**VILNIAUS UNIVERSITETAS**

**MATEMATIKOS IR INFORMATIKOS FAKULTETAS**

**PROJEKTINIS DARBAS**

**DUOMENŲ MOKSLAS 2KR. 1GR.**

**JURGIS MICKEVIČIUS**

**UŽDUOTIS 16**

**DVIGUBAI JUNGIOS KOMPONENTĖS**

**VILNIUS**

**2023**

**Turinys**

[**Įvadas** 3](#_Toc153271265)

[**2. Uždavinio formuluotė** 4](#_Toc153271266)

[**2.1 Paprastas uždavinio pavyzdys** 4](#_Toc153271267)

[**3 Realizuoti algoritmai** 4](#_Toc153271268)

[**3.1 Algoritmo veikimo principas:** 5](#_Toc153271269)

[**3.2 Pseudokodas:** 5](#_Toc153271270)

[**3.3 Taikymas algoritmo programoje:** 6](#_Toc153271271)

[**4. Eksperimentai** 6](#_Toc153271272)

[**4.1 Eksperimentas 1:** 7](#_Toc153271273)

[**4.2 Eksperimentas 2:** 7](#_Toc153271274)

[**4.3 Eksperimentas 3:** 7](#_Toc153271275)

[**4.4 Eksperimentas 4:** 8](#_Toc153271276)

[**4.5 Eksperimentas 5:** 8](#_Toc153271277)

[**4.6 Išvados:** 8](#_Toc153271278)

[**5. Sudėtingumo analizė** 9](#_Toc153271279)

[**5.1 Lentelė** 9](#_Toc153271280)

[**5.2 Diagrama sudėtingumo** 10](#_Toc153271281)

[**6. Paleidimo instrukcija** 10](#_Toc153271282)

[**7. Išvados** 12](#_Toc153271283)

[**8. Literatūra** 12](#_Toc153271284)

# **Įvadas**

Šio projektinio darbo tikslas yra susipažinti su efektyviu dvigubai jungių grafo komponenčių radimo algoritmu, taip pat parašyti programą, kuri taikydama šio algoritmo veikimo principą jį įgyvendintų, atlikti skirtingus eksperimentus, kaip keičiant viršūnių ir briaunų skaičių kinta algoritmo veikimo laikas, bei išanalizuoti patį algoritmą, tai yra jo sudėtingumą teoriška ir praktiškai, tam buvo pasitelktas laiko matavimas, per kurį algoritmas sprendžia uždavinį su atitinkamu skaičiumi n briaunų ir m viršūnių. Programa buvo sukurta rašant kodą ,,C++“ kalba. Kodo paleidimo instrukcija, kad galėtumėte išbandyti programos veikimą, yra pateikiama žemiau (,,6.Paleidimo instrukcija“).

# 

# **2. Uždavinio formuluotė**

**Dvigubai jungios komponentės:**

**Duota:** Neorientuotas grafas G, turintis viršūnių ir briaunų.

**Rasti:** Neorientuoto grafo G, turinčio n viršūnių ir m briaunų, dvigubai jungias (biconnected, angl.) komponentes, naudojant paiešką gilyn (DFS).

Uždavinyje naudosime dvigubai jungių grafo komponenčių radimo algoritmą[1][3], naudojant paiešką gilyn (DFS). Tokio tipo uždaviniai yra svarbūs, nes galime daug kur pritaikyti gyvenime, kaip pavyzdžiui eismo infrastruktūra, elektros tinklai ar kiti panašūs objektai, galime rasti tuos taškus, kurie yra kritiniai pavojaus atveju, kuomet nebegalima pasiekti tam tikros vietovės ar objekto, nes nebėra kelio iki jo.

## **2.1 Paprastas uždavinio pavyzdys**

**Duota:** Neorientuotas grafas G, turintis 10 viršūnių ir 12 briaunų.

**Briaunos**:

*1.pav Grafas G sudarytas iš 10 viršūnių ir 12 briaunų*

Paveikslėlis, kuriame yra eskizas, apskritimas, diagrama, piešimas

Automatiškai sugeneruotas aprašymas



**Rasti**: dvigubai jungias grafo G komponentes

**Atsakymas**: Gauname, kad yra 4 dvigubai jungios grafo komponentės

Paveikslėlis, kuriame yra eskizas, apskritimas, diagrama, baltas

Automatiškai sugeneruotas aprašymasPaveikslėlis, kuriame yra simbolis, apskritimas, baltas

Automatiškai sugeneruotas aprašymasPaveikslėlis, kuriame yra apskritimas, baltas, iliustracija

Automatiškai sugeneruotas aprašymasPaveikslėlis, kuriame yra eskizas, apskritimas, baltas, piešimas

Automatiškai sugeneruotas aprašymas

*2.pav 4 grafo G dvigubai jungios komponentės*

# **3 Realizuoti algoritmai**

Uždaviniui spręsti yra panaudojamas dvigubai jungių komponenčių radimo algoritmas, kuris yra skirtas rasti neorientuoto grafo dvigubai jungias ( angl. bicconected.) komponentes ir nutrūkimo taškus( angl. articulation points)[1]. Toliau pateikiamas algoritmo veikimo principas, pseudokodas ir kodas pritaikytas programuojant.

