**Svarbiausia pitakas, toliau tai tik sportinis interesas (Karolis Šakinis):)**

**GENERALINIS RĖMĖJAS: Dalius Makackas **

**Pirma klausimų grupė (4 balai):**

1. Dinaminis programavimas (15 sk. 358 psl.). Algoritmų sudarymo metodika (15.3 sk. 379-389 psl ir šios metodikos taikymas sprendžiant Konvejerio (Surinkimo linijos planavimo) (15.1 sk. Antras knygos leidimas) arba Bendro ilgiausio posekio radimo (15.4 sk. 390-395 psl.) uždavinį (rekursinės lygtys, optimalaus sprendinio reikšmės ir struktūros radimas, algoritmo sudėtingumo radimas…).

**Dinaminis programavimas** – [programavimo](https://lt.wikipedia.org/wiki/Programavimas) metodas, paremtas uždavinio skaidymu į mažesnes susijusias problemas, bei tų problemų sprendimų įsiminimu. Taigi laiko sąnaudos pakeičiamos atminties sanaudomis.

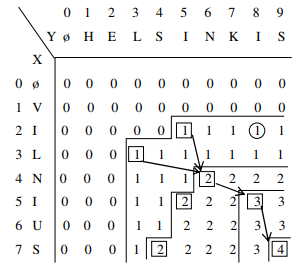
**1 pavyzdys:** X = 'V**IL**NIUS', Y = 'HE**L**SINKIS', σ = 32 (lietuviška abėcėlė), m = 7, n = 9 → S = 'INIS' arba 'LNIS', p = 4

Pavyzdyje negalime rasti ilgesnio negu keturių raidžių bendro X ir Y sekų posekio, kadangi trumpesnės sekos X simbolių 'V' ir 'U' sekoje Y niekur nėra, ir iš ženklų poros 'IL' galime išrinkti tik vieną raidę, nes abiejų tų simbolių nerandame toje pačioje tvarkoje sekoje Y. Kad gautumėme vieną iš bip posekių, reikia naikinti simbolius 'V', 'U' ir 'L' arba pirmąją 'I' iš sekos 'VILNIUS' ir simbolius 'H', 'E', pirmąją 'S', 'K' ir 'L' arba pirmąją 'I' iš sekos 'HELSINKIS'.

Naudojant Vagnerio ir Fischerio tiesmukišką, vadinamąjį brutalios jėgos, algoritmą, kuris remiasi dinaminiu programavimu, galime išspręsti problemą palyginant pasirinktą sekos X simbolį su visais sekos Y simboliais.

Rekursyvi taisyklė skaičiuoti įvestų sekų prefiksų bip ilgį:

Tiesmukiškas būdas išspręsti viršutinę rekursiją yra taikyti dinaminį programavimą ir užpildyti dvimatę lentelę R[0..m][0..n].



Sužinojus algoritmo skaičiavimo taisykles yra lengva suprasti, kad ieškant sekų X ir Y bip ilgio realybėje mums nereikėtų registruoti kitų matricos M vietų negu tas, kur yra minimalūs sutapimai. Jeigu žinome bip ilgio žemesnę ribą iš anksto, galėtume ignoruoti net dalį iš minimalių sutapimų . Taigi, Vagnerio ir Fischerio algoritmas daro kiekvieną kartą Ο(mn) žingsnius rašant r(i,j) skaičius visoms prefiksų poroms – nepaisant sekų savybių. Nors beveik visų bip algoritmų sudėtingumas yra blogiausiai lygio Ο(mn), dažniausiai daug iš jų dirba praktiškai labai greičiau. Svarbiausia savybė, kalbant apie efektyvumą yra, kad algoritmas rinktų ir saugotų tik problemai esminę informaciją. Reikia išrinkti gerai tinkamą duomenų struktūrą, bet yra svarbu rūpintis taip pat tuo, kad struktūros pastatymas netaptų per brangus.

2. Dinaminis programavimas (15 sk. 358 psl.). Algoritmų sudarymo metodika (15.3 sk. 379-389 psl.) ir šios metodikos taikymas sprendžiant strypų pjaustymo uždavinį (15.1 sk. 379-389 psl.) (rekursinės lygtys, optimalaus sprendinio reikšmės ir struktūros radimas, algoritmo sudėtingumo radimas…).

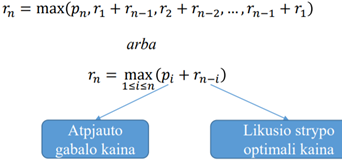
Uždavinį galime skaidyti taip: atpjauti strypą, paimti atpjauto strypo kainą ir likusiam strypui apskaičiuoti optimalią kainą. Iš to gauname rekursinę formulę:



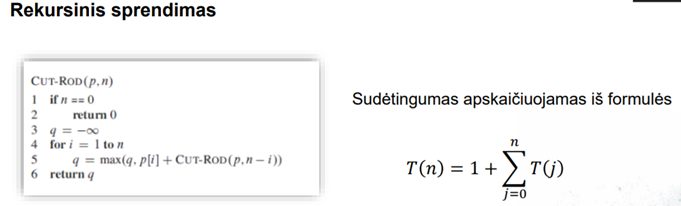
Apskaičiuoti optimalią reikšmę galime taip:

* Pradedame nuo vienetinio strypo ilgio, jo rastą kainą išsaugome
* Ieškome sekančio ilgio strypo optimalios kainos įvairiais būdais dalindami strypą ir buvusio ilgio strypo kainą imdami iš išsaugotų kainų masyvo. Kartojame kol pasieksime duoto strypo ilgį.

