KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

P170B400 Algoritmų sudarymas ir analizė

(P170B400)

Laboratorinių darbų ataskaita

Atliko:

IFF-1/9 gr. studentas

Nedas Liaudanskis

2023 m. gegužės 3 d.

Priėmė:

Doc. Pilkauskas Vytautas

Lekt. Kraujalis Tadas

Doc. Čalnerytė Dalia

Lekt. Makackas Dalius

KAUNAS 2023

1. UŽDUOTIS

1 Dalis:

Pateiktiems programinio kodo metodams **„methodToAnalysis(...)“** (gautiems atlikus užduoties pasirinkimo testą):

- Atlikite pateiktų procedūrų lygiagretinimą.
- Įvertinkite teorinį nelygiagretintų ir lygiagretintų procedūrų sudėtingumą.
- Atlikite realizuotų programinių kodų analizę ir apskaičiuokite įverčius „iš viršaus“ ir „iš apačios“. Atlikite našumo analizę (skaičiuojant programos vykdymo laiką arba veiksmų skaičių) ir patikrinkite, ar apskaičiuotas metodo asimptotinis sudėtingumas atitinka eksperimentinius rezultatus.
- 2 uždavinys - 3 balai / 3 uždavinys - 3 balai

Individualaus užduoties varianto lygčių kodas:

Pirmas metodas:

```
public static long methodToAnalysis (int [] arr)
    {
        long n = arr.Length;
        long k = n;
        for (int i = 0; i < n; i++)
        {

            if (arr[i] / 7 == 0)
            {
                k -= 2;
            }
            else
            {
                k += 3;
            }
        }
        return k;
    }
```

Antras metodas:

```
public static long methodToAnalysis (int n, int[] arr)
```

```
{
    long k = 0;
    for (int i = 0; i < n; i++)
    {
        k += k;
        k += FF9(i, arr);
    }
    k += FF9(n, arr);
    k += FF9(n/2, arr);
    return k;
}

public static long FF9(int n, int[] arr)
{
    if (n > 1 && arr.Length > n && arr[0] < 0)
    {
        return FF9(n - 2, arr) + FF9(n - 1, arr) + FF9(n / n, arr);
    }
    return n;
}
```

2 Dalis:

- Pateikite rekursinį uždavinio sprendimo algoritmą (rekursinis sąryšis su paaiškinimais), bei realizuokite programinį kodą sprendžiantį nurodytą uždavinį (rekursinis sprendimas netaikant dinaminio programavimo).
- Pritaikykite dinaminio programavimo metodologiją pateiktam uždaviniui (pateikti paaiškinimą), bei realizuokite programinį kodą sprendžiantį nurodytą uždavinį (taikant dinaminį programavimą).
- Atlikite realizuotų programinių kodų analizę ir apskaičiuokite įverčius „iš viršaus“ ir „iš apačios“. Atlikite našumo analizę (skaičiuojant programos vykdymo laiką arba veiksmų skaičių) ir patikrinkite, ar apskaičiuotas metodo asimptotinis sudėtingumas atitinka eksperimentinius rezultatus.

1. Pirma Dalis. Metodų „methodToAnalysis(...)“ išanalizavimas

Metodas:

```
public static long methodToAnalysis (int [] arr)
    {
        long n = arr.Length;
        long k = n;
        for (int i = 0; i < n; i++)
        {

            if (arr[i] / 7 == 0)
            {
                k -= 2;
            }
            else
            {
                k += 3;
            }
        }
        return k;
    }
```

Programinio kodo analizė

```
public static long methodToAnalysis(int[] arr)
    {
        long n = arr.Length;                // c1 | 1
        long k = n;                 // c2 | 1
        for (int i = 0; i < n; i++)           // c3 | n + 1
        {

            if (arr[i] / 7 == 0)              // c4 | n
            {
                k -= 2;                  // c5 | n
            }
            else
            {
```

```
            k += 3;                     // c6 | n
        }
    }

    return k;                       // c7 | 1

}
```

Asimptotinio sudėtingumo apskaičiavimas

$$T(n) = c1 + c2 + c3 * n + c4 * n + c5 * n + c6 * n + c7 = (c1 + c2 + c7) + n(c4 + c5 + c6 + c3) = O(n)$$

Kai if (arr[i] / 7 == 0) sąlyga nėra įgyvendinama $c5 = 0$.

