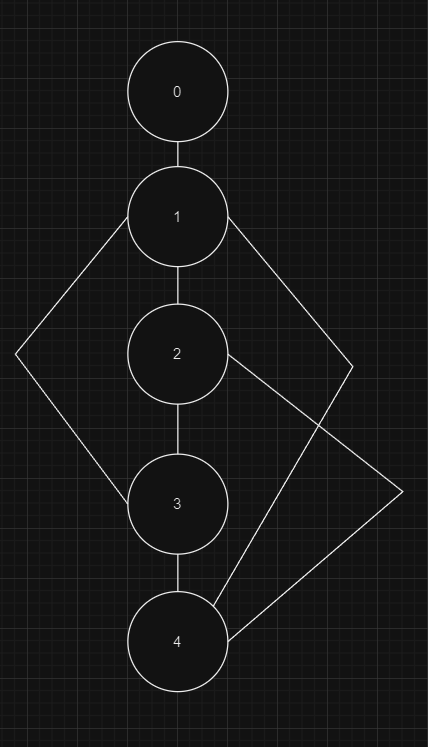
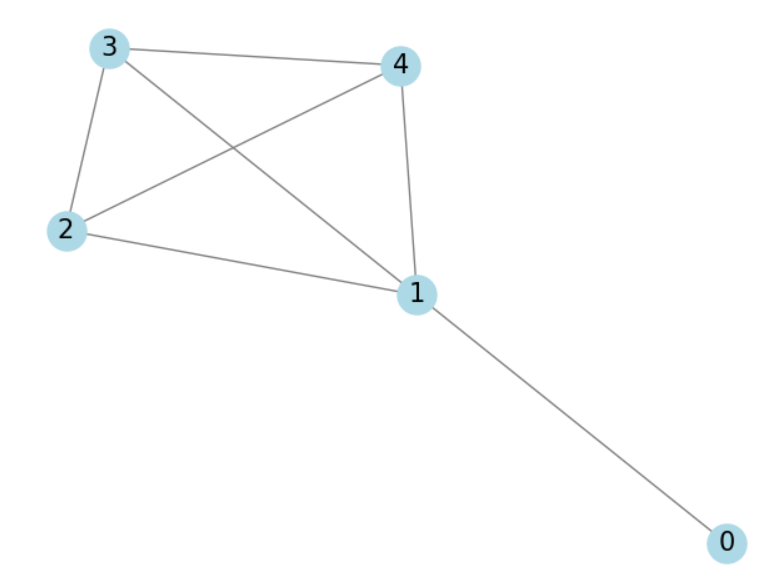
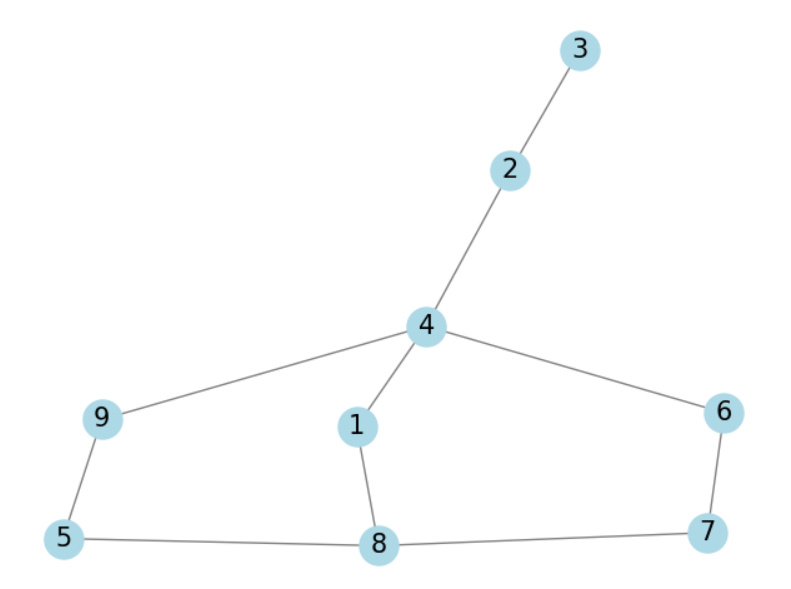
Vilniaus Universitetas  
  
