VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
MATEMATIKOS IR INFORMATIKOS
METODIKOS KATEDRA

Martynas Sabaliauskas

MEDŽIŲ VIZUALIZACIJOS ALGORITMAI IR JŲ TAIKYMAS

Magistro baigiamasis darbas

Leidžiu ginti .....
Darbo vadovas **prof. Eugenijus Manstavičius**

Vilnius 2012

TURINYS

1. Įvadas

Grafų vizualizacija – tai matematikos ir kompiuterių mokslo sritis, kurioje taikomi geometrijos, skaičiuojamosios matematikos ir įvairios informacijos interpretavimo, transformacijos metodai. Sukaupti duomenys, susidedantys iš tam tikrų objektų (grafo viršūnių) bei sąryšių tarp tų objektų (briaunų), dažniausiai vaizduojami plokštumoje arba erdvėje. Aišku, kad yra be galo daug skirtingų būdų tą patį grafą vizualiai pateikti kurioje nors aplinkoje, todėl išskiriami šie estetiniai reikalavimai [1]: minimalus briaunų susikirtimų skaičius, tolygus viršūnių pasiskirstymas, maža grafo briaunų ilgių imties dispersija ir, jei yra galimybė, simetrinis grafo vaizdavimas.

Siekiant vizualiai interpretuoti tam tikrą informaciją, pirmiausia reikia atsižvelgti į nagrinėjamų duomenų struktūrą. Jei struktūra dinaminė, tai vizualizacijos procesas tampa ypač sudėtingas ir nedeterminuotas laiko atžvilgiu. Priešingu atveju duomenys, atitinkantys grafus, vaizduojami statiniu pavidalu. Paprastai pasirenkama kuri nors vizualizacijos strategija: grafų vaizdavimas tiesiomis, ortogonaliomis, lenktomis linijomis; viršūnės išdėstomos hierarchiškai, radialiniu būdu ar tam tikroje gardelėje; grafas vaizduojamas plokštumoje, erdvėje ar užtemptas ant tam tikro paviršiaus ir t. t.

Mes nagrinėsime nedidelės apimties statinės struktūros duomenis, kurie grafų teorijoje atitinka medžius ir pagal pirmus tris vizualizacijos estetinius reikalavimus [1] vaizduosime juos plokštumoje ir erdvėje radialiniu būdu. Tačiau prieš pateikdami šio darbo rezultatus ir naujas idėjas, visų pirma apžvelkime medžių vizualizacijos algoritmus, labiausiai susijusius su nagrinėjama tema.

Binariųjų medžių vaizdavimas hierarchiniu būdu pirmą kartą pasiūlytas Wetherell ir Shannon [2] 1979 m. Realizavus šį Paskalio kalba parašytą algoritmą, medžio viršūnės buvo grupuojamos pagal lygius ir išdėstomos horizontalių lygiagrečių tiesių sistemoje. Vėliau tą patį algoritmą patobulino Reingold ir Tilford [3] sutaupydami operatyviosios atminties kompiuteryje. Walker [4] apibendrino dvejetainių medžių vizualizacijos algoritmą ir jį pritaikė bet kokiems n-tosios eilės numeruotiesiems medžiams vizualizuoti atliekant $O(n^2)$ žingsnių. Galiausiai 2002 m. Buchheim [5] pagerino Walker algoritmą, kuris buvo realizuojamas sugaištant tik $O(n)$ žingsnių.

Kompiuterine grafika paremta radialinio medžių vaizdavimo idėja pirmą kartą aprašyta 1992 m. australų informatiko Eades [6]. Šis metodas patogus tuo, kad koncentriniuose apskritimuose išdėstant medžio viršūnes, aiškiai atsispindi jų hierarchijos ir grafas plokštumoje „sutalpinamas“ į skritulį. Vis dėl to ne visada paranku medį vaizduoti radialiniu būdu, ypač jei egzistuoja viršūnės, turinčios daug palikuonių. Šią problemą nagrinėjo Melancon ir Herman [7], kurie 1998 m. pasiūlė tokius medžius vaizduoti žiediniu būdu. Vietoj vienos griežtos hierarchijos, pagal jų metodą, medis vizualiai interpretuojamas kaip grupė hierarchijų tarp viršūnių ir jų palikuonių, kurie ratu išdėstomi apie bendrą tėvą.

Virtualioje trimatėje erdvėje vaizduoti medžius 1991 m. pasiūlė Robertson [8]. Jo idėja paremta viršūnių išdėstymu hierarchijomis, atitinkančiomis lygiagrečių plokštumų

sistemą. Tarp hierarchijų pomedžiai vaizduojami ant kūgių paviršių, todėl algoritmas taip ir pavadintas – kūginiai medžiai.

Šiame darbe, remiantis radialiniu medžių vaizdavimu [6], Maple programos aplinkoje buvo realizuotas algoritmas, skirtas numeruotųjų medžių vizualizacijai plokštumoje. Po to, atsitiktinai generuojant tūkstančius įvairios eilės medžių, empiriškai nustatyta, jog ne visais atvejais tenkinami estetiniai reikalavimai [1]. Iškilusi problema spręsta įvedant papildomus parametrus, suteikiančius medžiui lankstumo, ilgiausius takus vaizduojant vienoje iš koncentrinių elipsių, medį „užtempiant“ ant erdvinių spiralinių paviršių.

Taigi mes apibendrinome radialino medžių vaizdavimo algoritmą [6]; pasiūlėme medžio centro paieškos metodą [9] taikyti ilgiausių takų radimui; sukūrėme iki šiol neaprašytą laurų vainiko algoritmą, skirtą vaizduoti įvairiems medžiams plokštumoje; patogumo dėlei pritaikėme Priuferio sugalvotą kodą [10] medžiams generuoti bei išvesti apibrėžto medžio Priuferio kodą; modifikuodami radialinį medžių vizualizacijos algoritmą pavaizdavome medžius koncentrinių elipsoidų sistemoje. Darbe pasiūlyta originali idėja vizualizuoti paiešką į plotį ir į gylį įterpiant papildomas procedūras, tam tikruose algoritmo realizacijos etapuose vaizduojančias grafus bei keičiančias jų briaunų ir viršūnių spalvas.

Mes nesiekėme visiškai optimizuoti algoritmų ir absoliučiai minimizuoti jų žingsnių skaičiaus, nes naudojome Maple programos bazines procedūras nežinodami jų trukmės įverčio. Mums svarbiausia idėja ir jos įgyvendinimas.

1.1 Sutartiniai žymenys

Operacijos algoritmų pseudokoduose:

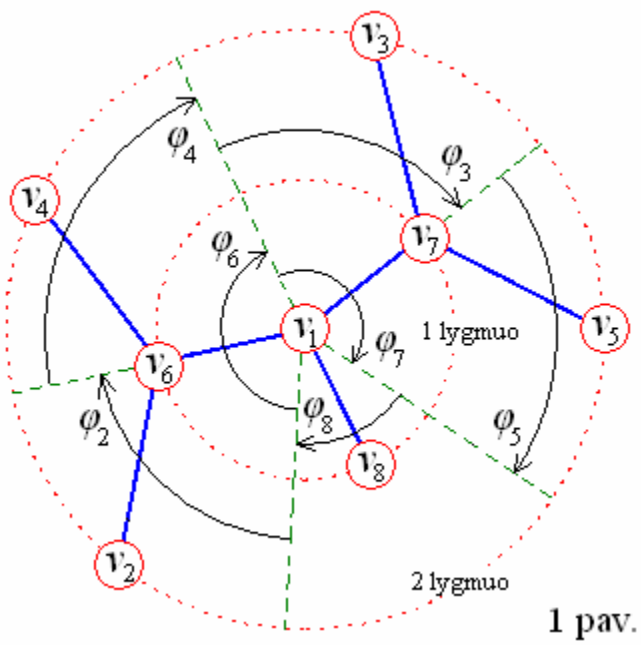
$A = \{a_1, a_2, a_3, ...\}$ – aibė; jos elementai skirtingi ir išrikiuoti didėjimo tvarka (vienmatis masyvas),
$B = [b_1, b_2, b_3, ...]$ – sąrašas, kurio elementai gali kartotis, svarbus jų eiliškumas (vienmatis masyvas),
$\{Ø\}$ – tuščioji aibė,
$[\ ]$ – tuščias sąrašas,
$|A|$ – aibės arba sąrašo elementų skaičius,
$A[k]$ – aibės arba sąrašo k - tasis elementas (A_k),
$enqueue(B, b_n)$ – naujo elemento b_n įrašymas į sąrašo galą,
$enqueue(B_1, B_2, ...)$ – dviejų ar kelių sąrašų apjungimas,
$dequeue(B)$ – pirmojo sąrašo elemento šalinimas,
$reverse(B)$ – sąrašo elementų išdėstymas atvirkščia tvarka,
$B[1]$ – sąrašo (aibės) B pirmasis elementas (sąrašo galva),
$\{[\ ]\}$ – tuščia sąrašų aibė (dvimatis masyvas),
$[\{Ø\}]$ – tuščias aibių sąrašas (dvimatis masyvas),
$\{\{Ø\}\}$ – tuščia aibių aibė (dvimatis masyvas),
$[[\ ]]$ – tuščias sąrašų sąrašas (dvimatis masyvas),
$A[m][n]$ – dvimačio masyvo elementas A_{mn},
$A_1 \cup A_2$ – aibių sąjunga,
$A_1 \cap A_2$ – aibių sankirta,
$A_1 \setminus A_2$ – aibių skirtumas,
$Adj[v](G)$ - viršūnių, gretimų viršūnei v grafe G, aibė (jei grafas žinomas, tai sutrumpintai rašysime $Adj[v]$).

Pastaba: sąrašai arba aibės pseudokoduose žymimi nebūtinai didžiosiomis raidėmis.

2. Medžių vizualizacija plokštumoje
2.1 Radialinis medžių vaizdavimas

Radialinis medis, dar vadinamas „radialiniu žemėlapiu“ – tai metodas, skirtas medžio struktūros vaizdavimui. Ši idėja pradėta plėtoti XX a. pradžioje Amerikoje [11], kai imta įvairių organizacijų struktūrą vaizduoti grafiškai. Vėliau, atsiradus pirmiesiems kompiuteriams, radialinis medžių vaizdavimas tapo programavimo uždaviniu.

Šio uždavinio pagrindinis tikslas – rasti medžio viršūnių koordinates, todėl kiekvienai tokiai viršūnei $v \in V$ reikia sukonstruoti lygmenį $m[v]$ ir kampą $\psi[v]$. Šie parametrai vienareikšmiškai nusako grafo viršūnių padėtis plokštumoje (prisiminkime polines koordinates). Radialinio žemėlapio viduryje vaizduojmas medžio centras (tam tikra grafo viršūnė, kuriai priskiriamas nulinis lygmuo). Šios viršūnės vaikai priklauso pirmajam lygmeniui, anūkai – antrajam ir t. t. Realizuojant medžio viršūnių lygmenų radimo algoritmą, patogu nustatyti kiekvienos viršūnės palikuonių skaičių $n[v]$, kurį vadinsime svoriu. Pagal gautus svorius medžio lygmenis dalinsime į sektorius $\varphi[v]$, kiekvienam sektoriui $\varphi[v]$ priskirsime po vieną viršūnę v (1 pav). Tada galėsime apskaičiuoti kampus $\psi[v]$, $v \in V$. Sutarkime žymėti v - tosios viršūnės tėvą $\pi[v]$ (tai gretima viršūnė, esanti arčiau medžio centro).



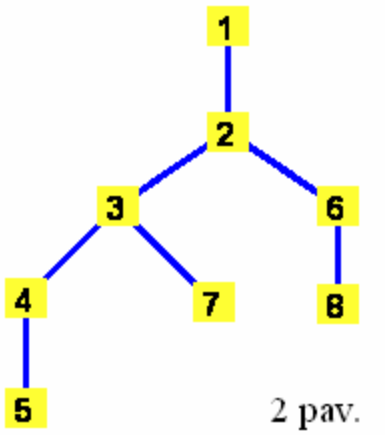
1 pav.

Aptarkime radialinio medžių vaizdavimo algoritmo žingsnius.

1. $G = (V, E)$ – inicializacija;
2. randama centrinė medžio viršūnė s, ($m[s] := 0$, $\varphi[s] := 2\pi$, $n[s] := |V| - 1$);
3. $\forall v \in V \setminus \{s\}$ nustatomas palikuonių skaičius $n[v]$;
4. $\forall v \in V \setminus \{s\}$ apskaičiuojamas lygmuo $m[v]$,
5. $\forall v \in V \setminus \{s\}$ išrikiuojamos pagal du parametrus: lygmenį $m[v]$ ir tėvystės atributą $\pi[v]$;
6. išrikiuotame sąraše $\forall v \in V \setminus \{s\}$ randamas sektorius $\varphi[v]$ bei kampas $\psi[v]$;
7. kiekvienos viršūnės v koordinatės $(x_v; y_v)$ apskaičiuojamos pagal formulę

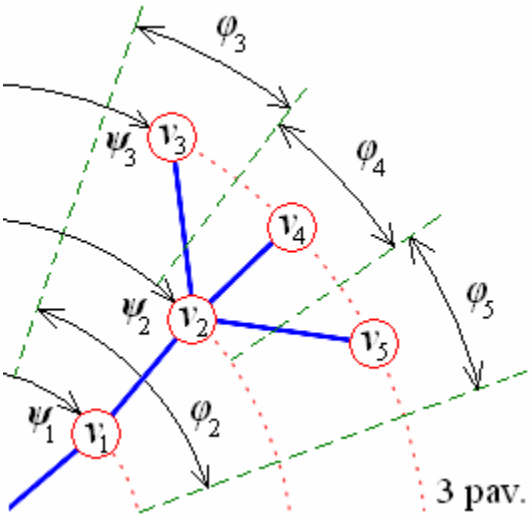
$$\begin{cases} x_v = m[v]\cos\psi[v] \\ y_v = m[v]\sin\psi[v] \end{cases}.$$

Adaptuokime šį algoritmą Maple programai, kurios aplinkoje grafas apibrėžiamas kaip nesutvarkytų viršūnių porų aibė. Pavyzdžiui, 2 pav. pavaizduotas medis inicializuojamas tokia komanda: G:=Graph({{1,2},{2,6},{2,3},{3,4},{3,7},{4,5},{6,8}}). Taip pat patogu atsitiktinai parinkti medį, turintį t viršūnių. Tokiu atveju naudojama procedūra RandomTree(t). Kur kas sudėtingesnis uždavinys rasti centrinę medžio viršūnę. Vienas iš galimų būdų - tai „lapų karpymo“ metodas [9]. Medžio lapu vadinsime viršūnę, incidenčią tik vienai briaunai. Tarkime, kad inicializuotame medyje žinome visus jo lapus, kuriuos vienu metu galime „nukirpti“, t.y. panaikinti grafo briaunų poaibį. Tos pačios iteracijos metu patogu apskaičiuoti kiekvienos nagrinėjamos viršūnės palikuonių skaičių. Po p iteracijų (jų skaičius sutampa su lygmenų kiekiu) medyje beliks viena arba dvi viršūnės. Jei lieka viena viršūnė, tai ji ir yra medžio centras, jei dvi – medžio centru laikysime viršūnę, turinčią daugiau gretimų viršūnių (vaikų). Galimas ir toks atvejis, kada abiejų viršūnių vaikų skaičius sutampa. Tada sutarkime medžio centru sutarkime laikyti mažesnį numerį turinčią viršūnę.



2 pav.

Sekančių algoritmo etapų metu, radus viršūnių lygmenis, vykdomas medžio viršūnių sąrašo rikiavimas pagal du atributus $m[v]$ ir $\pi[v]$. Tada apskaičiuojami kampai $\varphi[v]$ bei $\psi[v]$ ir galiausiai išvedamos viršūnių koordinatės. Tačiau daug patogiau visus šiuos etapus realizuoti iš karto – sudarius kintamą viršūnių sąrašą U_i, $i = 0,..., p-1$. Iš pradžių šiam sąrašui priskiriamos viršūnės, gretimos medžio centrui (jų pirmas lygmuo). Visoms šioms viršūnėms galime apskaičiuoti kampus $\varphi[v]$ ir $\psi[v]$ (pagal turimus svorius $n[v]$) bei koordinates $(x_v; y_v)$. Atlikus šiuos skaičiavimus, sąrašas U_i keičiamas sąrašu U_{i+1}, į jį įrašomos viršūnės, gretimos buvusio sąrašo viršūnėms ir kurių koordinačių dar nežinome. Tokia procedūra tęsiama tol, kol apskaičiuojamos koordinatės kiekvienai medžio viršūnei v.



3 pav.

Išveskime formules kampams $\varphi[v]$, $\psi[v]$ rasti. Trečiame paveikslėlyje pateiktas radialinio medžio fragmentas, kuriame matome, kad $n[v_1] = 4$, $n[v_2] = 3$, $n[v_3] = 0$, $n[v_4] = 0$, $n[v_5] = 0$. Jei kampas $\varphi[v_1]$ yra fiksuotas, tai sutarkime, kad $\varphi[v_2] = \varphi[v_1]$ ir $\varphi[v_3] = \varphi[v_4] = \varphi[v_5] = \dfrac{\varphi[v_2]}{3}$. Bendruoju atveju taikysime rekurenčiąją formulę

$$\varphi[v] = \frac{n[v]+1}{n[\pi[v]]} \cdot \varphi[\pi[v]], \quad v \in V \setminus \{s\}.$$

Tarkime, kad kampas $\psi[v_1]$ taip pat žinomas. Tada

$$\psi[v_2]=\psi[v_1],\ \psi[v_3]=\psi[v_2]-\frac{\varphi[v_2]}{2}+\frac{\varphi[v_3]}{2},\ \psi[v_4]=\psi[v_2]-\frac{\varphi[v_2]}{2}+\varphi[v_3]+\frac{\varphi[v_4]}{2},$$

$$\psi[v_5]=\psi[v_2]-\frac{\varphi[v_2]}{2}+\varphi[v_3]+\varphi[v_4]+\frac{\varphi[v_5]}{2}.$$

Todėl teisinga tokia rekurenčioji formulė

$$\psi[v]=\psi[\pi[v]]-\frac{\varphi[\pi[v]]}{2}+\frac{\varphi[v]}{2}+\sum_{\pi[k]=\pi[v]}\varphi[k],\ v\in V\setminus\{s\}.$$

Kampą $\sum_{\pi[k]=\pi[v]}\varphi[k]$ sutarkime žymėti $\beta\in[0,2\pi)$, kuris algoritmo vykdymo metu kis, t.y. tam tikrais laiko momentais bus priskiriamas nuliui. Kaskart apskaičiavę kampus $\varphi[v]$ ir $\psi[v]$, galėsime rasti medžio viršūnių koordinates:

$$\begin{cases} x_v = i\cdot\cos\psi[v] \\ y_v = i\cdot\sin\psi[v] \end{cases},$$

čia dydis $i=1,...,p$ bus lygus nagrinėjamam medžio lygmeniui.

2.1.1 Algoritmo pseudokodas

Įvesties duomenys: $G=(V,E)$ - medis.

R(*G*):

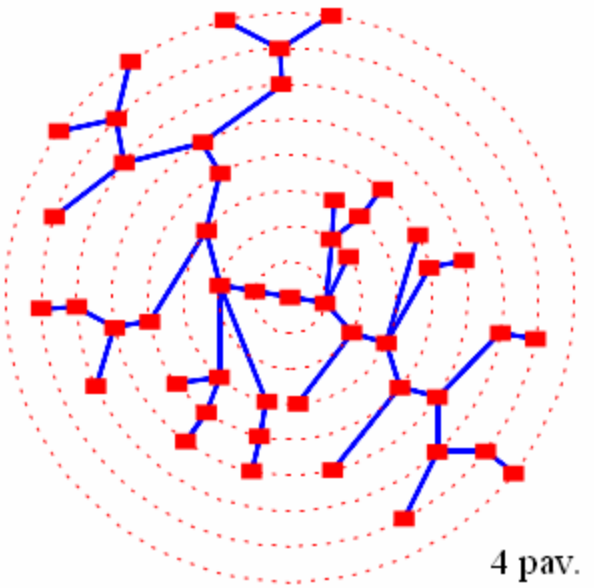
1. $G_1=G\quad(V_1=V,\ E_1=E)$
2. for each $t\in V_1$ do
3. $\quad n[t]=0$
4. $S=\{\varnothing\}$
5. $p=1$
6. while $|V_1|>2$ do
7. $\quad$ for each $t\in V_1$ do
8. $\quad\quad$ if $|Adj[t]|=1$ then
9. $\quad\quad\quad \pi[t]=Adj[t]$
10. $\quad\quad\quad S=S\cup\{t\}$
11. $\quad$ for each $u\in S$ do
12. $\quad\quad n[\pi[u]]=n[\pi[u]]+n[u]+1$
13. $\quad V_1=V_1\setminus S$
14. $\quad p=p+1$
15. if $|V_1|=1$ then
16. $\quad s=V_1$
17. $\quad$ else if $|Adj[V_1[1]]|=\max(|Adj[V_1[1]]|,|Adj[V_1[2]]|)$ then
18. $\quad\quad s=V_1[1]$

19. $\pi[V_1[2]] = s$
20. else
21. $s = V_1[2]$
22. $\pi[V_1[1]] = s$
23. $n[s] = |V| - 1$
24. $\pi[s] = 0$
25. $\varphi[s] = 2\pi$
26. $\psi[s] = 0$
27. $\beta = 0$
28. $x[s] = 0$
29. $y[s] = 0$
30. $U[0] = s$
31. $P = [\ ]$
32. $i = 0$
33. while $i < p$ do
34. for each $u \in U[i]$ do
35. for each $t \in \{Adj[u]\} \setminus \{\pi[u]\}$ do
36. $\varphi[t] = \frac{n[t]+1}{n[\pi[t]]} \cdot \varphi[\pi[t]]$
37. $\psi[t] = \psi[\pi[t]] - \frac{\varphi[\pi[t]]}{2} + \frac{\varphi[t]}{2} + \beta$
38. $\beta = \beta + \varphi[t]$
39. $x[t] = (i+1) \cdot \sin(\psi[t])$
40. $y[t] = (i+1) \cdot \cos(\psi[t])$
41. $P = enqueue(P, \{Adj[u]\} \setminus \{\pi[u]\})$
42. $\beta = 0$
43. $i = i + 1$
44. $U[i] = P$
45. $P = [\ ]$
46. END

Išvesties duomenys: koordinatės $(x[t], y[t])$ kiekvienai medžio viršūnei $t \in V$.

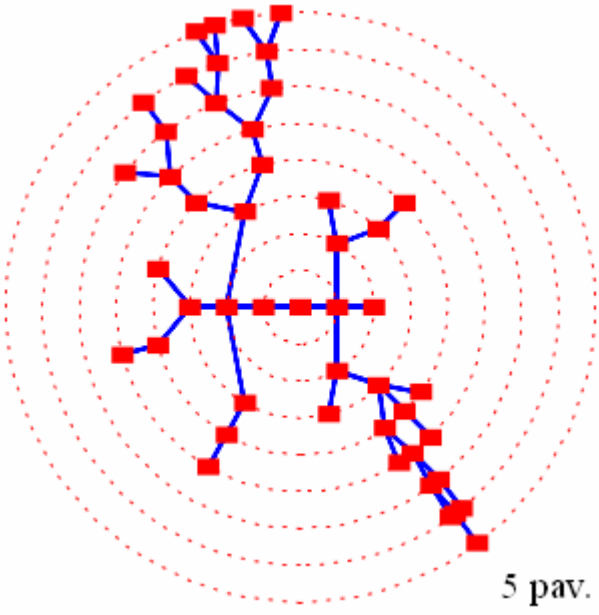
2.2 Apibendrintas radialinis medžių vaizdavimas

Kad ir koks patogus radialinis medžių vaizdavimo būdas, deja realizavus algoritmą ne visada gaunamas pakankamai išvaizdus medis (4 pav.). Pagrindiniai veiksniai, lemiantys tokį viršūnių išsidėstymą plokštumoje – tai per daug ilgas takas bei tendencija viršūnėms retai išsidėstyti kiekviename lygmenyje. Todėl šiame darbe siūlomi keli būdai spręsti iškilusią problemą. Pirmas būdas yra ilgiausio tako medyje radimas, bei jo vaizdavimas pasirinktame lygmenyje. Tačiau tokiu atveju gaunamas visiškai naujas algoritmas, kurį aptarsime vėliau. Kitas būdas – tai papildomų parametrų, nuo kurių priklauso viršūnių padėtys plokštumoje, įvedimas arba, kitaip sakant, radialinio medžių vaizdavimo algoritmo apibendrinimas. Tačiau kokius parametrus galime įvesti?



4 pav.

Prisiminkime, kad sektoriai $\varphi[v]$, $v \in V$ buvo apskaičiuojami pagal kiekvienos viršūnės palikuonių skaičių. O kaip atrodytų medis, jei tuos pačius sektorius $\varphi[v]$ apskaičiuotume pagal viršūnės $v \in V$ vaikų skaičių? Vaizdas būtų taip pat ne itin estetiškai gražus (5 pav). Todėl šiame darbe siūlomas kompromisas – įvesti papildomą parametrą $\lambda \in [0;1]$, kurį tolygiai mažinant nuo 1 iki 0, palaipsniui pereitume nuo 4 iki 5 paveikslėlio, t.y. sektoriai $\varphi[v]$ būtų apskaičiuojami pagal tam tikrą parametrą, kuris tolygiai kistų nuo kiekvienos viršūnės palikuonių iki vaikų skaičiaus. Pavyzdžiui, jei parinktume $\lambda = 0{,}2$, tai mūsų nagrinėjamas medis turėtų atrodyti taip: 6 pav. Sutarkime $\forall v \in V \setminus \{s\}$ (čia s – medžio centras) vaikų skaičiaus kiekį žymėti $vaik[v]$, kuris lygus $|Adj[v]|-1$.



5 pav.

Centrinei viršūnei s galioja lygybė $vaik[s]=|Adj[s]|$. Realizuojant algoritmą patogu atributus $vaik[v]$ ir $n[v]$ apskaičiuoti toje pačioje procedūroje, kuri bus aprašyta pseudokode.

Dabar susiekime medžio viršūnių koordinates su parametru $\lambda \in [0;1]$. Kampus $\varphi[v]$, $\psi[v]$ apskaičiuosime remdamiesi šiomis rekurenčiosiomis formulėmis:

$$\varphi[v] = \frac{n[v]\cdot\lambda + 1}{vaik[\pi[v]] + (n[\pi[v]] - vaik[\pi[v]])\cdot\lambda} \cdot \varphi[\pi[v]],\ \lambda \in [0;1],\ v \in V \setminus \{s\},$$

$$\psi[v] = \psi[\pi[v]] - \frac{\varphi[\pi[v]]}{2} + \frac{\varphi[v]}{2} + \sum_{\pi[k]=\pi[v]} \varphi[k],\ v \in V \setminus \{s\}.$$

Pastaroji lygybė sutampa su radialinio medžio viršūnių posūkio kampų $\psi[v]$, $v \in V$ apskaičiavimo formule.

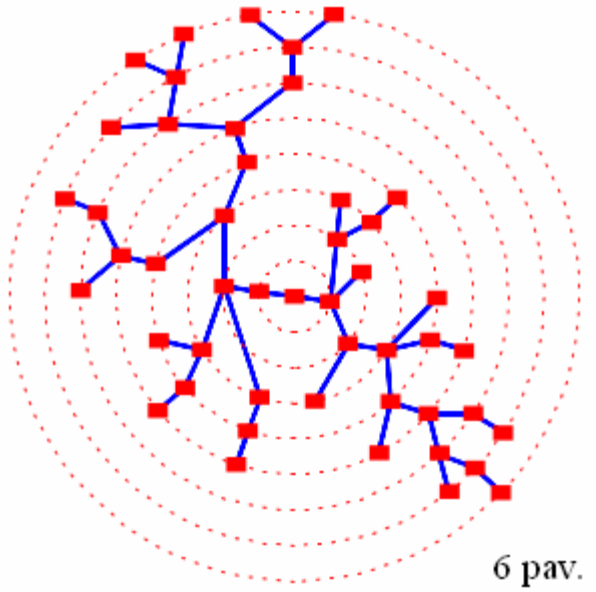
Galiausiai išvesdami medžio viršūnių koordinates $(x_v; y_v)$, naudosime dar tris parametrus: μ - elipsės ištemptumas, γ - kampas, kuriuo pasukama elipsių sistema ir υ - elipsių retėjimo parametras. Tada koordinates $(x_v; y_v)$ apskaičiuosime pagal formulę

$$\begin{cases} x_v = \mu \cdot i^{\upsilon} \cdot \sin\psi[v]\cos\gamma - i^{\upsilon} \cdot \cos\psi[v]\sin\gamma \\ y_v = \mu \cdot i^{\upsilon} \cdot \sin\psi[v]\sin\gamma + i^{\upsilon} \cdot \cos\psi[v]\cos\gamma \end{cases}.$$

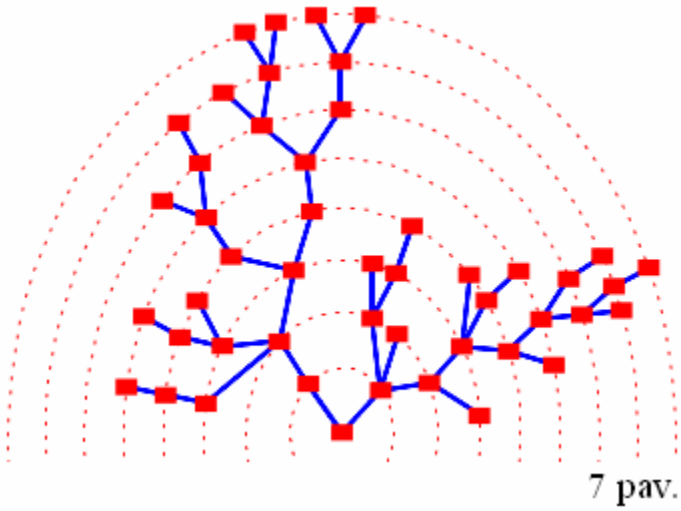
Dydis $i = 1,..., p$ sutaps su nagrinėjamamu medžio lygmeniu (kaip ir radialiniame meždių vizualizacijos algoritme).

Taip pat galime įvesti skritulio išpjovos kampą $\tau \in (0;2\pi]$, nurodantį skritulio dalį, kurioje bus vaizduojamas radialinis medis, tada algoritmo vykdymo eigoje priskirsime $\varphi[s] = \tau$, kur s – medžio centras.

Taigi medžio išvaizdumo problema dalinai išspręsta! Jeigu parinksime tokius parametrus: $\gamma = 0$, $\mu = \frac{9}{10}$, $\lambda = \frac{1}{5}$, $\tau = \pi$, $\upsilon = \frac{9}{10}$, tai realizavus apibendrinto radialinio medžio vaizdavimo algoritmą, 4, 5 ir 6 paveikslėliuose esantis medis atitiks 7 pav. medį.



6 pav.



7 pav.

2.2.1 Algoritmo pseudokodas

Įvesties duomenys: $G = (V, E)$ - medis,
$\lambda \in [0;1]$ - medžio viršūnių sektorių dydžių parametras,
μ - elipsės ištemptumas,
γ - kampas, kuriuo pasukama elipsių sistema,
υ - elipsių retėjimo parametras,
$\tau \in (0;2\pi]$ - skritulio išpjovos kampas.

