VILNIAUS UNIVERSITETAS
ŠIAULIŲ AKADEMIJA

INFORMACINIŲ TECHNOLOGIJŲ VALDYMO MAGISTRO STUDIJŲ PROGRAMA

LILIJA SINIAVSKA

Magistro studijų baigiamasis darbas

MAKSIMALAUS SRAUTO TINKLE ALGORITMO TAIKYMAS IŠTRŪKIMO UŽDAVINIAMS SPRĘSTI
MAXIMUM FLOW NETWORK ALGORITHM FOR SOLVING OUTAGE PROBLEMS

Darbo vadovas (-ė): dr. Vaidas Giedrimas

Šiauliai, 2023

Studijuojančiojo, teikiančio baigiamąjį darbą, GARANTIJA

WARRANTY of Final Thesis

Vardas, pavardė *Name, Surname*	**Lilija Siniavska**
Padalinys *Faculty*	**Šiaulių akademija** ***Šiauliai Academy***
Studijų programa *Study Programme*	**Informacinių technologijų valdymo magistro studijų program** **Master study program in information technology management**
Darbo pavadinimas *Thesis topic*	**Maksimalaus srauto tinkle algoritmo taikymas ištrūkimo uždaviniams spręsti** **Maximum flow network algorithm for solving outage problems**
Darbo tipas *Thesis type*	**Baigiamasis darbas** ***Final Thesis***

Garantuoju, kad mano baigiamasis darbas yra parengtas sąžiningai ir savarankiškai, kitų asmenų indėlio į parengtą darbą nėra. Jokių neteisėtų mokėjimų už šį darbą niekam nesu mokėjęs.

I guarantee that my thesis is prepared in good faith and independently, there is no contribution to this work from other individuals. I have not made any illegal payments related to this work.

Šiame darbe tiesiogiai ar netiesiogiai panaudotos kitų šaltinių citatos yra pažymėtos literatūros nuorodose.

Quotes from other sources directly or indirectly used in this thesis, are indicated in literature references.

Aš, Lilija Siniavska pateikdamas (-a) šį darbą, patvirtinu (*pažymėti*) ☑

Embargo laikotarpis
Embargo Period

Prašau nustatyti šiam baigiamajam darbui toliau nurodytos trukmės embargo laikotarpį:
I am requesting an embargo of this thesis for the period indicated below:

☐ _____ mėnesių / *months*
(embargo laikotarpis negali viršyti 60 mėn. / *an embargo period shall not exceed 60 months*).

☑ Embargo laikotarpis nereikalingas / *no embargo requested.*

Embargo laikotarpio nustatymo priežastis / *Reason for embargo period*:

TURINYS

PAVEIKSLĖLIŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

SANTRAUKA

Mūsų gyvenimas sparčiai keičiasi ir jame vyksta nuolatiniai pokyčiai ne tik artimoje aplinkoje, bet ir visuose kituose gyvenimo srityse. Jau dabar mokiklinukas nuo pirmos klasės tenka susidurti su informacijos gausa, spręsti iškilusius uždavinius. Jie ateina mokytis ir įgyti tam tikrų įgūdžių, Bet ne vienas mokykloje susiduria su įvairiausiomis kliūtimis, kurias reikia mokėti spręsti. Augant tenka išmokti surasti reikalingos informacijos, įgyti praktinių įgudžių.

Jau dabar mokyklose informacinės technologijos(IT) integruotos į pamokas nuo pirmųjų žingsnių mokykloje. Deja bet net turėdami vyresnėse klasėse IT pamokas dažnas mokinys mano, kad IT technologijos skirtos tik žaidimams ir pramogai. Taip mes turim vadovėlius su teorija ir kartais ją IT pamokose bandome pritaikyti praktiškai, tik tai neduoda rezultatų. Pagal apklausą 80% 5 – 8 klasių mokinių norėtu būti programišiais(chakeriais). Problema ta, kad jie net nesuvokia kas tai yra.

Norint sudominti tokius mokinius ir padidinti ugdymo kokybę mokytojas turi pats praktikuotis, ieškoti įvairiausių uždavinių sprendimo būdų. Vienas iš sprendimų pasinaudoti žaidimais.

Prieš įtraukiant žaidimus į ugdymo procesą reikalinga analizė ir supratimas kaip tai veikia. Pirmoje darbo dalyje trumpa apžvalga apie darbo tikslą, metodus, uždavinius bei darbo aktualumą.

Daug girdime apie maksimalius srauto tinklus, įvairiausius algoritmus ir optimizavimo metodus. Šiame darbe bandoma išsiaiškinti kas tai yra pasinaudojus įvairiausiais literarūros šaltiniais antrame skyriuje, analizinėje dalyje.

Trečiame skyriuje aprašomi žaidimų, programėlių ir pan. kurimo algoritmai, sukuriami paprasti žaidimais pasinaudojus kodų rašymo programomis. Buvo sukurti žaidimų, matematinių uždavinių ir laisvalaikio programėlių kodai.

Parašytos išvados ir įžvalgos.

SUMMARY

In todays fast pace environment we can feel frequent changes in our family, education, social, etc. Environments. Our first year pupils are already challenged with excess information, or how to deal with hefty tasks. Supposingly first year school children are starting school to learn some sort of social, educational, and problem solving skills, however most likely, they are already required to know at least basics of it, as certain ‚life quests' start to uncover. While growing pupils are required to start separating appropriate information, to get useful practical knowledge.

As for information being everything and everywhere, and being available most of the time, IT starts from first steps at school. Despite the fact that it has been tried to teach practical IT application from available text books at school, even in senior class it is still believed that the main reason of learning as well as using IT is for the purpose of gaming or other pleasure activities.

According to recent research, 80% off 5-8 grade students would love to become IT specialists, however looking deeper into it, school-children show near to no understanding what it is like to be an IT specialist.

For all these reasons, our day society teachers themselves are forced to acquire not only basic knowledge of IT, but to look for available alternative teaching methods, like integrating gaming into todays lessons. And ofcourse for such reason, deeper analysis and understanding of gaming society has to be looked into.

In the first part of the research it is looked at purpose and methods of the research; tasks and actuality of the work.

In IT we hear a lot. about max flow networks, algorithms or optimisation methods. In the second part of the research we will try to look what it actualy is, analysing most recent literature.

And last but not least, third part of the research is all about games, algorythms for creating apps, made of using video games creation algorythms. And finaly gaming, math and leasure activity app codes are being developed, including research conclusions and insights.

ĮVADAS

Pirmieji kompiuteriai buvo kuriami lempų pagrindu ir jų paskirtis tuo metu buvo daugiau spręsti mokslinius, inžinierinius ir karinius uždavinius. Šie kompiuteriai buvo didžiuliai, pajėgumas mažas bei nepatogūs eksploatuoti dėl sudėtingos duomenų įvesties ir išvesties, pakankamai komplikuoto naudojimo ir programavimo.

Sekantis kopiuteriai turėjo kur kas patogesnę duomenų įvestį ir išvestį, didesnį veikimo greitį, taip pat lankstesnį naudojimą ir programavimą.

Toliau kompiuterių pagrindas tapo integrinės mikroschemos, jos leido sumažinti kompiuterių korpusą, padidinti našumą ir galėjo suteikti daug daugiau galimybių naudotojiems. Čia jau buvo gaminami universalūs kompiuteriai bei pirmieji minikompiuteriai.

Kaip visuose srityse taip ir kompiuterinėse technologijose vyksta evoliucija. Sekantys kompiuteriai turėjo pagrindą didelės integracijos mikroschema. Šis pagrindas leido sukurti pirmuosius mikroprocesorius ir jie buvo pritaikyti asmeniniems kompiuteriams.

Toliau elektronikos elementai tobulėjo, tapo kokybiškesni. Kopiuterius pradėjo naudoti ne tik moksliniais tikslais, bet ir kasdieniniame gyvenime. Šiandienos kompiuteriai gali atlikti daug ir įvairių operacijų, bei pačių kompiuterių.

Kadangi kompiuterius pradėjo naudoti visur tai švietimo ugdyme atsirado poreikis mokyti vaikus naudotis šiais prietaisais. Taip atsirado mokomosios programos kurios efektyvino ugdymo procesą. Tikslas programų yra padėti pedagogui ir vaikams siekti geresnių rezultatų. Atsirado labai daug įvairiausių mokomųjų kompiurinių programų, kurios turėjo savų trukumų ir privalumų. Yra kelios krypties mokomosios programos:

- ✓ demonstravimo programos;
- ✓ pratybų ir kontrolės programos;
- ✓ imitavimo, eksperimentavimo, modeliavimo programos;
- ✓ pagalbinės mokymo programos;
- ✓ mokomieji žaidimai.

Dirbant su šiais mokymopagrindais , svarbiausias tampa ne be pedagogas o vaikas.

Darbo tikslas: pritaikyti maksimalaus srauto tinklo algoritmą ištrūkimo uždaviniams spręsti.

Darbo uždaviniai:

• Sukurti žaidimų taisykles, galimų veiksmų aibę, sukurti ištrūkimo žaidimui pritaikytą strategiją;

• Aprašyti ištrūkimo žaidimą matematinio optimizavimo uždaviniu, parinkti optimizavimo algoritmą;

• Ištrūkimo žaidimo maršruto paieška, kur maršrutas yra veiksmų aibė;

• Sukurti žaidimui pritaikytas sprendimo strategijas, apibrėžti sprendinių vertinimo kriterijus;

• Įvertinti sukurtų strategijų sprendinių paieškos efektyvumą, kodų pagalba sukurti žaidimus.

Darbo struktūra:

Darbą sudaro trys skyriai.

Pirmame skyriuje pristatomas darbo aktualumas ir kita. Taip pat supažindinama su optimizavimo algoritmo koncepcijomis.

Antrame skyriuje nagrinėjama teorinė medžiaga: problemos ištirtumas, tyrimuose naudojamų koncepcijų, metodų analizė, pasirinkimų pagrindimas, aptariamas ištrūkimo uždaviniai ir jų struktūra.

Trečiame skyriuje pristatomi parašyti kodai, konkretūs metodai, siūlymai, aprašoma kūrimo eiga, o paskutiniame - tyrimo rezultatai ir išvados.

Darbe taikomi metodai:

Tyrimo metodas – atliekamas eksperimentinis kodų rašymas, matematiniu optimizavimo uždaviniu aprašymas ištrūkimo uždavinio.

Darbo aktualumas:

Ištrūkimo uždavinys visų pirma yra kelio paieškos uždavinys, kurio metu lavinami mąstymo įgūdžiai ir siekiama rasti optimaliausią kelią.

Darbo problema:

Šio darbo metu sprendžiama kaip panaudojant algoritmą įveikti žaidimo kodų kūrimo kliūtis, gaunant kuo tinkamesnį sprendinį – optimaliausią uždavinio sprendimą. Galimybė panaudoti IT pamokose.

Darbo objektas:

Darbo objektas - ištrūkimo uždaviniai.

1. OPTIMIZAVIMO ALGORITMŲ PARINKIMAS

1.1. Optimizavimas

Spręsdami konkrečią optimizavimo problemą, pirmiausia reikia pasirinkti matematinį metodą, kuris leistų pasiekti galutinius rezultatus su mažiausiomis skaičiavimo sąnaudomis arba leistų gauti didžiausią informacijos kiekį apie norimą sprendimą. Vieno ar kito metodo pasirinkimą daugiausia lemia optimalios problemos formulavimas, taip pat naudojamo optimizavimo objekto matematinis modelis. Šiuo metu optimalioms problemoms spręsti dažniausiai naudojami šie metodai[37[:

- ✓ klasikinės analizės funkcijų tyrimo metodai;
- ✓ metodai, pagrįsti neapibrėžtų Lagrange veiksnių naudojimu;
- ✓ variacijų skaičiavimas;
- ✓ dinaminis programavimas;
- ✓ maksimumo principas;
- ✓ linijinis arba tiesinis programavimas;
- ✓ netiesinis programavimas.

Neįmanoma pasirinkti vieno metodo, kuris gali būti naudojamas visoms realybėje kylančioms problemoms spręsti. Kai kurie metodai šiuo atžvilgiu yra bendresnio pobūdžio, o kiti – mažiau bendri. Visa viena grupė metodų (klasikinės analizės funkcijų tyrimo metodai, Lagrange daugiklių metodas, netiesinio programavimo metodai) gali būti naudojami kartu su kitais metodais, pavyzdžiui, dinaminiu programavimu arba maksimaliu principu, tam tikruose optimalios problemos sprendimo etapuose.

Yra metodai specialiai sukurti optimalioms problemoms spręsti tam tikrais matematiniais modeliais. Kitaip tariant tiesinio programavimo matematinis modelis yra specialiai sukurtas spręsti problemas, susijusias su tiesinio optimalumo kriterijais ir tiesiniais kintamųjų apribojimais, leidžia išspręsti daugumą tokioje formuluotėje suformuluotų problemų. Taip pat geometrinis programavimas yra skirtas optimalioms problemoms spręsti, kurių optimalumo kriterijus ir apribojimai išreiškiami tam tikromis pozinomų funkcijomis.

Dinaminis programavimas puikiai tinka optimizuoti daugiapakopius procesus, ypač tuos, kuriuose kiekvieno etapo būsenai būdingas palyginti nedidelis kintamųjų skaičius. Tačiau, esant dideliam šių kintamųjų, t.y. esant dideliam kiekvieno etapo matmeniui, dinaminio programavimo metodo taikymas yra sudėtingas dėl riboto kompiuterių greičio ir atminties[39[.

Galbūt geriausias būdas pasirinkti optimizavimo metodą yra rasti tinkamiausią būdą atitinkamai problemai spręsti ir ištirti įvairių optimizavimo metodų taikymo galimybes ir patirtį.

1.1.1. Variacijų skaičiavimo metodas

Variacijų skaičiavimo metodai paprastai naudojami sprendžiant problemas, kuriose optimalumo kriterijai pateikiami funkcijų pavidalu ir kurių sprendimai yra nežinomos funkcijos. Tokios problemos paprastai kyla statiškai optimizuojant procesus su paskirstytais parametrais arba dinaminio optimizavimo problemomis. Variaciniai metodai leidžia sumažinti optimalios problemos sprendimą Eulerio diferencialinių lygčių sistemos integravimo, kurių kiekviena yra antrosios eilės netiesinė diferencialinė lygtis su ribinėmis sąlygomis, pateiktomis abiejuose integracijos intervalo galuose. Šiuo atveju šios sistemos lygčių skaičius yra lygus nežinomų funkcijų skaičiui, nustatytam sprendžiant optimalią problemą. Kiekviena funkcija randama integravus gautą sistemą.

Eulerio lygtys gaunamos kaip būtinos sąlygos funkciniam ekstremumui. Todėl funkcijos, gautos integruojant diferencialinių lygčių sistemą, turi būti patikrintos dėl funkcinio ekstremumo. Jei yra lygties tipo apribojimų, turinčių funkcijų formą, naudojami Lagrange veiksniai, kurie leidžia pereiti nuo sąlyginės problemos prie besąlyginės. Didžiausi sunkumai naudojant variacinius metodus kyla sprendžiant problemas, susijusias su apribojimais, tokiais kaip nelygybė. Pažymėtina, kad tiesioginiai funkcinio optimizavimo problemų sprendimo būdai yra svarbūs, nes jie paprastai leidžia sumažinti pradinę variacijos problemą iki netiesinės programavimo problemos, kuri kartais yra lengviau išspręsta nei Eulerio lygčių ribinės vertės problema.

1.1.2. Dinaminis programavimas

Dinaminis programavimas yra veiksmingas būdas spręsti atskirų daugiapakopių procesų optimizavimo problemas, kurioms optimalumo kriterijus apibrėžiamas kaip atskirų etapų optimalumo kriterijų papildoma funkcija. Be didelių sunkumų, šis metodas gali būti išplėstas iki atvejo, kai optimalumo kriterijus pateikiamas kitokia forma, tačiau šiuo atveju atskirų etapų matmuo paprastai padidėja. Iš esmės dinaminis programavimo metodas yra algoritmas, skirtas nustatyti optimalią kontrolės strategiją visuose proceso etapuose. Tuo pačiu metu kontrolės įstatymas kiekviename etape randamas nuosekliai sprendžiant konkrečias optimizavimo problemas visuose proceso etapuose, naudojant klasikinės analizės funkcijų arba netiesinio programavimo metodų tyrimo metodus.

Kintamųjų problemų apribojimai neturi įtakos bendram sprendimo algoritmui, tačiau į juos atsižvelgiama sprendžiant konkrečias optimizavimo problemas kiekviename proceso etape. Jei yra lygybės tipo apribojimų, kartais netgi įmanoma sumažinti šių dalinių problemų matmenį naudojant Lagrange veiksnius. Dinaminio programavimo metodo taikymas optimizuojant procesus su paskirstytais parametrais arba dinaminio optimizavimo problemomis lemia dalinių diferencialinių lygčių sprendimą. Užuot sprendus tokias lygtis, dažnai daug lengviau galvoti apie nuolatinį procesą kaip apie diskretišką su pakankamai dideliu etapų skaičiumi. Toks metodas yra pateisinamas, ypač

tais atvejais, kai problemos kintamieji yra riboti, o tiesioginį diferencialinių lygčių sprendimą apsunkina poreikis atsižvelgti į šiuos apribojimus.

Sprendžiant problemas dinaminio programavimo metodu, paprastai naudojami kompiuteriai, turintys pakankamai atminties, kad būtų galima saugoti tarpinius sprendimo rezultatus, kurie paprastai gaunami lentelės forma.

1.1.3. Maksimumo principas

Maksimumo principas naudojamas sprendžiant proceso optimizavimo problemas, aprašytas diferencialinių lygčių sistemose. Maksimumo principo matematinio aparato pranašumas yra tas, kad sprendimas gali būti apibrėžtas nepertraukiamų funkcijų forma; Tai būdinga daugeliui optimizavimo problemų, tokių kaip optimalios objekto valdymo problemos, aprašytos tiesinėmis diferencialinėmis lygtimis.

Optimalaus sprendimo suradimas naudojant maksimumo principą sumažinamas iki proceso diferencialinių lygčių sistemos ir sistemos integravimo pagalbinėms funkcijoms esant ribinėms sąlygoms, pateiktoms abiejuose integracijos intervalo galuose, t.y. ribinės vertės problemos sprendimui. Apribojimai gali būti taikomi kintamų pakeitimų apimčiai. Diferencialinių lygčių sistema yra integruota naudojant įprastas programas skaitmeniniuose kompiuteriuose. Maksimalus diferencialinių lygčių aprašytų procesų principas tam tikromis prielaidomis yra pakankama optimalumo sąlyga. Todėl paprastai nereikia papildomai tikrinti gautų sprendimų optimalumo. Atskiriems procesams maksimumo principas toje pačioje formuluotėje, kaip ir nuolatiniams procesams, paprastai yra nesąžiningas. Tačiau optimalumo sąlygos, gautos, kai jos naudojamos daugiapakopiams procesams, leidžia rasti gana patogius optimizavimo algoritmus.[18]

1.1.4. Tiesinis programavimas

Tiesinis programavimas yra matematinis aparatas, skirtas išspręsti optimalias problemas su tiesinėmis išraiškomis optimalumo kriterijui ir tiesiniais kintamųjų diapazono apribojimais. Tokios užduotys paprastai susiduriama sprendžiant optimalaus gamybos planavimo problemas su ribotu išteklių skaičiumi, nustatant optimalų transportavimo planą (transporto užduotis) ir kt.

Norint išspręsti daugybę tiesinio programavimo problemų, yra beveik universalus algoritmas – simplex metodas, kuris leidžia rasti optimalų sprendimą daugumai problemų ribotame iteracijų skaičiuje. Naudojamų apribojimų tipas (lygybė ar nelygybė) neturi įtakos algoritmo taikymui. Nereikia atlikti papildomų optimalumo bandymų gautiems sprendimams. Paprastai praktinėms tiesinio programavimo problemoms būdingas labai daug nepriklausomų kintamųjų. Todėl jiems išspręsti paprastai naudojami kompiuteriai, kurių reikiamą galią lemia sprendžiamos problemos matmuo.

1.1.5. Netiesinis programavimas

Netiesiniai programavimo metodai naudojami optimalioms netiesinių tikslų funkcijų problemoms spręsti. Nepriklausomi kintamieji taip pat gali būti suvaržyti netiesinių santykių forma lygčių ar nelygybės pavidalu. Iš esmės naudojami netiesiniai programavimo metodai, jei nė vienas iš pirmiau minėtų metodų nedaro pažangos sprendžiant optimalią problemą. Dėl šios priežasties šie metodai kartais taip pat vadinami tiesioginiais optimalių problemų sprendimo būdais. Norint gauti skaitinius rezultatus, svarbi vieta skiriama netiesiniam programavimui ir optimalių problemų sprendimui, naudojant tokius metodus kaip dinaminis programavimas, maksimalus principas ir kt., Tam tikruose jų taikymo etapuose. Pavadinimas „netiesiniai programavimo metodai“ vienija didelę skaitmeninių metodų grupę, iš kurių daugelis yra pritaikyti optimalioms atitinkamos klasės problemoms spręsti. Vieno ar kito metodo pasirinkimą lemia optimalumo kriterijaus apskaičiavimo sudėtingumas ir ribojančių sąlygų sudėtingumas, būtinas tirpalo tikslumas, turimo kompiuterio galia ir kt. Daugelis netiesinių programavimo metodų beveik nuolat naudojami kartu su kitais optimizavimo metodais, tokiais kaip nuskaitymo metodas dinaminiame programavime. Be to, šie metodai yra pagrindas kurti automatines optimizavimo sistemas – optimizatorius, kurie tiesiogiai naudojami gamybos procesams valdyti. Netiesinių programavimo metodų, kurie kaip minėta pirmiau, gali būti vadinami tiesioginiais, naudojimas naudoja informaciją gautą apskaičiuojant optimalumo kriterijų, kurio keitimas yra konkretaus veiksmo efektyvumo įvertinimas. Svarbi bet kurios optimalios problemos savybė yra jos matmuo *n*, kuris yra lygus kintamųjų skaičiui, kurio vertės yra būtinos norint vienareikšmiškai nustatyti optimizuojamo objekto būklę. Paprastai didelių matmenų problemų sprendimas yra susijęs su poreikiu atlikti daug skaičiavimų. Keletas metodų (pvz., dinaminis programavimas ir diskrečiojo maksimumo principas) yra specialiai sukurti siekiant išspręsti didelių matmenų procesų optimizavimo problemas, kurios gali būti pateikiamos kaip daugiapakopiai procesai, turintys santykinai mažą kiekvieno etapo matmenį.

