ЛЕКЦІЯ 12. РІВНОДОСТУПНА АДРЕСНА МАШИНА

Алгоритмічна система Поста

Система Поста дуже схожа на систему Тьюрінга, тому лише коротко опишемо машину Поста та сформулюємо головну тезу.

Машина Поста має нескінченну інформаційну стрічку, на яку записують інформацію в двійковому алфавіті $\{0,1\}$. Алгоритм задають як скінченну впорядковану послідовність перенумерованих правил - команд Поста.

Робота машини відбувається дискретними кроками, на кожному кроці виконується одна команда. Кожному кроку відповідає на інформаційній стрічці *активна комірка*. Для першої команди положення активної комірки фіксоване - це *початкова активна комірка*. Наступні положення активної комірки змінюються відповідно до команд Поста.

Є пость типів команд Поста:

1. відмітити активну комірку стрічки, тобто записати в неї 1 і перейти до виконання *i*-ї команди;
2. стерти відмітку активної комірки, тобто записати в неї 0 і перейти до виконання *i*-ї команди;
3. змістити активну комірку на одну позицію вправо і перейти до виконання *i*-ї команди;
4. змістити активну комірку на одну позицію вліво і перейти до виконання *i*-ї команди;
5. якщо активна комірка відмічена, то перейти до виконання *i*- ї команди, інакше - до виконання *j*-ї команди;
6. зупинка, закінчення роботи алгоритму.

Отже, в алгоритмічній системі Поста, на відміну від інших систем, у явному вигляді сформульовано принцип безумовного передавання (команди 1-4) і умовного передавання керування (команда 5).

Алгоритм, складений з довільної скінченної кількості команд Поста, називають ***алгоритмом Поста***.

***Теза Поста**: Для кожного алгоритму $A=\langle \varphi, P \rangle$ існує алгоритм Поста, який реалізує словникову функцію φ.*

Доведено, що система Поста еквівалентна іншим системам, зокрема, системам Тюрінга, Маркова, Черча-Кліні.

Операторні алгоритмічні системи

Однією з особливостей розглянутих алгоритмічних систем, за винятком машини Поста, є незмінність набору допустимих засобів, використовуваних для запису алгоритмів і процесів їхнього виконання. В системі нормальних алгоритмів такими є один оператор (підстановки) і один розпізнавач (входження); у системі рекурсивних функцій фіксованим є набір базових функцій $\left(S, O, I_m^n\right)$ і набір операторів $\left(S, R, M, M^l\right)$; під час обчислення на машині Тьюрінга елементарні такти обмежені фіксованим набором операцій (заміна символу в комірці, зміна стану машини і позиції головки).

Водночас під час вивчення конкретних алгоритмів бажано мати такі допустимі засоби їхнього опису, які найзручніші для цього класу алгоритмів. Наприклад, алгоритми лінійної алгебри найзручніше описувати за допомогою чотирьох арифметичних дій, а алгоритми обчислення функцій алгебри логіки - за допомогою тих логічних операцій, у термінах яких ці функції записані (або заперечення).

Операторні алгоритмічні системи першими почали орієнтувати на конкретні мови програмування, тобто на реальні обчислювальні машини. До визначення алгоритму в разі практичного використання ставлять такі вимоги:

- можливість вибору засобів опису алгоритмів залежно від класу алгоритмів;
- дозвіл формування команд алгоритму в процесі його виконання;
- дозвіл використання широкого діапазону елементарних логічних операцій над об'єктами.

Машина з довільним доступом до пам'яті

Однією з найпоширеніших операторних алгоритмічних систем, які найбільше наближені до практичної реалізації, є ***машина з довільним доступом до пам'яті***, або ***рівнодоступна адресна машина (РАМ)***. Машина з довільним доступом до пам'яті моделює обчислювальну машину з одним суматором, у якій команди програми не можуть самі себе змінювати.

РАМ складається з вхідної стрічки, з якої вона може лише читати, вихідної стрічки, на яку вона може лише записувати, і пам'яті (рис. 12.1).

Вхідна стрічка - це послідовність комірок, кожна з яких може містити ціле число (можливо, від'ємне). Кожного разу, коли символ зчитується з вхідної стрічки, її головка зчитування зміщується на одну комірку вправо.

Вихідна стрічка теж розбита на комірки, які спочатку порожні. Під час виконання команди запису в тій комірці вихідної стрічки, яку визначає головка, друкується ціле число, і головка зміщується на одну комірку вправо. Як тільки символ записано, його вже не можна змінити.