AR(*G, λ, μ, γ, υ, τ*):

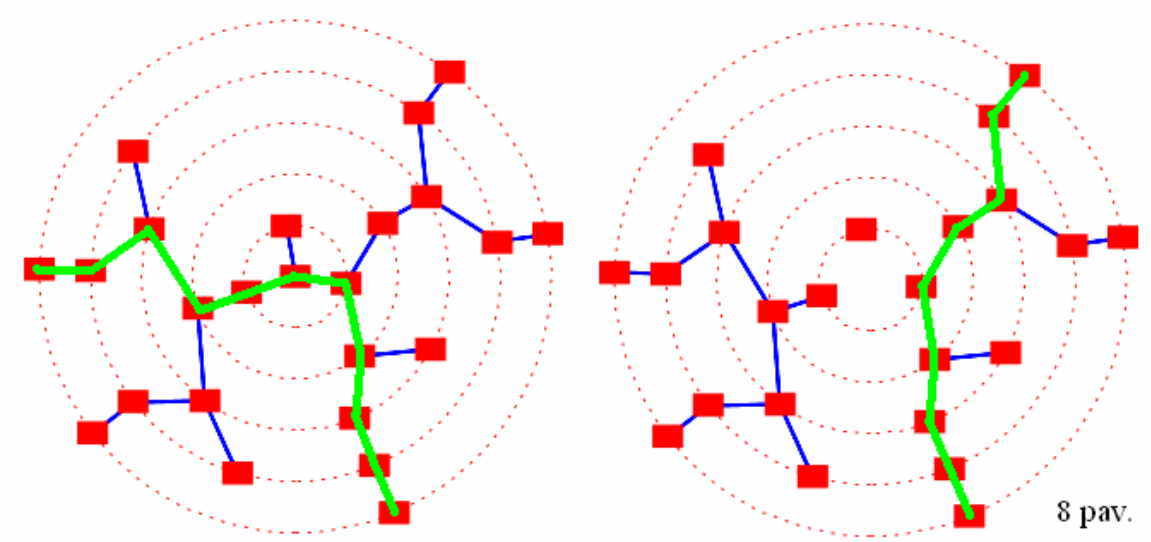
1. $G_1 = G \quad (V_1 = V,\ E_1 = E)$
2. for each $t \in V_1$ do
3. $\quad n[t] = 0$
4. $\quad vaik[t] = 0$
5. $S = \{\emptyset\}$
6. $p = 1$
7. while $|V_1| > 2$ do
8. $\quad$ for each $t \in V_1$ do
9. $\quad\quad$ if $|Adj[t]| = 1$ then
10. $\quad\quad\quad \pi[t] = Adj[t]$
11. $\quad\quad\quad S = S \cup \{t\}$
12. $\quad$ for each $u \in S$ do
13. $\quad\quad n[\pi[u]] = n[\pi[u]] + n[u] + 1$
14. $\quad\quad vaik[\pi[u]] = vaik[\pi[u]] + 1$
15. $\quad V_1 = V_1 \setminus S$
16. $\quad p = p + 1$
17. if $|V_1| = 1$ then
18. $\quad s = V_1$
19. $\quad$ else if $|Adj[V_1[1]]| = \max(|Adj[V_1[1]]|, |Adj[V_1[2]]|)$ then
20. $\quad\quad s = V_1[1]$
21. $\quad\quad \pi[V_1[2]] = s$
22. $\quad$ else
23. $\quad\quad s = V_1[2]$
24. $\quad\quad \pi[V_1[1]] = s$
25. $n[s] = |V| - 1$
26. $\pi[s] = 0$
27. $\varphi[s] = \tau$
28. $\psi[s] = 0$

29. $\beta = 0$
30. $x[s] = 0$
31. $y[s] = 0$
32. $U[0] = s$
33. $P = [\ ]$
34. $i = 0$
35. while $i < p$ do
36. for each $u \in U[i]$ do
37. for each $t \in \{Adj[u]\} \setminus \{\pi[u]\}$ do
38. $$\varphi[t] = \frac{n[t] \cdot \lambda + 1}{vaik[\pi[t]] + (n[\pi[t]] - vaik[\pi[t]]) \cdot \lambda} \cdot \varphi[\pi[t]]$$
39. $$\psi[t] = \psi[\pi[t]] - \frac{\varphi[\pi[t]]}{2} + \frac{\varphi[t]}{2} + \beta$$
40. $$\beta = \beta + \varphi[t]$$
41. $$x[t] = \mu \cdot (i+1)^{\upsilon} \cdot \sin\psi[v]\cos\gamma - (i+1)^{\upsilon} \cdot \cos\psi[v]\sin\gamma$$
42. $$y[t] = \mu \cdot (i+1)^{\upsilon} \cdot \sin\psi[v]\sin\gamma + (i+1)^{\upsilon} \cdot \cos\psi[v]\cos\gamma$$
43. $P = enqueue(P, \{Adj[u]\} \setminus \{\pi[u]\})$
44. $\beta = 0$
45. $i = i + 1$
46. $U[i] = P$
47. $P = [\ ]$
48. END

Išvesties duomenys: koordinatės $(x[t], y[t])$ kiekvienai medžio viršūnei $t \in V$.

2.3 Ilgiausi takai medžiuose

Tolesniam medžių vizualizacijos išvaizdumo problemos sprendimui prireiks pagalbinio algoritmo, skirto ilgiausių takų paieškai medyje. Ir čia kyla natūralus klausimas – kaip gi ieškosime ilgiausių takų? Jeigu atsitiktinai parinktume kurią nors medžio viršūnę, ji juk nebūtinai priklausytų vienam iš ilgiausių takų. Ar apskritai egzistuoja tokia medžio viršūnė, kuri bet kokiu atveju būtų ilgiausiame take? Atsakymas į šį klausimą – teigiamas, tai medžio centras. Šio teiginio įrodymas labai paprastas. Prisiminkime radialinį medžių vaizdavimą. Jei tokiu būdu pavaizduotas medis turi p lygmenų, tai į ilgiausią taką įeis mažiausiai $2p$ viršūnių. Tačiau pašalinę centrinę medžio viršūnę ilgiausią taką galėsime sudaryti iš daugiausiai iš $2p-1$ viršūnių (8 pav.). Išvada: medžio centras priklauso ilgiausiam takui.



8 pav.

Taigi ilgiausių takų paieškos algoritmo idėja paremta „lapų karpymo“ metodu [9]. Pradinį medį $G=(V,E)$ pažymėkime $G_1=(V_1,E_1)$. Kiekvienai viršūnei $v\in V_1$, kuri yra lapas, priskirkime dvimačio masyvo tipo atributą $q[v]=\{[v]\}$, kitu atveju (jei višūnė nėra lapas), $q[v]=\{\emptyset\}$. Medžio G_1 viršūnių lapų sąrašą žymėsime S, $S\in V_1$. Todėl $\forall v\in V_1$ tenkina sąlygą $|Adj[v]|=1$. Kol teisinga sąlyga $|V_1|>2$, vykdykime šiuos žingsnius:

1. $\forall v\in S$, į gretimos viršūnės atributą $q[Adj[v]]$ įrašykime visus ilgiausius takus: į viršūnių sąrašą $q[v]$ įterpiama viršūnė $Adj[v]$. Visų tokių ilgiausių takų atributas lygus $q[Adj[v]]$,
2. $G_1=G_1\setminus S$ ir S - naujo medžio G_1 lapų sąrašas,
3. jei $\forall v\in S$ $|Adj[v]|\neq 1$, tai $q[v]=\{\emptyset\}$.

Kai sąlyga $|V_1|>2$ nebėra teisinga, tai pagal likutinio medžio G_1 gretimų viršūnių atributus $q[v]$, ($\forall v$ gretima $u\in G_1$) sudaromi ilgiausi takai. Kad algoritmas būtų aiškesnis, pateiksime ilgiausių takų paieškos pseudokodą bei du pavyzdžius.

2.3.1 Algoritmo pseudokodas

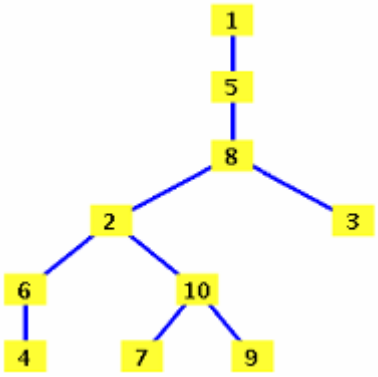
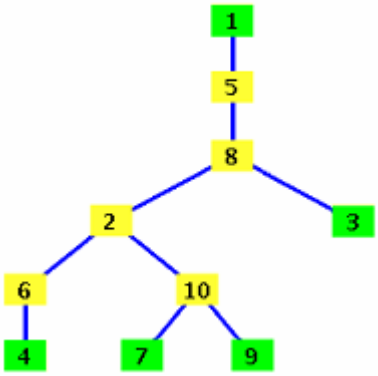
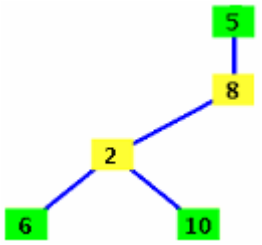
Įvesties duomenys: $G = (V, E)$ - medis.

IT(*G*):

1. $G_1 = G \ \ (V_1 = V, \ E_1 = E)$
2. $S = [\ ]$
3. for each $v \in V_1$ do
4. $\quad q[v] = \{[v]\}$
5. while $|V_1| > 2$ do
6. $\quad$ for each $v \in V_1$ do
7. $\quad\quad$ if $|Adj[v]| = 1$ then
8. $\quad\quad\quad S = enqueue(S, v)$ - sąrašas
9. $\quad$ for each $v \in V_1 \setminus S$ do
10. $\quad\quad q[v] = \{\emptyset\}$
11. $\quad$ for each $v \in S$ do
12. $\quad\quad$ for each $u \in q[v]$ do
13. $\quad\quad\quad q[Adj[v](G_1)] = q[Adj[v](G_1)] \cup \{enqueue(u, Adj[v](G_1))\}$
14. $\quad G_1 = G_1 \setminus S \ \ (V_1 = V_1 \setminus S)$
15. $\quad S_1 = S$
16. $\quad S = [\ ]$
17. $i = 1$
18. if $|V_1| = 1$ then
19. $\quad$ for m from 1 to $|S_1| - 1$ do
20. $\quad\quad$ for n from 1 to $|q[S_1[1]]|$ do
21. $\quad\quad\quad$ for k from 1 to $|q[S_1[m+1]]|$ do
22. $\quad\quad\quad\quad T[i] = enqueue(q[S_1[1][n]], V_1[1], reverse(q[S_1[m+1][k]]))$
23. $\quad\quad\quad\quad i = i + 1$
24. $\quad\quad q[S_1[1]] = \{q[S_1[1]] \cup q[S_1[m+1]]\}$ - aibė
25. $\quad$ else for m from 1 to $|p[V_1[1]]|$ do
26. $\quad\quad$ for n from 1 to $|p[V_1[2]]|$ do
27. $\quad\quad\quad T[i] = enqueue(q[V_1[1][m]], reverse(q[V_1[2][n]]))$
28. $\quad\quad\quad i = i + 1$
29. END

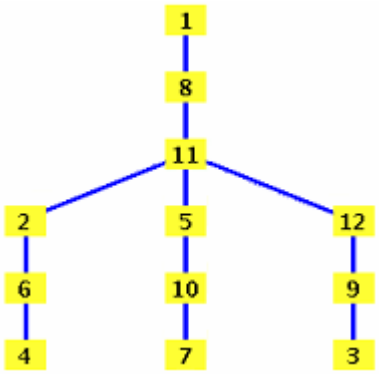
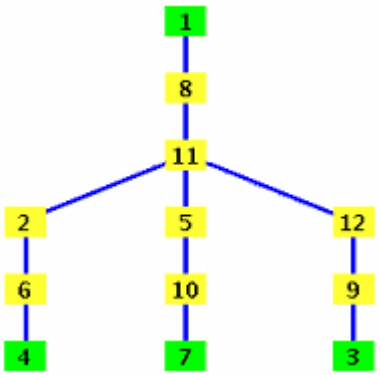
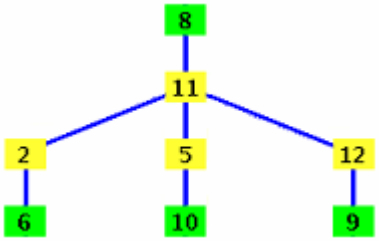
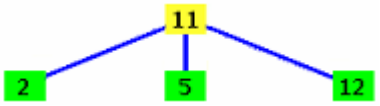
Išvesties duomenys: ilgiausi takai $T[1], \ T[2], \ldots$ medyje G.

2.3.2 pavyzdys. (Kai likutiniam medžiui G_1 teisinga sąlyga $|V_1| = 2$)

	$G = Graph\left(\begin{Bmatrix}\{1,5\},\{2,6\},\{2,8\},\{2,10\},\{3,8\},\\ \{4,6\},\{5,8\},\{7,10\},\{9,10\}\end{Bmatrix}\right)$, $G_1 = G$, $S = [\ ]$, $q[i] = \{[i]\}, i = 1,...,10.$
	$S = [1,3,4,7,9]$, $q[2] = q[5] = q[6] = q[8] = q[10] = \{\varnothing\}$, $q[5] = \{[1,5]\},\ q[8] = \{[3,8]\},\ q[6] = \{[4,6]\},\ q[10] = \{[7,10]\}$, $q[10] = \{[7,10],[9,10]\}$, $G_1 = G_1 / S = Graph(\{\{2,6\},\{2,8\},\{2,10\},\{5,8\}\})$, $S_1 = [1,3,4,7,9]$, $S = [\ ]$.
	$S = [5,6,10]$, $q[2] = q[8] = \{\varnothing\}$, $q[8] = \{[1,5,8]\},\ q[2] = \{[4,6,2]\},\ q[2] = \{[4,6,2],[7,10,2]\}$, $q[2] = \{[4,6,2],[7,10,2],[9,10,2]\}$, $G_1 = G_1 / S = Graph(\{\{2,8\}\}),\ S_1 = [5,6,10],\ S = [\ ]$.
	$T[1] = enqueue(q[V_1[1]][1], reverse(q[V_1[2]][1])) = [4,6,2,8,5,1]$, $T[2] = enqueue(q[V_1[1]][2], reverse(q[V_1[2]][1])) = [7,10,2,8,5,1]$, $T[3] = enqueue(q[V_1[1]][3], reverse(q[V_1[2]][1])) = [9,10,2,8,5,1]$.

Atsakymas: 3 ilgiausi takai $T[1] = [4,6,2,8,5,1]$, $T[2] = [7,10,2,8,5,1]$, $T[3] = [9,10,2,8,5,1]$.

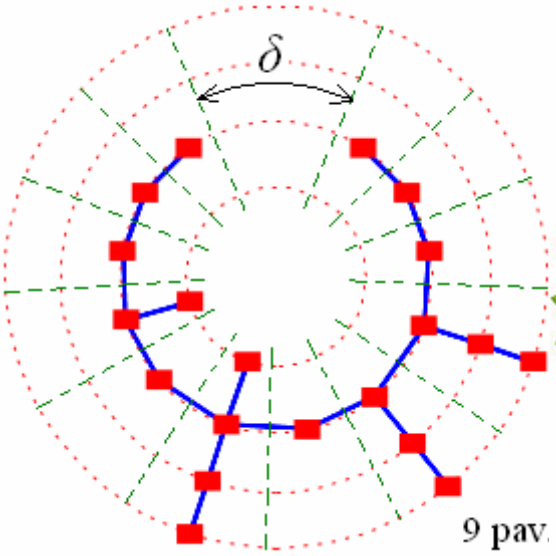
2.3.3 pavyzdys. (Jei likutiniam medžiui G_1 galioja lygybė $|V_1|=1$)

	$G = Graph\left(\begin{Bmatrix}\{1,8\},\{2,6\},\{2,11\},\{3,9\},\{4,6\},\{5,10\},\\ \{5,11\},\{7,10\},\{8,11\},\{9,12\},\{11,12\}\end{Bmatrix}\right)$, $G_1 = G$, $S = [\ ]$, $q[i] = \{[i]\}, i = 1,...,12.$
	$S = [1,3,4,7]$, $q[2] = q[5] = q[6] = q[8] = q[9] = q[10] = q[11] = q[12] = \{\emptyset\}$, $q[8] = \{[1,8]\},\ q[9] = \{[3,9]\},\ q[6] = \{[4,6]\},\ q[10] = \{[7,10]\}$, $G_1 = G_1 / S = Graph\left(\begin{Bmatrix}\{2,6\},\{2,11\},\{5,10\},\{5,11\},\\ \{8,11\},\{9,12\},\{11,12\}\end{Bmatrix}\right)$, $S_1 = [1,3,4,7]$, $S = [\ ]$.
	$S = [6,8,9,10]$, $q[2] = q[5] = q[11] = q[12] = \{\emptyset\}$, $q[2] = \{[4,6,2]\},\ q[11] = \{[1,8,11]\},\ q[12] = \{[3,9,12]\}$, $q[5] = \{[7,10,5]\}, G_1 = G_1 / S = Graph(\{\{2,11\},\{5,11\},\{11,12\}\})$, $S_1 = [6,8,9,10],\ S = [\ ]$.
	$S = [2,5,12],\ q[11] = \{\emptyset\},\ q[11] = \{[4,6,2,11]\}$, $q[11] = \{[4,6,2,11],[7,10,5,11]\}$, $q[11] = \{[3,9,12,11],[4,6,2,11],[7,10,5,11]\}$, $G_1 = G_1 / S = Graph(\{[11]\}),\ S_1 = [2,5,12],\ S = [\ ]$.
	$T[1] = enqueue(q[S_1[1]],11, reverse(q[S_1[2]])) = [4,6,2,11,5,10,7]$, $q[2] = \{[4,6,2],[7,10,5]\}$, $T[2] = enqueue(q[S_1[1]][1],11, reverse(q[S_1[3]])) = [4,6,2,11,12,9,3]$, $T[3] = enqueue(q[S_1[1]][2],11, reverse(q[S_1[3]])) = [7,10,5,11,12,9,3]$.

Atsakymas: 3 ilgiausi takai $T[1] = [4,6,2,11,5,10,7]$, $T[2] = [4,6,2,11,12,9,3]$, $T[3] = [7,10,5,11,12,9,3]$.

2.4 Laurų vainiko algoritmas

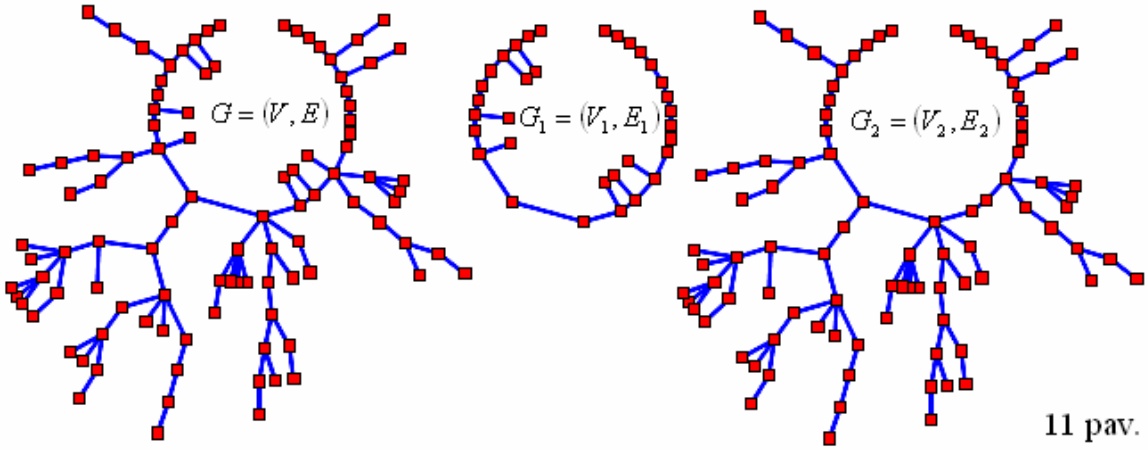
Šiame skyriuje toliau nagrinėsime medžių vizualizacijos estetiškumo problemą, kurią spręsime taikydami ilgiausių takų paieškos algoritmą IT(G). Akivaizdu, kad egzistuoja bent vienas ilgiausias takas $T[tk]=[v_{1k}, v_{2k}, ...]$, $tk = 1,2,...$ medyje $G=(V,E)$. Sutarkime šio grafo viršūnes išdėstyti lygmenimis, kaip ir radialiniame medžių vaizdavime (9 pav.). Kadangi tokia vizualizacija primena laurų vainiką (10 pav.), todėl sutarkime šį algoritmą taip ir pavadinti – laurų vainiko algoritmas. Mūsų užduotis –



9 pav.


10 pav.

$\forall v \in V$ apskaičiuoti koordinates $(x_v; y_v)$, vadinasi, kaip ir radialiniame medžių vaizdavime $\forall v \in V$ pirma reikia rasti atributus $m[v]$ ir $\psi[v]$. Tegu ilgiausio tako lygmuo m ir skritulio išpjovos (jame vaizduojamas grafas) kampas lygus $2\pi - \delta$. Likusių viršūnių (nepriklausančių ilgiausiam takui) lygmenims apskaičiuoti įvesime papildomą parametrą k, nurodantį palikuonių eilę laurų vainiko viduje (pavyzdžiui, 9 pav. $k=1$). Aišku, kad nagrinėjamą medį $G=(V,E)$ pagal k patogu padalinti į du medžius $G_1=(V_1,E_1)$ ir $G_2=(V_2,E_2)$, kurių viršūnėms teisinga sąlyga $(V_1 \setminus T[tk]) \cap (V_2 \setminus T[tk]) = \{\emptyset\}$ (11 pav.).



11 pav.

Ilgiausio tako $T[tk]$ viršūnių sektorių kampų dydžius apskaičiuosime pagal palikuonių skaičių $n[t]$, $t \in V \setminus T[tk]$. Tada kiekvienam medžiui $G_1 = (V_1, E_1)$ ir $G_2 = (V_2, E_2)$ taikysime rekurenčiąsias formules (analogiškas radialiniam medžių vaizdavimui) sektoriams $\varphi[v]$ bei kampams $\psi[v]$ rasti. Pateiksime laurų vainiko algoritmo idėją bei pseudokodą.

1. $G = (V, E)$ – inicializacija;
2. randami ilgiausi takai $T[tk]$, $tk = 1,2,...$ medyje $G = (V, E)$, tegu $tk = 1$;
3. $\forall v \in T[tk]$ nustatomas palikuonių skaičius $n[v]$, pagal kurį randame $\varphi[v]$;
4. $\forall v \in V \setminus T[tk]$ apskaičiuojamas tėvystės atributas $\pi[v]$ bei trumpiausias atstumas $mrange[v]$ iki pasirinkto ilgiausio tako $T[tk]$, kurio lymguo m;
5. pagal atributus $mrange[v]$ bei parametrą m, sugeneruojami du nauji medžiai $G_1 = (V_1, E_1)$ ir $G_2 = (V_2, E_2)$ (11 pav.);
6. $\forall v \in V \setminus T[tk]$ apskaičiuojame tokius atributus: $n[v]$ - palikuonių skaičius, $vaik[v]$ - vaikų skaičius, $n_1[v]$ - palikuonių skaičius grafe $G_1 = (V_1, E_1)$, $vaik_1[v]$ - vaikų skaičius grafe $G_1 = (V_1, E_1)$;
7. analogiškai radialiniam medžių vaizdavimo algoritmui, $\forall v \in G_1$ išrikiuojamos pagal du parametrus: lygmenį $m[v]$ ir tėvystės atributą $\pi[v]$;
8. išrikiuotame sąraše $\forall v \in G_1$, taikant rekurenčiąsias formules, randamas sektorius $\varphi[v]$, kampas $\psi[v]$ ir apskaičiuojamos koordinatės $(x_v; y_v)$;
9. $\forall v \in G_2$ išrikiuojame pagal $m[v]$ ir $\pi[v]$;
10. išrikiuotame sąraše $\forall v \in G_2$, taikydami rekurenčiąsias formules, apskaičiuojame $\varphi[v]$, $\psi[v]$, $(x_v; y_v)$.

Susiejus šią idėją su apibendrintu radialiniu medžių vaizdavimu, gaunamas visiškai naujas algoritmas, priklausantis nuo 10 parametrų. Galbūt kai kuriuos iš jų būtų patogiau fiksuoti, bet tam reikia gilesnės algoritmo analizės bei matematinių išvadų. Todėl pateiksime apibendrinto algoritmo pseudokodą.

2.4.1 Algoritmo pseudokodas

Įvesties duomenys:
$G = (V, E)$ - medis,
tk - ilgiausio tako numeris,
m - elipsės, kurioje vaizduojamas ilgiausias takas, lygmuo,
υ - elipsių retėjimo koeficientas,
k - maksimali palikuonių, vaizduojamų laurų vainiko viduje, eilė,
δ - skritulio išpjovos kampas,
α - kampas, kuriuo pasukama elipsių sistema,
μ - elipsių sistemos ištemptumo parametras,
λ_1 - ilgiausio tako palikuonių pasiskirstymo svoris, priklausantis intervalui $[0,1]$,
λ_2 - ilgiausio tako viršūnių pasiskirstymo svoris, priklausantis intervalui $[0,1]$.

LV(*G, tk, m, υ, k, δ, α, μ, λ1, λ2*):

1. $IT(G)$ - vykdoma ilgiausių takų paieška medyje G
2. $G_1 = G \ \ (V_1 = V, \ E_1 = E)$
3. for each $t \in V_1$ do
4. $\quad n[t] = 0$
5. $\quad mrange[t] = 0$
6. $\quad vaik[t] = 0$
7. $\quad n_1[t] = 0$
8. $\quad vaik_1[t] = 0$
9. $S = [\ ]$
10. $p = 1$
11. while $|V_1| > T[tk]$ do
12. $\quad$ for each $v \in V_1 \setminus T[tk]$ do
13. $\quad\quad$ if $|Adj[v]| = 1$ then
14. $\quad\quad\quad \pi[v] = Adj[v]$
15. $\quad\quad\quad S = [S \cup \{v\}]$
16. $\quad$ for each $u \in S$ do
17. $\quad\quad n[\pi[u]] = n[\pi[u]] + n[u] + 1$
18. $\quad\quad n_1[\pi[u]] = n[\pi[u]]$
19. $\quad\quad vaik[\pi[u]] = vaik[\pi[u]] + 1$
20. $\quad\quad vaik_1[\pi[u]] = vaik[\pi[u]]$
21. $\quad\quad mrange[\pi[u]] = mrange[u] + 1$
22. $\quad G_1 = G_1 \setminus \{S\}$
23. $\quad S = [\ ]$
24. $\quad p = p + 1$

25. for each $t \in T[tk]$ do
26. $vaik_1[t] = 0$
27. $G_1 = G \ (V_1 = V, \ E_1 = E)$
28. $G_2 = G_1 \ (V_2 = V_1, \ E_2 = E_1)$
29. $\theta = 0$
30. for each $t \in T[tk]$ do
31. $\varphi[t] = \dfrac{(n[t] \cdot \lambda_2 + 1)(2\pi - \delta)}{|T[tk]| + (|V| - |T[tk]|) \cdot \lambda_2}$
32. $\psi[t] = \dfrac{\delta + \varphi[t]}{2} + \theta$
33. $x[t] = \mu \cdot m^{\upsilon} \cdot \sin(\psi[t]) \cdot \cos(\alpha) - m^{\upsilon} \cdot \cos(\psi[t]) \cdot \sin(\alpha)$
34. $y[t] = \mu \cdot m^{\upsilon} \cdot \sin(\psi[t]) \cdot \sin(\alpha) + m^{\upsilon} \cdot \cos(\psi[t]) \cdot \cos(\alpha)$
35. for each $v \in Adj[t] \setminus T[tk]$ do
36. if $mrange[v] < k$ then
37. $V_1 = V_1 \setminus \{v\}$
38. $vaik_1[\pi[v]] = vaik_1[\pi[v]] + 1$
39. else
40. $V_2 = V_2 \setminus \{v\}$
41. $n_1[\pi[v]] = n_1[\pi[v]] - n[v] - 1$
42. $\theta = \theta + \varphi[t]$
43. $\theta = 0$
44. $U_0 = T[tk]$
45. $i = 0$
46. while $i < k$ do
47. for each $j \in U_i$ do
48. for each $t \in Adj[j](G_2) \setminus \pi[j]$ do
49. $\varphi[t] = \dfrac{n_1[t] \cdot \lambda_1 + 1}{vaik_1[\pi[t]] + (n_1[\pi[t]] - vaik_1[\pi[t]]) \cdot \lambda_1} \cdot \varphi[\pi[t]]$
50. $\psi[t] = \psi[\pi[t]] - \dfrac{\varphi[\pi[t]] - \varphi[t]}{2} + \theta$
51. $\theta = \theta + \varphi[t]$
52. $x[t] = \mu \cdot (m - i - 1)^{\upsilon} \cdot \sin(\psi[t]) \cdot \cos(\alpha) -$
$- (m - i - 1)^{\upsilon} \cdot \cos(\psi[t]) \cdot \sin(\alpha)$
53. $y[t] = \mu \cdot (m - i - 1)^{\upsilon} \cdot \sin(\psi[t]) \cdot \sin(\alpha) +$
$+ (m - i - 1)^{\upsilon} \cdot \cos(\psi[t]) \cdot \cos(\alpha)$
54. $P = [P \cup (Adj[j](G_2) \setminus \pi[j])]$
55. $\theta = 0$
56. $i = i + 1$

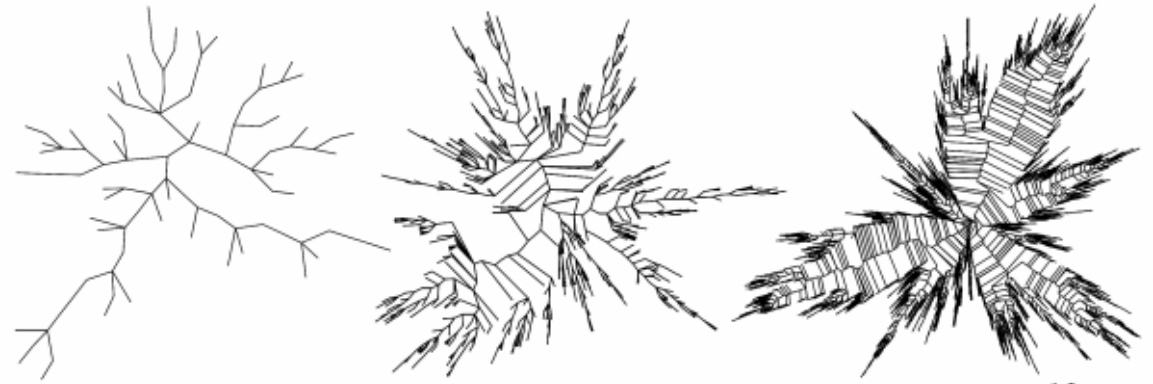
57. $U_i = P$
58. $P = [\ ]$
59. $U_0 = T[tk]$
60. $P = [\ ]$
61. $i = 0$
62. for each $t \in T[tk]$ do
63. $n_1[t] = n[t] - n_1[t]$
$vaik_1[t] = vaik[t] - n_1[t]$
64. while $i < p$ do
65. for each $j \in U_i$ do
66. for each $t \in Adj[j](G_1) \setminus \pi[j]$ do
67. $$\varphi[t] = \frac{n_1[t] \cdot \lambda_1 + 1}{vaik_1[\pi[t]] + (n_1[\pi[t]] - vaik_1[\pi[t]]) \cdot \lambda_1} \cdot \varphi[\pi[t]]$$
68. $$\psi[t] = \psi[\pi[t]] - \frac{\varphi[\pi[t]] - \varphi[t]}{2} + \theta$$
69. $\theta = \theta + \varphi[t]$
70. $$x[t] = \mu \cdot (m + i + 1)^{\upsilon} \cdot \sin(\psi[t]) \cdot \cos(\alpha) -$$
$$-(m + i + 1)^{\upsilon} \cdot \cos(\psi[t]) \cdot \sin(\alpha)$$
71. $$y[t] = \mu \cdot (m + i + 1)^{\upsilon} \cdot \sin(\psi[t]) \cdot \sin(\alpha) +$$
$$+(m + i + 1)^{\upsilon} \cdot \cos(\psi[t]) \cdot \cos(\alpha)$$
72. $P = [P \cup (Adj[j](G_1) \setminus \pi[j])]$
73. $\theta = 0$
74. $i = i + 1$
75. $U_i = P$
76. $P = [\ ]$
77. END

Išvesties duomenys: koordinatės $(x[t], y[t])$ kiekvienai medžio viršūnei $t \in V$.

3. **Medžių vizualizacija trimatėje erdvėje**

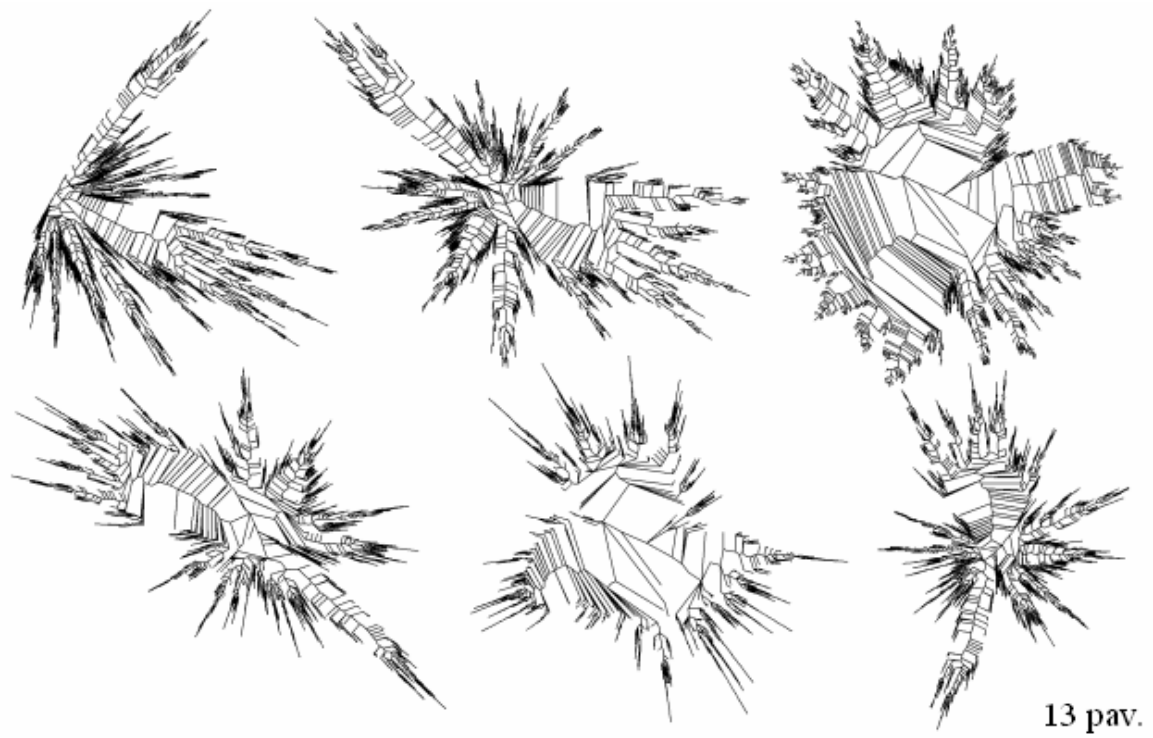
3.1 Radialinis medžių vaizdavimas

Toliau nagrinėsime bei spręsime medžių estetinio pateikimo problemą. Atkreipkime dėmesį į 12 paveikslėlyje radialiniu būdu plokštumoje pavaizduotus tris medžius, turinčius atitinkamai 100, 1000 ir 10000 viršūnių. Atlikti eksperimentai rodo, kad didinant grafo eilę, vizualiai pateikto medžio pakraščiuose viršūnės sutankėja.