1.1.6. Geometrinis programavimas

Geometrinis programavimas yra specialios netiesinių programavimo problemų klasės sprendimo metodas, kuriame optimalumo ir apribojimų kriterijus pateikiamas pozinomų pavidalu, išraiškos, kurios yra nepriklausomų kintamųjų galios funkcijų produktų suma. Su tokiomis užduotimis kartais tenka susidurti projektuojant. Be to, kai kurios netiesinės programavimo problemos kartais gali būti sumažintos iki šio vaizdavimo, naudojant apytikslį objektyvių funkcijų ir apribojimų vaizdą.

Konkretus optimalių problemų sprendimo metodų bruožas (išskyrus netiesinius programavimo metodus) yra tas, kad iki tam tikro etapo optimali problema yra išspręsta analitiškai t.y. tam tikros analitinės išraiškos randamos, pvz. baigtinių ar diferencialinių lygčių sistemos, iš kurių randamas optimalus sprendimas.

1.2. Žaidimų teorija ir sprendimų priėmimas

Klasikinės sistemos analizės problemos yra žaidimų problemos, susijusios su sprendimų priėmimu rizikos ir neapsisprendimo sąlygomis. Tiek operacijos tikslai, tiek operacijos sąlygos, tiek sąmoningi oponentų ar kitų asmenų, nuo kurių priklauso operacijos sėkmė, veiksmai gali būti neaiškūs. Buvo sukurti specialūs matematiniai metodai, skirti pagrįsti sprendimus rizikos ir netikrumo sąlygomis. Kai kuriais paprasčiausiais atvejais šie metodai leidžia iš tikrųjų rasti ir pasirinkti optimalų sprendimą. Sudėtingesniais atvejais šie metodai suteikia pagalbinę medžiagą, leidžiančią giliau suprasti sudėtingą situaciją ir įvertinti kiekvieną iš galimų sprendimų iš skirtingų perspektyvų bei priimti sprendimus atsižvelgiant į galimas jo pasekmes. Viena iš svarbiausių sprendimų priėmimo sąlygų šiuo atveju yra rizikos sumažinimas. Sprendžiant keletą praktinių operacijų tyrimų problemų (ekologijos srityje, užtikrinant gyvybės saugą ir kt.), Būtina išanalizuoti situacijas, kuriose susiduria dvi (ar daugiau) kariaujančios šalys, siekiančios skirtingų tikslų, o kiekvienos šalies veiksmų rezultatas priklauso nuo priešo pasirinktos veiksmų eigos. Tokios situacijos gali būti vadinamos konfliktinėmis situacijomis.

Žaidimo teorija yra matematinė konfliktinių situacijų teorija, kurios pagalba galima parengti rekomendacijas dėl racionalaus konflikto dalyvių elgesio. Kad būtų galima matematiškai analizuoti situaciją, neatsižvelgiant į antrinius veiksnius, sukuriamas supaprastintas, schematizuotas situacijos modelis, vadinamas žaidimu. Žaidimas žaidžiamas pagal tam tikras taisykles, kurios suprantamos kaip sąlygų sistema, reguliuojanti galimus žaidėjų veiksmų variantus; kiekvienos šalies turimos informacijos apie kitos šalies elgesį kiekis; Žaidimo rezultatas, į kurį veda kiekvienas duotas judesių rinkinys. Žaidimo rezultatas (laimėjimas ar pralaimėjimas) ne visada yra kiekybiškai įvertinamas, tačiau paprastai paprastai galima bent jau sąlyginai jį išreikšti skaitine verte.[42]

Žingsnis yra vieno iš žaidimo taisyklėse numatytų veiksmų pasirinkimas ir jo įgyvendinimas. Judesiai skirstomi į asmeninius ir atsitiktinius. Asmeninis žingsnis yra žaidėjo sąmoningas vieno iš galimų veiksmų krypčių pasirinkimas ir jo įgyvendinimas. Atsitiktinis žingsnis yra pasirinkimas iš daugybės galimybių, kurias daro ne žaidėjo sprendimas, o tam tikras atsitiktinis atrankos mechanizmas (monetos mėtymas, kortos pasirinkimas iš sumaišytos kaladės ir pan.). Kiekvienam atsitiktiniam judesiui žaidimo taisyklės nustato galimų rezultatų tikimybių pasiskirstymą. Žaidimą gali sudaryti tik jų asmeniniai judesiai arba tik atsitiktiniai judesiai, arba abiejų derinys. Kita pagrindinė žaidimo teorijos koncepcija yra strategija. Strategija a priori yra žaidėjo priimtų sprendimų („if-then“ tipo) sistema, kurią jis laikosi žaidimo metu, kuri gali būti pateikta algoritmo pavidalu ir vykdoma automatiškai.

Žaidimo teorijos tikslas yra parengti rekomendacijas dėl racionalaus žaidėjų elgesio konfliktinėje situacijoje, t. Y. Nustatyti „optimalią strategiją“ kiekvienam iš jų. Strategija, kuri yra optimali vienai metrikai, nebūtinai bus optimali kitoms. Žinant šiuos apribojimus ir todėl aklai

nesilaikant žaidimo metodais gautų rekomendacijų, vis dar galima protingai naudoti matematinį žaidimų teorijos aparatą, kad būtų parengta, jei ne tiksliai optimali, tai bent jau „priimtina“ strategija.

Žaidimai gali būti klasifikuojami pagal žaidėjų skaičių, strategijų skaičių, žaidėjų sąveikos pobūdį, laimėjimų pobūdį, ėjimų skaičių, informacijos būklę ir kt. [4, 57]. Priklausomai nuo žaidėjų skaičiaus, skiriami dviejų ir n žaidėjų žaidimai. Pirmasis iš jų yra labiausiai tiriamas. Trijų ar daugiau žaidėjų žaidimai yra mažiau tiriami dėl kylančių esminių sunkumų ir techninių galimybių gauti sprendimą.Priklausomai nuo galimų strategijų skaičiaus, žaidimai skirstomi į „baigtinius“ ir „begalinius“.Žaidimas vadinamas baigtiniu, jei kiekvienas žaidėjas turi tik baigtinį strategijų skaičių, ir begalinis, jei bent vienas iš žaidėjų turi begalinį strategijų skaičių.

Pagal sąveikos pobūdį žaidimai skirstomi į ne koalicinius žaidimus: žaidėjai neturi teisės sudaryti susitarimų ar sudaryti koalicijų; Koalicija (kooperatyvas) – gali prisijungti prie coalitions.In bendradarbiavimo žaidimų, koalicijos yra iš anksto nustatytos.

Pagal laimėjimų pobūdį žaidimai skirstomi į: nulinės sumos žaidimus (bendras visų žaidėjų kapitalas nesikeičia, bet yra perskirstomas tarp žaidėjų; visų žaidėjų laimėjimų suma yra lygi nuliui) ir ne nulinės sumos žaidimus.

Pagal laimėjimo funkcijų tipą žaidimai skirstomi į: matricą, bimatrix, nepertraukiamą, išgaubtą ir kt.

Matricos žaidimas yra nulinės sumos galutinis žaidimas tarp dviejų žaidėjų, kuriame 1 žaidėjo išmokėjimas pateikiamas matricos pavidalu (matricos eilutė atitinka 1 žaidėjo taikomos strategijos numerį, stulpelis yra žaidėjo taikomos strategijos numeris, matricos eilutės ir stulpelio sankirtoje yra 1 žaidėjo laimėjimai, atitinkantys taikomas strategijas).

Bimatrix žaidimas yra baigtinis žaidimas tarp dviejų žaidėjų, turinčių ne nulinę sumą, kurioje kiekvieno žaidėjo laimėjimai nustatomi matricomis atskirai atitinkamam žaidėjui (kiekvienoje matricoje eilutė atitinka 1 žaidėjo strategiją, stulpelis atitinka 2 žaidėjo strategiją, eilutės ir stulpelio sankirtoje pirmoje matricoje yra žaidėjo 1 laimėjimas, antroje matricoje – žaidėjo laimėjimas).

Nuolatinis žaidimas yra tas, kuriame kiekvieno žaidėjo laimėjimo funkcija yra tęstinė. Įrodyta, kad šios klasės žaidimai turi sprendimus, tačiau nebuvo sukurta jokių praktinių metodų jiems rasti.

1.3. Išvados

Buvo sukurti priimtini sprendimo būdai, susidedantys iš grynai optimalios strategijos (tam tikro skaičiaus) suradimo sprendimui ir tikimybių taikyti grynai optimalias strategijas. Tokia užduotis yra gana lengva išspręsti. Išnagrinėjus optimalius sprendimo metodus pasirinktas tiesinis uždavinių sprendimo metodas kaip pagrindinis ir geometrinis kaip papildantis.

2. TEORINĖ ANALIZĖ

2.1. Uždavinio kelio analizė

2002 m. T. Geisler ir T.W. Manikas publikuotame darbe ”Autonomous robot navigation system using a novel value encoded genetic algorithm” atlieka kelio paieškos tyrimą su įvairiomis algoritmo kombinacijomis bei parametrais[20]. Eksperimentiniu būdu nustačius optimalius operatorius ir parametrus mažose erdvėse, kelią planuojantis algoritmas išbandytas didesniuose lygiuose. Tyrimo metu nustatyta, jog algoritmas teikia geriausius rezultatus mažose bei vidutinio dydžio (iki 10 eilučių bei 20 stulpelių) erdvėse.[20] 2004 m. Publikuotame darbe ”Autonomous Local Path Planning for a Mobile Robot Using a Genetic Algorithm” nagrinėjama lokalios kelio paieškos tema. Šio darbo autorius patobulino ankstesnių tyrimų genotipą, turėjusį informacijos tik apie kelio lokaciją bei kelio kryptį. Ši kodavimo technika apribojo kelio planavimą tik eilutės arba stulpelio atžvilgiu vienu metu. Šią problemą išsprendęs publikuoto darbo autorius gavo geresnius už prieš tai tyrimų grupės gautus rezultatus. Kelio planavimas yra glaudžiai susijęs su nagrinėjama problema, kadangi kliūčių ruožo žaidime visų pirma reikia surasti kliūčių vengiantį kelią iki finišo.[53]

Įvairių žaidimų uždavinių sprendinio kokybė vertinama pagal du parametrus: Euklidinį atstumą iki finišo, apskaičiuojamą formule

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad (1)$$

bei sprendinio ilgį: trumpesnis sprendinys yra tinkamesnis už ilgesnį. Tinkamumui paskaičiuoti neretai naudojama sparčiai auganti funkcija siekiant padidinti tinkamesnių sprendinių parinkimo tikimybę. Tai įrodė ir 1996 m. Konferencijos publikacija tema ”Fitness Function Design for Genetic Algorithmsin Cost Evaluation Based Problems”. Anot publikacijos autorių, eksponentiškai augančios funkcijos tyrimų metu pademonstravo geresnius rezultatus už tiesinio augimo funkcijas statinių funkcijų grupėje. Eksponentinė sprendinių vertinimo strategija konverguodavo greičiau taip atitinkamoje generacijoje gaudama geresnius sprendinius už tiesinę sprendinių vertinimo strategiją [38]. Priklausomai nuo žaidimo lygio plokštesnė tinkamumo funkcijos forma gali optimizuoti uždavinio sprendimą situacijose, kuomet sprendinys konverguoja per anksti. Tokiu atveju nelieka duomenų, reikalingų uždaviniui išspręsti. Atsižvelgę į tinkamumo vertinimo strategijos tema atliktus tyrimus, darbą pradėsime naudodami eksponentinę tinkamumo strategiją.

Maksimalaus srauto uždavinys bei dualus jam minimalaus pjūvio uždavinys intensyviai nagrinėjami jau daugiau kaip tris dešimtmečius. Danzigo [11] simplekso metodas transportavimo uždaviniui, paskelbtas 1951 metais, sprendė maksimalaus srauto problemą kaip atskirą atvejį. Neužilgo Fordas ir Falkersonas [17] maksimalaus srauto uždaviniui spręsti pasiūlė didinančių

grandinių (angl. Augmenting path) metodą. Dinicas [12, 13] bei Edmondsas ir Karpas [14] įrodė, kad šio metodo sudėtingumas yra polinominis. Efektyvesni algoritmai maksimaliam srautui rasti yra pagrįsti blokuojančio srauto (angl. Blocking flow) ir priešsraučio koregavimo (angl. Push-relabel) metodais. Pirmąjį blokuojančio srauto metodą pasiūlė Dinicas [12]. Karzanovas [30] buvo pirmasis, kuris blokuojančių srautų radimą suformulavo kaip atskirą uždavinį ir jo sprendimui pasiūlė naudoti priešsraučius. Priešsraučio koregavimo metodą, įeinantį į Goldbergo algoritmą [23], galutinai išvystė Goldbergas ir Tarjanas [24].

Problema, kaip rasti maksimalų srautą tinkle, apskritai turėjo sprendimą, panašų į tuo metu žinomą transporto problemą, Simplex metodą Danzigas modifikavo 1951 m., Specialiai šiai problemai. Algoritmas nebuvo polinominis, kaip ir bet kuris simplex metodas. Tačiau ypatingas problemos tipas leido daryti prielaidą, kad jai išspręsti gali būti naudojami paprastesni ir greitesni algoritmai. Pirmasis toks algoritmas, nepanašus į simpleksinį metodą, buvo sukurtas Fordo ir Fulkersono 1956 m. [17]. Šio algoritmo veikimo laikas nebuvo polinominis, tačiau jo idėja ir matematinis pagrindimas sudarė kokybinės srauto teorijos pagrindą. Pasiūlytame metode buvo panaudota „papildomų kelių“ idėja.

2.2. Tinklai. Pagrindiniai apibrėžimai

Tinklas – tai orientuotas grafas *G(V,E)* be kilpų su išskirtu atitinkančiu įėjimu ir išėjimu – *I* ir *S*. Kiekvienam lankui priskirti naturalūs skaičiai $c(v_i;v_j)$ – lanko pralaidumas. Srautas tinkle nurodo kaip kai kurie objektai siunčiami arba perkeliami iš vienos grafo viršūnės į kitą, tam tikra lanko kryptimi. Objektų skaičius $f(v_i;v_j)$ perduodamas išilgai lanko $(v_i;v_j)$ negali būti didesnio pralaidumo negu $c(v_i;v_j)$ šio lanko: $0 \leq f(v_i;v_j) \leq c(v_i;v_j)$. Vadinasi jei egzistuoja lankas iš v_i į v_j tai nėra lanko iš v_j į v_i.[9]. Maksimalaus srauto problemos formulavimas tinkle, kuriame yra tam tikra lanko talpa, suformuojamas maksimalaus dydžio srautas F_{max} tarp įėjimo ir išėjimo. Šis srautas užtikrinamas kiekviename lanke $(v_i;v_j)$ turimo dydžio $f(v_i;v_j)$ perduodamo srauto vertę kiekviename lanke [37].

Maksimalaus srauto uždavinys tinkle turi atitikti šias sąlygas:

- ✓ lanko srautų, išeinančių iš tinklo šaltinio, suma nesikeičia ir turi būti lygi lanko srautų, patenkančių į srautą, sumai $\sum_{(I,v_i)\in E} f(I,v_i) = \sum_{(v_j,S)\in E} f(v_j,S)$;
- ✓ viršūnei v , kuri nėra nei įėjimas nei išėjimas, t.y. $v\neq I$, $v\neq S$, kiekis srauto vienetų, patenkančių į viršūnę, skaičius turi būti lygus srauto vienetų, išeinančių iš viršūnės, skaičiui (srauto išsaugojimas) $\sum_{(v_i,v)\in E} f(v_i,v) = \sum_{(v,v_j)\in E} f(v,v_j)$;
- ✓ maksimalus srautas kelyje nuo įėjimo *I* į išėjimą *S* nustatomas tuo lanku $(v_i;v_j)$, kuris turi mažiausią arba minimalią talpą iš visų tam keliui priklausančių lankų[30].

Lankas $(v_i; v_j)$ vadinamas prisotintu, jei jo pralaidumo galimybės $c(v_i; v_j)$ lygios srautui $f(v_i; v_j)$ einančiam per jį, o bet koks kelias į kurį jis įtrauktas bus vadinamas prisotintu keliu. Srautas vadinamas prisotintu, jei bet kuris kelias iš *I* į *S* turi lanką $(v_i; v_j)$, kuriam $f(v_i; v_j) = c(v_i; v_j)$. Pirmoji maksimalaus srauto problemos sprendimo dalis yra sočiųjų srautų suradimas. Bet sotusis srautas ne visada yra maksimalus. Srautas tinkle bus maksimalus tik tada kai jo dydis bus didesnis už bet kokius kitus srautus tame tinkle.[1]

Uždaviniai susije su srauto tinkluose teorija pristato vieną iš bazinių problemų tyriant operacijas ir inžinerijoje. Yra daug praktinių srauto teorijos pritaikymų. Tikslas – ištirti pagrindinias sąvokas ir algoritmus naudojamus srauto teorijoje. Taip pat nustatyti efektyviausią algoritmą ir suformuoti programos struktūrą. Todėl atkreipiame dėmesį į Fordo – falkersono algotitmą[17]. Šis algoritmas leidžia rasti maksimalų srautą iš šaltinio į imtuvą pagal tam tikrą tinklo konfigūraciją. Jei konfigūracija pasikeičia, sukuriama nauja gija. Tada, kuriant tinklą, galima numatyti, kad jis įgyvendins tam tikrus srautus, atitinkančius tam tikrą tinklo konfigūraciją, tarp daugelio galimų srautų, einančių iš pradžios į pabaigą.. Srautai gali skirtis ne tik tuo, kaip prijungti tinklo elementai, bet ir tuo, kiek šių elementų yra prijungti. Nustatant optimalią konfigūraciją, būtina naudoti maksimalaus bendro srauto arba didžiausio vidutinio srauto kriterijų. Programa veikia dialogo režimu su vartotoju ir yra pagrįsta moduliniu principu. Programa nurodo maksimalų tinklo viršūnių skaičių, kuris yra 40. Yra naudojami šie duomenis: tinklo mazgų matricą, tinklo mazgų koordinates, elementų numeriai. Moduliai jungiami pagal parametrus – duomenis ir bendrą duomenų bloką. Toks ryšys gaunamas, kai modulis gauna ir perduoda visus reikalingus duomenis kaip skambučio parametrus, šie duomenys yra paprasti kintamieji.

2.3. Maksimalaus srauto tinklas

Maksimalaus srauto tinkle problema buvo tiriama daugiau nei 60 metų. Susidomėjimą ja skatina didžiulė praktinė šios problemos svarba. Problemos sprendimo metodai naudojami transporto, ryšių ir elektros tinkluose, modeliuojant įvairius fizikos ir chemijos procesus, kai kuriose matricų operacijose, sprendžiant susijusias grafikų teorijos problemas.

Prieš šešiasdešimt metų šią problemą išsprendė simplex, naudojant linijinį programavimą, kuris buvo labai neefektyvus. „Ford“ ir „Falkerson“ pasiūlė apsvarstyti nukreiptą tinklą, kad išspręstų maksimalaus srauto problemą ir ieškotų sprendimo naudodami iteracinį algoritmą. Nepaisant to, kad buvo sukurta daug algoritmų ir metodų, kurie yra efektyvesni už „Ford-Falkerson“ metodą, vis dėlto jų metodo naudojimas sprendžiant šią problemą vis dar duoda teigiamą rezultatą priimtinu laiku[17].

Prieš svarstydami „Ford-Falkerson“ metodo algoritmą, išplėskime anksčiau naudotus žymėjimus. Kraštui *(i, j)* naudojame žymėjimą (C_{ij}, C_{ji}), kad pavaizduotume pralaidumą kryptimis *i*

→j ir j→i atitinkamai. Siekiant išvengti nesusipratimų tinklo C diagramoje, mes patalpinsime kraštą (i,j) arčiau mazgo C_{ij} ir C_{ji} arčiau mazgo kaip parodyta 1pav.

1 pav. Srauto pralaidumo galimybių[17]

Šio algoritmo idėja yra r 2 pav. Srauto pralaidumo galimybių[17] ıtais iš įėjimo į išėjimą. Apsvarstykime briauną *(I, J)*su pradine pralaidumo savybe (C_{ij}, C_{ji}). Algoritmo veikimo procese dalis pralaidumo savybių paima srautai einantis per briauną, rezultate ko kiekviena briauna turės likutinį pralaidumą. Naudosime užrašą (c_i, c_j) likutinių pralaidumo savybių vaizdavimui. Tinklas, kuriame visi kraštai turi likutinį pralaidumą, vadinamas likutiniu tinklu.