Tada $T(n) = (c1 + c2 + c7) + n(c4 + c6 + c3) = O(n)$, ši sąlyga sudėtingumo nekeičia.

Kai if (arr[i] / 7 == 0) sąlyga yra įgyvendinama $c6 = 0$.

Tada $T(n) = (c1 + c2 + c7) + n(c4 + c5 + c3) = O(n)$, ši sąlyga sudėtingumo nekeičia.

Todėl galiu teigti jog šios lygties apatiniai rėžiai yra tokie patys kaip viršutiniai: **Ω(n), O(n).**

Eksperimentinis tyrimas

Metodai

```
public static long methodToAnalysis (int n, int[] arr)
    {
        long k = 0;
        for (int i = 0; i < n; i++)
        {
            k += k;
            k += FF9(i, arr);
        }
        k += FF9(n, arr);
        k += FF9(n/2, arr);
        return k;
    }

    public static long FF9(int n, int[] arr)
    {
        if (n > 1 && arr.Length > n && arr[0] < 0)
        {
            return FF9(n - 2, arr) + FF9(n - 1, arr) + FF9(n / n, arr);
        }
        return n;
    }
```

Programinio kodo analizė

```
public static long methodToAnalysis2(int n, int[] arr)
    {
      long k = 0;                          // c1 | 1
      for (int i = 0; i < n; i++)              // c2 | n + 1
      {
        k += k;                            // c3 | n
        k += FF9(i, arr);                      // sum(i = 0; n = n; T(i) ) | n
      }
      k += FF9(n, arr);                        // sum(i = 0; n = n; T(n) | 1
      k += FF9(n / 2, arr);                      // sum(i = 0; n = n; T(n/2) | 1
      return k;                           // c4 | 1
```

```
        }

        public static long FF9(int n, int[] arr)
        {
          if (n > 1 && arr.Length > n && arr[0] < 0)      // c1 | 1
          {
            return FF9(n - 2, arr) + FF9(n - 1, arr) + FF9(n / n, arr);   // T(n-2) + T(n - 1) + T(n/n) | 1
          }

          return n;    // c3 | 1

}
```

Asimptotinio sudėtingumo radimas

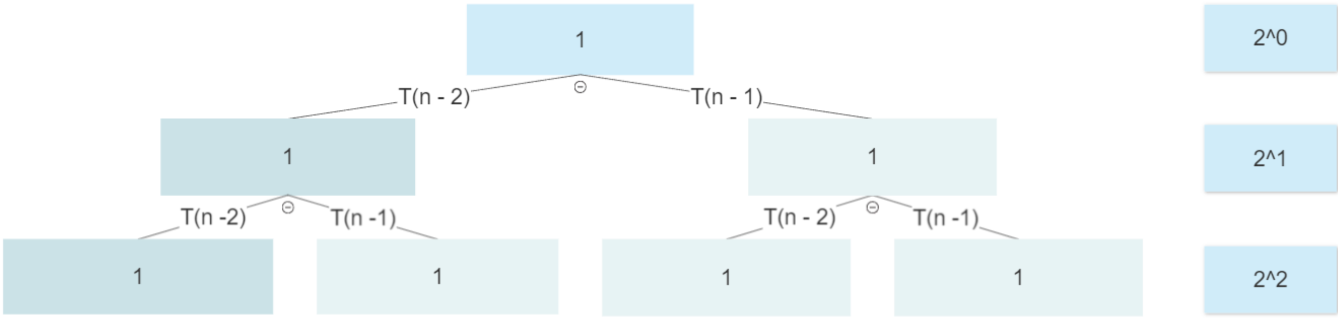
Pirmiausia reikia apsiskaičiuoti FF9 metodo sudėtingumą:

```
public static long FF9(int n, int[] arr)
        {
          if (n > 1 && arr.Length > n && arr[0] < 0)      // c1 | 1
          {
            return FF9(n - 2, arr) + FF9(n - 1, arr) + FF9(n / n, arr);   // T(n-2) + T(n - 1) + T(n/n) | 1
          }

          return n;    // c3 | 1

}
```

