**KAUNO TECHNOLOGIJOS UNIVERSITETAS**
**INFORMATIKOS FAKULTETAS**

# Modulio P170B400 „Algoritmų sudarymas ir analizė“

Laboratorinio darbo aprašas (ataskaita)

**Antras laboratorinis darbas**

**Dėstytojas**
doc. Vytautas Pilkauskas

**Studentas**
Emilija Vaisvalavičiūtė IFF-0/2

**KAUNAS, 2022**

**TURINYS**

## 1. UŽDUOTIS

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

- atlikite programinio kodo analizę, bei sudarykite rekurentinę lygį. Jei metodas neturi vidinių rekursinių kreipinių, apskaičiuokite pateikto metodo asimptotinį sudėtingumą. Jei metodo sudėtingumas priklauso nuo duomenų pateikiamų per parametrus – apskaičiuokite įverčius „iš viršaus“ ir „iš apačios“ *(2 balai).*
- Metodams, kurie turi rekurentinių kreipinių išspręskite rekurentinę lygtį apskaičiuodami jos asimptotinį sudėtingumą *(1 balas).*
- Atlikti eksperimentinį tyrimą (našumo testus: vykdymo laiką ir veiksmų skaičių) ir patikrinkite ar apskaičiuotas metodo asimptotinis sudėtingumas atitinka eksperimentinius rezultatus. Jei pateikto metodo asimptotinis sudėtingumas priklauso nuo duomenų, atitinkamai atliekant analizę reikia parinkti tokias testavimo duomenų imtis, kad rezultatai atspindėtų įvertinimus iš viršaus ir iš apačios *(1 balas).*

**2 užduoties dalis:** Pateiktas žodinis uždavinys, kurio pagrindu:

- Realizuokite programą skirtingoms pateikto uždavinio duomenų imtims generuoti. Jei duomenų aibė priklauso nuo kelių parametrų – šie parametrai turi būti parametrizuojami. Pvz. jei turi būti sugeneruotas jungusis grafas kurio viršūnių skaičius *v*, o briaunų skaičius gali kisti nuo *v* -1 (minimalaus jungusis grafas) iki *v(v -1)/2* (pilnasis grafas) – grafo užpildymo koeficientas turi būti vienas iš duomenų generavimo parametrų. Sugeneruotos duomenų imtys turi būti saugomos atskiruose failuose, kurie toliau bus naudojami atliekant algoritmų testavimą. *(2 balai)*
- Realizuokite algoritmą optimaliam pateikto uždavinio sprendimui rasti. Laisva forma aprašykite algoritmo logiką ir pateikite kelis testinius atvejus parodant jog sudarytas algoritmas veikia teisingai. Atlikite algoritmo analizę ir pateikite apskaičiuotą sudaryto algoritmo sudėtingumą. Atlikite eksperimentinį tyrimą (skaičiuojant programos vykdymo laiką ir veiksmų skaičių) ir nustatykite, ar pateikto algoritmo sudėtingumas atitinka apskaičiuotą algoritmo sudėtingumą, bei pateikite duomenų kiekius prie kurių pavyksta rasti optimalius sprendinius per pasirinktą programos vykdymo laiką (pvz. 30 sek.). *(2 balai)*
- Sudarykite strategiją ir realizuokite algoritmą kaip galima geresniam pateikto uždavinio sprendimui rasti, kurio sudėtingumas turėtų polinominę priklausomybę nuo įvedamų duomenų kiekio, (šiuo atveju algoritmas turi rasti ne būtinai optimalų, bet kaip galima geresnį sprendinį). Atlikite eksperimentinį tyrimą (skaičiuojant programos vykdymo laiką ir veiksmų skaičių) ir nustatykite, ar pateikto algoritmo sudėtingumas atitinka apskaičiuotą algoritmo sudėtingumą. Palyginkite rezultatus su optimalų sprendinį randančiu algoritmu problematikos ir našumo prasme. *(2 balai)*

