Алгоритми та моделі обчислень

Лекція 1

Тема 1. Вступ до теорії алгоритмів

1.1 Неформальне тлумачення алгоритму

Алгоритми та методи обчислень

Частина 1. Тема 1. Вступ до теорії алгоритмів

1.1 Неформальне тлумачення алгоритму

1.1.1 Історія поняття алгоритму

1.1.2 Визначення алгоритму

1.1.3 Основні властивості алгоритму

1.1.4 Параметри алгоритму

1.1.5 Базові структури алгоритмів(алгоритмічні конструкції). Теорема Бьома-Якопіні.

1.1.6 Рекурсивні алгоритми

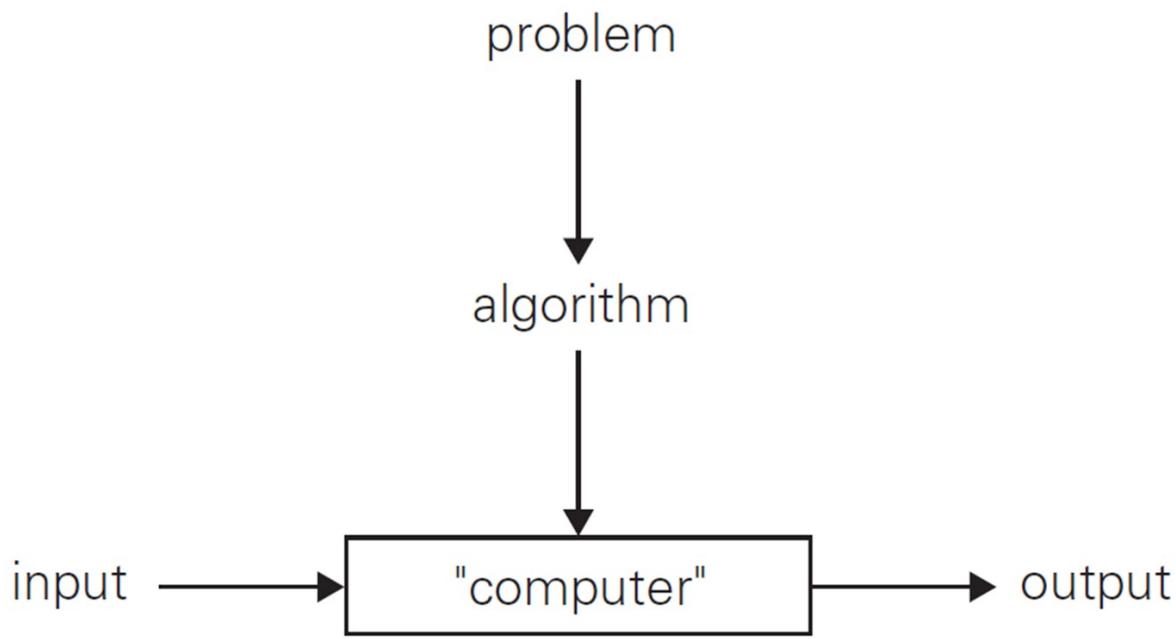
1.1.7 Паралельні алгоритми

1.1.8 Недетерміновані алгоритми

1.1.9 Імовірнісні алгоритми (увипадковлені алгоритми)

1.1.1 Історія поняття алгоритму

Слово алгоритм походить від algorithmi - латинської форми написання імені великого математика IX ст. Аль-Хорезмі, який сформулював правила виконання арифметичних дій.


problem
algorithm
input
"computer"
output

1.1.2 Визначення алгоритму

Алгоритм – фундаментальне поняття математики, йому не можна дати точне математичне визначення із залученням інших фундаментальних понять. Відсутність такого визначення не заважає інтуїтивному розумінню його змісту. Але в літературних джерелах наводяться різні варіанти розуміння сутності алгоритму.

Далі наводяться кілька можливих визначень поняття «алгоритм».

