KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Algoritmų sudarymas ir analizė(P170B400)

Laboratorinio darbo nr. 2 ataskaita

Atliko:

IFF-8/13 grupės studentas

Mykolas Paulauskas

Kaunas 2020

1. Užduotis

Duotai rekurentinei formulei:

12.

$$W[i,j] = \begin{cases} 0 & i = 0 \text{ or } j = 0, \\ W[i-1, j-1] & \text{if } x_i = y_j \leq 0, \\ x_i + W[i-1, j-1] & \text{if } x_i = y_j > 0, \\ \max\{W[i-1, j], W[i, j-1]\} & \text{otherwise.} \end{cases}$$

Sudaryti rekursinį ir dinaminį algoritmą. Sudarytus algoritmus programiškai realizuoti ir eksperimentiškai palyginti jų veikimo laikus ir sudėtingumą.

1.1. Programos pseudo kodo sudarymas

a) Naudojantis rekursines funkcijas galime sudaryti tokį algoritmą.

```
static int MWCS_Recurssive(int[] X, int[] Y, int m, int n)
{
   if (m == 0 || n == 0)
       return 0;
   if (X[m - 1] == Y[n - 1])
       return X[m - 1] + MWCS_Recurssive(X, Y, m - 1, n - 1);
   else
       return Math.Max(MWCS_Recurssive(X, Y, m, n - 1), MWCS_Recurssive(X, Y, m - 1, n));
}
```

b) Naudojantis dinaminio programavimo savybę įsiminti tarpinius duomenis sudaromas šis algoritmas

```
public static int[,] MWCS_BottomUp(int[] X, int[] Y)
        {
            int m = X.Length;
            int n = Y.Length;
            int[,] W = new int[m,n];

            for (int i = 1; i < m; i++)
            {
                for (int j = 1; j < n; j++)
                {
                    if(X[i] == Y[j])
                    {
                        if(X[i] < 0)
                        {
                            W[i, j] = W[i - 1, j - 1];
                        }
                        else
                        {
                            W[i, j] = X[i] + W[i - 1, j - 1];
                        }
                    }
                    else
                    {
                        W[i, j] = Math.Max(W[i, j - 1], W[i - 1, j]);
```

```
                }
            }
        }

        return W;
    }
```

Iš kodo matosi, algoritme yra sukami du ciklai, vienas m kartų, kitas n kartų. Todėl be sudėtingų skaičiavimų galima darytis išvadą, jog šio algoritmo sudėtingumas yra O(nm).

1.2. Eksperimentinis algoritmų sudėtingumo įvertinimas ir algoritmo veikimas

Pagal rekurentinę formulę ir detalesnį jos aprašymą suprantama, jog iš dviejų duotų teigiamų sveikų skaičių masyvų reikia atrinkti sunkiausią pasikartojančią seką. Algoritmo veikimą galima realizuoti tokioje lentelėje su pavyzdiniais duomenimis X = `{ 1, 5, 5, 2, 1, 8 }`, Y = `{ 5, 3, 2, 3, 1, 6, 8 }`:

	1	5	5	2	1	8
5	-	-	5	-	-	-
3	-	-	-	-	-	-
2	-	-	-	7	-	-
3	-	-	-	-	-	-
1	-	-	-	-	8	-
6	-	-	-	-	-	-
8	-	-	-	-	-	16

lentelė 1 Algoritmo veikimas

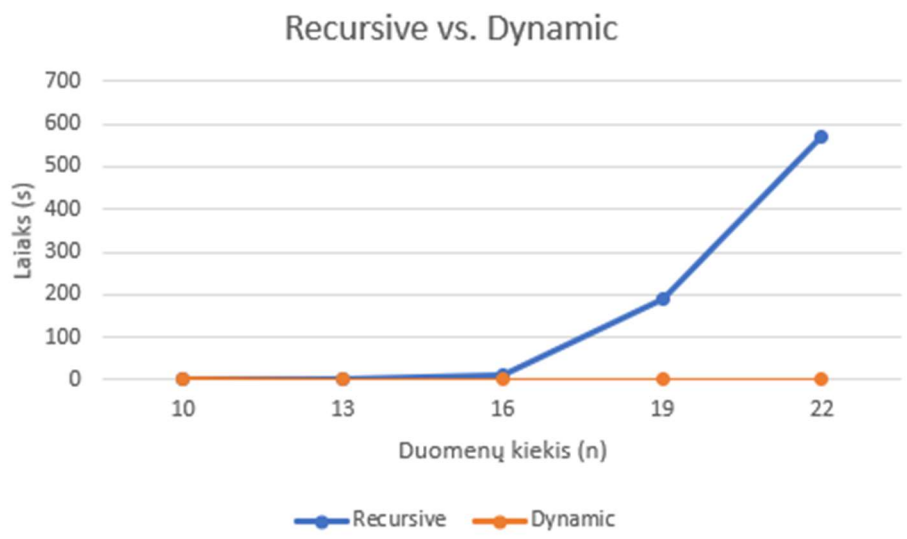
```
Uzduotis nr. 1

Dynamic: 16
Recursive: 16
```