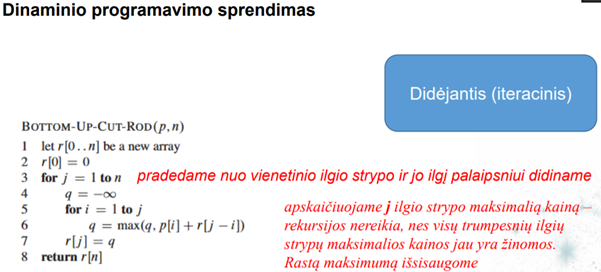
Sudėtingumas: kadangi skaičiuojame kiekvieno ilgio strypui kainą, padalinant visais įmanomais būdais kiekvieną strypą, bus naudojami du ciklai ir algoritmų sudėtingumas bus Ɵ(n2).



n-pradinio strypo ilgis, i-atpjauto ilgis, p-kaina atpjautam strypui, r-max kaina tam tikro ilgio strypui(kad ir supjaustytam)



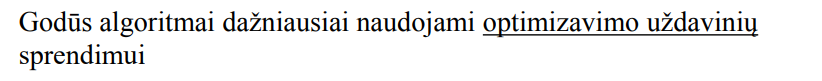


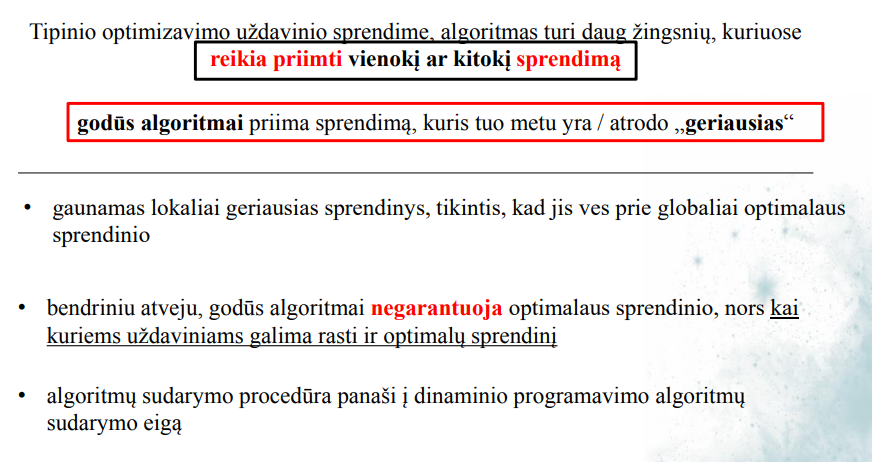




3. Dinaminis programavimas (15 sk. 358 psl.). Algoritmų sudarymo metodika (15.3 sk. 379-389 psl.) ir šios metodikos taikymas sprendžiant Matricų sekos optimalaus dauginimo uždavinį (15.3 sk. 379-389 psl.) (rekursinės lygtys optimalaus sprendinio reikšmės ir struktūros radimas, algoritmo sudėtingumo radimas…).

4. Godūs algoritmai (16 sk. 414psl.). Maksimalios procesų aibės radimo uždavinys (16.1 sk. 415-418 psl.) ir jo sprendimas (rekursinės lygtys, optimalaus sprendinio reikšmės ir struktūros radimas, algoritmo sudėtingumo radimas…) (16.2 sk. 423-427 psl.).





Godaus algoritmo elementai

Principai tokie patys kaip ir dinaminio programavimo algoritmų.

1. Randama uždavinio optimali struktūra.

2. Sudaromas rekursinis sprendimas.

3. Parodoma, kad esant godžiai strategijai lieka tik vienas

pagalbinis uždavinys.

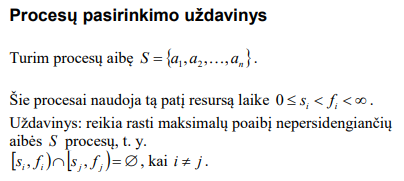
4. Parodoma (įrodoma), kad godus pasirinkimas yra

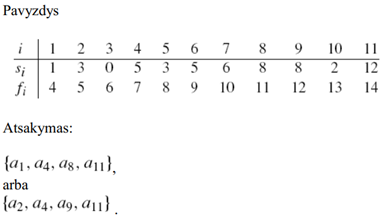
saugus.

5. Sudaromas rekursinis algoritmas, realizuojantis godžią

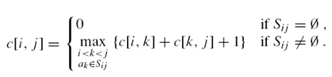
strategiją.

6. Rekursinis algoritmas transformuojamas į iteracinį.





Rekursine lygtis:

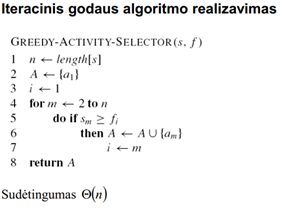


Užduoties paaiškinimas: Duota procesų aibė S su procesų pradžios laikais(s masyvas) ir pabaigos laikais(f masyvas). Rasti maksimalią nepersidengiančių procesų aibę.

Sprendimas:

Susidarome matricą kurioje saugosime procesus; į matricą įdedame pirmą procesą(pradžios ir pabaigos laikus); nuo antro proceso ieškome pirmo sutikto proceso, kurio pradžios laikas yra didesnis arba lygus matricoje esančio(pirmojo) proceso pabaigos laikui; rastą procesą įrašome į matricą; toliau ieškome nuo paskutinio rasto proceso vietos; kartojame kol praeisime visus procesus.

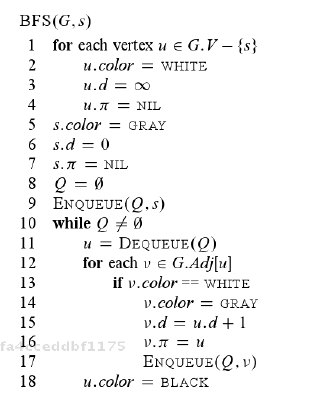
Algoritme naudojame tik vieną ciklą, kuris eis per matricą vieną kartą, todėl sudėtingumas yra Ɵ(n), t.y. procesų kiekis masyvuose s ir f.