1.1.2 Визначення алгоритму

Алгоритм — скінчений набір правил, який визначає послідовність операцій для розв’язку конкретної множини задач та володіє наступними важливими рисами: скінченністю, визначеністю, вводом, виводом, ефективністю.
Алгоритм — система обчислень, що виконується за чітко визначеними правилами, яка після деякої кількості кроків приводить до розв’язку поставленої задачі.
Алгоритм — чітко детермінована послідовність дій, яка описує процес перетворення об’єкту із початкового стану в кінцевий, записана за допомогою зрозумілих виконавцю команд.
Алгоритм — послідовність дій, направлених на отримання кінцевого результату за скінчену кількість кроків.
Алгоритм — послідовність дій, яка або приводить до розв’язку задачі, або пояснює, чому такий розв’язок отримати не можливо.
Алгоритм — точна, однозначна, скінчена послідовність дій, яку необхідно виконати для досягнення конкретної мети за скінчену кількість кроків.
Алгоритм — це скінчена послідовність команд, які потрібно виконати над вхідними даними для отримання результату.

1.1.2 Визначення алгоритму

Визначення абстрактного алгоритму (за Черкаським М.В.)

Задача – це сформульоване намагання, яке за набором вхідних даних і умов за допомогою деякого алгоритму дозволяє отримати результат, що повністю відповідає цьому набору вхідних даних і умов:

$M: y_1, y_2,\ldots y_m \leftarrow x_1, x_2,\ldots x_n$
, де M – умови; $x_1, x_2,\ldots x_n$ – вхідні дані; $y_1, y_2,\ldots y_m$ – результати; $\leftarrow$ – невизначений алгоритм.

Алгоритм – точний припис, який задає обчислювальний процес (що називається в цьому випадку алгоритмічним), що починається з довільного початкового даного (з деякої сукупності можливих для даного алгоритму початкових даних) і спрямований на отримання результату, який повністю визначається цим початковим даним

Для реалізації абстрактного алгоритму використовується уявний обчислювач, у якому не декларується апаратних засобів.

1.1.3 Основні властивості алгоритму

Скінченність. Алгоритм має завжди завершуватись після виконання скінченної кількості кроків.

Дискретність. Процес, що визначається алгоритмом, можна розділити на окремі елементарні етапи.

Визначеність. Кожен крок алгоритму має бути точно визначений.

Вхідні дані. Алгоритм має деяку кількість (можливо, нульову) вхідних даних.

Вихідні дані. Алгоритм має одне або декілька вихідних даних.

Ефективність. Алгоритм має завершуватися отриманням результату обчислень, що записуються у вихідні дані.

Масовість. Властивість алгоритму, яка полягає в тому, що алгоритм повинен забезпечувати розв'язання однотипних задач для будь-яких вхідних даних.

1.1.4 Параметри алгоритму *(на основі публікацій проф. Черкаського М.В.)*

Модель - це формалізоване, спрощене представлення реального об'єкту.

Будь-яка наукова модель має цільовий характер, тобто вона будується для конкретних цілей і для вирішення певних задач. Вона достатньо точно відображає ті сторони модельованого явища, які мають ключове значення для вирішення поставленої задачі. Укладена в моделі помилка рано чи пізно заважає вирішувати інші задачі, пов'язані з модельованим об'єктом. Вирішити нові задачі можна або шляхом уточнення, узагальнення первинної моделі, або шляхом побудови нової моделі.

Розглядаючи найзагальніше представлення алгоритму за допомогою простої блок-схеми алгоритму можна виділити: систему вхідних даних, систему проміжних результатів та систему кінцевих результатів(рис. 1.1).