## 2. PIRMA DALIS

### 2.1. Pirmas metodas

#### 2.1.1. Programinio kodo analizė

```
public static long methodToAnalysis(int[] arr)    Laikas | Kiekis
  {
      long n = arr.Length;                         c1     | 1
      long k = n;                                  c2     | 1

      for (int i = 0; i < n * n; i++)              c3     | n^2
      {
          if (arr[0] > 0)                          c4     | n^2 - 1
          {
              for (int j = 0; j < n; j++)          c5     | n
              {
                  k -= 2;                          c6     | n - 1
              }

              for (int j = 0; j < n * n / 2; j++) c7     | n^2/2
              {
                  k += 3;                          c8     | n^2/2 - 1
              }
          }
      }
      return k;                                    c9     | 1
```

$$T(n) = \begin{cases} c1 + c2 + c3n^2 + c4(n^2 - 1)((c5n + c6(n - 1)) + (c7\frac{n^2}{2} + c8(\frac{n^2}{2} - 1))) + c9, kai\ arr[0] > 0 \quad (1) \\ c1 + c2 + c3n^2 + c4(n^2 - 1) + c9, kai\ arr[0] <= 0 \quad (2) \end{cases}$$

1) $c1 + c2 + c3n^2 + c4c5n^3 + c4c6n^3 - c4c6n^2 + c4c7\frac{n^4}{2} + c4c8\frac{n^4}{2} - c4c8n^2 - c4c5n - c4c6n + c4c6 - c4c7\frac{n^2}{2} - c4c8\frac{n^2}{2} + c4c8 + c9 = (\frac{1}{2}c4(c7 + c8))\boldsymbol{n^4} + (c4(c5 + c6))\boldsymbol{n^3} + (c3 - (c4(c6 + c8)) - \frac{1}{2}c4(c7 + c8))\boldsymbol{n^2} + (-c4(5n + 6n))\boldsymbol{n} + (c1 + c2 + c4(c6 + c8) + c9)$

2) $(c3 + c4)\boldsymbol{n^2} + (c1 + c2 - c4 + c9)$

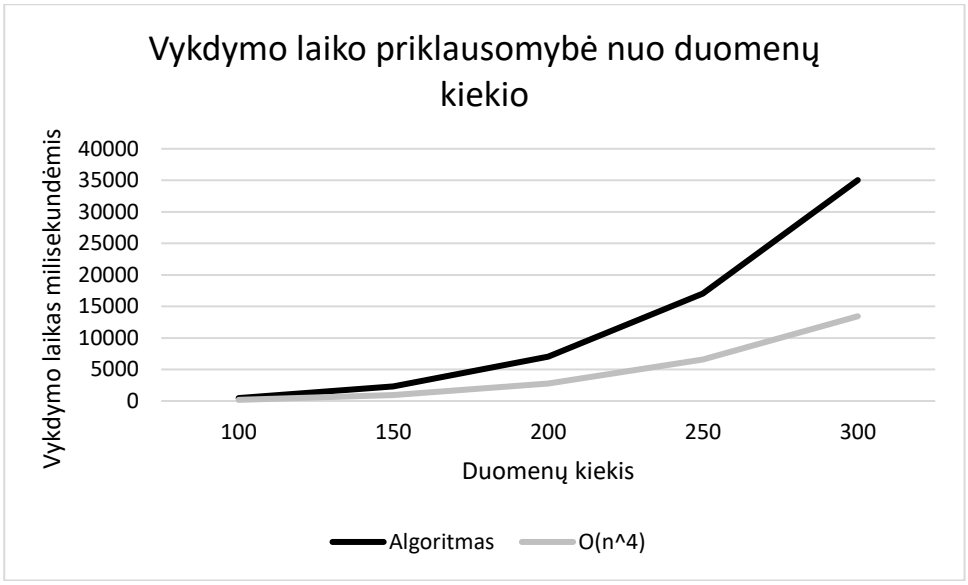
1) $an^4 + bn^3 + cn^2 + dn + e \rightarrow O(n^4)$