**3.1 Algoritmo veikimo principas:**

1. Naudojant paiešką gilyn (angl. depth-first search, DFS) einame per visas viršūnes ir priskiriame kiekvienai viršūnei n atitinkama numerį, pagal tai kelinta ji buvo aplankyta paieškos gilyn metu.
2. Aplankius visas viršūnes reikia nustatyti iš kiekvienos viršūnės mažiausią tašką[1] (angl. lowest point) kiekvienai viršūnei n. Tai yra mažiausia viršūnė, kurią galima pasiekti iš esamos viršūnės.
3. Turėdami paieškos gilyn priskirtus numerius kiekvienai viršūnei ir mažiausius taškus(angl.lowest points) priskirtus kiekvienai viršūnei, galime rasti nutrūkimo taškus( angl. articulation points).
4. Lyginame viršūnės eilės numerį, kuris buvo gautas paieškos gilyn metu, su mažiausia priskirta reikšme sekančiai viršūnei.
5. Jeigu mažiausia reikšmė yra didesnė arba lygi, tai eilės numeris, kuris žymi viršūnę, bus nutrūkimo taškas(angl. articulation point). Visos briaunos kurios patenka į intervalą-steką tarp lygintos mažiausios reikšmės ir eilės numerio, sudarys dvigubai jungią komponentę, tai yra nuo esamos viršūnės iki jos tėvo, sudarys dvigubai jungią komponentę, įskaitant ir pačią viršūnę.
6. Toliau vykdome algoritmą, kartojame jį tol, kol palygina visas viršūnes iš eilės pagal paieška gilyn.
7. Gauname visas dvigubai jungias komponentes ir nutrūkimo taškus(angl. articulation points).

## **3.2 Pseudokodas:**

S yra tuščias stekas, i-kintamasis (eilės numeris),v yra viršūnė ir V viršūnių aibė, num yra numeris, kuris priskiriamas paieškos gilyn metu, adj(v) sąrašas viršūnių kurios siejasi su esama viršūne, lowpt yra mažiausia reikšmė, kurią galima pasiekti iš viršūnės v[1].

i <- 0

S <- tuščias stekas

**for** **do** num(x) <- 0

**for** , **do** **if** num(x) = 0, **then** BICON(x, 0)

**procedūra** BICON(v, u)

**for** w Adj(v) **do if** num(w) =0 **then**

Viršūnė v yra nutrūkimo taškas(angl. articulation point). Suformuojame naują dvigubai jungią komponentę, kurioje taip pat yra (v,w). Pašaliname briaunas iš steko.

*3.pav Dvigubai jungių komponenčių radimo algoritmo pseudokodas*

**return**

## **3.3 Taikymas algoritmo programoje:**

void DvigubaiJungikomponente(int v, int u) {

int j = i++; //eiles numeris

numeris[v] = low\_taskas[v] = j; //numeris paieškos gilyn

aplankyta[v] = 1; //aplankymas žymimas

int vaikas = 0;

for (int w : adjList[v]) {//eis per visas virsunes

if (!aplankyta[w]) {

vaikas++;

S.push({ v, w });//pridedame į steką porą

tevas[w] = v; //nustatome tėvą, kaip viršūne

DvigubaiJungikomponente(w, v); // v nustatomas kaip w tėvas

low\_taskas[v] = std::min(low\_taskas[v], low\_taskas[w]);//nustatome mažiausią reikšmę

if ((u != -1 && low\_taskas[w] >= numeris[v]) || (u == -1 && vaikas > 1)) {

NutrukimoTaskas.insert(v);//nutrūkimo taško tikrinimas,

}

if (low\_taskas[w] >= numeris[v]) {

std::vector<std::pair<int, int>> komponente;

std::pair<int, int> briauna;

do {// iš steko visos briaunos,kurios dvigubai jungios, kol prieiname (v,w)

briauna = S.top();

S.pop();// iš steko šaliname briaunas

komponente.push\_back(briauna);