1.1.4 Параметри алгоритму *(на основі публікацій проф. Черкаського М.В.)*

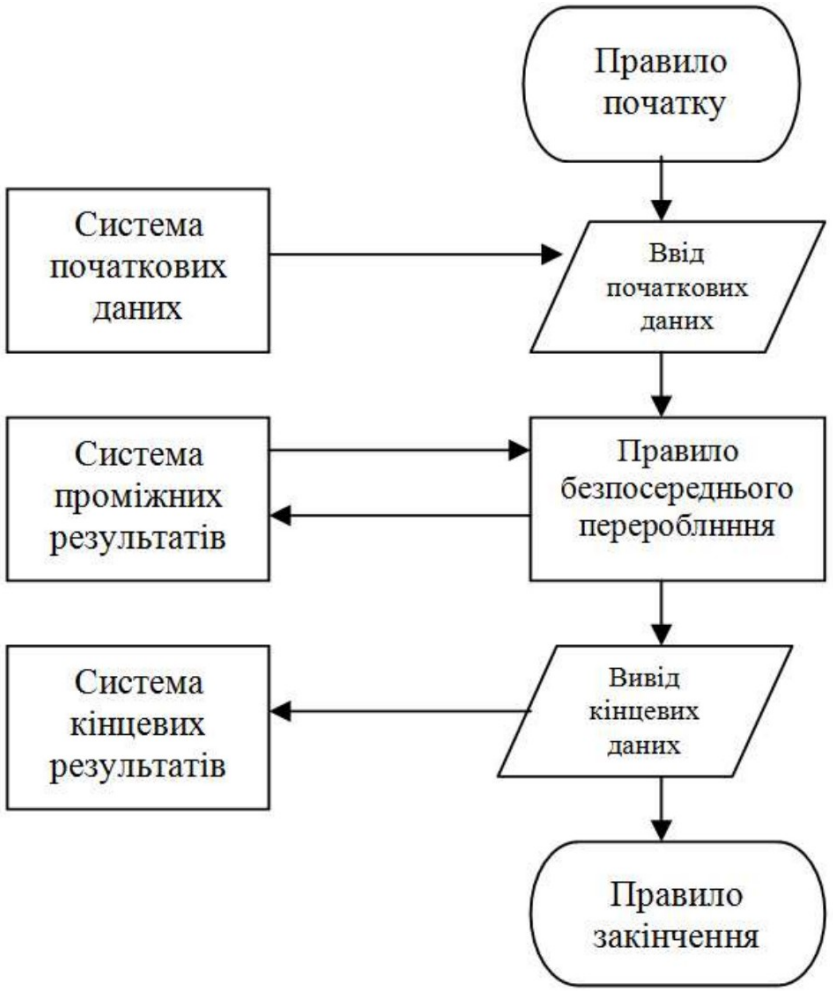


Рис. 1.1.

1.1.4 Параметри алгоритму *(на основі публікацій проф. Черкаського М.В.)*

Тоді можна сформувати параметричну модель алгоритму(рис 1.2).