pav. 1 Algoritmo veikimo rezultatas programoje

	Recursive	Dynamic
10	0.0041077	0.0009382
13	0.1856799	0.0000347
16	13.085935	0.0000469
19	190.850935	0.0000369
22	572.552804	0.0001845

pav. 2 Algoritmo greičio lentelė



pav. 3 Algoritmų greičio grafikas

2. Užduotis

Programiškai analizuoti ir įvertinti pateiktos užduoties algoritmą:

Įmonė savo klientams pardavinėja įvairiausio ilgio medieną. Vieni klientai perka trumpus rąstus malkoms, kiti ilgesnius baldų gamybai, treti labai ilgus namų statybai ir pan. Įmonė nori optimizuoti savo veiklą ir pardavinėti atitinkamai supjautus rąstus iš kurių pelnas būtų didžiausias. Pvz.: jei rąsto ilgis 5m, tai kaip reikia supjaustyti rąstą, kad pardavimo pelnas būtų didžiausias. Žinoma, kad už 1m rąstą mokama 1Eur, už 2m – 4 Eur, už 5m – 5 Eur. Ats.: geriausia supjaustyti rąstą po 2m, o uždirbtas pelnas bus 9 Eur (4+4+1)

2.1.Programog kodas

```
static (int, string) CutRod(int[] price, int n)
        {
            int[] val = new int[n + 1];
            val[0] = 0;
            string combination = "";

            for (int i = 1; i <= n; i++)
            {
                int max_val = int.MinValue;
                for (int j = 0; j < i; j++)
                    max_val = Math.Max(max_val, price[j] + val[i - j - 1]);
                val[i] = max_val;
            }

            return (val[n], combination);
          }
```

2.2.Algoritmo veikimas ir sudėtingumo įvertinimas

```
Uzduotis nr. 2

Didziausias pelnas: 9
```

pav. 4 Algoritmo veikimo pavyzdys su duotais duomenimis užduotyje

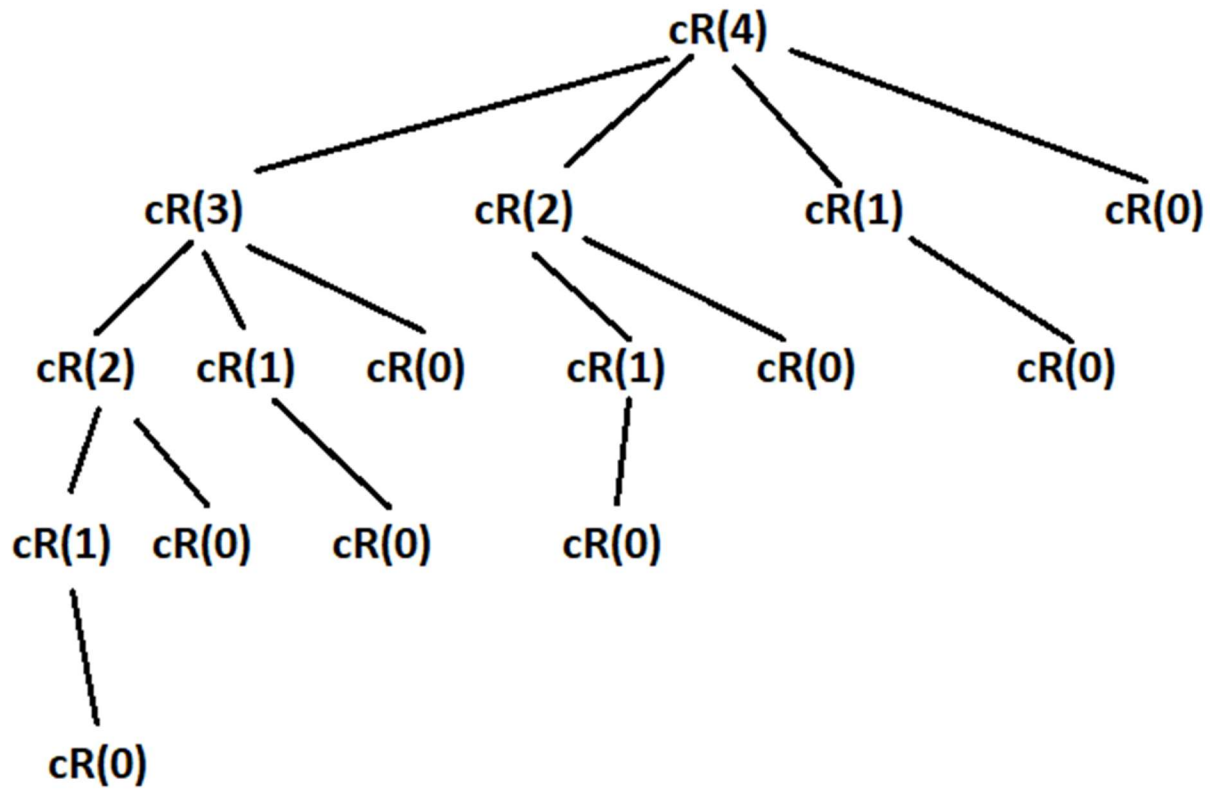
Algoritmo veikimas su šiais duomenimis:

```
length   | 1   2   3   4   5   6   7   8
-----------------------------------------
price    | 1   5   8   9  10  17  17  20
```

```
Uzduotis nr. 2

Didziausias pelnas: 22
```

