ЛАБОРАТОРНА РОБОТА №1

Назва роботи: алгоритм; властивості, параметри та характеристики складності алгоритму.

Мета роботи: проаналізувати складність заданих алгоритмів.

1. Загальні відомості.

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.)*.

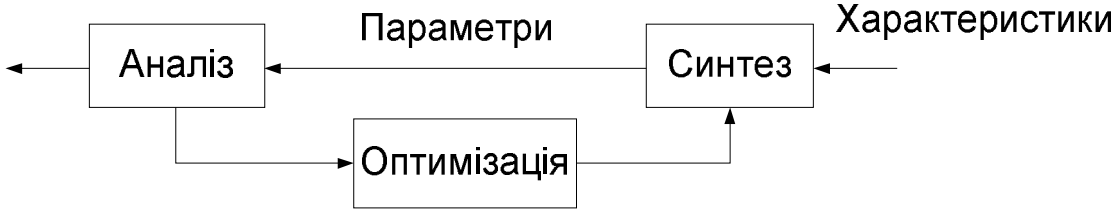


Рис. 1.

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
```

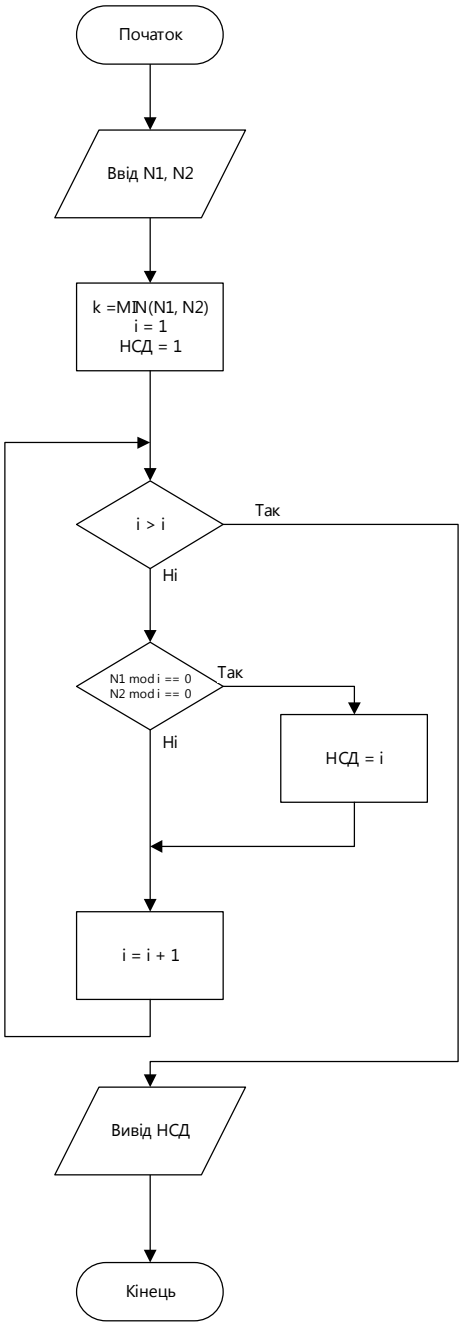
Рекурсивна версія:

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

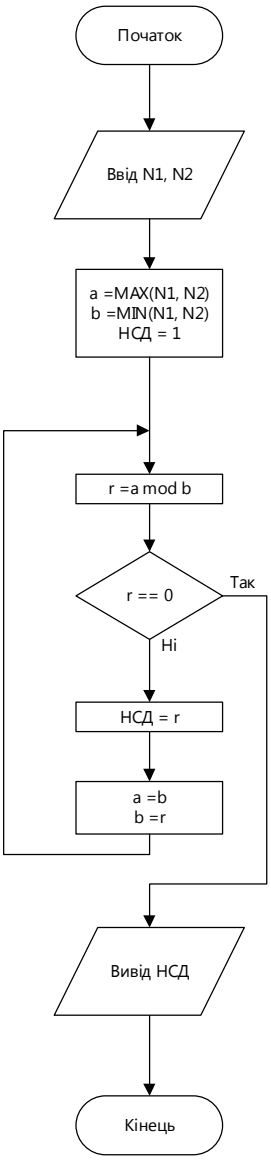
Для того, щоб довести ефективність алгоритму, потрібно порівняти його з таким, який приймається за неефективний. Прикладом такого неефективного алгоритму є процедура послідовного перебору можливих розв’язань задачі. Будемо вважати, що алгоритм перебору утворений в результаті структурного синтезу, на 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.

2. Приклад програми.

Лістинг 2.1

```
//to measure time it is better to run with Linux

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N1  18
#define N2  27

#define MIN(a,b) (((a)<(b))?(a):(b))
#define MAX(a,b) (((a)>(b))?(a):(b))