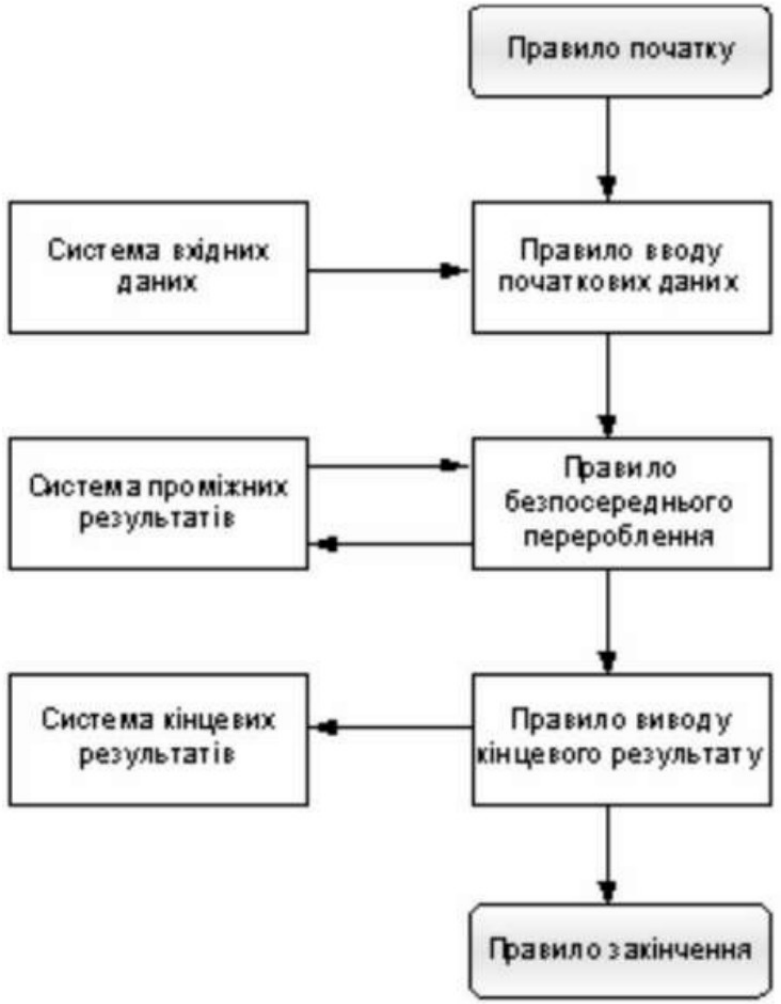


Рис. 1.2.

1.1.4 Параметри алгоритму *(на основі публікацій проф. Черкаського М.В.)*

Тому загалом можна виділити такі параметри алгоритму:

1) правило початку;

2) правило вводу даних;

3) система(потенційно нескінченна) вхідних даних;

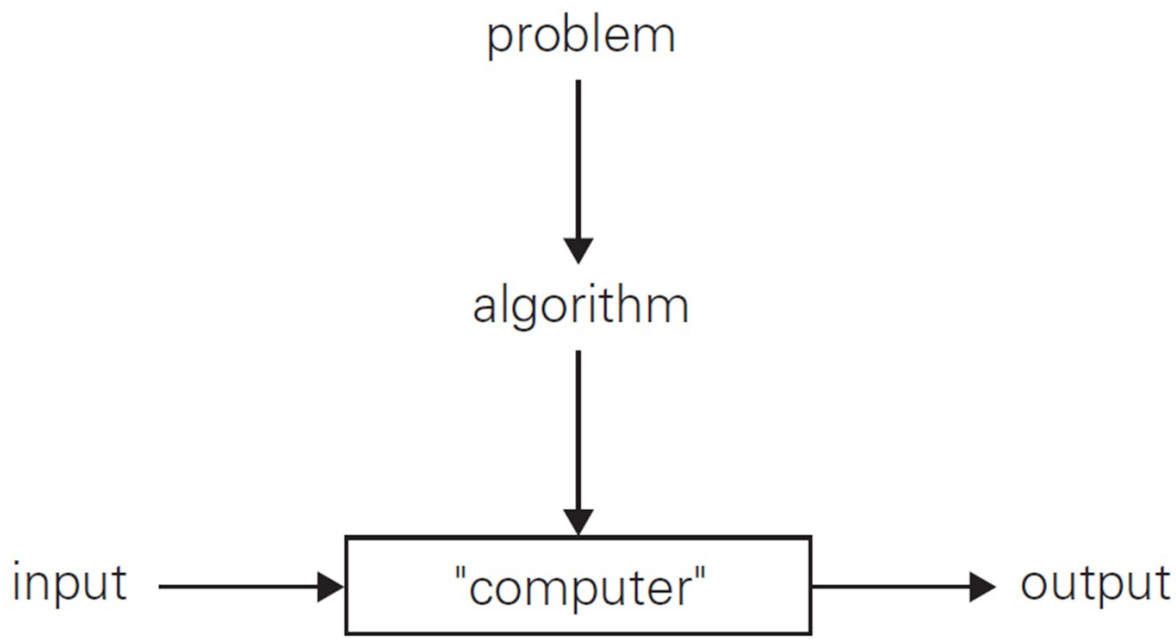
4) правило безпосереднього перероблення;

5) система проміжних результатів;

6) система кінцевих результатів;

7) правило виводу;

8) правило закінчення.


problem
algorithm
input
"computer"
output

1.1.4 Параметри алгоритму *(на основі публікацій проф. Черкаського М.В.)*

У своїх роботах проф. Черкаський також часто оперує поняттям задачі.