2) $an^2 + b \rightarrow \Omega(n^2)$

#### 2.1.2. Eksperimentinis tyrimas

**Arr[0] = 1 (t.y. Arr[0] > 0)**

Vykdymo laiko priklausomybė nuo duomenų kiekio

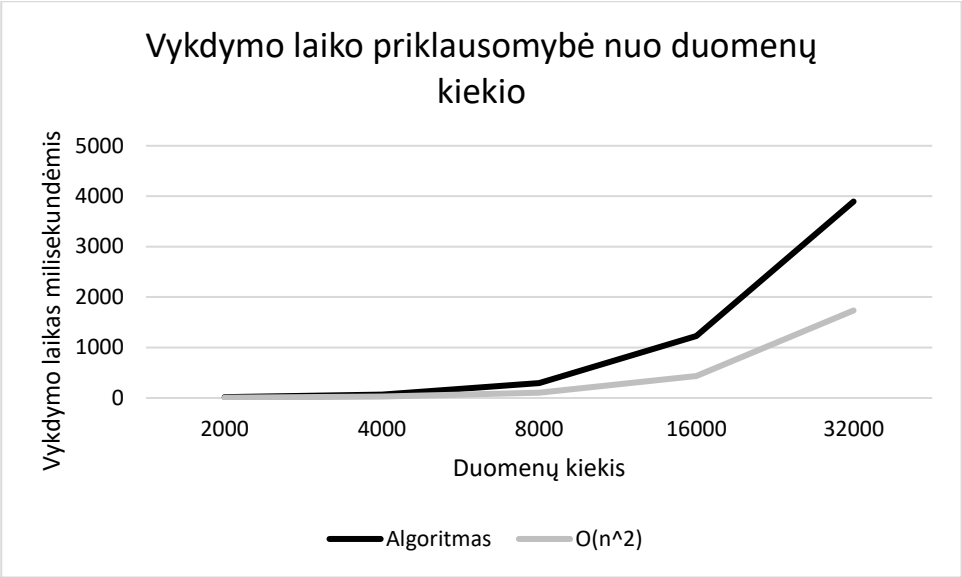



Veiksmų kiekio matavimas

```
Duomenu kiekis 100 veiksmu kiekis 90003
Duomenu kiekis 150 veiksmu kiekis 202503
Duomenu kiekis 200 veiksmu kiekis 562506
Duomenu kiekis 250 veiksmu kiekis 1125009
Duomenu kiekis 300 veiksmu kiekis 1935012
```