} while (briauna != std::make\_pair(v, w));

dvigubai\_jungios\_komponentes.push\_back(komponente);//pridedame prie komponente

}

}

else if (w != u && numeris[w] < numeris[v]) {

S.push({ v, w }); // jei w yra galinė briauna ne tėvas v, tai į steką įdedame

low\_taskas[v] = std::min(low\_taskas[v], numeris[w]);// atnaujiname lowpt reikšmę

}

}

}

*4.pav Dvigubai jungių komponenčių algoritmo taikymo kodas programoje*

# **4. Eksperimentai**

Atlikau 5 eksperimentus išsiaiškinti, kaip kinta kodo veikimo greitis esant rankiniam duomenų įvedimui, kai žinome briaunas, tada kai žinome briaunas, bet nuskaitome iš failo ir kai programa automatiškai generuoja. Rankiniam įvedimui naudojau paprasto uždavinio pavyzdžio pateikto grafo briaunas, iš failo pateikiau tokias pat briaunas, kaip ir rankiniam įvedimui, kad palyginti greitį, o automatiniam generavimui pasirinkau, jog pirmu variantu bus 10 briaunų ir 12 viršūnių, tam, kad būtų galima palyginti kiek skiriasi laikas, jeigu žinai briaunas ir jeigu automatiškai generuoja programa grafą. Kiti eksperimentai buvo atliekami su automatiniu generavimu, tačiau keičiant viršūnių skaičių ir briaunų skaičių. Toliau pateikiami 5 eksperimentai ir kiek laiko užtruko programa, laikas nurodomas atlikus kiekvieną kartą 5 matavimus ir paskaičiavus vidurkį, sekundėmis.

## **4.1 Eksperimentas 1:**

Pirmas eksperimentas, naudojau rankinį įvedimą programoje, pasirinkau grafą, kuris pateiktas ,,Paprastas uždavinio sprendimas“. Turime 10 viršūnių ir 12 briaunų:

Paveikslėlis, kuriame yra tekstas, ekrano kopija, programinė įranga, Šriftas

Automatiškai sugeneruotas aprašymas

*5.pav Eksperimentas su 10 viršūnių ir 12 briaunų(rankinis įvedimas)*

Gavome, kad laikas buvo: 0,0678 sekundės

Penkių paleidimų vidurkis: 0,0651sekundės.

## **4.2 Eksperimentas 2:**

Antram eksperimentui naudojau duomenis tokius pat, kaip pirmame eksperimente, bet nuskaičiau iš failo, tam, kad palyginti, kiek skiriasi laikas.

**Paveikslėlis, kuriame yra tekstas, ekrano kopija, programinė įranga, Šriftas

Automatiškai sugeneruotas aprašymas**

*6.pav Eksperimentas su 10 viršūnių ir 12 briaunų(skaitymas iš failo)*

Gavome, kad laikas buvo: 0,0901 sekundės.

Penkių paleidimų vidurkis: 0,0749 sekundės.

## **4.3 Eksperimentas 3:**

Trečiam eksperimentui naudojau automatinį grafo generavimą ir pasirinkame, kad būtų 10 viršūnių ir 12 briaunų, tam, kad palyginti galėtume, laiko skirtumą, kai programa pati sugeneruoja grafą. Paveikslėlis, kuriame yra tekstas, ekrano kopija, monitorius, programinė įranga

Automatiškai sugeneruotas aprašymas

*7.pav Eksperimentas su 10 viršūnių ir 12 briaunų(automatinis generavimas)*

Gavome, kad laikas buvo: 0,033 sekundės.

Penkių paleidimų vidurkis: 0,0367 sekundės.

## **4.4 Eksperimentas 4:**

Ketvirtam eksperimentui naudojau automatinį grafo generavimą ir pasirinkau, kad būtų 12 viršūnių ir 20 briaunų.

**Paveikslėlis, kuriame yra tekstas, programinė įranga, Šriftas, Multimedijos programinė įranga

Automatiškai sugeneruotas aprašymas**

*8.pav Eksperimentas su 12 viršūnių ir 20 briaunų(automatinis generavimas)*

Gavome, kad laikas buvo: 0,0649 sekundės.

Penkių paleidimų vidurkis: 0,0572 sekundės.

## **4.5 Eksperimentas 5:**

Penktam eksperimentui pasirinkau, kad grafas turės 20 viršūnių ir 45 briaunas, naudojau automatinį generavimą.

Paveikslėlis, kuriame yra tekstas, programinė įranga, ekrano kopija, Multimedijos programinė įranga

Automatiškai sugeneruotas aprašymas

*9.pav Eksperimentas su 20 viršūnių ir 45 briaunomis(automatinis generavimas)*

Gavome, kad laikas buvo: 0,1134 sekundės.