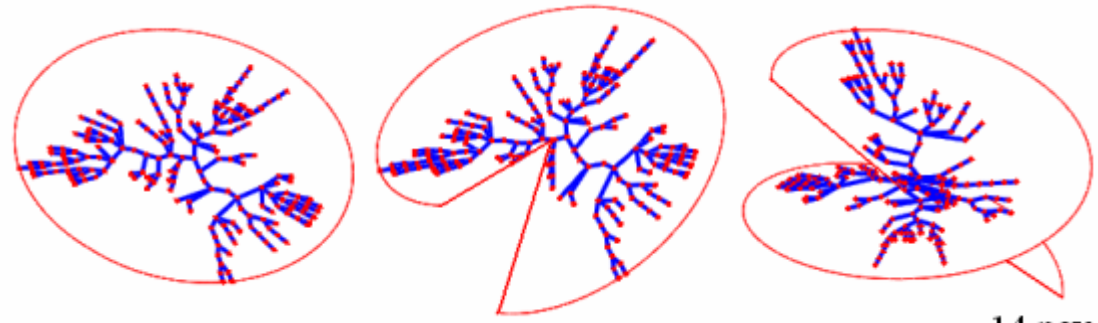
12 pav.

Deja ne visada papildomi parametrai gelbsti išvaizdaus medžių pateitimo klausimu. Pavyzdžiui, atsitiktinai parinkto medžio, kurio eilė lygi 5000, viršūnių sutankėjimas neišvengiamas kad ir kaip keistume parametrus (13 pav.).



13 pav.

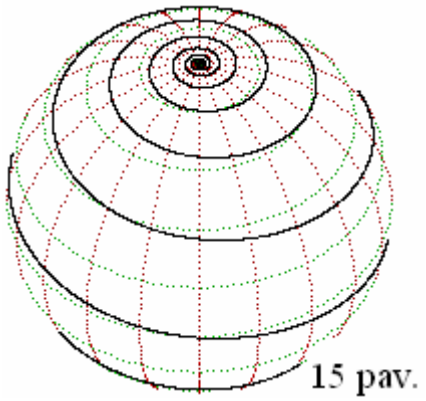
Kaip spręsti iškilusią problemą? Aišku, kad didžiausią įtaką viršūnių sutankėjimui daro skritulio, kuriame vaizduojamas medis, išpjovos kampas $\tau \in (0;2\pi]$. Didinant τ, medžio viršūnių sektoriai platėja, taigi lapams skiriama daugiau vietos plokštumoje.



14 pav.

Vaizduodami grafą erdvėje, galime šį kampą didinti kiek norime rutulio viduje gaudami spiralinę plokštumą, ant kurios užtemptas medis (14 pav.). Taigi viskas ko reikia radialiniam medžių vaizdavimui erdvėje – papildomos koordinatės $z_v, v \in V$ ir spiralės, einančios sferos (bendru atveju elipsoido) paviršiumi (15 pav.), parametrinės lygties [12]:

$$\begin{cases} x = \dfrac{r\cos\omega}{\cosh(\cot(c)(\omega-\omega_0))} \\ y = \dfrac{r\sin\omega}{\cosh(\cot(c)(\omega-\omega_0))} \\ z = r\tanh(\cot(c)(\omega-\omega_0)) \end{cases}$$



15 pav.

Lygčių sistemoje konstantos r, c ir ω_0 nurodo atitinkamai rutulio spindulio ilgį, spiralės sutankėjimą (praretėjimą) ir jos padėtį sferoje (ω_0 patogu prilyginti 0); parametras ω sieja lygtis. Pritaikykime spiralės parametrinį pavidalą medžio $G = (V, E)$ vizualizacijai trimatėje erdvėje. Tarkime, kad realizavę dalį radialinio grafų vaizdavimo algoritmo, apskaičiavome posūkių kampus $\psi[v],\ \forall v \in V$. Šie kampai ne kas kita, kaip ω kiekvienai medžio viršūnės koordinatei $(x_v; y_v; z_v)$. Rutulio spindulys r atitinka lygmenį $m[v],\ \forall v \in V$. Tada bendru atveju teisingos lygybės:

$$\begin{cases} x_v = \dfrac{\mu_1\mu_2 i^v \cos\psi[v]}{\cosh(\cot(c)(\psi[v]-\omega_0))} \\ y_v = \dfrac{\mu_1 i^v \sin\psi[v]}{\cosh(\cot(c)(\psi[v]-\omega_0))} \\ z_v = i^v \cdot \tanh(\cot(c)(\psi[v]-\omega_0)) \end{cases}$$

Parametrai μ_1 ir μ_2 nusako elipsoido ištemptumą, $i = m[v]$ - medžio lygmuo, υ - elipsoidų retėjimo koeficientas. Jei pastebėjote, atsisakėme įvairių pasukimų parametrų, nes daugelyje kompiuterinių programų yra galimybė trimatėje erdvėje dinamiškai judinti objektus rankiniu būdu (pavyzdžiui Maple aplinkoje).

3.1.1 Algoritmo pseudokodas

Įvesties duomenys: $G = (V, E)$ - medis,
$\lambda \in [0;1]$ - medžio viršūnių sektorių dydžių parametras,
μ_1, μ_2 - elipsoido ištemptumo parametrai,
ω_0 - spiralės padėties sferoje parametras,
c - spiralės tankumo parametras,
υ - elipsoidų retėjimo parametras,
$\tau \in (0;\infty)$ - skritulio išpjovos kampas.

R3D(*G, λ, μ1, μ2, ω0, c, υ, τ*):

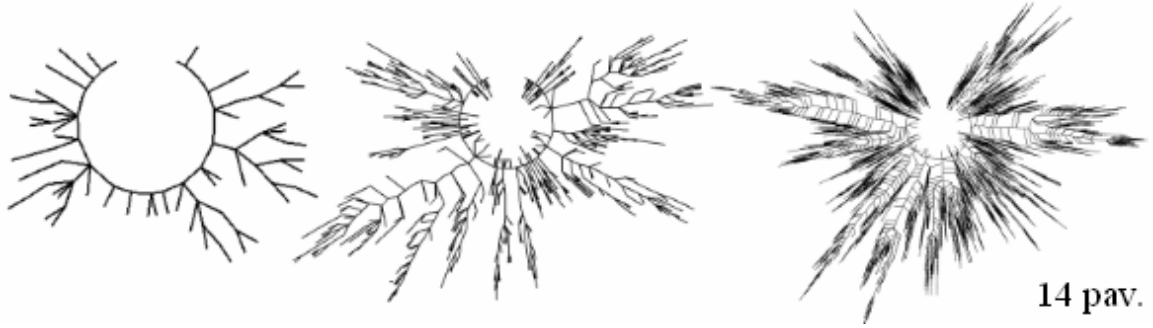
1. $G_1 = G \quad (V_1 = V,\ E_1 = E)$
2. for each $t \in V_1$ do
3. $\quad n[t] = 0$
4. $\quad vaik[t] = 0$
5. $S = \{\emptyset\}$
6. $p = 1$
7. while $|V_1| > 2$ do
8. $\quad$ for each $t \in V_1$ do
9. $\quad\quad$ if $|Adj[t]| = 1$ then
10. $\quad\quad\quad \pi[t] = Adj[t]$
11. $\quad\quad\quad S = S \cup \{t\}$
12. $\quad$ for each $u \in S$ do
13. $\quad\quad n[\pi[u]] = n[\pi[u]] + n[u] + 1$
14. $\quad\quad vaik[\pi[u]] = vaik[\pi[u]] + 1$
15. $\quad V_1 = V_1 \setminus S$
16. $\quad p = p + 1$
17. if $|V_1| = 1$ then
18. $\quad s = V_1$
19. $\quad$ else if $|Adj[V_1[1]]| = \max(|Adj[V_1[1]]|, |Adj[V_1[2]]|)$ then
20. $\quad\quad s = V_1[1]$
21. $\quad\quad \pi[V_1[2]] = s$
22. $\quad$ else
23. $\quad\quad s = V_1[2]$

24. $\pi[V_1[1]] = s$
25. $n[s] = |V| - 1$
26. $\pi[s] = 0$
27. $\varphi[s] = \tau$
28. $\psi[s] = 0$
29. $\beta = 0$
30. $x[s] = 0$
31. $y[s] = 0$
32. $z[s] = 0$
33. $U[0] = s$
34. $P = [\ ]$
35. $i = 0$
36. while $i < p$ do
37. for each $u \in U[i]$ do
38. for each $t \in \{Adj[u]\} \setminus \{\pi[u]\}$ do
39. $$\varphi[t] = \frac{n[t] \cdot \lambda + 1}{vaik[\pi[t]] + (n[\pi[t]] - vaik[\pi[t]]) \cdot \lambda} \cdot \varphi[\pi[t]]$$
40. $$\psi[t] = \psi[\pi[t]] - \frac{\varphi[\pi[t]]}{2} + \frac{\varphi[t]}{2} + \beta$$
41. $$\beta = \beta + \varphi[t]$$
42. $$x[t] = \frac{\mu_1 \mu_2 i^{\upsilon} \cos \psi[v]}{\cosh(\cot(c)(\psi[v] - \omega_0))}$$
43. $$y[t] = \frac{\mu_1 i^{\upsilon} \sin \psi[v]}{\cosh(\cot(c)(\psi[v] - \omega_0))}$$
44. $$z[t] = i^{\upsilon} \cdot \tanh(\cot(c)(\psi[v] - \omega_0))$$
45.
46. $P = enqueue(P, \{Adj[u]\} \setminus \{\pi[u]\})$
47. $\beta = 0$
48. $i = i + 1$
49. $U[i] = P$
50. $P = [\ ]$
51. END

Išvesties duomenys: koordinatės $(x[t], y[t], z[t])$ kiekvienai medžio viršūnei $t \in V$.

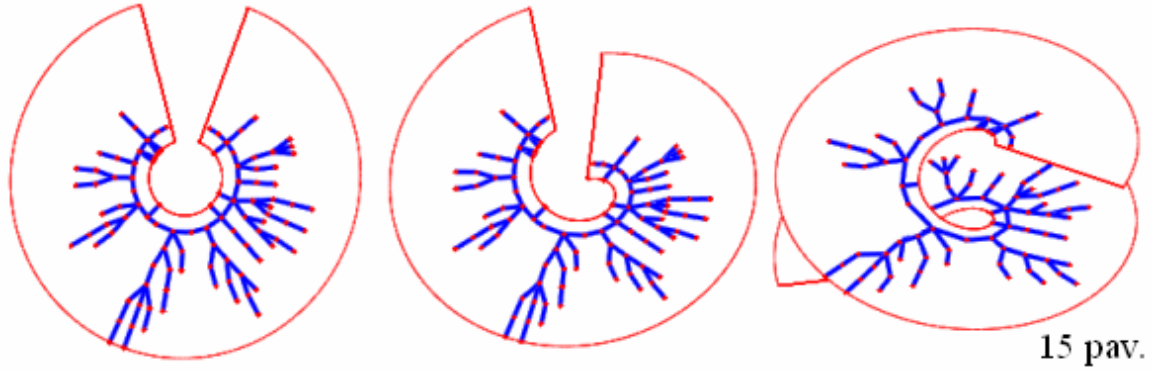
3.2 Laurų vainiko algoritmas trimatei erdvei

Išnagrinėję radialinį medžių vaizdavimą erdvėje, pamėginkime tą pačią vizualizacijos idėją pritaikyti laurų vainiko algoritmui. Viršūnių sutankėjimas plokštumoje didinant grafo eilę ir čia neišvengiamas (14 pav.).



14 pav.

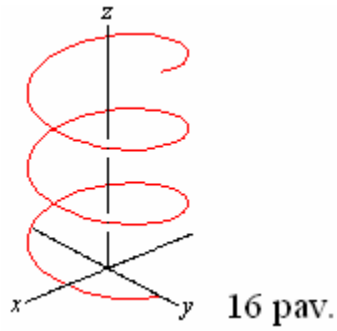
Pereidami nuo plokštumos prie trimatės erdvės (15 pav.), vėlgi turime galimybę neribotai didinti skritulio, ant kurio užtemtas medis, išpjovos kampą.



15 pav.

Pasinaudokime spiralės, einančios cilindro paviršiumi (16 pav.), parametrine lygties forma

$$\begin{cases} x = \sin t, \\ y = \cos t, \\ z = t. \end{cases}$$



16 pav.

Susiekime šią lygčių sistemą su medžio viršūnių koordinatėmis:

$$\begin{cases} x_v = \mu_1 i^{\upsilon} \sin \psi[v] \\ y_v = i^{\upsilon} \cos \psi[v] \\ z_v = \mu_2 \psi[v] \end{cases}$$

Papildomi parametrai μ_1 ir μ_2 parodo atitinkamai spiralės ištemptumą (x ir y ašių kryptimi) ir spiralės sutankėjimą (z ašies kryptimi). $i = m[v]$ - medžio lygmuo, υ - spiralių retėjimo koeficientas.

3.2.1 Algoritmo pseudokodas

Įvesties duomenys:
$G = (V, E)$ - medis,
tk - ilgiausio tako numeris,
m - elipsės, kurioje vaizduojamas ilgiausias takas, lygmuo,
υ - spiralių retėjimo koeficientas,
k - maksimali palikuonių, vaizduojamų laurų vainiko viduje, eilė,
δ - skritulio, kuriame vaizduojamas medis, išpjovos kampas,
μ_1 - elipsių sistemos ištemptumo parametras,
μ_2 - spiralės ištemptumo parametras,
λ_1 - ilgiausio tako palikuonių pasiskirstymo svoris, priklausantis intervalui $[0,1]$,
λ_2 - ilgiausio tako viršūnių pasiskirstymo svoris, priklausantis intervalui $[0,1]$.

LV3D(*G, tk, m, υ, k, δ, μ1, μ2, λ1, λ2*):

1. $IT(G)$ - vykdoma ilgiausių takų paieška medyje G
2. $G_1 = G \ \ (V_1 = V, \ E_1 = E)$
3. for each $t \in V_1$ do
4. $\quad n[t] = 0$
5. $\quad mrange[t] = 0$
6. $\quad vaik[t] = 0$
7. $\quad n_1[t] = 0$
8. $\quad vaik_1[t] = 0$
9. $S = [\ ]$
10. $p = 1$
11. while $|V_1| > T[tk]$ do
12. $\quad$ for each $v \in V_1 \setminus T[tk]$ do
13. $\quad\quad$ if $|Adj[v]| = 1$ then
14. $\quad\quad\quad \pi[v] = Adj[v]$
15. $\quad\quad\quad S = [S \cup \{v\}]$
16. $\quad$ for each $u \in S$ do
17. $\quad\quad n[\pi[u]] = n[\pi[u]] + n[u] + 1$
18. $\quad\quad n_1[\pi[u]] = n[\pi[u]]$
19. $\quad\quad vaik[\pi[u]] = vaik[\pi[u]] + 1$
20. $\quad\quad vaik_1[\pi[u]] = vaik[\pi[u]]$
21. $\quad\quad mrange[\pi[u]] = mrange[u] + 1$
22. $\quad G_1 = G_1 \setminus \{S\}$
23. $\quad S = [\ ]$
24. $\quad p = p + 1$

25. for each $t \in T[tk]$ do
26. $vaik_1[t] = 0$
27. $G_1 = G \ (V_1 = V, \ E_1 = E)$
28. $G_2 = G_1 \ (V_2 = V_1, \ E_2 = E_1)$
29. $\theta = 0$
30. for each $t \in T[tk]$ do
31. $$\varphi[t] = \frac{(n[t] \cdot \lambda_2 + 1)(2\pi - \delta)}{|T[tk]| + (|V| - |T[tk]|) \cdot \lambda_2}$$
32. $$\psi[t] = \frac{\delta + \varphi[t]}{2} + \theta$$
33. $x[t] = \mu_1 m^{\upsilon} \sin \psi[t]$
34. $y[t] = m^{\upsilon} \cos \psi[t]$
35. $z[t] = \mu_2 \psi[t]$
36. for each $v \in Adj[t] \setminus T[tk]$ do
37. if $mrange[v] < k$ then
38. $V_1 = V_1 \setminus \{v\}$
39. $vaik_1[\pi[v]] = vaik_1[\pi[v]] + 1$
40. else
41. $V_2 = V_2 \setminus \{v\}$
42. $n_1[\pi[v]] = n_1[\pi[v]] - n[v] - 1$
43. $\theta = \theta + \varphi[t]$
44. $\theta = 0$
45. $U_0 = T[tk]$
46. $i = 0$
47. while $i < k$ do
48. for each $j \in U_i$ do
49. for each $t \in Adj[j](G_2) \setminus \pi[j]$ do
50. $$\varphi[t] = \frac{n_1[t] \cdot \lambda_1 + 1}{vaik_1[\pi[t]] + (n_1[\pi[t]] - vaik_1[\pi[t]]) \cdot \lambda_1} \cdot \varphi[\pi[t]]$$
51. $$\psi[t] = \psi[\pi[t]] - \frac{\varphi[\pi[t]] - \varphi[t]}{2} + \theta$$
52. $\theta = \theta + \varphi[t]$
53. $x[t] = \mu_1 (m - i - 1)^{\upsilon} \cdot \sin(\psi[t])$
54. $y[t] = (m - i - 1)^{\upsilon} \cdot \cos(\psi[t])$
55. $z[t] = \mu_2 \psi[t]$
56. $P = [P \cup (Adj[j](G_2) \setminus \pi[j])]$
57. $\theta = 0$
58. $i = i + 1$
59. $U_i = P$

60. $P = [\ ]$
61. $U_0 = T[tk]$
62. $P = [\ ]$
63. $i = 0$
64. for each $t \in T[tk]$ do
65. $n_1[t] = n[t] - n_1[t]$
$vaik_1[t] = vaik[t] - n_1[t]$
66. while $i < p$ do
67. for each $j \in U_i$ do
68. for each $t \in Adj[j](G_1) \setminus \pi[j]$ do
69. $$\varphi[t] = \frac{n_1[t] \cdot \lambda_1 + 1}{vaik_1[\pi[t]] + (n_1[\pi[t]] - vaik_1[\pi[t]]) \cdot \lambda_1} \cdot \varphi[\pi[t]]$$
70. $$\psi[t] = \psi[\pi[t]] - \frac{\varphi[\pi[t]] - \varphi[t]}{2} + \theta$$
71. $\theta = \theta + \varphi[t]$
72. $x[t] = \mu_1 (m + i + 1)^{\upsilon} \cdot \sin(\psi[t])$
73. $y[t] = (m + i + 1)^{\upsilon} \cdot \cos(\psi[t])$
74. $z[t] = \mu_2 \psi[t]$
75. $P = [P \cup (Adj[j](G_1) \setminus \pi[j])]$
76. $\theta = 0$
77. $i = i + 1$
78. $U_i = P$
79. $P = [\ ]$
80. END

Išvesties duomenys: koordinatės $(x[t], y[t], z[t])$ kiekvienai medžio viršūnei $t \in V$.

4. Priuferio kodas

Mes nagrinėjome numeruotuosius medžius (jie buvo apibrėžiami kaip nesutvarkytų viršūnių porų aibė) ir vaizdavome juos grafiškai. Tačiau yra ir kitų būdų, skirtų medžių inicializacijai. Vieną jų aptarsime.

Kiekvienam n-tos eilės medžiui egzistuoja abipusiškai vienareikšmiškas kodas $\alpha = [a_1, a_2, ..., a_{n-2}]$, vadinamas Priuferio kodu. Kaip jis sudaromas? Tegu $G = (V, E)$ - n-tos eilės numeruotasis medis. Jei $n = 2$, tai sutarkime, kad $\alpha = [\ ]$ (tuščias sąrašas). Kai $n > 2$, imame lapą $v \in V$, kurio numeris mažiausias. Tada į Priuferio kodą įrašome $a_1 = Adj[v]$, t. y. gretimos lapui viršūnės numeris ir „nukerpame“ lapą $v \in V$, $V = V \setminus \{v\}$. Šį ciklą tęsiame tol, kol medyje G belieka dvi viršūnės ir išvedame Priuferio kodą $\alpha = [a_1, a_2, ..., a_{n-2}]$. Kad būtų aiškiau, pateiksime šio algoritmo pseudokodą.

Įvesties duomenys: $G = (V, E)$ - medis.

1. $G_1 = G$ $(V_1 = V,\ E_1 = E)$
2. for i from 1 to $|V| - 2$ do
3. $S = [\ ]$
4. for each $v \in V_1$ do
5. if $|Adj[v]| = 1$ then
6. $S = [S \cup \{v\}]$ - sąrašas
7. $p[i] = Adj[\min(S)]$
8. $V_1 = V_1 \setminus \{\min(S)\}$
9. $\alpha = [p[1], p[2], ..., p[|V| - 2]]$
10. END

Išvesties duomenys: Priuferio kodas α.

Dabar aptarkime algoritmą, pagal kurį inicializuojamas medis žinant jo Priuferio kodą. Turime $\alpha = [a_1, a_2, ..., a_{n-2}]$. Kadangi šiame kode yra $n - 2$ elementų, tai medis turės n viršūnių: $V = \{1, 2, ..., n\}$; priskirkime $V_1 = V$. Sudarykime briaunų aibę E pagal tokią taisyklę. Iš aibės V_1 imkime mažiausią elementą b, kurio nėra sąraše α ir įterpkime briauną $\{\alpha_1, b\}$ į aibę E: $E = E \cup \{\alpha_1, b\}$. Tada iš aibės V_1 pašaliname elementą b, $V_1 = V_1 \setminus \{b\}$ ir ištriname pirmą sąrašo α elementą: $dequeue(\alpha)$. Šį ciklą tęsiame tol, kol sąraše α nebelieka elementų ir tada į aibę E įrašome briauną, kurios viršūnės - 2 likę aibės V_1 elementai. Gauname medį $G = (V, E)$.

Pastaba: programuojant kurioje nors aplinkoje pradiniai (įvesties) duomenys yra formalūs parametrai, kurių keisti negalime, todėl Priuferio algoritmo pseudokoduose įvedami nauji pagalbiniai kintamieji: $G_1 = G$ ir $\alpha_1 = \alpha$.

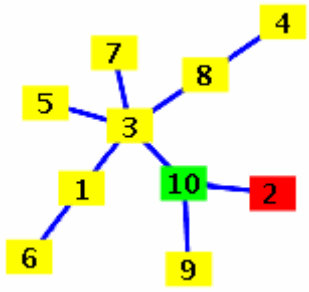
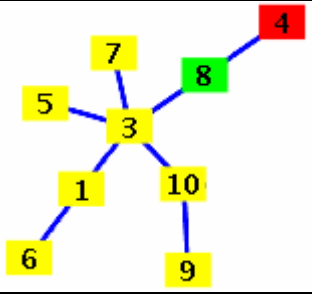
Įvesties duomenys: Priuferio kodas α.

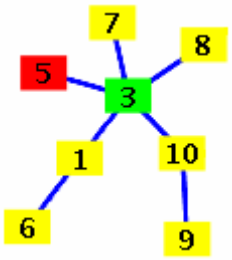
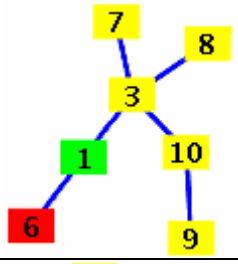
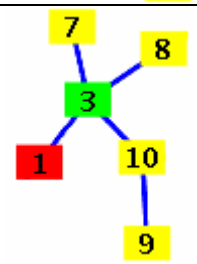
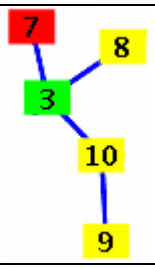
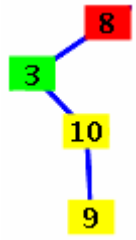
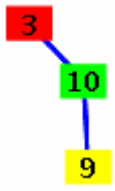
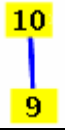
1. $\alpha_1 = \alpha$
2. $V = \{1,2,...,|\alpha|+2\}$
3. $E = \{\emptyset\}$
4. $i = 1$
5. while $\alpha_1 \neq [\ ]$ do
6. if $V[i] \notin \alpha_1$ then
7. $E = E \cup \{V[i], \alpha_1[1]\}$
8. $dequeue(\alpha_1)$
9. $V = V \setminus V[i]$
10. $i = 1$
11. else
12. $i = i + 1$
13. $G = (V, E)$
14. END

Išvesties duomenys: $G = (V, E)$ - medis.

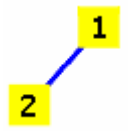
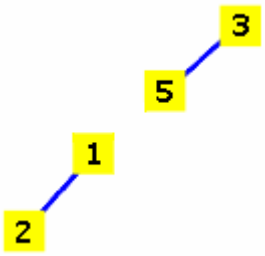
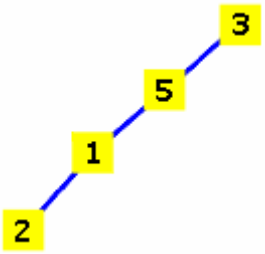
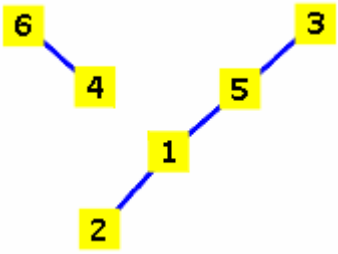
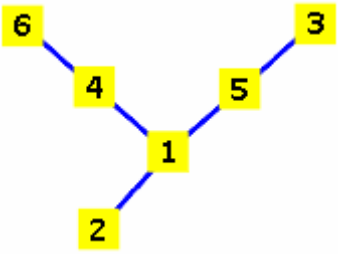
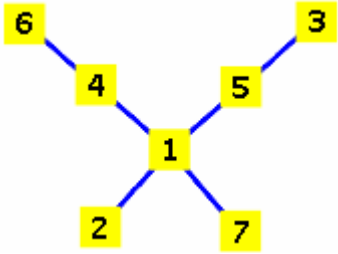
Išvada: kadangi korektiškai apibrėžėme bijekcinį atvaizdį tarp n-tos eilės medžio $G = (V, E)$ ir Priuferio kodo $\alpha = [a_1, a_2, ..., a_{n-2}]$, kur $a_i \in V$, $i = 1,...,n-2$, todėl iš viso egzistuoja n^{n-2} numeruotųjų n-tos eilės medžių.

4.1 pavyzdys. Sudarykime medžio $G = (V, E)$ Priuferio kodą α.

	$G = Graph(\{\{1,3\},\{1,6\},\{2,10\},\{3,5\},\{3,7\},\{3,8\},\{3,10\},\{4,8\},\{9,10\}\})$ $\alpha = [10]$ $V = V \setminus \{2\}$
	$G = Graph(\{\{1,3\},\{1,6\},\{3,5\},\{3,7\},\{3,8\},\{3,10\},\{4,8\},\{9,10\}\})$ $\alpha = [10,8]$ $V = V \setminus \{4\}$

	$G = Graph(\{\{1,3\},\{1,6\},\{3,5\},\{3,7\},\{3,8\},\{3,10\},\{9,10\}\})$ $\alpha = [10,8,3]$ $V = V \setminus \{5\}$
	$G = Graph(\{\{1,3\},\{1,6\},\{3,7\},\{3,8\},\{3,10\},\{9,10\}\})$ $\alpha = [10,8,3,1]$ $V = V \setminus \{6\}$
	$G = Graph(\{\{1,3\},\{3,7\},\{3,8\},\{3,10\},\{9,10\}\})$ $\alpha = [10,8,3,1,3]$ $V = V \setminus \{1\}$
	$G = Graph(\{\{3,7\},\{3,8\},\{3,10\},\{9,10\}\})$ $\alpha = [10,8,3,1,3,3]$ $V = V \setminus \{7\}$
	$G = Graph(\{\{3,7\},\{3,10\},\{9,10\}\})$ $\alpha = [10,8,3,1,3,3,3]$ $V = V \setminus \{8\}$
	$G = Graph(\{\{3,10\},\{9,10\}\})$ $\alpha = [10,8,3,1,3,3,3,10]$ $V = V \setminus \{3\}$
	Kadangi $\lvert V \rvert = 2$, algoritmas baigiamas ir išvedamas α.

4.2 pavyzdys. Pagal Priuferio kodą $\alpha = [1,5,1,4,1]$ sugeneruokime medį $G = (V, E)$.

	$\alpha = [1,5,1,4,1]$ $V_1 = \{1,2,3,4,5,6,7\}$ $E = \{\{2,1\}\}$
	$\alpha = [5,1,4,1]$ $V_1 = \{1,3,4,5,6,7\}$ $E = \{\{2,1\},\{3,5\}\}$
	$\alpha = [1,4,1]$ $V_1 = \{1,4,5,6,7\}$ $E = \{\{2,1\},\{3,5\},\{5,1\}\}$
	$\alpha = [4,1]$ $V_1 = \{1,4,6,7\}$ $E = \{\{2,1\},\{3,5\},\{5,1\},\{6,4\}\}$
	$\alpha = [4]$ $V_1 = \{1,4,7\}$ $E = \{\{2,1\},\{3,5\},\{5,1\},\{6,4\},\{4,1\}\}$
	$\alpha = [\ ]$ $V_1 = \{1,7\}$ $E = \{\{2,1\},\{3,5\},\{5,1\},\{6,4\},\{4,1\},\{1,7\}\}$, $G = Graph(\{\{1,2\},\{1,4\},\{1,5\},\{1,7\},\{3,5\},\{4,6\}\})$, algoritmo pabaiga.

5. Medžių vizualizacijos algoritmų taikymas

5.1 Paieškos į plotį vizualizacija

Nagrinėsime paieškos į plotį algoritmą [13], kurį modifikuosime įterpdami papildomas procedūras, vaizduojančias grafą bei keičiančias jo briaunų ir viršūnių spalvas tam tikruose algoritmo realizacijos etapuose.

Grafo $G = (V, E)$ vizualizacijos procedūros:
$HighlightVertex(G, V_1, color)$ - grafo G viršūnių (-ės) $V_1 \in V$ nuspalvinimas tam tikra spalva;
$HighlightEdges(G, E_1, color)$ - grafo G briaunų (-os) $E_1 \in E$ nuspalvinimas tam tikra spalva;
$DrawGraph(G)$ - grafo vaizdavimas kompiuterio ekrane.

Naudosime penkias spalvas:
$color\_1$ - pradinio (inicializuoto) grafo viršūnių spalva;
$color\_2$ - algoritmo vykdymo metu nagrinėjamų viršūnių spalva;
$color\_3$ - algoritmo vykdymo išnagrinėtų (pereitų) viršūnių spalva;
$color\_4$ - pradinio grafo briaunų spalva;
$color\_5$ - briaunų, incidenčių nagrinėjamai viršūnei, spalva.

Algoritmo realizacijos metu pereinamos visos grafo viršūnės, pasiekiamos iš pradinės viršūnės s; formuojamas sąrašas Q apdorojamų viršūnių, kurioms priskiriami spalvos c, atstumo d ir tėvystės π atributai. Jei atributas viršūnei nepriskiriamas, jis prilyginamas NIL-ui.

Realizavus algoritmą, atributai $d[u]$, $\forall u \in V$ atitinka trumpiausius atstumus nuo pradinės viršūnės s iki u, $\forall u \in V$; atributai $\pi[u]$, $\forall u \in V$ parodo, iš kurios viršūnės algoritmo realizacijos metu buvo ateita į viršūnę u; $c[u]$ atitinka viršūnės u spalvą, $\forall u \in V$.

Maple programos aplinkoje patogu parinkti šias spalvas: $color\_1$ - žalia, $color\_2$ - geltona, $color\_3$ - raudona, $color\_4$ - pilka, $color\_5$ - juoda. Pateiksime modifikuoto paieškos į plotį algoritmo pseudokodą ir konkretų pavyzdį.

5.1.1 Algoritmo pseudokodas

Įvesties duomenys:

$G = (V, E)$ - medis,

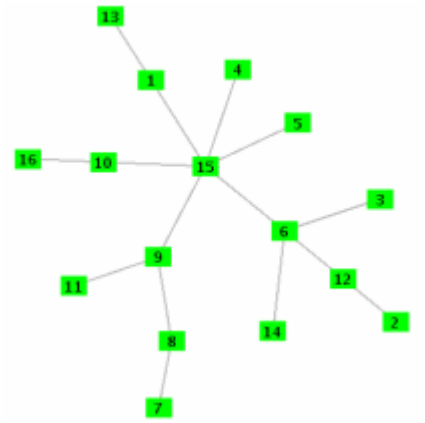
s - pradinė viršūnė.