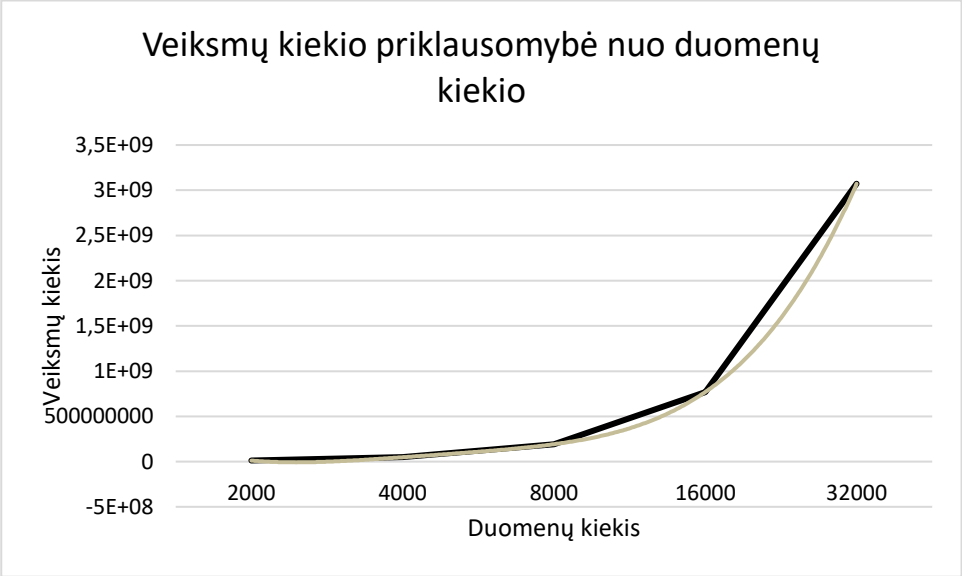

**Arr[0] = 0**

Vykdymo laiko priklausomybė nuo duomenų kiekio




Veiksmų kiekio matavimas

```
Duomenu kiekis 2000 veiksmu kiekis 12000003
Duomenu kiekis 4000 veiksmu kiekis 48000003
Duomenu kiekis 8000 veiksmu kiekis 192000003
Duomenu kiekis 16000 veiksmu kiekis 768000003
Duomenu kiekis 32000 veiksmu kiekis 3072000003
```




Išvada – praktiškai gautas sudėtingumas atitinka teoriškai nustatytą

### 2.2. Antras metodas

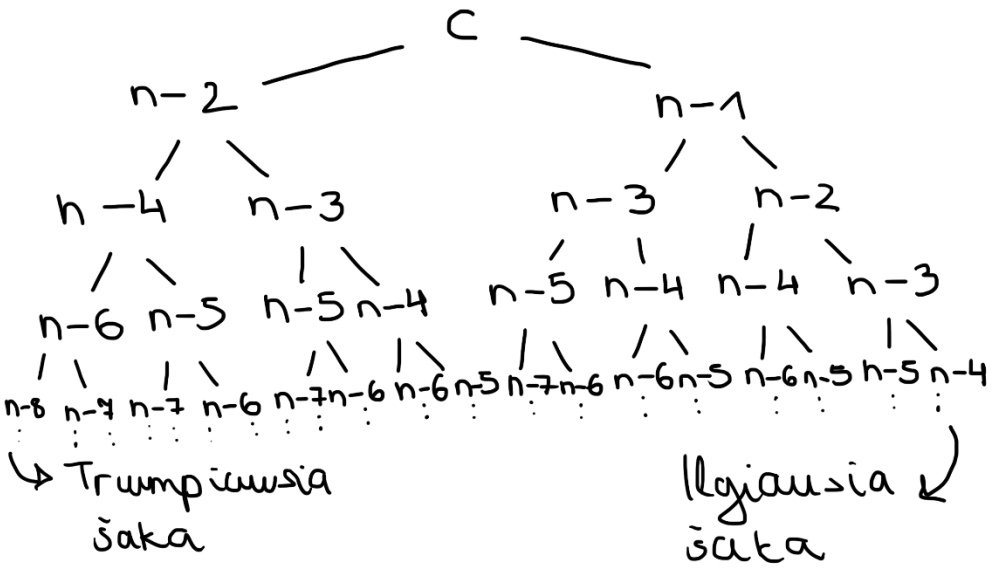
#### 2.2.1. Programinio kodo analizė

```
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(1) | 1
    }

    return n;                                                                   c2 | 1
```

}

$$T(n) = \begin{cases} c1 + c2, & \text{kai neatitinka bent vienos sąlygos} \\ c1 + T(n-2) + T(n-1) + 1 + c2, & \text{kai atitinka visas sąlygas} \end{cases}$$

Pirmu atveju, bus įvykdyta tik vieną kartą ir nekvies rekursijos -> Ω(1)
Antram atvejui išsiaiškinti spręsime lygtį T(n - 1) + T(n - 2) + c, kur c yra skaičius atsirandantis iš FF9(n/n)=FF9(1) iškvietimo, kuris niekad nepatenka į if sąlygos vidų, nes neatitinka n > 1