Laisvam mazgui *j* gaunančiam srautą iš mazgo *i* priskirsim žymėjimą a_h, kur *a* – srauto dydis tekantis iš mazgo *i* į mazgą *j*. Maksimalaus srauto nustatymo algoritmas pagal Ford-Falkerson metodą apima šiuos veiksmus [17]:

- Žingsnis 1.Visoms briaunoms *(i, j)* likutinį pralaidumą padarysime lygiu pradiniam pralaidumui, t.y. prilyginsime $(C_{jp}, C_{jj})=(C_p, C_{ji})$. Priskirsim *a=00* ir pažimyme mazga *1*su koordinatėm [00, -]. Tarkime, kad *i=1* ir pereiname prie sekančio žingsnio.
- Žingsnis 2. Apibrėžiame dauguma 5 kaip mazgų rinkinį, į kurį galima pereiti iš mazgo *i* briauna su teigiamu liekamuoju pralaidumu (t.y. $C_n > 0$ visiems *j*, priklausantiems S), mazgas *j* nėra pažymėtas dabartinėje iteracijoje. Jei S_t nėra tuščioji dauguma, atlikite trečiąjį veiksmą, kitaip pereikite prie 4 veiksmo.
- Žingsnis 3. Erdvėje S_t, randame mazgą *L* tokį, kuris lygus *{c y}* čia *y*, priklausantis S_j. Tarkime = *c,k* ir mazgą *k* pažymime *[a^ I]*. Jei paskutinis žymėjimas yra pažymėtas išėjimo mazgas (t.y. jei *k = n*), tai rastas ištisinis kelias ir pereiname prie penktojo žingsnio, Kitu atveju *i = k*, tai pereiname prie 2 žingsnio..
- Žingsnis 4(grįžimas atgal). Jei *I = 1* tai kelias neįmanomas, ir todėl pereiname prie 6 žingsnio. Jei *i* nėra lygus *1*, randame pažymėtą mazgą *r* šalia esančio mazgo *i*(nustatome pagal žymėjimą *[ah r]*) ir pašaliname mazgą *i* iš mazgų gausos, esančio šalia mazgo *r*. Sprendžiame kad *I=r* ir grįžtame prie antro žingsnio. Grįžimas atgal šiame etape įvykdomas tada kai algoritmas turi pašalinti tarpinį mazgą iki to momento kai bus įdiegtas kelias nuo vieno galo iki kito galo.
- Žingsnis 5(nustatykite likutinį tinklą). Pažymėkime N_p = *(1, k, ..., rr)* mazgų rinkinį, per kurį rastas kelias nuo galo iki galo eina iš šaltinio mazgo (mazgas *1)* į išėjimo

mazgą (mazgas *p)*. Tada maksimalus srautas, einantis per tą kelią, apskaičiuojamas kaip

$$f_p = \min\{a_1, a_{k1}, a_{k2}, \dots, a_n\}. \quad (2)$$

Liekamoji briaunos pralaidumo galia, liekamojo kelio perėjimas, mažėja dydžiu f_p srauto kryptimi ir padidinama tuo pačiu kiekiu priešinga kryptimi. Todėl krašto (*i, j)*, kuris yra perėjimo kelio dalis, dabartinės grynosios balansinės vertės ($C_{p,}\, c_u$) pasikeis taip[23, 30]:

$$(c_{ij} - f_p, c_{ji} + f_p), jei\ srautas\ iš\ mazgo\ i\ į\ mazgą\ j, \quad (3)$$

$$(c_{ij} + f_p, c_{ji} - f_p), jei\ srautas\ iė\ mazgo\ j\ į\ mazgą\ i. \quad (4)$$

Tada atkuriame visus mazgus, ištrintus atliekant 4 veiksmą. Tarkime, kad *i = 1* ir grįžtame prie antrojo žingsnio, kad randame naują kelią nuo galo iki galo.

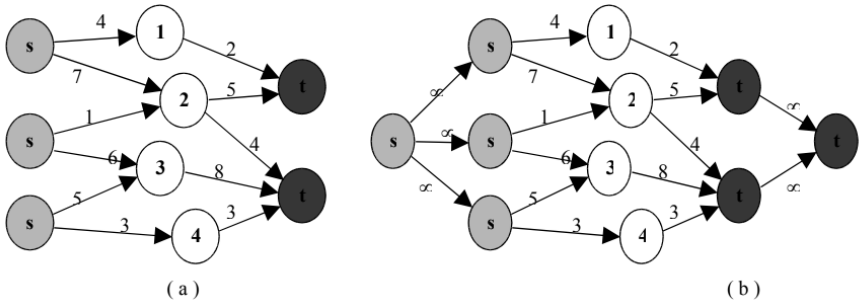
- Žingsnis 6(sprendimas). Jei yra *m* rastųištisynių kelių maksimalus srautas apskaičiuojamas pagal formulę

$$F = f + /2 + \cdots + /. \quad (5)$$

Atsižvelgiant į briaunos pradinio (C_P C_n) ir galutinio (c_{ip} c_p) pralaidumo *(I,J)* vertes, optimalų srautą per tą briauną galime apskaičiuoti sekančiu būdu. Tarkime *(a, b) =* ($C_u - C_p$ C_p- c_p). Jei *a> O*, srautas per briauną (*I,J*) yra *a*. Jei *b > 0,* srautas yra lygus *b*. Atkreipiame dėmesį, kad atvejis, kai *a> 0* ir *b > 0* tuo pačiu metu, yra neįmanomas.

2.3.1. Maksimalaus srauto uždavinys

Maksimalaus srauto uždavinys yra rasti didžiausią srautą duotajam tinklui *G* su šaltiniu *s* ir nuotakiu *t*. Maksimalaus srauto apskaičiavimo uždavinys yra tiesinio programavimo uždavinys, todėl jam gali būti taikomi tiesinio programavimo uždavinių sprendimo metodai. Maksimalaus srauto uždavinys gali turėti ne vieną, o kelis šaltinius ir nuotakius(2 pav.):



3 pav. Tinklas su keliais šaltiniais ir nuotakiais[30]

Maksimalaus srauto uždavinys su keliais šaltiniais (sakykime, jų yra *m*) suvedamas į uždavinį su vienu šaltiniu galime pridėti superšaltinį s ir lankus (*s, si*), kiekvienam *i = 1, 2, ..., m*. Lankams (*s, si*) suteikiamas pralaidumas *c (s, si) = ∞.* Tuos pačius veiksmus atliekame ir nuotakiaims (2 pav, b). Srautas 2 pav., (a) pavaizduotu tinklu atitinka srautą 2 pav. (b) pavaizduotu tinklu ir atvirkščiai.

2.4. Algoritmų teorinis pagrindimas

Algoritmas yra viena iš pagrindinių informatikos ir matematikos sąvokų. Kartu su modeliavimo, algoritmizacija – yra vienas iš kompiterinių technologijų mokslo metodas. Algoritmai yra sisteminis mokslo tyrimų objektas šiuolaikinėje matematikoje, kurioje tyriamos bendrosios algoritmų savybės. Šie tyrimai yra riba tarp matematikos ir informatikos, o taip pat greta matematinės logikos.[44]

Matematikos encklopedinis žodynas[50] pateikta štai tokia algoritmo sąvokos formuluotė: „Algoritmas(algorythm) – tikslus eiliškumas, kuris apibrėžtas skaičiavimo procesu(šiuo atveju vadinamas algoritminiu), kuris prasideda pradiniais duomenimis(iš tam tikrų galimų pradinių duomenų duotam algoritmui) ir yra skirti gauti pilnai šiais duomenimis pagrįstą rezultatą.“

Šioje algoritmo formuluotėje algoritmo sąvoka „skaičiavimo procesas“ neriekėtu suprasti siaura prasme, tik skaitmeniniai simboliai ir jų deriniai. Algoritmai gali veikti su atsitiktiniais dydžiais, simboliais ir jų deriniais, pvz. jau mokykloje matematikos panokose yra sprendžiami uždaviniai su raidinėmis reikšmėmis. Su logoritmo sąvoką yra labai glaudžiai susijusi sąvoka „algoritmo vykdytojas“. Čia vykdytojas gali būti vienas asmuo ar žmonių grupė, robotas, kompiuteris, programavimo įranga ir kt. Vykdytojai veikia formaliai, jie tik griežtai vykdo algoritmo komandas nekreipdamas jokio dėmesio į užduoties turinį[5]. Tuo pat metu vykdytojas gauna norimą rezultatą. Tai yra viena iš svarbiausių algoritmų savybių. Todėl pagal[37] nagrinėjamą logoritmą galima apibūdinti septiniais parametrais:

- ✓ Visuma galimų pradinių duomenų.
- ✓ Visuma galimų rezultatų.
- ✓ Visuma galimų tarpinių rezultatų.
- ✓ Tiesioginio apdirbimo taisyklė.
- ✓ Užbaigimo taisyklė.
- ✓ Rezultato gavimo taisyklė.

Algoritmo sąvoka, bendrąja jos forma, yra viena iš pagrindinių sąvokų matematikoje, kuri nepripažįsta paprastesnių sąvokų termonuose. Bandyme apibūdinti algoritmą t.y. formalizuoti jį, yra parašyta labai daug mokslinių darbų bei straipsnių. Bet daugmaž nustatyta terminologija susijusi tik su tam tikra sritimi, kurioje ir yra naudojama ši sąvoka. Tuo pat metu kiekvienas patikslinimas algoritmo sąvokos susideda iš ankščiau jau minėtų 7 algoritmo parametrų tiksliai aprašo tam tikrą sritį, kurioje šis parametras gali keistis[3].

Verta pastebėti, kad normaliniai algoritmai gali skirtis vienas nuo kito abėcėle ir pakaitine sistema. Tais atvejais kai svarstymo objektai leidžia pristatimą žodžių pavidalą kai kuriose abėcėlėse, normaliniai algoritmai yra veiksmingas įrankis tyrimuose, kur reikalaujama tiksli algoritmo sąvoka.

Maksimalaus srauto tinkle problemą galima suformuluoti taip: yra tam tikras neorientuotas grafas, pavyzdžiui, miesto kelių ir gatvių tinklas, kuriuo eismas teka iš šaltinio (eismo pradžios) į srautą (eismo pabaigos tašką). Kiekvienas tinklo kraštas turi savo svorį. Miesto kelių tinklo atveju tai gali būti kelio plotis arba jo pajėgumas, t. Y. Transporto priemonių skaičius per laiko vienetą. Didžiausia tam tikro tinklo srauto problema yra rasti maksimalų transporto priemonių, judančių nuo viršūnės pradžios(šaltinio) iki grafo galutinės viršūnės(nutekėjimo) skaičių[30].

Tuo pačiu metu esant tokiam uždaviniui būtina nustatyti nagrinėjamo tinklo srautų intervalines vertes. Tai leidžia atsižvelgti į pradinių duomenų neapibrėžtumą, o tai padidina priimto sprendimo patikimumą.[18] Šį uždavinį galima išspręsti bendru linijinio programavimo metodu. Arba yra dar vienas metodas, kurį pasiūlė Lesteris Randolphas Fordas ir Delbertas Ray Fulkersonas, ir šis metodas pagrįstas žymėjimu. Naudojant šį metodą atsirado keletas veiksmingų algoritmų. Galime peržiūrėti Fordo – Fulkersono algoritmo įgyvendinimą maksimaliam srautui rasti. Algoritmas prasideda nuo nulinio srauto, o tada kiekvienoje iteracijoje padidėja srautas tinkle. Kiekviename žingsnyje ieškoma tinklo, kuris padidina srauto kiekį[17]. Tuo pačiu metu srautas didėja išilgai šios grandinės lankų, kol jis tampa prisotintas. Tokia didėjanti grandinė yra grandinė nuo šaltinio iki nutekėjimo, kurios visi lankai yra leidžiami. Norint padidinti tinklo srauto vertę tam tikru kiekiu *Q,* būtina padidinti srautą *Q* tuo pačiu kiekiu kiekviename suderintame grandinės lanke ir sumažinti jį kiekviename nesuderintame lanke. Fordas ir Fulkersonas įrodė, kad srautas tinkle, kuriam neįmanoma sukurti didėjančios grandinės, yra maksimalus. Norėdami rasti tokią didėjančią grandinę, naudojamas ženklinimo metodas. Ženklinimo procesas prasideda tinklo šaltinyje ir baigiasi tinklo nutekėjime. Didėjanti grandinė nuo šaltinio iki nutekėjimo egzistuoja, jei nutekėjimas yra paženklintas. Etiketės taikomos tinklo viršūnėms, kuriose yra reikalinga informacija, kad būtų galima atkurti tinklą ir nustatyti, kokiu kiekiu srautas tinkle gali būti pakeistas. Tokiu atveju viršūnė gali būti vienoje iš šių būsenų: nepažymėta, pažymėta ir peržiūrėta. Žymėjimo metodas prasideda nuo nekontroliuojamo srauti. Tada bandoma gauti didesnio dydžio srautą. Skaičiavimas baigiasi, kai gaunamas didžiausias srautas. Algoritmas susideda iš nuoseklios visų galimų kelių paieškos nuo N_S mazgo iki N_t mazgo, kurie padidina srautą. Mazgai gauna specialius ženklus, nurodančius kryptį, kuria galima padidinti tam tikrą lanko srautą. Suradus kelią, kuris padidina srautą, nustatomas maksimalus šio kelio pajėgumas. Tada srautas padidinamas šia suma, o visi mazgų ženklai ištrinami. Prasideda naujų mazgų žymų išdėstymas pagal naujai gautą srautą. Algoritmas susideda iš dviejų etapų.

- ✓ 1 etapas – mazgai pažymimi, kiekvienas mazgas yra vienoje iš trijų būsenų: pažymėtas ir peržiūrėtas, pažymėtas ir neperžiūrėtas arba nepažymėtas. Pradžioje visi mazgai nepažymėti. Laisvojo mazgo N_j žymėjimas visada susideda iš dviejų dalių. Pirmoji dalis yra indeksas *j* mazgo N_t, kuris rodo, kad galima „išsiųsti“ srautą iš N_t į N_j. Antroji pastabos dalis yra

skaičius, nurodantis maksimalią srauto liniją, kurią galima „išsiųsti“ iš N_S šaltinio į N_j nepažeidžiant lanko pralaidumo apribojimų. Mazgams suteikiami žymėjimai kurie yra gretutiniai pažymėtiems ir neperžiūrėtiems mazgams, kol bus pažymėtas N_t mazgas arba neįmanoma bus daugiau pažymėti nei vieno mazgo, o N_t nutekėjimas liks nepažymėtas. Jei N_t negali būti pažymėtas, tai neegzistuoja kelio iš N_S į N_t padidinančio srautą, todėl sumodeliuotas srautas yra maksimalus. Jei N_t pažymėtas, tai 2 žingsnyje galima rasti kelią, kuris padidina srautą.

- ✓ 2 etapas – srauto pakeitimas. Etapai 1 ir 2 kartojasi iki tol kol srauto didinimas tampa neįmanomu. Aptariamas algoritmas naudojamas srautams modeliuoti įvairiose srityse.[18]

Maksimalaus srauto tinkle problema buvo tiriama daugiau nei 60 metų. Susidomėjimą ja skatina didžiulė praktinė šios problemos svarba. Problemos sprendimo metodai naudojami transporto, ryšių ir elektros tinkluose, modeliuojant įvairius fizikos ir chemijos procesus, kai kuriose matricų operacijose, sprendžiant susijusias grafikų teorijos problemas ir net ieškant tgeb grupių *WWW*.

2.4.1. Algoritmų pagrindinės savybės

Kad algoritmas atitiktų savo paskirtį, jis turi turėti šias pagrindines savybes [14, 30, 31, 35]:

- ✓ Efektyvumas. Reikia nurodyti objektų rinkinį, su kuriuo algoritmas veiks. Formalizuotas (užkoduotas) šių objektų vaizdavimas vadinamas šaltinio duomenimis. Algoritmas pradeda dirbti su tam tikru įvesties duomenų rinkiniu, kuris vadinamas įvesties duomenimis ir dėl savo darbo jis sukuria duomenis, vadinamus išvestimi. Algoritmo tikslas yra gauti konkretų rezultatą, turintį tam tikrą ryšį su šaltinio duomenimis.
- ✓ Tikrumas. Norėdami išspręsti problemą, kiekvienas algoritmo žingsnis turi būti aiškiai apibrėžtas ir neleisti savavališkai interpretuoti.
- ✓ Masyvumas. Tas pats algoritmas gali būti naudojamas daugeliui tam tikros klasės problemų, kurios skiriasi duomenimis, išspręsti.
- ✓ Diskretiškumas. Algoritmas atliekamas baigtine žingsnių seka, t.y. algoritmas sumažina problemos sprendimą iki atskirų paprastesnių problemų sprendimo.
- ✓ Efektyvumą. Algoritmas turi būti vykdomas pakankamai ribotu laiku, o ne tik baigtiniu laiku.
- ✓ Galutinių. Problemos sprendimas būtinai gaunamas baigtiniu žingsnių skaičiumi, kai veikia pagal algoritmą. Tuo pačiu metu sukuriamas begalinis procesas, kuris susilieja su norimu sprendimu. Procesas tam tikru momentu sustoja. Gauta vertė laikoma apytiksliu nagrinėjamos problemos sprendimu, kurio tikslumas priklauso nuo žingsnių skaičiaus.

- ✓ Kompaktiškumas. Ši savybė reiškia algoritmo pateikimo glaustumą. Praradus kompaktiškumą, algoritmas praranda teisę egzistuoti dideliu mastu.

2.4.2. Algoritmų tipai

Pagal savybes ir įgyvendinimo metodus galima išskirti šiuos algoritmų tipus.

Tiesinis algoritmas – tai nurodymų rinkinys, kuriame visi veiksmai atliekami nuosekliai vienas po kito vieną kartą tokia tvarka, kokia jie yra. Linijinis programavimas yra matematinis aparatas, skirtas išspręsti optimalias problemas su tiesinėmis išraiškomis optimalumo kriterijui ir linijiniams kintamųjų diapazono apribojimams. Norint išspręsti įvairias linijinio programavimo problemas, yra beveik universalus algoritmas – simplex metodas, leidžiantis rasti optimalų sprendimą daugumai problemų ribotame iteracijų skaičiuje. Naudojamų apribojimų tipas (lygybė ar nelygybė) neturi įtakos algoritmo taikymui. Nereikia atlikti papildomų optimalumo bandymų gautiems sprendimams. Paprastai praktinėms linijinio programavimo problemoms būdingas labai daug nepriklausomų kintamųjų. Todėl jiems išspręsti paprastai naudojami kompiuteriai, kurių reikiamą galią lemia sprendžiamos problemos matmuo[2].

Išsišakojimo algoritmas – tai algoritmas, kuriame yra blokas tam tikrai sąlygai (loginei išraiškai) patikrinti. Priklausomai nuo būklės patikrinimo rezultato, keičiasi algoritmo žingsnių vykdymo seka, t.y. vykdoma tam tikra komandų seka. Tokiu atveju pasirenkate vieną iš dviejų ar daugiau galimų variantų. Tokią eigą galima užrašyti.

Ciklinis algoritmas yra algoritmas, apibūdinantis tų pačių skaičiavimų kartojimą skirtinguose duomenų variantuose, kad būtų gautas norimas rezultatas. Reikiamus skaičiavimus atlieka pagrindinis kilpos blokas – kilpos korpusas. Ciklinį procesą organizuoja pagalbiniai ciklo blokai. Jie nustato pradinę vertę, naujas duomenų reikšmes ir patikrina ciklinio proceso pabaigos sąlygą.

Norėdami nustatyti bet kurį ciklą, turite atlikti šiuos veiksmus[34]:

- ✓ Nustatyti pradines kilpos parametrų reikšmes prieš prasidedant kilpai.
- ✓ Pakeisti kilpos parametrus prieš kiekvieną paskesnį ciklo pakartojimą.
- ✓ Patikrinti sąlygas, kad kilpa kartotų arba baigtų ciklą.
- ✓ Eiti į kilpos pradžią, jei ji nebaigta, arba išeiti iš kilpos kilpos pabaigoje.

Pagal skaičiaus pakartojimų nustatymo metodą ciklai skirstomi į:

- ✓ ciklus su nežinomu pakartojimų skaičiumi;
- ✓ ciklai su aiškiu pakartojimų skaičiumi.

Pavyzdžiui, C ++ programavimo programoje ciklu su nežinomu skaičiumi pasikartojimų yra komanda *while* ir iki...*wkile*, o ciklo su užduotu skaičiumi pasikartojimų – *for*.

2.4.3. Grafinis algoritmų aprašymas

Grafiniame algoritmų aprašyme naudojamos grafinių simbolių sekos, kurios yra tarpusavyje susijusios ir atlieka tam tikras funkcijas. Grafinių vaizdų konfigūraciją, sąrašą ir dydį, taip pat

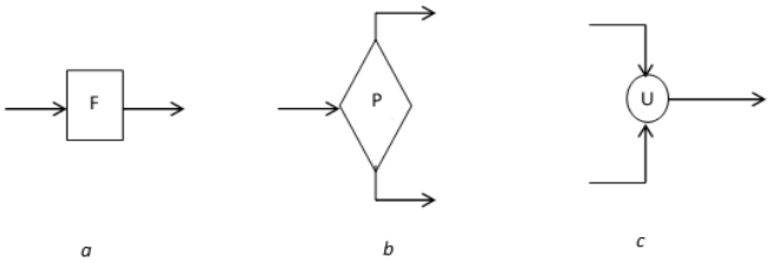
algoritmų schemų kūrimo taisykles. 1 lentelėje pateikti pagrindiniai simboliai, naudojami algoritmams apibūdinti [33, 29, 5]

1 lentelė. Pagrindiniai simboliai algoritmui apibūdinti

Simbolis	Simbolio pavadinimas
	Algoritmo pradžia–pabaiga. Informacijos apdorojimo proceso nutraukimas. Simbolis turi vieną įvestį ir vieną išvestį.
	Duomenys. Duomenų įvedimas ir išvedimas. Nešėjas nėra apibrėžtas. Simbolis turi vieną įvestį ir vieną išvestį.
	Procesas. Bet kokios rūšies duomenų tvarkymas. Konkrečios operacijos ar operacijų grupės atlikimas, dėl kurio pasikeičia informacijos reikšmė, forma ar išdėstymas. Simbolis turi vieną įvestį ir vieną išvestį.
	Sprendimas. Jungiklio tipo funkcija. Jis turi vieną įvestį ir keletą alternatyvių išėjimų, iš kurių tik vieną galima suaktyvinti apskaičiavus tame simbolyje apibrėžtas sąlygas. Simbolis turi vieną įvestį ir keletą alternatyvių išėjimų.
	Paruošimas. Komandos ar komandų grupės modifikavimas, siekiant paveikti kai kurias vėlesnes funkcijas (programos inicijavimas, indekso modifikavimas ir kt.). Simbolis turi vieną įvestį ir vieną išvestį.
	Dokumentas. Rodo suvestinės duomenis lengvai skaitoma forma. Rezultato informaciją perduokite į spausdintuvą. Simbolis turi vieną įvestį ir vieną išvestį

Grafinis algoritmo aprašymas vadinamas schema. Struktūrinė schema yra nukreiptas grafikas, nurodantis algoritmo komandų vykdymo tvarką. Grafiko viršūnės, parodytos pav. 3 gali būti vieno iš trijų tipų [5]:

- ✓ Funkcinė viršūnė (F). Ši viršūnė turi vieną įvestį ir vieną išvestį.
- ✓ Predikato viršūnė (P). Ši viršūnė turi vieną įvestį ir du išėjimus. P funkcija gali perduoti valdymą išilgai vienos iš šakų, priklausomai nuo P reikšmės (t t.y. true, TIESA arba f t.y. false, MELAS). Dažnai vietoj t rašoma „taip“ arba „“, o vietoj f rašoma „ne“ arba „-„.
- ✓ Vienijanti viršūnė („susiliejimo“ viršūnė) (U). Tai užtikrina valdymo perdavimą iš vieno iš dviejų įėjimų į išvestį.



