-1) + \ldots + 1 = \frac{n(n+1)}{2}$. Далі зліва до слова дописується ще одна $*$ і $**$ міняється на $\#$, після чого решітка переноситься вкінець слова, знищуючи при цьому всі зірочки. Насамкінець виконується підстановка $\# \mapsto$ і алгоритм Маркова зупиняє свою роботу. Таким чином, на другому етапі виконується $2 + 2n$ підстановок. Отже,

$$t_M(n) = \frac{n(n+1)}{2} + 2 + 2n = \frac{n^2 + 5n + 4}{2} = O(n^2).$$

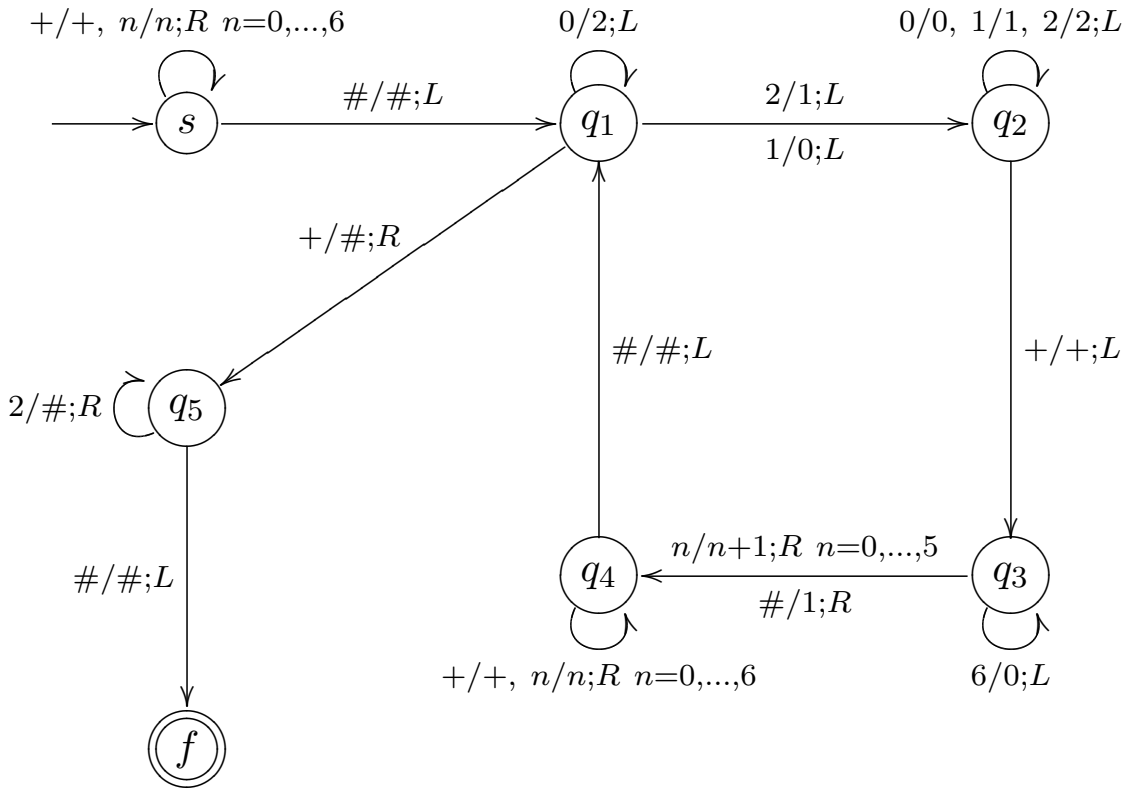
З виведення видно, що $**b_n * b_{n-1}\ldots * b_2 * b_1$ – слово максимальної довжини, тому $s_M(n) = 2n + 1 = O(n)$.

Таким чином, задача розв'язується за поліноміальний час і належить класу $\mathbb{P}$.

8.3. Приклад. Знайти довжину (кількість цифр) натурального числа, записаного в m-ковій системі числення, тобто знайти функцію, яка кожному числу $n = a_km^k + a_{k-1}m^{k-1} + \ldots a_1m + a_0$, де $a_k \neq 0$ і $0 \leq a_i < m$, ставить у відповідність число $|\overline{a_ka_{k-1}\ldots a_0}| = k + 1$.