Penkių paleidimų vidurkis: 0,0987 sekundės.

## **4.6 Išvados:**

Atlikus penkis eksperimentus su skirtingais įvesties būdais bei automatiniu generavimu galime pastebėti, kad įvedimas ranka briaunų ir automatinis generavimas su 10 viršūnių ir 12 briaunų trunka panašiai, skiriasi tik porą šimtųjų sekundžių dalies, tačiau automatinis generavimas vis tiek buvo greitesnis apie 40%. Lyginant tuos pačius duomenis, kai saugomi faile ir įrašomi ranka, pastebime, kad iš failo trunka ilgiau, nes skirtumas lyginant vidurkius 5 matavimų yra apie 0,01sekundės, kadangi ranka vidutiniškai 0,0651 sekundės, o iš failo 0,0749sekundės. Tai gali būti dėl to, kad reikia duomenis pasiimti iš failo analizės metu, o ne tiesiogiai iš kompiliatoriaus eilutės, tad vien dėl to gali trukti ilgiau. Analizuojant automatinį generavimą su didesniu skaičiumi briaunų ir viršūnių, galima pastebėti, kad laikas pailgėja, nes grafe padaugėja briaunų ir viršūnių skaičius, kaip 4 ir 5 eksperimentas, ketvirtame naudojame 12 viršūnių ir 20 briaunų, o 5 eksperimente jau 20 viršūnių ir 45 briaunas, o vidutinis laikas beveik dvigubai skiriasi.

Apibendrinant visus eksperimentus, galima prieiti išvadą, kad geriausias laikas nepriklausomai nuo viršūnių ir briaunų skaičiaus yra su automatiniu grafo generavimu, o blogiausias laikas yra kai skaitome duomenis iš failo.

# **5. Sudėtingumo analizė**

Sudėtingumo analizės metu analizavau, dvigubai jungių grafo komponenčių algoritmo sudėtingumą, atlikau analizę, kada viršūnių skaičius yra toks pat, naudojau n=200, o briaunų skaičių didinau nuo 200 kaskart pridėdamas po 50, tai yra 200, 250, 300 ir taip toliau iki kol pasieks 1650, nors maksimaliai galėtume paimti, briaunų skaičių iki 19900, bet kadangi artėjant prie maksimumo programa ne visada sugeneruoja tokį grafą, kuriame tikrai bus nutrūkimo taškas, tad sudėtingumo analizei buvo paimtas ne maksimalus skaičius, o mažesnis, tačiau taip pat tinkamas, nes buvo atlikti 30 laiko matavimų, kiekvieną matavimą kartojau penkis kartus, tam, kad paskaičiuoti vidurkį, kas leido daryti objektyvias išvadas apie algoritmo sudėtingumą.

## **5.1 Lentelė**

*1.lentelė Algoritmo sudėtingumo laiko analizės lentelė*

|  |  |  |
| --- | --- | --- |
| **Nr.** | **Briaunų skaičius** | **Laikas** |
| **1** | 200 | 0,3790 |
| **2** | 250 | 0,3954 |
| **3** | 300 | 0,4589 |
| **4** | 350 | 0,4598 |
| **5** | 400 | 0,4917 |
| **6** | 450 | 0,5241 |
| **7** | 500 | 0,5405 |
| **8** | 550 | 0,5657 |
| **9** | 600 | 0,5978 |
| **10** | 650 | 0,6123 |
| **11** | 700 | 0,6267 |
| **12** | 750 | 0,6373 |
| **13** | 800 | 0,6684 |
| **14** | 850 | 0,6851 |
| **15** | 900 | 0,7217 |
| **16** | 950 | 0,7454 |
| **17** | 1000 | 0,7691 |
| **18** | 1050 | 0,7928 |
| **19** | 1100 | 0,7953 |
| **20** | 1150 | 0,8157 |
| **21** | 1200 | 0,8361 |
| **22** | 1250 | 0,8411 |
| **23** | 1300 | 0,8478 |
| **24** | 1350 | 0,8496 |
| **25** | 1400 | 0,8701 |
| **26** | 1450 | 0,8876 |
| **27** | 1500 | 0,8971 |
| **28** | 1550 | 0,9014 |
| **29** | 1600 | 0,9175 |
| **30** | 1650 | 0,9654 |

## **5.2 Diagrama sudėtingumo**

*10.pav Algoritmo sudėtingumo laiko analizės diagrama*

Atlikus laiko matavimą buvo pastebėta, kad dvigubai jungių grafo komponenčių algoritmo sudėtingumas analizuojant laiką kinta ganėtinai tolygiai, tai yra didinant briaunų skaičių proporcingai auga ir laikas, tad galime teigti, kad esant pastoviam viršūnių skaičiui, briaunų skaičiaus didinimas algoritmo sudėtingumui didelės įtakos nedaro, kadangi laikas didėja tolygiai. Šio algoritmo sudėtingumas yra , V yra viršūnės, o E yra briaunos.