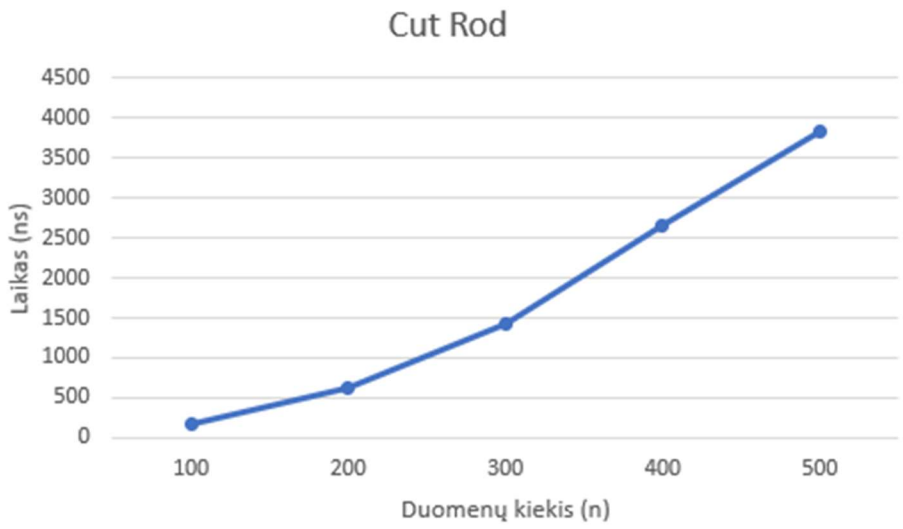
pav. 5 Algoritmo veikimas



pav. 6 Problemos išskaidymas į mažesnes problemas

	Cut Rod
100	167
200	629
300	1416
400	2645
500	3841

pav. 7 Algoritmo greičio lentelė



pav. 8 Algoritmo greičio grafikas

Iš kodo matosi, kad algoritme yra sukami du ciklai, dėl to be sudėtingų skaičiavimų galima teigti, jog šio algoritmo sudėtingumas yra O(n^2)

3. Užduotis

1 užduoties rekursinio algoritmo vykdymo paskirstymas skirtingiems procesoriaus branduoliams.

3.1.Kodas

```
static int MWCS_Parallel(int[] X, int[] Y, int[,] W)
        {
            int m = X.Length;
            int n = Y.Length;

            Parallel.For(1, m,
                i =>
                {
                for (int j = 1; j < n; j++)
                {
                    if (X[i] == Y[j])
                    {
                        if (X[i] < 0)
                        {
```

