підготував старший викладач каф. ЕОМ Козак Назар Богданович

Практичне заняття № 2
(за темою лабораторної роботи №1)

Здавна найбільшу увагу приділяли дослідженням алгоритму з метою мінімізації часової складності розв'язання задач. Але зміст складності алгоритму не обмежується однією характеристикою. В ряді випадків не менше значення має складність логіки побудови алгоритму, різноманітність його операцій, зв'язаність їх між собою. Ця характеристика алгоритму називається програмною складністю. В теорії алгоритмів, крім часової та програмної складності, досліджуються також інші характеристики складності, наприклад, місткісна, але найчастіше розглядають дві з них – часову і програмну. Якщо у кінцевому результаті часова складність визначає час розв'язання задачі, то програмна складність характеризує ступінь інтелектуальних зусиль, що потрібні для синтезу алгоритму. Вона впливає на витрати часу проектування алгоритму.

Вперше значення зменшення програмної складності продемонстрував аль-Хорезмі у своєму трактаті "Про індійський рахунок". Алгоритми реалізації арифметичних операцій, описані аль-Хорезмі у словесній формі, були першими у позиційній десятковій системі числення. Цікаво спостерігати, як точно і послідовно описує він алгоритм сумування, користуючись арабською системою числення і кільцем (нулем). В цьому опису є всі параметри алгоритму. Це один з перших відомих у світі вербальних арифметичних алгоритмів.

Схема розроблення будь-якого об'єкту складається з трьох операцій: синтез, аналіз та оптимізація *(Рис.1.)*.

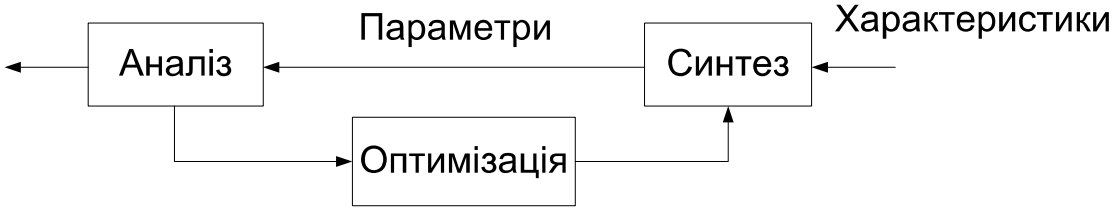


Рис. 1.

Існує два види синтезу: структурний і параметричний. Вихідними даними для структурного синтезу є параметри задачі – сформульоване намагання і набори вхідних даних. В результаті структурного синтезу отримують алгоритм, який розв'язує задачу в принципі.

Параметричний синтез змінами параметрів створює таку його структуру, яка дозволяє зменшити часову складність попередньої моделі. Існує багато способів конструювання ефективних алгоритмів на основі зміни параметрів. Розглянемо спосіб зміни правила безпосереднього перероблення на прикладі задачі знаходження найбільшого спільного дільника двох натуральних чисел..

Алгоритм знаходження найбільшого спільного дільника, яким ми користуємось для цієї цілі і донині, був запропонований Евклідом приблизно в 1150 році до н.е. у геометричній формі, в ньому порівняння величин проводилося відрізками прямих, без використання арифметичних операцій Алгоритм розв'язку передбачав повторювання віднімання довжини коротшого відрізка від довжини довшого.

Опис алгоритму. Маючи два натуральні числа *a* та *b*, повторюємо *обчислення* для пари значень *b* та залишку від ділення *a* на *b* (тобто *a* mod *b*). Якщо *b*=0, то *a* є шуканим НСД.

Обчислення

Ітераційна версія:

