KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

KOMPIUTERIŲ KATEDRA

Algoritmų sudarymas ir analizė labarotorinis darbas NR.2

Atliko:
Dominykas Adomaitis IFIN-8/3

Priėmė:
dėst: Andrius Kriščiūnas

KAUNAS, 2020

Turinys

1. Uždavinys

Duotai rekurentinei formulei

$$F(n) = \begin{cases} F(n-1) + F(n-4) + 9 * F\left(\frac{n}{5}\right) + 1, & kai\ n > 1, \\ 1 & kitais\ atvejais. \end{cases}$$

o Sudarykite du algoritmus ir įvertinkite jų sudėtingumus:

• Tiesiogiai panaudojant rekursiją;

• Panaudojant dinaminio programavimo metodo savybę, kad galime įsiminti dalinius sprendinius.

o Programiškai realizuokite ir eksperimentiškai įvertinkite bei palyginkite abiejų algoritmų sudėtingumą.

1.1 Programos pseudo kodo sudarymas.

a) Galime sudaryti tokį algoritmą panaudojant rekursines funkcijas:

```
//T(n) = F(n-1) + F(n-4) + 9*F(n/5) + 1
static long F1(int n)
{
    long sum = 0;
    if (n <= 1)
        return 1;
    long a = F1(n - 1);
    long b = F1(n - 4);
    for (int i = 0; i < 9; i++)
    {
        sum += F1(n / 5);
    }
    long d = 0;
    return d = a + b + sum + 1;
}
```

Kaina	Kiekis
c1	1
c1	1
c1	1
F1(n-1)	1
F2(n-2)	1
c1	n+1
F1(n/5)	9
c1	1
c1	1

F1 = 5 * c1 + F1(n-1) + F1(n-4) + 9 * F1(n+5) + c1 * n + 1 = O(3^n)

b) Sudarydami dinaminio programavimo algoritmą panaudosime savybę, kad galime įsiminti dalinių sprendinių vertes:

```
//T(n) = F(n-1) + F(n-4) + 9*F(n/5) + 1
static long F2_aux(int n, List<long> r)
{
    long q;
    if (n <= 1)
        return 1;
    //Tikrina ar reiksme yra turimame liste
    //jeigu yra ima is list jei ne eina to
    if (r[n] > int.MinValue)
        return r[n];
    q = int.MinValue;

    long a = F2_aux(n - 1, r);
    long b = F2_aux(n - 4, r);
    long c1 = F2_aux(n / 5, r);
    long c2 = F2_aux(n / 5, r);
    long c3 = F2_aux(n / 5, r);
    long c4 = F2_aux(n / 5, r);
    long c5 = F2_aux(n / 5, r);
    long c6 = F2_aux(n / 5, r);
    long c7 = F2_aux(n / 5, r);
    long c8 = F2_aux(n / 5, r);
    long c9 = F2_aux(n / 5, r);
    q = a + b + c1 + c2 + c3 + c4 + c5 + c6 + c7 + c8 + c9 + 1;
    r[n] = q;
    return q;
}
```

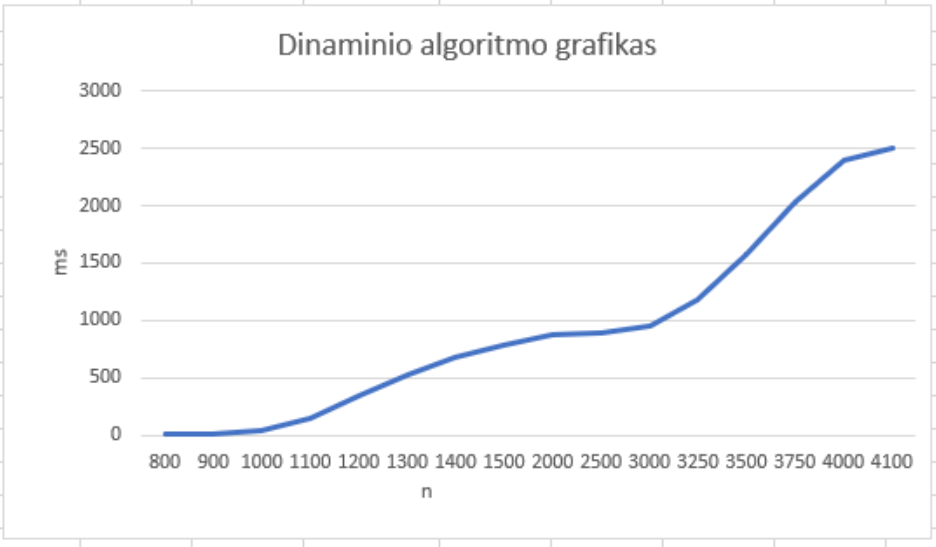
Kaina	Kiekis
c1	1
c1	1
c1	1
c1	
c1	
c1	1
c1	1
F2_aux(n - 1, r)	1
F2_aux(n - 4, r)	1
F2_aux(n / 5, r)	1
F2_aux(n / 5, r)	1
F2_aux(n / 5, r)	1
F2_aux(n / 5, r)	1
F2_aux(n / 5, r)	1
F2_aux(n / 5, r)	1
F2_aux(n / 5, r)	1
F2_aux(n / 5, r)	1
F2_aux(n / 5, r)	1
c1	1
c1	1
c1	1