$$F(n) = \begin{cases} C1 + c3, \text{ kai neatitinka bent vienos sąlygos} \\ C1 + T(n - 1) + T(n - 2) + T(1) + c3, \text{ kai atitinka visas sąlygas} \end{cases}$$

Geriausiu atveju, kodo ciklai neįvyksta, kai pagal sąlygą n <= 1 && arr.Length <= n && arr[0] >= 0. Tada geriausiu atveju asimptotinis sudėtingumas yra <u>**Ω(1)**</u>

Blogiausiu atveju, reikia išspręsti lygtį F(n) = C + T(n - 1) + T(n -2) + T(1)



Iš diagramos gauname, jog aukštis šios funkcijos yra 2^h, kadangi turime 2 šakas su skirtingais dydžiais tai turėsime viršutinius rėžius ir apatinius. $n \geq h \geq \frac{n}{2}$.

Tai, **$O(T(n)) = 2^n$, $\Omega(T(n)) = 2^{\frac{n}{2}}$.**

Iš lygties gauname jog $F(n) = O(2^n)$

Toliau įsistačius F(n) sudėtingumo reikšmę galime surasti viso metodo sudėtingumą:

```
public static long methodToAnalysis2(int n, int[] arr)
    {
       long k = 0;                          // c1 | 1
       for (int i = 0; i < n; i++)            // c2 | n + 1
       {
          k += k;                          // c3 | n
          k += FF9(i, arr);                  // 2^n | n
       }
       k += FF9(n, arr);                     // 2^n | 1
       k += FF9(n / 2, arr);                   // 2^n
return k;                          // c4 | 1

    }
```

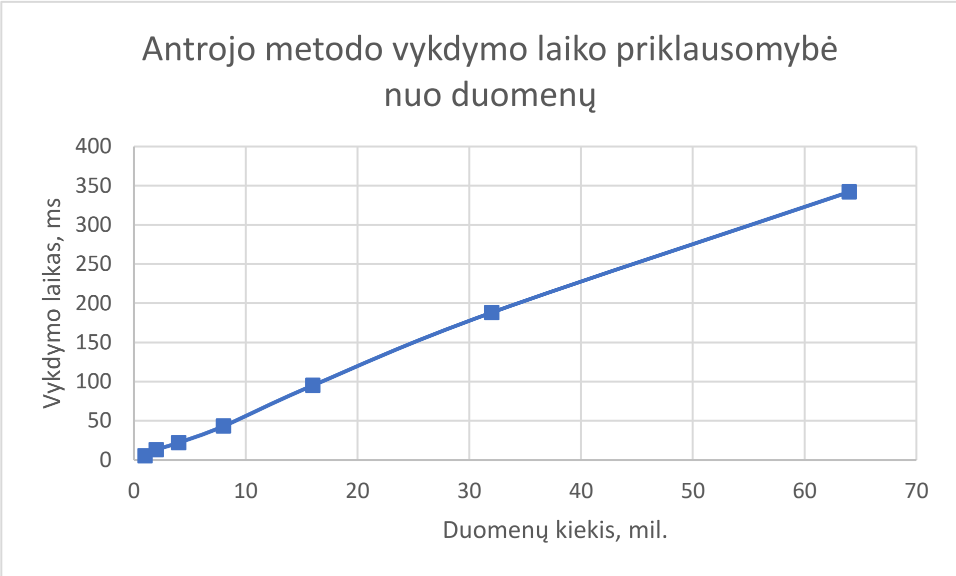
$$T(n) = \left\{ c1 + c2 + c3 * n + n2^n + 2*2^n \right.$$

$$T(n) = \left\{ n + n2^n + 2^n + c \right.$$

Iš lygties gauname:

$T(n) = n + n2^n + 2^n + c$, tai galiu teigti, jog **$T(n) = O(n2^n)$**

Eksperimentinis tyrimas



Metodų lygiagretinimas

```
public static long FF9(int n, int[] arr)
        {
            if (n > 1 && arr.Length > n && arr[0] < 0)        // c1 | 1
            {
                return FF9(n - 2, arr) + FF9(n - 1, arr) + FF9(n / n, arr);    // T(n-2) + T(n - 1)
+ T(n/n) | 1
            }

            return n;      // c3 | 1

        }



        public static long methodToAnalysisParallel(int[] arr)
        {
            long n = arr.Length;
            long k = n;

            Parallel.For(0, n, i =>
            {
                if (arr[i] / 7 == 0)
                {
                    Interlocked.Add(ref k, -2);
                }
                else
                {
                    Interlocked.Add(ref k, 3);
                }
            });

            return k;
```

```
}

public static long methodToAnalysis2Parallel(int n, int[] arr)
{
    long k = 0;

    Parallel.For(0, n, i =>
    {
        Interlocked.Add(ref k, k);
        Interlocked.Add(ref k, FF9(i, arr));
    });

    Interlocked.Add(ref k, FF9(n, arr));
    Interlocked.Add(ref k, FF9(n / 2, arr));


    return k;

}
```