Matematikos ir informatikos Fakultetas  
Informatikos Katedra  
3 Kursas 2 Grupe  
**Gvidonas Pupelis**   
Užduotis: Projektas - 16  
DVIGUBAI JUNGIOS KOMPONENTĖS

Užduotis:  
DVIGUBAI JUNGIOS KOMPONENTĖS

Duota: Neorientuotas grafas G, turintis n viršūnių ir m briaunų..

Rasti: Duoto grafo dvigubai jungias (biconnected, angl.) komponentes, naudojant paiešką gylyn (DFS).

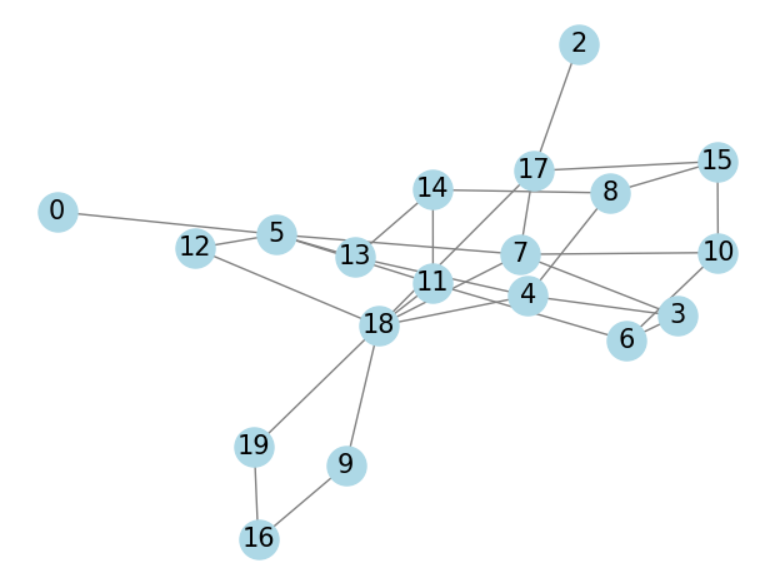
Realizuoti dvigubai jungių grafo komponenčių radimo algoritmą ir ištirti jo sudėtingumą  
Šis algoritmas dažnai naudojamas patikrinti tinklo patikimumui. Nes dvigubai jungi dalis daug rečiau visa suges. Taip yra dėl to nes dvigubai jungi komponente turi du priėjimos prie to pačio tinklo nario. Tad jei vienas narys sugenda yra galimybe jį apeiti per kitą vietą. Tad mes galime sukonstruoti grafa kuris butu tinklo atitikmuo ir tada galėsime pritaike šį algoritma matyti kurios grafo dalys yra kritinės(Artikuliacijos taškai) ir kurios dalys tinklo yra pakankamai gerai apsaugotos nuo gedimu.  
  
Pavizdys:  
Tarkime turime pagrindini serveri(3) ir jisai sujungtas su trimis namais, kurie irgi yra sujungti tarpusavyje(1)(2)(4). Ir yra vienas namas kuris yra toliau(0) ir yra sujungtas tiktais su vienu namu(1). Grafas atrodytu taip:  
  
  
Taigi padare DFS   
  
Gausime medi dešinėje taigi dabar galime matyti kad iš ketvirtos galime grįžti tik iki pirmos per kitas krastines kurios nėra atradimo kraštinės ir kadangi tai mažiausia pasiekiama viršūnė iš (4). Tai yra artikuliacijos taškas nes iš žemiau esančių briaunų negalime pasiekti aukštesnės viršūnės ne per pagrindini kelia. tad čia reikia atskirti grafa. Taigi viena jungi komponente yra 1,2,3,4. Tada musu grafas sumažėja iki 0 ir 1 ir kadangi yra tik viena kraštinė tai irgi yra jungi komponente.  
  