Задача – це сформульоване намагання, яке за набором вхідних даних і умов за допомогою деякого алгоритму дозволяє отримати результат, що повністю відповідає цьому набору вхідних даних і умов:

M: y1, y2,…ym ← x1, x2,…xn
, де M – умови; x1, x2,…xn – вхідні дані; y1, y2,…ym – результати; ← – невизначений алгоритм.

1.1.4 Параметри алгоритму *(на основі публікацій проф. Черкаського М.В.)*

Тоді параметричну модель алгоритму можна доповнити до пари задача-алгоритм(рис 1.3).

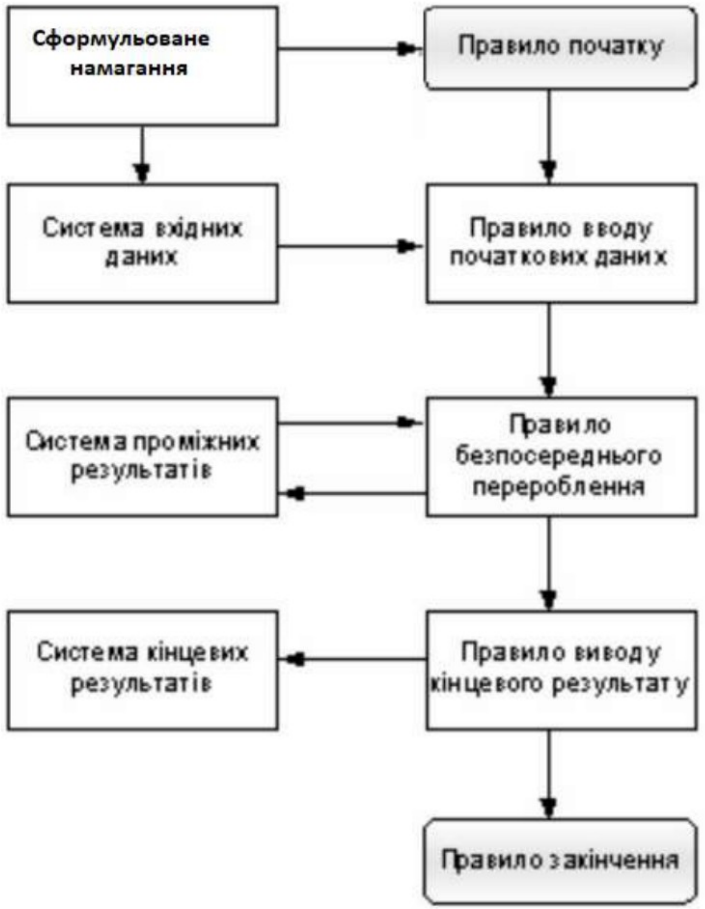


Рис. 1.3.

1.1.5 Базові структури алгоритмів *(алгоритмічні конструкції)*. Теорема Бьома-Якопіні.

Можна виділити такі базові структури алгоритмів:

- лінійні,
- Розгалужені,
- циклічні

та змішані.

1.1.5 Базові структури алгоритмів*(алгоритмічні конструкції)*. Теорема Бьома-Якопіні.

Лінійні алгоритми

Лінійним називається алгоритм(або фрагмент алгоритму), в якому окремі команди виконуються послідовно оді за одними, не залежно від значень вхідних даних і проміжних результатів.

Розгалужений алгоритм

Часто хід обчислювального процесу залежить від вихідних даних або проміжних результатів і алгоритм реалізується в одному з декількох, заздалегідь передбачених

1.1.5 Базові структури алгоритмів*(алгоритмічні конструкції)*. Теорема Бьома-Якопіні.

Лінійні алгоритми

Лінійним називається алгоритм(або фрагмент алгоритму), в якому окремі команди виконуються послідовно оді за одними, не залежно від значень вхідних даних і проміжних результатів.