Оскільки маємо, що $m^k \leq a_km^k + a_{k-1}m^{k-1} + \ldots + a_1m + a_0 \leq (m-1)m^k + (m-1)m^{k-1} + \ldots + (m-1)m + (m-1) = m^{k+1} - 1 < m^{k+1}$, то $k \leq \log_m n < k + 1$. Таким чином, $k = [\log_m n]$ і довжина натурального числа n, записаного в m-ковій системі числення, дорівнює $1 + [\log_m n] = 1 + [\frac{\ln n}{\ln m}] = O(\ln n)$. З останньої рівності видно, що асимптотична оцінка довжини числа не залежить від того, в якій системі числення воно записане.

8.4. Приклад. Побудувати машину Тюрінґа для визначення суми чисел, заданих у сімковій та трійковій системах числення (результат отримати в сімковій системі числення) і оцінити її складність.



Очевидно, що алфавіт $A = \{0, 1, 2, 3, 4, 5, 6, +\}$. Множина станів складається з семи елементів: $Q = \{s, q_1, q_2, q_3, q_4, q_5, f\}$. Нехай доданки відокремлюються знаком $+$, порожні секції, як і раніше, позначатимемо $\#$. На стрічці ліворуч від знака $+$ розміщено доданок у сімковій системі числення, а праворуч – доданок у трійковій системі числення. Ідея розв'язання задачі така: спочатку в стані s переміщуємося до останнього символу, потім від розміщеного праворуч трійкового числа в стані q_1 віднімаємо одиницю, діючи за правилами трійкової арифметики, після чого в стані q_2 переміщуємося ліворуч та до сімкового числа додаємо одиницю в стані q_3, діючи за правилами сімкової арифметики. Далі повертаємося до правого числа з допомогою стану q_4 і переходимо в стан q_1. Повторюємо всі операції доти, доки розміщений праворуч від знака $+$ доданок не буде вичерпано.

Оцінимо складність даного алгоритму. Очевидно, що функція $t_T(\omega)$ буде приймати максимальне значення на словах ω довжини n виду: $a + b_1 b_2 \dots b_{n-2}$. Оцінимо, на стільки може збільшитися

довжина слова в процесі додавання. Оскільки

$$\overline{b_1 b_2 \dots b_{n-2}} = b_1 3^{n-3} + b_2 3^{n-4} + \dots + b_{n-3} 3 + b_{n-2} < 3^{n-2},$$

то при додаванні цього числа в ліву частину загальна довжина слова збільшиться не більше, ніж на $\log_7 3^{n-2} = (n-2)\log_7 3$. Враховуючи крайні порожні клітинки і той факт, що кількість циклів, які здійснить машина Тюрінґа при додаванні, не перевищує 3^{n-2}, маємо, що ємнісна складність

$$s_T(n) \le n \log_7 3 - \log_7 9 + n + 2 = O(n),$$

а часова складність

$$t_T(n) \le 2(n \log_7 3 - \log_7 9 + n + 2)3^{n-2} = O(n3^n).$$

Отже, дана машина Тюрінґа розв'язує задачу за експоненціальний час.

Рекомендована література : [**5**, с. 374–384], [**13**, с. 129–204], [**17**, с. 330–353].

Питання та вправи до параграфа 8.

8.1. Дайте означення (асимптотичної) часової і ємнісної складності машини Тюрінґа та алгоритму Маркова.

8.2. Коли кажуть, що певна задача розв'язується за поліноміальний час?

8.3. Яка задача називається важкорозв'язною?

8.4. Чи можна стверджувати, що а) $2^{n+1} = O(2^n)$; б) $2^{2n} = O(2^n)$?

8.5. Довести, що для полінома $f(n) = a_m n^m + \dots + a_1 n + a_0$, де $a_m > 0$, $a_i \in \mathbb{R}$, $i = \overline{1, m}$, має місце рівність $f(n) = \Theta(n^m)$.

8.6. Доведіть, що $f(n) = n^{O(1)}$ тоді і тільки тоді, коли існує таке додатнє m, для якого $f(n) = O(n^m)$ (вважаємо, що $f(n) > 1$).