Taigi musu pavyzdžio 3 namai yra gana saugus nes net jei kitas kuris nors namas atsijungtu jie galetu testi savo darba tinkle tačiau 0 yra priklausomas nuo 1 namo.   
  
Experimentas 1:  
Turime grafa:  
  
Algoritmo veikimo laikas: 0.000000 sekundes

Dvigubai sujungtos komponentės (viršūnės):

{2, 3}

{2, 4}

{1, 4, 5, 6, 7, 8, 9}

Experimentas 2:  


Turint toki grapha:  
Algoritmo veikimo laikas: 0.000000 sekundes

Dvigubai sujungtos komponentės (viršūnės):

{17, 2}

{16, 9, 18, 19}

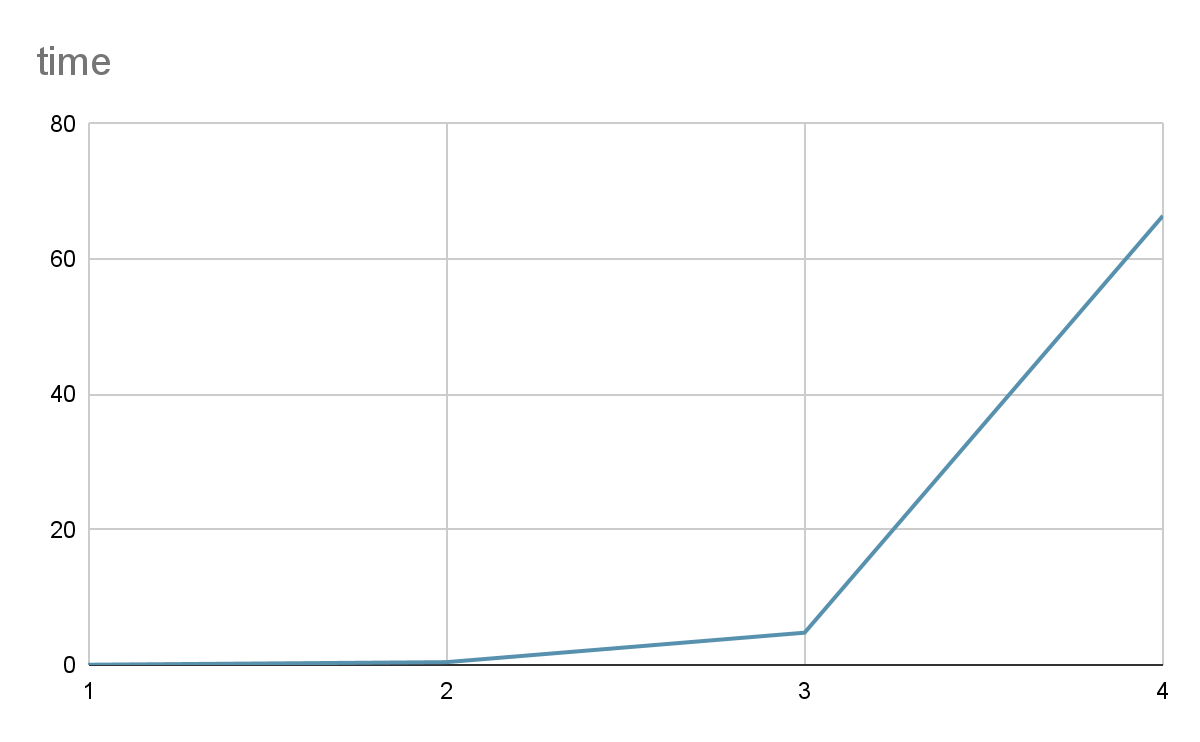
{3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 17, 18}