Розгалужений алгоритм

Часто хід обчислювального процесу залежить від вихідних даних або проміжних результатів і алгоритм реалізується в одному з декількох, заздалегідь передбачених (можливих) напрямків. Такі напрямки часто називаються гілками. Кожна гілка може бути будь-якого ступеня складності, а може взагалі не містити команд, тобто бути виродженою. Вибір тієї або іншої гілки здійснюється в залежності від результату перевірки умови з конкретними даними. У кожному випадку алгоритм реалізується тільки по одній гілці, а виконання інших виключається.

1.1.5 Базові структури алгоритмів*(алгоритмічні конструкції)*. Теорема Бьома-Якопіні.

Циклічний алгоритм

Виконання деякої частини алгоритму багаторазово при різних значеннях деяких змінних називається циклічним обчислювальним процесом. Циклами називаються повторювані ділянки обчислень. Для організації циклів необхідно: задати початкове значення змінної, що визначає цикл (параметр циклу), змінювати цю змінну перед кожним повторенням циклу, перевіряти умову продовження (закінчення) циклу. У циклічних алгоритмах виконання деяких операторів (груп операторів) здійснюється багаторазово з тими ж або модифікованими даними.

У залежності від способу організації кількості повторень циклу розрізняють три типи циклів:

1) цикл із заданою умовою закінчення роботи (ЦИКЛ - ДО);

2) цикл із заданою умовою продовження роботи (ЦИКЛ - ПОКИ);

3) цикл із заданою умовою повторень роботи (ЦИКЛ З ПАРАМЕТРОМ).

1.1.5 Базові структури алгоритмів*(алгоритмічні конструкції)*. Теорема Бьома-Якопіні.

Теорема про структуроване програмування Бьома–Якопіні є результатом теорії мов програмування. Вона стверджує, що клас control-flow graphs (графивів потоку керування), для якого історично склалася назва блок-схеми алгоритмів (аналогічне англомовне поняття flowcharts, яке в графічному поданні має деякі відмінності), може обчислювати будь-яку обчислювану функцію, якщо підпрограми поєднуються лише трьома базовими способами (структури керування). Це:

- виконання однієї підпрограми, а потім іншої підпрограми(послідовність, англ. ***sequence***);
- Виконання однієї з двох підпрограм за значенням булевого виразу(вибір, англ. ***selection***);
- повторне виконання підпрограми, доки логічний вираз є істинним(ітерація, англ. ***iteration***).

1.1.6 Рекурсивні алгоритми

Рекурсія – це визначення поняття або процесу за допомогою його простішої або попередньої версії.

Рекурсивний алгоритм - це алгоритм, який використовує власний виклик(або декілька викликів) для розв'язання проблеми шляхом розбиття її на менші підзадачі того ж типу. Зазвичай розглядаються рекурсивні алгоритми, в яких рекурсія закінчується, тобто є умова виходу.

Розробці рекурсивних алгоритмів передує рекурсивна тріада – етапи моделювання завдання, на яких визначається набір параметрів і співвідношень між ними. Рекурсивну тріаду складають ***параметризація***, ***виділення бази*** і ***декомпозиція***.

1.1.6 Рекурсивні алгоритми

На етапі ***параметризації*** з постановки задачі виділяються параметри, які описують вхідні дані. При цьому деякі подальші розробки рішення можуть вимагати введення додаткових параметрів, які не обумовлені в умови, але використовуються при складанні залежностей. Необхідність у додаткових параметрах часто виникає також при вирішенні завдань оптимізації рекурсивних алгоритмів, в ході яких скорочується їхня тимчасова складність.

Виділення ***бази рекурсії*** припускає знаходження в розв'язуваній задачі тривіальних випадків, результат для яких очевидний і не потребує проведення розрахунків. Вірно знайдена база рекурсії забезпечує завершеність рекурсивних звернень, які в кінцевому підсумку зводяться до базового випадку. Перевизначення бази або її динамічне розширення в ході рішення задачі часто дозволяють оптимізувати рекурсивний алгоритм за рахунок досягнення базового випадку за більш короткий шлях звернень.

