Grafo viršūnių spalvinimas

Lukas Klusis

Vilniaus universitetas

Matematikos ir informatikos fakultetas

Gegužes 11 d., 2014

Turinys

1 Įvadas

1.1 Užduotis[1]

Duota: Duotas neorientuotas grafas G, turintis n viršūnių ir m briaunų.
Rasti: Grafo G viršūnių nuspalvinimą minimaliu spalvų kiekiu taip, kad dvi gretimos viršūnės būtų skirtingų spalvų (kiekvienos briaunos galai būtų skirtingų spalvų)

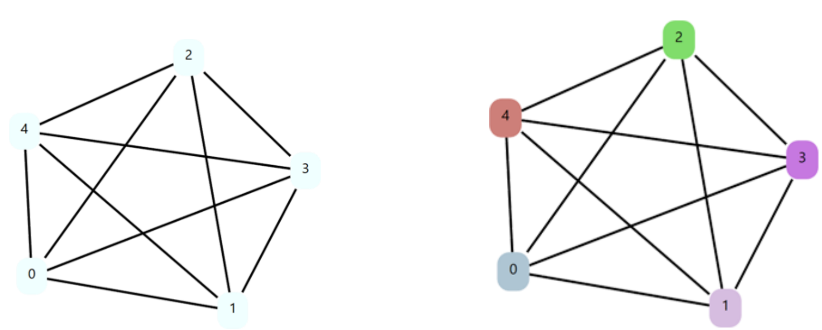
1.2 Užduoties aprašymas

Tai yra žinoma Grafo viršūnių spalvinimo problema, kurios sprendimui nėra polinominio laiko algoritmo, t.y. šis uždavinys yra priskiriamas NP[2] sudėtingumo klasei. Tačiau atskirais atvejais šiam uždaviniui galima surasti polinominį algoritmą.

Analizuojant šį uždavinį gali tekti susidurti su papildomomis sąvokomis ir apibrėžimais, iš kurių kiekvieną apibrėšime eigoje.

Apibrėžimas 1. grafo $G = (V,E)$ dvi viršūnės v_i, $v_j \in V$ $(i \neq j)$ yra vadinamos *gretimomis viršūnės* jeigu egzistuoja briauna $(v_i, v_j) \in E$.

Apibrėžimas 2. grafo *G chromatiniu skaičiumi* yra vadinamas mažiausias kiekis spalvų, kurių reikia norint nuspalvinti grafą *G* taip, kad jokios dvi gretimos viršūnės nebūtų vienos spalvos. Šis dydis yra žymimas kaip $\chi(G)$.



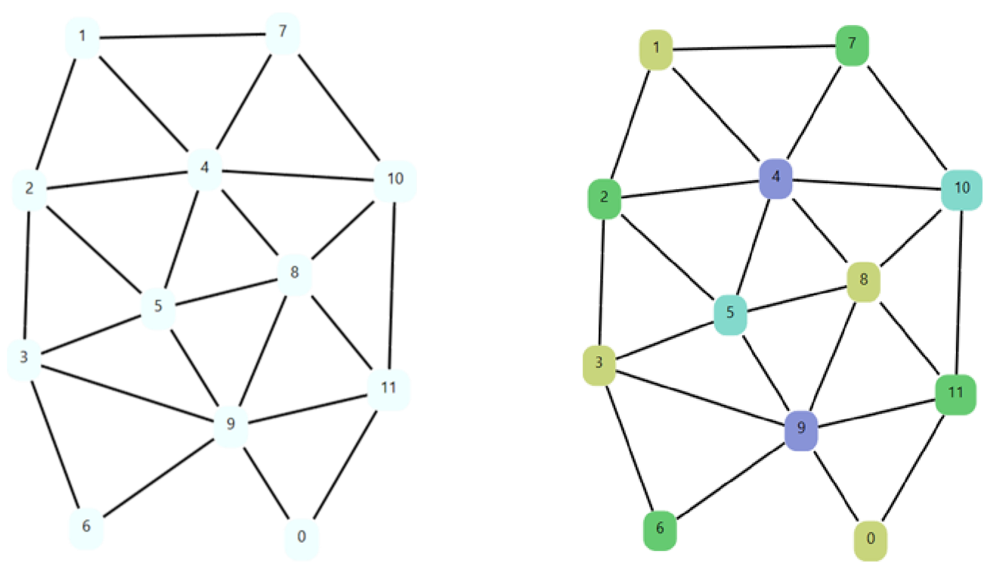
Pavyzdys 1. Pilnojo grafo K_5 nuspalvinimas 5 spalvomis, $\chi(K_5) = 5$

