НОРМАЛЬНІ АЛГОРИТМИ А.А. МАРКОВА.

Ми вже знаємо, що кожен загальний спосіб завдання алгоритмів прийнято називати алгоритмічною системою. Алгоритмічна система, яка основана на відповідності між словами в абстрактному алфавіті, містить у собі об'єкти двоякої природи: *елементарні оператори* та *елементарні розпізнювачі*.

Елементарні оператори – алфавітні оператори, які задаються достатньо просто, та за допомогою послідовного виконання яких реалізуються будь-які алгоритми в розглядуваній алгоритмічній системі.

Елементарні розпізнювачі – об'єкти, які служать для розпізнавання тих чи інших властивостей інформації, яка переробляється алгоритмом. Окрім цього, в залежності від результатів розпізнавання, вони змінюють послідовність, в якій йдуть один за одним елементарні оператори.

Для ілюстрації набору елементарних операторів і порядку їхнього слідування при завданні конкретного алгоритму зручно користуватися орієнтованими графами особливого вигляду – *граф-схемами* відповідних алгоритмів.

Граф-схема алгоритму представляє собою скінченну множину з'єднаних між собою вершин (або геометричних фігур) які прийнято називати *вузлами*. Кожному вузлу, крім особливих вузлів, які називаються *входом* і *виходом*, зіставляється який-небудь елементарний оператор чи елементарний розпізнювач. З кожного вузла, якому зіставлений оператор, а також з вихідного вузла виходить точно по одній дузі. З кожного вузла, якому зіставлений розпізнювач, виходить точно по дві дуги. З вихідного вузла не виходить ні однієї дуги. Число дуг, які входять у вузол граф-схеми, може бути будь-яким.

Розглянемо граф-схему, яка зображена на рисунку 4.1. Тут позначено: EP_i, $i = 1, 2, 3$ – елементарні розпізнювачі; EO_i – елементарні оператори.

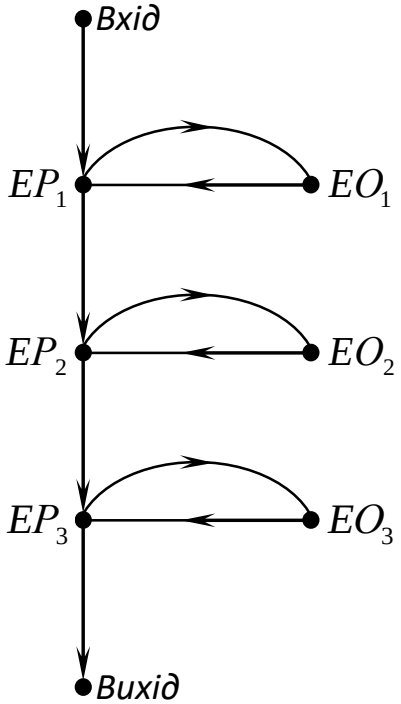

Рисунок 4.1. – Загальний вигляд граф-схеми алгоритму

У даному випадку для вершини, яка відповідає входу, маємо: $\overset{+}{P}$ або $P^+(x)=0$; $\overset{-}{P}$ або $P^-(x)=1$. Для вершин, які відповідають елементарним операторам – $P^+(x)=1$; $P^-(x)=1$. Для вершин, які відповідають елементарним розпізнювачам – $P^+(x)=2$; $P^-(x)=2$. Для вершини, яка відповідає виходу – $P^+(x)=1$; $P^-(x)=0$.

У загальному випадку, число дуг, які входять у вершини EP_i, може бути будь-яким.

Алгоритм, визначений граф-схемою, працює наступним чином. Вхідне слово поступає на вхід і рухається по напрямках, які указані стрілками. Коли слово оказується в розпізнавальному вузлі, то здійснюється перевірка умови, яка зіставлена вузлу. У випадку якщо умова виконується, то слово направляється в операторний вузол, у випадку якщо умова не виконується – до наступного розпізнювача.