4 pav. Blok – schemose naudojamos viršūnės yra šios: *a* – funkcinė viršūnė, *b* – predikato viršūnė, *c* – sąjungos viršūnė[5]

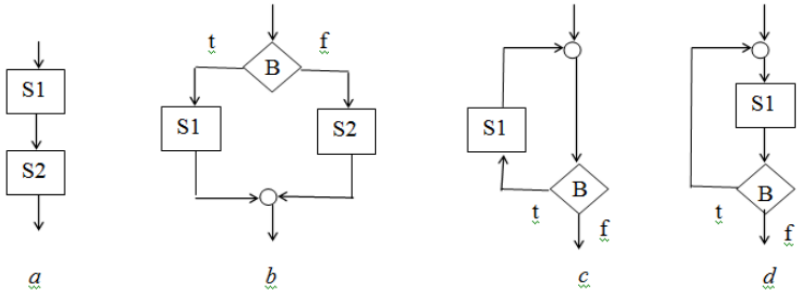
Funkcinė viršūnė naudojama funkcijai $F:X \rightarrow Y$.

Predikato viršūnė naudojama funkcijai (predikatui) $P: X \rightarrow \{T,F\}$, t. Y. Loginės išraiškos, kuri os perduoda valdymą vienai iš dviejų šakų.

Sąjungos viršūnė reiškia kontrolės perkėlimą iš vieno iš gaunamų filialų į vieną iš išeinančių filialų. 3 paveiksle pavaizduotos elementariosios schemos naudojamos kuriant pagrindines schemas, kurios yra ypač svarbios algoritmavimo praktikai:

- ✓ sekanti arba kompozicija (4 a pav.);
- ✓ išsišakojimas arba alternatyva (4 b pav.);
- ✓ ciklas arba iteracija (4 c, d pav.).

Šių pagrindinių struktūrinių schemų naudojimas pagal Bem–Jacopini teoremą [33, 5] gali sudaryti struktūrinę schemą.



5 pav. Pagrindinės struktūrinės schemos, naudojamos blokinėse struktūrinėse schemose, yra šios:
a – sekantis, b – išsišakojimas,
c – ciklas su išankstine sąlyga,
d – kilpa su postsąlyga[33]

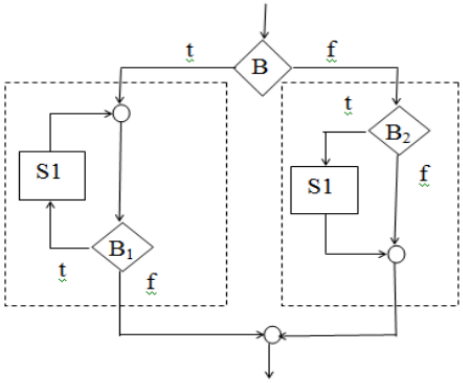
Kiekviena iš struktūrinių schemų, parodytų pav. 4 yra lygiavertis konkrečiam bet kurios struktūrinės programavimo kalbos teiginiui. Pavyzdžiui, C^{++} kalba blok – schema pavaizduota

- ✓ pav. 4 a – priskyrimo operatorių seka,
- ✓ pav. 4 b – operatorius *if (B) S1; else S2;* ,
- ✓ pav. 4 c – operatorius *while(B) S1;*,

✓ Pav. 4 d – operatorius *do S1; while(B);*.

Kur *B* aiškinama kaip Bulio loginė išraiška, o *S1* ir *S2* – kaip programinės įrangos operatoriai.

Dažnai naudojamas sutrumpintas operatorius *if (B) S1*; kurio blok – schema skiriasi nuo blok – schemos 4 pav.. jei nėra *S2* bloko. Svarbus šių struktūrinių blok – schemų bruožas yra tas, kad jos turi vieną įvestį ir vieną išvestį. Iš to išplaukia, kad bet kuri iš jų sudaryta struktūrinė blok – schema taip pat turės vieną įvestį ir vieną išvestį, pavz. Struktūrinę schemą, parodytą 5 pav.



6 pav. Struktūrinė blok – schema[33]

Jos programinis analogas programavimo kalboje turi išraišką:

if (B)

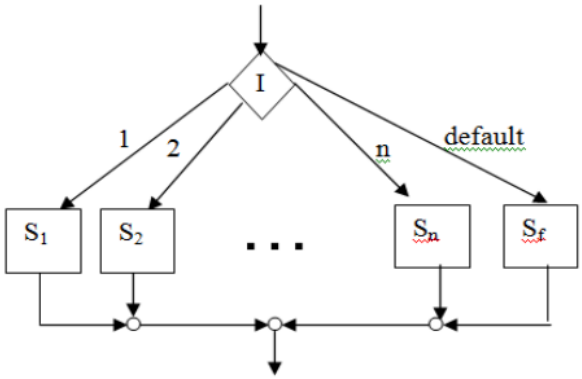
while (B1)

S1;

else if (B2)

S2;

Struktūrinis programavimas leidžia sukurti daugiau elementarių struktūrų, nei siūlo Böhmas ir Jacopini. Taip yra dėl to, kad beveik visos programavimo kalbos apibrėžia jungiklio operatorių, kuris turi daug šakų. Operatoriaus jungiklio algoritminė struktūra parodyta 6 pav..



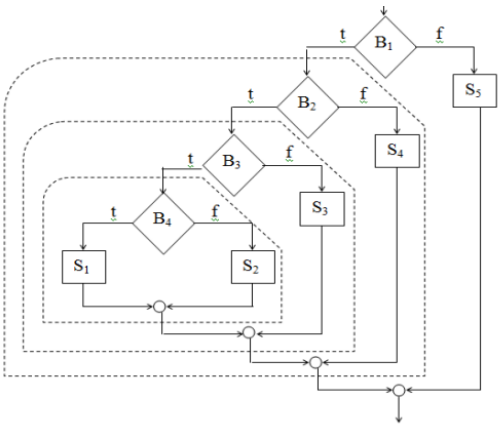
7 pav. Operatoriaus jungiklio algoritminė struktūra[33, 5]

Jos programinis analogas programavimo kalboje turi išraišką:

```
switch(I)
}
    case 1: S1;
    case 2: S2;
    . . .
    case n: Sn;
   default:
            Sf;
}
```

Operatoriuje *switch* leidžiama nenaudoti bloko *Sf*. Tokiu atveju, jei išraiškos *I* su reikšmėmis neatitinka etikečių, nurodytų po tarnybinio žodžio *case*, operatorius switch ignoruojamas.

Kuriant schemą algoritmui apibūdinti, reikia laikytis šios taisyklės. Pirma, įgyvendinama schema, kurioje yra dideli programos blokai. Tada kiekvienam dideliam blokui struktūrinio programavimo metodikoje atliekamas tolesnis detalizavimas. Žingsnis po žingsnio algoritmo skaidymo į mažesnes dalis procesas vadinamas programavimu „iš viršaus į apačią“ [33]. Struktūrinis programavimas „iš viršaus į apačią“ yra programavimo procesas „iš viršaus į apačią“, kuris apsiriboja struktūrinių schemų naudojimu.



8 pav. Trijų lizdų gylio struktūrinė bloko schema [33]

Iš 7 pav. Matyti, kad struktūrizuotoje programoje valdymo struktūra yra medis. Dėl šios priežasties tokios programos vadinamos medžio struktūros programomis.

Pažymėtina, kad šis algoritmų kūrimo metodas negarantuoja klaidų nebuvimo programoje. Pavyzdžiui, patirtis rodo, kad jei valdymo struktūrų įterpimo gylis yra didesnis arba lygus trims, klaidos tikimybė labai padidėja.

Terminas „programavimas be goto“ dažnai naudojamas kaip struktūrinio programavimo aprašymas. Struktūrinis programavimas nedraudžia naudoti goto teiginio, tačiau goto pareiškimo naudojimas daro programos tekstą mažiau suprantamą.

2.4.4. Fordo – Falkersono algoritmas

Jei tinkle susidaro tam tikras srautas tai pasinaudojus Fordo – Falkersono teorema galima pasakyti ar srautas bus maksimalus ar ne. Ši teorema parodo, kad maksimalus srautas tinkle yra lygus maksimaliam pralaidumui visose atkarpose[17]

$$F_{max} = C_{min}\left(\frac{A}{B}\right) \qquad (6)$$

Pagrindinės šio logoritmo sąvokos yrs trys:

- ✓ liekamieji tinklai yra sudaryti iš tų lankų, kuriais galima padidinti srautą;
- ✓ papildomi keliai;
- ✓ pjūviai.

Liekamųjų tinklų formalės apibrėžimas toks: tegul $G = (V,E)$ – tinklas, turintis pradžios viršūnę s ir pabaigos viršūnę t. Tegul f – srautas tame tinkle. Kiekvienai viršūnių porai u ir v apibrėžkime liekamąjį pralaidumą iš u į v taip:

$$c_f(u,v) = c(u,v) - f(u,v) \qquad (7)$$

Ši pralaidumo išraiška nusako, kiek dar srauto galima nukreipti iš u į v. Pvz., jei $c(u,v) = 16$, o $f(u,v) = 11$, tai lanku (u,v) dar galima papildomai perduoti $c_f(u,v)=5$ vienetus srauto. Liekamasis pralaidumas $c_f(u,v)$ gali būti didesnis už $c(u,v)$, jei duotu momentu $f(u,v)$ yra neigiamas. Pavyzdžiui, jei $c(u,v) = 16$, o $f(u,v) = -4$, tai $c_f(u,v) = 20$. Tada galima srautą padidinti 4, pašalinus priešinį srautą, ir dar nukreipus 16 vienetų, neviršijant lanko (u,v) pralaidumo. Taigi tinklas $G_f = (v,E_f)$, kur $E_f = \{ (u,v) \in V \times V : c_f(u,v) > 0 \}$, vadinamas tinklo G liekamuoju tinklu.

Papildomi keliai dar kitaip vadinami didinančios grandinės yra kelias iš pradinės viršūnės s į paskutinę viršūnę t liekamuoju tinklu G_f. Iš liekamojo tinklo apibrėžimo išplaukia, kad visais papildomo kelio lankais (u,v) galima perduoti kažkiek srauto, neviršijant nei vieno lanko pralaidumo. Imame, kad f – srautas tinkle $G = (V,E)$. Didžiausias srautas, kurį galima perduoti didinančiąja grandine p, vadinamas liekamuoju kelio p pralaidumu $c_f(p) = \min \{ c_f(u,v) : (u,v) \in p \}$.

Tinklo G = (V,E) pjūviu (cut) vadinamas toks aibės V padalinimas į dvi dalis S ir $T = V / S$, kad $s \in S$, o $t \in T$. Pjūvio (S,T) pralaidumu vadinama pjūvį kertančių lankų pralaidumų suma $c(S,T)$. Srautui f apibrėžiamas srauto dydis per pjūvį (S,T) – tai suma $f(S,T)$ pagal pjūvį kertančius lankus. Maksimalus pjūvis tai pjūvis su didžiausiu pralaidumu, o minimalus – su mažiausiu pralaidumu.

Maksimalaus srauto uždavinys Fordo – Falkersono metodu [17] realizuojamas pažingsniui. Pradžioje laikoma, kad srautas lygus nuliui. Kiekvienu sekančiu žingsniu srautas didinamas. Tai atliekama randant didinančiąją grandinę, kuria galima praleisti dar truputį srauto, ir ją panaudojant. Šios iteracijos atliekamos tol, kol įmanoma surasti nors vieną didinančiąją grandinę. Metodą galima užrašyti taip[17]:

laikyti, kad srautas f = 0

while *egzistuoja papildomas kelias p*

do *papildyti f keliu p*

return *f*

Fordo – Falkersono procedūros darbo trukmė priklauso nuo to, kaip ieškomas kelias *p* (4-ji eilutė)[17]. Teoriškai algoritmas gali iš viso nesustoti, jei srauto prieaugis vyks vis mažėjančiais žingsneliais, taip ir nepasiekiant maksimumo. Vienok jeigu didinančiąją grandinę parinkinėti naudojant paieškos platyn algoritmą [1], tai algoritmas dirbs polinominį laiką.

2.4.5. Genetinis algoritmas

Kas gi yra tas genetinis algoritmas? Genetinis algoritmas(GA) yra paieškos ir optimizavimo algoritmas, pagrįstas biologiniu natūralios atrankos principu.[27]

Genetiniai algoritmai yra skirti optimizavimo ir modeliavimo problemoms spręsti, nuosekliai atrenkant, derinant ir keičiant norimus parametrus, naudojant mechanizmus, primenančius biologinę evoliuciją. Genetiniai algoritmai yra skirti optimizavimo problemoms spręsti. Tokios užduoties pavyzdys yra neuro tinklo mokymas, t.y. tokių svorių verčių parinkimas, kai pasiekiama minimali paklaida. Tuo pačiu metu genetinis algoritmas grindžiamas atsitiktinės paieškos metodu[15].

Iš biologijos žinome, kad bet kurį organizmą gali atstovauti jo fenotipas, kuris iš tikrųjų lemia, kas yra objektas realiame pasaulyje, ir genotipas, kuriame yra visa informacija apie objektą chromosomų rinkinio lygiu. Tuo pačiu metu kiekvienas genas, t.y. genotipo informacijos elementas, turi savo fenotipo atspindį. Tokiu būdu uždavinių sprendimui mums reikia pristatyti kiekvieną objekto bruožą tokia forma, kuri būtų tinkama naudoti genetiniame algoritme. Visas tolesnis genetinio algoritmo mechanizmų veikimas atliekamas genotipo lygiu, todėl galima daryti be informacijos apie objekto vidinę struktūrą, kas ir lemia jo platų taikymą įvairiose užduotyse.

Dažniausiai pasitaikančiame genetinio algoritmo tipų genotipui pavaizduoti naudojamos objekto bitų eilutės. Tuo pačiu metu kiekvienas fenotipo objekto atributas atitinka vieną objekto genotipo geną. Genas – bitų eilutė, dažniausiai fiksuoto ilgio, kuri parodo to bruožo vertę[52].

2.4.5.1. Kodavimas

Norint užkoduoti tokias funkcijas, galima naudoti paprasčiausią parinktį – šios funkcijos bitų vertę. Tada būtų labai lengva naudoti tam tikro ilgio geną, kurio pakaktų visoms galimoms tokio bruožo reikšmėms pavaizduoti. Tačiau, deja, toks kodavimas turi savų trūkumų[35].

Pagrindinis trūkumas yra tas, kad gretimi skaičiai skiriasi kelių bitų reikšmėmis, pavyzdžiui, skaičiai 7 ir 8 bitų vaizde skiriasi 4 pozicijomis, todėl genetiniam algoritmui sunku veikti ir padidėja laikas, per kurį jis susilieja. Siekiant išvengti šios problemos, geriau naudoti kodavimą, kuriame gretimi skaičiai skiriasi mažesniu pozicijų skaičiumi, idealiu atveju – vieno

bito verte. Toks kodas yra Gray kodas, kurį tikslinga naudoti įgyvendinant genetinį algoritmą. Gray kodų vertės aptariamos toliau pateiktoje lentelėj

2 lentelė. Dešimtainių kodų (kairėje) ir pilkų kodų (dešinėje) atitikimas[45]

Dešimtainis kodas	Dvejetainė reikšmė	Šešioliktainė reikšmė	Dešimtainis kodas	Dvejetainė reikšmė	Šešioliktainė reikšmė
0	0000	0h	0	0000	0h
1	0001	1 val.	1	0001	1 val.
2	0010	2 val.	3	0011	3 val.
3	0011	3 val.	2	0010	2 val.
4	0100	4 val.	6	0110	6 val.
5	0101	5 val.	7	0111	7 val.
6	0110	6 val.	5	0101	5 val.
7	0111	7 val.	4	0100	4 val.
8	1000	8 val.	12	1100	Ch
9	1001	9 val.	13	1101	Dh
10	1010	Ah	15	1111	Šv.
11	1011	Bh	14	1110	Eh
12	1100	Ch	10	1010	Ah
13	1101	Dh	11	1011	Bh
14	1110	Eh	9	1001	9 val.
15	1111	Šv.	8	1000	8 val.

Taigi, koduodami sveikojo skaičiaus funkciją, ją padalijame į tetradus ir transformuojame kiekvieną Gray kodą pagal tetradus[45].

Praktiškai įgyvendinant genetinius algoritmus, paprastai nereikia konvertuoti bruožų verčių į genų vertes. Praktiškai yra atvirkštinė problema, kai būtina nustatyti atitinkamo bruožo vertę pagal geno vertę.Tokiu būdu užduotis dekoduoti genų vertę, kuriem atitinka sveikojo skaičiaus bruožai, yra traviali.

Pats paprasčiausiais kodavimo būdas, kuris yra paviršiuje būtu panaudoti bitų įvaizdį, nors toks variantas turi tokius pat trūkumus kaip ir kodavimas sveikiems skaičiams. Todėl praktikoje paprastai taikoma tokia veiksmų seka:

1. Padalijamas visas leistinųjų įverčių intervalas į tam tikras dalis reikiamu tikslumu.
2. Suteikia tamtikrą vertę genai kaip sveikąjį skaičių apibrėžianti intervalo skaičių(panaudojant Gray kodą).
3. Parametro vertė yra skaičius, kuris yra šio intervalo vidurys[45].

Koduodami ne skaitinius duomenis, pirmiausia turime juos konvertuoti į skaičius.

Norint nustatyti objekto fenotipą (t.y. objektą apibūdinančių bruožų reikšmes), reikia žinoti tik tas genų kurios atitinka šias vertes, t.y. objekto genotipą. Šiuo atveju genų rinkinys, apibūdinantis objekto genotipą, yra chromosoma. Kai kuriuose diegimuose jis taip pat vadinamas asmeniu. Tokiu budu įgyvendinanat genetinį algoritmą chromosoma yra fiksuoto ilgio bitų eilutė. Tačiau genas atitinka kiekvieną linijos dalį. Genų ilgis chomosomoje gali būti vienodas arba skirtingas. Dažniausiai naudojami to paties ilgio genai.

2.4.5.2. Pagrindiniai genetiniai algoritmai

Kaip žinome, evoliucijos teorijoje svarbų vaidmenį atlieka tai, kaip tėvų bruožai perduodami jų palikuonims. Genetiniuose algoritmuose operatorius, vadinamas kryžminiu veisimu, yra atsakingas už tėvų bruožų perdavimą palikuonims. Šis operatorius nustato tėvų bruožų perdavimą palikuonims. Tai veikia taip[15, 20,38]:

- ✓ iš populiacijos išrenkami du asmenys, kurie bus tėvai;
- ✓ nustatomas lūžio taškas (paprastai atsitiktinai);
- ✓ palikuonis apibrėžiamas kaip pirmojo ir antrojo tėvų dalies sujungimas.

Panagrinėsim šio operatoriaus funkcionavimą 8 pav.

Chromosome_1	0000000000
Chromosome_2	1111111111

9 pav. Dvi chromosomos[42]

Tarkime lūžis įvyksta po 3-čiojo chromosomų bito, kas tada atsitinka parodyta 9 pav.

Chromosome_1	0000000000	>>	000	1111111	Resultant_chromosome_1
Chromosome_2	1111111111	>>	111	0000000	Result_chromosome_2

10 pav. Chromosomos po lūžio[42]

Tada, esant 0,5 tikimybei, viena iš gautų chromosomų yra identifikuojama kaip palikuonis.

Kitas genetinis operatorius yra skirtas išlaikyti įvairovę populiacijoje. Tai vadinama mutacijos operatoriumi. Su šiuo operatoriumi kiekvienas bitas chromosomoje yra apverstas tam tikra tikimybe.

Be to, taip pat naudojamas vadinamasis inversijos operatorius, kuris susideda iš to, kad chromosoma dalijasi į dvi dalis, o tada jie keičia vietas. Schematiškai tai galima pavaizduoti taip(10 pav.):

000	1111111	>>	1111111	000

11 pav. Chromosomos dalijimasis į dvi dalis[42]

Pagrinde genetinio algoritmo funkcionavimui pakanka dviejų genetinių operatorių, tačiau praktiškoje naudojami kai kurie papildomi operatoriai arba šių operatorių modifikacijos. Pavyzdžiui, kryžminimas gali būti ne vieno taško (kaip aprašyta aukščiau), bet kelių taškų, kai susidaro keli lūžio taškai (dažniausiai du). Be to, kai kuriuose algoritmo diegimuose mutacijos operatorius yra tik vieno atsitiktinai pasirinkto chromosomos bito inversija.

2.4.5.3. Genetinio algoritmo funkcionavimo schema

Žinodami kaip interpretuoti genų reikšmes, galime panagrinėti genetinio algoritmo veikimo aprašymus ir genetinio algoritmo veikimo schemą klasikinėje versijoje[38]:

1. Inicijuoti pradinį laiko momentą $t = 0$. Atsitiktiniu būdu suformuoti pradinę populiaciją kuri susideda iš *k* asmenų $B_0 = A_1, A_2, \dots, A_k$.
2. Apskaičiuoti tinkamumą kiekvieno asmens $F_{A_i} = fit(A_i), i = 1 \dots k$ ir populiacija $F_t = fit(B_t)$. Šios funkcijos reikšmės parodo koks asmens sutapimas su chromosoma sprendžiant uždaviniu.
3. Rinktis asmenį A_C iš populiacijos $A_C = Get(B_t)$.
4. Su tam tikra tikimybe(crosoverio tikimybe P_C)išrinkti antrą asmenį iš populiacijos $A_{C_1} = Get(B_t)$ ir atlikti operatoriaus crosingą $A_C = crossing(A_C, A_{C1})$.
5. Su tam tikra tikimybe(mutacijos tikimybe P_m) atlikti operatoriaus mytaciją $A_C = mutation(A_C)$.
6. Su tam tikra tikimybe(inversijos tikimybe P_i) atlikti operatoriaus inversiją $A_C = inversion(A_C)$.
7. Gautą chromosomą įterpti į naują populiaciją $insert(B_{t+1}, A_C)$.
8. Įvykdyti operacijas, pradedant nuo 3 punkto *k* kartų.
9. Padidinti esamos epochos numerį $t = t + 1$.
10. Jei sustojimo sąlyga įvykdyta, darbas užbaigiamas, kitaip perinama į 2 punktą .