**Antra klausimų grupė (2 balai):**

***1. Paieškos į plotį algoritmas (22.2 sk. 594-597 psl.) ir sudėtingumo įvertinimas.***

**Paieška į plotį** (*breadth-first search*, *BFS*) – paieškos grafe arba grafo apėjimo būdas, kai pasirinkus pradinę viršūnę pirmiausia yra aplankomos visos jos kaimynės (t. y., viršūnės, sujungtos grafo briaunomis su pradine viršūne), po to kaimynių kaimynės ir t. t., kol randamas ieškomas tikslas arba kol yra apeinamos visos grafo viršūnės ir briaunos.



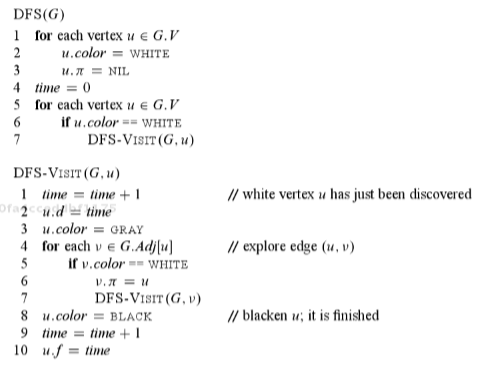
Korektiška sakyti, jog sudėtingumas priklauso nuo viršūnių ir briaunų, todėl sudėtingumas yra

**O(V + B)**. (Kiekvienai aplankytai viršūnei reikia suskaičiuoti gretimas viršūnes, o tai galima padaryti tik einant per briaunas)

***2. Paieškos į gylį algoritmas (22.3 sk. 603-606 psl.) ir sudėtingumo įvertinimas.***

**Paieška į gylį** (*depth-first search* ar *DFS*) – paieškos grafe arba grafo apėjimo būdas, kai pasirinkus pradinę viršūnę einama grafo briaunomis kiek įmanoma giliau, renkantis vis naują viršūnę; kai paskutinė aplankyta viršūnė naujos (dar neaplankytos) kaimynės neturi, tada grįžtama iki artimiausios neaplankytos briaunos ir vėl ieškoma kuo giliau tol, kol bus rastas ieškomas tikslas arba kol bus aplankytos visos grafo viršūnės ir briaunos.

Taikymų pavyzdžiai: patikrinti, ar grafas jungus, identifikuoti neorientuoto grafo jungumo komponentes, patikrinti, ar egzistuoja kelias grafe iš viršūnės u iki viršūnės v.

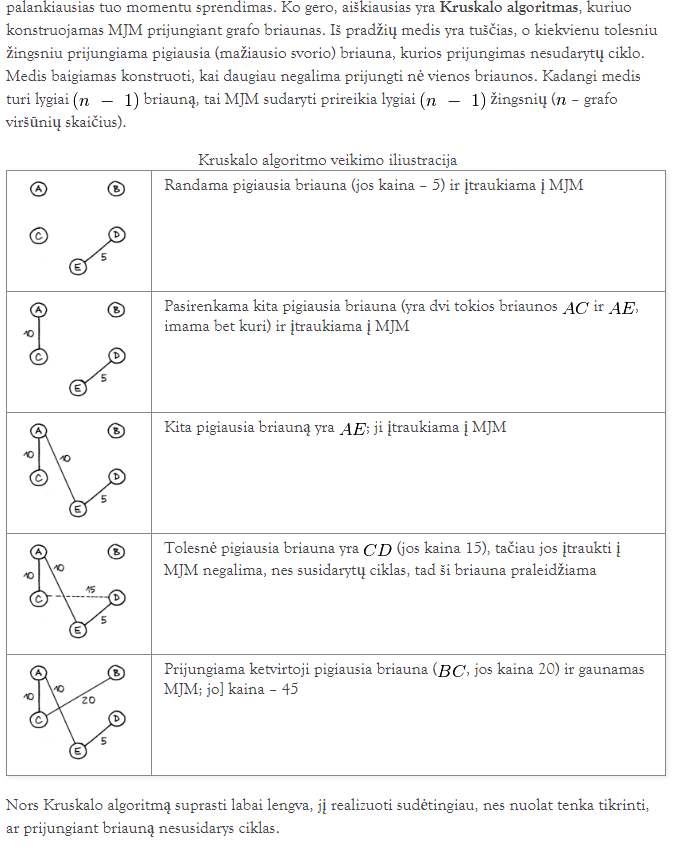


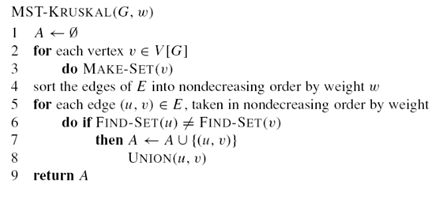
norint nustatyti ar grafas jungus, jis bus jungus tuomet jei Lankyk(viršūnė) išoriniame cikle bus iškviestas tik vieną kartą, galima Lankyk iškviesti su bet kuria viršūne ir vėliau patikrinti, ar aplankytos visos viršūnės.  
norint nustatyti trumpiausią kelią ne svoriniame grafe nuo duotosios viršūnės t iki visų grafo viršūnių. Tuomet Lankyk(I) procedūrą reikėtų šiek tiek praplėst

Korektiška sakyti, jog sudėtingumas priklauso nuo viršūnių ir briaunų, todėl sudėtingumas yra