Якщо вхідне слово p, яке подане на вхід граф-схеми, проходячи через вузли схеми і перетворюючись, попадає через скінченне число кроків на вихід, то вважається, що *алгоритм застосовний до слова* p (можна сказати, що слово p входить в область визначення алгоритму). Результатом дії алгоритму на слово p

буде те слово, яке оказується на виході схеми.

Якщо після подачі слова p на вхід графа його перетворення і рухи по граф-схемі продовжуються нескінченно довго, не приводячи на вихід, то вважається, що *алгоритм незастосовний до слова* p (слово p не входить в область визначення алгоритму).

У 1954 році математик А.А. Марков запропонував алгоритмічну систему, яка передбачає безпосередній доступ до різних частин перетворюваного вхідного слова і назвав її *нормальним алгоритмом*.

В нормальних алгоритмах як елементарний оператор використовується *оператор підстановки*, а як елементарний розпізнювач – *розпізнювач входження*.

Розпізнювач входження (PB) перевіряє умову – чи здійснюється входження розглядуваного слова p_1 як підслова деякого заданого слова q.

Оператор підстановки ($ОП$) замінює перше зліва входження слова p_1 в слово q на деяке задане слово p_2. Оператор підстановки задається звичайно у вигляді: $p_1 \to p_2$.

Як приклад, розглянемо алгоритм, представлений на рисунку 4.2. Він призначений для переробки слова $abcabca$ за допомогою оператора $bc \to cb$ і через два кроки приводить до слова: $abcabca \to acbabca \to acbacba$.

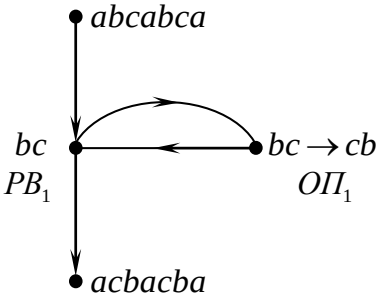

Рисунок 4.2 – Граф-схема узагальненого нормального алгоритму

Послідовність слів p_1, p_2, p_3, …, p_n, яка дістається у процесі реалізації алгоритму, прийнято називати *дедуктивним ланцюжком*, який веде від слова p_1 до слова p_n.

Відзначимо, що алгоритми, які задаються графами, складеними винятково з розпізнювачів входження слів та операторів підстановки, називають *узагальненими нормальними алгоритмами*. При цьому передбачається, що до кожного оператора підстановки веде тільки одна дуга, яка виходить з відповідного розпізнювача.

На графі, який зображений на рисунку 4.3, проілюстрована робота ще одного узагальненого нормального алгоритму. Як видно, у результаті його роботи буде породжений наступний дедуктивний ланцюжок: $bcbaab \to bcabab \to bcaabb \to baaabb \to baaaac$.

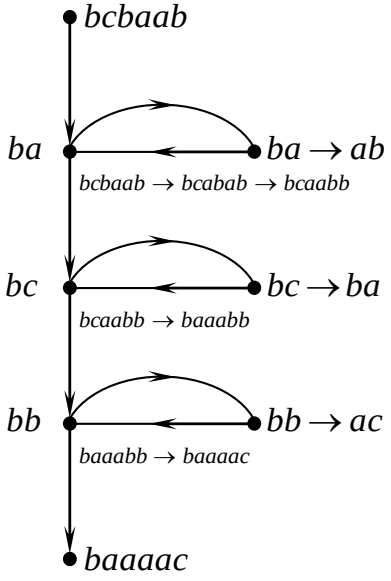

Рисунок 4.3

Нарешті, перейдемо до характеристики власне нормальних алгоритмів.

Нормальними алгоритмами називаються такі узагальнені нормальні алгоритми, граф-схеми яких задовольняють наступним умовам:

1. Усі вузли, які відповідають розпізнювачам (P), упорядковуються за допомогою їхньої нумерації від 1 до n.

2. Дуги, що виходять з вузлів, які відповідають операторам підстановки, приєднуються або до вузла, який відповідає першому розпізнювачу, або до вихідного вузла. У першому випадку підстановка називається *звичайною*, а в другому випадку – *заключною*.