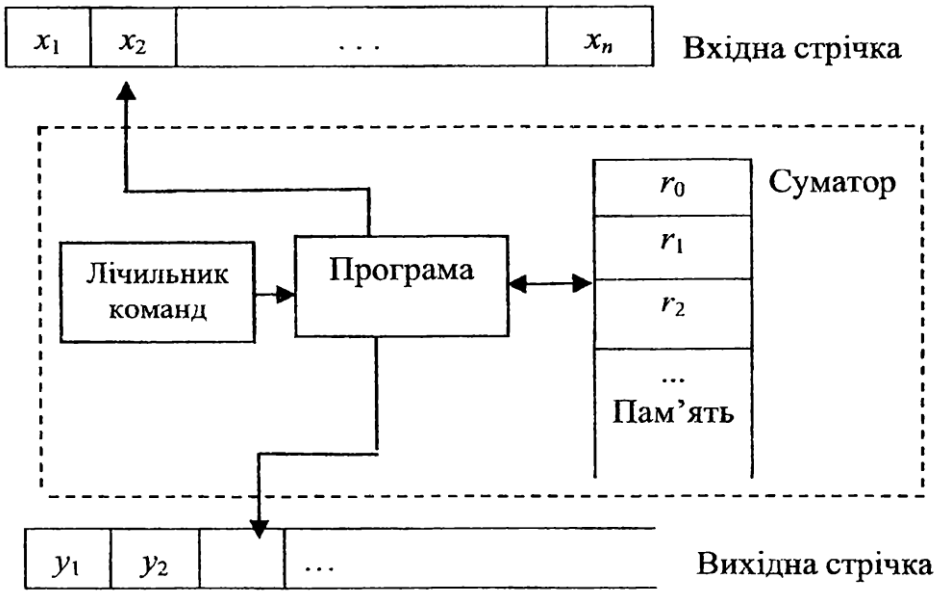

Рис. 12.1.

Пам'ять складається з послідовності регістрів $r_0, r_1, \ldots,$ кожен з яких здатний зберігати довільне ціле число. Зазначимо, що верхня межа на кількість регістрів, які можна використовувати, не визначена. Така ідеалізація допустима у випадку, коли:

1. розмір задачі достатньо малий, щоб вона помістилась в основній пам'яті обчислювальної машини;
2. цілі числа, які беруть участь в обчисленнях, достатньо малі, щоб їх можна було помістити в одну комірку.

Програма для РАМ не записується в пам'ять і тому не може сама себе змінювати. Програма - це послідовність (можливо) помічених команд. Вони нагадують ті, які звичайно трапляються в реальних обчислювальних машинах. Передбачають, що є арифметичні команди, команди введення-виведення, непряма адресація (для індексації масивів) і команди розгалуження.

Всі обчислення відбуваються в першому регістрі r_0, який називають ***суматором***; він, як і будь-який регістр пам'яті, може зберігати довільне ціле число.

Таблиця 12.1.

Таблиця команд РАМ

Код операції	**Адреса**	**Дія**
LOAD	Операнд	Операнд заноситься в суматор
STORE	Операнд	Інформація з суматора заноситься в регістр (операнд)
ADD	Операнд	До вмісту суматора додається операнд
SUB	Операнд	Від вмісту суматора віднімається операнд
MULT	Операнд	Вміст суматора домножується на операнд
DIV	Операнд	Вміст суматора ділиться на операнд
READ	Операнд	Черговий вхідний символ заноситься в регістр (операнд)
WRITE	Операнд	Операнд або його вміст записують на вихідну стрічку
JUMP	Мітка	Безумовний перехід на команду із заданою міткою
JGTZ	Мітка	Якщо вміст суматора додатний, то відбувається перехід на команду із заданою міткою, інакше -- на наступну команду

JZERO	Мітка	Перехід на команду із заданою міткою відбувається тоді, коли вміст суматора дорівнює нулю
HALT		Завершення програми

Приклад набору команд наведено в таблиці 12.1. Кожна команда складається з двох частин - *коду операції* та *адреси*.

Тип операнда може бути одним з таких:

1. $=i$, означає ціле число, i називають літералом;
2. i, вміст регістра i (ї повинно бути невід'ємним), будемо позначати $c(i)$;
3. $*i$, означає непряму адресацію, тобто значенням операнда є вміст регістру $j - c(j)$, де $j = c(i)$ - ціле число, що міститься в регістрі i; якщо $j < 0$, то машина зупиняється.

Невизначені команди можна вважати еквівалентними HALT. У разі виконання перших восьми команд з таблиці лічильник команд збільшується на 1, тому такі команди виконуються послідовно; JUMP, JGTZ, JZERO - команди переходів.

Загалом РАМ-програма визначає відображення з множини вхідних стрічок у множину вихідних стрічок. Оскільки на деяких вхідних стрічках програма може не зупинятись, то це відображення є частковим (тобто для деяких входів воно може бути невизначене). Це відображення можна інтерпретувати різними способами. Дві важливі інтерпретації - інтерпретація у вигляді функції та інтерпретація у вигляді мови.