Blogiausiu atveju aukštis n, o sudėtingumas O(n)=$n2^n$

```
public static long methodToAnalysis(int n, int[] arr)
{                                                                   Laikas | Kiekis
    long k = 0;                                                        c1 | 1

    for (int i = 0; i < n; i++)                                        c2 | n
    {
        k += k;                                                        c3 | n - 1
        k += FF9(i, arr);                                 FF9(i)=FF9(n-1) | n - 1
    }

    k += FF9(n, arr);                                           FF9(n) | 1
    k += FF9(n / 2, arr);                                     FF9(n/2)| 1

    return k;                                                          c4 | 1

}
```

$$T(n) = c1 + c2n + c3(n-1) + FF9(n-1)(n-1) + FF9(n) + FF9(n/2) + c4$$

Jau nustatėme, jog FF9 O(n)=$n2^n$, todėl šio algoritmo O(n)=$n^2 \times 2^n$, o Ω(n)=n (tik tada, kai n=1)

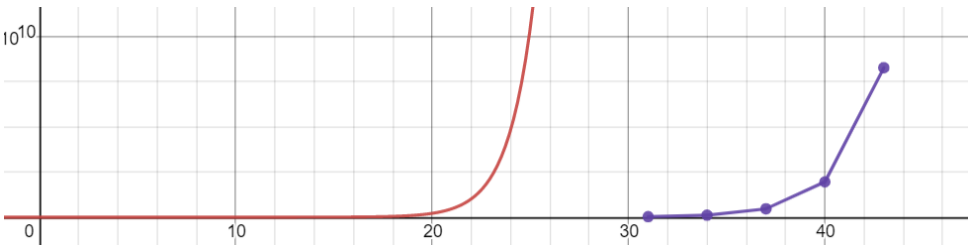
### 2.2.2. Eksperimentinis tyrimas

Vykdymo laiko priklausomybė nuo duomenų kiekio


Veiksmų kiekio matavimas

```
Duomenu kiekis 31 veiksmu kiekis 51334757
Duomenu kiekis 34 veiksmu kiekis 217443491
Duomenu kiekis 37 veiksmu kiekis 921044897
Duomenu kiekis 40 veiksmu kiekis 3901548311
Duomenu kiekis 43 veiksmu kiekis 16526966381
```


Palyginimas su grafu nubrėžtu naudojantis priemone Desmos(https://www.desmos.com/calculator)
Dešinėje nagrinėjamo algoritmo grafas. Matoma, jog kilimo tendencija labai panaši




Išvada – praktiškai gautas sudėtingumas atitinka teoriškai nustatytą

## 3. ANTRA DALIS

### 3.1. Užduotis

Parašykite programą, kuri žemėlapyje (miestai su keliais / jungusis nepilnasis grafas su fiksuotomis viršūnių koordinatėmis) rastų geriausią maršrutą iš pasirinkto miesto aplanakant visus likusius miestus ir grįžtant atgal. Kelionės iš vieno miesto į kitą kaina lygi $\sqrt{\iota_{ij}}$, čia $\iota_{ij}$ atstumas tarp i-ojo ir j-ojo miestų.

### 3.2. Algoritmas duomenų imtims generuoti

#### 3.2.1. Programinis kodas

```
public static List<string> genEdges(int cityAmount, double fullness)
        {
            List<string> edges = new List<string>();
            int fullGraph = (cityAmount * (cityAmount - 1)) / 2;
            for(int i = 0; i < cityAmount - 1; i++)
            {
                string edge = i.ToString() + " " + (i + 1).ToString();
                edges.Add(edge);
            }
            while (edges.Count < fullGraph * fullness)
            {
                Random ran = new Random();
                int start = ran.Next(0, cityAmount);
                int end = ran.Next(0, cityAmount);
                string edge = start.ToString() + " " + end.ToString();
                string reverseEdge = end.ToString() + " " + start.ToString();
                if (!edges.Contains(edge) && !edges.Contains(reverseEdge))
                    edges.Add(edge);
            }
            return edges;
        }

        public static void generateGraphDataFile(string fileName, int cityAmount,
                                                  double fullness, int minDistance,
                                                  int maxDistance)
        {
            if (fullness >= 1)
            {
                Console.WriteLine("Grafas turi buti nepilnasis, pasirinkite junguma
                                  ribose 0.1 - 0.9");
                return;
            }

            File.AppendAllText(fileName, cityAmount.ToString() + "\n");
            List<string> edges = genEdges(cityAmount, fullness);
            for (int i = 0; i < edges.Count; i++)
            {
                Random ran = new Random();
                int dist = ran.Next(minDistance, maxDistance);
                File.AppendAllText(fileName, edges[i] + " " + dist.ToString() + "\n");
            }
        }
```