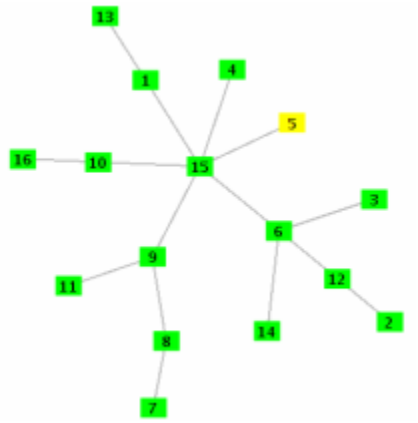
P-Plot(*G, s*):

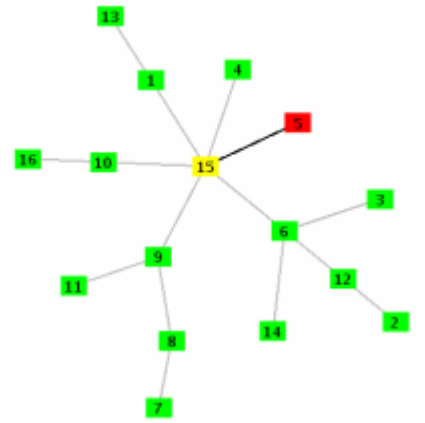
1. Realizuojamas kuris nors vizualizacijos algoritmas
2. $HighlightEdges(G, E, color\_4)$
3. $HighlightVertex(G, V, color\_1)$
4. $DrawGraph(G)$
5. for each $u \in V \setminus \{s\}$ do
6. $\quad c[u] = color\_1$
7. $\quad d[u] = \infty$
8. $\quad \pi[u] = NIL$
9. $c[s] = color\_2$
10. $d[s] = 0$
11. $\pi[s] = NIL$
12. $Q = [s]$
13. $HighlightVertex(G, s, color\_2)$
14. $DrawGraph(G)$
15. while $Q \neq [\ ]$ do
16. $\quad u = Q[1]$
17. $\quad$ for each $v \in Adj[u]$ do
18. $\quad\quad$ if $c[v] = color\_1$ then
19. $\quad\quad\quad c[v] = color\_2$
20. $\quad\quad\quad d[v] = d[u] + 1$
21. $\quad\quad\quad \pi[v] = u$
22. $\quad\quad\quad HighlightEdges(G, \{v, u\}, color\_5)$
23. $\quad\quad\quad HighlightVertex(G, v, color\_2)$
24. $\quad\quad\quad enqueue(Q, v)$
25. $\quad\quad$ if $c[v] = color\_2$ then
26. $\quad\quad\quad HighlightEdges(G, \{v, u\}, color\_5)$
27. $\quad dequeue(Q)$
28. $\quad HighlightVertex(G, u, color\_3)$
29. $\quad c[u] = color\_3$
30. $\quad DrawGraph(G)$
31. END

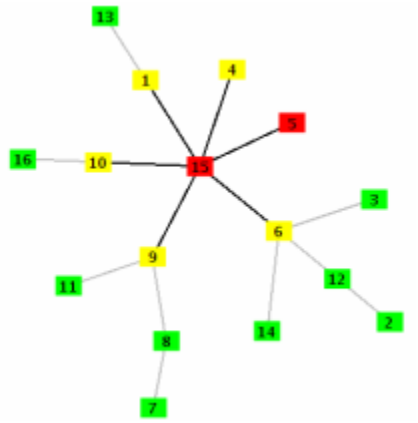
Išvesties duomenys: esminių algoritmo žingsnių vizualizacija.

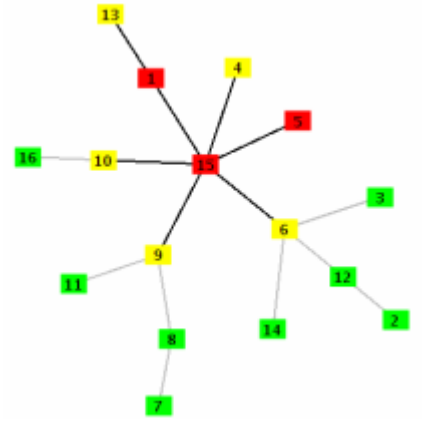
5.1.2 Pavyzdys. Paieška į plotį **P-Plot(*G, 5*)** medyje, kuris inicializuotas Priuferio kodu $\alpha = [12,6,15,15,8,9,9,15,6,1,15,6,15,10]$ bei pavaizduotas plokštumoje radialiniu būdu.

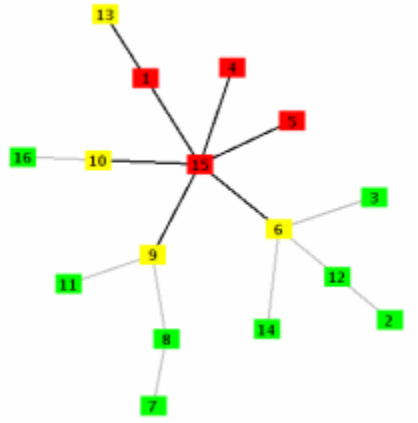
<table>
<tr>
<td> 

$Q = [\ ]$.</td>
<td> 

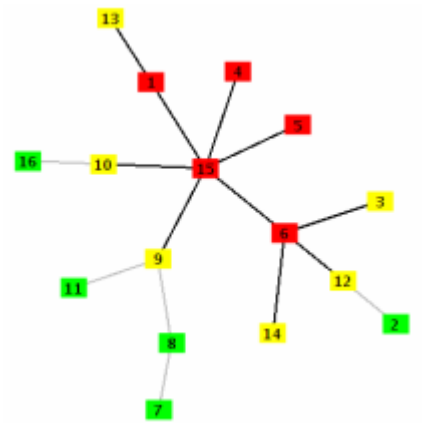
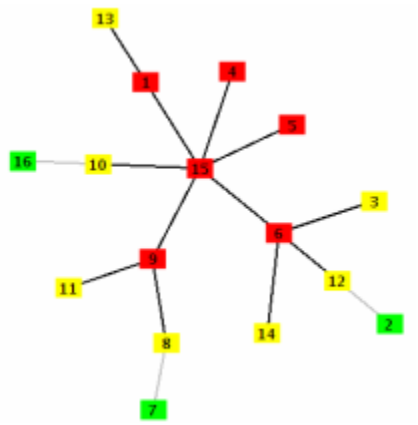
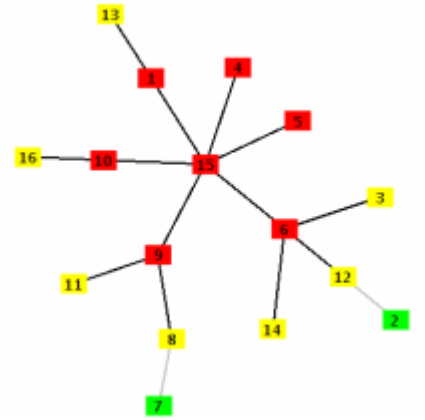
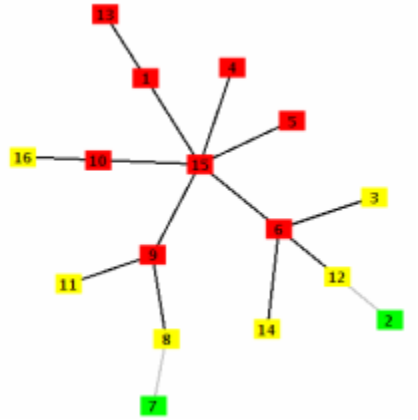
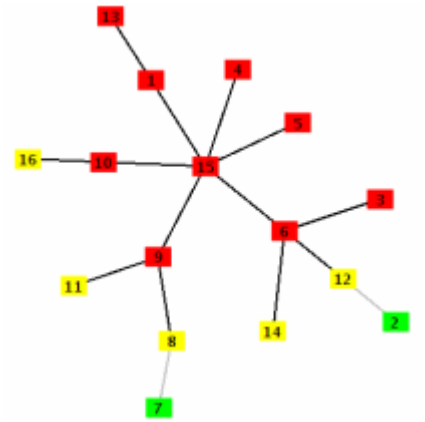
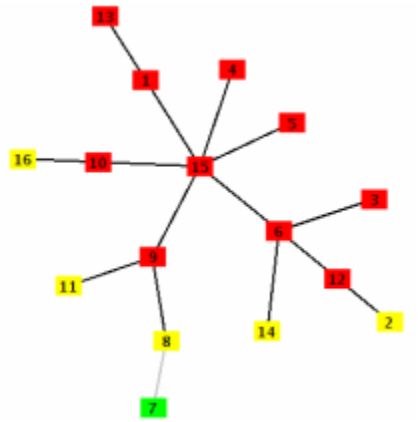
$s = 5$, $d[5] = 0$, $\pi[5] = NIL$, $Q = [5]$.</td>
</tr>
<tr>
<td> 

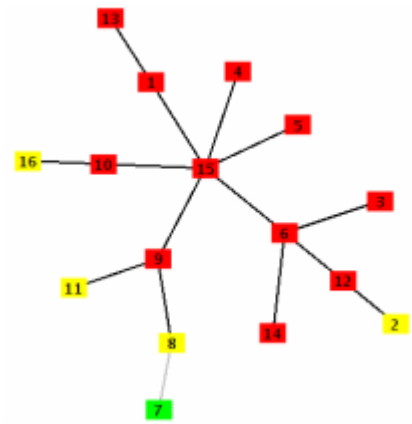
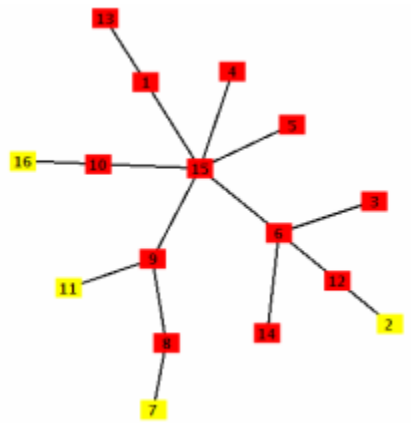
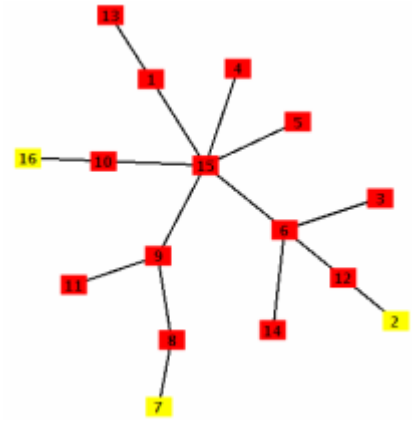
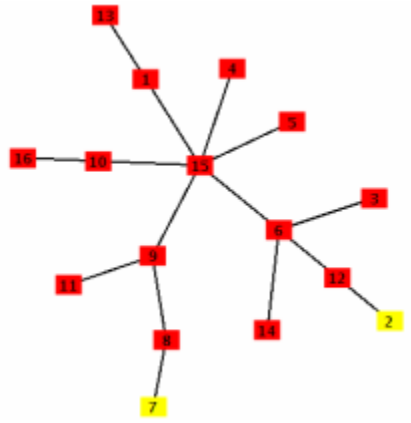
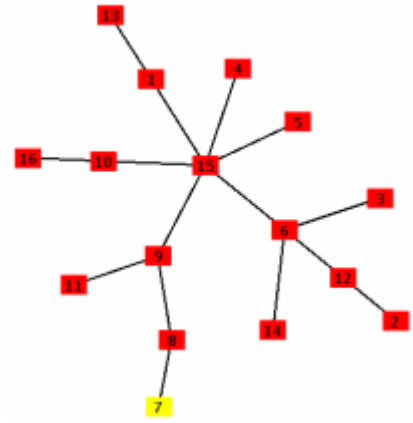
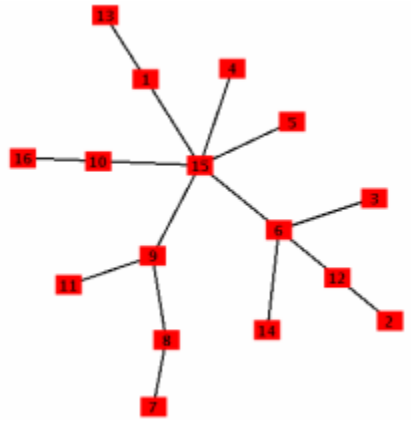
$d[15] = 1$, $\pi[15] = 5$, $Q = [15]$.</td>
<td> 

$d[1] = 2$, $\pi[1] = 15$, $d[4] = 2$, $\pi[4] = 15$,
$d[6] = 2$, $\pi[6] = 15$, $d[9] = 2$, $\pi[9] = 15$,
$d[10] = 2$, $\pi[10] = 15$, $Q = [1,4,6,9,10]$.</td>
</tr>
<tr>
<td> 

$d[13] = 3$, $\pi[13] = 1$, $Q = [4,6,9,10,13]$.</td>
<td> 

$Q = [6,9,10,13]$.</td>
</tr>
</table>

 $d[3]=3,\ \pi[3]=6,\ d[12]=3,\ \pi[12]=6,$ $d[14]=3,\ \pi[14]=6,\ Q=[9,10,13,3,12,14].$	 $d[8]=3,\ \pi[8]=9,\ d[11]=3,\ \pi[11]=9,$ $Q=[10,13,3,12,14,8,11].$
 $d[16]=3,\ \pi[16]=10,$ $Q=[13,3,12,14,8,11,16].$	 $Q=[3,12,14,8,11,16].$
 $Q=[12,14,8,11,16].$	 $d[2]=4,\ \pi[2]=12,\ Q=[14,8,11,16,2].$

 $Q=[8,11,16,2]$.	 $d[7]=4$, $\pi[7]=8$, $Q=[11,16,2,7]$.
 $Q=[16,2,7]$.	 $Q=[2,7]$.
 $Q=[7]$.	 $Q=[\ ]$, algoritmo pabaiga.

5.2 Paieškos į gylį vizualizacija

Aptarsime paieškos į gylį algoritmo [14] žingsnių vizualizaciją. Pagrindinis tikslas – pereiti grafo $G = (V, E)$ viršūnes pagal tokią strategiją: nauja viršūnė atrandama einant briauna, incidenčia ką tik atrastai viršūnei. Jei grįžtama į tą pačią atrastą viršūnę, paieškos procesas tęsiamas peršokant į dar neatrastas viršūnes pagal jų numeraciją.

Vizualizacijai naudosime praeitame skyriuje apibrėžtas procedūras bei penkias spalvas:

$color\_1$ - pradinio (inicializuoto) grafo viršūnių spalva;

$color\_2$ - algoritmo vykdymo metu nagrinėjamų viršūnių spalva;

$color\_3$ - algoritmo vykdymo išnagrinėtų (pereitų) viršūnių spalva;

$color\_4$ - pradinio grafo briaunų spalva;

$color\_5$ - briaunų, kuriomis buvo eita algoritmo realizacijos metu, spalva.

Kaip ir praeitame skyriuje, algoritmo žingsnių vizualizacijai parinksime tokias spalvas: $color\_1$ - žalia, $color\_2$ - geltona, $color\_3$ - raudona, $color\_4$ - pilka, $color\_5$ - juoda.

Algoritmo vykdymo metu viršūnei $u \in V$ priskiriami šie atributai: $d[u]$ - viršūnės pastebėjimo laiko momentas, $f[u]$- viršūnės apdorojimo pabaigos laiko momentas, $\pi[u]$ - viršūnės tėvas, $c[u]$ - spalvos atributas. Nepriskirtus viršūnėms atributus žymėsime NIL-ais. Globalus laikas bus žymimas raide t.

Verta paminėti rekursinio tipo procedūrą $VISIT(u)$, į kurią kreipiamasi kaskart grafe aplankant naują viršūnę u. Kadangi realizuojant algoritmą procedūra $VISIT(u)$ kreipiasi pati į save, todėl nauja viršūnė ieškoma iš ką tik atrastos viršūnės t.y. vyksta paieška į gylį.

Schematiškai pavaizduosime modifikuoto paieškos į gylį algoritmo vykdymo eigą.

5.2.1 Algoritmo pseudokodas

Įvesties duomenys:

$G = (V, E)$ - medis,

P-Gyl(*G*):

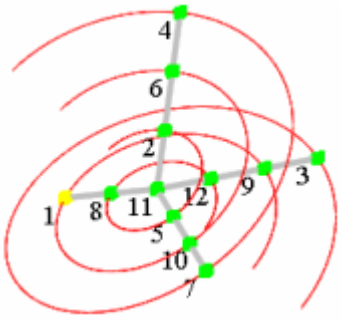
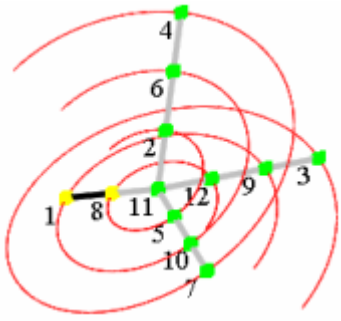
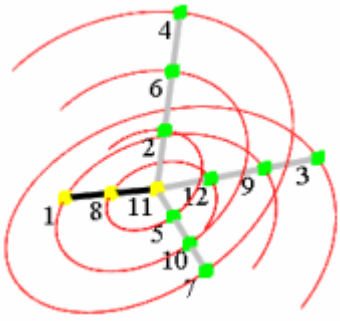
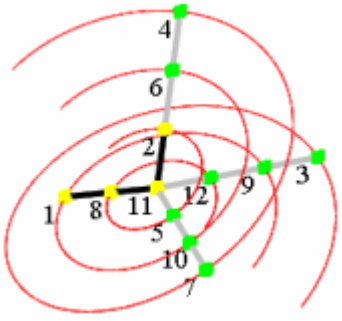
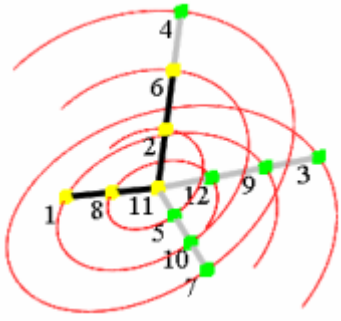
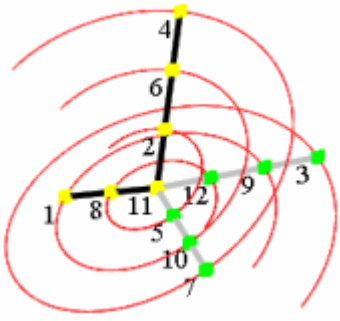
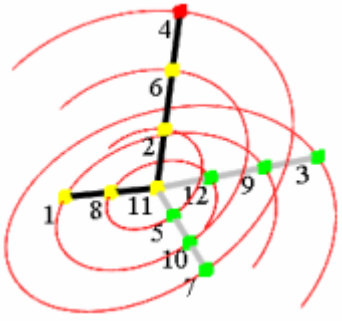
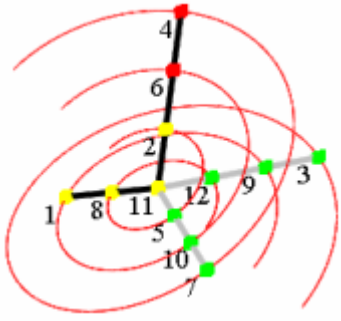
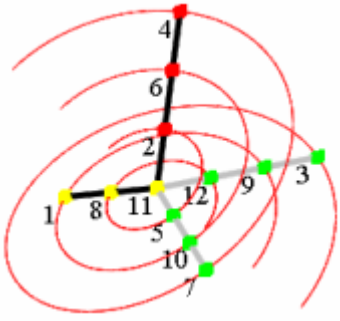
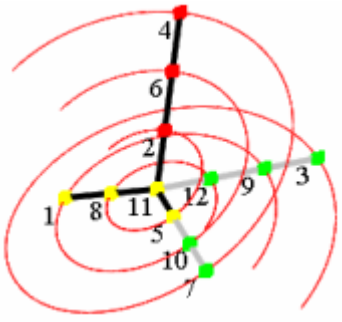
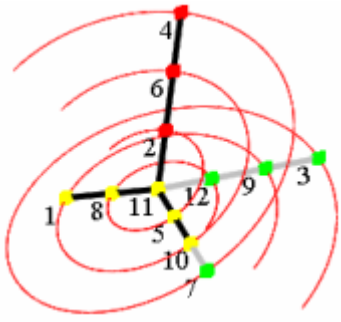
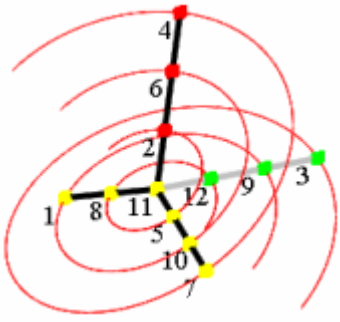
1. Realizuojamas kuris nors vizualizacijos algoritmas
2. $HighlightEdges(G, E, color\_4)$
3. $HighlightVertex(G, V, color\_1)$
4. $DrawGraph(G)$
5. for each $u \in V$ do
6. $\quad c[u] = color\_1$
7. $\quad \pi[u] = NIL$
8. $t = 0$
9. for each $u \in V$ do
10. $\quad$ if $c[u] = color\_1$ then
11. $\quad\quad VISIT(u)$

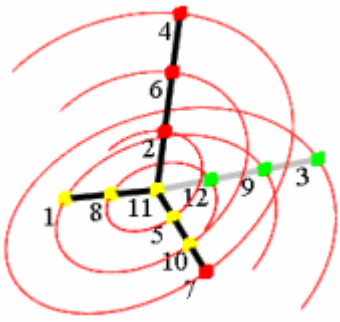
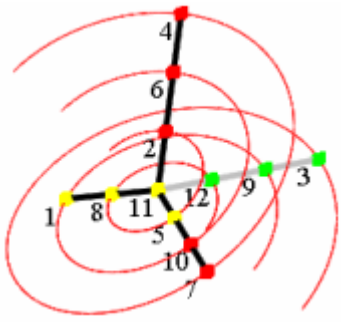
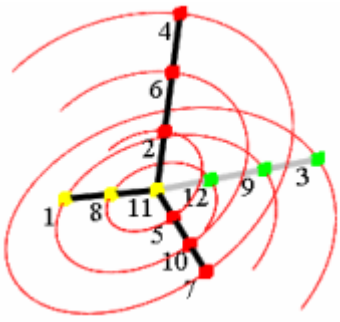
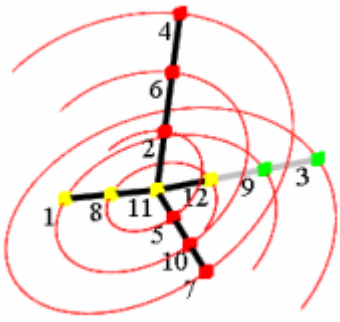
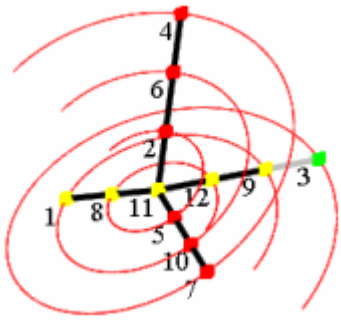
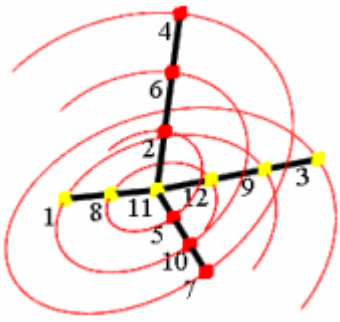
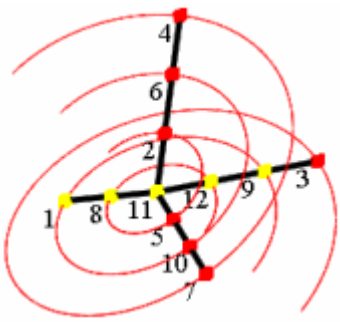
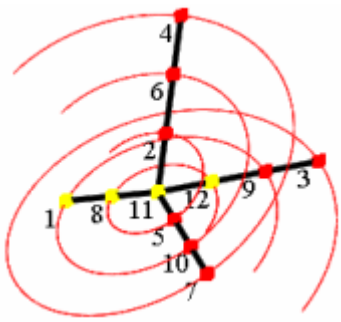
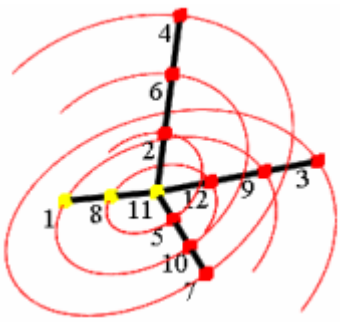
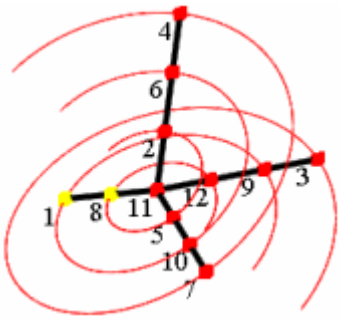
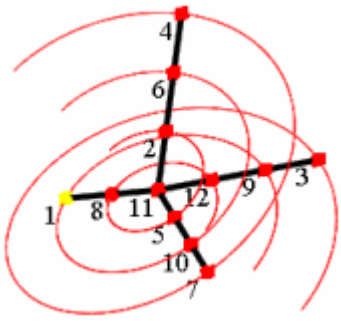
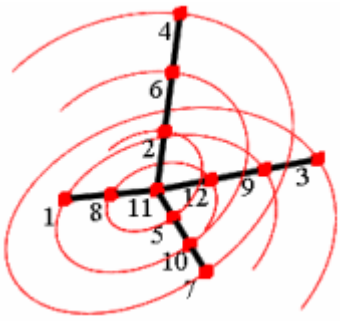
VISIT(*u*):

1. $c[u] = color\_2$
2. $HighlightVertex(G, u, color\_2)$
3. $t = t + 1$
4. $d[u] = t$
5. $DrawGraph(G)$
6. for each $v \in Adj[u]$ do
7. $\quad HighlightEdges(G, \{v, u\}, color\_5)$
8. $\quad$ if $c[v] = color\_1$ then
9. $\quad\quad \pi[v] = u$
10. $\quad\quad VISIT(v)$
11. $c[u] = color\_3$
12. $HighlightVertex(G, u, color\_3)$
13. $t = t + 1$
14. $f[u] = t$
15. $DrawGraph(G)$
16. END

Išvesties duomenys: esminių algoritmo žingsnių vizualizacija.

5.2.2 Pavyzdys. Medis inicializuotas Priuferio kodu $\alpha = [8,9,6,2,11,10,11,12,5,11]$ ir vizualizuotas trimatėje erdvėje radialiniu būdu. Realizuota paieška **P-Gyl(*G*)**.

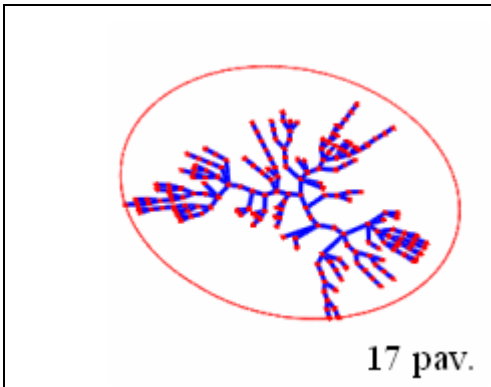
 $\pi[1] = NIL,\ d[1] = 1.$	 $\pi[8] = 1,\ d[8] = 2.$	 $\pi[11] = 8,\ d[11] = 3.$
 $\pi[2] = 11,\ d[2] = 4.$	 $\pi[6] = 2,\ d[6] = 5.$	 $\pi[4] = 6,\ d[4] = 6.$
 $f[4] = 7.$	 $f[6] = 8.$	 $f[2] = 9.$
 $\pi[5] = 11,\ d[5] = 10.$	 $\pi[10] = 5,\ d[10] = 11.$	 $\pi[7] = 10,\ d[7] = 12.$

 $f[7]=13.$	 $f[10]=14.$	 $f[5]=15.$
 $\pi[12]=11,\ d[12]=16.$	 $\pi[9]=12,\ d[9]=17.$	 $\pi[3]=9,\ d[3]=18.$
 $f[3]=19.$	 $f[9]=20.$	 $f[12]=21.$
 $f[11]=22.$	 $f[8]=23.$	 $f[1]=24.$

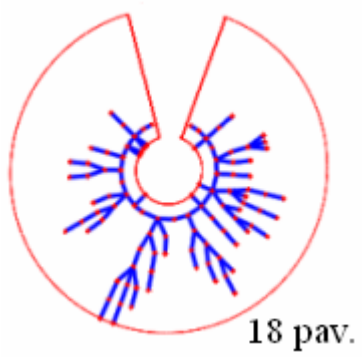
Darbo apibendrinimas ir išvados

Šiame darbe išplėsdami radialinio medžių vaizdavimo idėją [6], pasiūlėme keturis naujus algoritmus, skirtus medžių vizualizacijai plokštumoje ir erdvėje:

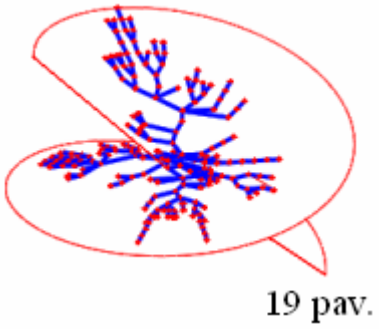
- apibendrintą radialinį medžių vaizdavimą (17 pav.),
- laurų vainiko algoritmą (18 pav.),
- apibendrintą radialinį medžių vaizdavimą trimatėje erdvėje (19 pav.),
- laurų vainiko algoritmą trimatei erdvei (20 pav.).



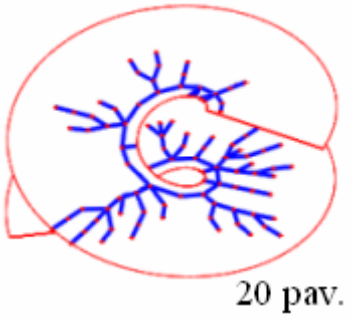
17 pav.



18 pav.



19 pav.



20 pav.

Remdamiesi medžio centro paieškos metodu [9] ir Priuferio pasiūlytu medžio kodu [10], Maple programos aplinkoje realizavome tris papildomus algoritmus:

- ilgiausių takų paiešką medyje,
- medžio generavimą pagal Priuferio kodą,
- Priuferio kodo sudarymą pagal medį.

Darbe pasiūlėme originalią idėją vizualizuoti paiešką į plotį [13] ir į gylį [14] įterpiant papildomas procedūras, tam tikruose algoritmo realizacijos etapuose vaizduojančias grafus bei keičiančias jų briaunų ir viršūnių spalvas.

Mūsų darbą galima būtų pratęsti nagrinėjant ir kuriant grafų, digrafų vizualizacijos algoritmus, realizuojant juos tam tikrose aplinkose bei taikant įvairiems grafų paieškos algoritmams vaizduoti.

Summary

In this paper, referring to radial tree drawing, through the medium of Maple program, an algorithm has been realized. It was meant to carry out the visualization of rooted trees on a plane. After that, randomly generating thousands of trees of a different order, it has been empirically identified that aesthetic requirements are not suitable in all cases. The arisen problem has been tackled by introducing additional parameters: they have added flexibility to trees, the longest paths representing in one of the concentric ellipsis, pulling the tree on dimensional spiral surfaces.

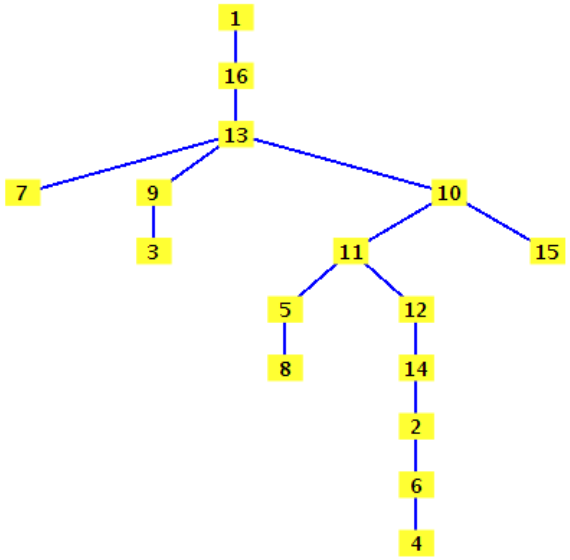
So we have summarized the algorithm of radial tree drawing, we have suggested to apply the method of finding the tree centre in identifying the longest trails. We have created a not described yet method of the algorithm of the laurel wreath, which is meant for portraying various trees on a plane. For ease of application we have also adjusted the code introduced by Prüfer: it is meant for generating trees and for deducing Prüfer‘s code of a determined tree. By modifying the algorithm of radial tree drawing, we have pictured the trees in a system of concentric ellipsoids. The paper suggests an original idea to visualize the breadth first search and the depth first search by inserting additional procedures, showing graphs and changing the colors of their edges and summits at certain stages of the realization of the algorithm.