F2_aux = 10 * c1 + F2_aux(n - 1, r) + F2_aux(n - 4, r) + (9 * F2_aux(n / 5, r) = O(n)

1.2 Eksperimentinis algoritmų sudėtingumo įvertinimas

n	Laikas(ms)
30	5
32	10
34	19
36	35
38	66
40	126
42	240
44	451
46	846
48	1612
50	3079
52	5832
54	12050
56	20991
58	40175

n	Laikas(ms)
800	7
900	18
1000	39
1100	156
1200	349
1300	526
1400	684
1500	792
2000	880
2500	887
3000	950
3250	1176
3500	1569
3750	2038
4000	2399
4100	2509



2. Uždavinys:

Duota n sveikųjų skaičių seka X: $x1$,$x2$,… , xn (gali būti ir neigiamų skaičių). Reikia rasti nuoseklų posekį (negalima praleisti narių), kurio narių suma būtų mažiausia. Tarkime, duota seka X: (2; −5;−2; −5; −4; 4; 2; −2; 5; −5). Tada posekis su mažiausia narių suma: (−5; −2; −5; −4)

2.1 Pradinio uždavinio suskaidymas į mažesnius uždavinius

1) Koks yra mažiausios sumos posekis kiekviename indekse.

2) Koks yra mažiausias sumos posekis visoje sekoje.

2.2 Dinaminio programavimo uždavinio programos pseudo kodo sudarymas.

```
static int smallestSumSubarr(int[] arr, int n)
{
    int last_idx = 0;
    int start_idx = 0;
    int end_idx = 0;

    // to store the minimum value that is ending up to the current index
    int min_ending_here = 2147483647;
    // to store the minimum value encountered so far
    int min_so_far = 2147483647;
    // traverse the array elements
    for (int i = 0; i < n; i++)
    {
        // if min_ending_here > 0, then it could
        // not possibly contribute to the
        // minimum sum further
        if (min_ending_here > 0)
        {
            min_ending_here = arr[i];
            last_idx = i;
        }
        // else add the value arr[i] to
        // min_ending_here
        else
        {
            min_ending_here += arr[i];
        }
        if (min_so_far > min_ending_here)
        {
            min_so_far = min_ending_here;
            start_idx = last_idx;
            end_idx = i;
        }
    }
    return min_so_far;
```

Kaina	Kiekis
c1	1
c1	1
c1	1
c1	1
c1	1
c1	n
c1	n - 1
c1	n - 1
c1	n - 1
c1	n - 1
c1	n - 1
c1	n - 1
c1	n - 1
c1	n - 1
c1	n - 1

smallestSumSubarr = 5*c1 + c1*n + (10*c1*(n-1)) = O(n)

Naudojamas modifikuotas Kadane’s algoritmas. Kiekviename indekse yra ieškomas mažiausia suma turintis posekis kuris baigiasi tame indekse. Mažiausia suma turintis posekis bus arba tame indekse esantis elementas arba esamo indekso elementas sujungtas su prieš tai buvusio indekso mažiausia suma turintčiu posekiu. Taip gaunamas kiekvieno indekso (local) mažiausia suma turintis posekis. Atsakymas gaunamas kai iš visų indeksu mažiausiu posekių (local) išrenkamas mažiausias (global).