Teorinis nelygia gretintų metodų įvertinimas

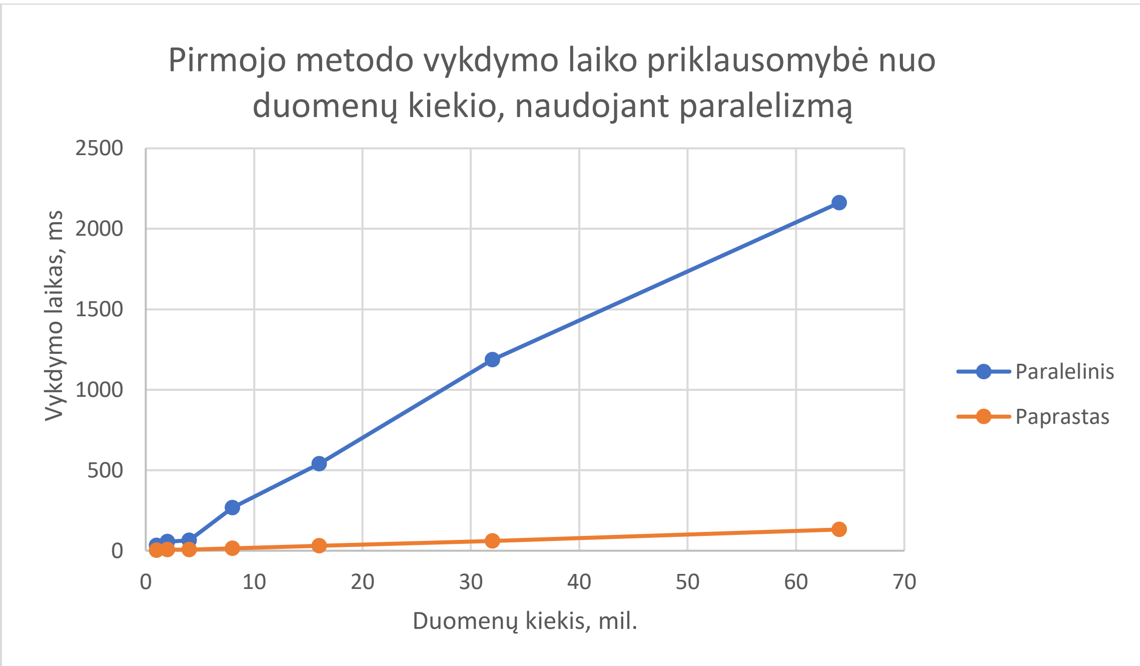
Pirmojo metodo sudėtingumas: **O(n)**

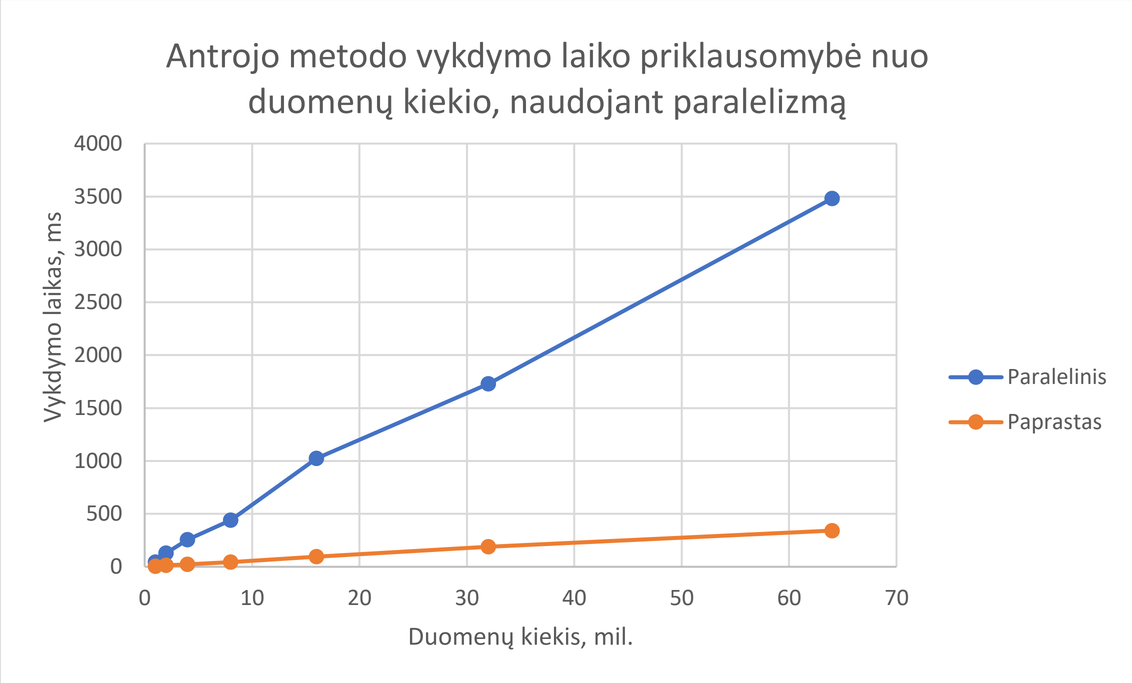
Antrojo metodo sudėtingumas**:** **O(n2^n)**

Teoriškai mąstant vykdant abi programas vieną po kitos gautume sudėtingumą: **O(n + n2^n).** Jį supaprastinus gauname, jog teoriškas įvertis rodo, jog šių metodų sudėtingumas būtų. **O(n2^n)**

Teoriškai žiūrint į paralelizmo veikimo principą, šiuos metodus suskirstant skirtingoms procesoriaus dalims, po mažesnius uždavinius skaičiavimai tūrėtų būti greitesni. Tačiau atlikus greitaveikos testą atsakymai gavosi atvirkščiai. Greičiau buvo įžvigdyta paprasta programa, negu programa, kuri buvo paleista paralelizmo paremtais veikimo principais.