8.7. Порівняти час сортування масиву з 1 млн. чисел на двох комп'ютерах, перший з яких виконує 100 млн. операцій за секунду, а другий – 1 млн. Для першого комп'ютера складений алгоритм з складністю $2n^2$ операцій, а для другого – $50nlog_2n$.

8.8. При якому найменшому значенні n алгоритм, що вимагає $100n^2$ операцій, ефективніший за алгоритм, що вимагає 2^n операцій?

8.9. Дослідити, якою буде двійкова довжина числа, одержаного а) додаванням, б) множенням n додатних чисел, двійкова довжина кожного з яких не перевищує k.

8.10. Нехай A і B – $m_1 \times m_2$ і $m_2 \times m_3$ матриці відповідно. Визначити кількість арифметичних операцій, необхідних для обчислення їх добутку за звичайним алгоритмом. Оцінити складність.

8.11. Визначити найефективнішу послідовність множення матриць A, B, C, якщо їх розмірності відповідно

а) 10×5, 5×50, 50×1;

б) 20×50, 50×10, 10×40.

8.12. Визначити найефективнішу послідовність множення матриць A, B, C, D, якщо їх розмірності відповідно

а) 10×2, 2×5, 5×20, 20×3;

б) 20×5, 5×10, 10×40, 40×3.

8.13. Побудувати поліноміальний нормальний алгоритм Маркова в алфавіті $\{a, b\}$ з часовою складністю $O(1)$, який видаляє у вхідному слові, яке містить не менше ніж три букви, третю букву?

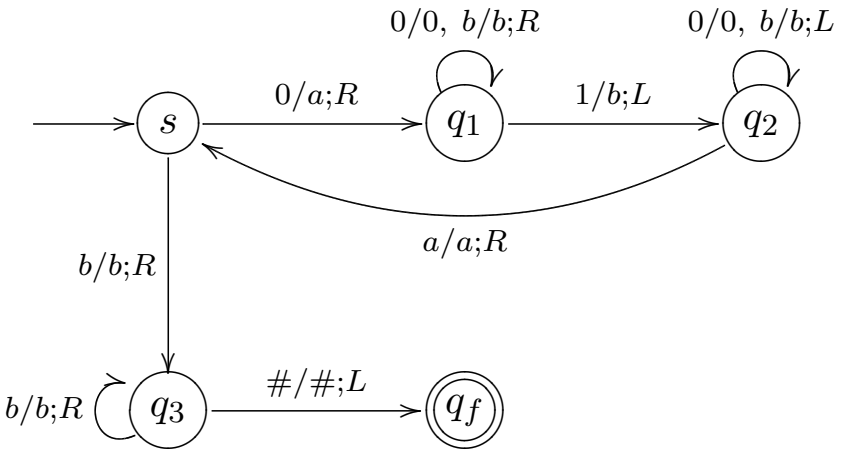
8.14. Побудувати ефективні нормальний алгоритм Маркова та машину Тюрінґа, які обчислюють функцію $f(n) = n - 3$ натуральних четвіркових аргументів. Оцінити їх часову та ємнісну складність.

8.15. Нормальний алгоритм Маркова (A, P) задається схемою

$$P: \begin{cases} ba\# \to a\#b, \text{ де } a, b \in A \\ *a \to a\#a*, \text{ де } a \in A \\ \# \to + \\ + \to \\ * \mapsto \\ \to * \end{cases}$$

Оцінити його часову та ємнісну складність.

8.16. Оцінити часову та ємнісну складність МТ, заданої графом:



8.17. Показати, що наступний НАМ обчислює функцію $f : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$, $f(n, m) = nm$, аргументи якої задані в унарній системі числення:

$$\begin{cases} b| \to |b \\ a| \to |ba \\ a \to \\ |* \to *a \\ *| \to * \\ * \to \\ b \to | \end{cases}.$$

Оцінити його часову та ємнісну складність.