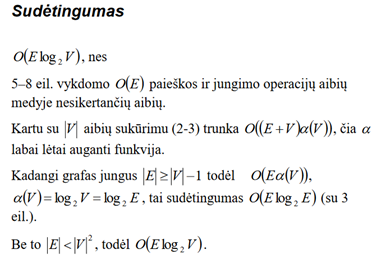
**O(V + B)**. (Kiekvienai aplankytai viršūnei reikia suskaičiuoti gretimas viršūnes, o tai galima padaryti tik einant per briaunas)

***3. Kruskalo algoritmas (23.2 sk. 631-633 psl.) ir sudėtingumo įvertinimas.***

**-**

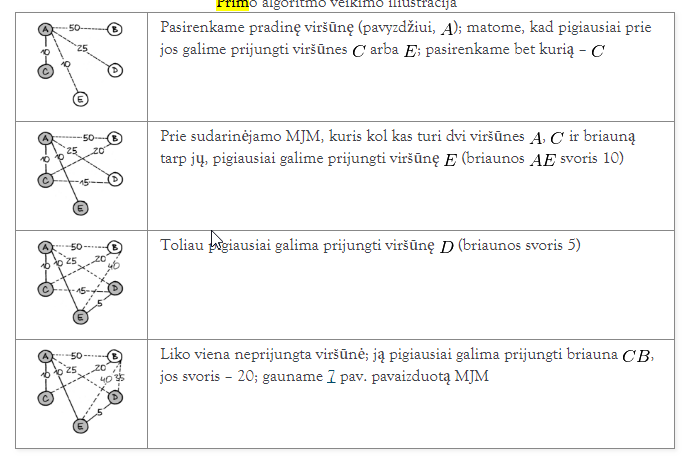


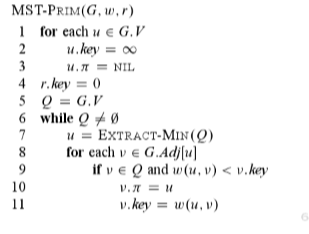
Pradžioje turime tuščią aibę. Turime išsirikuoti briaunas pagal svorį didėjimo tvarka.Tada, kol įmanoma, atliekama tokia operacija: iš visų briaunų, kurias įjungus prie jau parinktų, nesusidaro ciklas, išrenkama mažiausio svorio briauna. Kai tokių briaunų nebelieka, algoritmas baigia darbą. Tada pografis, kurį sudaro visos jo viršūnės ir surastos briaunos, sudaro grafo karkasinį medį.



***4. Prima algoritmas (23.2 sk. 634-636 psl.) ir sudėtingumo įvertinimas***

**Primo algoritmu** konstruojamas prijungiant grafo briaunas, tačiau pradedama nuo medžio, kurį sudaro viena laisvai pasirinkta viršūnė. Prijungiamoji briauna taip pat turi būti pigiausia, tačiau lygiai viena briaunos viršūnė turi priklausyti konstruojamam medžiui. Ši sąlyga garantuoja, kad prijungiant briauną nesusidarys ciklas.





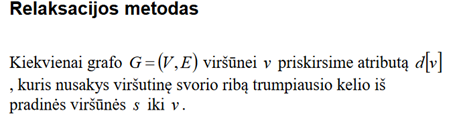
Efektyvumas priklauso nuo prioritetinės eilės realizacijos. Naudodami *binary heap* gausime O(|E| log |V|), naudodami *Fibonacci heap* gausime O(|E| + |V| log |V|).

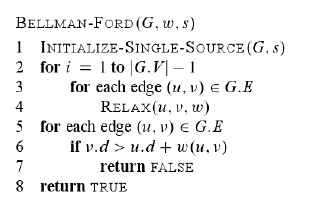
***5. Trumpiausi keliai iš vienos viršūnės (24 sk. 641-650 psl.). Relaksacijos metodas.***

***Belmano –Fordo algoritmas (24.1 sk. 651 psl.). Sudėtingumo įvertinimas.***

**Trumpiausio kelio problema** – grafų teorijos problema, bendru atveju formuluojama kaip radimas tokio kelio tarp dviejų svorinio grafo (arba daugiau) viršūnių, kad briaunų svorių suma būtų mažiausia.

Trumpiausiam kelyje ciklai negalimi. Jei yra ciklas teigiamas trumpiausiam kelyje, jį pašalinus rasime dar trumpesnį kelią. Nulinio svorio ciklus galime ignoruoti.

******

******

Algoritmas rasti trumpiausią kelią iš viršūnės s iki visų kitų viršūnių, kai briaunų svoriai yra skirtingi.

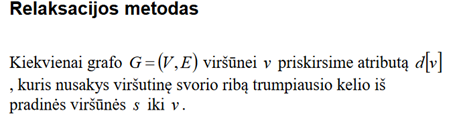
Algoritmas gali atpažinti neigiamo svorio ciklo egzistavimą.

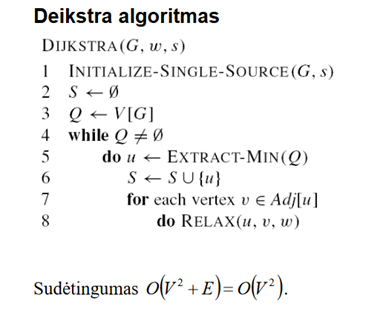
Efektyvumas -- O(V\*E), nes O(V) - inicializacija in line 1, o O(E), nes for ciklas line 5-7

***6. Trumpiausi keliai iš vienos viršūnės. Relaksacijos metodas. Deikstra algoritmas (24.3 sk. 658-659 psl.). Sudėtingumo įvertinimas (24.3 sk. 661-662 psl.).***

**Trumpiausio kelio problema** – grafų teorijos problema, bendru atveju formuluojama kaip radimas tokio kelio tarp dviejų svorinio grafo (arba daugiau) viršūnių, kad briaunų svorių suma būtų mažiausia.

Trumpiausiam kelyje ciklai negalimi. Jei yra ciklas teigiamas trumpiausiam kelyje, jį pašalinus rasime dar trumpesnį kelią. Nulinio svorio ciklus galima ignoruoti.

