Ministerul Educaţiei, Culturii și Cercetării  
Universitatea Tehnică a Moldovei

Departamentul Informatică și Ingineria Sistemelor

**RAPORT**

Lucrarea de laborator nr.1  
la Analiza și Proiectarea Algoritmilor

A efectuat:   
st. gr. TI-206 Borș Nicoleta

A verificat: Ernest Bîtca

Chişinău – 2021

**Tema**: Analiza algoritmilor (Timpul de execuţie al algoritmilor)

**Scopul lucrării:** Analiza empirică a algoritmilor

**Sarcina:**

1. Implementați algoritmii propuși într-un limbaj de programare
2. Stabiliți proprietăţile datelor de intrare în raport cu care se face analiza
3. Alegeți metrica pentru compararea algoritmilor
4. Efectuaţi analiza empirică a algoritmilor propuşi
5. Faceți o prezentare grafică a datelor obținute
6. Faceţi o concluzie asupra lucrării efectuate

#### **Întrebări de control:**

#### **Enumeraţi factorii ce influenţează timpul de execuţie al algoritmului.**

#### Timpul de rulare al unui program depinde de următorii factori: datele de intrare, calitatea codului generat de compilator, natura şi viteza de execuţie a instrucţiunilor programului, complexitatea algoritmului care stă la baza programului.

#### **Care sunt etapele analizei empirice?**

#### Etapele analizei empirice sunt:

#### Se stabileşte scopul analizei.

#### Se alege metrica de eficienţă ce va fi utilizată (număr de execuţii ale unei/unor operaţii sau timp de execuţie a întregului algoritm sau a unei porţiuni din algoritm.

#### Se stabilesc proprietăţile datelor de intrare în raport cu care se face analiza (dimensiunea datelor sau proprietăţi specifice).

#### Se implementează algoritmul într-un limbaj de programare.

#### Se generează mai multe seturi de date de intrare.

#### Se execută programul pentru fiecare set de date de intrare.

#### Se analizează datele obţinute

#### **În care cazuri se face analiza empirică a algoritmilor?**

#### Analiza empirică se face:

#### pentru a obţine informaţii preliminare privind clasa de complexitate a unui algoritm;

#### pentru a compara eficienţa a doi (sau mai mulţi) algoritmi destinaţi rezolvării aceleiaşi probleme;

#### pentru a compara eficienţa mai multor implementări ale aceluiaşi algoritm;

#### pentru a obţine informaţii privind eficienţa implementării unui algoritm pe un anumit calculator.

const fib1 = (n) => (n < 2n ? n : fib1(n - 1n) + fib1(n - 2n));

const fib2 = (n) => {

  let i = 1n,

    j = 0n;

  for (let k = 1n; k <= n; k++) {

    j = i + j;

    i = j - i;

  }

  return j;

};

const fib3 = (n) => {

  let i = 1n,

    j = 0n,

    k = 0n,

    h = 1n,

    t = 1n;

  while (n > 0n) {

    if (n % 2n != 0) {

      t = j \* h;

      j = i \* h + j \* k + t;

      i = i \* k + t;

    }

    t = h \* h;

    h = 2n \* k \* h + t;

    k = k \* k + t;

    n = n / 2n;

  }

  return j;

};

let numar = prompt('Introduceti n: ');

let n = BigInt(numar.toString());

*//fib1------------------------------------------------------------------------*

const startfib1 = window.performance.now();

console.log(`Pentru functia fib1: f${numar} = ${fib1(n)}`); *// O(2^n)*

const endfib1 = window.performance.now();

console.log(`Se ruleaza in ${endfib1 - startfib1} ms`);

*//fib2------------------------------------------------------------------------*

const startfib2 = window.performance.now();

console.log(`Pentru functia fib2: f${numar} = ${fib2(n)}`); *//O(n)*

const endfib2 = window.performance.now();

console.log(`Se ruleaza in ${endfib2 - startfib2} ms`);

*//fib3------------------------------------------------------------------------*

const startfib3 = window.performance.now();

console.log(`Pentru functia fib3: f${numar} = ${fib3(n)}`); *//O(log2N)*

const endfib3 = window.performance.now();

console.log(`Se ruleaza in ${endfib3 - startfib3} ms`);

#### Tabelul 1.1 – Rezultatele functiei fib1 pentru n = 10, 40, 55, 100.

|  |  |
| --- | --- |
| Încercarea | Fib1 |
| n = 10 | |
| 1 |  |
| 2 |  |
| 3 |  |
| Media: | 0.20 ms |
| n = 40 | |
| 1 |  |
| 2 |  |
| 3 |  |
| Media: | 17708.76 ms |
| n = 55 | |
| 1 |  |

#### Această metodă este foarte ineficienta, deoarece recalculează de mai multe ori aceleaşi valori. Complexitatea acestui algoritm este de O().