Eksperimentinis tyrimas





2. Antra Dalis:

Užduoties sąlyga

Reikia pakrauti ***W*** keliamosios galios laivą ***n*** pavadinimų daiktais. Tegul m_j – j-tojo pavadinimo daiktų, kuriuos reikia pakrauti, skaičius; r_i – pelnas, kurį atneša vienas pakrautas i-tojo pavadinimo daiktas; w_i – i-tojo pavadinimo vieno daikto svoris. Reikia maksimizuoti pelną neviršijant keliamosios galios.

Programos kodas

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

namespace Program
{
    class Program
    {
        static void Main()
        {
            int W = 2000; // Maximum power
            int n = 12; // Number of items
            int[] m = { 2, 3, 1, 4, 2, 4, 8, 1, 4, 3, 2, 3, 1, 4, 2, 4, 8, 1, 4, 3 }; // Number of
items of each name
            int[] r = { 10, 20, 15, 25, 18, 14, 36, 5, 16, 21, 10, 20, 15, 25, 18, 14, 36, 5, 16, 21
}; // Profit per item
            int[] w = { 5, 10, 8, 15, 12, 5, 120, 41, 12, 14, 5, 10, 8, 15, 12, 5, 120, 41, 12, 14
}; // Power per item

            Calculator Results = new Calculator(n, W);
```

```
            var watch = new System.Diagnostics.Stopwatch();

            watch.Start();
            Results.CalculateRec(W, m, r, w, n);
            watch.Stop();
            Console.WriteLine($"\nRekursinio sprendimo skaičiavimų laikas:
{watch.ElapsedMilliseconds} ms");
            Console.WriteLine($"\nRekurentinio sprendimo rekurentinių ciklų skaičius:
{Results.Counter}");
            watch.Reset();



            watch.Start();
            Results.CalculateDin(W, m, r, w, n);
            watch.Stop();
            Console.WriteLine($"\nDinaminio sprendimo skaičiavimų laikas:
{watch.ElapsedMilliseconds} ms");
            watch.Reset();
            Console.WriteLine("");

            Results.TotalItemCalculator(n, m);
            Console.WriteLine("Maksimalus daiktų skaičius: " + Results.TotalItemCounter);
            Console.WriteLine("");
            Results.Printer();

        }

    }

    /// <summary>
    /// Class used to calculate all the answers
    /// </summary>
    class Calculator
    {
        /// <summary>
        /// Maximum profit using recursive solution
        /// </summary>
        int MaxProfitRec;
        /// <summary>
        /// Maximum profit using recursive dynamic solution
        /// </summary>
        int MaxProfitDyn;
        /// <summary>
        /// Profit made on each item
        /// </summary>
        int[,] Profit;
        /// <summary>
        /// Counts the number of recursions made
        /// </summary>
        public int Counter;

        public int TotalItemCounter;

        /// <summary>
        /// Constructor class for Calculator
        /// </summary>
        /// <param name="n"> Number of items </param>
        /// <param name="W"> Maximum weight </param>
        public Calculator(int n, int W)
        {
             this.MaxProfitRec = 0;
             this.MaxProfitDyn = 0;
             this.Profit = new int[n + 1, W + 1];
```

```
        this.Counter = 0;
        this.TotalItemCounter = 0;
    }

    /// <summary>
    /// Calculates the answer recursively, finds the amount of profit that can be made, maximum
    /// </summary>
    /// <param name="W"> Maximum power </param>
    /// <param name="m"> Number of items of each name </param>
    /// <param name="r"> Profit per item </param>
    /// <param name="w"> Power per item </param>
    /// <param name="n"> Number of item </param>
    /// <returns></returns>
    private int Recursivly(int W, int[] m, int[] r, int[] w, int n)
    {
        Counter++;
        if (n == 0 || W == 0)
        {
            return 0;
        }

        int maxProfitWithoutNthItem = Recursivly(W, m, r, w, n - 1);