Priedai

(algoritmų pirminiai tekstai Maple 13 – Maple 14 programų aplinkose)

Priuferio kodo sudarymas pagal medį

```
> restart;
> with(GraphTheory):
> with(RandomGraphs):
> with(Statistics):

> priufer1:=proc(T)
> local A,S,i,v;
> global priufer;
> priufer:='priufer';
> A:=T;
> for i from 1 to Count(Vertices(T))-2 do
> S:=[]:
> for v in Vertices(A) do if
> Count(Neighbors(A,v))=1 then
> S:=[`union`({S[]},{v})[]]
> end if;
> end do;
> priufer[i]:=Neighbors(A,min(S))[];
> A:=DeleteVertex(A,min(S));
> end do;
> priufer:=[seq(priufer[j],j=1..Count(Vertices(T))-2)];
> return(priufer);
> end proc;

> T:=RandomTree(16):DrawGraph(T);
```

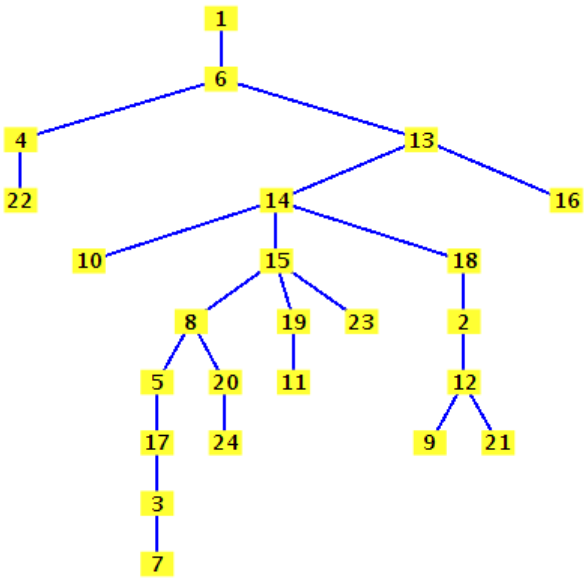



```
> priufer1(T);
```

[16, 9, 6, 2, 14, 13, 5, 11, 13, 12, 11, 10, 10, 13]

Medžio generavimas pagal Priuferio kodą

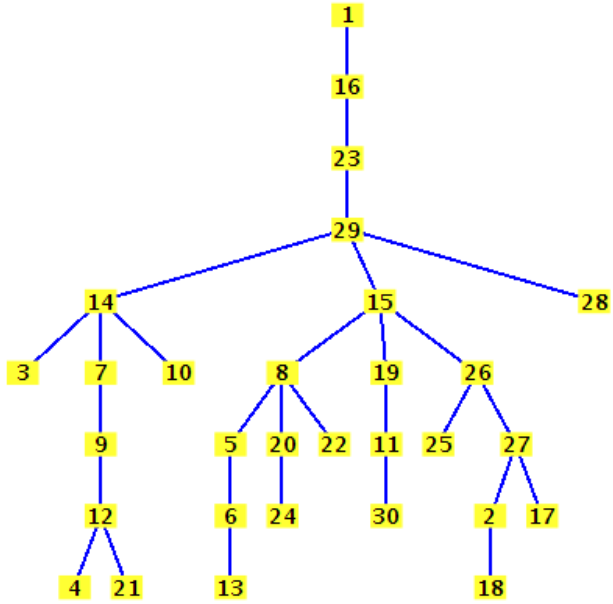
```
> restart;
> with(GraphTheory):
> with(RandomGraphs):
> with(Statistics):

> priufer2:=proc(priufer)
> local copy,V,vert,k,G;
> copy:=priufer;
> V:={seq(i,i=1..Count(copy)+2)}:
> vert:={}:
> k:=1:
> while is(copy<>[]) do
> if is(V[k] in copy)=false then
> vert:={`union`({vert[]},{{V[k],copy[1]}})[]}:
> copy:=[seq(copy[j],j=2..Count(copy))]:
> V:=[`minus`({V[]},{V[k]})[]]:
> k:=1:
> else k:=k+1
> end if:
> end do:
> vert:={`union`({vert[]},{{V[]}})[]}:
> G:=Graph(vert):
> return(DrawGraph(G));
> end proc;

> priufer:=[6,3,17,12,14,19,13,5,8,15,12,2,18,14,4,6,13,14,15,15,8,20];

> priufer2(priufer);
```



Ilgiausių takų paieškos algoritmas

```
> restart;
> with(GraphTheory):
> with(Statistics):
> with(RandomGraphs):
> with(ListTools):

> it:=proc(G)
> local G1,t,q,s,v,u,n,p0,p1,p2,S,S1;
> global takas,i;
> takas:='takas';
> G1:=G:
> S:=[]:
> for t in Vertices(G) do
> q[t]:={[t]};
> end do:
> while Count(Vertices(G1)) > 2 do
> for t in Vertices(G1) do
> if Count(Neighbors(G1,t))=1 then
> S:=[`union`({S[]},{t})[]];
> end if:
> end do:
> for v in `minus`({Vertices(G1)[]},{S[]}) do
> q[v]:={};
> end do;
> for v in S do
> for u in q[v] do
> n:=Neighbors(G1,v)[]:
> q[n]:=`union`(q[n],{[u[],n]});
> end do;
> end do;
> G1:=DeleteVertex(G1,S);
> S1:=S:S:=[]:
> end do;
> i:=1:
> if Count(Vertices(G1))=1 then
> for p0 from 1 to Count(S1)-1 do
> for p1 from 1 to Count([q[S1[1]][]]) do
> for p2 from 1 to Count([q[S1[p0+1]][]]) do
> takas[i]:=[q[S1[1]][p1][],Vertices(G1)[],Reverse(q[S1[p0+1]][p2])[]];
> i:=i+1
> end do;
> end do;
> q[S1[1]]:={q[S1[1]][],q[S1[p0+1]][]};
> end do;
> else
> for p1 from 1 to Count([q[Vertices(G1)[1]][]]) do
> for p2 from 1 to Count([q[Vertices(G1)[2]][]]) do
> takas[i]:=[q[Vertices(G1)[1]][p1][],Reverse(q[Vertices(G1)[2]][p2])[]];
> i:=i+1
> end do;
> end do;
> end if;
> return();
> end proc;
```

```
> G:=RandomTree(30):DrawGraph(G);
```




```
> it(G);
> for j from 1 to i-1 do Takas[j]=takas[j] end do;
```

$Takas_1$ = [4, 12, 9, 7, 14, 29, 15, 8, 5, 6, 13]
$Takas_2$ = [4, 12, 9, 7, 14, 29, 15, 26, 27, 2, 18]
$Takas_3$ = [21, 12, 9, 7, 14, 29, 15, 8, 5, 6, 13]
$Takas_4$ = [21, 12, 9, 7, 14, 29, 15, 26, 27, 2, 18]

Papildoma procedūra

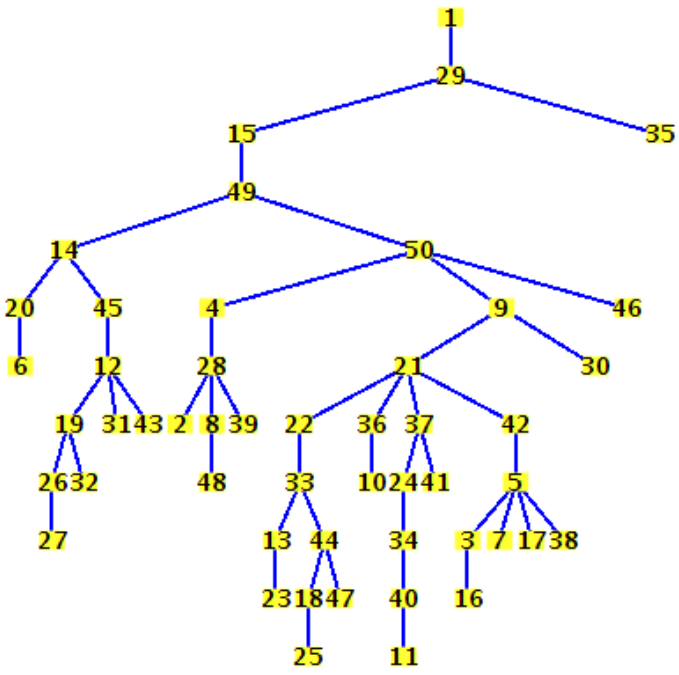
(skirta pasirinkimui, ar medis bus generuojams pagal Priuferio kodą, ar atsitiktinai)

```
> priufer3:=proc()
> if priufer[]=none then priufer1(G) else priufer2(priufer)
> end if;
> end proc;
```

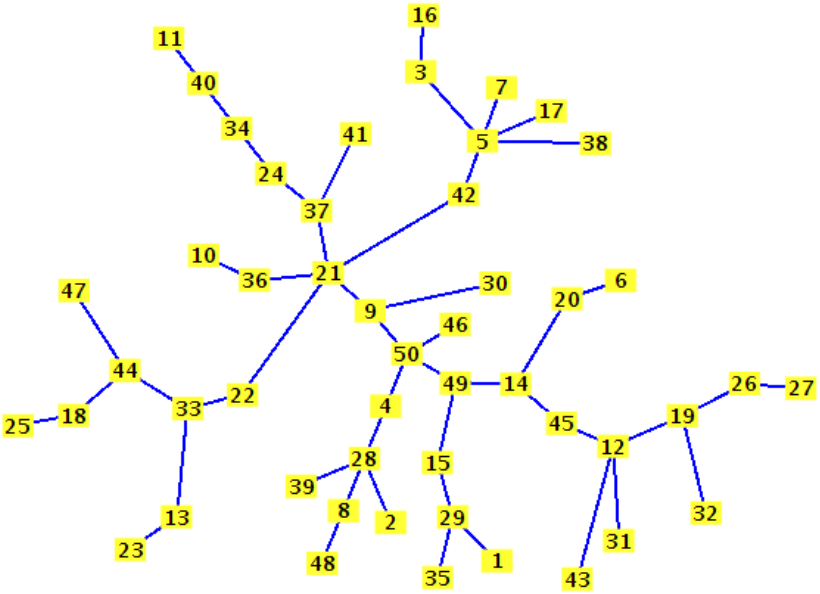
Radialinis medžių vaizdavimo algoritmas

```
> restart;
> with(GraphTheory):
> with(Statistics):
> with(LinearAlgebra):
> with(RandomGraphs):
> with(plots):

> radialinis := proc(G)
> local G1, t, n, m, S, v, pi, u, s, phi, alpha, beta, x, y, U, P, i, j, coor,
V2, plot1; global p;
> G1:=G:
> for t in Vertices(G1) do
> n[t]:=0; m[t]:=0;
> end do:
> S:=[]:p:=1;
> while Count(Vertices(G1)) > 2 do
> for v in Vertices(G1) do if
> Count(Neighbors(G1,v))=1 then
> pi[v]:=Neighbors(G1,v)[];
> S:=[`union`({S[]},{v})[]]
> end if;end do;
> for u in S do
> n[pi[u]]:=n[pi[u]]+n[u]+1
> end do;
> G1:=DeleteVertex(G1,S);
> S:=[]:
> p:=p+1;
> end do;
> if Count(Vertices(G1))=1 then
> s:=Vertices(G1)[]; p:=p-1
> else if
Count(Neighbors(G,Vertices(G1)[1]))=max(Count(Neighbors(G,Vertices(G1)[1])),Coun
t(Neighbors(G,Vertices(G1)[2]))) then
> s:=Vertices(G1)[1]; pi[Vertices(G1)[2]]:=s; else
> s:=Vertices(G1)[2]; pi[Vertices(G1)[1]]:=s;
> end if;end if;
> n[s]:=Count(Vertices(G))-1;
> pi[s]:=0;
> phi[s]:=2*Pi;
> alpha[s]:=0;
> beta:=0;
> x[s]:=0:
> y[s]:=0:
> U[0]:=s;
> P:=[];
> i:=0;
> while i<p do
> for j in U[i] do
> for t in `minus`({Neighbors(G,j)[]},{pi[j]})[] do
> phi[t]:=(n[t]+1)/n[pi[t]]*phi[pi[t]];
> alpha[t]:=alpha[pi[t]]-phi[pi[t]]/2+phi[t]/2+beta;
> beta:=beta+phi[t];
> x[t]:=(i+1)*sin(alpha[t]);
> y[t]:=(i+1)*cos(alpha[t]);
```

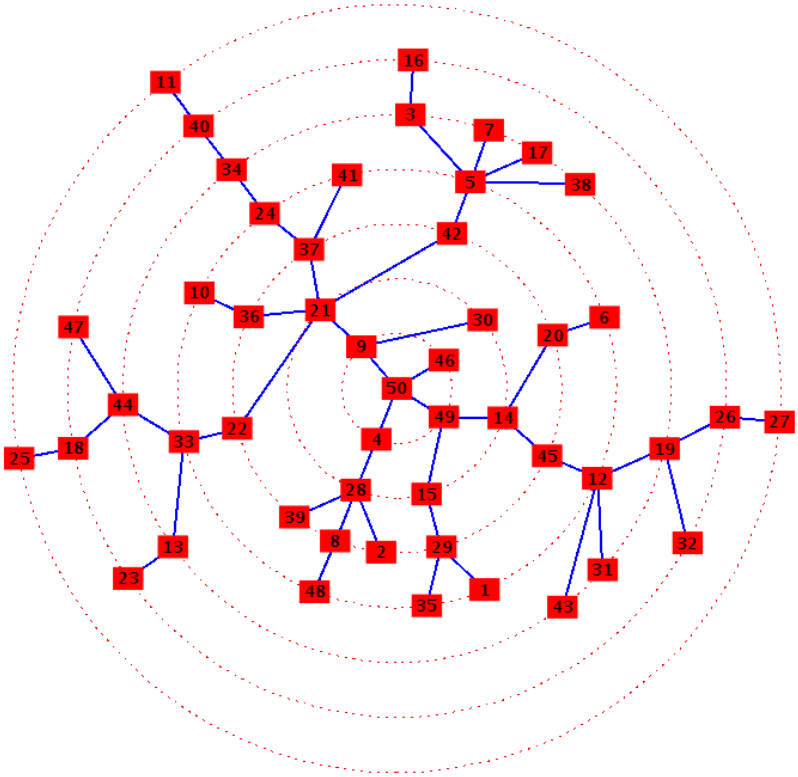
```
> end do;
> P:=[P[],`minus`({Neighbors(G,j)[]},{pi[j]})[]];
> beta:=0;
> end do;
> i:=i+1;
> U[i]:=P;
> P:=[];
> end do;
> for i from 1 to Count(Vertices(G)) do
> coor[i]:=[evalf(x[i],5),evalf(y[i],5)]
> end do;
> V2:=[seq(coor[i],i=1..Count(Vertices(G)))];
> SetVertexPositions(G,V2);
> plot1:=DrawGraph(G);
> return(DrawGraph(G));
> end proc;

> radialinis2 := proc(p,plot1)
> local i,plot2;
> plot2:=plot({seq([i*sin,i*cos,-Pi..Pi],i=1..p)},color=red,axes=none,
numpoints=100,linestyle=dot);
> HighlightVertex(G,Vertices(G),red);
> display([DrawGraph(G),plot2]);
> end proc;

> G:=RandomTree(50);

> DrawGraph(G);K:=DrawGraph(G):
```



```
> radialinis(G);
```




```
> radialinis2(p,plot1);
```

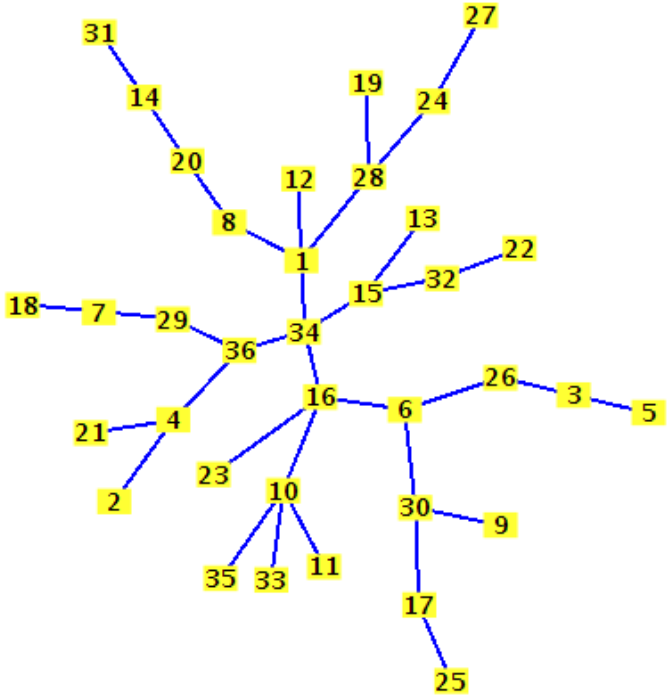


Apibendrintas radialinis medžių vaizdavimo algoritmas

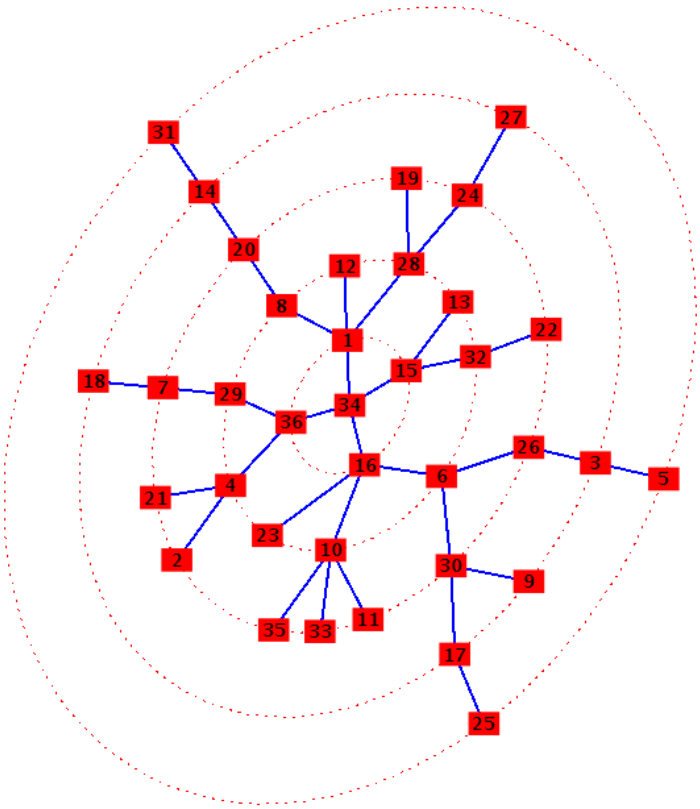
```
> restart;
> with(GraphTheory):
> with(Statistics):
> with(LinearAlgebra):
> with(RandomGraphs):
> with(plots):

> radialinis := proc(G)
> local vaik, G1, t, n, m, S, v, pi, u, s, phi, alpha, beta, x, y, U, P, i, j,
coor, V2, plot1; global p;
> G1:=G:
> for t in Vertices(G1) do
> n[t]:=0; m[t]:=0; vaik[t]:=0;
> end do:
> S:=[]:p:=1;
> while Count(Vertices(G1)) > 2 do
> for v in Vertices(G1) do if
> Count(Neighbors(G1,v))=1 then
> pi[v]:=Neighbors(G1,v)[];
> S:=[`union`({S[]},{v})[]]
> end if;end do;
> for u in S do
> n[pi[u]]:=n[pi[u]]+n[u]+1;
> vaik[pi[u]]:=vaik[pi[u]]+1;
> end do;
> G1:=DeleteVertex(G1,S);
> S:=[]:
> p:=p+1;
> end do;
> if Count(Vertices(G1))=1 then
> s:=Vertices(G1)[]; p:=p-1
> else if
Count(Neighbors(G,Vertices(G1)[1]))=max(Count(Neighbors(G,Vertices(G1)[1])),Coun
t(Neighbors(G,Vertices(G1)[2]))) then
> s:=Vertices(G1)[1]; pi[Vertices(G1)[2]]:=s; else
> s:=Vertices(G1)[2]; pi[Vertices(G1)[1]]:=s;
> end if;end if;
> n[s]:=Count(Vertices(G))-1;
> pi[s]:=0;
> phi[s]:=beta1;
> alpha[s]:=0;
> beta:=0;
> x[s]:=0:
> y[s]:=0:
> U[0]:=s;
> P:=[];
> i:=0;
> while i<p do
> for j in U[i] do
> for t in `minus`({Neighbors(G,j)[]},{pi[j]})[] do
> phi[t]:=(n[t]*svoris+1)/(vaik[pi[t]]+(n[pi[t]]-
vaik[pi[t]])*svoris)*phi[pi[t]];
> alpha[t]:=alpha[pi[t]]-phi[pi[t]]/2+phi[t]/2+beta;
> beta:=beta+phi[t];
```

```
> x[t]:=param*(i+1)^koef*sin(alpha[t])*cos(psi)-
(i+1)^koef*cos(alpha[t])*sin(psi);
>
y[t]:=param*(i+1)^koef*sin(alpha[t])*sin(psi)+(i+1)^koef*cos(alpha[t])*cos(psi);
> end do;
> P:=[P[],`minus`({Neighbors(G,j)[]},{pi[j]})[]];
> beta:=0;
> end do;
> i:=i+1;
> U[i]:=P;
> P:=[];
> end do;
> for i from 1 to Count(Vertices(G)) do
> coor[i]:=[evalf(x[i],5),evalf(y[i],5)]
> end do;
> V2:=[seq(coor[i],i=1..Count(Vertices(G)))];
> SetVertexPositions(G,V2);
> plot1:=DrawGraph(G);
> return(DrawGraph(G));
> end proc;

> radialinis2 := proc(p,plot1)
> local i,plot2;
> plot2:=plot({seq([i^koef*sin*param*cos(psi)-
i^koef*cos*sin(psi),i^koef*sin*param*sin(psi)+i^koef*cos*cos(psi),-
Pi..Pi],i=1..p)},color=red,axes=none,numpoints=100,linestyle=dot);
> HighlightVertex(G,Vertices(G),red):display([DrawGraph(G),plot2]);
> end proc;

> G:=RandomTree(36);DrawGraph(G):K:=DrawGraph(G):

> psi:=Pi*4/3; # kampas, kuriuo pasukama elipsių sistema
> param:=4/3; # elipsės ištempimo parametras
> svoris:=0.3; # pradinės viršūnės palikuonių pasiskirstymo svoris,
priklausantis intervalui [0,1]
> beta1:=2*Pi; # skritulio išpjovos, kurioje vaizduojamas grafas, kampas
> koef:=1.1; # elipsių retėjimo parametras
> priufer:=[4,3,26,30,10,1,15,7,29,28,4,36,32,16,17,30,6,24,28,1,36,6,
16,14,20,8,1,34,15,34,10,10,16,34]; # Priūferio kodas
```

```
> priufer3();radialinis(G,psi,param,svoris,beta1,koef);
```




```
> radialinis2(p,plot1);
```

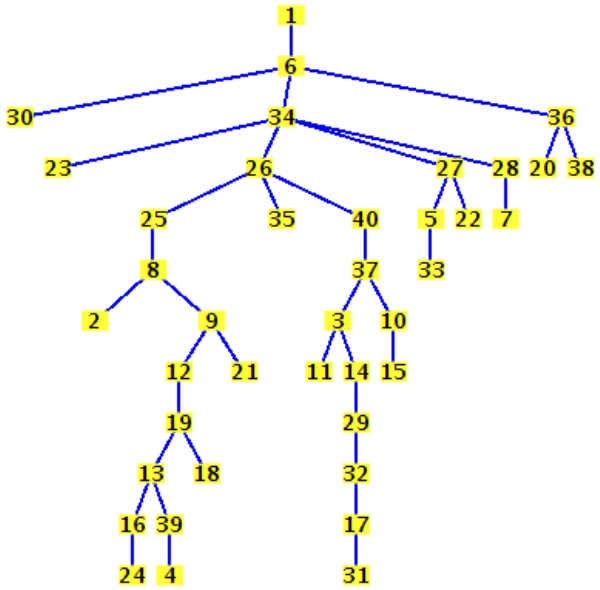


Laurų vainiko algoritmas

```
> restart;
> with(GraphTheory):
> with(Statistics):
> with(LinearAlgebra):
> with(RandomGraphs):
> with(ListTools):
> with(plots):

> lauruvainikas:=proc(G,tk)
> local G1, t, maxrange, S, v, pi, u, G2, tile, phi, psi, x, y, U, P, i, j,
coor, V2, n1, vaik1, n, vaik;
> global plot1, takas, p;
> it(G);
> G1:=G;
> for t in Vertices(G1) do
> n[t]:=0; maxrange[t]:=0; vaik[t]:=0; n1[t]:=0; vaik1[t]:=0;
> end do:
> S:=[]:p:=1;
> while Count(Vertices(G1)) > Count(takas[tk]) do
> for v in `minus`({Vertices(G1)[]},{takas[tk][]}) do if
> Count(Neighbors(G1,v))=1 then
> pi[v]:=Neighbors(G1,v)[];
> S:=[`union`({S[]},{v})[]]
> end if;end do;
> for u in S do
> n[pi[u]]:=n[pi[u]]+n[u]+1;
> n1[pi[u]]:=n[pi[u]];
> vaik[pi[u]]:=vaik[pi[u]]+1;
> vaik1[pi[u]]:=vaik[pi[u]];
> maxrange[pi[u]]:=maxrange[u]+1
> end do;
> G1:=DeleteVertex(G1,S);
> S:=[]:p:=p+1;
> end do;
> for t in takas[tk] do
> vaik1[t]:=0;
> end do;
> DeleteEdge(G, Trail(takas[tk][]));
> G1:=G;
> G2:=G;
> tile:=0;
> for t in takas[tk] do
> phi[t]:=(n[t]*svoris2+1)*(2*Pi-delta)/(Count(takas[tk])+(Count(Vertices(G))-
Count(takas[tk]))*svoris2);
> psi[t]:=delta/2+tile+phi[t]/2;
> x[t]:=param*main^koef*sin(psi[t])*cos(alpha)-main^koef*cos(psi[t])*sin(alpha);
> y[t]:=param*main^koef*sin(psi[t])*sin(alpha)+main^koef*cos(psi[t])*cos(alpha);
> for v in `minus`({Neighbors(G,t)[]},{takas[tk][]}) do if
> maxrange[v]<inside then
> G1:=DeleteVertex(G1,[v]);
> vaik1[pi[v]]:=vaik1[pi[v]]+1;
> else
> n1[pi[v]]:=n1[pi[v]]-n[v]-1;
> G2:=DeleteVertex(G2,[v]);
```

```
> end if;
> end do;
> tile:=tile+phi[t];
> end do;
> tile:=0;
> U[0]:=takas[tk];
> P:=[];
> i:=0;
> while i<inside do
> for j in U[i] do
> for t in `minus`({Neighbors(G2,j)[]},{pi[j]})[] do
> phi[t]:=(n1[t]*svoris+1)/(vaik1[pi[t]]+(n1[pi[t]]-
vaik1[pi[t]])*svoris)*phi[pi[t]];
> psi[t]:=psi[pi[t]]-phi[pi[t]]/2+phi[t]/2+tile;
> tile:=tile+phi[t];
> x[t]:=param*(main-i-1)^koef*sin(psi[t])*cos(alpha)-(main-i-
1)^koef*cos(psi[t])*sin(alpha);
> y[t]:=param*(main-i-1)^koef*sin(psi[t])*sin(alpha)+(main-i-
1)^koef*cos(psi[t])*cos(alpha);
> end do;
> P:=[P[],`minus`({Neighbors(G2,j)[]},{pi[j]})[]];
> tile:=0;
> end do;
> i:=i+1;
> U[i]:=P;
> P:=[];
> end do;
> U[0]:=takas[tk];
> P:=[];
> i:=0;
> for t in takas[tk] do
> n1[t]:=n[t]-n1[t];
> vaik1[t]:=vaik[t]-vaik1[t];
> end do;
> while i<p do
> for j in U[i] do
> for t in `minus`({Neighbors(G1,j)[]},{pi[j]})[] do
> phi[t]:=(n1[t]*svoris+1)/(vaik1[pi[t]]+(n1[pi[t]]-
vaik1[pi[t]])*svoris)*phi[pi[t]];
> psi[t]:=psi[pi[t]]-phi[pi[t]]/2+phi[t]/2+tile;
> tile:=tile+phi[t];
> x[t]:=param*(main+i+1)^koef*sin(psi[t])*cos(alpha)-
(main+i+1)^koef*cos(psi[t])*sin(alpha);
> y[t]:=param*(main+i+1)^koef*sin(psi[t])*sin(alpha)+(main+i+1)^koef*cos(psi[t])
*cos(alpha);
> end do;
> P:=[P[],`minus`({Neighbors(G1,j)[]},{pi[j]})[]];
> tile:=0;
> end do;
> i:=i+1;
> U[i]:=P;
> P:=[];
> end do;
> for i from 1 to Count(Vertices(G)) do
> coor[i]:=[evalf(x[i],5),evalf(y[i],5)]
> end do;
> AddEdge(G, Trail(takas[tk][]));
```

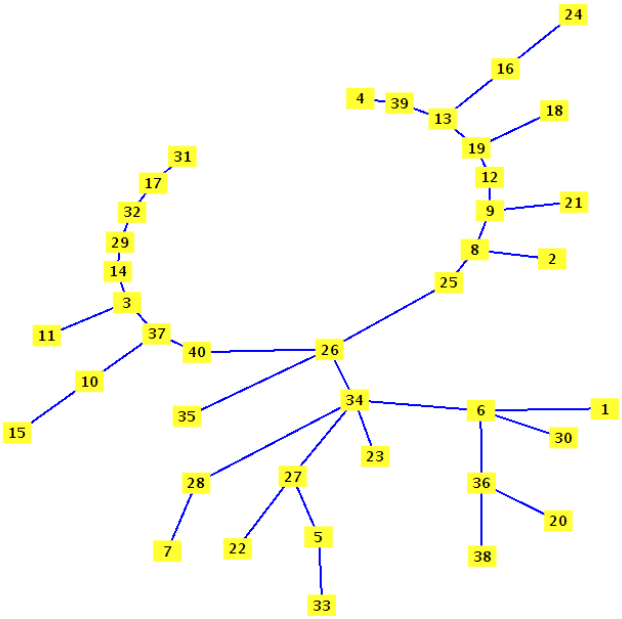
```
> V2:=[seq(coor[i],i=1..Count(Vertices(G)))];
> SetVertexPositions(G,V2);
> plot1:=DrawGraph(G);
> RETURN(DrawGraph(G));
> end proc;

> lauras := proc(plot1)
> local i,plot2;
> plot2:=plot({seq([i^koef*sin*param*cos(alpha)-i^koef*cos*sin(alpha),
i^koef*sin*param*sin(alpha)+i^koef*cos*cos(alpha),-Pi..Pi],i=main-
inside..main+p-1)},color=red,axes=none,numpoints=100,linestyle=dot);
> HighlightVertex(G,Vertices(G),red);
> display([DrawGraph(G),plot2]);
> end proc;

> G:=RandomTree(40);DrawGraph(G);K:=DrawGraph(G):
```




```
> tk:=1; # ilgiausio tako nr.
> main:=3; # elipsės, kurioje vaizduojamas ilgiausias takas, nr.
> koef:=1.3; # elipsių retėjimo koeficientas
> inside:=0; # max palikuonių, vaizduojamų viduje
> delta:=Pi/4; # kampas tarp ilgiausio tako krašte esančių viršūnių
> alpha:=Pi/10; # kampas, kurio pasukama elipsių sistema
> param:=11/7; # elipsės ištempimo parametras
> svoris:=0.8; # ilgiausio tako palikuonių pasiskirstymo svoris, priklausantis
intervalui [0,1]
> svoris2:=0.3; # ilgiausio tako viršūnių pasiskirstymo svoris, priklausantis
intervalui [0,1]
> priufer:=[none]; # Priuferio kodas
```

```
> priufer3();lauruvainikas(G,tk,main,inside,delta,alpha,param,svoris,
svoris2); # vykdomas apibendrintas laurų vainiko algoritmas
```

[6,8,39,28,3,10,37,19,36,9,27,34,16,13,34,6,17,32,29,14,3,37,5,27,34,26,40,36,6,34,26,13,19,12,9,8,25,26]

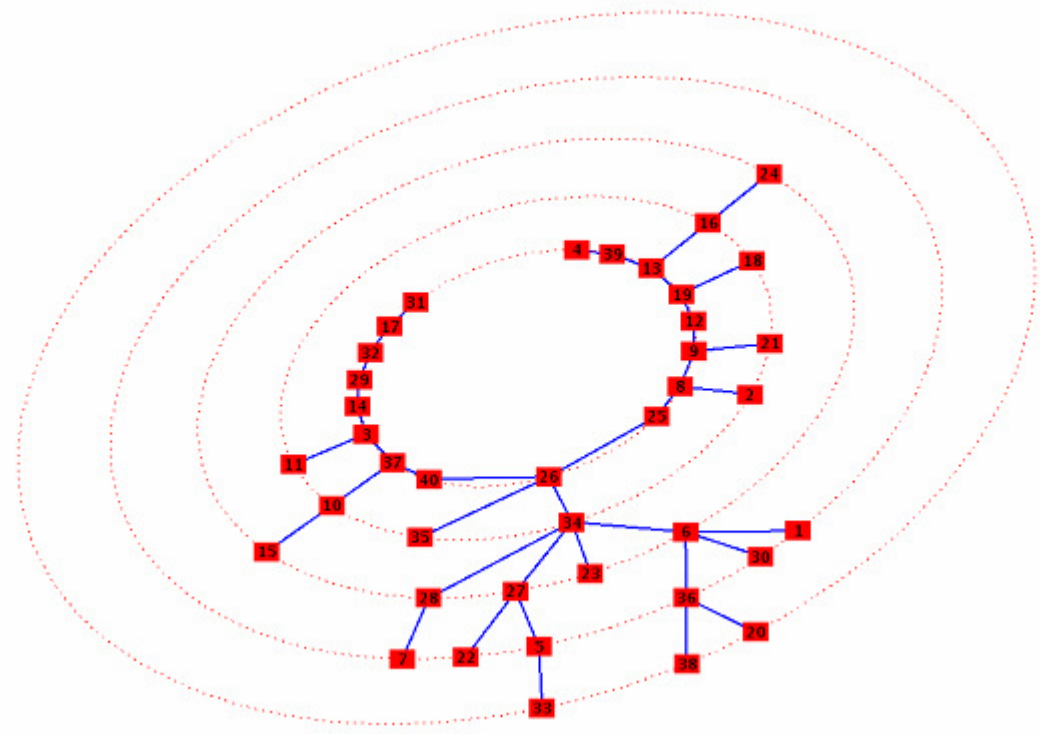



```
> for i from 1 to Count([convert(takas,set)[]]) do Takas[i]=takas[i] end do; #
išvedami visi ilgiausi takai
```