[1] 12 užduotis - http://uosis.mif.vu.lt/~valdas/ALGORITMAI/Uzduotys2014
[2] NP compedium - http://www.csc.kth.se/~viggo/wwwcompendium/node15.html

Apibrėžimas 3. *Planarus grafas* – grafas $G = (V,E)$ yra vadinamas *planariuoju* arba *plokščiu* jeigu jį galima pavaizduoti plokštumoje taip, kad jo briaunos nesikirstų niekur išskyrus viršūnių taškus.

Šie apibrėžimai buvo vieni iš pagrindinių problemoje, kuri pirmą kartą buvo įvardinta 1852 metais Francis Guthrie. Joje iškelta hipotezė, kad kiekvieną planarųjį grafą G galima korektiškai nuspalvinti su ne daugiau negu 4 spalvomis (t.y. $\chi(G) \leq 4$). Šią problemą nuo pat jos paskelbimo laikotarpio sekė eilė įrodymų, tačiau daugelis iš jų buvo neteisingi. 1890 metais buvo įrodyta kiek paprastesnė teorema, kad kiekvieną planarų grafą G galima nuspalvinti 5 skirtingomis spalvomis (t.y. $\chi(G) \leq 5$). Galiausiai 1989 metais buvo paskelbtas ir pradinės teoremos įrodymas besiremiantis kompiuteriniais sprendimais.

Šiame darbe mes taip pat nagrinėsime kiek lengvesnį variantą – ieškosime planaraus grafo G nuspalvimą tokį, kad jį sudarytų ne daugiau negu 5 skirtingos spalvos (t.y. $\chi(G) \leq 5$)



Pavyzdys 2. Planaraus grafo nuspalvinimas 4 spalvomis.

2 Algoritmų kūrimas

Mūsų tikslas yra sukurti algoritmą, kuris galėtų surasti bet kokio grafo G minimalų chromatinį skaičių. Šiam tikslui mes naudosime dvi skirtingas algoritmų kūrimo strategijas:

- Paieška su grįžimu (Zeil)
- Godųjį (spalvinimo) algoritmą (Kreher & Stinson, pp. 14-28)

2.1 Duomenų struktūros

Abiem algoritmams mes naudosime grafą G, *kuris yra sudarytas iš dviejų sąrašų:*

- sąrašas viršūnių
- sąrašas briaunų

Taip pat kiekviena viršūnė turi nuorodas į gretimas viršūnes.
Papildomai naudojamas sąrašas, kuriame saugomos spalvos.
Verta pastebėti, kad visų aukščiau įvardintų sąrašų elementų eiliškumas yra svarbus.