#define REPEAT_COUNT 1000000
#define REPEATOR(count, code) \
for (unsigned int indexIteration = (count); indexIteration--;){ code; }
#define TWO_VALUES_SELECTOR(variable, firstValue, secondValue) \
       (variable) = indexIteration % 2 ? (firstValue) : (secondValue);

double getCurrentTime(){
       clock_t time = clock();
       if (time != (clock_t)-1) {
              return ((double)time / (double)CLOCKS_PER_SEC);
       }
       return 0.; // else
}

unsigned long long int f1_GCD(unsigned long long int variableN1, unsigned long long int
variableN2){
    unsigned long long int returnValue = 1;

       for(unsigned long long int i = 1, k = MIN(variableN1, variableN2); i <= k; i++){
           if(!(variableN1 % i || variableN2 % i)){
               returnValue = i;
           }
       }

       return returnValue;
}

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;
}

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
}

int main() {
    unsigned long long int vN1 = N1, vN2 = N2, a = MAX(vN1, vN2), b = MIN(vN1, vN2),
        vN1_ = vN1, vN2_ = vN2, a_ = a, b_ = b,
        returnValue;

        double startTime, endTime;

        // f1_GCD
        startTime = getCurrentTime();
        REPEATOR(REPEAT_COUNT,
            TWO_VALUES_SELECTOR(vN1, 16, vN1_);
            TWO_VALUES_SELECTOR(vN2, 4, vN2_);
            returnValue = f1_GCD(a, b);
        )
        endTime = getCurrentTime();
        printf("f1_GCD return %d \r\nrun time: %dns\r\n\r\n",
        returnValue,
        (unsigned int)((endTime - startTime) * (double)(1000000000 / REPEAT_COUNT)));

        // f2_GCD
        startTime = getCurrentTime();
        REPEATOR(REPEAT_COUNT,
            TWO_VALUES_SELECTOR(a, 16, a_);
            TWO_VALUES_SELECTOR(b, 4, b_);
            returnValue = f2_GCD(a, b);
        )
        endTime = getCurrentTime();
        printf("f2_GCD return %d \r\nrun time: %dns\r\n\r\n",
        returnValue,
        (unsigned int)((endTime - startTime) * (double)(1000000000 / REPEAT_COUNT)));

        // f3_GCD
        startTime = getCurrentTime();
        REPEATOR(REPEAT_COUNT,
            TWO_VALUES_SELECTOR(a, 16, a_);
            TWO_VALUES_SELECTOR(b, 4, b_);
            returnValue = f3_GCD(a, b);
        )
        endTime = getCurrentTime();
        printf("f3_GCD return %d \r\nrun time: %dns\r\n\r\n",
        returnValue,
        (unsigned int)((endTime - startTime) * (double)(1000000000 / REPEAT_COUNT)));

        printf("Press any key to continue . . .");
        getchar();

        return 0;
}
```

3. Спрощене завдання.

Скласти програму (C/C++), яка дозволяє провести порівняння трьох алгоритмів(алгоритму перебору, ітераційної версії алгоритму Евкліда та рекурсивної версії алгоритму Евкліда) знаходження НСД за характеристикою часової складності для таких вхідних даних:

Варіант	**Вхідні дані**	
	N1	**N2**
1	112	107
2	146	141
3	166	161
4	186	181
5	206	201
6	226	221
7	246	241
8	266	261
9	286	281
10	306	301
11	326	321
12	346	341
13	366	361
14	386	381
15	406	401
16	426	421
17	446	441
18	466	461
19	486	481
20	506	501
21	526	521
22	546	541
23	566	561
24	586	581
25	606	601
26	626	621
27	646	641
28	666	661
29	686	681
30	706	701

**** Для отримання 50% балів за лабораторну роботу можна використати наявний програмний код** з лістингу 2.1. Для отримання 100% балів за лабораторну роботу потрібно написати власний код.*

4. Завдання базового рівня.

Скласти програму (C/C++), яка реалізовує заданий згідно варіанту алгоритм та дозволяє провести порівняння цього алгоритму з аналогічним алгоритмом прямого перебору за характеристикою часової складності для вхідних даних, що вводяться під час роботи програми.

Варіант	**Алгоритм**
1	Алгоритм Косараджу
2	Алгоритм Борувки
3	Алгоритм Крускала
4	Алгоритм Прима
5	Алгоритм Дейкстри
6	Алгоритм Флойда-Воршала
7	Алгоритм Джонсона
8	Алгоритм Беллмана-Форда
9	Алгоритм Данцига
10	Алгоритм Лі
11	Алгоритм Ахо-Корасік
12	Алгоритм Бояра-Мура
13	Алгоритм Бояра-Мура-Горспула
14	Алгоритм Кнута-Моріса-Прата
15	Алгоритм Коменц-Вальтер
16	Алгоритм Рабіна-Карпа
17	Алгоритм Нідлмана-Вунша
18	Алгоритм Барнса-Хата
19	Алгоритм Діксона
20	Алгоритм Луна
21	Алгоритм Штрассена
22	Алгоритм Копперсміта-Віноградa
23	Алгоритм Пана
24	Алгоритм Біні
25	Алгоритми Шенхаге
26	Алгоритм Малхотри-Кумара-Махешварі
27	Алгоритм Галіла-Наамада
28	Алгоритм Ахьюа-Орліна-Тар'яна
29	Алгоритм Черіяна-Хейджрапа-Мехлхорна
30	Алгоритм Келнера-Мондри-Спілман-Тена

5. Зміст звіту

- Титульний лист;
- Завдання;
- Алгоритм рішення завдання;
- Код програми;
- Екранна форма з результатами роботи програми;
- Висновки.