Список літератури

1. Алферова З.В. Теория алгоритмов / З.В. Алферова – М.: «Статистика», 1973. – 164 с.
2. Ахо А. Теория синтаксического анализа, перевода и компиляции / А. Ахо, Дж. Ульман. – М.: Мир, 1978. – Т. 1. – 611 с.
3. Безущак О.О. Елементи теорії чисел: Навчальний посібник / О.О. Безущак, О.Г. Ганюшкін. – К.: Видавничо-поліграфічний центр «Київський університет», 2003. – 202 с.
4. Белоусов А.И. Дискретная математика: Учеб. для вузов / А.И. Белоусов, С.Б. Ткачев. – 3-е изд. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2004. – 744 с.
5. Бондаренко М.Ф. Комп'ютерна дискретна математика: підручник / М.Ф. Бондаренко, Н.В. Білоус, А.Г. Руткас. – Харків: «Компанія СМІТ», 2004. – 480 с.
6. Гаврилків В.М. Формальні мови та алгоритмічні моделі / В.М. Гаврилків. – Івано-Франківськ: «Сімик», 2012. – 172 с
7. Гаврилків В.М. Регулярні вирази у програмних продуктах: навчальний посібник / В.М. Гаврилків. – Івано-Франківськ: «Сімик», 2012. – 72 с.
8. Гаврилов Г.П. Задачи и упражнения по дискретной математике: Учеб. пособие / Г.П. Гаврилов, А.А. Сапоженко. – 3-е изд., перераб. – М.: ФИЗМАТЛИТ, 2005. – 416 с.
9. Гашков С.Б. Современная элементарная алгебра в задачах и решениях / С.Б. Гашков. – М.: МЦНМО, 2006. – 328 с.
10. Завало С.Т. Алгебра і теорія чисел, ч. 2 / С.Т. Завало, В.М. Костарчук, Б.І. Хацет. – К.: Вища школа, 1976. – 384 с.
11. Игошин В.И. Математическая логика и теория алгоритмов: учеб. пособие / В.И. Игошин. – М.: Изд. центр «Академия», 2008. — 448 с.

12. Капітонова Ю.В. Основи дискретної математики / Ю.В. Капітонова, С.Л. Кривий, О.А. Летичевський, Г.М. Луцький, М.К. Печурін. – К.: Наукова думка, 2002. – 580 с.
13. Кривий С.Л. Дискретна математика: Вибрані питання / С.Л. Кривий. – К.: Вид. дім «Києво-Могилянська академія», 2007. – 572 с.
14. Мальцев А.И. Алгоритмы и рекурсивные функции / А.И. Мальцев – М.: Наука, 1986. – 368 с.
15. Марков А.А. Теория алгоритмов / А.А. Марков, Н.М. Нагорный. – М.: Наука, 1984. – 432 с.
16. Мозговой М.В. Классика программирования: алгоритмы, языки, автоматы, компиляторы. Практический подход / М.В. Мозговой. – СПб.: Наука и Техника, 2006. – 320 с.
17. Нікольський Ю.В. Дискретна математика / Ю.В. Нікольський, В.В. Пасічник, Ю.М. Щербина. – К.: Видавнича група BHV, 2007. – 368 с.
18. Новиков Ф.А. Дискретная математика для програмистов / Ф.А. Новиков. – СПб.: Питер, 2001. – 304 с.
19. Оре. О. Приглашение в теорию чисел: Пер с англ. / О. Оре. – Изд. 2-е, стереотипное. – М.: Едиториал УРСС, 2003. – 128 с.
20. Пильщиков В.Н. Машина Тьюринга и алгоритмы Маркова. Решение задач / В.Н. Пильщиков, В.Г. Абрамов, А.А. Вылиток, И.В Горячая. – М.: МГУ, 2006. – 47 с.
21. Самохин А.В. Математическая логика и теория алгоритмов / А.В. Самохин. – Москва, 2003. – 237 с.
22. Тишин В.В. Дискретная математика в примерах и задачах / В.В. Тишин – СПб.: БХВ-Петербург, 2008. – 352 с.
23. Ding-Zhu Du. Problem Solving in Automata, Languages, and Complexity / Ding-Zhu Du, Ker-I Ko. – New York: WIP, 2001. – 388 p.
24. Salomaa A. Formal Languages / A. Salomaa. – New York: Academic Press, 1973. – 281 p.

Покажчик