2.2 Paieška su grįžimu

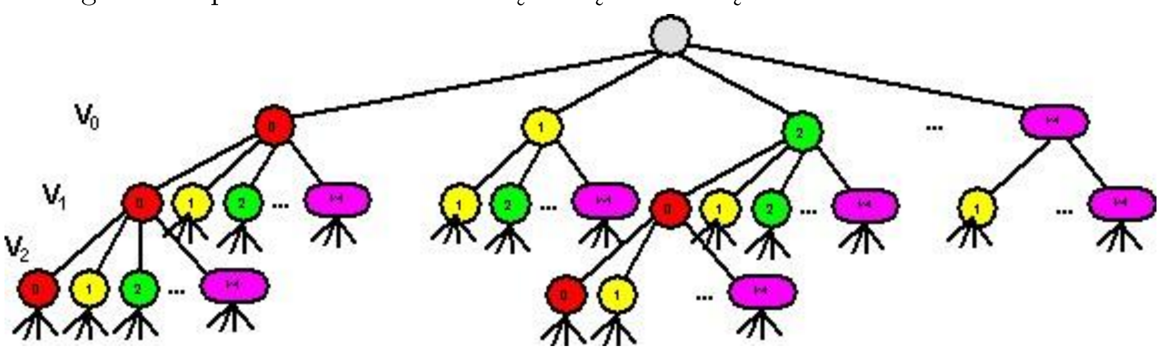
Paieškos su grįžimu algoritmas yra artimas brutalios jėgos (*angl. brute force)* algoritmui, kuris kaip žinia greitai norimų rezultatų neduoda. Naudojant paiešką su grįžimu sprendimą dažniausiai gauname greičiau ir efektyviau negu naudojant brutalios jėgos algoritmą, tačiau gali pasitaikyti variantų, kai teks perrinkti visus arba didžiąją dalį galimų variantų, kad rastumėme ieškomo sprendimo.

Minimalaus grafo $G = (V,E)$ chromatinio skaičiaus suradimą naudojant paiešką su grįžimu galima apibrėžti šiais žingsniais:

1) Nustatyti pradinį spalvų kiekį lygu 0 ir nuspalviname visas grafo G viršūnes pradine spalva, kuri mūsų pasirinkimu bus juoda.
2) Padidiname esamų spalvų sąrašą viena spalva ir pradedama spalvinti grafą G

3) Nustatome indeksą i lygų 0; jį naudosime viršūnių ėmimui iš eilės pagal viršūnių sąrašo $V = \{v_1, v_2, \ldots, v_n\}$ eiliškumą
4) Padidiname i vienetu ir imame viršūnę $v_i \in V$
5) Iš eilės imame visas spalvas iš spalvų sąrašo ir kiekvienai spalvai tikriname ar ši nėra panaudota nei vienai viršūnei $v_i^{kaimynė}$ kuri yra gretima v_i. Jeigu randame tokią spalvą, ja nuspalviname viršūnę v_i ir keliaujame į 6 žingsnį, jeigu tokios spalvos neradome, grįžtame atgal, t.y.:
    a. Jeigu $i > 1$ tęsiame darbą su viršūnę v_{i-1}.
    b. Jeigu $i = 1$ grįžtame prie žingsnio 2)
6) Jeigu $i < n$ (t.y. dar liko nenuspalvintų viršūnių), grįžtame prie 4) žingsnio.

Šio algoritmo sprendimo medis turėtų tokią struktūrą[3]:



Algoritmo implementacija C# programavimo kalboje:
```
//turime apsibrėžę konstantinę spalvą pavadinimu Black, ji simbolizuoja
//nenuspalvintą viršūnę
//patogumo dėlei turime atsitiktinį skirtingų spalvų generatorių RandomColor

//algoritmo rezultatas – chromatinis grafo skaičius
public int RunBacktrackingAlgorithm()
        {
            ColorsList = new List<Color>();
            randomColor = new RandomColor();
            ColorsList.Add(randomColor.NextColor());

            foreach (var vertex in Graph.Vertices)
            {
                vertex.Background = Black;
            }
            while (Graph.Vertices.Any(x => x.Background.Equals(Black)))
```