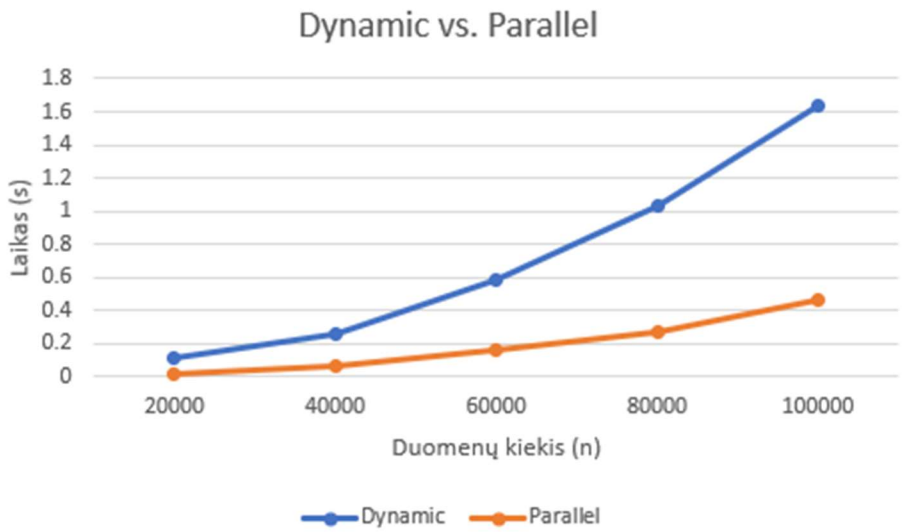
```
                    W[i, j] = W[i - 1, j - 1];
                }
                else
                {
                    W[i, j] = X[i] + W[i - 1, j - 1];
                }
            }
            else
            {
                W[i, j] = Math.Max(W[i, j - 1], W[i - 1, j]);
            }
        }
    });

    return W[m - 1, n - 1];
}
```

3.2. Efektyvumas

	Dynamic	Parallel
20000	0.1129874	0.018469
40000	0.2626103	0.0699245
60000	0.5813344	0.154915
80000	1.0269078	0.2679228
100000	1.6328831	0.4595632

pav. 9 Algoritmų greičio palyginimas



pav. 10 Algoritmų greičio palyginimo grafikas

4. Išvados

Parašyti ir įgyvendinti visi algoritmai. Iš šio laboratorinio darbo aiškus rezultatas, jog algoritmus galima optimizuoti naudojantis dinaminio programavimo savybes. Pirmoje užduotyje buvo sunku aiškintis kaip

iš medžiagoje duoto dinaminio algoritmo perdaryti į rekursinį. Tačiau ieškodamas daugiau informacijos viskas pavyko be didesnių sunkumų. Antroje užduotis buvo atlikta nesunkiai. Trečioje užduotyje buvo susidurta su problemomis paralelizuojant algoritmo veikimą. Joje naudojau max() funkciją ir jinai vykstant paraleliems procesams skaičiavimus vykdė neteisingai, nes neturėjo pilnos reikiamos informacijos. Šią klaidą ištaisius likusi užduotis buvo atlikta be problemų. Laboratorinio darbo metu daugiau sužinojau ir labiau supratau rekursinius metodus, išmokau paralelizuoti tam tikrus procesus naudojantis paralelinį for ciklą.

5. Priedas

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

namespace Lab2
{
    class Data
    {
        private static Random random = new Random();

        public static int[] GenRandomArray(int count)
        {
            int[] result = new int[count];
            for (int i = 0; i < count; i++)
            {
                result[i] = random.Next(0, 20);
            }
            return result;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {

            int[] X1 = new int[] { 1, 5, 5, 2, 1, 8 };
            int[] Y1 = new int[] { 5, 3, 2, 3, 1, 6, 8 };

            //int[] X1 = Data.GenRandomArray(15);
            //int[] Y1 = Data.GenRandomArray(15);

            Stopwatch stopwatch = new Stopwatch();

            int m = X1.Length;
            int n = Y1.Length;

            Bench();

            Console.WriteLine("Uzduotis nr. 1\n");
```

```
        stopwatch.Start();
        int[,] W = MWCS_BottomUp(X1, Y1);
        Console.WriteLine("Dynamic: " + W[m, n]);
        stopwatch.Stop();
        TimeSpan dTime = stopwatch.Elapsed;

        stopwatch.Restart();
        Console.WriteLine("Recursive: " + MWCS_Recurssive(X1, Y1, m, n ) + "\n");
        stopwatch.Stop();
        TimeSpan rTime = stopwatch.Elapsed;

        Console.WriteLine("Recursive vs. Dynamic Time");
        Console.WriteLine("Recursive: " + rTime);
        Console.WriteLine("Dynamic: " + dTime);