# **6. Programos naudojimo instrukcija**

Norėdami išbandyti praktiškai, kaip veikia šis algoritmas pridedu instrukciją, kaip pasileisti kodą. Visą projektą sudaro keletas atskirų failų:

* Algoritmai.cpp (pagrindinis kodas parašytas c++ programavimo kalba).
* Grafas.txt (skaitymui iš failo arba pasirinkus ranka, tiesiog atsidarius „Grafas.txt“ nukopijuoti duomenis. 10 viršūnių yra)

Paleidimo instrukcija:

1. Reikia atsidaryti internetinę svetainę, kuri leidžia paleisi, c++ kodą, tai yra: <https://www.onlinegdb.com/online_c++_compiler>
2. Tuomet įkelti ,,Algoritmai.cpp“ failą, tiesiog jį pridėti, paspaudus ant nuotraukoje esančio mygtuko.

Paveikslėlis, kuriame yra tekstas, programinė įranga, ekrano kopija, Multimedijos programinė įranga

Automatiškai sugeneruotas aprašymas

1. Taip pat reikia įsikelti ,,Grafas.txt“ failą, tam, kad būtų galimybė skaityti duomenis iš failo. Atliekame šį veiksmą, kaip ir prieš tai.
2. Įsikėlus šiuos du failus reikia išvalyti main.cpp, kadangi šio failo ištrinti negalima, nes naudojame internetinę aplinką, kad galėtume paleisti kodą, tad viskas kas yra main.cpp ištriname.

Paveikslėlis, kuriame yra tekstas, ekrano kopija, programinė įranga, Kompiuterio piktograma

Automatiškai sugeneruotas aprašymas

Paveikslėlis, kuriame yra tekstas, programinė įranga, Multimedijos programinė įranga, Kompiuterio piktograma

Automatiškai sugeneruotas aprašymas

1. Atlikus šiuos veiksmus, galime spausti žalią mygtuką, kuris yra viršuje ,,Run‘‘

Paveikslėlis, kuriame yra tekstas, ekrano kopija, programinė įranga, Kompiuterio piktograma

Automatiškai sugeneruotas aprašymas

1. Toliau veiksmai bus nurodomi programoje, kompiliatoriaus apačioje:

Paveikslėlis, kuriame yra tekstas, ekrano kopija, programinė įranga, kompiuteris

Automatiškai sugeneruotas aprašymas

1. **Išvados**

Atlikus projektinį darbą, kuriame susipažinau ir analizavau ,,Dvigubai jungios komponentės‘‘ uždavinį pastebėjau, kad šis uždavinys yra labai aktualus mūsų dabartinam pasauliui, nes padeda nustatyti kritines infrastruktūros vietas. Kalbant apie patį uždavinio sprendimą, galime pastebėti, kad geriausiai kodas sprendžia užduotį, kai duomenis saugome ne faile, o pateikiame rankiniu įvedimu arba automatiškai programa generuoja grafą ir tada sprendžia uždavinį. Uždavinio sprendimo laikas taikant dvigubai jungių komponenčių radimo algoritmą tolygiai didėja daugėjant briaunų, taip pat jeigu bandome automatiškai generuoti grafą, kuris turi n viršūnių ir pagal n maksimumą m briaunų, dažniausiai nutrūkimo taškų (angl. articulation points) neras, kadangi jie grafe neegzistuos..

# **8. Literatūra**

1. E.M. Reingold, J. Nievergelt, N. Deo, Combinatorial Algorithms: Theory and Practice, Prentice-Hall, Englewood Cliffs, NJ, 1977, pp. 331—335.
2. N. Christofides, Graph Theory, Academic Press, 1975, pp. 331—336.
3. V. Dičiūnas. ‚‚Algoritmų analizės pagrindai‘‘ 77psl.

<https://www.mif.vu.lt/katedros/cs/Asmen/algoritmu_analize.pdf>

1. A. Juozapavičius. Duomenų struktūros ir efektyvūs algoritmai, Vilnius: Petro ofsetas, 2007.
2. biconnected\_components.pdf

<https://drive.google.com/drive/u/0/home>

E

C

G

A

F

D

B

J

I

H

E

C

G

A

F

D

B

H

I

B