Dabar pažvelkime į atskirus algoritmo etapus.Svarbiausią vaidmenį sėkmingame algoritmo funkcionavime atlieka tėvų chromosomų atrankos etapas 3 ir 4 žingsniai. Tokiu atveju galimi įvairūs variantai. Dažniausiai naudojamas atrankos būdas vadinamas rulete. Taikant šį metodą, chromosomų atrankos tikimybę lemia jos tinkamumas t.y. $P_{GetA_i} = Fit(A_i)/Fit(B_t)$. Šio metodo naudojimas lemia tai, kad padidėja tikimybė, kad bruožai bus perduoti labiau pritaikytiems asmenims palikuonims.

Kitas dažniausiai naudojamas metodas yra turnyrų atranka. Ji susideda iš to, kad atsitiktinai atrenkami keli asmenys iš gyventojų (paprastai 2), o nugalėtoju išrenkamas asmuo, labiausiai prisitaikantis[41].

Be to, kai kurie algoritmo diegimai naudoja vadinamąją elitizmo strategiją, o tai reiškia, kad asmenys, turintys didžiausią tinkamumą, garantuoja perėjimą į naują populiaciją. Elitizmo naudojimas paprastai leidžia pagreitinti genetinio algoritmo konvergenciją. Elitinės strategijos naudojimo trūkumas yra tas, kad ji padidina tikimybę, kad algoritmas pasieks lokalinį minimumą.

Kitas svarbus dalykas yra išjungimo kriterijų apibrėžimas. Paprastai jie yra arba maksimalaus algoritmo veikimo epochų skaičiaus apribojimas, arba jo konvergencijos nustatymas, paprastai lyginant gyventojų tinkamumą keliose epochose ir sustojant, kai šis parametras stabilizuojamas.

2.4.5.4. Genetinio algoritmo taikymas sprendinio paieškai

Genetinis algoritmas sėkmingai naudojama modeliuoti ūkio subjektus, turinčius ribotą racionalų elgesį: finansinį prognozavimą, investicijas ir kt. Viena iš įdomių užduočių yra portfelio optimizavimas. Žaidimų teorijoje genetinis algoritmas irgi naudojamas, pavyzdžiui, pabėgimo dilemai išspręsti. Robotikoje - puikiai naudojama valdyti humanoidinį robotą ir optimizuoti maršruto planavimą. Aviacijoje tai skirta skrydžio valdymo sistemai. Kalbant apie aviaciją, „General Electric“ ir „Rensselaer“ politechnikos instituto mokslininkai panaudojo genetinį algoritmą, kad sukurtų reaktyvinio variklio turbiną, kuri naudojama šiuolaikiniuose lėktuvuose. Pirmuoju atveju tinkamumo funkcija gali nustatyti per tam tikrą laiką pagamintų dalių skaičių, o antruoju atveju ji gali „nubausti“ prieštaringas tvarkaraščio šakas. Kūrybiškumo galimybės yra neribotos[41].

Pagrindinis genetinių algoritmų pranašumas programoje yra tas, kad jie ieško pasaulinio optimalaus sprendimo. Populiariausi optimizavimo algoritmai pasirenka pradinį sprendimą, kuris vėliau keičiasi viena ar kita kryptimi. Taigi, gauname gerą padalijimą, bet ne visada optimalų. Rekombinacijos ir mutacijos operatoriai leidžia gauti sprendimus, kurie nėra panašūs į pradinius. Matematiniems optimizavimo uždaviniams spręsti taikomas genetinis algoritmas, kuris padeda surasti atstumui, iki finišo ir įėjimo, įvertinti optimaliausią sprendimą. Geriausi sprendiniai turi didžiausią galimybę būti atrinkti rekombinacijai bei perduoti vaikui savo genetinę informaciją.

Tinkamiausi tėvai bei sukurti vaikai sudaro naujos generacijos populiaciją. Algoritmas vykdomas kol sukuriamas nurodytas generacijų skaičius.

Genetinio algoritmo privalumai. Šitas algoritmas turi unikalių privalumų[38]:

- ✓ siūlo keletą sprendimų, iš kurių galima rinktis, o ne tik vieną;
- ✓ nagrinėja kelis taškus vienu metu, o ne vieną po kito, kad objektyvi funkcija nepatektų į vietinį ekstremumą;
- ✓ optimizavimas vyksta nuolat kintančiomis aplinkos sąlygomis, o gyventojai turi prisitaikyti; gali pasiūlyti patenkinamą NP-hard sunkios problemos sprendimą;
- ✓ leidžia lygiagrečiai skaičiuoti;
- ✓ jis tinka spręsti problemas, susijusias su įvairiais parametrais (svarbiausia yra nustatyti tinkamą tinkamumo funkciją).

Genetinio algoritmo trūkumai:

- ✓ „tik geras sprendimas“ taip pat kartais yra trūkumas;
- ✓ daug taškų paieškos erdvėje taip pat kartais yra trūkumas;
- ✓ ne visada patogu galvoti apie genų ir chromosomų problemą.

2.5. Žaidimų matricos

Paanalizuokima baigtinį žaidimą, kuriame pirmasis žaidėjas *A* turi *m* strategijų, o antrasis žaidėjas *B* turi *n* strategijų. Toks žaidimas vadinamas $m \times n$ žaidimu. Pažymėkime strategijas $A_1, A_2, ..., A_m$; ir $B_1, B_2, ..., B_n$. Tarkime, kad kiekviena pusė pasirinko tam tikrą strategiją: A_i arba B_j. Jei žaidimas susideda tik iš asmeninių judesių, tada strategijų pasirinkimas vienareikšmiškai lemia žaidimo rezultatą – vienos iš pusių pergalę a_{ij}. Jei žaidime, be asmeninių atsitiktinių judesių, tada strategijų poros A_i ir *B* laimėjimai yra atsitiktinis kintamasis, kuris priklauso nuo visų atsitiktinių judesių rezultatų. Šiuo atveju natūralus tikėtino laimėjimo įvertinimas yra tikėtina atsitiktinio laimėjimo vertė, kuri taip pat žymima a_{ij}.[53]

Tarkime, kad mes žinome a_{ij} reikšmes kiekvienai strategijų porai. Šios vertės gali būti parašytos stačiakampio lentelės (matricos) pavidalu, kurios eilutės atitinka A_i strategijas, o stulpeliai – B_j strategijas. Tada apskritai matricos žaidimą galima parašyti pagal šią mokėjimo matricą 11 pav:

	B_1	B_2	...	B_n
A_1	a_{11}	a_{12}	...	a_{1n}
A_2	a_{21}	a_{22}	...	a_{2n}
...	...	...	...	...
A_m	a_{m1}	a_{m2}	...	a_{mn}

12 pav. bendras vaizdas mokėjimo matricos matricos žaidime[53]

A_i yra 1 žaidėjo strategijos pavadinimas, B_j yra 2 žaidėjo strategijos pavadinimas, a_{ij} yra 1 žaidėjo laimėjimo vertė, kai jis pasirenka *i-ąją* strategiją, o žaidėjas 2 yra *j-oji* strategija. Kadangi šis žaidimas yra nulinės sumos žaidimas, 2 žaidėjo laimėjimo vertė yra priešinga žaidėjo 1 laimėjimo vertės ženklui.

Kiekvienas žaidėjas stengiasi maksimaliai padidinti savo laimėjimą, atsižvelgdamas į priešingo žaidėjo elgesį. Todėl 1 žaidėjui būtina nustatyti minimalias laimėjimų vertes kiekvienoje iš strategijų ir tada rasti maksimalią iš šių verčių, t.y. nustatyti vertę

$$V_H = max_i\ min_j\ a_{ij} \quad (8)$$

arba rasime minimalias kiekvienos mokėjimo matricos eilutės reikšmes ir nustatysime didžiausią iš tų verčių. V_H vertė vadinama matricos maksiminu arba mažesne žaidimo kaina. Žaidėjo strategija, atitinkanti maksiminui V_H, vadinama maksimino strategija. Akivaizdu, kad jei laikysimės maksimalios strategijos, tada mums garantuojamas ne mažesnis nei V_H laimėjimas bet kuriame priešininko elgesyje. Todėl V_H vertė yra garantuotas minimumas, kurį galime pasiekti laikydamiesi atsargiausios strategijos. Pagal matricos žaidimą žaidėjo praradimo vertė yra lygi žaidėjo praradimo vertei žaidėjui2

$$V_H = max_i\ min_j\ a_{ij} \quad (9)$$

Arba galime rasti maksimalias kiekvieno mokėjimo matricos stulpelio reikšmes ir nustatyti mažiausią iš tų verčių. V_B vertė vadinama matricos minimumu, viršutine žaidimo kaina arba minimaksiniu laimėjimu. Priešininko strategija, atitinkanti pergalę, vadinama jo minimaksine strategija. Laikydamasis savo atsargiausios minimaksinės strategijos, priešininkas garantuoja, kad bet kuriuo atveju jis praras ne daugiau kaip V_B.

Tuo atveju, jei $V_н$ ir V_B vertės nesutampa, išlaikant žaidimo taisykles (a_{ij} koeficientus) ilgainiui, kiekvieno žaidėjo strategijų pasirinkimas pasirodo nestabilus. Jis įgyja stabilumą tik tada, kai $V_H = V_B = V$. Šiuo atveju sakoma, kad žaidimas turi sprendimą grynose strategijose, o strategijos, kuriose pasiekiamas *V*, turi optimalias grynas strategijas. *V* vertė vadinama žaidimo grynąja kaina [8].

Matricos grynose strategijose yra sprendimas 12 pav.. Šiuo atveju 1 žaidėjui optimali gryna strategija būtų A_1 strategija, o 2 žaidėjui B_4 strategija būtų optimali švari strategija.

	B_1	B_2	B_3	B_4	Min_j
A_1	17	16	15	14	**14**
A_2	11	18	12	13	11
A_3	18	11	13	12	11
Max_i	18	18	15	**14**	

13 pav. Mokėjimo matrica, kurioje yra grynų strategijų sprendimas[53]

13 pav. matricoje nėra sprendimo grynose strategijose, nes mažesnė žaidimo kaina pasiekiama strategijoje A_1 ir jos vertė yra 12, o viršutinė žaidimo kaina pasiekiama B_4 strategijoje ir jos vertė yra 13.

	B_1	B_2	B_3	B_4	Min_j
A_1	17	16	15	12	**12**
A_2	11	18	12	13	11
A_3	18	11	13	12	11
Max_i	18	18	15	**13**	

14 pav. Mokėjimo matrica, kurioje nėra sprendimo grynose strategijose[53]

Mokėjimo matricos tvarka (eilučių ir stulpelių skaičius) gali būti sumažinta pašalinant dominuojančias ir pasikartojančias strategijas. *K** strategija vadinama dominuojančia *K*** strategija, jei įvykdomas priešingo žaidėjo elgesio santykis.

$$A_{K^*} < A_{K^{**}} \qquad (10)$$

Čia A_{k^*} ir $A_{k^{**}}$ yra laimėjimų vertės, kai žaidėjas pasirenka atitinkamai K^* ir K^{**} strategijas. Tuo atveju, jei

$$A_{K^*} = A_{K^{**}} \quad (11)$$

Strategija K^* vadinama strategijos K^{**} dubliavimu**.** Pavyzdžiui, matricoje su dominuojančiomis ir besidubliuojančiomis strategijomis strategija A_1 dominuoja strategijos A_2 atžvilgiu, strategija B_6 dominuoja B_3, B_4 ir B_5 strategijų atžvilgiu, o strategija B_5 yra dubliuojama B_4 strategijos atžvilgiu.

	B_1	B_2	B_3	B_4	B_5	B_6
A_1	1	2	3	4	4	7
A_2	7	6	5	4	4	8
A_3	1	8	2	3	3	6
A_4	8	1	3	2	2	5

15 pav. Mokėjimo matrica su dominuojančiomis ir pasikartojančiomis strategijomis[53]

Šių strategijų žaidėjai nepasirinks, nes žinoma, kad jie yra pralaimėtojai, o šių strategijų pašalinimas iš mokėjimo matricos neturės įtakos apatinės ir viršutinės žaidimo kainos, aprašytos šioje matricoje, nustatymui. Nedominuojančių strategijų rinkinys, gautas sumažinus mokėjimo matricos matmenį, taip pat vadinamas Pareto rinkiniu.

2.6. Išvados

Algoritmų kūrimas yra programavimo proceso ir problemų sprendimo dalis. Gerai suprojektuotas, jis leidžia supaprastinti sudėtingumą, padidinti programų efektyvumą ir užtikrinti rezultatų patikimumą bei tikslumą.

Išnagrinėjus du algoritmus, Fordo – Folkersono ir Genetinį, padaryta išvada, kad būten maksimalaus srauto algoritmo ištrūkimo uždaviniams spręsti tinkamiausias yra Fordo – Folkersono.

3. UŽDAVINIŲ SPRENDIMO PROJEKTINĖ DALIS

3.1. Uždavinio sprendimo etapai

Sprendžiant bet kurį uždavinį naudojant kompiuterį, būtina atlikti šiuos veiksmus:

1. Problemos pareiškimas.
2. Užduoties įforminimas.
3. Algoritmo konstrukcija.
4. Programos sudarymas programavimo kalba.
5. Programos derinimas ir testavimas.
6. Ši etapų seka vadinama technologine problemų sprendimo grandine kompiuteryje.

Iš šio sąrašo 3, 4 ir 5 elementai yra tiesiogiai susiję su programavimu. Užduoties nustatymo etape aiškiai suformuluota, kas pateikiama ir ką reikėtų gauti sprendžiant problemą. Svarbus dalykas yra nustatyti visą šaltinių duomenų rinkinį. Neteisingai parinktas šaltinis arba reikalingi duomenys gali lemti rezultatus, kurie gali mūsų netenkinti. Todėl problemos nustatymo etape, pirma, būtina nustatyti ir išvardyti visus pradinius ir reikalingus duomenis ir, antra, nustatyti sąlygas, kuriomis galima gauti reikiamus rezultatus ir kuriomis tai nėra. Ir galiausiai turite nustatyti, kurie rezultatai bus laikomi teisingais. Taigi, aiškus užduoties formulavimas reiškia pradinių duomenų ištraukimą iš informacijos apie tiriamą reiškinį ar objektą ir nustatant, koks bus rezultatas. Pažymėtina, kad problemos pareiškimo tikslumas ir aiškumas yra pusė jo sprendimo sėkmės.

Problemos formalizavimo etape problema paprastai verčiama į matematinių formulių, lygčių ir santykių kalbą. Jei sprendimas reikalauja matematinio tam tikro realaus objekto, reiškinio ar proceso aprašymo, formalizavimas yra lygiavertis atitinkamam matematiniam modeliui.

Algoritmo kūrimo etape patyrę programuotojai paprastai iš karto rašo programas programavimo kalbomis. Tuo pačiu metu jie nenaudoja jokio algoritmų, pvz., Schemų, aprašymo būdo. Tačiau švietimo tikslais naudinga naudoti schemas ir tada išversti gautą algoritmą į programavimo kalbą. Algoritmo kūrimo procesą sudaro, pirma, informacijos įvedimo veiksmų parinkimas ir užsakymas ir, antra, skaičiavimų, kurie griežtai atitinka pasirinktus problemų sprendimo būdus, organizavimas. Kuriant algoritmą, naudojant žingsnis po žingsnio detalizavimo metodą, būtina užtikrinti, kad algoritmas būtų racionalus ir atitiktų visas savybes.

Programos kodo sudarymo programavimo kalba etape algoritmas verčiamas į programą. Programos rašymas užsienio programavimo kalba t.y. pats programavimas, esant problemų sprendimo algoritmams, yra algoritmų kodavimas programavimo kalba.

Programos derinimo ir testavimo etape problemos sprendimas yra kompiuterinio eksperimento atlikimas. Tokiu atveju būtina pakartotinai paleisti programą vykdymui pagal skirtingas pradinių sąlygų vertes. Nagrinėjant informacinį modelį programos forma programavimo aplinkoje, reikia atlikti šiuos veiksmus[30 maers]:

a) paleisti pasirinktą programavimo aplinką;

b) surinkti programos kodą;

c) išsaugokite šį kodą diske;

d) paleiskite programą, kad ją vykdytumėte.

Gautų rezultatų skaičiavimo ir analizės etape vykdoma sukurta ir suderinta programa, analizuojami gauti rezultatai ir, jei reikia, tiriamas matematinis modelis koreguojamas kartojant 2 – 5 etapus.

3.2. Kelių radimas

Dirbtinis intelektas daugiausia susijęs su subjekto veiksmų pasirinkimu pagal dabartines sąlygas. Tradicinėje dirbtinio intelekto literatūroje tai vadinama "intelektualaus agento" valdymu. Agentas paprastai yra žaidimo veikėjas, tačiau tai taip pat gali būti automobilis, robotas ar net kažkas abstraktesnio - visa subjektų grupė, šalis ar civilizacija. Bet kuriuo atveju tai yra subjektas, kuris stebi savo aplinką, ja remdamasis priima sprendimus ir veikia pagal tuos sprendimus. Tai kartais vadinama "Sense/Think/Act" ciklu:

- ✓ Suvokimas: agentas atpažįsta arba yra perduodamas informaciją apie aplinką, kuri gali turėti įtakos jo elgesiui (pvz., netoliese esantys pavojai, kolekcionuojami daiktai, lankytinos vietos ir pan.)
- ✓ Mąstymas: agentas nusprendžia, kaip reaguoti (pvz., nusprendžia, ar pakankamai saugu rinkti daiktus, ar kovoti, ar geriau pirmiausia pasislėpti)
- ✓ Veiksmas: agentas atlieka veiksmus, kad įgyvendintų savo sprendimus (pvz., pradeda judėti keliu į priešą ar objektą ir pan.). Tada dėl simbolių veiksmų situacija pasikeičia, todėl ciklas turi pasikartoti su naujais duomenimis.

Realaus pasaulio dirbtinio intelekto užduotys, ypač tos, kurios yra aktualios šiandien, yra linkusios sutelkti dėmesį į "suvokimą". Pavyzdžiui, savarankiškai važiuojantys automobiliai turi fotografuoti kelią priešais juos, derindami juos su kitais duomenimis (radaru ir lidaru) ir bandydami interpretuoti tai, ką mato. Paprastai šią užduotį išsprendžia mašininis mokymasis, kuris ypač gerai veikia su dideliais kiekiais triukšmingų realaus pasaulio duomenų (pvz., Kelio priešais automobilį nuotraukomis ar keliais vaizdo įrašo kadrais) ir suteikia jiems tam tikrą prasmę, išgaudamas semantinę informaciją, pvz., "Priešais mane yra dar vienas automobilis". Tokios užduotys vadinamos klasifikavimo problemomis. Žaidimai yra neįprasti tuo, kad jiems nereikia sudėtingos sistemos šiai informacijai išgauti, nes tai yra neatsiejama modeliavimo dalis. Nereikia vykdyti vaizdo atpažinimo algoritmų, kad aptiktumėte priešą priešais jus; Žaidimas žino, kad ten yra priešas ir gali perduoti šią informaciją tiesiogiai sprendimų priėmimo procesui. Todėl "suvokimas" šiame cikle paprastai yra labai supaprastintas, o visi sunkumai kyla realizuojant "mąstymą" ir "veikimą".

3.3. Ištrūkimo uždaviniai

Kaip įprasta galvoti kad ištrūkimo uždaviniai tai žaidimuose herojus(agentas) ieško kelio kaip išeiti apeinant įvairiausias kliūtis. Bet tai ne visai taip. Ištrūkimo uždavinys gali būti sprendžiamas kuriant failų skirstytoją, fotografijų tvarkytoją, užduočių įvairiausios programėlės ir t.t. Kiekvienam sprendimui reikalingas algoritmas. Algoritmų kūrimas yra pagrindinis programavimo proceso žingsnis. Jis apibrėžia veiksmų seką, kurią kompiuteris turi atlikti, kad išspręstų konkrečią užduotį arba gautų rezultatą.

Rezultatų failui pasirinkta kableliais atskirtų reikšmių (comma separated values, CSV) failo struktūra. CSV formatas yra pripažintas formatas, naudojamas perduoti duomenis tarp programų. Pakankamai daug programų, dirbančių su duomenimis, gali eksportuoti duomenis CSV formatu, ir jos visos taip pat gali priimti duomenis. CSV failo struktūros aprašymą galima rasti [16]. Rezultatų failo laukai:

- taktinis procesoriaus dažnis megahercais. Jis gali būti reikalingas, kai rezultatų failas kaupiamas, skaičiuojant srautą skirtingo pajėgumo kompiuteriais;
- tinklo tipas. Pildomas tik tinklams, kuriuos sugeneravo vidinis programos atsitiktinio tinklo generatorius;
- viršūnių skaičius. Visų viršūnių, įskaitant ir superšaltinį bei supernuotakį, skaičius;
- lankų skaičius;
- eilučių skaičius. Kvadratinio arba stačiakampio tinklo eilučių skaičius. Be superšaltinio ir supernuotakio;
- stulpelių skaičius. Kvadratinio arba stačiakampio tinklo stulpelių skaičius. Be superšaltinio ir supernuotakio;
- mazgų skaičius. Viršūnių skaičius be superšaltinio ir suprnuotakio;
- didžiausias lankų pralaidumas;
- viršūnių laipsnis;
- atsitiktinių skaičių generatoriaus parametras. Turi prasmę, kai norima dar kartą sugeneruoti tokį patį tinklą;
- įvairūs metodai, kurių trukmė milisekundėmis.