{0, 5}

Tada sugeneruojami labai dideli atsitiktiniai grafai ir žiūrime kaip greitai veikia musu programa kiekvienas grafas tures n virsuniu ir 1.5\*n krastinius:  
1 = (10000, 15000) = Algoritmo veikimo laikas: 0.024023 sekundes  
2 = (100000, 150000) = Algoritmo veikimo laikas: 0.405999 sekundes

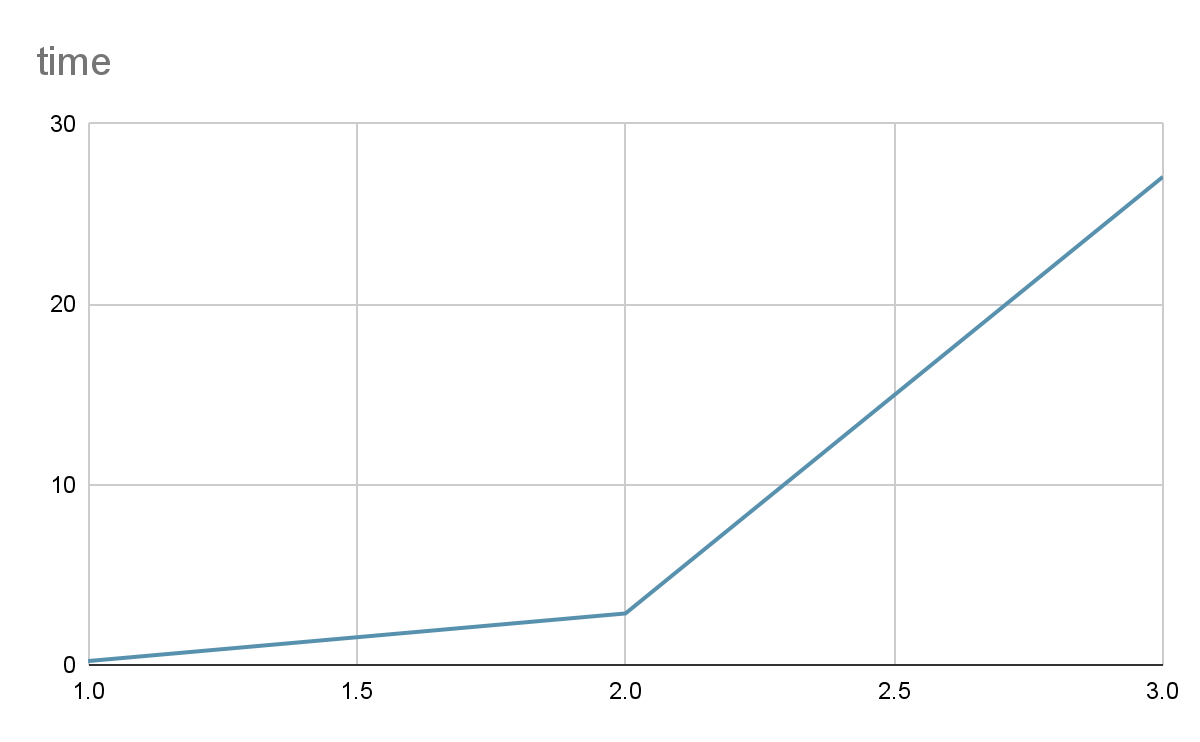
3 = (1000000, 1500000) = Algoritmo veikimo laikas: 4.762969 sekundes

4 = (10000000, 15000000) = Algoritmo veikimo laikas: 66.402997 sekundes



Jeigu didiname krastiniu kieki:  
1 = (10000, 150000) = Algoritmo veikimo laikas: 0.213000 sekundes

2 = (10000, 1500000) = Algoritmo veikimo laikas: 2.853000 sekundes  
3 = (10000, 15000000) = Algoritmo veikimo laikas: 27.062085 sekundes



Is situ grafu ir praktiniu duomenų matome kad musu duotasis algoritmas turi O(n + V) sunkuma nes peklus duomenu kieki 10 kartu laikas irgi maždaug padidėja 10 kartu. Ir nesvarbu ar didinam krastines ar virsunes.