        int maxProfit = 0;
        for (int k = 1; k <= m[n - 1] && k * w[n - 1] <= W; k++)
        {
            int profitWithKItems = k * r[n - 1] + Recursivly(W - k * w[n - 1], m, r, w, n - 1);
            maxProfit = Math.Max(maxProfit, profitWithKItems);
        }

        return Math.Max(maxProfitWithoutNthItem, maxProfit);
    }


    /// <summary>
    /// Calculates the answer dynamically, finds the amount of profit that can be made, maximum
    /// </summary>
    /// <param name="W"> Maximum power </param>
    /// <param name="m"> Number of items of each name </param>
    /// <param name="r"> Profit per item </param>
    /// <param name="w"> Power per item </param>
    /// <param name="n"> Number of item </param>
    /// <returns></returns>
    private void CalculateDyn(int W, int[] m, int[] r, int[] w, int n)
    {
        int[,] P = new int[n + 1, W + 1];

        for (int i = 0; i <= n; i++)
        {
            for (int j = 0; j <= W; j++)
            {
                if (i == 0 || j == 0)
                {
                    P[i, j] = 0;
                }
                else if (w[i - 1] > j)
                {
                    P[i, j] = P[i - 1, j];
                }
                else
                {
                    int maxProfit = 0;
                    for (int k = 0; k <= m[i - 1] && k * w[i - 1] <= j; k++)
                    {
```

```
                            maxProfit = Math.Max(maxProfit, k * r[i - 1] + P[i - 1, j - k * w[i - 
1]]);
                        }
                        P[i, j] = maxProfit;
                    }
                }
            }
            MaxProfitDyn = P[n, W];
        }

        /// <summary>
        /// Calculates the answer recursively, finds the amount of profit that can be made, maximum
        /// </summary>
        /// <param name="W"> Maximum power </param>
        /// <param name="m"> Number of items of each name </param>
        /// <param name="r"> Profit per item </param>
        /// <param name="w"> POwer per item </param>
        /// <param name="n"> Number of item </param>
        /// <returns></returns>
        public void CalculateRec(int W, int[] m, int[] r, int[] w, int n)
        {
            MaxProfitRec = Recursivly(W, m, r, w, n);
        }

        /// <summary>
        /// Calculates the answer dynamically, finds the amount of profit that can be made, maximum
        /// </summary>
        /// <param name="W"> Maximum weight </param>
        /// <param name="m"> Number of items of each name </param>
        /// <param name="r"> Profit per item </param>
        /// <param name="w"> Power per item </param>
        /// <param name="n"> Number of item </param>
        /// <returns></returns>
        public void CalculateDin(int W, int[] m, int[] r, int[] w, int n)
        {

            CalculateDyn(W, m, r, w, n);
        }

        public void TotalItemCalculator(int n, int[] m)
        {
            for(int i = 0; i <= n; i ++)
            {
                TotalItemCounter+=m[i];
            }
        }


        /// <summary>
        /// Prints the answers
        /// </summary>
        public void Printer()
        {
            Console.WriteLine("Maksimalus pelnas naudojant rekursinį skaičiavimo metodą: " + 
MaxProfitRec);
            Console.WriteLine("Maksimalus pelnas naudojant dinaminį skaičiavimo metodą: " + 
MaxProfitDyn);
        }

    }


}
```

Programos rezultatai

Pradiniai duomenys:

```
int W = 2000; // Maximum power
            int n = 15; // Number of items
            int[] m = { 2, 3, 1, 4, 2, 4, 8, 1, 4, 3, 2, 3, 1, 4, 2, 4, 8, 1, 4, 3 }; // Number of items of each name
            int[] r = { 10, 20, 15, 25, 18, 14, 36, 5, 16, 21, 10, 20, 15, 25, 18, 14, 36, 5, 16, 21 }; // Profit per
item
            int[] w = { 5, 10, 8, 15, 12, 5, 120, 41, 12, 14, 5, 10, 8, 15, 12, 5, 120, 41, 12, 14 }; // Power per
item
```

Rezultatai:

Rekursinio sprendimo skaičiavimų laikas: 3103 ms

Rekurentinio sprendimo rekurentinių ciklų skaičius: 342945169

Dinaminio sprendimo skaičiavimų laikas: 1 ms

Maksimalus daiktų skaičius: 48

Maksimalus pelnas naudojant rekursinį skaičiavimo metodą: 938

Maksimalus pelnas naudojant dinaminį skaičiavimo metodą: 938

Sudėtingumų skaičiavimai

```
private void CalculateDyn(int W, int[] m, int[] r, int[] w, int n)
        {
            int[,] P = new int[n + 1, W + 1];

            for (int i = 0; i <= n; i++)
            {
                Console.Write(i + ": ");
                for (int j = 0; j <= W; j++)
                {
                    if (i == 0 || j == 0)
                    {
                        P[i, j] = 0;
                    }
                    else if (w[i - 1] > j)
                    {
                        P[i, j] = P[i - 1, j];
                    }
```

```
                else
                {
                    int maxProfit = 0;
                    for (int k = 0; k <= m[i - 1] && k * w[i - 1] <= j; k++)
                    {

                        //When k = 0; We look for the maximum profit from other field,
                        //below P[i-1, j] and set that amount to max, the next iteration,
                        //we compare the max with one item that is i nad so on, and add max
                        //items on top of it from other maximums we calculated before.
                        maxProfit = Math.Max(maxProfit, k * r[i - 1] + P[i - 1, j - k * w[i -
1]]);
                    }
                    P[i, j] = maxProfit;
                }
                Console.Write(P[i, j] + " ");
            }
            Console.Write("\n");
        }
        MaxProfitDyn = P[n, W];


        for(int j = 0; j <= W; j++)
        {
            if(j == 0)
            {
                Console.Write("    " + j);
            }
            else
            {
                Console.Write(" " + j);
            }

        }
    }
```

Dinaminio skaičiavimo metodu akivaizdžiai matome, jog metodas turi 3 ciklus, vienas kitame, kurie priklauso nuo 3 skirtingų dydžių, W-maksimalios jėgos, m-specifinio daikto skaičiaus ir n-esančių daiktų skaičius. Todėl galiu teigti ,jog metodo sudėtingumas yra **O(nmW).**

```
    private int Recursivly(int W, int[] m, int[] r, int[] w, int n)
        {
            Counter++;                              // c1 | 1
            if (n == 0 || W == 0)                    // c2 | 1
            {
                return 0;                              // c3 | 1
            }

            int maxProfitWithoutNthItem = Recursivly(W, m, r, w, n - 1);          // T(n - 1) | 1

            int maxProfit = 0;                                                    // c4 | 1
            for (int k = 1; k <= m[n - 1] && k * w[n - 1] <= W; k++)              // c5 | m +1
            {
                int profitWithKItems = k * r[n - 1] + Recursivly(W - k * w[n - 1], m, r, w, n -
    1);         // T(n - 1) | m
                maxProfit = Math.Max(maxProfit, profitWithKItems);        // c6 | m
            }

            return Math.Max(maxProfitWithoutNthItem, maxProfit);        // c7 | 1
        }
```

Rekursinio. Iš programos kodo analizės gauname lygtį: **T(n, m) = C + mT(n -1) + T(n -1);**

Greičiausiai augantis dydis šioje lygtyje yra **mT(n -1)**, todėl jis ir bus šios lygties sprendinys.

m kartų vykdant tą pačią rekursine, gauname sudėtingumą **O(n, m) =** m^n, šis sudėtingumas puikiai atitinka ir greitaveikos rezultatus, kurie parodo, jog dinaminis metodas, yra greitesnis už rekursinį.
O(n, m) = m^n > **O(nmW).**

Programos rezultatų greitaveika