3. Вхідний вузол приєднується дугою до першого розпізнювача.

Нормальні алгоритми прийнято задавати *схемою алгоритму*, яка представляє собою упорядковану сукупність (множину) підстановок всіх операторів даного алгоритму. При цьому звичайні підстановки записуються так само, як в узагальнених алгоритмах, у вигляді двох слів, з'єднаних стрілкою ($p_1 \to p_2$), а заключна

підстановка позначається стрілкою з крапкою ($p_1 \to \cdot p_2$). Процес виконання підстановок закінчується лише тоді, коли жодна з підстановок незастосовна до отриманого слова або коли виконана яка-небудь заключна підстановка.

У загальному вигляді, граф-схема нормального алгоритму може бути такою, як показано на рисунку 4.4.

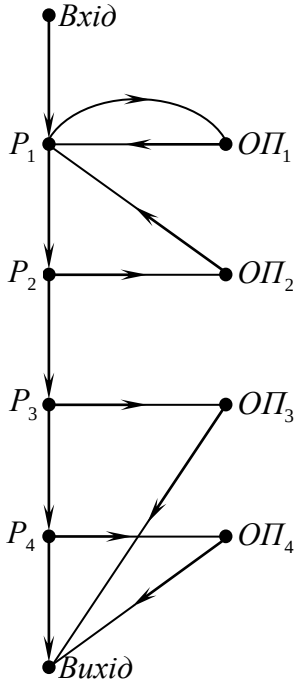

Рисунок 4.4 – Загальний вигляд граф-схеми нормального алгоритму

Наявність у нормальних алгоритмів підстановок двох видів – звичайної та заключної є необхідною *умовою універсальності* нормальних алгоритмів, тобто можливості побудови нормального алгоритму, еквівалентного будь-якому наперед заданому алгоритму. Універсальність нормального алгоритму виражається *принципом нормалізації*, який говорить, що для будь-якого алгоритму (алфавітного відображення, яке задається конструктивно) у довільному алфавіті A можна побудувати еквівалентний йому *нормальний алгоритм над алфавітом* A.

Розшифруємо поняття нормального алгоритму над алфавітом. У ряді випадків не вдається побудувати нормальний алгоритм, еквівалентний даному алгоритму в алфавіті A, якщо використовувати в операторах підстановки алгоритму тільки символи цього алфавіту. Однак можна побудувати шуканий нормальний алгоритм, якщо зробити розширення алфавіту A деякою кількістю нових символів. У цьому випадку говорять, що побудований алгоритм є алгоритмом над алфавітом A, хоча алгоритм як і раніше буде застосовуватися лише до слів у вхідному алфавіті A.

Одномісна часткова словарна функція $F(p)$, яка задана в алфавіті A називається *нормально обчислюваною*, якщо існує нормальний алгоритм N над алфавітом A такий, що для будь-якого слова p в алфавіті A виконується рівність $F(p) = N(p)$.

Як приклад розглянемо словарну функцію $F(p) = pa$ в стандартному алфавіті $C = \{0,1\}$. Нехай $N(p)$ – нормальний алгоритм у розширеному алфавіті $C' = \{0,1,*\}$, який має схему: $*0 \to 0*$; $*1 \to 1*$; $* \to \cdot a$; $\Lambda \to *$. Оператор підстановки $\Lambda \to *$ відразу наказує додати символ $*$ у початок вхідного слова p. Це необхідно зробити виходячи з тих же міркувань, якими ми керувалися використовуючи оператор $e \to *$ при розв'язанні задачі 3.

Візьмемо яке-небудь слово в алфавіті C, наприклад $p = 011110$, і подамо його на

вхід граф-схеми, яка зображена на рисунку 4.5.