$Takas_1$ = [4,39,13,19,12,9,8,25,26,40,37,3,14,29,32,17,31]

$Takas_2$ = [24,16,13,19,12,9,8,25,26,40,37,3,14,29,32,17,31]

```
> lauras(plot1); # vaizduojama elipsių sistema
```

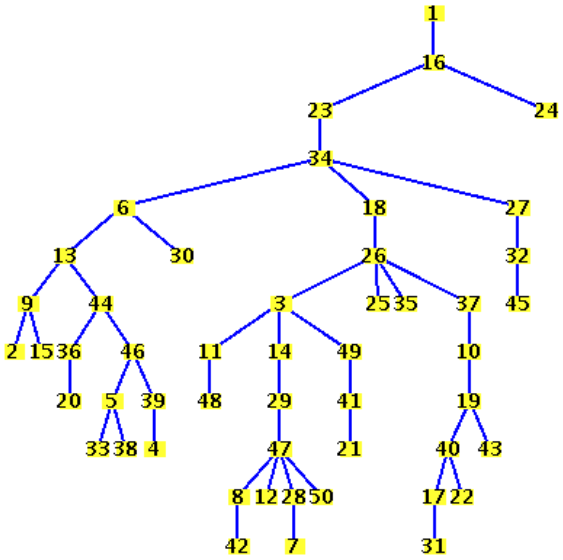


Apibendrintas radialinis medžių vaizdavimas trimatėje erdvėje

```
> restart;
> with(GraphTheory):
> with(Statistics):
> with(LinearAlgebra):
> with(RandomGraphs):
> with(plots):
> with(plottools):
> with(VectorCalculus):

> radialinis := proc(G)
> local G1, t, n, vaik, m, S, v, pi, u, s, beta, U, P, i, j, B, A, A1, B1, C1,
coor, x, y, z, phi, alpha; global p, plot1;
> G1:=G:
> for t in Vertices(G1) do
> n[t]:=0; m[t]:=0; vaik[t]:=0;
> end do:
> S:=[]:p:=1;
> while Count(Vertices(G1)) > 2 do
> for v in Vertices(G1) do if
> Count(Neighbors(G1,v))=1 then
> pi[v]:=Neighbors(G1,v)[];
> S:=[`union`({S[]},{v})[]]
> end if;end do;
> for u in S do
> n[pi[u]]:=n[pi[u]]+n[u]+1;
> vaik[pi[u]]:=vaik[pi[u]]+1;
> end do;
> G1:=DeleteVertex(G1,S);
> S:=[]:
> p:=p+1;
> end do;
> if Count(Vertices(G1))=1 then
> s:=Vertices(G1)[]; p:=p-1
> else if
Count(Neighbors(G,Vertices(G1)[1]))=max(Count(Neighbors(G,Vertices(G1)[1])),Coun
t(Neighbors(G,Vertices(G1)[2]))) then
> s:=Vertices(G1)[1]; pi[Vertices(G1)[2]]:=s; else
> s:=Vertices(G1)[2]; pi[Vertices(G1)[1]]:=s;
> end if;end if;
> n[s]:=Count(Vertices(G))-1;
> pi[s]:=0;
> phi[s]:=beta1;
> alpha[s]:=0;
> beta:=0;
> x[s]:=0:
> y[s]:=0:
> z[s]:=0:
> U[0]:=s;
> P:=[];
> i:=0;
> while i<p do
> for j in U[i] do
> for t in `minus`({Neighbors(G,j)[]},{pi[j]})[] do
```

```
> phi[t]:=(n[t]*svoris+1)/(vaik[pi[t]]+(n[pi[t]]-
vaik[pi[t]])*svoris)*phi[pi[t]];
> alpha[t]:=alpha[pi[t]]-phi[pi[t]]/2+phi[t]/2+beta;
> beta:=beta+phi[t];
> x[t]:=tank*param*(i+1)^koef*cos(alpha[t])/cosh(cot(zeta)*(alpha[t]-lambda));
> y[t]:=tank*(i+1)^koef*sin(alpha[t])/cosh(cot(zeta)*(alpha[t]-lambda));
> z[t]:=(i+1)^koef*tanh(cot(zeta)*(alpha[t]-lambda));
> end do;
> P:=[P[],`minus`({Neighbors(G,j)[]},{pi[j]})[]];
> beta:=0;
> end do;
> i:=i+1;
> U[i]:=P;
> P:=[];
> end do;
> i:='i':
> for i from 1 to Count(Vertices(G)) do
> coor[i]:=[evalf(x[i],5),evalf(y[i],5),evalf(z[i],5)];
> B[i]:=point(coor[i])
> end do;
> for i from 1 to Count(Vertices(G))-1 do
> A[i]:=curve([coor[Edges(G)[i][1]],coor[Edges(G)[i][2]]]);
> end do;
> A1:=display({seq(A[i],i=1..Count(Vertices(G))-1)},color = blue,thickness = 3);
> B1:=display({seq(B[i],i=1..Count(Vertices(G)))}, color = blue,
symbolsize=pointsize,color = yellow);
> C1:=textplot3d([seq([evalf(x[i],5)+up,evalf(y[i],5)+up,evalf(z[i],5)+up,i],
i=1..Count(Vertices(G)))],font = [TIMES, ROMAN, fontsize]);
> plot1:=display(A1,B1,C1,scaling=constrained);
> return(display([A1,B1,C1],scaling=constrained));
> end proc;

> radialinis2 := proc(p,plot1)
> local i,plot2,t,Sp;for i from 1 to p do
> Sp[i]:=SpaceCurve(`<,>`(tank*param*i^koef*cos(t)/cosh(cot(zeta)*(t-lambda)),
tank*i^koef*sin(t)/cosh(cot(zeta)*(t-lambda)), i^koef*tanh(cot(zeta)*(t-
lambda))), t = -beta1/2 .. beta1/2,thickness=1,color=red,numpoints=1000)
> end do;
> plot2:=seq(Sp[i],i=1..p);
> return(display([plot1,plot2],scaling=constrained));
> end proc;
```

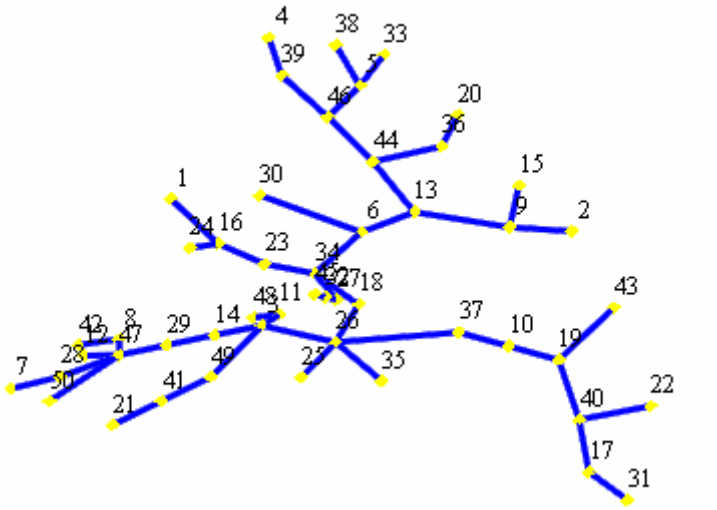
```
> G:=RandomTree(50);DrawGraph(G);K:=DrawGraph(G):
```



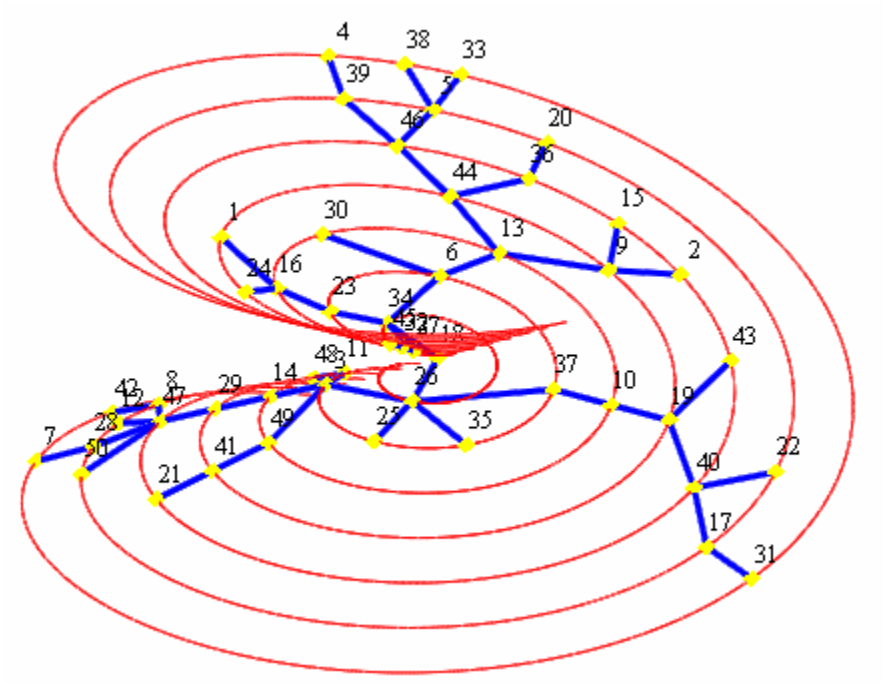

```
> up:=0.2; # teksto atstumas iki taško
> fontsize:=10; # šrifto dydis
> pointsize:=12; # taško dydis
> param:=5/5; # elipsės ištempimo parametras
> svoris:=0.1; # pradinės viršūnės palikuonių pasiskirstymo svoris,
priklausantis intervalui [0,1]
> beta1:=3*Pi; # skritulio išpjovos, kurioje vaizduojamas grafas, kampas
> koef:=1; # elipsių retėjimo / tankėjimo koeficientas
> tank:=2/3; # tankumo parametras
> zeta:=Pi/2-0.1; # spiralės tankumo parametras
> lambda:=0; # spiralės padėties sferoje parametras
> priufer:=[none]; # Priuferio kodas
```

```
> priufer3();radialinis(G,zeta,param,svoris,tank,beta1,koef,lambda);
```

[16,9,39,28,47,9,13,36,41,40,16,23,34,26,47,6,17,40,5,26,44,5,46,46,19,49,8,47,19,10,37,26,32,27,34,44,13,6,34,18,26,3,11,3,3,14,29,47]



```
> radialinis2(p,plot1);
```

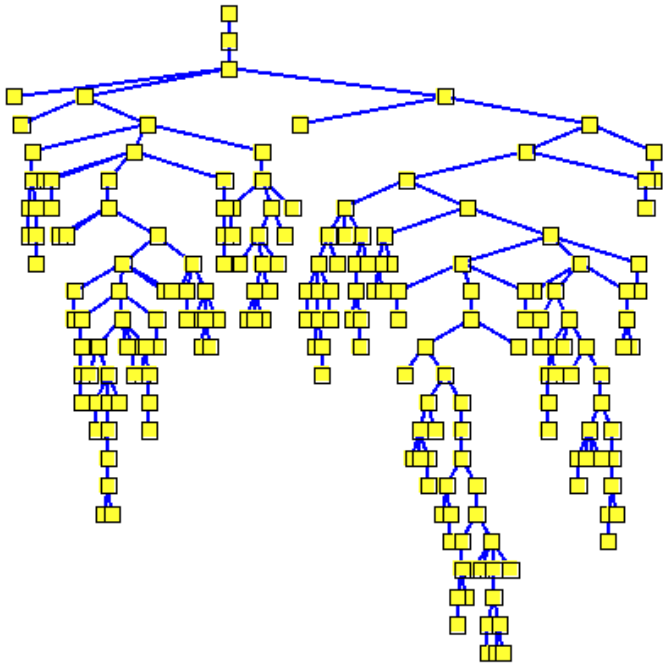


Laurų vainiko algoritmas trimatei erdvei

```
> restart;
> with(GraphTheory):
> with(Statistics):
> with(LinearAlgebra):
> with(RandomGraphs):
> with(ListTools):
> with(plottools):
> with(VectorCalculus):
> with(plots):

> lauruvainikas:=proc(G,tk)
> local G1, t, maxrange, S, v, pi, u, G2, tile, phi, psi, x, y, U, P, i, j, V2,
n1, vaik1, n, vaik, z, B, A, A1, B1, C1;
> global plot1, takas, p, coor;
> it(G);
> G1:=G;
> for t in Vertices(G1) do
> n[t]:=0; maxrange[t]:=0; vaik[t]:=0; n1[t]:=0; vaik1[t]:=0;
> end do:
> S:=[]:p:=1;
> while Count(Vertices(G1)) > Count(takas[tk]) do
> for v in `minus`({Vertices(G1)[]},{takas[tk][]}) do if
> Count(Neighbors(G1,v))=1 then
> pi[v]:=Neighbors(G1,v)[];
> S:=[`union`({S[]},{v})[]]
> end if;end do;
> for u in S do
> n[pi[u]]:=n[pi[u]]+n[u]+1;
> n1[pi[u]]:=n[pi[u]];
> vaik[pi[u]]:=vaik[pi[u]]+1;
> vaik1[pi[u]]:=vaik[pi[u]];
> maxrange[pi[u]]:=maxrange[u]+1
> end do;
> G1:=DeleteVertex(G1,S);
> S:=[]:p:=p+1;
> end do;
> for t in takas[tk] do
> vaik1[t]:=0;
> end do;
> DeleteEdge(G, Trail(takas[tk][]));
> G1:=G;
> G2:=G;
> tile:=0;
> for t in takas[tk] do
> phi[t]:=(n[t]*svoris2+1)*(2*Pi-delta)/(Count(takas[tk])+(Count(Vertices(G))-
Count(takas[tk]))*svoris2);
> psi[t]:=delta/2+tile+phi[t]/2;
> x[t]:=param*main^koef*sin(psi[t]);
> y[t]:=main^koef*cos(psi[t]);
> z[t]:=psi[t]*tank;
> for v in `minus`({Neighbors(G,t)[]},{takas[tk][]}) do if
> maxrange[v]<inside then
> G1:=DeleteVertex(G1,[v]);
> vaik1[pi[v]]:=vaik1[pi[v]]+1;
```

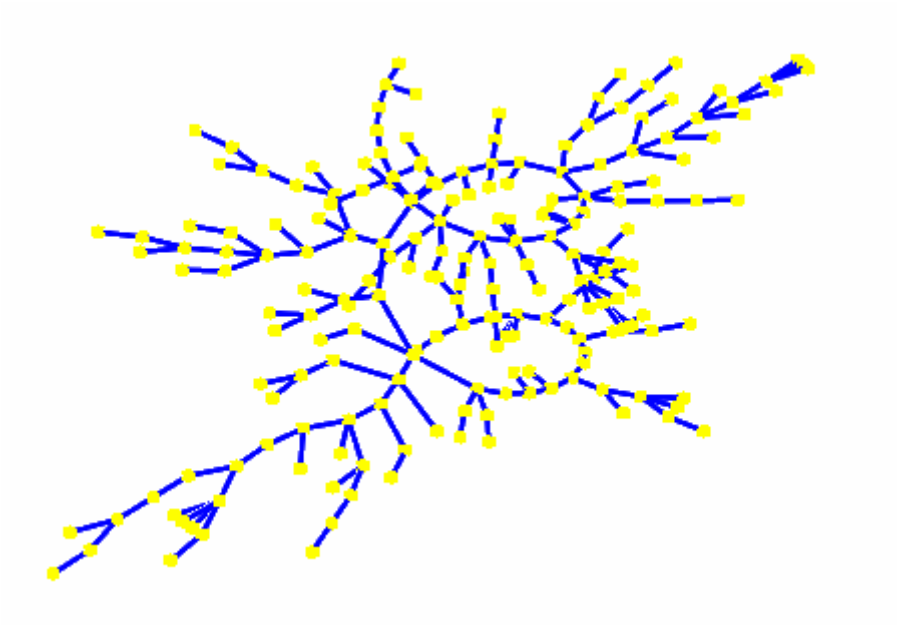
```
> else
> n1[pi[v]]:=n1[pi[v]]-n[v]-1;
> G2:=DeleteVertex(G2,[v]);
> end if;
> end do;
> tile:=tile+phi[t];
> end do;
> tile:=0;
> U[0]:=takas[tk];
> P:=[];
> i:=0;
> while i<inside do
> for j in U[i] do
> for t in `minus`({Neighbors(G2,j)[]},{pi[j]})[] do
> phi[t]:=(n1[t]*svoris+1)/(vaik1[pi[t]]+(n1[pi[t]]-
vaik1[pi[t]])*svoris)*phi[pi[t]];
> psi[t]:=psi[pi[t]]-phi[pi[t]]/2+phi[t]/2+tile;
> tile:=tile+phi[t];
> x[t]:=param*(main-i-1)^koef*sin(psi[t]);
> y[t]:=(main-i-1)^koef*cos(psi[t]);
> z[t]:=psi[t]*tank;
> end do;
> P:=[P[],`minus`({Neighbors(G2,j)[]},{pi[j]})[]];
> tile:=0;
> end do;
> i:=i+1;
> U[i]:=P;
> P:=[];
> end do;
> U[0]:=takas[tk];
> P:=[];
> i:=0;
> for t in takas[tk] do
> n1[t]:=n[t]-n1[t];
> vaik1[t]:=vaik[t]-vaik1[t];
> end do;
> while i<p do
> for j in U[i] do
> for t in `minus`({Neighbors(G1,j)[]},{pi[j]})[] do
> phi[t]:=(n1[t]*svoris+1)/(vaik1[pi[t]]+(n1[pi[t]]-
vaik1[pi[t]])*svoris)*phi[pi[t]];
> psi[t]:=psi[pi[t]]-phi[pi[t]]/2+phi[t]/2+tile;
> tile:=tile+phi[t];
> x[t]:=param*(main+i+1)^koef*sin(psi[t]);
> y[t]:=(main+i+1)^koef*cos(psi[t]);
> z[t]:=psi[t]*tank;
> end do;
> P:=[P[],`minus`({Neighbors(G1,j)[]},{pi[j]})[]];
> tile:=0;
> end do;
> i:=i+1;
> U[i]:=P;
> P:=[];
> end do;
> for i from 1 to Count(Vertices(G)) do
> coor[i]:=[evalf(x[i],5),evalf(y[i],5),evalf(z[i],5)];
> B[i]:=point(coor[i])
```

```
> end do;
> AddEdge(G, Trail(takas[tk][]));
> for i from 1 to Count(Vertices(G))-1 do
> A[i]:=curve([coor[Edges(G)[i][1]],coor[Edges(G)[i][2]]]);
> end do;
> A1:=display({seq(A[i],i=1..Count(Vertices(G))-1)},color = blue,thickness = 3);
> B1:=display({seq(B[i],i=1..Count(Vertices(G)))}, color = blue,
symbolsize=pointsize,color = yellow);
> C1:=textplot3d([seq([evalf(x[i],5)+up,evalf(y[i],5)+up,evalf(z[i],5)
+up,i],i=1..Count(Vertices(G)))],font = [TIMES, ROMAN, fontsize]);
> plot1:=display(A1,B1,C1,scaling=constrained);
> end proc;

> lauras := proc(p,plot1)
> local i,plot2,t,Sp;
> for i from main-inside to p+main-1 do
> Sp[i]:=SpaceCurve(`<,>`(param*i^koef*sin(t), i^koef*cos(t), tank*t), t =
delta/2..2*Pi-delta/2,thickness=1,color=red,numpoints=1000)
> end do;
> plot2:=seq(Sp[i],i=main-inside..p+main-1);
> return(display([plot1,plot2],scaling=constrained));
> end proc;

> G:=RandomTree(200);DrawGraph(G);K:=DrawGraph(G):
```



```
> tk:=1; # ilgiausio tako nr.
> up:=0.2; # teksto atstumas iki taško
> fontsize:=1; # šrifto dydis
> pointsize:=10; # taško dydis
> main:=4; # elipsės, kurioje vaizduojamas ilgiausias takas, nr.
> koef:=1.3; # elipsių retėjimo koeficientas
> inside:=1; # max palikuonių, vaizduojamų viduje, ilgis
> delta:=-2*Pi; # kampas tarp ilgiausio tako krašte esančių viršūnių
> param:=11/11; # elipsės ištempimo parametras
> svoris:=0.2; # ilgiausio tako palikuonių pasiskirstymo svoris, priklausantis
intervalui [0,1]
> svoris2:=0.1; # ilgiausio tako viršūnių pasiskirstymo svoris, priklausantis
intervalui [0,1]
> tank:=2; # tankumo parametras
> priufer:=[none]; # Priuferio kodas

> priufer3(G);lauruvainikas(G,tk); # vykdomas apibendrintas laurų vainiko
algoritmas
```

[20,84,45,64,175,140,4,3,78,83,79,197,121,11,54,158,90,105,100,66,92,61,177,172,147,142,174,145,84,105,173,49,4,12,73,162,92,196,3,188,183,110,136,44,149,28,14,12,52,198,16,180,36,195,165,21,78,68,100,68,117,163,51,93,90,50,145,13,39,110,32,85,192,152,78,73,66,144,109,10,40,69,125,191,18,13,87,66,34,78,191,73,80,131,72,116,15,101,34,51,44,187,84,28,3,98,150,40,98,144,128,180,36,189,188,56,23,55,85,41,137,131,110,39,107,106,88,121,65,157,96,165,188,15,56,171,100,43,126,113,105,5,42,149,101,197,49,180,18,90,168,195,9,170,197,106,174,127,100,198,101,177,49,116,162,90,131,73,130,130,92,71,105,137,77,187,67,58,107,11,96,17,40,116,171,53,157,51,53,9,109,183,4,172,137,129,144,48]



```
> for i from 1 to Count([convert(takas,set)[]]) do Takas[i]=takas[i] end do; #
išvedami visi ilgiausi takai
```

$Takas_1$ = [47,61,174,127,100,198,77,187,67,58,107,11,96,17,40,116,171,53,9,109,183,4,172,137,105,71,92,130,73,131,90,18,180,128,163,117,68,115]

$Takas_2$ = [47,61,174,127,100,198,77,187,67,58,107,11,96,17,40,116,171,53,9,109,183,4,172,137,105,71,92,130,73,131,90,18,180,128,163,117,68,119]

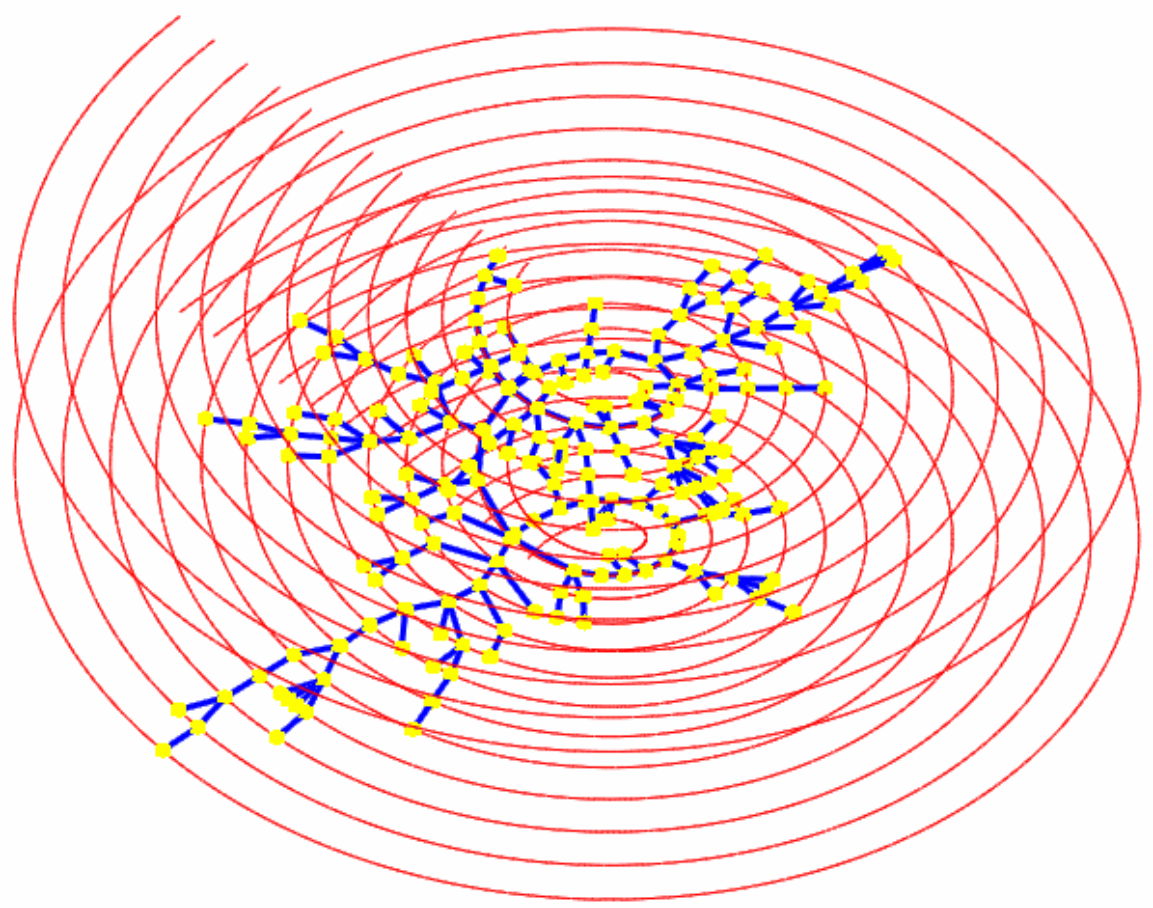
$Takas_3$ = [176,106,174,127,100,198,77,187,67,58,107,11,96,17,40,116,171,53,9,109,183,4,172,137,105,71,92,130,73,131,90,18,180,128,163,117,68,115]

$Takas_4$ = [176,106,174,127,100,198,77,187,67,58,107,11,96,17,40,116,171,53,9,109,183,4,172,137,105,71,92,130,73,131,90,18,180,128,163,117,68,119]

$Takas_5$ = [194,106,174,127,100,198,77,187,67,58,107,11,96,17,40,116,171,53,9,109,183,4,172,137,105,71,92,130,73,131,90,18,180,128,163,117,68,115]

$Takas_6$ = [194,106,174,127,100,198,77,187,67,58,107,11,96,17,40,116,171,53,9,109,183,4,172,137,105,71,92,130,73,131,90,18,180,128,163,117,68,119]

```
> lauras(p, plot1); # vaizduojama spiralių sistema
```

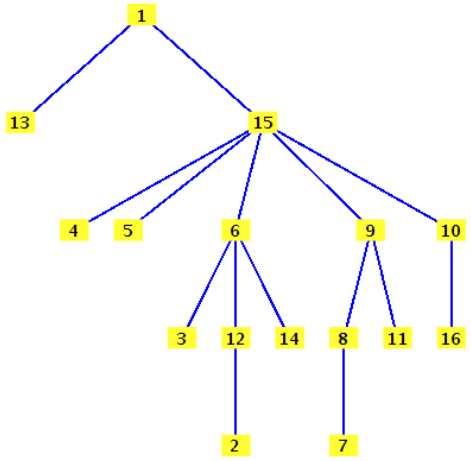


Medžių vizualizacijos algoritmų taikymas

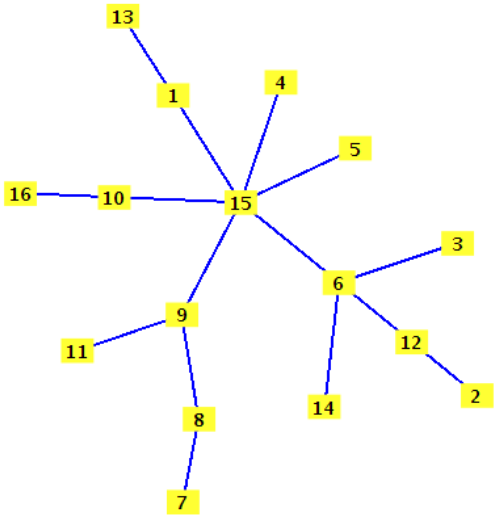
Paieška į plotį plokštumoje

```
> restart;
> with(GraphTheory):
> with(Statistics):
> with(LinearAlgebra):
> with(RandomGraphs):
> with(plots):
> with(queue):

> pplot:=proc(G,s)
> local u, Q, v, i;
> global draw, tt, c, d, pi;
> HighlightEdges(G, Edges(G), color_4);
> HighlightVertex(G,Vertices(G),color_1);
> tt:=1:
> draw[tt]:=DrawGraph(G);
> tt:=tt+1:
> for u in Vertices(DeleteVertex(G,s)) do
> c[u]:=color_1;
> d[u]:=infinity;
> pi[u]:=NIL;
> end do;
> c[s]:=color_2;
> d[s]:=0;
> pi[s]:=NIL;
> Q:=new(s):
> HighlightVertex(G,[s],color_2);
> draw[tt]:=DrawGraph(G);
> tt:=tt+1:
> while length(Q)<>0 do
> u:=front(Q);
> for v in Neighbors(G,u) do
> if c[v]=color_1 then
> c[v]:=color_2;
> d[v]:=d[u]+1;
> pi[v]:=u;
> HighlightEdges(G, {v,u}, color_5);
> HighlightVertex(G,[v], color_2);
> enqueue(Q,v);
> end if;
> if c[v]=color_2 then
> HighlightEdges(G, {v,u}, color_5)
> end if;
> end do;
> dequeue(Q);
> HighlightVertex(G,[u], color_3);
> c[u]:=color_3;
> draw[tt]:=DrawGraph(G);
> tt:=tt+1:
> end do;
> return();
> end proc;
```

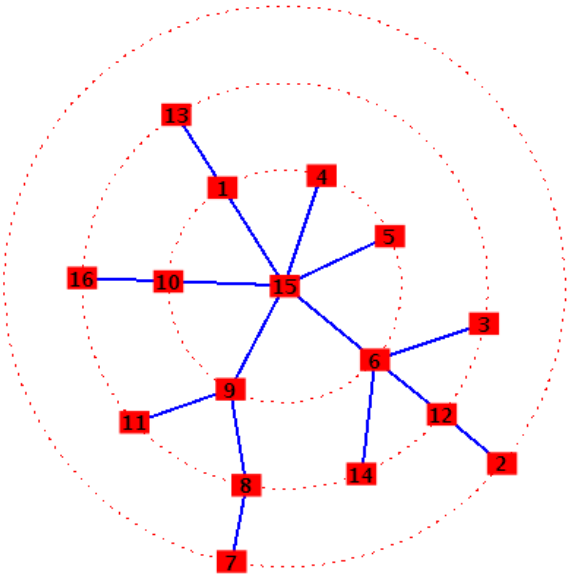
```
> G:=RandomTree(16);DrawGraph(G);K:=DrawGraph(G):
```