[3] https://secweb.cs.odu.edu/~zeil/cs361/web/website/Lectures/npprobs/pages/ar01s01s01.html

```
            {
                var rootVertex = Graph.Vertices.First(x => x.Background.Equals(Black)
);
                if (!ColorVertex(Graph.Vertices))
                {
                    ColorsList.Add(randomColor.Color());
                }
            }
            return ColorsList.Count;
        }

private bool ColorVertex(IEnumerable<Vertex> verticesToColor)
{
    //patikriname ar bent vienas elementas yra likęs
    if ( !verticesToColor.Any(x => true))
    {
        return true;
    }
    var vertex = verticesToColor.First();

    foreach (var color in ColorsList)
    {
       bool canColor = CanColor(vertex, color);

        if (canColor)
        {
            vertex.VertexControl.Background = Color;

             //ieškome ar galime nuspalvinti viršūnes esančias sąraše toliau, negu
             //nuspalvinta viršūne
            if (ColorVertex(verticesToColor.Skip<Vertex>(1)))
            {
                 return true;
            }
         }
      }
      //jeigu su dabartinės viršūnės spalvinimu, negalėjome nuspalvinti viršūnių
      //esančių sąraše toliau vadinasi negalime nuspalvinti ir šios viršūnės,
      //atstatome viršūnės spalvą į pradinę ir grįžtamą prie viršūnės esančios
      //anksčiau viršūnių sąraše.
      vertex.VertexControl.Background = Black;
      return false;
}

private bool CanColor(Vertex vertex, Color Color)
{
     return !vertex.ChildVertices.Any(x => x.Background.Equals(Color));
}
```

Matomai šis algoritmas nėra greitas, jo sudėtingumas yra $O(k^n)$, kur k yra spalvų skaičius, o n viršūnių skaičius.

2.3 Godusis algoritmas

Godusis algoritmas iš eilės ima po vieną viršūnę ir parenka jam pirmą galimą spalvą iš spalvų sąrašo. Godusis algoritmas iš tiesų ne visada gražiną optimalų sprendimą ar randą tikslų chromatinį skaičių, tačiau šis algoritmas yra daug paprastesnis ir įvarios jo modifikacijos yra naudojamos kai kurių teoremų įrodymams ar pritaikomas realaus gyvenimo uždaviniams, kuriems nereikia griežto tikslumo.

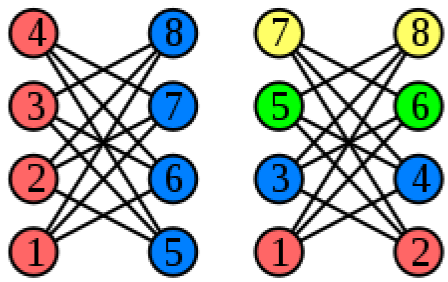
Algoritmo implementacija C# programavimo kalboje gali atrodyti taip:

```
public int RunGreedyAlgorithm()
{
    //susikuriame spalvų sąrašą. Pradžioje jame nėra nei vienos spalvos
    Colors = new List<Color>();
    foreach (var vertex in Graph.Vertices)
    {
       //kiekvienai viršūnei iš eilės priskiriame pirmą galimą spalvą
       vertex.VertexControl.Background = GetLowestColor(vertex);
    }
    return Colors.Count();

}

private Color GetLowestColor(Vertex vertex)
{
   //ieškome pirmos galimos spalvos esančios arčiausiai spalvų sąrašo pradžios
   foreach (var color in Colors)
   {
      if (vertex.ChildVertices.All(x => x.Background != color))
      {
           return color;
      }
   }
   //jeigu tokios spalvos surasti nepavyko, tai pridedame naują spalvą (padidiname
   //ieškomo grafo chromatinį skaičių vienetu) ir gražiname šią spalvą
   Colors.Add(randomColor.NextColor());
   return Colors.Last();
}
```

Šis algoritmas yra tiesinis, t.y. jo sudėtingumo klasė yra *O(n)*, tačiau šis algoritmas gali ir klysti.
Pavyzdžiui priklausomai nuo to kaip bus surūšiuotos pilnojo dvidalio grafo $K_{n,n}$ viršūnės algoritmas gali pateikti skirtingo chromatinio skaičiaus sprendimus.