******

****

Grafas gali būti orientuotas arba neorientuotas, negali būti neigiamu viršūnių grafas turi būti jungus.

Inicializuojama pradzios virsune

Sukuriama struktura žymėti aplankytom virsunem, S virsune pažymima kaip aplankyta.

Kol yra neaplankytu virsuniu vykdoma:

Čia naudojama prioritetinė eilė. Operacija extract\_min randa elementą u, kurio svoris (distance[u]) yra mažiausias.

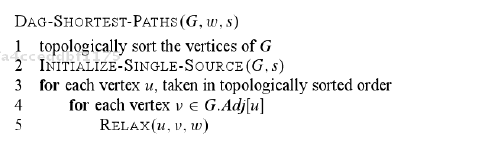
Ir kiekviena virsunej atliekamas relaksacijos metodas

Ir viršūnė pažymima kaip aplankyta

Gaunamas sudetingumas O(V^2).

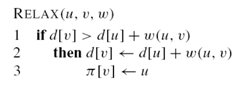
***7. Trumpiausių kelių paieška iš vienos viršūnės orientuotame acikliniame grafe ir sudėtingumo įvertinimas. (24.2 sk. 665 psl.).***

*Algoritmas:*



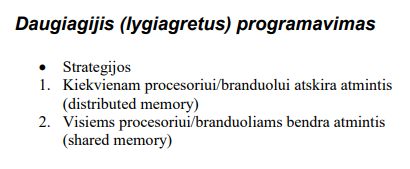
Algoritmas rasti trumpiausią kelią iš viršūnės s iki visų kitų viršūnių, kai briaunų svoriai yra skirtingi.  
Grafas turi būti beciklis (DAG'as).

1 eilute visa grafiką paverčia topologiniu grafu (O(V+E)).2 eilute užtrunka O(v). 3-5 iteruoja kiekvieną viršūnę. Susumavus 4-5 eilutės atpalaiduoja kiekvieną kraštinę viena karta. Kiekvienas vidinis for ciklas užtrunka Ɵ(1)laiko, tad sudetingumas O(V+E).



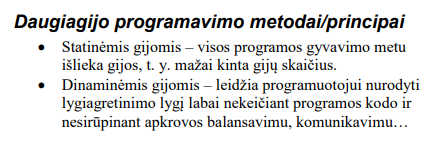
**Trečia klausimų grupė (4 balai):**

***8. Daugiagijo dinaminio programavimo metodika. (27 sk. 772-791 psl.)***

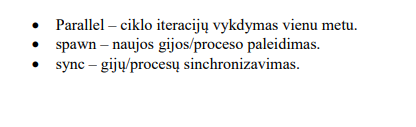
****

***Paskirstyta atmintis*** - kiekvieno procesoriaus atmintis yra privati ir turi būti siunčiama atvira žinutė tarp procesorių kad būtų galima pasiekti kito procesoriaus atmintį.

***Bendrai naudojama atmintis*** - kiekvienas procesorius gali tiesiogiai prieiti bet kurią atminties vietą.

****

Daugiagijis dinaminis programavimas leidžia programeriams apibrėžti lygiagretumą aplikacijose per daug negalvojant apie komunikacijų protokolus, apkrovos balansavimą ir apie kitas statinių gijų programavimo užgaidas.



Pagrindiniai terminai naudojami DDP - pavyzdys - Fibonačio skaičiaus skaičiavimas.

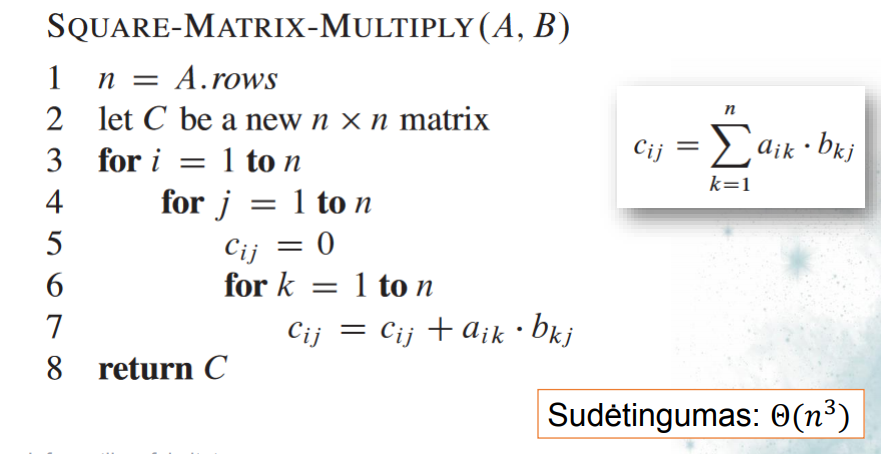
Gijos - virtualių procesorių programų abstrakcija

* Gijos dalinasi bendra atmintimi
* Prižiūri savo stack’ą ir programų skaičių
* Vykdo kodą nepriklausomai nuo kitų gijų
* OS užkrauna giją į procesorių, įvykdo jį ir pakeičia į kitą giją kai to reikia.