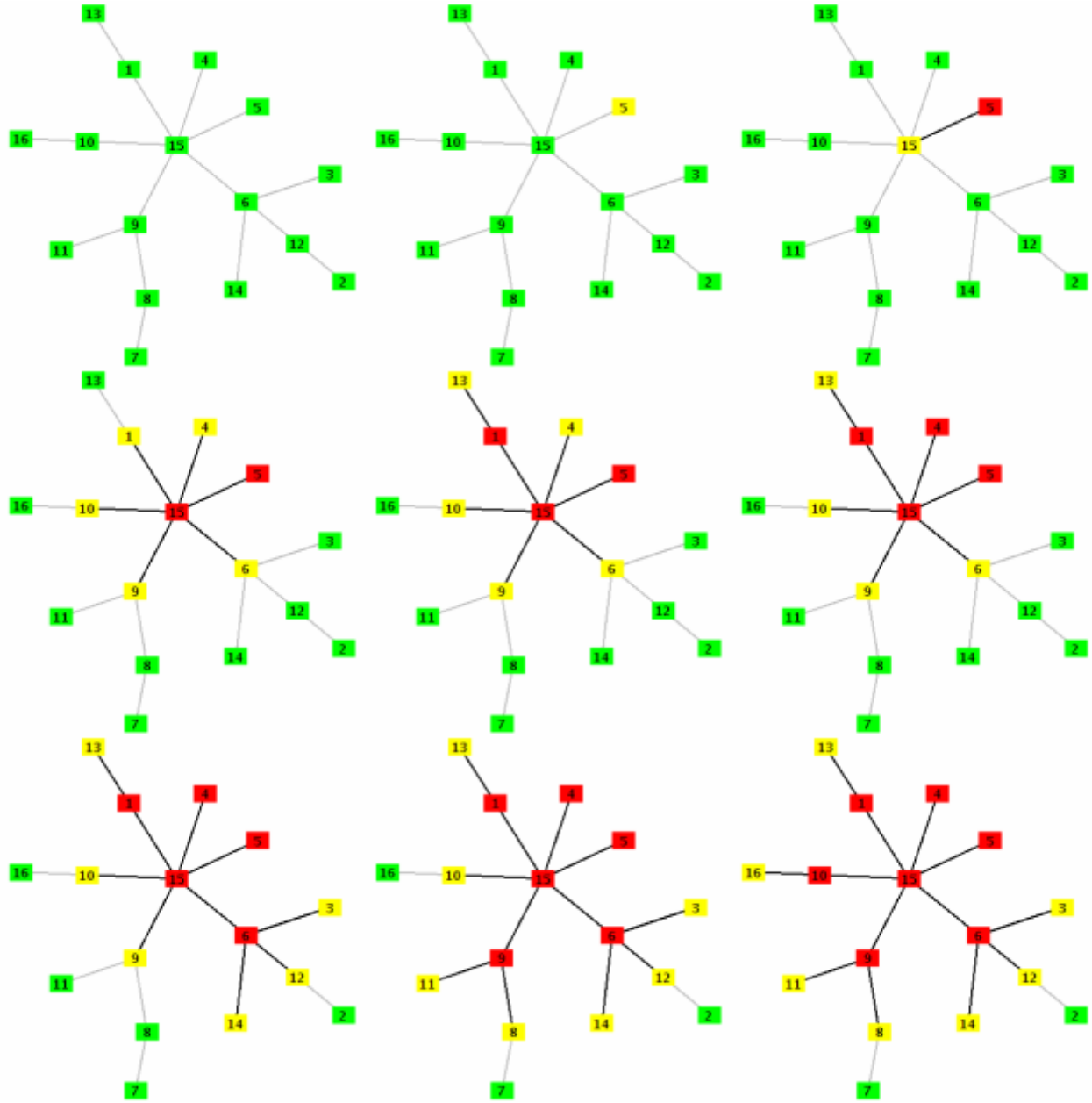

```
> radialinis(G, Pi*4/3, 3/3, 0.2, 2*Pi, 0.8);
```

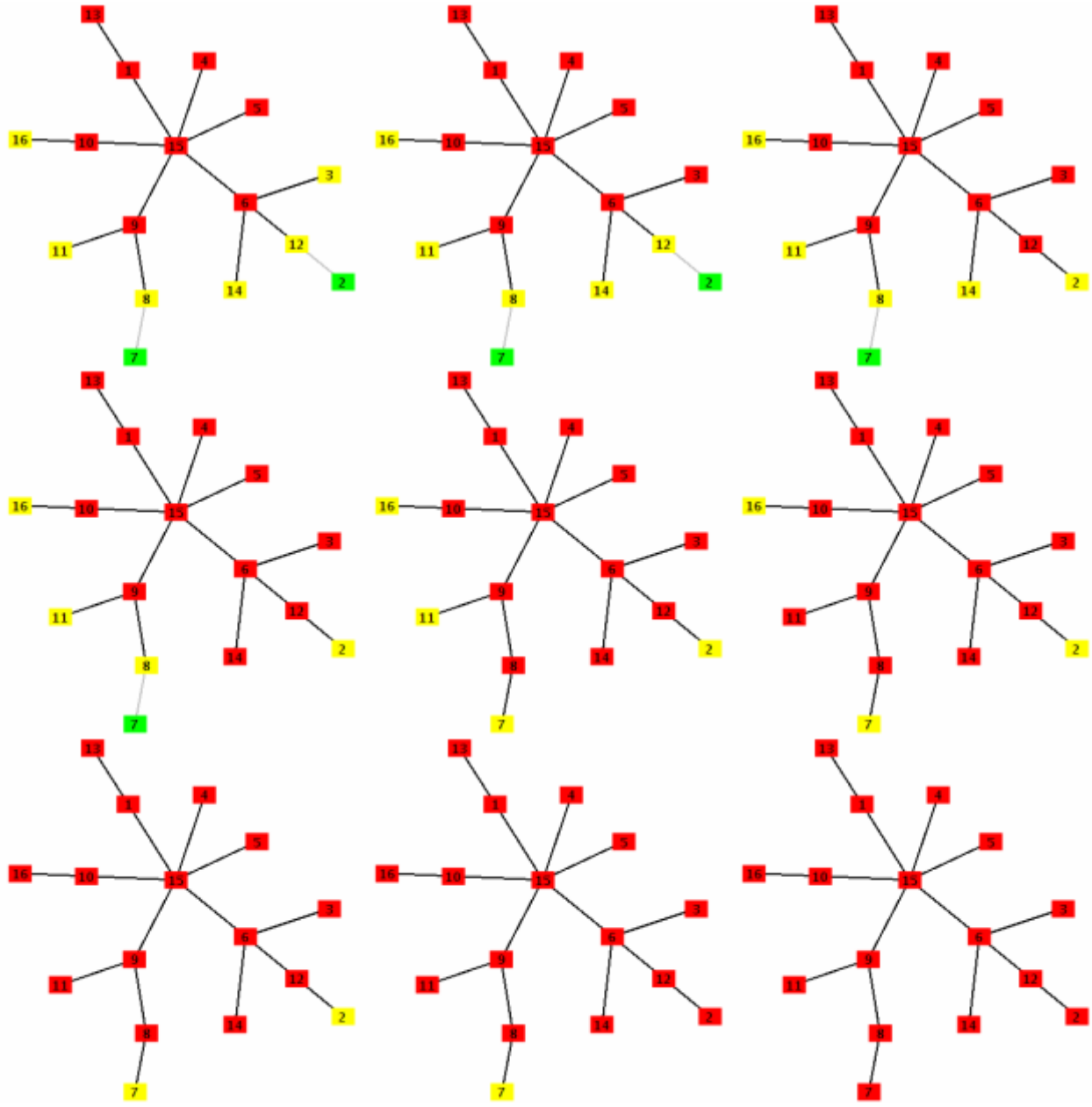



```
> radialinis2(p,plot1);
```



```
> s:=5: # pradinė viršūnė
> color_1:=green; # pradinio (inicializuoto) grafo viršūnių spalva
> color_2:=yellow; # algoritmo vykdymo metu nagrinėjamų viršūnių spalva
> color_3:=red; # algoritmo vykdymo išnagrinėtų (pereitų) viršūnių spalva
> color_4:=grey; # pradinio grafo briaunų spalva
> color_5:=black; # briaunų, incidenčių nagrinėjamai viršūnei, spalva

> pplot(G,s);
> for i from 1 to tt-1 do draw[i] end do;
```

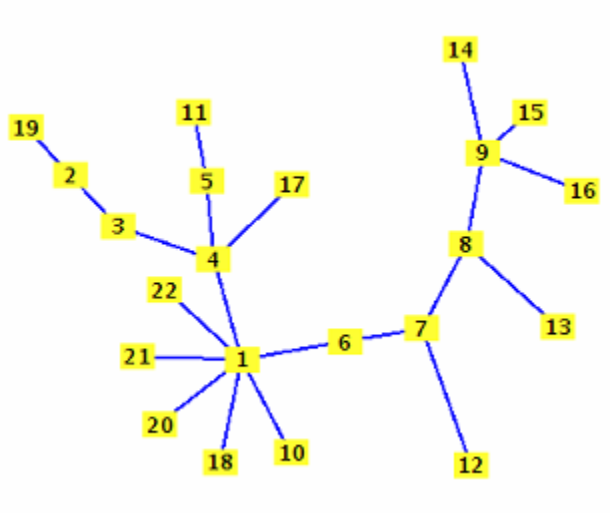



13 1 4 5 16 10 15 3 6 9 11 12 14 2 8 7

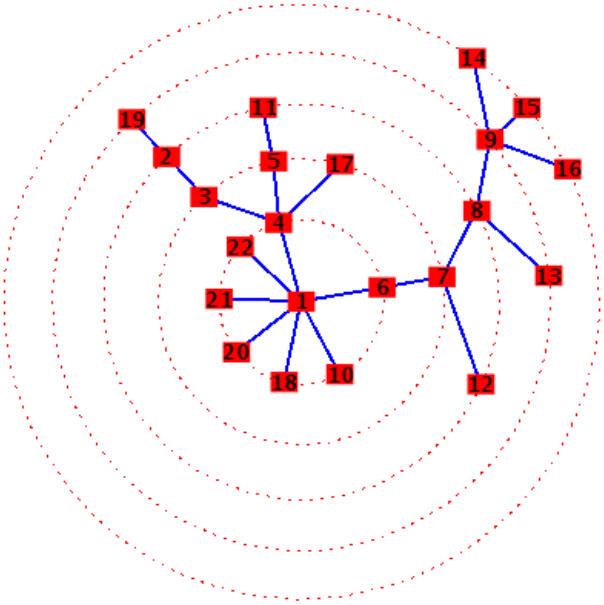
Paieška į gylį plokštumoje

```
> restart;
> with(GraphTheory):
> with(Statistics):
> with(LinearAlgebra):
> with(RandomGraphs):
> with(plots):
> with(queue):

> VISIT:=proc(u)
> local v;
> global c,pi,d,f,tt,draw;
> c[u]:=color_2;
> HighlightVertex(G,[u],color_2);
> tt:=tt+1;
> d[u]:=tt;
> draw[tt]:=DrawGraph(G);
> for v in Neighbors(G,u) do
> HighlightEdges(G, {v,u}, color_5);
> if c[v]=color_1
> then pi[v]:=u;
> VISIT(v)
> end if; end do;
> c[u]:=color_3;
> HighlightVertex(G,[u],color_3);
> tt:=tt+1;
> f[u]:=tt;
> draw[tt]:=DrawGraph(G);
> end proc;

> pgyl:=proc(G)
> local u, i;
> global c, pi, tt, d, f, draw;
> HighlightEdges(G, Edges(G), color_4);
> HighlightVertex(G,Vertices(G),color_1);
> tt:=0:
> draw[tt]:=DrawGraph(G);
> for u in Vertices(G) do
> c[u]:=color_1:
> pi[u]:=NIL:
> d[u]:=0:
> f[u]:=0:
> end do;
> DrawGraph(G);
> for u in Vertices(G)
> do if c[u]=color_1
> then VISIT(u)
> end if;
> end do;
> return();
> end proc;
```

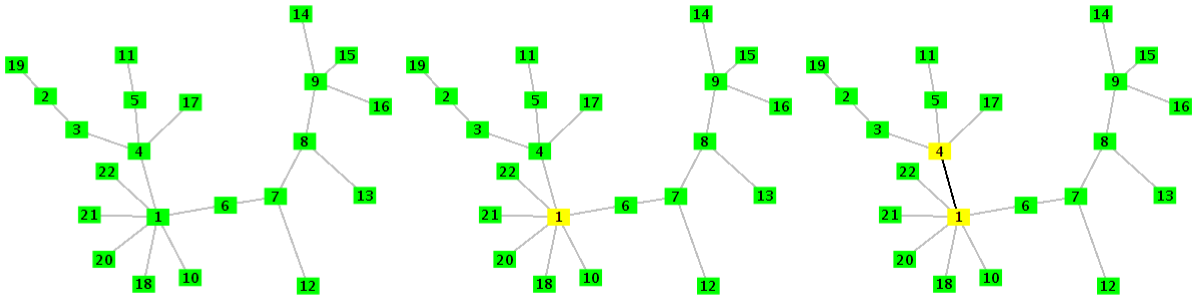
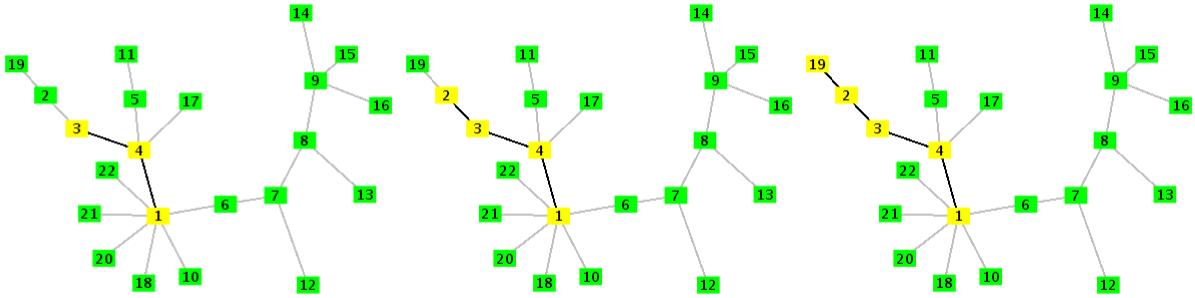
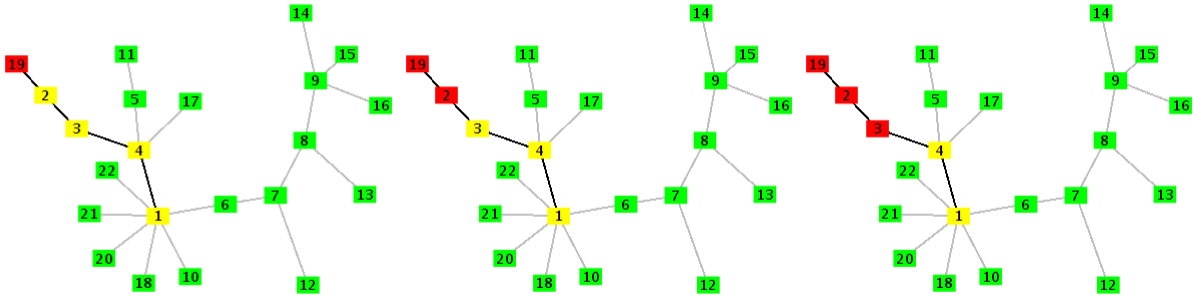
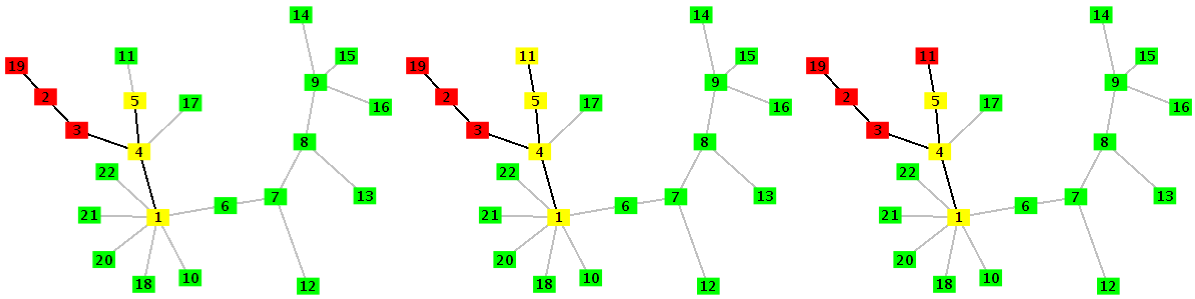
```
> priufer:=[1,5,4,7,8,9,9,9,8,7,6,1,4,1,2,3,4,1,1,1]; # Priuferio kodas

> priufer3();radialinis(G, Pi*4/3, 3/3, 0.2, 2*Pi, 0.8);
```

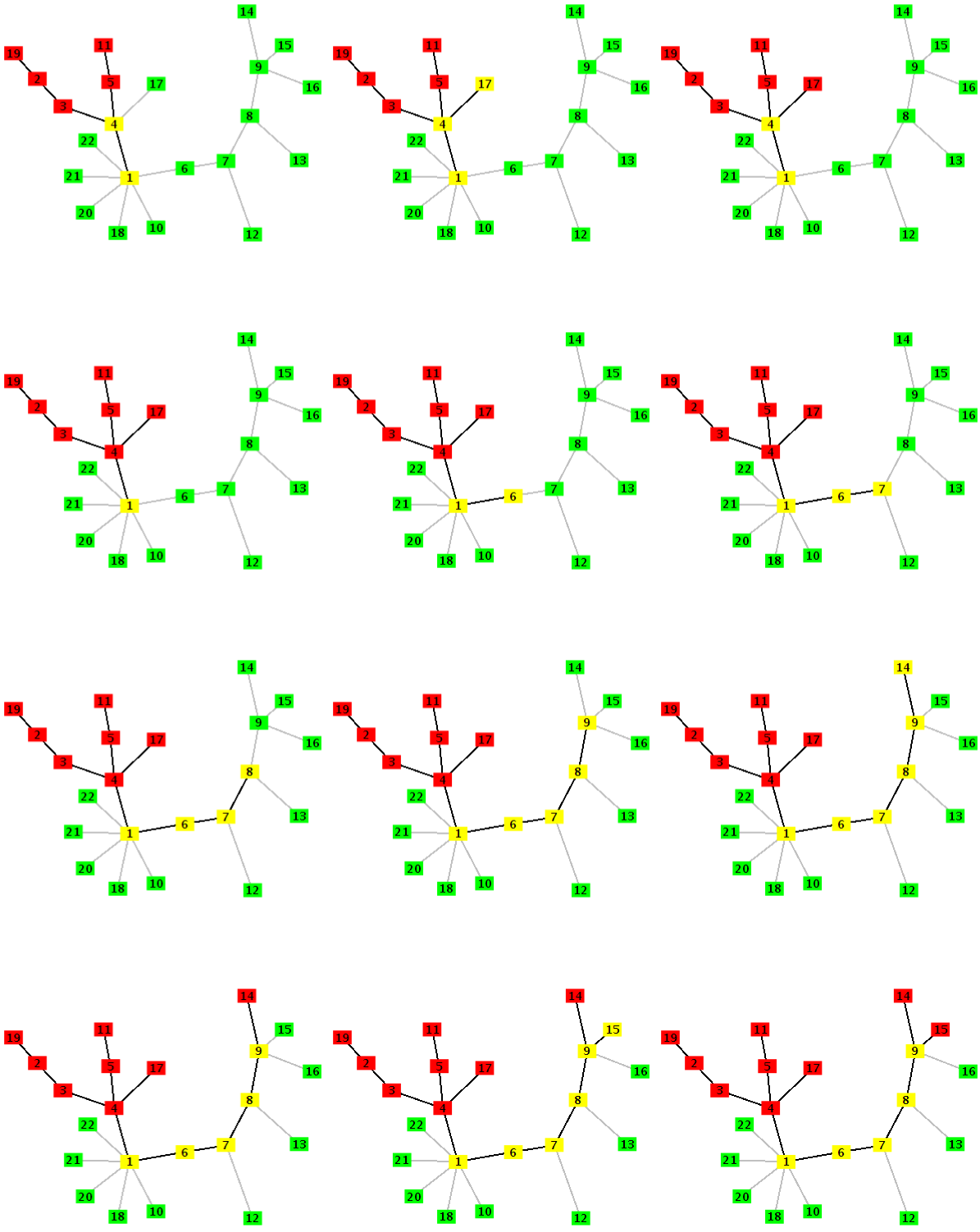


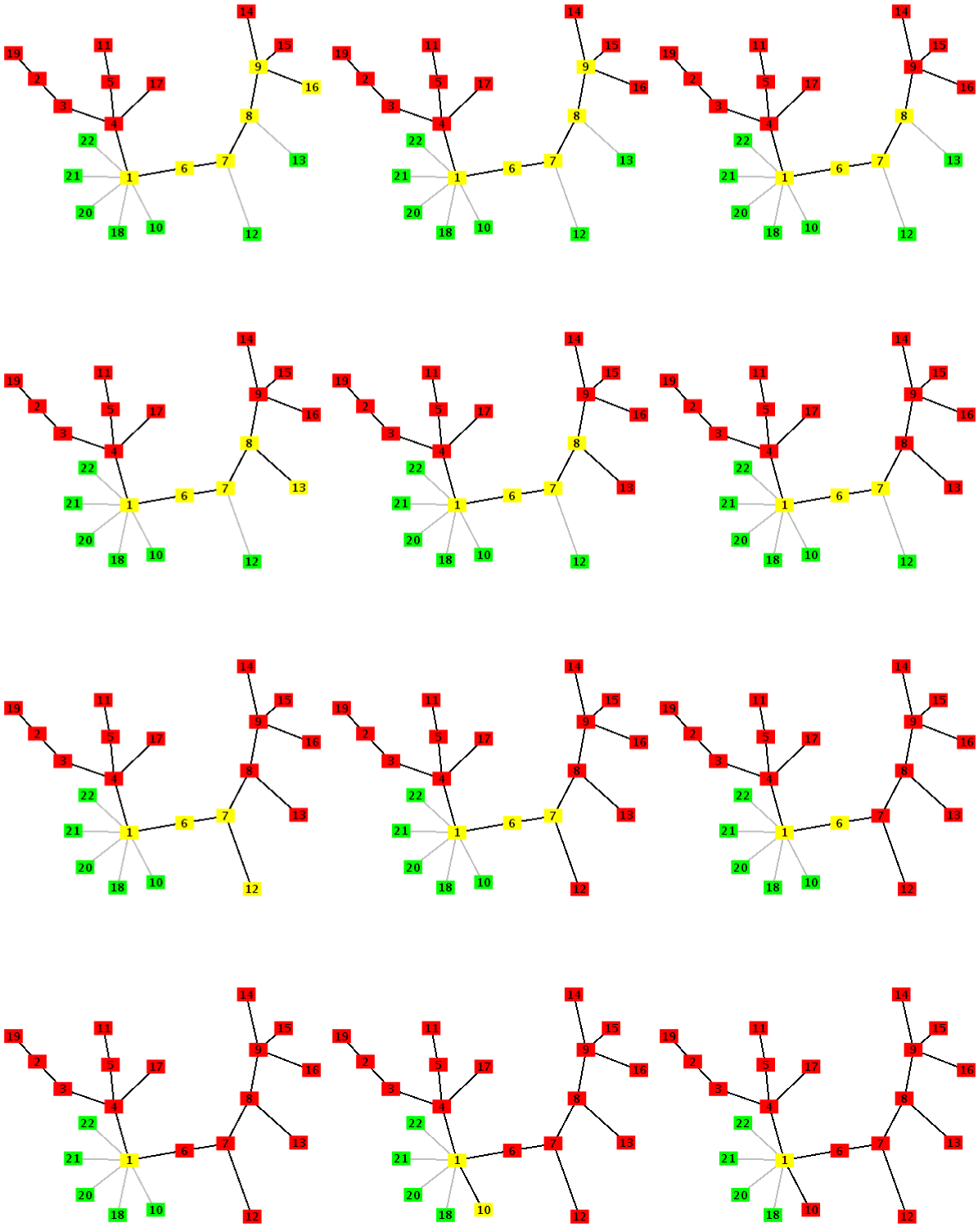

```
> radialinis2(p,plot1);
```

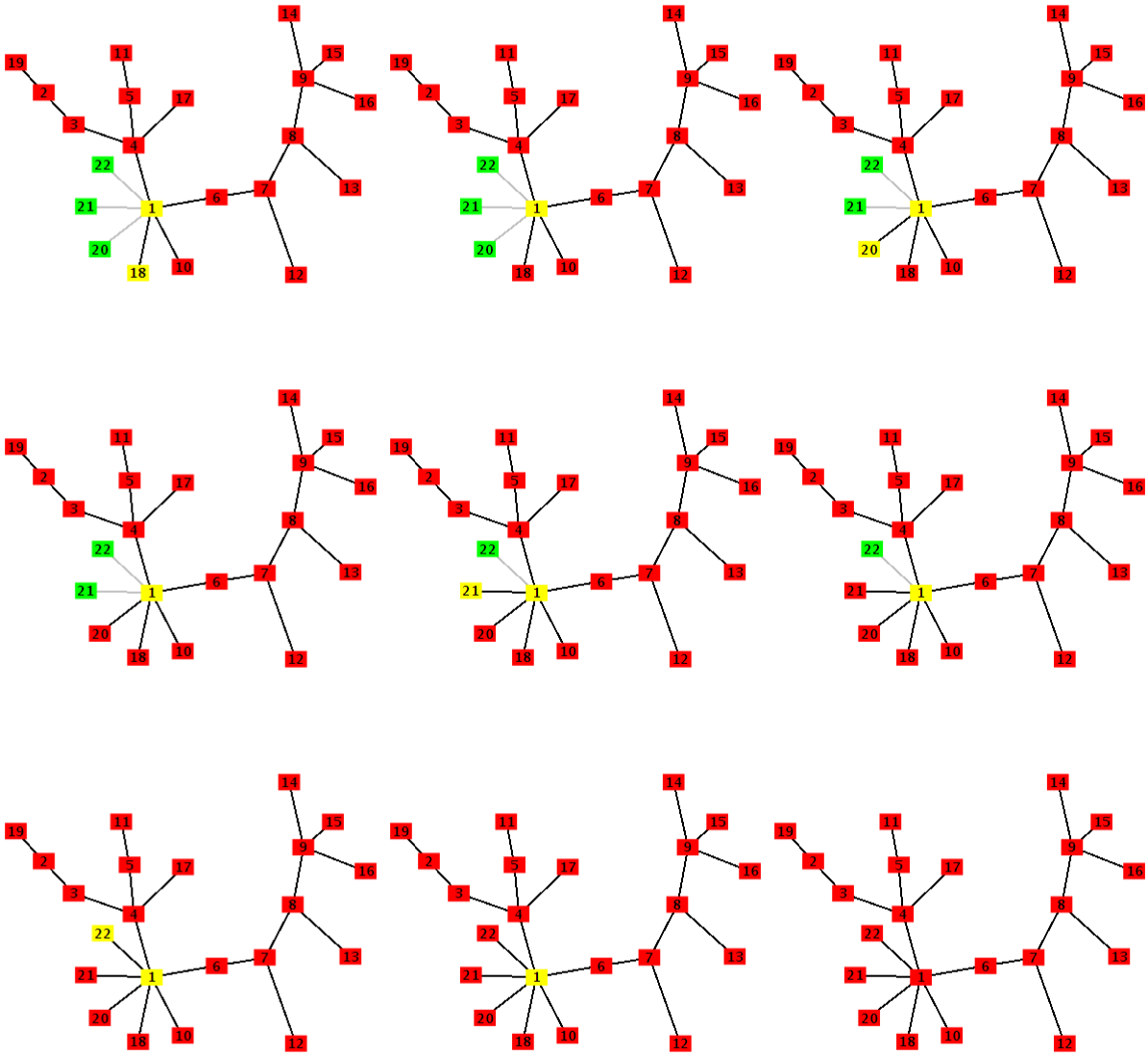



```
> color_1:=green; # pradinio (inicializuoto) grafo viršūnių spalva
> color_2:=yellow; # algoritmo vykdymo metu nagrinėjamų viršūnių spalva
> color_3:=red; # algoritmo vykdymo išnagrinėtų (pereitų) viršūnių spalva
> color_4:=grey; # pradinio grafo briaunų spalva
> color_5:=black; # briaunų, kuriomis buvo eita algoritmo realizacijos metu,
spalva

> pgyl(G);
> for i from 1 to tt-1 do draw[i] end do;
```




19 2 3 4 11 5 17 22 21 1 20 18 10 6 7 12 8 13 9 15 16 14



```
> for i from 1 to Count(Vertices(G)) do printf ("c[%d]=%q, ",i,c[i]) end do;
```
c[1]=red, c[2]=red, c[3]=red, c[4]=red, c[5]=red, c[6]=red, c[7]=red, c[8]=red, c[9]=red, c[10]=red, c[11]=red, c[12]=red, c[13]=red, c[14]=red, c[15]=red, c[16]=red, c[17]=red, c[18]=red, c[19]=red, c[20]=red, c[21]=red, c[22]=red,

```
> for i from 1 to Count(Vertices(G)) do printf ("pi[%d]=%q, ",i,pi[i]) end do;
```
pi[1]=NIL, pi[2]=3, pi[3]=4, pi[4]=1, pi[5]=4, pi[6]=1, pi[7]=6, pi[8]=7, pi[9]=8, pi[10]=1, pi[11]=5, pi[12]=7, pi[13]=8, pi[14]=9, pi[15]=9, pi[16]=9, pi[17]=4, pi[18]=1, pi[19]=2, pi[20]=1, pi[21]=1, pi[22]=1,

```
> for i from 1 to Count(Vertices(G)) do printf ("d[%d]=%q, ",i,d[i]) end do;
```
d[1]=1, d[2]=4, d[3]=3, d[4]=2, d[5]=9, d[6]=16, d[7]=17, d[8]=18, d[9]=19, d[10]=34, d[11]=10, d[12]=30, d[13]=27, d[14]=20, d[15]=22, d[16]=24, d[17]=13, d[18]=36, d[19]=5, d[20]=38, d[21]=40, d[22]=42,

```
> for i from 1 to Count(Vertices(G)) do printf ("f[%d]=%q, ",i,f[i]) end do;
```
f[1]=44, f[2]=7, f[3]=8, f[4]=15, f[5]=12, f[6]=33, f[7]=32, f[8]=29, f[9]=26, f[10]=35, f[11]=11, f[12]=31, f[13]=28, f[14]=21, f[15]=23, f[16]=25, f[17]=14, f[18]=37, f[19]=6, f[20]=39, f[21]=41, f[22]=43,

Grafų paieškos algoritmai trimatėje erdvėje

Vizualizacijos procedūros

```
> InitColors:=proc()
> local i,j;
> global A1,B1,A,B,G,coor;
> for i from 1 to Count(Vertices(G)) do
> B[i]:=display({point(coor[i])},color = color_1, symbolsize=size1);
> end do;
> for j from 1 to Count([Edges(G)[]]) do
> A[j]:=display({curve([coor[Edges(G)[j][1]],coor[Edges(G)[j][2]]])}, color =
color_4, thickness = size2):
> end do;
> A1:=display({seq(A[i],i=1..Count(Vertices(G))-1)});
> B1:=display({seq(B[i],i=1..Count(Vertices(G)))});
> return();
> end proc;
```

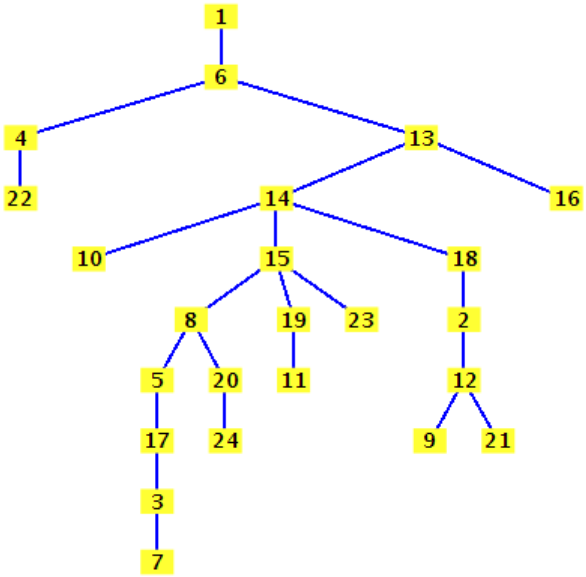
```
> HighlightVertex3d:=proc(n,sp)
> local i;
> global B,B1,G;
> B[n]:=display({point(coor[n])},color = sp, symbolsize=size1);
> B1:=display([seq(B[i],i=1..Count(Vertices(G)))],scaling=constrained);
> return();
> end proc;
```

```
> HighlightEdges3d:=proc(q1,q2,sp)
> local i,k;
> global A,A1,G;
> k:=1;
> while {q1,q2}<>Edges(G)[k] do
> k:=k+1;
> end do;
> A[k]:=display({curve([coor[Edges(G)[k][1]],coor[Edges(G)[k][2]]])}, color =
sp, thickness = size2);
> A1:=display([seq(A[i],i=1..Count([Edges(G)[]]))],scaling=constrained);
> return();
> end proc;
```

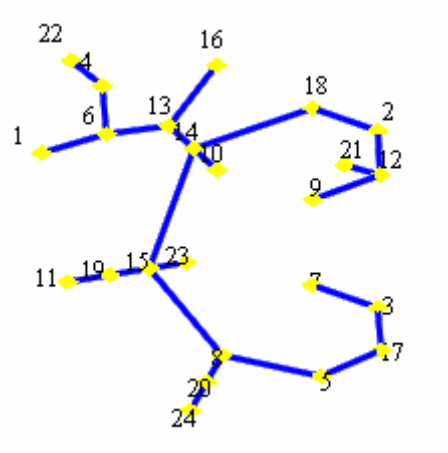
Paieška į plotį trimatėje erdvėje

```
> restart;
> with(GraphTheory):
> with(ArrayTools):
> with(Statistics):
> with(LinearAlgebra):
> with(RandomGraphs):
> with(plots):
> with(plottools):
> with(VectorCalculus):
> with(ListTools):
> with(queue):

> pplot3d:=proc(G,s)
> local u, Q, v, i;
> global draw, tt, c, d, pi;
> InitColors():
> tt:=1:
> draw[tt]:=display([A1,B1,C1],scaling=constrained);
> tt:=tt+1:
> for u in Vertices(DeleteVertex(G,s)) do
> c[u]:=color_1;
> d[u]:=infinity;
> pi[u]:=NIL;
> end do;
> c[s]:=color_2;
> d[s]:=0;
> pi[s]:=NIL;
> Q:=new(s):
> HighlightVertex3d(s,color_2);
> draw[tt]:=display([A1,B1,C1],scaling=constrained);
> tt:=tt+1:
> while length(Q)<>0 do
> u:=front(Q);
> for v in Neighbors(G,u) do
> if c[v]=color_1 then
> c[v]:=color_2;
> d[v]:=d[u]+1;
> pi[v]:=u;
> HighlightEdges3d(v,u,color_5);
> HighlightVertex3d(v,color_2);
> enqueue(Q,v);
> end if;
> if c[v]=color_2 then
> HighlightEdges3d(v,u,color_5)
> end if;
> end do;
> dequeue(Q);
> HighlightVertex3d(u,color_3);
> c[u]:=color_3;
> draw[tt]:=display([A1,B1,C1],scaling=constrained);
> tt:=tt+1:
> end do;
> return();
> end proc;
```

```
> G:=RandomTree(24);DrawGraph(G);K:=DrawGraph(G):
```