        Console.WriteLine("\nUzduotis nr. 2\n");
        int[] priceArray = new int[] { 1, 4, 3, 4, 5 };
        (int, string) rez = CutRod(priceArray, priceArray.Length);
        Console.WriteLine("Didziausias pelnas: " + rez.Item1);

        Console.WriteLine("\nUzduotis nr. 3\n");

        X1 = Data.GenRandomArray(10000);
        Y1 = Data.GenRandomArray(10000);
        m = X1.Length;
        n = Y1.Length;

        stopwatch.Restart();
        W = MWCS_BottomUp(X1, Y1);
        Console.WriteLine("Dynamic: " + W[m, n]);
        stopwatch.Stop();
        dTime = stopwatch.Elapsed;

        stopwatch.Restart();
        Console.WriteLine("Parallel: " + MWCS_Parallel(X1, Y1, W) + "\n");
        stopwatch.Stop();
        TimeSpan pTime = stopwatch.Elapsed;

        Console.WriteLine("Dynamic vs. Parallel dynamic Time");
        Console.WriteLine("Dynamic: " + dTime);
        Console.WriteLine("Parallel: " + pTime + "\n");
    }

    public static int[,] MWCS_BottomUp(int[] X, int[] Y)
    {
        int m = X.Length;
        int n = Y.Length;
        int[,] W = new int[m + 1, n + 1];

        for (int i = 1; i <= m; i++)
        {
            for (int j = 1; j <= n; j++)
            {
                if (X[i - 1] == Y[j - 1])
                {
                    if (X[i - 1] < 0)
                    {
                        W[i, j] = W[i - 1, j - 1];
```

```
                    }
                    else
                    {
                        W[i, j] = X[i - 1] + W[i - 1, j - 1];
                    }
                }
                else
                {
                    W[i, j] = Math.Max(W[i, j - 1], W[i - 1, j]);
                }
            }
        }

        return W;
    }

    static int MWCS_Recurssive(int[] X, int[] Y, int m, int n)
    {
        if (m == 0 || n == 0)
            return 0;
        if (X[m - 1] == Y[n - 1])
            return X[m - 1] + MWCS_Recurssive(X, Y, m - 1, n - 1);
        else
            return Math.Max(MWCS_Recurssive(X, Y, m, n - 1), MWCS_Recurssive(X, Y, m
- 1, n));
    }

    static int MWCS_Parallel(int[] X, int[] Y, int[,] W)
    {
        int m = X.Length;
        int n = Y.Length;

        Parallel.For(1, m,
            i =>
            {
            for (int j = 1; j < n; j++)
            {
                if (X[i] == Y[j])
                {
                    if (X[i] < 0)
                    {
                        W[i, j] = W[i - 1, j - 1];
                    }
                    else
                    {
                        W[i, j] = X[i] + W[i - 1, j - 1];
                    }
                }
                else
                {
                    W[i, j] = Math.Max(W[i, j - 1], W[i - 1, j]);
                }
            }
        });

        return W[m - 1, n - 1];
    }
```

```
        static (int, string) CutRod(int[] price, int n)
        {
            int[] val = new int[n + 1];
            val[0] = 0;
            string combination = "";

            for (int i = 1; i <= n; i++)
            {
                int max_val = int.MinValue;
                for (int j = 0; j < i; j++)
                    max_val = Math.Max(max_val, price[j] + val[i - j - 1]);
                val[i] = max_val;
            }

            return (val[n], combination);
        }

        static void Bench()
        {
            for (int i = 0; i < 5; i++)
            {
                int[] X1 = Data.GenRandomArray(10 + i * 3);
                int[] Y1 = Data.GenRandomArray(10 + i * 3);

                Stopwatch stopwatch = new Stopwatch();

                int m = X1.Length;
                int n = Y1.Length;

                stopwatch.Start();
                int[,] W = MWCS_BottomUp(X1, Y1);
                Console.WriteLine((i + 1) + "Dynamic: " + W[m, n]);
                stopwatch.Stop();
                TimeSpan dTime = stopwatch.Elapsed;

                stopwatch.Restart();
                Console.WriteLine((i + 1) + "Recursive: " + MWCS_Recurssive(X1, Y1, m, n)
+ "\n");
                stopwatch.Stop();
                TimeSpan rTime = stopwatch.Elapsed;

                Console.WriteLine((i + 1) + " - Recursive vs. Dynamic Time");
                Console.WriteLine("Recursive: " + rTime);
                Console.WriteLine("Dynamic: " + dTime + "\n");
            }
        }
    }
}
```