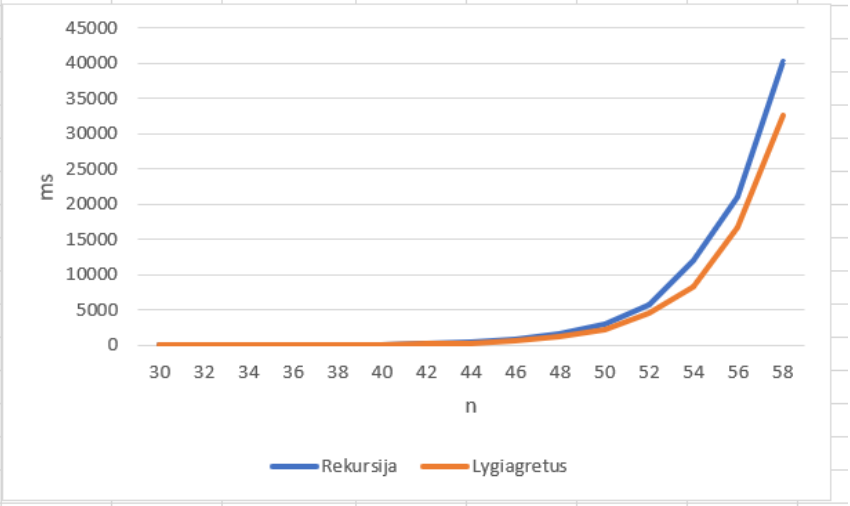
2.3 Eksperimentinis algoritmų sudėtingumų įvertinimas

n	Laikas(ms)
50000000	149
100000000	295
150000000	446
200000000	586
250000000	738
300000000	879
350000000	1033
400000000	1169
450000000	1318
500000000	1504

3. Uždavinys

Panaudojus pirmo uždavinio duotą rekurentinę formulę realizuoti jai algoritmą tiesiogiai panaudojant rekursiją bei lygiagretų programavimą. Eksperimentiškai palyginti vykdymo laikus, kai nenaudojamas lygiagretus programavimas ir naudojamas lygiagretus programavimas, bei paskaičiuoti išlygiagretinimo koeficientą.

n	Rekursija	Lygiagretus
30	5	14
32	10	14
34	19	19
36	35	34
38	66	54
40	126	102
42	240	189
44	451	338
46	846	664
48	1612	1226
50	3079	2299
52	5832	4560
54	12050	8295
56	20991	16751
58	40175	32656



Išvada: Lygiagretus veikia greičiau už rekursija.

4. Priedai

1 ir 3 užduotys

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Diagnostics;

namespace Tesiogine_rekursija
{
    class Program
    {
        static void Main(string[] args)
        {
            int n = 50;
            long laikas;
            Stopwatch watch = Stopwatch.StartNew();

            Console.WriteLine("Kiekis n: " + n);
            Console.WriteLine("Rekursija: " + F1(n));
            watch.Stop();
            laikas = watch.ElapsedMilliseconds;
            Console.WriteLine(laikas + "ms" + "\n");

            watch.Reset();
            watch.Start();
            Console.WriteLine("Dinaminis: " + F2(n));
            watch.Stop();
            laikas = watch.ElapsedMilliseconds;
            Console.WriteLine(laikas + "ms" + "\n");

            watch.Reset();
            watch.Start();
            Console.WriteLine("Lygiagretus: " + F3(n));
            watch.Stop();
            laikas = watch.ElapsedMilliseconds;
            Console.WriteLine(laikas + "ms" + "\n");

        }


        //T(n) = F(n-1) + F(n-4) + 9*F(n/5) + 1
        static long F1(int n)
        {
            long sum = 0;
            if (n <= 1)
                return 1;
            long a = F1(n - 1);
            long b = F1(n - 4);
            for (int i = 0; i < 9; i++)
            {
                sum += F1(n / 5);
            }
            long d = 0;
            return d = a + b + sum + 1;
        }

        //T(n) = F(n-1) + F(n-4) + 9*F(n/5) + 1
        static long F2_aux(int n, List<long> r)
        {
```