Naudojantis Fordo–Falkersono algoritmu, kiekviename žingsnyje laisvai pasirenkama didinančioji grandinė p ir srautas f didinamas dydžiu $cf(p)$. Toliau pateikiamas algoritmas einamosioms srauto reikšmėms saugoti naudoja masyvą $f[u,v]$. Laikoma, kad funkcija $c(u,v)$ paskaičiuojama per laiką $O(1)$.

for *kiekvienam lankui* $(u,v) \in E[G]$

do $f[u,v] \leftarrow 0$

$f[v,u] \leftarrow 0$

while *likusiame tinkle Gf egzistuoja kelias p iš s į t*

do *cf(p)← min { cf(u,v) : (u,v) ∈ p }*

for *kiekvienam kelio p lankui (u,v)*

do *f[u,v] ← f[u,v] + cf(p)*

f[v,u] ← - f[u,v]

1-3 eilutėse nustatomos pradinės srauto reikšmės, ciklas 4-8 eilutėse randa didinančiąją grandinę p grafe Gf ir padidina srautą f. Jei didinančiųjų grandinių nėra, reiškia rastas srautas yra maksimalus.

Gerai suprojektuotas, jis leidžia programuotojams supaprastinti sudėtingas užduotis, padidinti kompiuterinių programų efektyvumą.

Išnagrinėkime pavyzdį, kaip rasti didžiausią skaičių masyve. Turime atlikti šiuos žingsnius:

1. Supratimas: turime rasti didžiausią skaičių tam tikrame sveikųjų skaičių masyve.

2. Analizė, planavimas: reikia pereiti visus elementus, ieškoti maksimalios vertės. Įvestis yra sveikųjų skaičių sąrašas, o rezultatas bus didžiausias skaičius.

3. Dizainas: sukurkime maksimalų kintamąjį, inicijuokime jį pirmojo elemento reikšme. Eikime per visus kitus elementus. Jei dabartinis elementas yra didesnis už maksimalaus kintamojo vertę, atnaujinkime maksimalią vertę. Perėję visus elementus, maksimalus kintamasis turės didžiausią skaičių.

4. Detalizavimas: sukurkime maksimalų kintamąjį, priskirkime jam pirmojo masyvo elemento reikšmę[0]. Kiekvienam masyvo [i] elementui tarp 1 ir n. Jei masyvas [i] yra didesnis už maksimalų kintamąjį, atnaujinkime maksimalią reikšmę: max = masyvas[i]. Grąža max.

5. Įgyvendinimas, testavimas: mes parašysime programos kodą pasirinkta programavimo kalba (pavyzdžiui, Python), išbandysime jį įvairiuose bandymų masyvuose. Štai Python diegimo pavyzdys:

```
def find_max(array):
    max = array[0]
    for i in range(1, len(array)):
        if array[i] > max:
            max = array[i]
            return max
```

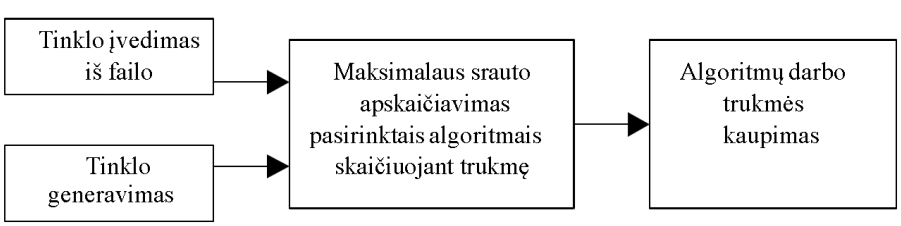
Pasinaudojus šiuo kodu galima bet kokio didžio masyve surasti ir ištraukti max skaičių tame masyve. Programa veikia, tik prie *numbers = [5, 2, 9, 1, 7, 101, 1, 0, 91, 101]* į laužtinius skliaustus galime įkelti bet kokius reikiamus skaičius(1 priedas). Pasinaudojus kodu ir pakeitus *max* į *min* galima tokiu pat būdu rasti mažiausią skaičių masyve. Turėdami tokį pradinį kodą ir ji keičiant pagal poreikius galima atlikti nemažai veiksmų su skaičių masyvais.

Pasinaudojus Fordo–Falkersono algoritmu ir Genetiniu algoritmu galime surašyti kodą su daugiau kintamųjų. Rašant programos kodą, svarbiausia užduotis yra nustatyti jo vykdymo rezultatą. Ši užduotis ne visada yra lengva, ir dažnai kreipiamasi į kodo analizę ir naudojami įvairūs įrankiai, kad išsiaiškinti, kas bus rodoma ekrane.

Pradiniai duomenys programinei įrangai nuskaitomi iš failų. Kaip papildomas duomenų šaltinis yra pageidautinas tinklo generatorius, galintis automatiškai sugeneruoti atsitiktinį tinklą. Tokių programų darbo rezultai būna:

- laiko intervalas, jis reikalingas kad galima būtu rasti *max* srautą kiekvienu pasirinktu metodu. Tokie rezultatai kaupiami, kad galima būtu atlikti jų statistinę analizę.
- Pasirinktu metodu ar algoritmu paskaičiuoto srauto aprašas kuris yra skirtas *max* srauto algoritmo vizualizavimui(15 pav.)

Programos duomenų transformavimo į rezultatus pagal apskaičiavimo trukmę schema 15 pav.

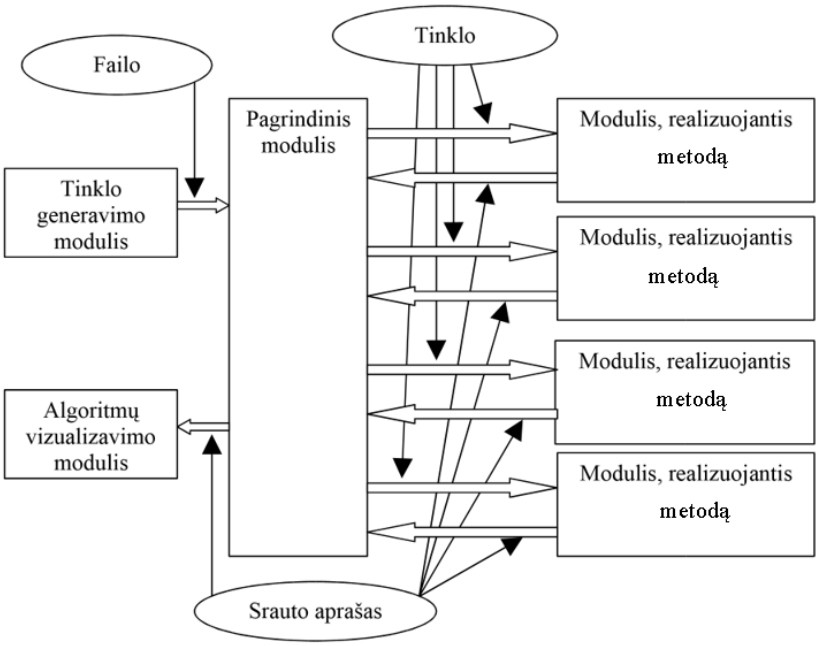


16pav. Duomenų transformavimo schema

Šiuolaikinės programavimo kalbos suteikia galimybių derinti ir išbandyti kodą, todėl lengviau nustatyti programos rezultatus. Tačiau tai nereiškia, kad užduotis tampa nereikšminga. Galų gale, programinės įrangos aplinka gali turėti įtakos kodo vykdymui, taip pat gali kilti sunkumų, susijusių su darbu su išorinėmis bibliotekomis ir paslaugomis. Pvz. Programos pagalba sužinoti koks oras yra bet kuriame pasaulio mieste. Čia reiks prisiregistruoti prie orų saito OMW

```
import pyowm
owm = pyowm.OWM( '6b03bb1940317df1d954b7ae9eb89aa6' )
mgr = owm.weather_manager()
observation = mgr.weather_at_place('city')
w = observation.weather
print(w)
```

Tai pradinis kodas, panaudojus jį sukuriama programa kuri užduoda klausimus, gauna atsakymą iš vartotojo ir pateikia reikalingą informaciją(priedai 2, 3, 4). Atrodo kelios eilutės o panaudojus algoritmą galima sudaryti tam tikrą programą. Programą galima pasikrauti komandinėje eilutėje(priedas 5). Programinės įrangos aplinka gali turėti įtakos kodo vykdymui, taip pat gali kilti sunkumų, susijusių su darbu su išorinėmis bibliotekomis ir paslaugomis.



17 pav. Duomenų išskaidymo struktūra

Moduliai susiję tarpusavyje per šiuos duomenis:

- ✓ tinklo generavimo modulis teikia duomenis (tinklo aprašą) pagrindiniam moduliui;
- ✓ pagrindinis modulis gauna duomenis iš tinklo generavimo modulio (failo vardą) ir algoritmų realizavimo modulių (srauto aprašą);
- ✓ teikia duomenis algoritmų realizavimo moduliams (tinklo aprašą) ir vizualizavimo moduliui (srauto aprašą);
- ✓ algoritmų realizavimo moduliai gauna duomenis iš pagrindinio programos modulio (tinklo aprašą), ir grąžina jam rezultatus (srauto aprašą);
- ✓ vizualizavimo modulis gauna duomenis (srauto aprašą) iš pagrindinio programos modulio.

Modulio paskirtis – valdyti visus kitus programos modulius: nurodyti duomenų failą, paleisti atsitiktinio tinklo generatorių, nurodyti algoritmų realizavimo modulių darbo režimą ir juos aktyvuoti. Modulis dirba cikliniu režimu – atlikus minėtus veiksmus, galima juos kartoti, vėl pasirenkant duomenų failą ir t.t. Modulio sąsaja – grafinė.

3.3.1. Žaidimo „Gyvatukas“ kūrimas

Nors "Gyvatukas" yra paprastas žaidimas, mes turime gana daug kodo, kurį galime parašyti. Žaidimo kūrimo etapai suskirstyti į nuoseklius žingsnius, kurių kiekvienas įgyvendina skirtingą žaidimo dalį. Prieš pradėdami koduoti, turime pagalvoti, kokių taisyklių turime nepamiršti kurdami žaidimą. Padarykime keletą pagrindinių taisyklių, turinčių įtakos žaidimui:

- žaidimo tikslas yra surinkti kuo daugiau taškų, padidinant gyvatės ilgį: kai žaidimo ekrane suvartojate specialų objektą, gyvatės ilgis padidėja vienu bloku;
- grotuvas pradeda judėti iš kairės ekrano pusės;

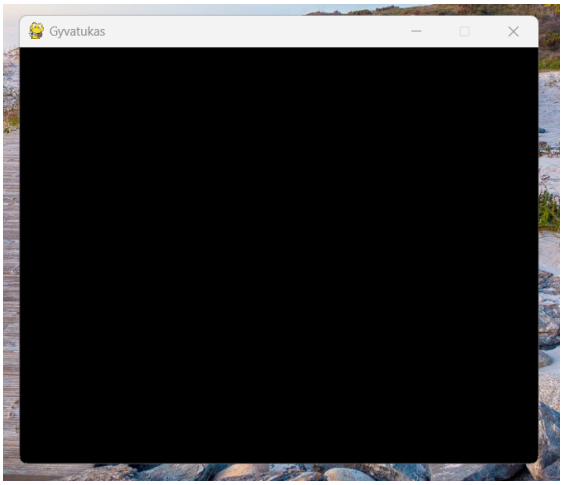
- žaidėjas gali judėti kairėn, dešinėn, aukštyn arba žemyn;
- žaidėjas negali pasitraukti iš ekrano, jei jie susiduria su ribomis, žaidimas baigiasi;
- žaidimas baigiasi, kai gyvatė atsitrenkia į save;
- žaidimas taip pat baigiasi, kai vartotojas uždaro langą.

Norėdami sukurti žaidimo langą naudojame "Pygame", panaudojame funkciją *display.set_mode()* ir perduoti jam norimą lango dydį pikseliais. Toliau naudojame *init()* ir *quit()* metodus, kad aktyvuotume biblioteką kodo pradžioje ir užbaigtume kodo pabaigoje.

Update () metodas naudojamas ekrano turiniui atnaujinti. Taip pat yra *flip()* metodas, kuris veikia panašiai kaip *update()*. Skirtumas tas, kad *update()* metodas atnaujina tik atliktus pakeitimus, o *flip()* metodas perpiešia visą ekraną. Bet jei *update()* metodui neperduoti jokių parametrų, visas ekranas taip pat bus atnaujintas.

```
 import pygame
pygame.init()
dis = pygame.display.set_mode((500, 400))
pygame.display.update()
pygame.display.set_caption('Gyvatukas')
game_over = False
while not game_over:
    for event in pygame.event.get():
     print(event)
pygame.quit()
```

Surašę kodą aktyvuojame ir mums išlenda langas sukurtas pagal nurodytus parametrus ir su mūsų užduotu pavadinimu. Norėdami, kad žaidimo langas turėtų pavadinimą, mes naudojame *pygame.display.set_caption('')* (pavadinimas rašomas kabutėse).



Dabar inicijuosime kintamuosius, kurie nustato spalvą. Mes juos naudosime norėdami priskirti spalvą ekranui, pačiai gyvatei ir maistui. *Pygame* naudoja standartinę RGB schemą, o tai

reiškia, kad bet kokia spalva yra raudonos, žalios ir mėlynos spalvos derinys, kurio intensyvumą galime keisti. Mūsų gyvatė yra stačiakampis, todėl stačiakampiams sukurti naudosime funkciją *draw.rect()*. Tai leidžia nustatyti stačiakampio dydį ir spalvą.

Jei gyvatė atsitrenkia į ekrano kraštą, žaidėjas pralaimi ir žaidimas baigiasi. Norėdami užkoduoti šią taisyklę, panaudojame teiginį *if*, kuris nustato gyvatės *x* ir *y* koordinates ir analizuoja, ar jos nepatenka į žaidimo lauko ribas. Dabar, jei gyvatė pasieks ekrano kraštą, žaidimas baigsis ir ekrane pasirodys pranešimas: „*pralaimėjote! paspauskite Q norint išeiti iš žaidimo arba C norint tęsti*“. Toliau papildome kodą dar vienu kintamuoju. Dabar, kai prasidės žaidimas, maistas bus rodomas be pačios gyvatės. Mūsų atveju tai yra juodo kvadrato forma. Be to, mes pakeitėme žaidimo lauko ir gyvatės spalvą, padidindami jų kontrastą. Papildykime savo kodą taip, kad gyvatės ilgis padidėtų valgant maistą. Norėdami tai padaryti, mums reikės sąrašo, kuriame būtų saugomas dabartinis gyvatės ilgis. Atsižvelkime į kitą svarbų taisyklių punktą: kai gyvatės galva susiduria su savo kūnu, žaidimas baigiasi.

Pridėkime dabartinio žaidimo rezultato rodymą. Norėdami tai padaryti, sukurkime *Your_score* funkciją. Jis parodys gyvatės ilgį, atimdamas iš jo 1 (galų gale, 1 yra pradinis gyvatės dydis, ir tai nėra žaidėjo pasiekimas).

```
34
35    def Your_score(score):
36        value = score_font.render("surinkote balų:" + str(score), True, yellow)
37        dis.clit(value, [0, 0])
38
```

Ir atskirai parašome gyvatės ilgio nustatymo taisyklę, atimdami vieną iš dabartinio gyvatės ilgio

```
106             Your_score(Length_of_snake - 1)
```

Dabar žaidimo lauke bus rodomas dabartinis rezultatas(8 priedas). Galime manyti, kad mūsų darbas "Gyvatė" yra baigtas. Mes visiškai įgyvendinome žaidimą, kurį planavome darbo pradžioje. Visas pilnai kodas 8 priede.

3.3.2. Žaidimas „Kamuoliukas“

Yra žaidimo laukas – paprastas stačiakampis su tvirtomis sienomis. Kai kamuolys paliečia sieną ar lubas, jis atšoka kita kryptimi. Jei jis atsitrenkia į grindis, tai pralaimėjimas. Kad taip neatsitiktų, platforma skrenda palei žemiau esančias grindis, o valdimas rodyklių pagalba. Užduotis yra kuo

ilgiau pastatyti platformą po kamuoliu. Už kiekvieną sėkmingą kamuolio gelbėjimą gaunamas vienas taškas.

Norėdami įgyvendinti šią žaidimo logiką, turite numatyti šiuos elgesio scenarijus:

- žaidimas prasideda;
- kamuolys pradeda judėti;
- jei paspaudžiamos rodyklės kairėn arba dešinėn, perkeliama platforma;
- jei kamuolys liečia sienas, lubas ar platformą, atšoka;
- jei rutulys paliečia platformą, padidinamas rezultatas vienu;
- jei kamuolys nukrenta ant grindų, parodyti pranešimą ir užbaikti žaidimą.

Sunkumas toks, kad viskas vyksta lygiagrečiai ir nepriklausomai vienas nuo kito. Tai yra, kol kamuolys skrenda, galime perkelti platformą arba palikti ją vietoje. O kai kamuolys atšoka nuo sienų, tai netrukdo ir kitiems objektams judėti ir bendrauti tarpusavyje. Taigi turime apibrėžti tris klases: platformą, patį kamuolį ir rezultatą. Nustatyti, kaip jie reaguoja į vienas kito veiksmus. Mums nereikia patiems apibrėžti lauko – tam yra paruošta biblioteka. Ir tada mes rašysime metodus šiose klasėse, kurie bus atsakingi už mūsų objektų elgesį. Visa tai klausiame vieną kartą, o tada objektai sugalvoja, kaip reaguoti vienas į kitą ir ką daryti skirtingose situacijose. Mes griežtai nenustatome viso algoritmo, bet nustatome žaidimo taisykles.

Norėdami sukurti langą, kuriame bus matoma grafika, naudojame *Tk()* klasę. Tai tik langas, bet ne turinys. Norėdami, kad turinys būtų rodomas, turime sukurti drobę – matomą lango dalį. Ten piešime savo žaidimą. Drobės klasė yra atsakinga už drobę, todėl turėsime sukurti savo objektą iš šios klasės ir tada dirbti su šiuo objektu.

Jei priverstinai neapribosime platformos greičio, ji judės akimirksniu, nes kompiuteris labai greitai apskaičiuoja ir akimirksniu perkels jį į kitą kraštą. Todėl dirbtinai apribosime judėjimo laiką, o tam mums reikės modulio *„Time“*. Mums dar reikia atsitiktinai nustatyti pradinę kamuolio padėtį ir platformą, kad būtų įdomiau žaisti. Už tai atsakingas atsitiktinis modulis - jis padeda generuoti atsitiktinius skaičius ir maišyti duomenis.

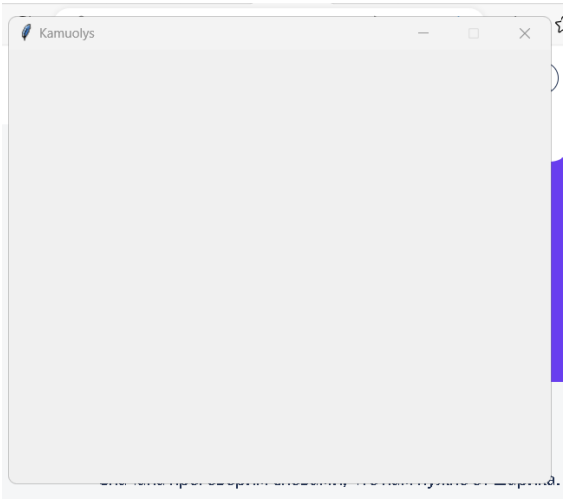
Pradedame kurti kurti kodą.

```
from tkinter import *
import time
import random
tk = Tk()
tk.title('Kamuolys')
tk.resizable(0, 0)
tk.wm_attributes('-topmost', 1)
canvas = Canvas(tk, width = 500, height = 400, highlightthickness = 0)
```

canvas.pack()

tk.update()

Šiuo kodu pajungiam reikalingas bibliotekas ir žaidimo lauko nustatymus. Atsidaro sukurtas langas.



Dabar reikia apsispręsti ką norime, kad mūsų kamuolys galėtu daryti. Jis turi sugebėti:

- nustatyti savo pradinę padėtį ir judėjimo kryptį;
- suprasti, kada jis palietė platformą;
- piešti pats save ir suprasti kada jam reikia nupiešti save naujoje padėtyje (pvz., atšokus nuo sienos).

To turėtu pakakti, kad kamuolys galėtu gyventi savo gyvenimą ir galėtu bendrauti su aplinka. Tuo pačiu metu reikia prisiminti, kad kiekvienoje klasėje turi būti konstruktorius, kuris yra kodas, atsakingas už naujo objekto sukūrimą. Be šito, negalime sukurti kamuolio.

```
class Ball:
    def __init__(self, canvas, paddle, score, color):
        self.canvas = canvas
        self.paddle = paddle
        self.score = score
        self.id = canvas.create_oval(10, 10, 25, 25, fill = color)
        self.cnavas.move(self.id, 245, 100)
        starts = [-2, -1, 1, 2]
        random.shuffle(starts)
        self.x = starts[0]
        self.y = -2
        self.canvas_height = self.canvas.winfo_height()
        self.canvas_width = self.canvas.winfo_width()
        self.hit_bottom = False

    def hit_paddle(self, pos):
        paddle_pos = self.canvas.coords(self.paddle.id)
        if pos[2] >= paddle_pos[0] and pos[0] <= paddle_pos[2]:
            if pos[3] >= paddle_pos[1] and pos[3] <= paddle_pos[3]:
                self.score.hit()
                return True
            return False
        def draw(self):
            self.canvas.coords(self.id)
            if pos[1] <= 0:
                self.y = 2
            if pos[3] >= self.canvas_height:
                self.hit_bottom = True
                canvas.create_text(250, 120, text = 'Pralaimėjote', font = ('Courier', 30), fill = 'red')
            if self.hit_paddle(pos) == True:
                self.y = -2
            if pos[0] <= 0:
                self.x = 2
            if pos[2] >= self.canvas_width:
                self.x = -2
```

Padarome tą patį su platforma: pirmiausia aprašome jos elgesį, o tada išverčiame į kodą. Taigi, štai ką platforma turėtų sugebėti padaryti:

- judėti kairėn arba dešinėn, priklausomai nuo paspaustos rodyklės;
- suprasti kada žaidimas prasidėjo ir galima judėti.

