**VILNIAUS UNIVERSITETAS**

**MATEMATIKOS IR INFORMATIKOS FAKULTETAS**

**PROJEKTINIS DARBAS**

**DUOMENŲ MOKSLAS 2KR. 1GR.**

**JURGIS MICKEVIČIUS**

**UŽDUOTIS 16**

**DVIGUBAI JUNGIOS KOMPONENTĖS**

**VILNIUS**

**2023**

**TURINYS**

1. **Įvadas**............................................................................................................................................................ **2**
2. **Uždavinio formuluotė**.................................................................................................................................. **3**

**2.1Paprastas uždavinio pavyzdys**.....................................................................................................................**4**

1. **Realizuoti algoritmai**....................................................................................................................................**4**
2. **Eksperimentai**
3. Sudėtingumo analizė
4. Išvados
5. Paleidimo instrukcija
6. Literatūra

**Įvadas**

Šio projektinio darbo tikslas yra susipažinti su efektyviu dvigubai jungių grafo komponenčių radimo algoritmu, dar vadinamu Tarjano algoritmu, taip pat parašyti programą, kuri taikydama šio algoritmo veikimo principą jį įgyvendintų, tai pat išanalizuoti algoritmą, tai yra jo sudėtingumą teoriška ir praktiškai. Programa bus įgyvendinam rašant kodą ,,C++“ kalba. Kodo paleidimo instrukcija yra pateikiama žemiau.

**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, naudojant paiešką gylyn (DFS). Šis algoritmas dar yra vadinama Tarjano algoritmu. Tokio tipo uždaviniai yra pritaikomi dažnai ir mūsų gyvenime, kaip 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.

**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 artikuliacinius-nutrūkimo taškus( angl. articulation points). Toliau pateikiamas algoritmo veikimo principas, pseudokodas ir kodas pritaikytas programuojant.

**3.1 Jo veikimo principas**:

1. Naudojant paiešką gilyn (angl. depth-first search, DFS), 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 artikuliacinius /nutrūkimo taškus( angl. articulation points).
4. Lyginame viršūnės numerį su mažiausia priskirta reikšme.
5. Jeigu mažiausia reikšmė yra didesnė arba lygi numeriui, tai tas eilės numeris bus artikuliacinis taškas. 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.
6. Toliau vykdome algoritmą, kartojame jį tol kol palygina visas viršūnes iš eilės pagal paieška gilyn.
7. Gauname visas dvigubai jungias komponentes ir artikuliacinius taškus.

**3.2 Pseudokodas**:

S yra tuščias stekas, v yra viršūnės ir V viršūnių aibė, num yra numeris, adj(v) gražina gretimas viršūnes, lowpt yra mažiausia reikšmė.

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 artikuliacinis/lūžio taškas. 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 Kodas programoje:**

void DvigubaiJungikomponente(int v, int u) {

int j = i++;

num[v] = lowpt[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);

lowpt[v] = std::min(lowpt[v], lowpt[w]);

if ((u != -1 && lowpt[w] >= num[v]) || (u == -1 && vaikas > 1)) {

arrtikuliacinisTaskas.insert(v);

}

if (lowpt[w] >= num[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 && num[w] < num[v]) {

S.push({ v, w });

lowpt[v] = std::min(lowpt[v], num[w]);

}

}

}

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

**4. Eksperimentai**

Atlikau 4 eksperimentus išsiaiškinti, kaip kinta kodo veikimo greitis esant rankiniam duomenų įvedimui, kai žinome duomenis ir kai programa automatiškai generuoja. Rankiniam įvedimui naudojau paprasto uždavinio pavyzdžio pateikto grafo briaunas, o automatiniam generavimui pasirinkau, jog bus pirmu variantu 10 briaunų ir 12 viršūnių, tam, kad būtų galima palyginti kiek skiriasi laikas, jeigu žinai briaunas ir jeigu automatiškai generuoja programa grafa. Antru variantu su automatiniu grafo generavimu pasirinkau, kad būtų 10 viršūnių ir 24 briaunos. Trečiu eksperimento variantu su automatiniu generavimu pasirinkau 10 viršūnių ir 50 briaunų.

**4.1 Eksperimentas 1:**

**4.2 Eksperimentas 2:**

**4.3 Eksperimentas 3:**

**4.4 Eksperimentas 4:**

**4.5 Išvados:**

**5. Sudėtingumo analizė**

**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