1.1.6 Рекурсивні алгоритми

Декомпозиція являє собою зведення загального випадку до більш простих підзадач, які відрізняються від вихідної задачі набором вхідних даних. Декомпозиційні залежності описують не тільки зв'язок між задачам і підзадачами, а й характер зміни значень параметрів на черговому кроці. Від обраних відносин залежить трудомісткість алгоритму, так як для однієї і тієї ж задачі можуть бути складені різні залежності. Перегляд відносин декомпозиції доцільно проводити комплексно, тобто паралельно з коригуванням параметрів і аналізом базових випадків.

1.1.6 Рекурсивні алгоритми

Ітераційна версія:

```
НСД( a, b )
      поки b ≠ 0
           c = остача від ділення a на b
           a = b
           b = c
      повернути a
```

Рекурсивна версія:

```
НСД( a, b )
      якщо b == 0
         поверни a
      інакше
         повернути НСД( b, остача від ділення a на b )
```

1.1.6 Рекурсивні алгоритми

Ітераційна версія алгоритму .

```
unsigned long long int f2_GCD(unsigned long long int a, unsigned long long int b){
        for(unsigned long long int aModB;
                aModB = a % b,
                a = b,
                b = aModB;
        );

        return a;
}
```

Рекурсивна версія алгоритму.

```
unsigned long long int f3_GCD(unsigned long long int a, unsigned long long int b){
    if(!b){
            return a;
        }

        return f3_GCD(b, a % b); // else
}
```

1.1.7 Паралельні алгоритми

Паралельний алгоритм – це алгоритм, який може виконувати кілька операцій за одиницю час.

Якщо «звичайні» алгоритми можна описувати в моделі абстрактної машини з довільним *доступом (англ. random-access machine, RAM)*, то подібним чином можна описувати паралельні алгоритми в моделі абстрактної паралельної машини з довільним доступом *(англ. Parallel random-access machine, PRAM)*. PRAM-машина працює на основі спільної пам’яті. Інші моделі можуть використовувати передачу абстракцію передачі повідомлень замість спільної пам’яті.

1.1.8 Недетерміновані алгоритми

Недетермінований алгоритм – це алгоритм, який передбачає декілька шляхів обробки одних і тих самих вхідних даних, без будь-якого уточнення який саме варіант буде обраний.

Недетермінована мова програмування — це мова, яка може вказувати в певних точках програми(так звані «точки вибору») різні альтернативи для виконання програми. На відміну від конструкції if-then-else(або switch-case), метод вибору між цими альтернативами безпосередньо не вказується програмістом; програма повинна вирішувати під час виконання між альтернативами за допомогою деякого загального методу, застосованого до всіх точок вибору. Програміст визначає обмежену кількість альтернатив, але пізніше програма повинна вибрати між ними. (Насправді «вибір» є типовою назвою для недетермінованого оператора.) Може бути сформована ієрархія точок вибору, причому вибори вищого рівня ведуть до розгалужень, які містять у собі вибори нижчого рівня.

1.1.9 Імовірнісні алгоритми (увипадковлені алгоритми)

Імовірнісні алгоритми *(Probabilistic algorithms)* та рандомізовані (увипадковлені) алгоритми *(Randomized algorithms)*.

Увипадковлені алгоритми (англ. randomized algorithms) — це алгоритми, які використовують елемент випадковості як частину своєї логіки.

Іноді розрізняють увипадковлені алгоритми та ***ймовірнісні алгоритми (англ. probabilistic algorithms)***, які, залежно від випадкового входу, можуть видати некоректний результат (приміром алгоритм Монте-Карло) або зазнати невдачі в його отриманні (приміром алгоритм Лас-Вегаса).

У другому випадку (коли випадковий вхід може призводити до випадкового виконання і випадкового виходу) використання терміну «алгоритм» вже під питанням. У разі випадкового виходу, формально це вже не алгоритм. Однак, у деяких випадках, ймовірнісні алгоритми є єдиним практичним способом розв'язання проблеми.

Реальні комп’ютерні системи увипадковлений алгоритм реалізовують із використанням генератора псевдовипадкових чисел замість генератора дійсно випадкових чисел, тому реальна поведінка може відрізнтись від очікуваної теоретично.