Pavyzdys 3[4]. Grafo $K_{4,4}$, priklausomai nuo pateikto viršūnių sąrašo eiliškumo gali būti nuspalvintas 2 arba 4 spalvomis

3 Algoritmų analizavimas

Jau minėjome, kad paieškos su grįžimu algoritmo sudėtingumo klasė yra $O(k^n)$, o godžiojo algoritmo *O(n)*. Tačiau pasižiūrėkime kokie yra praktiniai šių algoritmų panaudojimo rezultatai.

Analizuojant algoritmus buvo automatiškai generuojami grafai[5]. Grafams generuoti buvo naudojama papildoma sąvoka kuri yra aktuali tik šio darbo kontekste - grafo dydis

Apibrėžimas 4. Grafo dydis – tai statistinis vidutinis dydis, tiesiškai apibūdinantis generuojamo grafo briaunų ir viršūnių skaičių.
SVARBU. Šio darbo kontekste grafo dydis įgyja kitą nei įprasta naudoti grafų teorijoje sąvoką. Įprastai grafo dydžiu vadinamas grafo briaunų skaičius.

Iš viso buvo sugeneruota daugiau negu 5900 skirtingų grafų ir kiekvienas iš jų buvo nuspalvintas du kartus, naudojant godųjį algoritmą ir naudojant paieškos su grįžimu algoritmą.

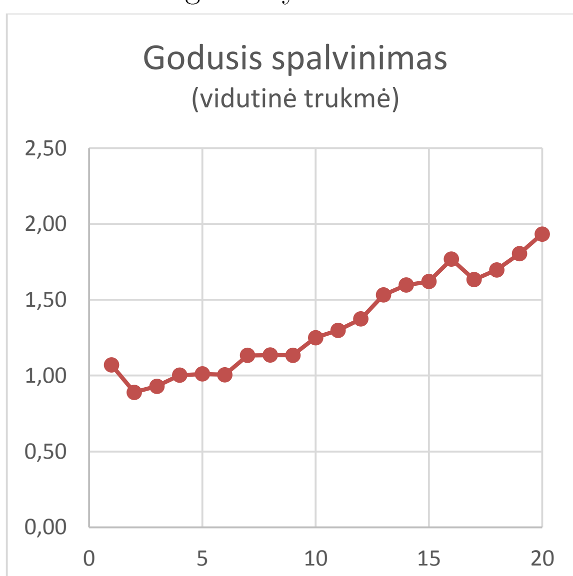
---

[4] http://en.wikipedia.org/wiki/Greedy_coloring
[5] http://graphstream-project.org/doc/Generators/Random-graph-generator_1.0/

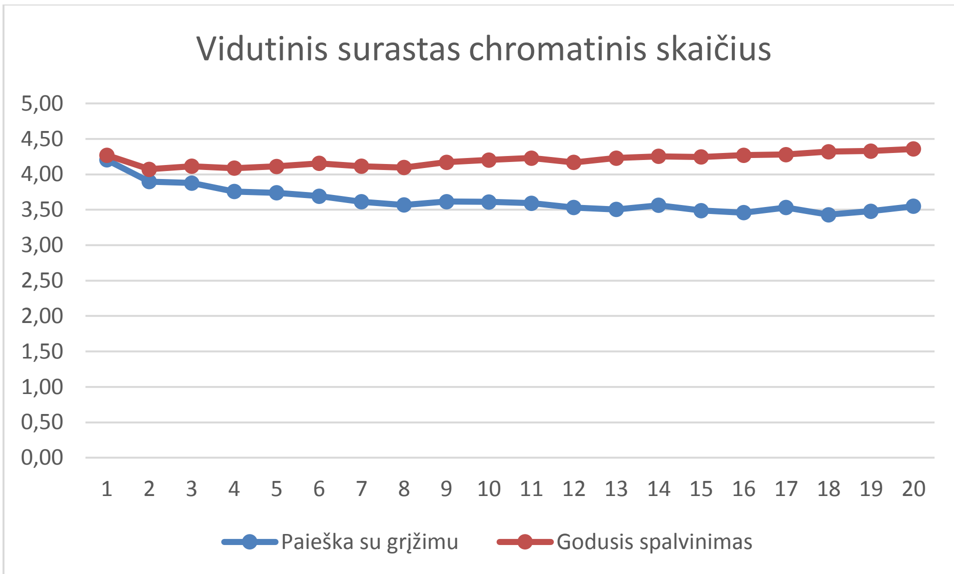
Šio bandymo rezultatus galime pavaizduoti lentele