Припустимо, що програма $П$ завжди зчитує з вхідної стрічки n цілих чисел і записує на вихідну стрічку не більше одного цілого числа. Нехай $x_1, x_2, \ldots, x_n$ - цілі числа, які містяться в n перших комірках вхідної стрічки, і нехай програма $П$ записує одне число y у першу комірку вихідної стрічки, а потім через якийсь час зупиняється. Тоді кажуть, що $П$ обчислює функцію $f(x_1, x_2, \ldots, x_n) = y$. Легко довести, що РАМ, як і будь-яка інша розумна модель обчислювальної машини, може обчислювати точно частково-рекурсивні функції. Іншими словами, для довільної частково-рекурсивної функції f можна

написати РАМ-програму, яка обчислює *f*, і для довільної РАМ-програми можна задати еквівалентну їй частково-рескурсивну функцію ($K_{\text{ч.р.}} = K_{\text{РАМ}}$).

Інший спосіб інтерпретувати програму для РАМ - це розглянути її з погляду мови, яку вона розпізнає. Символи алфавіту можна подати цілими числами $1, 2, \ldots, k$. РАМ розпізнає мову в такому сенсі. Нехай на вхідній стрічці міститься ланцюжок символів $s = a_1 a_2 \ldots a_n$, причому символ a_1 - у першій комірці, символ a_2 - у другій і так далі, а в $(n+1)$-й комірці розміщено 0-символ, який використовуватиметься як маркер кінця вхідного рядка (кінцевий маркер).

Вхідний *ланцюжок s розпізнає РАМ-програма П*, якщо *П* прочитує всі його символи і кінцевий маркер, пише 1 у першій комірці вихідної стрічки і зупиняється.

Мовою, яку розпізнає програма П, називають множину всіх ланцюжків (слів), які вона допускає. Для вхідних ланцюжків, що не належать мові, яку розпізнає програма *П*, машина може надрукувати на вихідній стрічці символ, відмінний від 1, і зупинитись, а може не зупинитись узагалі.

Легко довести, що мову розпізнає деяка РАМ, тоді й лише тоді, коли вона рекурсивно-перелічувана. Мову розпізнає РАМ, яка зупиняється на всіх входах, тоді і лише тоді, коли вона рекурсивна.

Зазначимо, що *рекурсивна мова* - це мова з алгоритмічно розв'язною проблемою входження, тобто характеристична функція якої є частково-рекурсивною.

Приклад. Нехай на вхідну стрічку РАМ подано два цілі числа a_1, a_2. Побудувати РАМ-програму для знаходження значення виразу $S = a_1 + a_2 + 1$.

Програма:

```
READ 1
READ 2
LOAD 1
ADD 2
ADD =1
```

```
STORE 1
WRITE 1
HALT
```

Обчислювальна складність РАМ-програм

Часова складність РАМ-програми - це функція $f(n)$, яка дорівнює найбільшому (на всіх входах розміру n) із підсумованих часів, затрачених на кожну команду, що працює.

Часова складність у середньому - це середнє, взяте по всіх входах розміру n, тих же сум.

Такі ж поняття визначають для ємнісної складності РАМ-програми, у цьому разі підсумовують ємність всіх регістрів, до яких було звертання.

Щоб точно визначити часову та ємнісну складності, треба задати час, необхідний для виконання кожної РАМ-команди, і обсяг пам'яті, використовуваної кожним регістром. Розглянемо два такі вагові критерії для РАМ-програм.

У випадку *рівномірного вагового критерію* кожна РАМ- команда затрачає на своє виконання одну одиницю часу і кожен регістр використовує одну одиницю пам'яті.

Друге визначення бере до уваги обмеженість розміру реальної комірки пам'яті, його називають *логарифмічним ваговим критерієм*. Нехай $l(i)$ - логарифмічна функція на цілих числах, задана рівностями

$$l(i) = \begin{cases} \lfloor \log|i| \rfloor + 1, & i \neq 0 \\ 1, & i = 0 \end{cases},$$

де $\lfloor x \rfloor$ - нижнє ціле від x. Тут враховано, що для зображення цілого числа n в регістрі потрібно $\lfloor \log|n| \rfloor + 1$ бітів.

Логарифмічні ваги для трьох можливих видів операндів такі:

$=i$ $l(i)$

i $l(i)+l\big(c(i)\big)$.