Lygiagrečios platformos:

* Nested lygiagretumas - leidžia įjungti paprogrames
* Planuoja - turi schedulerį (automatinis load-balance skaičiavimas)
* Turi lygiagrečius ciklus - ciklų iteracijos gali būti vykdomos lygiagrečiai

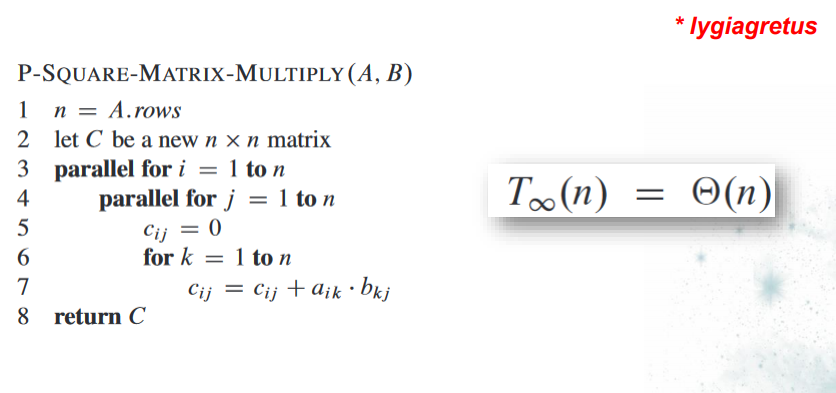
***9. Daugiagijai matricų dauginimo algoritmai ir jų vykdymo laikų bei lygiagretinimo koeficientų įvertinimas. (27.2 sk. 792-797 psl.)***

****

**Tekstinis variantas, kuris padėjo gauti 10:**

**Pasileidžiame du ciklus nuo 1 iki n (i ir j kintamieji) tada c matricos elemnta i,j verciamę i nulį, darome trečia ciklą K nuo 1 iki n. Tada c i,j elemntas lygus c i,j sudečiai su a i,k ir b k,j sandaugai. Sudėtinguma matot.**

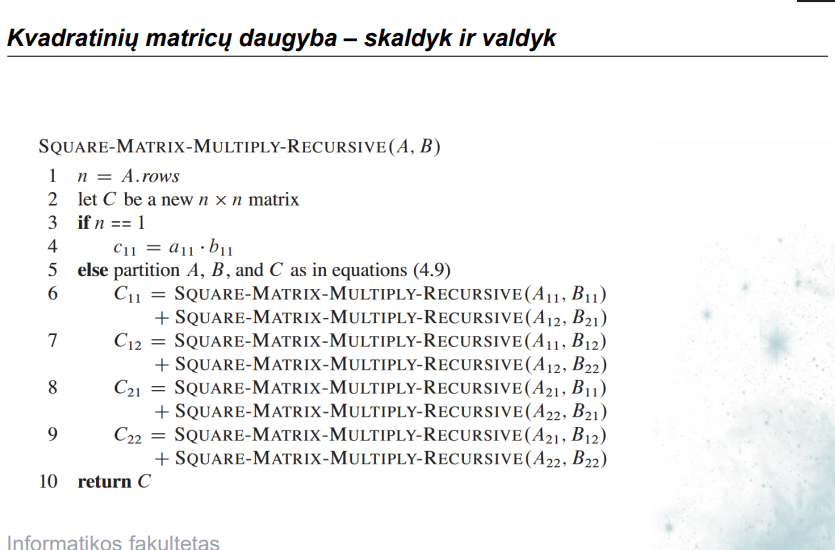
Lygiagretaus programavimo:



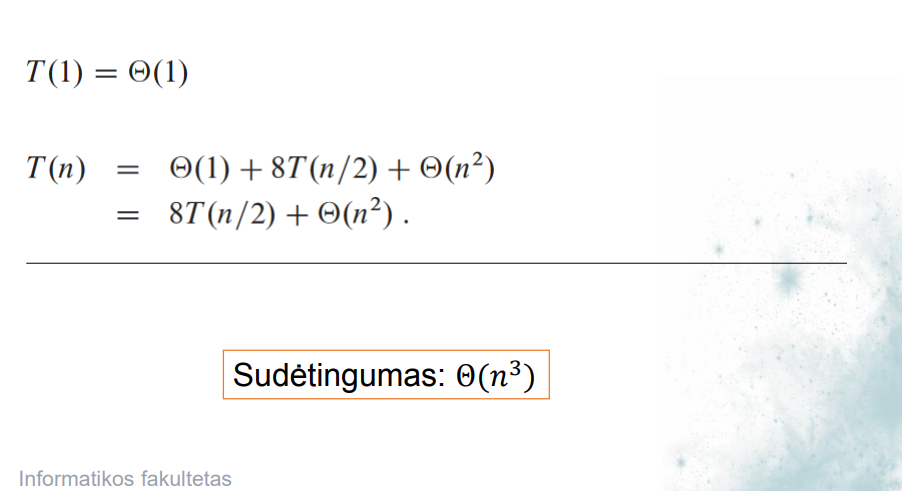
Viskas tas pats kaip ir paprastam tik parareliškai paleidžiam ciklus i ir j, taip sumažindami iki teta(n) sudetingumo, jis krenta, nes realiai tų ciklų sudėtingumas virsta 1.

Išlygiagretinimo Rezultatas: 

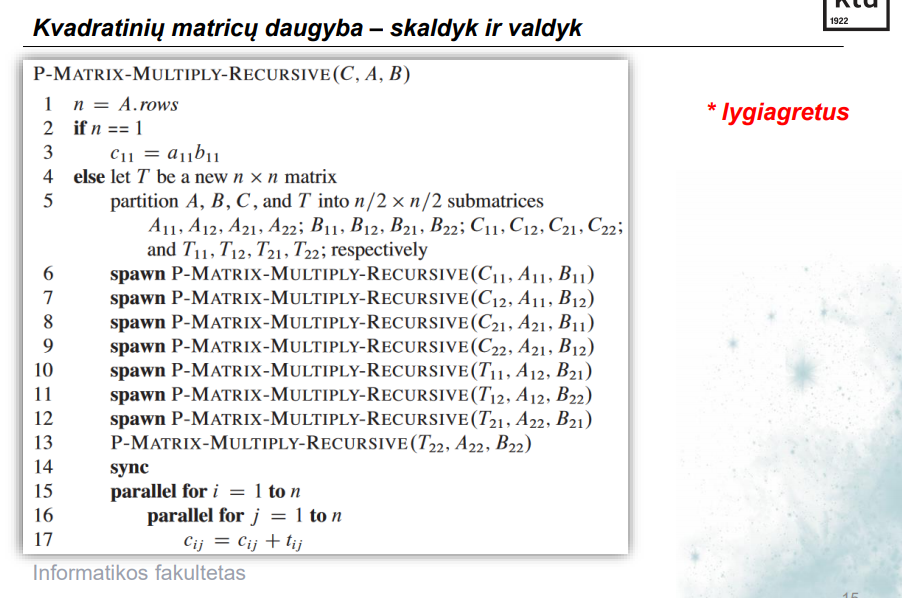
Skaldyk ir valdyk:



Prisikiriame n a eilučių kiekį. Tada susikuriame naują matricą C, kur bus n x n dydžio. Pasirašome rekursijos stabdį, kai n= 1, tuo atveju c elemntas lygus a ir b elemento sandaugai, kitu atveju dalinamę a,b,c matricas į n/2\*n/2 dydžio matricas, tad realiai gauname iš vienos matricos 4. Ir tada kviečiame reukrsine sudėti su skirtingomis matricu dalių sudėtimis. Pasibaigus visam rekursijos procesui turesime apjungti C dalys į viena vientisą n x n dydžio matricą. Taip gausime pilnai sudaugintą matricą.

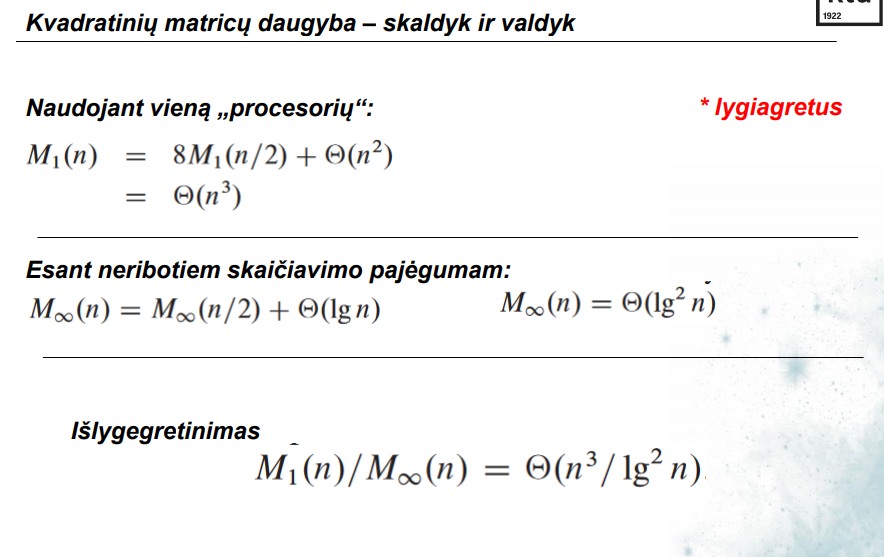


Lygiagretus:



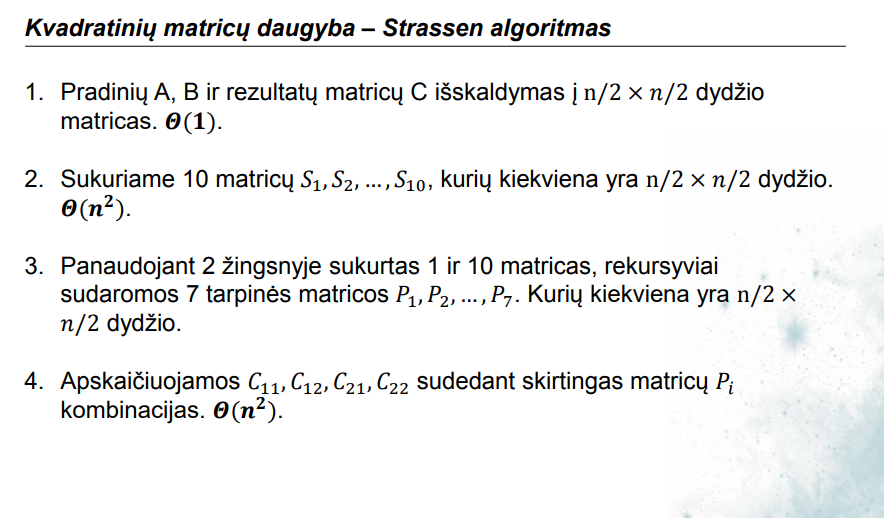
Prisikiriame n reikšmei a eilučių kiekį. Tada susikuriame naują matricą T, kuri bus n x n dydžio. (C nebekuriame, nes vis persisiunčiame rekursiskai į metodą, tad jį turime susikurti pirma karta main metode).Pasirašome rekursijos stabdį, kai n= 1, tuo atveju c elemntas lygus a ir b elemento sandaugai, kitu atveju dalinamę a,b,c,t matricas į n/2\*n/2 dydžio matricas.

Susikuriame 8 metodus, kurie rekursiskai daugins matricas, pirmuose 4 siusime A ir B submatricas kartu su C submatricom, o kiti 4 metodai skaičiuos T submatricos reikšmes su A ir B submatricom, (lygegriačiai skaičiuos T ir C elementus). Po visų lygegriačių rekursijų gausime dvi matricas C ir T, kurios turės dalį rezultatų, tad juos turime apjungti į C matricą per dvigubą ciklą, kad apeitumę visą matricos dydį n x n (T matricą papildomai atsirantas, nes pvz C ims A11 B11, o T ims A12,B12, o C nebedarys ka darė T, taip paskirstis darbą procesoriui).