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

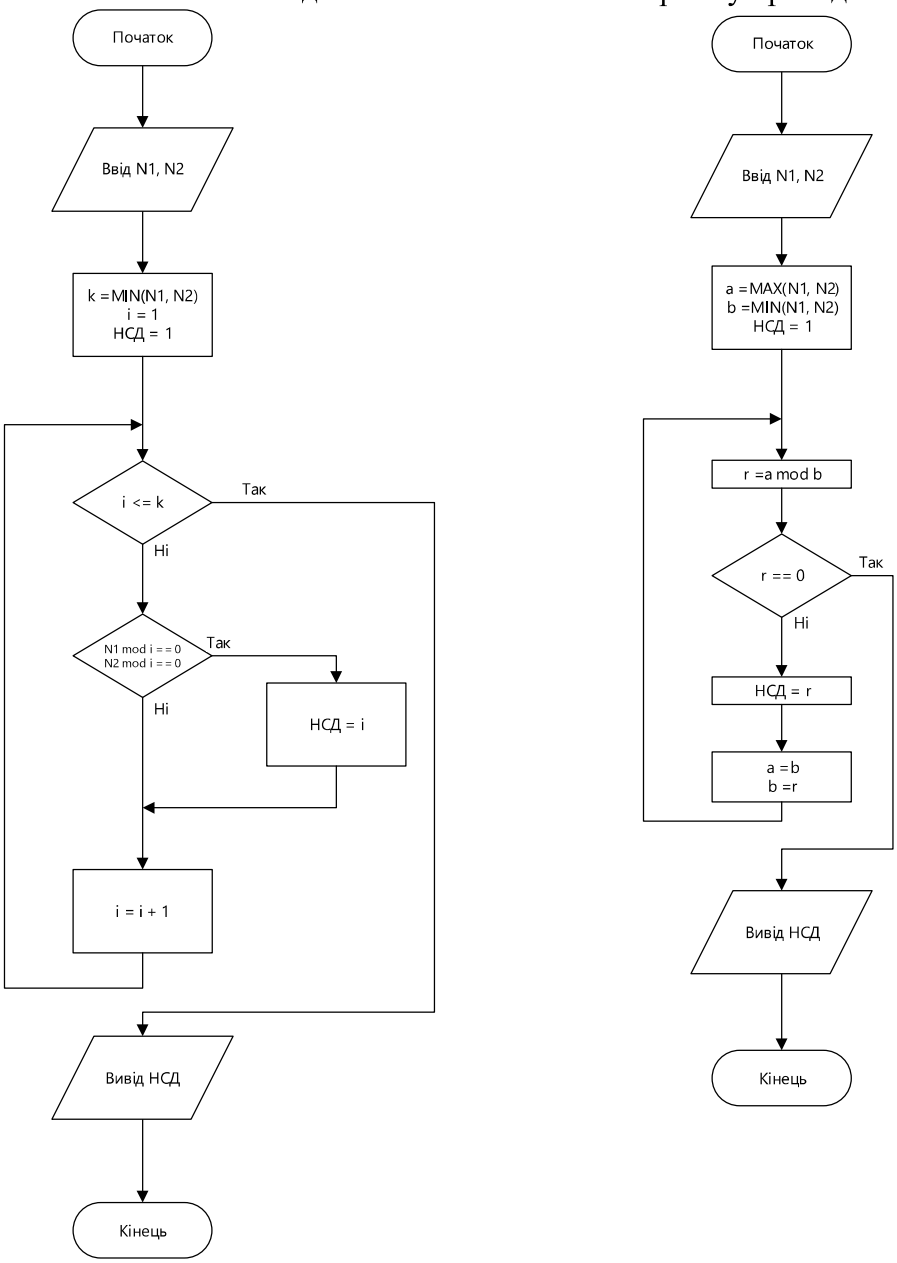
Рекурсивна версія:

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

Для того, щоб довести ефективність алгоритму, потрібно порівняти його з таким, який приймається за неефективний. Прикладом такого неефективного алгоритму є процедура послідовного перебору можливих розв'язань задачі. Будемо вважати, що алгоритм перебору утворений в результаті структурного синтезу, на основі вхідних даних та намагання знайти серед всіх допустимих чисел такє, що є найбільшим дільником двох заданих чисел.

Ефективність, як правило, визначається такою характеристикою як часова складність, що вимірюється кількістю операцій, необхідних для розв'язання задачі.

Дослідимо розв'язання задачі знаходження найбільшого спільного дільника двох цілих чисел ($N_1>0$, $N_2>0$, $N_1 \geq N_2$) алгоритмом перебору і алгоритмом Евкліда. Алгоритм перебору заснований на операції інкременту змінної n від одиниці до меншого (N_2) з двох заданих чисел і перевірці, чи ця змінна є дільником заданих чисел. Якщо це так, то значення змінної запам'ятовується і операції алгоритму продовжуються. Якщо ні, то операції алгоритму продовжуються без запам'ятовування. Операції алгоритму закінчуються видачею з пам'яті знайденого останнім спільного дільника. Блок-схема алгоритму приведена на *рис.2(а)*.



а) б)

Рис2. Блок-схема алгоритму перебору (а) і Евкліда (б).

Адаптований до сучасної арифметики алгоритм Евкліда використовує циклічну операцію ділення більшого числа на менше, знаходження остачі (r) і заміну числа, яке було більшим, на число, яке було меншим, а меншого числа на остачу. Всі перераховані операції виконуються в циклі. Операції циклу закінчуються, коли остача дорівнює нулю. Останній дільник є найбільшим спільним дільником. Блок-схема алгоритму приведена на *рис.2(б)*.

Аналіз цих двох алгоритмів показує, що часова складність алгоритму перебору значно перевищує часову складність алгоритму Евкліда. Для обох алгоритмів часова складність є функцією від вхідних даних, а не їх розміру. В таких випадках при порівнянні ефективності алгоритмів користуються порівнянням часових складностей визначених для найгіршого випадку. Часова складність для найгіршого випадку (Lmax) представляє собою максимальну часову складність серед всіх вхідних даних розміру N.

Часова складність $L_{\max}$ для алгоритму перебору:

$$L_{\max} = C \cdot N_2 \qquad (1)$$

де C – константа , яка дорівнює кількості операцій в кожній ітерації.

Для цілих чисел n $(1 \le n < r_i)$ алгоритм Евкліда знаходження найбільшого спільного дільника має найбільшу часову складність для пари чисел r_{i-1} і r_{i-2} , де 1 , 2 , 3 , ... , r_{i-2}, r_{i-1} , r_i – числа Фібоначчі.

Алгоритм Евкліда є ефективним за часовою складністю у порівняні з алгоритмом перебору. Мінімізація часової складності дозволяє за всіх інших рівних умова збільшити продуктивність розв’язання задачі.