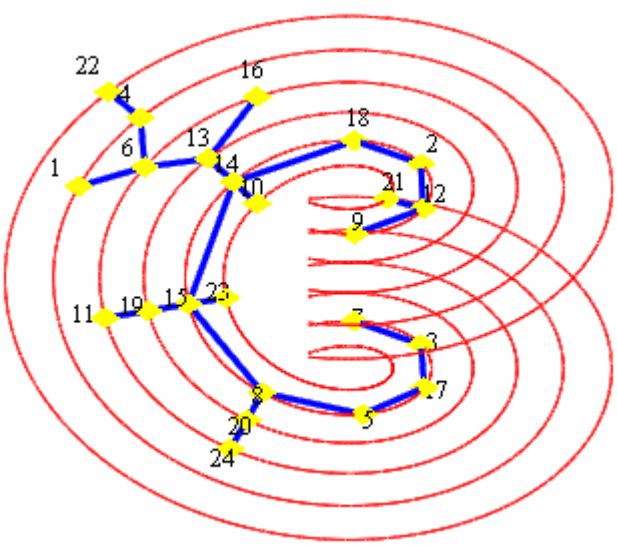

```
> up:=1.15; # teksto atstumas iki taško
> fontsize:=10; # šrifto dydis
> pointsize:=10; # taško dydis

> lauruvainikas(G,1, 4, 1.3, 1, -Pi, 1, 0.2, 0.1, 2); # vykdomas apibendrintas
laurų vainiko algoritmas
```



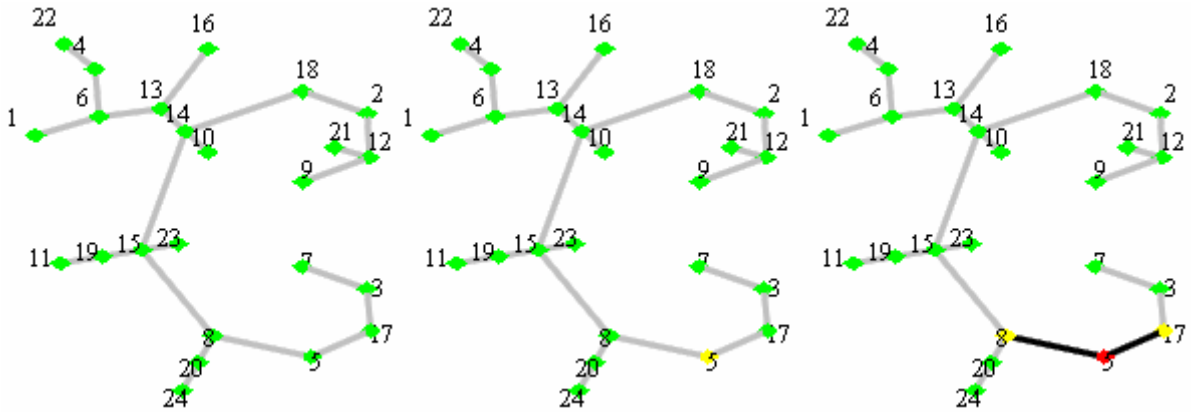

```
> for i from 1 to Count([convert(takas,set)[]]) do Takas[i]=takas[i] end do; #
išvedami visi ilgiausi takai
```

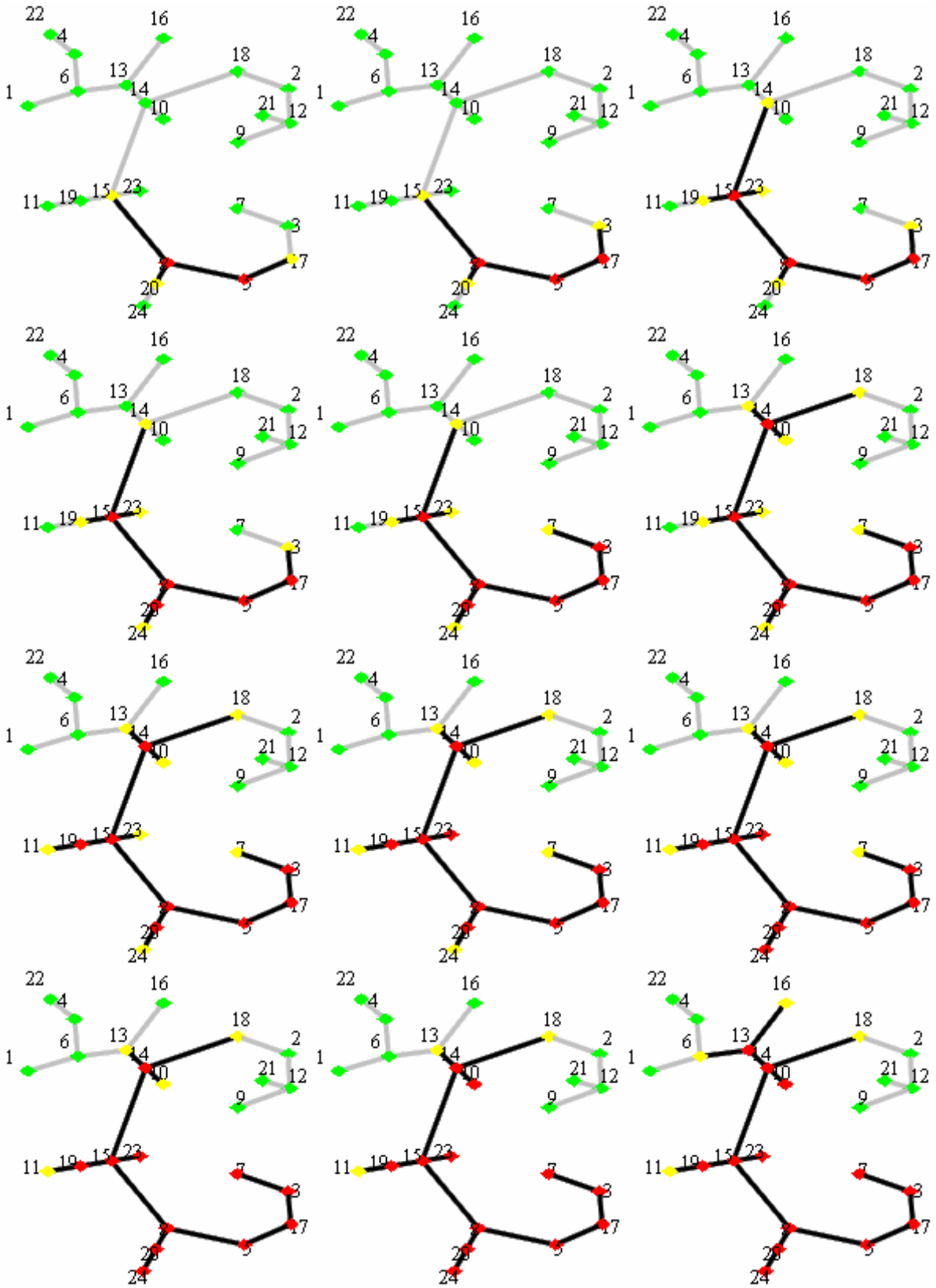
$Takas_1$ = [7,3,17,5,8,15,14,18,2,12,9]
$Takas_2$ = [7,3,17,5,8,15,14,18,2,12,21]
$Takas_3$ = [7,3,17,5,8,15,14,13,6,4,22]

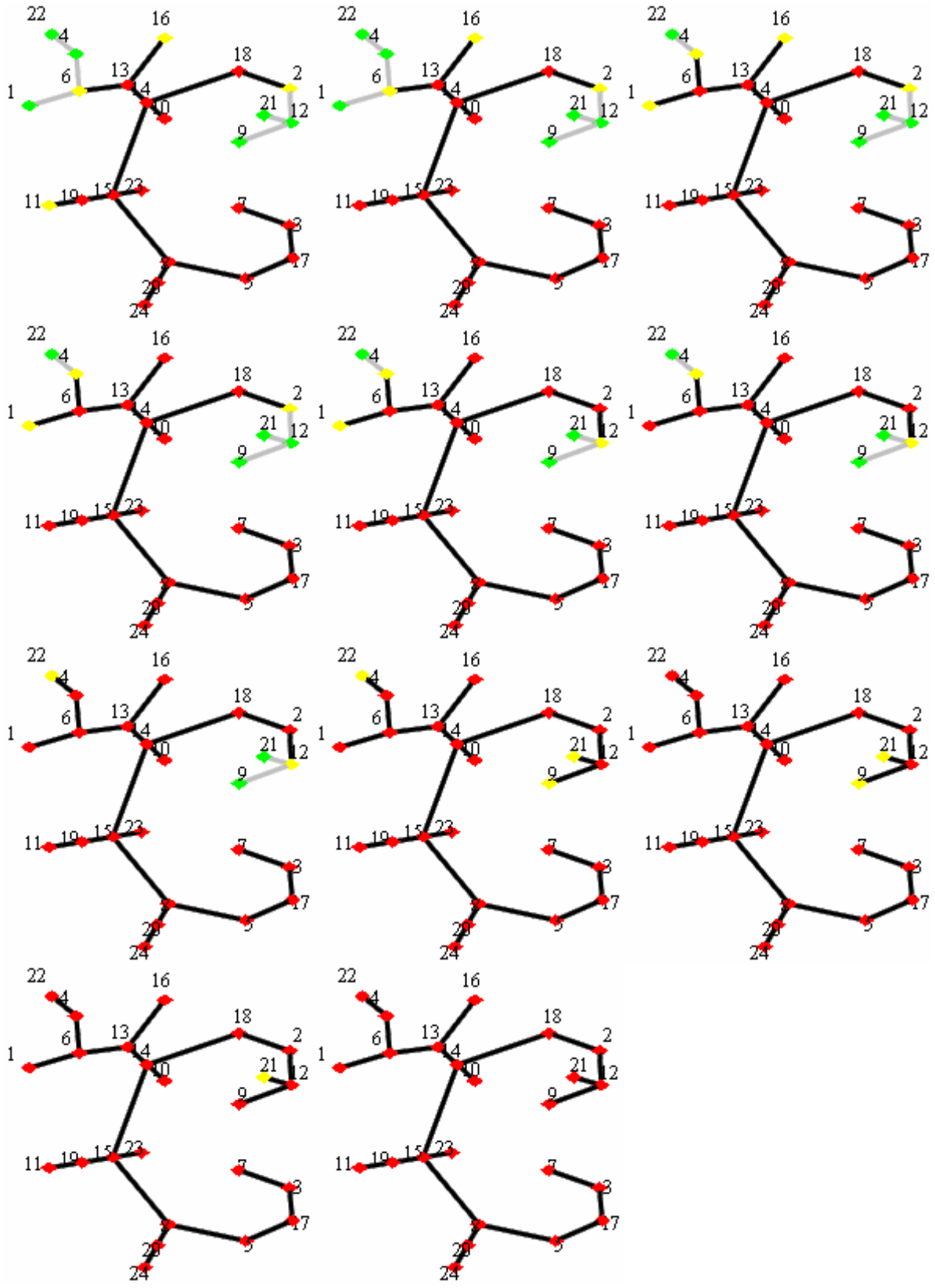
```
> lauras(p, plot1); # vaizduojama elipsių sistema
```




```
> s:=5: # pradinė viršūnė
> color_1:=green; # pradinio (inicializuoto) grafo viršūnių spalva
> color_2:=yellow; # algoritmo vykdymo metu nagrinėjamų viršūnių spalva
> color_3:=red; # algoritmo vykdymo išnagrinėtų (pereitų) viršūnių spalva
> color_4:=grey; # pradinio grafo briaunų spalva
> color_5:=black; # briaunų, incidenčių nagrinėjamai viršūnei, spalva

> pplot3d(G,s);
> for i from 1 to tt-1 do draw[i] end do;
```





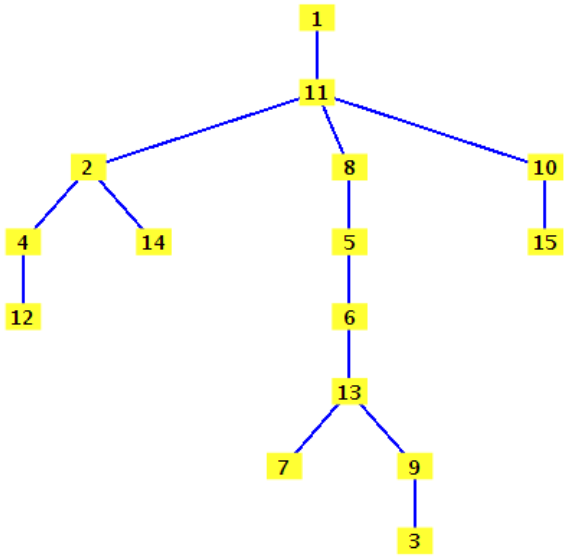


```
> for i from 1 to Count(Vertices(G)) do printf ("c[%d]=%q, ",i,c[i]) end do;
c[1]= red, c[2]= red, c[3]= red, c[4]= red, c[5]=red, c[6]=red, c[7]=red, c[8]=red,
c[9]=red, c[10]=red, c[11]=red, c[12]=red, c[13]=red, c[14]= red, c[15]=red,
c[16]=red, c[17]=red, c[18]=red, c[19]=red, c[20]=red, c[21]=red, c[22]=red,
c[23]=red, c[24]=red,
> for i from 1 to Count(Vertices(G)) do printf ("d[%d]=%q, ",i,d[i]) end do;
d[1]=5, d[2]=4, d[3]=5, d[4]=6, d[5]=0, d[6]=5, d[7]=4, d[8]=1, d[9]=4, d[10]=2,
d[11]=1, d[12]=2, d[13]=3, d[14]=3, d[15]=3, d[16]=4,
> for i from 1 to Count(Vertices(G)) do printf ("pi[%d]=%q, ",i,pi[i]) end do;
pi[1]=16, pi[2]=14, pi[3]=9, pi[4]=6, pi[5]=NIL, pi[6]=2, pi[7]=13, pi[8]=5,
pi[9]=13, pi[10]=11, pi[11]=5, pi[12]=11, pi[13]=10, pi[14]=12, pi[15]=10,
pi[16]=13,
```

Paieška į gylį trimatėje erdvėje

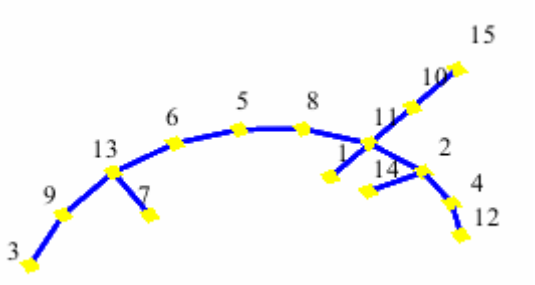
```
> restart;
> with(GraphTheory):
> with(ArrayTools):
> with(Statistics):
> with(LinearAlgebra):
> with(RandomGraphs):
> with(plots):
> with(plottools):
> with(VectorCalculus):
> with(ListTools):
> with(queue):

> VISIT:=proc(u)
> local v;
> global c,pi,d,f,tt,draw;
> c[u]:=color_2;
> HighlightVertex3d(u,color_2);
> tt:=tt+1;
> d[u]:=tt;
> draw[tt]:=display([A1,B1,C1],scaling=constrained);
> for v in Neighbors(G,u) do
> HighlightEdges3d(v,u,color_5);
> if c[v]=color_1
> then pi[v]:=u;
> VISIT(v)
> end if; end do;
> c[u]:=color_3;
> HighlightVertex3d(u,color_3);
> tt:=tt+1;
> f[u]:=tt;
> draw[tt]:=display([A1,B1,C1],scaling=constrained);
> end proc;

> pgyl3d:=proc(G)
> local u, i;
> global c, pi, tt, d, f, draw;
> InitColors():
> tt:=0:
> draw[tt]:=display([A1,B1,C1],scaling=constrained);
> for u in Vertices(G) do
> c[u]:=color_1:
```

```
> pi[u]:=NIL:
> d[u]:=0:
> f[u]:=0:
> end do;
> for u in Vertices(G)
> do if c[u]=color_1
> then VISIT(u)
> end if;
> end do;
> return();
> end proc;

> G:=RandomTree(15);DrawGraph(G);K:=DrawGraph(G):
```




```
> up:=1.1; # teksto atstumas iki taško
> fontsize:=10; # šrifto dydis
> pointsize:=25; # taško dydis

> priufer3(G);lauruvainikas(G,tk); # vykdomas apibendrintas laurų vainiko
algoritmas
```

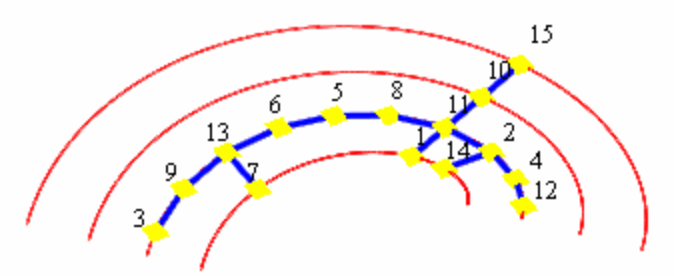
[11,9,13,13,4,2,6,5,8,11,2,11,10]



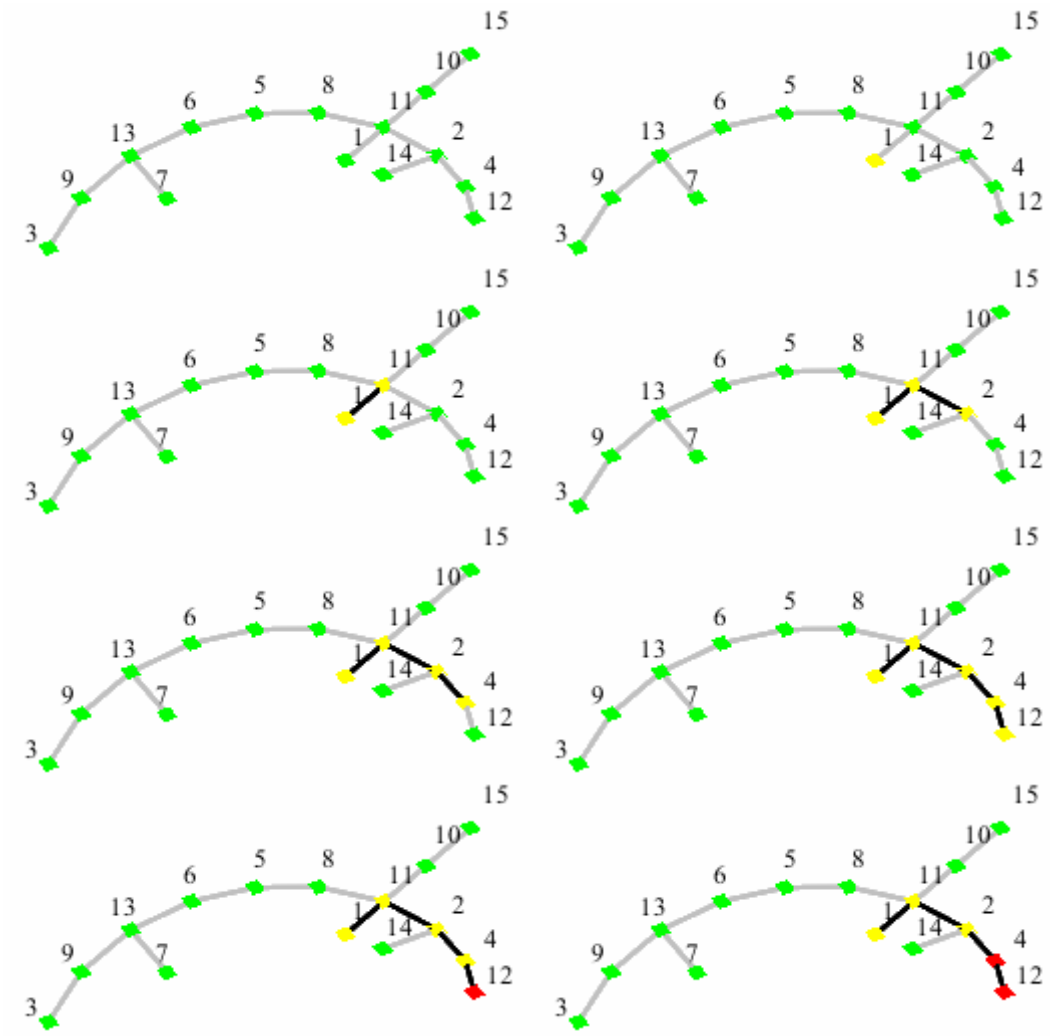

```
> for i from 1 to Count([convert(takas,set)[]]) do Takas[i]=takas[i] end do; #
išvedami visi ilgiausi takai
```

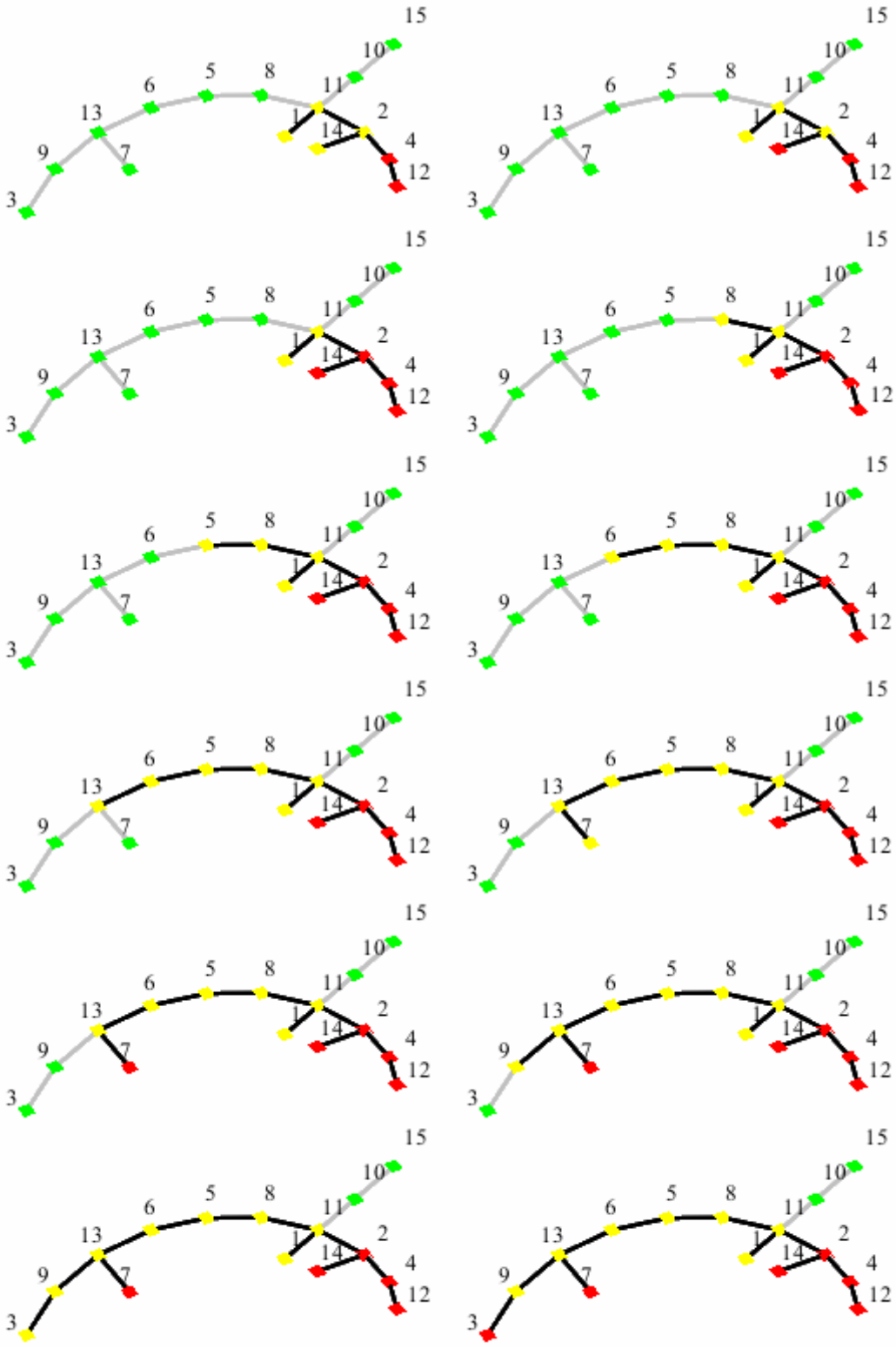
$Takas_1$ = [3,9,13,6,5,8,11,2,4,12]

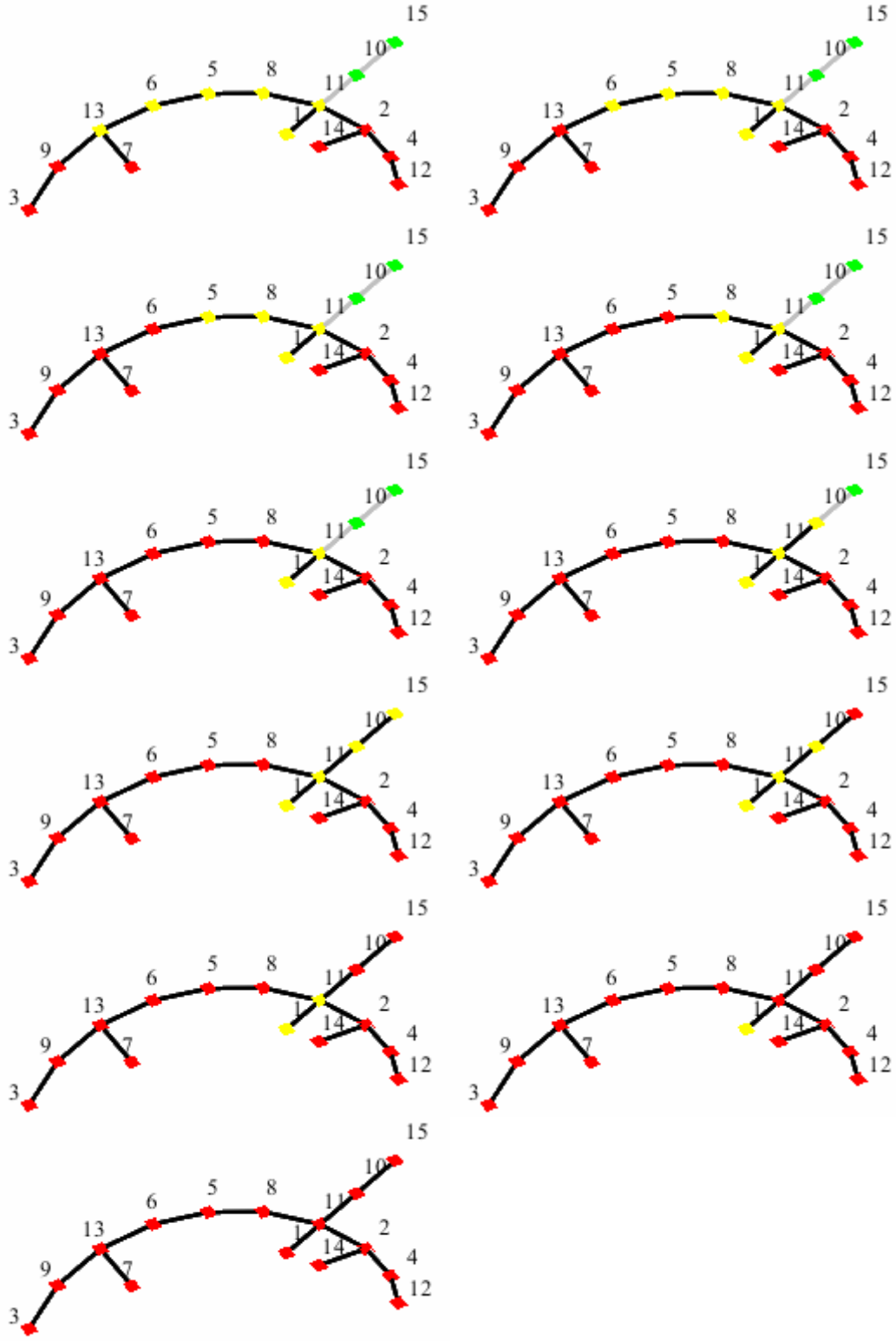
```
> lauras(p, plot1); # vaizduojama spiralių sistema
```




```
> color_1:=green; # pradinio (inicializuoto) grafo viršūnių spalva
> color_2:=yellow; # algoritmo vykdymo metu nagrinėjamų viršūnių spalva
> color_3:=red; # algoritmo vykdymo išnagrinėtų (pereitų) viršūnių spalva
> color_4:=grey; # pradinio grafo briaunų spalva
> color_5:=black; # briaunų, kuriomis buvo eita algoritmo realizacijos metu,
spalva

> pgyl3d(G,s);
> for i from 0 to tt do draw[i] end do;
```




15
10
6
5
8
11
13
2
1
14
4
9
7
12
3


15
10
5
8
6
11
13
2
4
9
7
12
3

```
> for i from 1 to Count(Vertices(G)) do printf ("c[%d]=%q, ",i,c[i]) end do;
c[1]=red, c[2]=red, c[3]=red, c[4]=red, c[5]=red, c[6]=red, c[7]=red, c[8]=red,
c[9]=red, c[10]=red, c[11]=red, c[12]=red, c[13]=red, c[14]=red, c[15]=red,
> for i from 1 to Count(Vertices(G)) do printf ("pi[%d]=%q, ",i,pi[i]) end do;
pi[1]=NIL, pi[2]=11, pi[3]=9, pi[4]=2, pi[5]=8, pi[6]=5, pi[7]=13, pi[8]=11,
pi[9]=13, pi[10]=11, pi[11]=1, pi[12]=4, pi[13]=6, pi[14]=2, pi[15]=10,
> for i from 1 to Count(Vertices(G)) do printf ("d[%d]=%q, ",i,d[i]) end do;
d[1]=1, d[2]=3, d[3]=18, d[4]=4, d[5]=12, d[6]=13, d[7]=15, d[8]=11, d[9]=17,
d[10]=25, d[11]=2, d[12]=5, d[13]=14, d[14]=8, d[15]=26,
> for i from 1 to Count(Vertices(G)) do printf ("f[%d]=%q, ",i,f[i]) end do;
f[1]=30, f[2]=10, f[3]=19, f[4]=7, f[5]=23, f[6]=22, f[7]=16, f[8]=24, f[9]=20,
f[10]=28, f[11]=29, f[12]=6, f[13]=21, f[14]=9, f[15]=27,
```

Literatūros sąrašas

[1] Di Battista et al. (1994), Section 2.1.2, Aesthetics, pp. 14–16.

[2] C.Wetherell and A. Shannon. Tidy drawings of trees. IEEE Trans. Softw. Eng., 5(5):514–520, 1979.

[3] E. M. Reingold and J. S. Tilford. Tidier drawings of trees. IEEE Trans. Softw. Eng., 7(2):223–228, 1981

[4] I. J. Q. Walker. A node-positioning algorithm for general trees. Softw. Pract. Exper., 20(7):685–705, 1990.

[5] C. Buchheim, M. J¨unger, and S. Leipert. Improving walker's algorithm to run in linear time. In GD '02: Revised Papers from the 10th International Symposium on Graph Drawing, pages 344–353, London, UK, 2002. Springer-Verlag.

[6] P. Eades. Drawing free trees. Bulletin of the Institute for Combinatorics and its Applications, October.

[7] G. Melanc¸on and I. Herman. Circular drawings of rooted trees. Technical report, Amsterdam, The Netherlands, The Netherlands, 1998.

[8] G. G. Robertson, J. D. Mackinlay, and S. K. Card. Cone trees: animated 3d visualizations of hierarchical information. In CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems, pages 189–194, New York, NY, USA, 1991. ACM.

[9] Greg Book & Neeta Keshary, Radial Tree Graph Drawing Algorithm for Representing Large Hierarchies, University of Connecticut, December 2001.

[10] Heinz Prüfer: Neuer Beweis eines Satzes über Permutationen. Arch. Math. Phys. 27: 742-744. 1918.

[11] W. H. Smith., Graphic Statistics in Management (McGraw-Hill Book Company, New York, ed. First, 1924)

[12] James Aleksander, Loxodromes: A Rhumb Way to Go, Mathematics Magazine, Vol. 77, No. 5, December 2004

[13] Knuth, Donald E. (1997), The Art Of Computer Programming Vol 1. 3rd ed., Boston: Addison-Wesley, ISBN 0-201-89683-4

[14] Even, Shimon (2011), Graph Algorithms (2nd ed.), Cambridge University Press, pp. 46–48, ISBN 978-0-521-73653-4.