Grafo dydis	bandymų sk.	Paieška su grįžimu vidutinė trukmė (milisek)	Paieška su grįžimu vidutinis spalvų sk.	Godusis algoritmas vidutinė trukmė (milisek)	Godusis algoritmas vidutinis spalvų sk.	vidutinis viršūnių sk.	vidutinis briaunų sk.
1	400	4,74	4,21	1,07	4,27	6	11,94
2	400	5,76	3,90	0,89	4,07	7	13,76
3	400	7,60	3,88	0,93	4,12	8	16,04
4	400	10,91	3,76	1,00	4,09	9	18,07
5	400	15,49	3,74	1,01	4,11	10	20,23
6	400	24,58	3,69	1,01	4,16	11	22,14
7	400	30,63	3,61	1,13	4,12	12	23,82
8	400	36,26	3,57	1,14	4,10	13	25,64
9	400	84,48	3,62	1,13	4,17	14	28,04
10	500	122,44	3,61	1,25	4,20	15	30,03
11	300	225,26	3,59	1,30	4,23	16	32,14
12	300	175,54	3,53	1,37	4,17	17	34,04
13	200	382,77	3,51	1,53	4,23	18	35,80
14	200	578,78	3,57	1,60	4,26	19	38,02
15	200	1159,11	3,49	1,62	4,25	20	39,28
16	200	688,37	3,46	1,77	4,27	21	41,73
17	128	1994,14	3,53	1,63	4,28	22	43,75
18	100	2229,41	3,43	1,70	4,32	23	45,16
19	100	5602,08	3,48	1,80	4,33	24	47,68
20	100	8973,65	3,55	1,93	4,36	25	50,32

Bei grafiškai, kaip priklauso algoritmo veikimo trukmė nuo grafo dydžio.