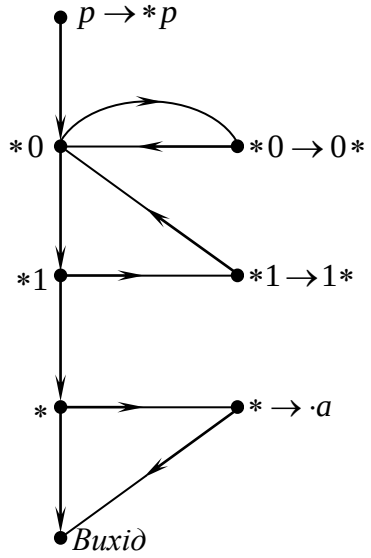

Рисунок 4.5 – Алгоритм обчислення словарної функції $F(p) = pa$

Дедуктивний ланцюжок перетворення вхідного слова p у вихідне слово має вигляд:

$$011110 \to *011110 \to 0*11110 \to 01*1110 \to 011*110 \to$$
$$\to 0111*10 \to 01111*0 \to 011110* \to 011110a.$$

Таким чином, нормальний алгоритм у розширеному алфавіті C' із зазначеною схемою підстановок обчислює словарну функцію $F(p)$, яка задана в стандартному алфавіті C.

Математично довести принцип нормалізації неможливо, оскільки поняття довільного алгоритму не є строго визначеним математичним поняттям.

Той чи інший алгоритм називається *нормалізованим*, якщо еквівалентний йому нормальний алгоритм побудувати можна, і *ненормалізованим* – в іншому випадку. Принцип нормалізації, сформульований вище, тепер можна сформулювати в іншій формі: *всі алгоритми нормалізовані*.

Справедливість цього принципу заснована на тому, що всі відомі у даний час алгоритми є нормалізованими, крім того, способи композиції, які дозволяють будувати нові алгоритми з уже відомих, не виходять за межі класу нормалізованих алгоритмів.

Розглянемо найбільш розповсюджені види композицій нормальних алгоритмів.

I. Суперпозиція алгоритмів.

Нехай дані два алгоритми A і B. При їхній суперпозиції, вихідне слово першого алгоритму A розглядається як вхідне слово другого алгоритму B. Результатом суперпозиції алгоритмів A і B буде новий алгоритм C, який можна представити у вигляді:

$$C(p) = B(A(p)).$$

Суперпозиція може виконуватися для будь-якого скінченного числа алгоритмів.

Суперпозицію узагальнених нормальних алгоритмів можна розглядати як узагальнений нормальний алгоритм. Для цього достатньо вихідний вузол граф-схеми кожного такого попереднього алгоритму сполучити з вхідним вузлом послідуючого алгоритму.

II. Об'єднання алгоритмів.

Об'єднанням алгоритмів A і B в одному і тому же алфавіті X називається алгоритм C у цьому же алфавіті, який перетворює будь-яке слово p, яке міститься в перетинанні областей визначення алгоритмів A і B в записані поруч слова $A(p)$ і $B(p)$. Таким чином,

$$C(p) = A(p), B(p).$$

Відзначимо, що на всіх інших вхідних словах цей алгоритм вважається невизначеним.

Розглянемо наступний найпростіший приклад, який ілюструє роботу композиції об'єднання алгоритмів. Нехай задані: $X = \{a, b\}$; $A = \{ab \to ba\}$, $B = \{ba \to ab\}$. Тоді для слова $p = aba$ з області перетинання алгоритмів A і B маємо: $A(p) = aba \to baa$; $B(p) = aba \to aab \Rightarrow C(p) = baaaab$.

III. Розгалуження алгоритмів.

Це вже комбінація трьох алгоритмів A, B і C. Результат цієї композиції позначається буквою D. Область визначення алгоритму D співпадає з перетинанням областей визначення всіх трьох алгоритмів, а для будь-якого слова p з цього перетинання маємо:

$$D(p) = \begin{cases} A(p), \text{ якщо } C(p) = e; \\ B(p), \text{ якщо } C(p) \neq e. \end{cases}.$$

Нехай задані: $X = \{a, b\}$; $A = \{ab \to ba\}$, $B = \{ba \to ab\}$, $C = \{ab \to a, ba \to e\}$. Розглянемо дію алгоритму D на слова aba і bab.