$*i$ $l(i)+l\big(c(i)\big)+l\Big(c\big(c(i)\big)\Big)$

Таблиця 12.2

Логарифмічні ваги РАМ-команд

Команда	**Вага**
LOAD *a*	$t(a)$ – вага операнда *a*
STORE *i*	$l\big(c(0)\big)+l(i)$
STORE **i*	$l\big(c(0)\big)+l(i)+l\big(c(i)\big)$
ADD *a*	$l\big(c(0)\big)+t(a)$
SUB *a*	$l\big(c(0)\big)+t(a)$
MULT *a*	$l\big(c(0)\big)+t(a)$
DIV *a*	$l\big(c(0)\big)+t(a)$
READ **i*	$l(\text{вхід})+l(i)+l\big(c(i)\big)$
READ *i*	$l(\text{вхід})+l(i)$
WRITE *a*	$t(a)$
JUMP *b*	1
JGTZ *b*	$t\big(c(0)\big)$
JZERO *b*	$t\big(c(0)\big)$
HALT	1

Логарифмічний ваговий критерій ґрунтується на грубому припущенні, що ціна виконання команди (її вага) пропорційна до довжини її операндів (таблиця 12.2).

Визначимо *логарифмічну ємнісну складність* РАМ-програми як суму за всіма регістрами, які працювали, включаючи суматор, величин $l(x_i)$, де

x_i - найбільше за абсолютною величиною ціле число, що містилося в *i*-му регістрі за весь час обчислень.

Зрозуміло, що програма може мати різні часові складності залежно від того, який критерій використовують - рівномірний чи логарифмічний. Якщо припустити, що для зберігання кожного числа, яке трапляється в задачі, достатньо одного машинного слова, то користуються рівномірною ваговою функцією. В іншому випадку для реалістичного аналізу складності більше може підходити логарифмічна вага.

Поряд з РАМ розглядають ще іншу модель обчислювальної машини - *машину з довільним доступом до пам'яті і програмою, що зберігається в пам'яті (РАСП)*, оцінки якої не перевищують відповідні оцінки РАМ більше ніж у *k* разів (*k* = const).

Зв'язок машин Тьюрінга і РАМ

Головно машини Тьюрінга застосовують у визначенні нижніх оцінок, необхідних для розв'язання заданої задачі. У більшості випадків вдається знайти нижні оцінки лише з точністю до поліномної зв'язаності. Для точніших оцінок треба розглядати специфічніші деталі конкретних моделей.

Розглянемо зв'язок між РАМ та МТ. Очевидно, що РАМ може моделювати роботу *k*-стрічкової МТ, розміщуючи по одній клітинці МТ в регістр. Зокрема *i*-ту клітинку *j*-ї стрічки можна зберігати в регістрі $ki + j + c$, де *c* - деяка стала, яка дає змогу зали шити певну кількість вільних регістрів для проміжних обчислень, у тому числі *k* регістрів для запам'ятовування положення *k* головок МТ. РАМ може читати з клітинок МТ, використовуючи непряму адресацію за допомогою регістрів, які містять положення головок на стрічках.

Припустимо, що задана МТ має часову складність $T(n) \geq n$. Тоді РАМ може прочитати її вхід, запам'ятати його в регістрах, які відображають першу стрічку, і змоделювати цю машину Тьюрінга за час $O(T(n))$ у разі рівномірного вагового критерію і $O(T(n)\log_2 T(n))$ у випадку логарифмічного.

У будь-якому випадку час на РАМ обмежений зверху поліномом від часу МТ, бо довільна функція $O\left(T\left(n\right)\log_2 n\right)$ є, зрозуміло, $O\left(T^2\left(n\right)\right)$.

Обернене твердження правильне лише у разі логарифмічного вагового критерію для РАМ. За рівномірного критерію *n*-крокова програма для РАМ без входу може обчислювати числа порядку 2^{2^n}, що потре бус порядку 2^n кліток МТ лише для запам'ятовування і зчитування. Тому у разі рівномірного критерію немає поліномального зв'язку між РАМ та МТ.

Якщо зростання чисел за порядком не перевищує розміру задачі, то варто використовувати РАМ з рівномірною вагою. Хоча під час аналізування алгоритмів зручніше застосовувати рівномірний ваговий критерій з огляду на його простоту, однак його треба відкинути, якщо визначають нижні межі на часову складність.

Для логарифмічної ваги виконується така теорема.

***Теорема**. Нехай L - мова, яку розпізнає деяка РАМ за час $T(n)$ у випадку логарифмічної ваги. Якщо в РАМ-програмі немає множень і ділень, то на багатострічкових машинах Тьюрінга L має часову складність, не більшу від* $O\left(T^2\left(n\right)\right)$.

Якщо в РАМ-програмі є команди множення і ділення, то можна написати підпрограми для МТ, які виконують ці операції за допомогою додавання та віднімання. Легко довести, що логарифмічні ваги цих підпрограм не більші, ніж квадрат логарифмічних ваг відповідних команд.

***Теорема**. РАМ і РАСП з логарифмічною вагою і багатострічкові машини Тьюрінга поліномально зв'язані.*

Аналогічний результат отримують і для ємнісної складності.