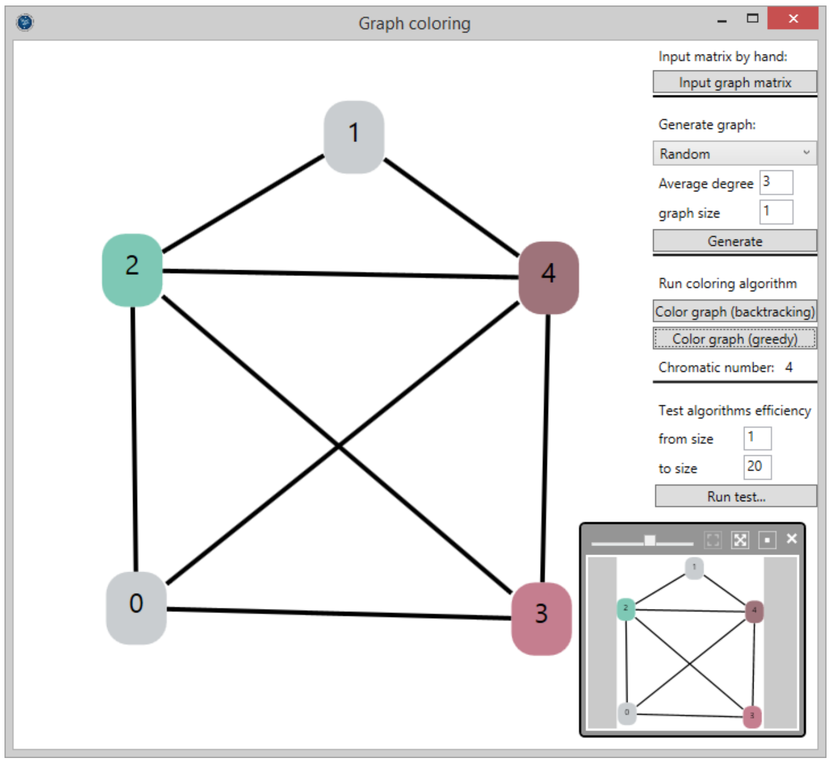


Verta atkeipti dėmesį ir į algoritmų korektiškumą. Paieška su grįžimu visada pateikia optimalų sprendimo būdą, kai tuo tarpu chromatinis skaičius surastas godžiuoju algoritmu yra beveik visada didesnis negu optimalaus sprendinio.



3.1 Programos naudojimas

Prie šio darbo pridedama ir programa skirta minėtų algoritmų išmėginimui. Programa turi vartotojui skirtą grafinę sąsają.



Nors grafine sąsaja ir paprasta naudotis, papildomai paaiškinti galima šį funkcionalumą:

Norint įvesti savo grafo duomenis reikia spausti mygtuką „Input graph matrix“ iššokusiame lange pasirinkti norimo grafo dydį ir spausti mygtuką „Enter“, toliau suvesti matricos duomenis, kur skaičius reiškia briaunos svorį, o bet koks tekstas reiškia, kad briauna tarp dviejų viršūnių neegzistuoja.

Galima grafą ir sugeneruoti, tam reikia parinkti pagrindinį dydį „graph size“ ir pasirinktinai „Average degree“ bei spausti „Generate“ ir jums bus sugeneruotas grafas automatiškai. ☺

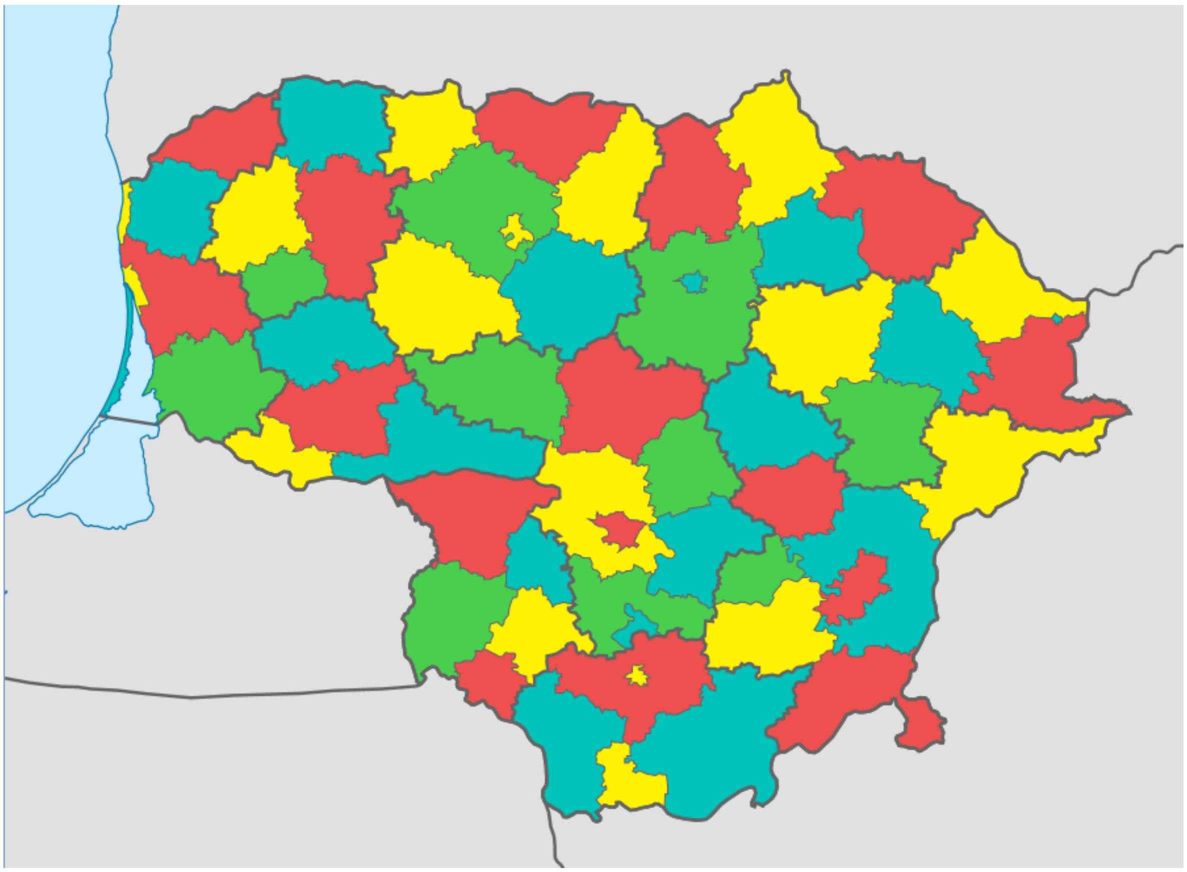
Galima ir patiems atlikti pateiktų algoritmų efektyvo testą pasirenkant intervalą nuo kokio iki kokio dydžio grafai bus generuojami. Su kiekvienu grafo dydžiu bus generuojama po 100 grafų ir kiekvienam bus pritaikomi abu spalvinimo algoritmai, o rezultatai bus išvesti į failą csv formatu. Verta priminti, kad toks testas gali užtrukti.

