**VILNIAUS UNIVERSITETAS**

**MATEMATIKOS IR INFORMATIKOS FAKULTETAS**

**PROJEKTINIS DARBAS**

**DUOMENŲ MOKSLAS 2KR. 1GR.**

**JURGIS MICKEVIČIUS**

**UŽDUOTIS 16**

**DVIGUBAI JUNGIOS KOMPONENTĖS**

**VILNIUS**

**2023**

**Turinys**

1.

[**Įvadas** 3](#_Toc153228557)

[**2. Uždavinio formuluotė** 4](#_Toc153228558)

[**2.1 Paprastas uždavinio pavyzdys** 4](#_Toc153228559)

[**3 Realizuoti algoritmai** 4](#_Toc153228560)

[**3.1 Algoritmo veikimo principas:** 5](#_Toc153228561)

[**3.2 Pseudokodas:** 5](#_Toc153228562)

[**3.3 Taikymas algoritmo programoje:** 6](#_Toc153228563)

[**4. Eksperimentai** 6](#_Toc153228564)

[**4.1 Eksperimentas 1:** 7](#_Toc153228565)

[**4.2 Eksperimentas 2:** 7](#_Toc153228566)

[**4.3 Eksperimentas 3:** 7](#_Toc153228567)

[**4.4 Eksperimentas 4:** 8](#_Toc153228568)

[**4.5 Eksperimentas 5:** 8](#_Toc153228569)

[**4.6 Išvados:** 8](#_Toc153228570)

[**5. Sudėtingumo analizė** 8](#_Toc153228571)

[**6. Išvados** 8](#_Toc153228572)

[**7. Paleidimo instrukcija** 8](#_Toc153228573)

[**8. Literatūra** 8](#_Toc153228574)

# **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, bei išanalizuoti patį algoritmą, tai yra jo sudėtingumą teoriška ir praktiškai. Programa bus įgyvendinam rašant kodą ,,C++“ kalba. Kodo paleidimo instrukcija yra pateikiama žemiau (,,7.Paleidimo instrukcijos“).

# **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], 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 komponentes

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). 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 mažiausias viršūnes priskirtas 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.

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.

**return**

*3.pav dvigubai jungių komponenčių radimo algoritmo pseudokodas*

## **3.3 Taikymas algoritmo programoje:**

void DvigubaiJungikomponente(int v, int u) {

int j = i++;

numeris[v] = low\_taskas[v] = j;

aplankyta[v] = 1;

int vaikas = 0;

for (int w : adjList[v]) {

if (!aplankyta[w]) {

vaikas++;

S.push({ v, w });

tevas[w] = v;

DvigubaiJungikomponente(w, v);

low\_taskas[v] = std::min(low\_taskas[v], low\_taskas[w]);

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

NutrukimoTaskas.insert(v);

}

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

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

std::pair<int, int> briauna;

do {

briauna = S.top();

S.pop();

komponente.push\_back(briauna);

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

dvigubai\_jungios\_komponentes.push\_back(komponente);

}

}

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

S.push({ v, w });

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

}

}

}

*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 grafa. 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 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

Gavome, kad laikas buvo: 0,2526 sekundės

Penkių paleidimų vidurkis: 0,2865 sekundės.

## **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, Multimedijos programinė įranga

Automatiškai sugeneruotas aprašymas**

Gavome, kad laikas buvo: 0,4137 sekundės.

Penkių paleidimų vidurkis: 0,5749 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, Šriftas, programinė įranga

Automatiškai sugeneruotas aprašymas

Gavome, kad laikas buvo: 0,1459 sekundės.

Penkių paleidimų vidurkis: 0,2427 sekundės.

## **4.4 Eksperimentas 4:**

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

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

Automatiškai sugeneruotas aprašymas**

Gavome, kad laikas buvo: 0,3167 sekundės.

Penkių paleidimų vidurkis: 0,3244 sekundės.

## **4.5 Eksperimentas 5:**

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

Paveikslėlis, kuriame yra tekstas, Šriftas, ekrano kopija

Automatiškai sugeneruotas aprašymas

Gavome, kad laikas buvo: 0,3838 sekundės.

Penkių paleidimų vidurkis: 0,6851 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 12%. Lyginant tuos pačius duomenis, kai saugomi faile ir įrašomi ranka, kai programa paleidžiama, pastebime, kad iš failo grafas ir jo analizė trunka ilgiau, nes skirtumas lyginant vidurkius 5 matavimų yra vos ne dvigubai ilgiau, kadangi ranka vidutiniškai 0,2865 sekundės, o iš failo 0,5749 sekundės. 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 labai pailgėja, nes grafe padaugėja briaunų ir viršūnių skaičius, kaip 4 ir 5 eksperimentas, 4 naudojame 12 viršūnių ir 20 briaunų, o 5 eksperimente jau 20 viršūnių ir 45 briaunas, o vidutinis laikas skiriasi dvigubai.

Apibendrinant visus eksperimentus, galima prieiti išvadą, kad geriausias laikas nepriklausomai nuo viršūnių ir briaunų skaičiaus yra su automatiniu grafo generavimu.

# 

# **5. Sudėtingumo analizė**

Sudėtingumo analizės metu analizavau, dvigubai jungių grafo komponenčių algoritmo sudėtingumą, keletą aspektų, pirmą atlikau analizę, kada viršūnių skaičius yra toks pat, naudojau n=650, o briaunų skaičių didinau nuo 650 kaskart pridėdamas po 250, tai yra 900, 1150, 1400 ir taip toliau iki kol pasieks 12900, nors maksimaliai galėtume paimti, briaunų skaičių iki 210925, 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 atlikta 40 laiko matavimų, kiekvieną matavimą kartojau tris kartus, tam, kad paskaičiuoti vidurkį, kas leido daryti objektyvias išvadas apie algoritmo sudėtingumą.

Kita dvigubai jungių grafo komponenčių algoritmo sudėtingumo analizė buvo atlikta matuojant laiką tik su dvigubai didesniu skaičiumi viršūnių, tai yra n=1300, pradinis skaičius buvo pasirinktas taip pat proporcingai didesnis, negu viršūnių skaičius tai yra 1300 ir kaskart didinau po 500, maksimalus briaunų skaičius buvo pasirinktas 30800 briaunų, nors maksimaliai galėtų būti 844350 briaunų, tačiau tokie skaičiai pasirinkti atsižvelgiant į pirmos sudėtingumo analizės duomenis, tai yra taikant proporciją, kad galima būtų po to objektyviai lyginti rezultatus, viso atlikau 60 matavimų laiko. Taip pat buvo paskaičiuotas kiekvienam matavimui vidurkis 3 matavimų su kiekvienu skaičiumi briaunų.

# **6. Išvados**

# **7. Paleidimo instrukcija**

# **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‘‘ PUSLAPIAI???????????????????????????????????????.

https://www.mif.vu.lt/katedros/cs/Asmen/algoritmu\_analize.pdf

E

C

G

A

F

D

B

J

I

H

E

C

G

A

F

D

B

H

I

B