### 3.3. Sprendimas

#### 3.3.1. Programinis kodas

Duomenų apdorojimas

```
public static int[,] readEdgeMatrix (string fileName)
{
    string[] lines = File.ReadAllLines(fileName);
    int verticeAmount = int.Parse(lines[0]);
    int[,] edgeMatrix = new int[verticeAmount, verticeAmount];
    for (int i = 1; i < verticeAmount+1; i++)
    {
        string[] values = lines[i].Split();
        int start = int.Parse(values[0]);
        int end = int.Parse(values[1]);
        int weight = int.Parse(values[2]);
        edgeMatrix[start,end] = weight;
        edgeMatrix[end,start] = weight;
    }
    return edgeMatrix;
}
```

Algoritmas užduoties sprendimui

```
public static void solve(int[,] edgeMatrix, ref Stack<int> path,
                         ref List<int> visited, int vertice)
{
    double minWeight = int.MaxValue;
    int next = -1;
    if (visited.Count == edgeMatrix.GetLength(1) && path.Peek() == 0)
        return;
    for (int i = 0; i < edgeMatrix.GetLength(1); i++)
    {
        if (edgeMatrix[vertice, i] > 0)
        {
            if(visited.Count == edgeMatrix.GetLength(1) && i == 0)
            {
                minWeight = Math.Sqrt(edgeMatrix[vertice, i]);
                next = i;
            }
            else if (!visited.Contains(i) && Math.Sqrt(edgeMatrix[vertice, i])
                     < minWeight)
            {
                minWeight = Math.Sqrt(edgeMatrix[vertice, i]);
                next = i;
            }
            else if (!visited.Contains(i) && visited.Contains(next))
            {
                next = i;
                minWeight = Math.Sqrt(edgeMatrix[vertice, i]);
            }
            else if (next == -1 && path.Count > 1 && path.Skip(1).First() != i)
            {
                next = i;
                minWeight = Math.Sqrt(edgeMatrix[vertice, i]);
            }
            else if (next == -1)
            {
                next = i;
                minWeight = Math.Sqrt(edgeMatrix[vertice, i]);
            }
        }
```

```
            }

            if (!visited.Contains(next))
                visited.Add(next);

            path.Push(next);
            Console.WriteLine("Miestas: " + next + " Kaina: " +
                              String.Format("{0:0.00}", minWeight));

            solve(edgeMatrix, ref path, ref visited, next);
            return;
        }
```

### 3.3.2. Testai

#### Testas 1

Pradinių duomenų failas

```
5
0 1 8
1 2 5
2 3 5
3 4 3
1 4 9
0 4 6
3 0 6
4 2 3
```

Rezultatas

```
Miestas: 0
Miestas: 1 Kaina: 2,83
Miestas: 2 Kaina: 2,24
Miestas: 3 Kaina: 2,24
Miestas: 4 Kaina: 1,73
Miestas: 1 Kaina: 3,00
Miestas: 0 Kaina: 2,83
```

#### Testas 2

Pradinių duomenų failas

```
10
0 1 4
1 2 3
2 3 6
3 4 3
4 5 1
5 6 5
6 7 7
7 8 3
8 9 2
8 1 2
7 9 3
4 2 8
2 6 1
4 6 9
```

```
4 7 7
2 9 8
0 9 5
7 0 3
0 6 6
3 7 6
0 5 4
5 2 9
6 3 8
9 6 4
5 1 4
1 3 2
9 3 9
5 7 7
2 7 8
6 8 5
3 8 4
1 9 7
0 2 1
8 5 8
5 9 5
4 9 9
```