#### Tabelul 1.2 – Numărul de iterații pentru funcția fib3

|  |  |
| --- | --- |
| n | Nr. de iterații |
| 5 | = 32 |
| 10 | = 1024 |
| 40 | = 1099511627776 |
| 55 | = 36028797018963968 |
| 100 | = 1267650600228229401496703205376 |

#### Observăm că acest algoritm este ineficient atât pentru numere mari, cât și pentru numere mici. Cu cât n-ul este mai mare, cu atât mai mare este și numărul de iterații, astfel timpul de execuție crește enorm. Pe lângă aceasta, acest algoritm recalculează de mai multe ori aceleaşi valori.

#### Tabelul 2 – Rezultatele functiei fib2 pentru n = 10, 40, 55, 100.

|  |  |
| --- | --- |
| Încercarea | Fib2 |
| n = 10 | |
| 1 |  |
| 2 |  |
| 3 |  |
| Media: | 0 ms |
| n = 40 | |
| 1 |  |
| 2 |  |
| 3 |  |
| Media: | 0.03 ms |
| n = 55 | |
| 1 |  |
| 2 |  |
| 3 |  |
| Media: | 0.16 |
| n = 100 | |
| 1 |  |
| 2 |  |
| 3 |  |
| Media: | 0.1 |

#### Complexitatea acestui algoritm este de O(n).

#### Tabelul 2.2 – Numărul de iterații pentru funcția fib2

|  |  |
| --- | --- |
| n | Nr. de iterații |
| 5 | 5 |
| 10 | 10 |
| 40 |  |
| 55 |  |
| 100 |  |

#### Tabelul 3 – Rezultatele functiei fib3 pentru n = 10, 40, 55, 100.

|  |  |
| --- | --- |
| Încercarea | Fib3 |
| n = 10 | |
| 1 |  |
| 2 |  |
| 3 |  |
| Media: | 0.1 ms |
| n = 40 | |
| 1 |  |
| 2 |  |
| 3 |  |
| Media: | 0 ms |
| n = 55 | |
| 1 |  |
| 2 |  |
| 3 |  |
| Media: | 0.06 ms |
| n = 100 | |
| 1 |  |
| 2 |  |
| 3 |  |
| Media: | 0.03 ms |

#### Complexitatea acestui algoritm este de O().

#### Tabelul 3.2 – Numărul de iterații pentru funcția fib3

|  |  |
| --- | --- |
| n | Nr. de iterații |
| 5 | = 2,32 |
| 10 | = 3,32 |
| 40 | = 5,32 |
| 55 | = 5,78 |
| 100 | = 6,64 |

#### Observăm că algoritmul utilizat la funcția fib3 deși are mai puține iterații decât celelalte, este cel mai puțin rapid pentru numerele mici, deoarece numărul de operații este mai mare decât al funcției fib2, astfel pentru n = 5 vom avea T(n) 20, pe când la fib2 T(n) 10 , de două ori mai puțin. Pentru numerele mari însă fib3 este mai eficient decât fib2, deoarece va executa cu mai puține iterații.

#### Pentru a observa mai ușor difențele dintre acești trei algoritmi o să analizăm graficele de mai jos:

#### Analizând aceste grafice ne dăm seama că concluziile care le-am tras mai sus sunt corecte. Pe lângă aceasta, timpul de execuție nu depinde doar de algoritmul utlizat și de datele de intrate, în cazul nostru **n**, ci și de perfomanța calculatorului.

#### **Concluzie:**

#### În urma efectuării lucrării de laborator nr.1 la tema “Analiza algoritmilor. Timpul de execuţie al algoritmilor” am observat cât de tare este influențat timpul de execuție de algoritmul utilizat. Pe lângă asta am avut posibilitatea să analizez trei algoritmi pentru deter minarea numărului lui Fibonacci, cât de eficienți sau ineficienți sunt și care este timpul lor de execuție și complexitatea acestora. Mai mult decât atât am pus pus în practică cunoștințele acumulate la orele de curs la disciplina “Analiza și Proiectarea Algoritmilor”. Datorită acestui laborator am aflat și practic ce înseamna analiza empirică a algoritmilor, nu doar teoretic.

#### **Bibliografia:**

#### Else - <https://else.fcim.utm.md/course/view.php?id=7>

#### The Fibonacci numbers - <http://www.maths.surrey.ac.uk/hosted-sites/R.Knott/Fibonacci/fibtable.html>

#### Analiza empirică a algoritmilor - <https://else.fcim.utm.md/pluginfile.php/39289/mod_resource/content/1/L1_apa.pdf>

#### Algoritmi. Big O. Complexitatea Algoritmilor <https://www.youtube.com/watch?v=AtKtaoZDogs>

#### Algoritmi. Notatia T(n). Timpul de executie. Complexitatea Algoritmilor - <https://www.youtube.com/watch?v=_jSTVwoLqaA>