Для слова $p = aba$:	Для слова $p = bab$:
$A(p) = aba \to baa$; $B(p) = aba \to aab$;	$A(p) = bab \to bba$; $B(p) = bab \to abb$;
$C(p) = aba \to aa$.	$C(p) = bab \to ba \to e$.
Оскільки $C(p) \neq e$, то $D(p) = B(p) = aab$.	Оскільки $C(p) = e$, то $D(p) = A(p) = bba$.

IV. Повторення (ітерація) алгоритмів.

Ітерація представляє собою композицію двох алгоритмів A і B. Результат цієї композиції позначається буквою C. Для будь-якого вхідного слова p відповідне йому вихідне слово $C(p)$ визначається у результаті послідовного багаторазового застосування алгоритму A доти, поки не дістанеться слово, яке переробляється алгоритмом B у деяке фіксоване слово.

Розглянемо приклад. Нехай задані: $X = \{a, b\}$; $A = \{ab \to ba\}$, $B = \{bbbaa \to ab\}$. Багаторазове застосування алгоритму A до слова $p = ababb$ описується наступним дедуктивним ланцюжком: $ababb \to baabb \to babab \to bbaab \to bbaba \to bbbaa$, що приводить до слова, яке може переробити алгоритм B. Таким чином, $C(p) = B(bbbaa) = ab$.

Велике значення для нормальних алгоритмів, як і для будь-якої універсальної алгоритмічної системи, має задача побудови *універсального алгоритму*, який повинен виконувати роботу будь-якого нормального алгоритму, якщо задана його схема у вигляді операторів підстановок.

Для побудови універсального алгоритму U може бути застосована, наприклад, наступна схема. Фіксується деякий алфавіт (наприклад, стандартний двійковий алфавіт C) і для всіх інших можливих алфавітів задається деякий спосіб їхнього кодування в алфавіті C. Для символів, які застосовуються у схемах нормальних алгоритмів (вхід, вихід, звичайна і заключна підстановки), а також для роздільника операторів підстановки використовуються окремі спеціальні коди.

Тоді, якщо заданий деякий нормальний алгоритм N, його кодують одним словом N^U, яке представляє собою зображення алгоритму N в стандартному алфавіті. Вхідне слово p кодують у слові p^U, яке, відповідно, буде представляти зображення слова.

Наприклад, задано алгоритм $N = \{xy \to x, y \to y\}$ і вхідне слово $p = xxyx$. Задамо

наступне кодування в стандартному алфавіті: $\Gamma x = 010$; $\Gamma y = 020$; $\Gamma \to = 030$; $\Gamma <$ *роздільник* $>= 040$; $\Gamma <$ *вхід/ вихід* $>= 050$. Тоді зображення слова є p^U =010 010 020 010, а зображення алгоритму N^U =050 010 020 030 010 040 020 030 020 050.

Справедлива наступна теорема А.А. Маркова про універсальний нормальний алгоритм. Існує такий універсальний нормальний алгоритм U, який для будь-якого нормального алгоритму N і будь-якого вхідного слова p з області визначення алгоритму N переводить слово N^U p^U (яке дістається приписуванням зображення слова p (p^U) до зображення алгоритму N (N^U)) у слово R^U, яке є зображенням відповідного вихідного слова $R = N(p)$, у яке алгоритм N переробляє слово p. Якщо ж слово p вибирається так, що алгоритм N до нього незастосовний, то і універсальний алгоритм U незастосовний до слова N^U p^U.

Теорема Маркова має велике значення, тому що з неї випливає можливість побудови машини, яка може виконувати роботу будь-якого нормального алгоритму, а в силу принципу нормалізації – роботу довільного алгоритму. У цьому випадку програма буде представляти собою слово N^U, а вихідні дані – слово p^U.

Принцип нормалізації хоча і доведений для усіх відомих алгоритмів, однак фактична його реалізація є досить складною. Тому на практиці машини будують на підставі інших алгоритмічних систем.