Kuriame kodą.

```
class Paddle:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill = color)
        start_1 = [40, 60, 90, 120, 150, 100, 200]
        random.shuffle(start_1)
        self.starting_point_x = start_1[0]
        self.canvas.move(self.id, self.starting_point_x, 300)
        self.x = 0
        self.canvas_width = self.canvas.winfo_width()
        self.canvas.bind_all('<KeyPress-Right>', self.turn_right)
        self.canvas.bind_all('<KeyPress-Left>', self.turn_left)
        self.started = False
        self.canvas.bind_all('<KeyPress-Return>', self.start_game)

    def turn_right(self, event):
        self.x = 2

    def turn_left(self, event):
        self.x = -2

    def start_game(self, event):
        self.started = True

    def draw(self):
        self.canvas.move(self.id, self.x, 0)
        pos = self.canvas.coords(self.id)
        if pos[0] <= 0:
            self.x = 0
        elif pos[2] >= self.canvas_eidth:
            self.x = 0
```

Galima neatskirti skaičiavimus į atskirą klasę ir kiekvieną kartą ją apdoroti rankiniu būdu. Bet čia tikrai lengviau padaryti klasę, nustatyti reikiamus metodus, kad jie galėtų išsiaiškinti, ką ir kada daryti. Iš skaičiavimų reikia, kad jie teisingai reguotu kai balai didėja ir tai rodytu ekrane.

```
class Score:
    def __init__(self, cnvas, color):
        self.score = 0
        self.canvas = canvas
        self.id = canvas.create_text(450, 10, text = self.score, font = ('Courier', 15), fill = color)
    def hit(self):
        self.score += 1
        self.canvas.itemconfig(self.id, text = self.score)
```

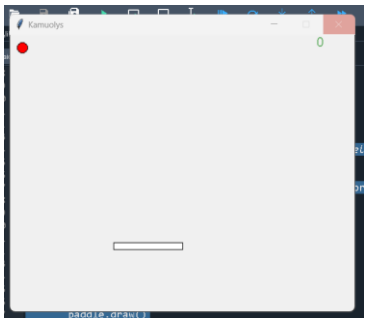
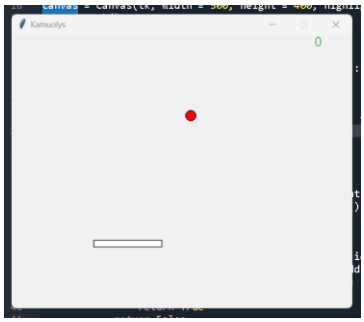
Mes turime viską, kad galėtume parašyti patį žaidimą. Jau atlikome visus būtinus pasiruošimus ir tereikia sukurti konkrečius kamuolio objektus, platformą ir rezultatą bei pasakyti jiems, kokia tvarka ką daryti. Žaidimo esmė yra ne numesti kamuolį. Kol tai neįvyks, viskas juda, bet kai tik kamuolys nukrenta, turite parodyti pranešimą apie žaidimo pabaigą ir sustabdyti programą.

```
score = Score(canvas, 'green')
paddle = Paddle(canvas, 'White')
ball = Ball(canvas, paddle, score, 'red')
while not ball.hit_bottom:
    if paddle.started == True:
        ball.draw()
        paddle.draw()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
time.sleep(3)
```

Taip glaustai atrodo paties žaidimo kodas. Visą kodą galima apžvelgti 9 priede. Remdamiesi šiuo kodu, galime modifikuoti žaidimą:

- pridėti antrą rutulį;
- dažyti elementus kita spalva;
- pakeisti rutulio dydį;
- pakeisti platformos greitį;

- daryti viską iš karto;
- pakeisti programos logiką į kitą.

3.3.3. Interneto gričio matuoklis

Ryšio greitis yra informacijos kiekis, kurį kompiuteris gauna arba siunčia per laiko vienetą. Tokiu atveju sekundė laikoma laiko vienetu, o kilobitas arba megabitas yra informacijos kiekis. Jei jūsų greitis yra 10 Mbps, tai reiškia, kad maksimalus bet kurio failo atsisiuntimo greitis per naudojamas programas bus apie 1, 25 megabaitų per sekundę.

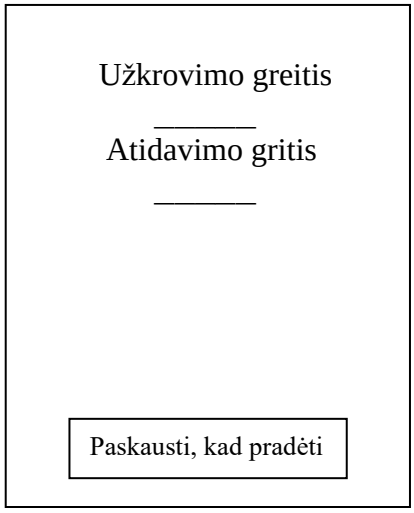
Tikrindami greitį, turėtumėte nepamiršti, kad atsisiunčiant failus iš bet kurio serverio, tarp kompiuterio ir paties serverio yra dešimtys tarpinių serverių, kurie taip pat gali keisti greitį. Tai yra, jei serveris buvo toje pačioje lentelėje kaip ir jūsų kompiuteris ir jie buvo sujungti vienas su kitu tuo pačiu laidu, tada galėtume kalbėti apie tiksliausius rezultatus.

Kuriamiems žaidimų kodams reikia platformos, kuria galėtu naudoti mokiniai ir per pamokas ir po pamokų(pvz. namuose). Viena, šiuo metu, iš populiariausiu platformu ne tik akademinėje erdvėje, bet ir mokykloje, yra personaliniai kompiuteriai. Mokykloje visose klasėse yra bent po vieną kompiuterį, o kompiuterinėse klasėse yra jų daugiau, visi jie skirti ugdymo procesui tobulinti. Kiekvienoje mokykloje yra bent viena kompiuterių klasė kurioje mokiniai mokomi IT technologijų meno. Kompiuteriai duoda šiek tiek daugiau laisvės tiek pedagogui tiek mokiniui.

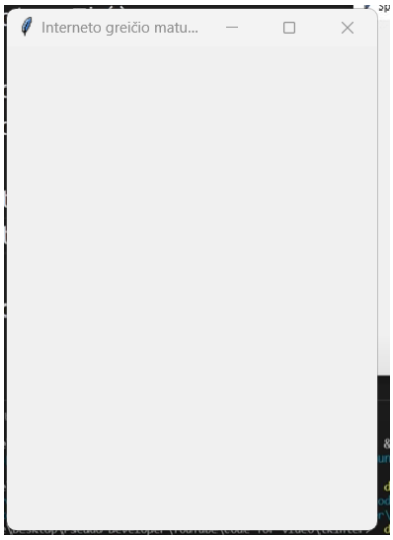
Mokimo programoje IT technologijų pamokose yra viena iš temų programavimas. Norisi mokinius sudominti ir parodyti, kad galima mokytis pradėti nuo paprastų naudojamą programų. Ir kas žino gal kurio nors mokinio svajonė tapti „chakeriu“ išsipildys.

Paprastos ir pakankamai aiškios, bet įdomios programos išmatuoti interneto greitį kūrimas.

Kai kuriais atvejais, norint nustatyti programos rezultatą, būtina naudoti matematinius skaičiavimus arba loginę analizę. Kartais tam reikia naudoti specialius įrankius, pvz., kūrimo aplinkas arba integruotas kūrimo aplinkas (IDE), kurie leidžia stebėti ir analizuoti kodą vykdymo metu. Mums pakaks tik maketo langui(interfeisui).



Išspręsime paprastą bet naudingą uždavinį. Mes matuosime interneto greitį mygtuko paspaudimu. O rezultatą matavimų išvesime į ekraną teksto pavidalu. Rašome funkcionalą grafinio dviejų funkcinių vienetų sandūros(interfeiso) ir importuojame visas klases su komanda *from tkinter import **. Šiuolaikinėse operacinėse sistemose bet koks naudotojo priedas pririštas prie lango. Jį galima vadinti pagrindiniu, nes jame išsidėsto visi valdikliai(widgets). Aukščiausio lygio lango objektas kūriamas komanda *Tk()*. Kintamasis *root,* užduodame mūsų langui matmenys naudojant *title* ir *geometry*. Kad langas neužsidarinėtu tol kol nepanorės naudotojas naudojam komandą *mainloop()*. Gavome štai tokį nedidelį langą.



Dabar mums reikia mygtuko kurį reikės paspausti. Su kodu *button* mes tai įvykdome ir gauname kodą

```
from tkinter import *
root = Tk()

root.title("Interneto greičio matuoklis")
root.geometry("300x400")
```

button = Button(root)

root.mainloop()

Su komanda *button.pack()* pridedame migtuką kurį paspaudus paleisime mūsų programą. Surašome mygtuko komandą, jo dydžio ir įrašome tekstą ant mygtuko.

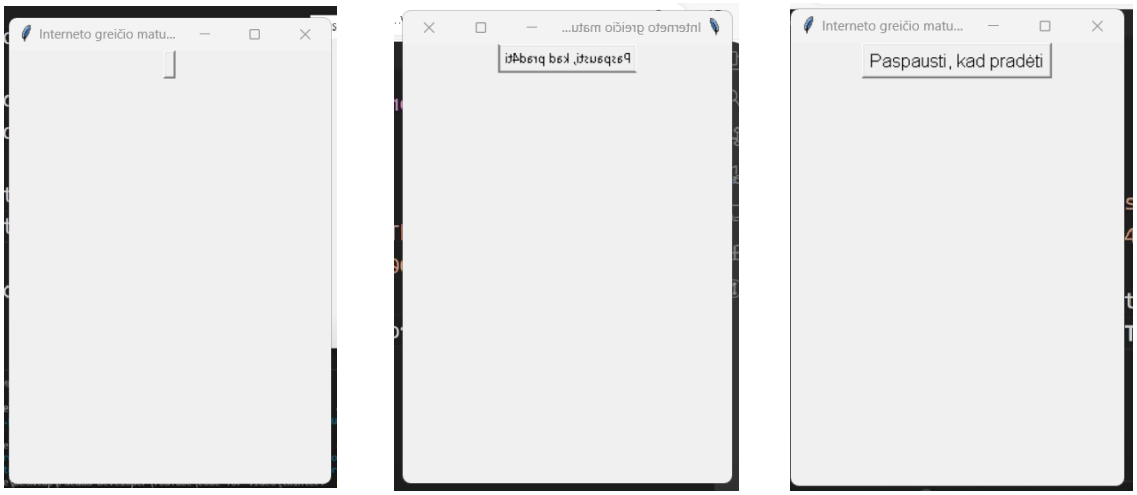
root.title("Interneto greičio matuoklis")

root.geometry("300x400")

button = Button(root, text = "Paspausti, kad pradėti", font = 40)

button.pack()

Gauname štai tokius langus.pirmas tik mygtukas, antras jau mums reikiamo dydžio mugtukas su užrašu, o trčias užrašo šriftas įskaitomas ir norimo dydžio.



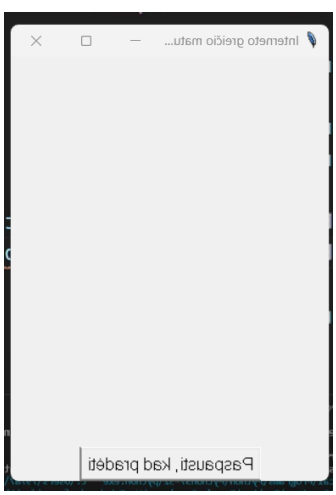
Mygtukas kitoje vietoje negu mūsų makete. Mums padės komanda *pack.* Mes surašome *side = BOTTOM* ir gauname mygtuką apačioje, bet jis guli ant linijos. Pagal mūsų sudarytą maketą jis turėtu būti aukščiau. Argumentas *pady = 40* parodo, kad mes parašėme atstumą nuo linijos *y* kordinatės kryptimi į viršų.

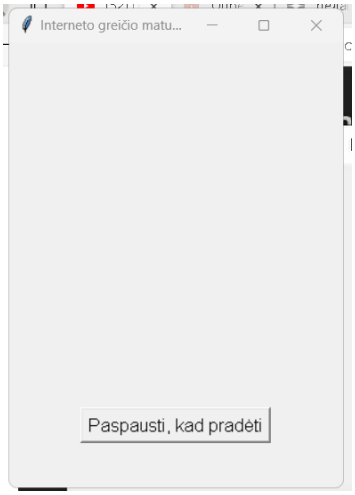
root.title("Interneto greičio matuoklis")

root.geometry("300x400")

button = Button(root, text = "Paspausti, kad pradėti", font = 40)

button.pack(side = BOTTOM, pady = 40)





Dabar mums reikia atvaizduoti interneto greitį, mes turėsime du rodikliu: užkrovimo greitį ir atidavimo greitį. Reikia sukurti du objektus *label* metodu.

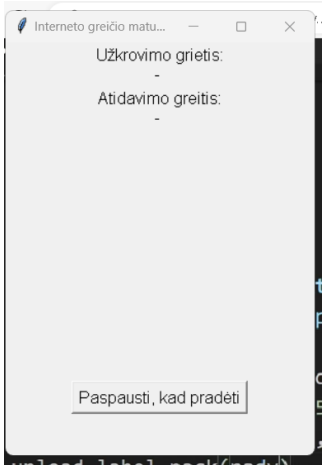
download_label = Label(root, text = "Užkrovimo grietis:\n- ", font = 35)

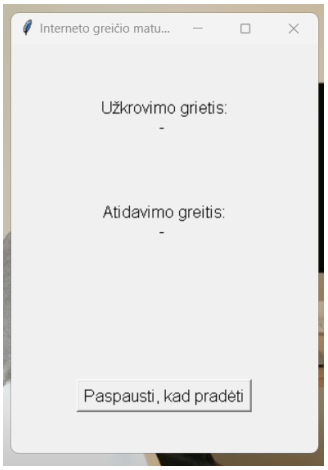
download_label.pack(pady = (50, 0))

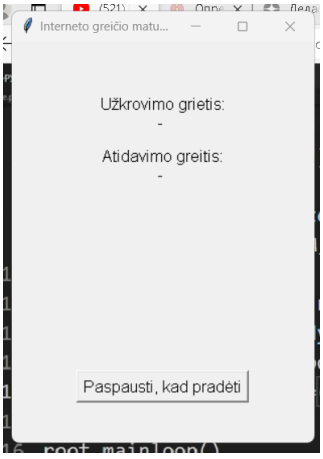
upload_label = Label(root, text = "Atidavimo greitis:\n- ", font = 35)

upload_label.pack(pady = (10, 0))

Argumentuose nurodome tekstą ir norimą šrifto dydį. Visa tai pakuojame ir gauname užrašus.







Dabar mums reikia importuoti *speedtest* , surašome matavimo programą ir parašome kintamuosius, o kadangi šiais laikais greitis matuojamas megabitais reikia padalinti iš 10^6 nes programa skaičiuoja bitais. Dar šiek tiek pakeičiame jau sukurtus objektus, nes turime jų kintamąsias, kad tekstas neliptu vienas ant kito.

from speedtest import Speedtest

def test():

download = Speedtest().download()

upload = Speedtest().upload()

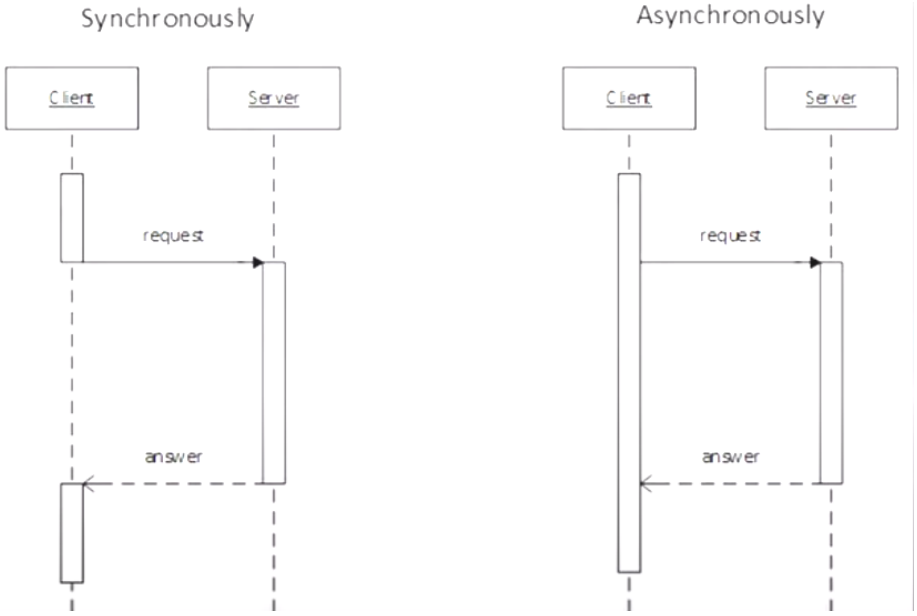
*download_speedtest = round(download / (10**6), 2)*

*upload_speedtest = round(upload / (10**6), 2)*

download_label.config(text = "Užkrovimo grietis:\n " + str(download_speed) + "MbPs")

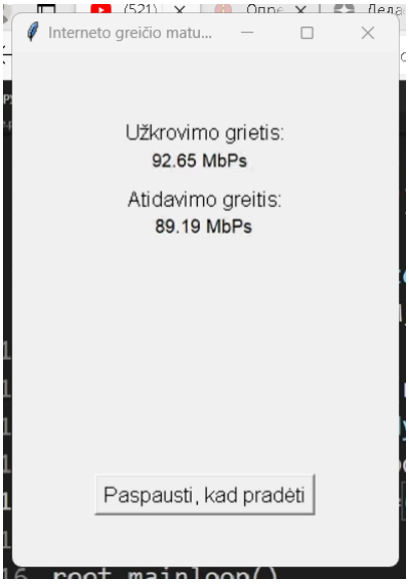
upload_label.config(text = "Atidavimo greitis:\n " + str(upload_speed) + "MbPs")

Greičio programos testavimui reikia iki 15s laiko. Mūsu programa veikia vieno srauto režime tai kol programa dirbs jokių veiksmų su langu atlikti negalėsime.



18 pav. Srautų pasiskirstymas

Kai programa suskaičiuoja gauname rezultatą.



Yra vienas niuansas kuris išaiškėjo bandymų metodu. Programa nesuveikia jei nėra tiesioginio srauto.

IŠVADOS

Algoritmų kūrimas yra programavimo proceso ir problemų sprendimo dalis. Gerai suprojektuotas, jis leidžia supaprastinti sudėtingumą, padidinti programų efektyvumą ir užtikrinti rezultatų patikimumą bei tikslumą.

Rašant programos kodą, svarbiausia užduotis yra nustatyti jo vykdymo rezultatą. Ši užduotis ne visada yra lengva, ir dažnai turime kreiptis į kodo analizę ir naudoti įvairius įrankius, kad išsiaiškintumėme, kas bus rodoma ekrane. Šiuolaikinės programavimo kalbos suteikia galimybių derinti ir išbandyti kodą, todėl lengviau nustatyti programos rezultatus. Tačiau tai nereiškia, kad užduotis tampa nereikšminga. Galų gale, programinės įrangos aplinka gali turėti įtakos kodo vykdymui, taip pat gali kilti sunkumų, susijusių su darbu su išorinėmis bibliotekomis ir paslaugomis.

Programų kūrimas tai tam tikras algoritmas kuris užrašomas kuria nors programavimo kalba. Programavimas apima šiuos elementus meno, mokslo, matematikos, inžinerijos. Tai rodo kad mokinys mokydamasis programavimo kalbos visapusiškai lavinasi. Įvairiausi uždaviniai pasižymi svarbiomis savybėmis kuios turi įtaką mokymuisi. Jie skatina domėtis ir meta iššūkius besidomintiems mokiniams. Mokidamiesi programavimo jie gali daryti klaidas ir eksperimentuoti, taip ugdydami mąstymo įgūdžius.

Panaudota sistema patogi naudoti; jos pagalba galima sukurti kiekvieno veiksmo darbo srautą taip sumažinus trukmę. Taip pat galima kaupti duomenis statistiniam apdorojimui ir vaizdžiai iliustruoti maksimalaus srauto procesą pasirinktame tinkle.