```
    long q;
    if (n <= 1)
        return 1;
    //Tikrina ar reiksme yra turimame liste
    //jeigu yra ima is list jei ne eina toliau
    if (r[n] > int.MinValue)
        return r[n];
    q = int.MinValue;

    long a = F2_aux(n - 1, r);
    long b = F2_aux(n - 4, r);
    long c1 = F2_aux(n / 5, r);
    long c2 = F2_aux(n / 5, r);
    long c3 = F2_aux(n / 5, r);
    long c4 = F2_aux(n / 5, r);
    long c5 = F2_aux(n / 5, r);
    long c6 = F2_aux(n / 5, r);
    long c7 = F2_aux(n / 5, r);
    long c8 = F2_aux(n / 5, r);
    long c9 = F2_aux(n / 5, r);
    q = a + b + c1 + c2 + c3 + c4 + c5 + c6 + c7 + c8 + c9 + 1;
    r[n] = q;
    return q;
}

static long F2(int n)
{
    if (n <= 1)
        return 1;
    List<long> r = new List<long>();
    for (int i = 0; i <= n; i++)
    {
        r.Add(int.MinValue);
    }
    return F2_aux(n, r);
}

class CustomData
{
    public long TNum;
    public long TResult;
}

//T(n) = F(n-1) + F(n-4) + 9*F(n/5) + 1
static long F3(int n)
{
    long funkc = 0;
    if (n < 6) funkc = F1(n);
    else
    {
        int countCPU = 3;
        Task[] tasks = new Task[countCPU];
        tasks[0] = Task.Factory.StartNew(
            (Object p) =>
            {
                var data = p as CustomData; if (data == null) return;
                data.TResult = F1(n - 1);
            },
                new CustomData() { TNum = 0 });


        tasks[1] = Task.Factory.StartNew(
```

```
                    (Object p) =>
                    {
                        var data = p as CustomData; if (data == null) return;
                        data.TResult = F1(n - 4);
                    },
                        new CustomData() { TNum = 0 });

                tasks[2] = Task.Factory.StartNew(
                    (Object p) =>
                    {
                        var data = p as CustomData; if (data == null) return;
                        data.TResult = F1(n / 5);
                    },
                        new CustomData() { TNum = 0 });

                Task.WaitAll(tasks);
                funkc = (tasks[0].AsyncState as CustomData).TResult
                    + (tasks[1].AsyncState as CustomData).TResult
                    + 9 * (tasks[2].AsyncState as CustomData).TResult + 1;
            }
            return funkc;
        }
    }
}
```

2 užduotis

```
using System;
using System.Diagnostics;

class GFG
{
    static int smallestSumSubarr(int[] arr, int n)
    {
        int last_idx = 0;
        int start_idx = 0;
        int end_idx = 0;

        // to store the minimum value that is ending up to the current index
        int min_ending_here = 2147483647;
        // to store the minimum value encountered so far
        int min_so_far = 2147483647;
        // traverse the array elements
        for (int i = 0; i < n; i++)
        {
            // if min_ending_here > 0, then it could
            // not possibly contribute to the
            // minimum sum further
            if (min_ending_here > 0)
            {
                min_ending_here = arr[i];
                last_idx = i;
            }
            // else add the value arr[i] to
            // min_ending_here
            else
            {
                min_ending_here += arr[i];
            }

            if (min_so_far > min_ending_here)
```

```
            {
                min_so_far = min_ending_here;
                start_idx = last_idx;
                end_idx = i;
            }
        }

        /*
        //Console.WriteLine("Start index = " + start_idx);
        //Console.WriteLine("End index = " + end_idx);
        Console.WriteLine("\n\nSmallest sum contiguous subarray ");
        for (int i = start_idx; i <= end_idx; i++)
            Console.Write(arr[i] + " ");
        */

        return min_so_far;
    }

    public static void Main()
    {
        int n = 500000000;
        int[] Num = new int[n];
        Random RandomNumber = new Random();
        for (int i = 0; i < n; i++)
        {
            Num[i] = RandomNumber.Next(-10000, 10000);
        }
        Console.WriteLine(n);
        long laikas;
        Stopwatch watch = Stopwatch.StartNew();

        /*
        int[] Num = { 4, 2, -2, 5, -5, -2, -5, -4, 4, 2, -2, 2, -5,
            -2, -5, -4, 4, 2, -2, 5, -5 };
        //int n = Num.Length;

        Console.WriteLine("Original array");
        for (int i = 0; i < n; i++)
            Console.Write(Num[i] + " ");
        */
        watch.Start();
        Console.WriteLine("\n\nSmallest sum: " + smallestSumSubarr(Num, n));
        watch.Stop();
        laikas = watch.ElapsedMilliseconds;
        Console.WriteLine(laikas + "ms" + "\n");
    }
}
```