Rezultatas

```
Miestas: 0
Miestas: 1 Kaina: 2,00
Miestas: 8 Kaina: 1,41
Miestas: 9 Kaina: 1,41
Miestas: 8 Kaina: 1,41
Miestas: 7 Kaina: 1,73
Miestas: 6 Kaina: 2,65
Miestas: 5 Kaina: 2,24
Miestas: 4 Kaina: 1,00
Miestas: 3 Kaina: 1,73
Miestas: 2 Kaina: 2,45
Miestas: 1 Kaina: 1,73
Miestas: 0 Kaina: 2,00
```

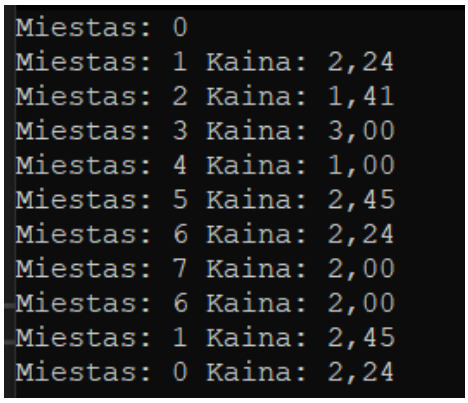
**Testas 3**

Pradinių duomenų failas

```
8
0 1 5
1 2 2
2 3 9
3 4 1
4 5 6
5 6 5
6 7 4
6 1 6
0 7 9
3 6 8
6 2 4
0 6 9
7 2 4
2 5 1
1 5 6
2 4 1
```

```
2 0 7
1 7 3
5 7 4
4 1 1
6 4 3
0 3 7
3 1 8
```

Rezultatas




### 3.3.3. Algoritmo analizė

Tarkime, jog edgeMatrix ilgis (tai vienodų dimensijų matrica, tai nesvarbu į kurią pusę) = n

```
public static void solve(int[,] edgeMatrix, ref Stack<int> path,                Laikas | Kiekis
                         ref List<int> visited, int vertice)
{
    double minWeight = int.MaxValue;                                              c1 | 1
    int next = -1;                                                                c2 | 1
    if (visited.Count == edgeMatrix.GetLength(1) && path.Peek() == 0)             c3 | 1
        return;                                                                   c4 | 1
    for (int i = 0; i < edgeMatrix.GetLength(1); i++)                             c5 | n
    {
        if (edgeMatrix[vertice, i] > 0)                                           c6 | n - 1
        {
            if(visited.Count == edgeMatrix.GetLength(1) && i == 0)                c7 | 1
            {
                minWeight = Math.Sqrt(edgeMatrix[vertice, i]);                    c8 | 1
                next = i;                                                         c8 | 1
            }
            else if (!visited.Contains(i) && Math.Sqrt(edgeMatrix[vertice, i])    c9 | 1
                     < minWeight)
            {
                minWeight = Math.Sqrt(edgeMatrix[vertice, i]);                    c8 | 1
                next = i;
            }
            else if (!visited.Contains(i) && visited.Contains(next))             c10 | 1
            {
                next = i;                                                         c8 | 1
                minWeight = Math.Sqrt(edgeMatrix[vertice, i]);                    c8 | 1
            }
            else if (next == -1 && path.Count > 1 && path.Skip(1).First() != i)   c11 | 1
            {
                next = i;                                                         c8 | 1
                minWeight = Math.Sqrt(edgeMatrix[vertice, i]);                    c8 | 1
            }
            else if (next == -1)                                                 c12 | 1
            {
                next = i;                                                         c8 | 1
                minWeight = Math.Sqrt(edgeMatrix[vertice, i]);                    c8 | 1
            }
        }
    }

    if (!visited.Contains(next))                                                 c13 | 1
        visited.Add(next);                                                       c14 | 1
```