LITERATŪRA

1. Акулич Н.А. *Математическое программирование в примерах и задачах.* –М.: Высшая школа, 2013
2. Aliabajeva V.G., Pastuchova G.V.. *Teorija algoritmų: mokomoji medžiaga*. Perm, 2013
3. Bäck.T. *Evolutionary Algorithms in Theory and Practice.* Oxford Univ. Press, 1996
4. Blei D. M., Ng A. Y., Jordan M. I., *Latent Dirichlet Allocation*, Journal of Machine Learning Research, 3 (2002), 993–1022.
5. Buford J. F., Yu H., Lua E., *P2P Networking and Applications*, The Morgan Kaufmann Series in Networking, 2009.
6. Burnett. M.*Blocking Brute Force Attacks.* UVA Computer Science, 2007
7. *Center for Discrete Mathematics & Theoretical Computer Science*, [žiūrėta 20231201] Prieiga internetu: http://dimacs.rutgers.edu/
8. *Computer Science & Engineering*, University of Washington [žiūrėta 20230408]. Prieiga internetu: http://www.cs.washington.edu/homes/anderson
9. Cormen T. H., Leiserson Ch. E., Rivest R. L. *Introduction to Algorithms*, by The Massachusetts Institute of Technology, Second Edition, Cambridge , Massachusetts London, England, 2001
10. Daud A, Li J. ,Zhou L., Muhammad F., *Knowledge discovery through directed probabilistic topic models*, Frontiers of Computer Science in China, 4:2 (2010), 280–301.
11. Dantzig G. B. *Application of the Simplex Method to a Transportation Problem.* In T. C. Koopmans, editor, Activity Analysis and Production and Allocation, pages 359-373. Wiley, New York,1951.
12. Dinic E. A.. *Algorithm for Solution of a Problem of Maximum Flow in Networks with Power Estimation.* Soviet Math. Dokl., 11, 1970
13. Диниц Е. А. *Метод поразрядного сокращения невязок и транспортные задачи. Исследования по дискретной математике*. Наука, Москва, 1973.
14. Edmonds J. and Karp R. M. *Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems*. J. Assoc. Comput. Mach., 19, 1972.
15. Eiben. A. *Genetic algorithms with multi-parent recombination*. In 1994
16. Ferguson T. S., *A bayesian analysis of some nonparametric problems*, The Annals of Statistics, 1:2 (1973), 209–230.
17. Ford L. R., Fulkerson Jr. and D. R.. *Maximal Flow Through a Network*. Canadian Journal of Math., 8, 1956.
18. Gagarin, Yu.E. Nikitenko, U.V. Mosin E.D.. *Determination of the maximum network flow using the tagging method*. Bauman Moscow State Technical University. 2022

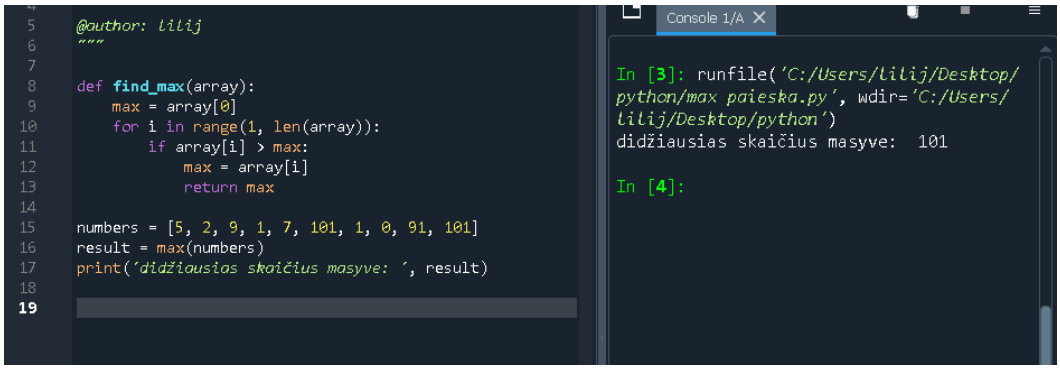
19. Ganeriwal S., Balzano L. K., Mani B., *Reputation-based Framework for High Integrity Sensor Networks*, ACM Transactions on Sensor Networks, 5., 2007
20. Geisler T. ir Manikas T. W.. *Autonomous robot navigation system using a novel value encoded genetic algorithm.* The 2002 45th Midwest Symposium on Circuits and Systems, 2002. MWSCAS-2002. Tom. 3, p. III–III, 2002
21. Gelman A., Carlin J. B., Stern H. S., Rubin D. B., *Bayesian Data Analysis*, Chapman and Hall/CRC, 2013.
22. Ghein L., *MPLS Fundamentals*, Cisco Press, 2006.
23. Goldberg A. V. *A New Max-Flow Algorithm.* Technical Report MIT/LCS/TM-291, Laboratory for Computer Science, M.I.T., 1985.
24. Goldberg A. V. and Tarjan R. E. *A New Approach to the Maximum Flow Problem.* J. Assoc. Comput. Mach., 35, 1988.
25. Goldberg A. V., Rao S.. *Length Functions for Flow Computations.* Technical Report # 97-055, NEC Research Institute, Inc., March 1997; revised August 1997
26. Goldberg A. V. *Recent Developments in Maximum Flow Algorithms,* Technical Report #97-045, NEC Research Institute, Inc., April 1998
27. Hollandas J. H.. *Genetiniai algoritmai.* "Mokslo pasaulyje", 1992
28. Jolliffe I. T., *Principal components analysis*, Springer-Verlag, New York, 1986.
29. Каймин В.А. *Методы разработки программ на языках высокого уровня: Учебное пособие*. М.: МИЭМ, 1985.
30. Karzanov A. V. *Determining the Maximal Flow in a Network by the Method of Preflows.* Soviet Math. Dok., 15, 1974.
31. Кинг Д . *Создание эффективного программного обеспечения.* М.: Мир, 1991
32. Kintsch W., *Handbook of Latent Semantic Analysis*, Erlbaum, Hillsdale, NJ, 2007
33. Knorr E. M., Ng R. T., *Algorithms for Mining Distance-Based Outliers in Large Datasets*", Proceedings of the 24th International Conference on Very Large Data Bases, 1., 1998
34. Knorr E. M., Ng R. T., *Finding Intensional Knowledge of Distance-based Outliers*, Proceedings of the 25th International Conference on Very Large Data Bases, 1 (1999).
35. Кнут Д.Э. *Искусство программирования.* Т. 1-3. М.: Издательский дом «Вильямс», 2000
36. Кормен Т.Х., Лейзерсон Ч.И., Ривест Р.Л., Штайн К. *Алгоритмы: построение и анализ.* Пер. с англ. М.: Издательский дом «Вильямс»,200
37. Lawler. E. *Network Flows // Combinatorial Optimization: Networks and Matroids.* Dover, 2011
38. Lima J. A, Gracias N., Pereira H., ir Rosa A.. *Fitness function design for genetic algorithms in cost evaluation based problems.* Proceedings of IEEE International Conference on Evolutionary Computation, 1996.

39. Listopadskis N.. *Kombinatorinio optimizavimo uždaviniai ir jų sprendimo algoritmai.* [žiūrėta 20231212] Prieiga internete: http://www.technologijos.lt/zyme/Kur-ir-ka studijuoti?tid=2832&rikiavimas=0&a=0&ystart=&ystop=&tipas=0&tik_rubrikos=0&pp=5.
40. Майерс Г. *Искусство тестирования программ.* М.: Финансы и статистика, 1982.
41. Michalewicz. Z. *Genetic Algorithms + Data Structures = Evolution Programs,* Springer-Verlag. Springer Berlin Heidelberg, 1992
42. Minka T., Lafferty J., *Expectation-propagation for the generative aspect model*, Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence, 2002.
43. Моисеев. Н.Н. *Математические задачи системного анализа*. М: Наука, 1981
44. Mogiliovas A.V., Pak N.I., Henner E.K. *Informatika.* Vadovėlis mokiniams. Ped. Universitetų. – 4-asis leidimas, ster. Maskva: Leidybos centras "Akademiya", 2007
45. Nadeau T. D., Gray K., *SDN: Software Defined Networks*, O'Reilly Media, 2013
46. Natarajan A.M. Balamurugan R. and Premalatha K.. *Stellar-mass black hole optimization for biclustering microarray gene expression data.* Applied Artificial Intelligence an International Journal, 2015
47. Носков А. Н., Манов И. А. *Разработка механизма адаптивной маршрутизации трафика в программно-конфигурируемых сетях*, [žiūrėta 20231231] Prieiga internete: https://www.mais-journal.ru/jour/article/view/270
48. Paramasivan B. A, Prakash M. J, Kaliappan M., *Development of a secure routing protocol using game theory model in mobile ad hoc networks*, Journal of Communications and Networks, 1:15 2015
49. Plukas K., Mačikėnas E., Jarašiūnaitė B., Mikuckienė I.. *Taikomoji diskrečioji matematika,* Kaunas, Technologija, 2002.
50. Prokhorovas Y.V.; Raudonas. kol.: S.I. Adyan, N.S. Bakhvalov, V.I. Bityutskov, A.P. Ershov, L.D. Kudryavtsev, A.L. Onishchik, A.P. Yushkevich. *Matematinis enciklopedinis žodynas.* /Hl. Maskva, Tarybinė enciklopedija Publ., 1988
51. Samsudin N. A. a, Bradley A. P. b, *Extended na?ve bayes for group based classification Advances in Intelligent Systems and Computing*, 1st International Conference on Soft Computing and Data Mining, 287., 2014
52. Sedighi K. H., Ashenayi K., Manikas T. W., Wainwright R. L. ir Heng-Ming Tai. *Autonomous local path planning for a mobile robot using a genetic algorithm.* Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753), 2004
53. Shipman C. M., Hopkinson K. M., Lopez J., *Con-resistant trust for improved reliability in a smart-grid special protection system*, IEEE Transactions on Power Delivery, 13:1., 2015

54. Sleator D. D. and Tarjan R. E. *A data structure for dynamic trees*. Journal Computer Systems Science, 1983
55. Sleator D. D. and Tarjan R. E. *Self-adjusting binary search trees*. Journal of the M M Press 32 (3), 2015
56. *The Programmer's File Format Collection*, [žiūrėta 20230129]. Prieiga internetu: http://www.wotsit.org/
57. Vapnik V., *Statistical Learning Theory*, Wiley, 1998.
58. Veshegna Sh., *Kachestvo obslujivaniya v IP-setyah*, Cisco Press, 2003
59. ftp://dimacs.rutgers.edu/pub/challenge/graph/doc/ccformat.tex
60. https://loginom.ru/blog/ga-math
61. https://learn.microsoft.com/lt-lt/power-automate/desktop-flows/introduction

PRIEDAI

1 priedas. Max radimas skaičių masyve.



```
@author: lilij
"""

def find_max(array):
    max = array[0]
    for i in range(1, len(array)):
        if array[i] > max:
            max = array[i]
            return max

numbers = [5, 2, 9, 1, 7, 101, 1, 0, 91, 101]
result = max(numbers)
print('didžiausias skaičius masyve: ', result)
```

```
Console 1/A

In [3]: runfile('C:/Users/lilij/Desktop/
python/max paieska.py', wdir='C:/Users/
lilij/Desktop/python')
didžiausias skaičius masyve:  101

In [4]:
```

2 priedas. Orų prognozė1

```
import pyowm

owm = pyowm.OWM( '6b03bb1940317df1d954b7ae9eb89aa6' )
mgr = owm.weather_manager()

observation = mgr.weather_at_place('Vilnius')
w = observation.weather

print(w)
```

```
In [19]: runfile('C:/Users/lilij/.spyder-py3/temp.py', wdir='C:/Users/lilij/.spyder-py3')
<pyowm.weatherapi25.weather.Weather - reference_time=2023-12-25 16:44:02+00:00, status=clouds,
detailed_status=overcast clouds>

In [20]:
```

3 priedas. Orų prognozė2

```
import pyowm

city = input('koks miestas jus domina?: ')

owm = pyowm.OWM( '6b03bb1940317df1d954b7ae9eb89aa6' )
mgr = owm.weather_manager()

observation = mgr.weather_at_place('Vilnius')
w = observation.weather
temperature = w.temperature('celsius')

print('Mieste ' + city + ' dabar temperatūra:' + str(temperature))
```

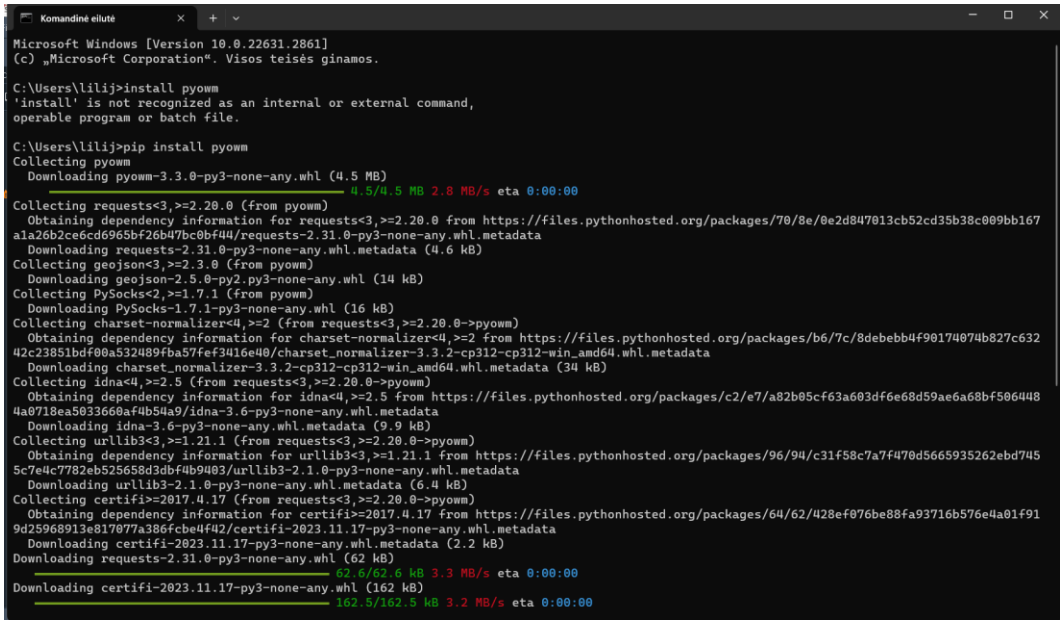
```
In [20]: runfile('C:/Users/lilij/.spyder-py3/temp.py', wdir='C:/Users/lilij/.spyder-py3')

koks miestas jus domina?: Kaunas
Mieste Kaunas dabar temperatūra:{'temp': 3.18, 'temp_max': 3.73, 'temp_min': 3.13, 'feels_like':
-2.44, 'temp_kf': None}

In [21]:
```

4 priedas. Orų prognozė3

```python
import pyowm

city = input('koks miestas jus domina?: ')

owm = pyowm.OWM( '6b03bb1940317df1d954b7ae9eb89aa6' )
mgr = owm.weather_manager()

observation = mgr.weather_at_place('city')
w = observation.weather

temperature = w.temperature('celsius')['temp']


print('Mieste ' + city + ' dabar temperatūra:' + str(temperature) + ' pagal celsiju.')
print( 'Nurodytame mieste ' + w.detailed_status)
```

```
In [3]: runfile('C:/Users/lilij/.spyder-py3/temp.py', wdir='C:/Users/lilij/.spyder-
py3')

koks miestas jus domina?: Kaunas
Mieste Kaunas dabar temperatūra:15.65 pagal celsiju.
Nurodytame mieste light rain

In [4]:
```

5 priedas. Komandinė eilutė



6 priedas. Programos kodas rūšiavimas

```
from watchdog.observers import Observer
import os
import time
from watchdog.events import FileSystemEventHandler


class Handler(FileSystemEventHandler):
    def on_modified(self, event):
        for filename in os.listdir(folder_track):
            file = folder_track + '/' + filename
            new_path = folder_dest + '/' + filename
            os.rename(file, new_path)
```

```
folder_track = '/Users/LAPTOP-LILIJA/Lilija Siniavska/atsiųsti failai'
folder_dest = '/User/LAPTOP-LILIJA/Lilija Siniavska/Darbalaukis/nauja'

handle = Handler()
observer = Observer()
observer.schedule(handle, folder_track, recursive=True)
observer.start()


try:
   while(True):
      time.sleep(10)
except KeyboardInterrupt:
   observer.stop()

observer.join()
```

7 priedas. Programos kodas rūšiavimas pagal požymius

```
from watchdog.observers import Observer
import os
import time
from watchdog.events import FileSystemEventHandler


class Handler(FileSystemEventHandler):
   def on_modified(self, event):
      for filename in os.listdir(folder_track):
         extension = filename.split(".")
         if len(extension) > 1 and(extension[1].lower() == "jpg" or extension[1].lower() ==
"png" or extension[1].lower() =="svg"):
            file = folder_track + '/' + filename
            new_path = folder_dest + '/' + filename
            os.rename(file, new_path)

folder_track = '/Users/LAPTOP-LILIJA/Lilija Siniavska/atsiųsti failai'
folder_dest = '/User/LAPTOP-LILIJA/Lilija Siniavska/Darbalaukis/nauja'

handle = Handler()
observer = Observer()
observer.schedule(handle, folder_track, recursive=True)
observer.start()


try:
   while(True):
      time.sleep(10)
except KeyboardInterrupt:
   observer.stop()

observer.join()
```

8 priedas. Žaidimo „Gyvatukas“ kodas

```
import pygame
import time
import random

pygame.init()

white = (255, 255, 255)
yellow = (255, 255, 102)
black = (0, 0, 0)
red = (213, 50, 80)
green = (0, 255, 0)
blue = (50, 153, 213)

dis_width = 800
dis_height = 600
dis = pygame.display.set_mode((dis_width, dis_height))
pygame.display.set_caption('Gyvatukas')
clock = pygame.time.Clock()
snake_block = 10
snake_speed = 15
font_tyle = pygame.font.SysFont("bahnschrift", 25)
font_style = pygame.font.SysFont("comicsansms", 35)

def Your_score(score):
    value = score.font_style("surinkote balų:" + str(score), True, yellow)
    dis.blit(value, [0, 0])

def our_snake(snake_block, snake_list):
    for x in snake_list:
        pygame.draw.rect(dis, black, [x[0], x[1], snake_block, snake_block])

def message(msg, color):
    mesg = font_style.render(msg, True, color)
    dis.blit(mesg, [dis_width / 6, dis_height / 3])

def gameLoop():
    game_over = False
    game_close = False
    x1 = dis_width / 2
    y1 = dis_height / 2
    x1_change = 0
    y1_change = 0
    snake_List = []
    Length_of_snake = 1
    foodx = round(random.randrange(0, dis_width - snake_block) / 10.0) * 10.0
    foody = round(random.randrange(0, dis_height - snake_block) / 10.0) * 10.0
    while not game_over:
        while game_close == True:
            dis.fill(blue)
```

```
        message("pralaimėjote! paspauskite Q norint išeiti iš žaidimo arba C norint tęsti", red)
        Your_score(Length_of_snake - 1)
        pygame.display.update()
        for event in pygame. event.get():
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_q:
                    game_over = True
                    game_close = False
                if event.key == pygame.K_c:
                    gameLoop()
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            game_over = True
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                x1_change = -snake_block
                y1_change = 0
            elif event.key == pygame.K_RIGHT:
                x1_change = snake_block
                y1_change = 0
            elif event.key == pygame.K_UP:
                y1_change = -snake_block
                x1_change = 0
            elif event.key == pygame.K_DOWN:
                y1_change = snake_block
                x1_change = 0
    if x1 >= dis_width or x1 < 0 or y1 >= dis_height or y1 < 0:
            game_over = True
    x1 += x1_change
    y1 += y1_change
    dis.fill(blue)
    pygame.draw.rect(dis, green, [foodx, foody, snake_block, snake_block])
    snake_Head = []
    snake_Head.append(x1)
    snake_Head.append(y1)
    snake_List.append(snake_Head)
    if len(snake_List) > Length_of_snake:
        del snake_List[0]
    for x in snake_List[:-1]:
        if x == snake_Head:
            game_close = True
    our_snake(snake_block, snake_List)
    Your_score(Length_of_snake - 1)
    pygame.display.update()
    if x1 == foodx and y1 == foody:
        foodx = round(random.randrange(0, dis_width - snake_block)/10.0) * 10.0
        foodx = round(random.randrange(0, dis_height - snake_block)/10.0) * 10.0
        Length_of_snake += 1
    clock.tick(snake_speed)
pygame.quit()
quit()
```

gameLoop()

9 priedas. Žaidimo „Kamuolys“ kodas

```
from tkinter import *
import time
import random

tk = Tk()
tk.title('Kamuolys')
tk.resizable(0, 0)
tk.wm_attributes('-topmost', 1)
canvas = Canvas(tk, width = 500, height = 400, highlightthickness = 0)
canvas.pack()
tk.update()

class Ball:
   def __init__(self, canvas, paddle, score, color):
      self.canvas = canvas
      self.paddle = paddle
      self.score = score
      self.id = canvas.create_oval(10, 10, 25, 25, fill = color)
      self.cnavas.move(self.id, 245, 100)
      starts = [-2, -1, 1, 2]
      random.shuffle(starts)
      self.x = starts[0]
      self.y = -2
      self.canvas_height = self.canvas.winfo_height()
      self.canvas_width = self.canvas.winfo_width()
      self.hit_bottom = False

   def hit_paddle(self, pos):
      paddle_pos = self.canvas.coords(self.paddle.id)
      if pos[2] >= paddle_pos[0] and pos[0] <= paddle_pos[2]:
         if pos[3] >= paddle_pos[1] and pos[3] <= paddle_pos[3]:
            self.score.hit()
            return True
         return False
      def draw(self):
         self.canvas.move(self.id, self.x, self.y)
         pos = self.canvas.coords(self.id)
         if pos[1] <= 0:
            self.y = 2
         if pos[3] >= self.canvas_height:
            self.hit_bottom = True
            canvas.create_text(250, 120, text = 'Pralaimėjote', font = ('Courier', 30), fill =
'red')
         if self.hit_paddle(pos) == True:
            self.y = -2
         if pos[0] <= 0:
            self.x = 2
         if pos[2] >= self.canvas_width:
```

```
            self.x = -2

class Paddle:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill = color)
        start_1 = [40, 60, 90, 120, 150, 180, 200]
        random.shuffle(start_1)
        self.starting_point_x = start_1[0]
        self.canvas.move(self.id, self.starting_point_x, 300)
        self.x = 0
        self.canvas_width = self.canvas.winfo_width()
        self.canvas.bind_all('<KeyPress-Right>', self.turn_right)
        self.canvas.bind_all('<KeyPress-Left>', self.turn_left)
        self.started = False
        self.canvas.bind_all('<KeyPress-Return>', self.start_game)

    def turn_right(self, event):
        self.x = 2

    def turn_left(self, event):
        self.x = -2

    def start_game(self, event):
        self.started = True

    def draw(self):
        self.canvas.move(self.id, self.x, 0)
        pos = self.canvas.coords(self.id)
        if pos[0] <= 0:
            self.x = 0
        elif pos[2] >= self.canvas_width:
            self.x = 0


class Score:
    def __init__(self, canvas, color):
        self.score = 0
        self.canvas = canvas
        self.id = canvas.create_text(450, 10, text = self.score, font = ('Courier', 15), fill =
color)
    def hit(self):
        self.score += 1
        self.canvas.itemconfig(self.id, text = self.score)



score = Score(canvas, 'green')
paddle = Paddle(canvas, 'White')
ball = Ball(canvas, paddle, score, 'red')
while not ball.hit_bottom:
    if paddle.started == True:
```

```
            ball.draw()
            paddle.draw()
        tk.update_idletasks()
        tk.update()
        time.sleep(0.01)
time.sleep(3)
```

10 priedas. Interneto greičio matuoklio kodas

```
    rom tkinter import *
    from speedtest import Speedtest

    def test():
        download = Speedtest().download()
        upload = Speedtest().upload()
        download_speedtest = round(download / (10**6), 2)
        upload_speedtest = round(upload / (10**6), 2)

        download_label.config(text = "Užkrovimo grietis:\n " + str(download_speed) + "MbPs")
        upload_label.config(text = "Atidavimo greitis:\n " + str(upload_speed) + "MbPs")

    root = Tk()

    root.title("Interneto greičio matuoklis")
    root.geometry("300x400")

    button = Button(root, text = "Paspausti, kad pradėti", font = 40)
    button.pack(side = BOTTOM, pady = 40)

    download_label = Label(root, text = "Užkrovimo grietis:\n- ", font = 35)
    download_label.pack(pady = (50, 0))
    upload_label = Label(root, text = "Atidavimo greitis:\n- ", font = 35)
    upload_label.pack(pady = (10, 0))

root.mainloop()
```