4 Algoritmo pritaikymas

4.1 Lietuvos žemėlapio nuspalvinimas

Kiekvienas žemėlapis yra planarus grafas, todėl kiekvieną žemėlapį galima nuspalvinti 4 skirtingomis spalvomis.

Lietuvos administracinio suskirstymo žemėlapiui pritaikius godųjį spalvinimo algoritmą rezultatą galėtume gauti tokį:



Pavyzdys 4. Lietuvos administracinio žemėlapio nuspalvinimas 4 spalvomis

5 Išvados

Šie du algoritmai yra skirtingų kraštutinių variantų realizacija.
Paieška su grįžimu visada suranda optimalų sprendimą, tačiau ši paieška užtrunka labai ilgai, todėl yra neefektyvi.
Godusis spalvinimo algoritmas, suranda sprendimą labai greitai, nors veikia efektyviai planariems grafams, tačiau ne visada tenka spalvinti tik planariuosius grafus bei surastas sprendimas ne visada yra optimalus, todėl kai kuriems taikymams gali būti visiškai netinkamas.
Yra sukurta ir daugybė kitų, efektyvesnių ar konkrečiai situacijai labiau tinkančių algoritmų ir tolimesnę pažintį šia tema galima būtų tęsti nagrinėjantis 5 spalvų teoremą[6] ar 4 spalvų teoremą[7].

6 Literatūra

Kreher, D., & Stinson, D. (n.d.). *Generation, enumeration adn Search.* Retrieved from Combinatorial Algorithms: http://www2.denizyuret.com/bib/kreher/donald1999combinatorial/combinatorialA.pdf

Zacharovas, V. (2012 m. gegužės 29 d.). *Kombinatorika ir grafų teorija.* Nuskaityta iš http://www.mif.vu.lt/~vytzach/cgi-bin/pdf_g.pl

Zeil, J. S. (n.d.). *Graph Coloring: A Backtracking Solution.* Nuskaityta iš Old Dominion University, Dept. of Computer Science: https://secweb.cs.odu.edu/~zeil/cs361/web/website/Lectures/npprobs/pages/ar01s01s01.html

[6] http://en.wikipedia.org/wiki/Five_color_theorem
[7] http://en.wikipedia.org/wiki/Four_color_theorem