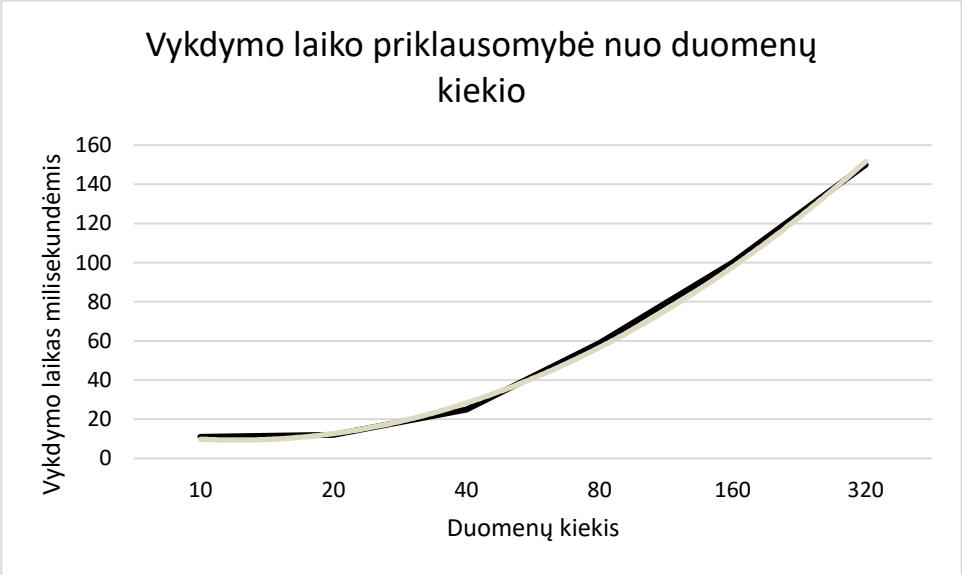
```
            path.Push(next);                                                    c15 | 1
            Console.WriteLine("Miestas: " + next + " Kaina: " +
                              String.Format("{0:0.00}", minWeight));            c16 | 1

            solve(edgeMatrix, ref path, ref visited, next);                     T(n) | 1
            return;                                                             c17 | 1
        }
```

$$T(n) = (c1 + c2 + c3 + c4 + c7 + c8 + c9 + c10 + c11 + c12 + c13 + c14 + c15 + c16 + c17) + (c5 + c6)n + T(n)$$

Rekurentinė lygtis T(n) + n, tačiau rekursinių iškvietimų kiekis gali labai skirtis priklausomai nuo grafo kraštinių, nes iškvietimas baigiasi ne nustatytu metu pagal matematinę išraišką su n reikšme, o pasiekus pabaigos sąlygą. Net tam pačiam n esant, bet pasikeitus struktūrai (kas su kuo jungiasi) gali skirtis

Tikėtina, jog $\Theta(n^2)$ ir $\Omega(n^2)$, nes reikia praeiti kiekvieną miestą(viršūnę) bent kartą

### 3.3.4. Eksperimentinis tyrimas




Prilyginus grafą $O(n^2)$, matoma, jog sutampa kilimo tendencija, todėl galima spręsti, jog toks yra ir šio algoritmo sudėtingumas

### 3.3.5. Duomenų kiekiai, kuriem esant išeina rasti sprendimą

Buvo užsibrėžta, jog algoritmas veikia efektyviai, jei rezultatą grąžina per 30s arba mažiau. Buvo rastos tokios duomenų imtys, kuriom esant vykdymo laikas arti tos ribos (vadinasi didinant jau perkoptų ribą):

Su 700 miestų (grafo viršūnių) ir 0.2 jungumo koeficientu užtrunka 29s

Su 550 miestų (grafo viršūnių) ir 0.3 jungumo koeficientu užtrunka 27s

Su 450 miestų (grafo viršūnių) ir 0.4 jungumo koeficientu užtrunka 28s

Su 420 miestų (grafo viršūnių) ir 0.5 jungumo koeficientu užtrunka 30s

Su 370 miestų (grafo viršūnių) ir 0.6 jungumo koeficientu užtrunka 30s

Su 340 miestų (grafo viršūnių) ir 0.7 jungumo koeficientu užtrunka 28s

Su 310 miestų (grafo viršūnių) ir 0.8 jungumo koeficientu užtrunka 27s

Su 280 miestų (grafo viršūnių) ir 0.9 jungumo koeficientu užtrunka 28s