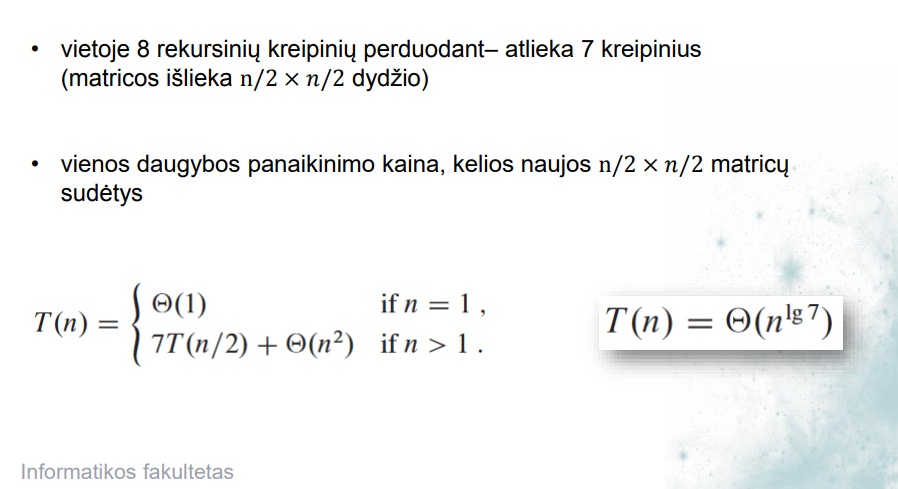


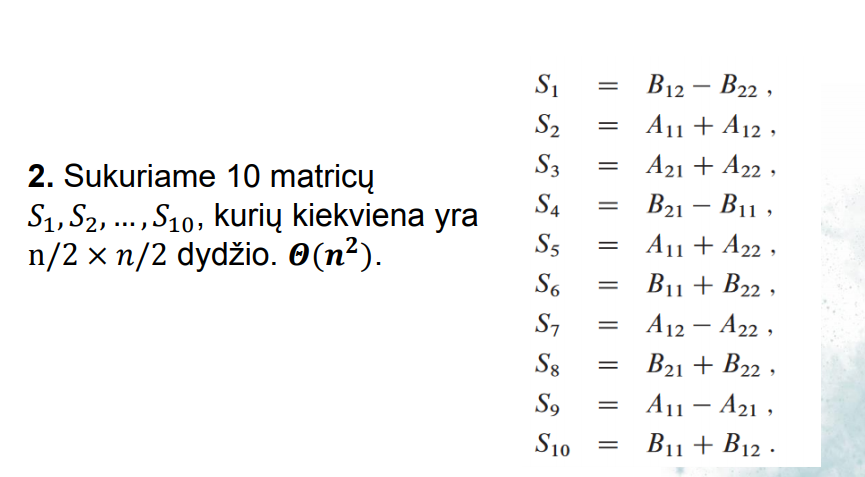
Čia mintinai išmokit, nes ką žinau kaip čia palengvinti.

Strassen metodas:

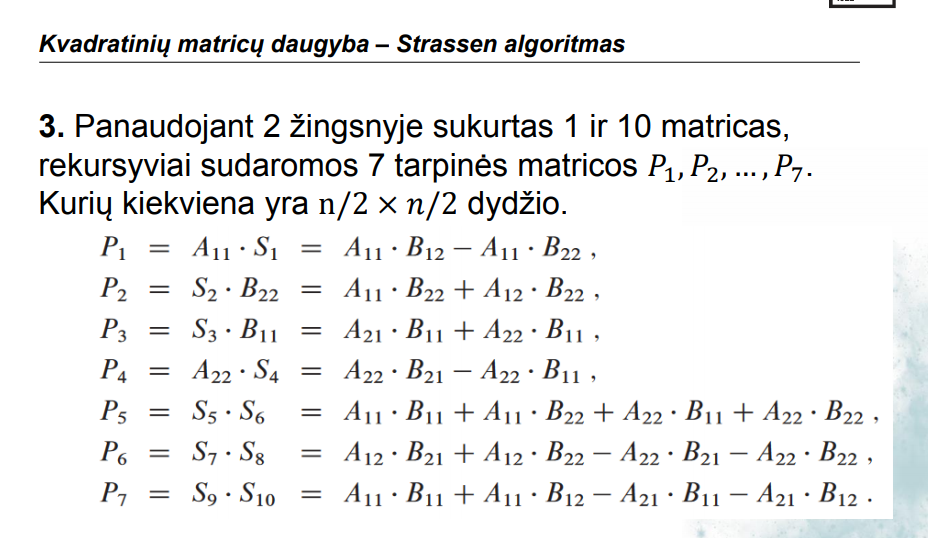


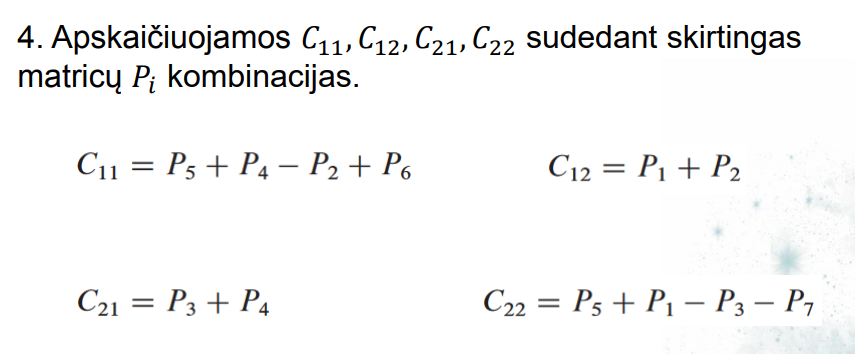
Realiai parašiau kas viršuje nuotraukos ir apačioje esančias lygtis su paaiškinimu.



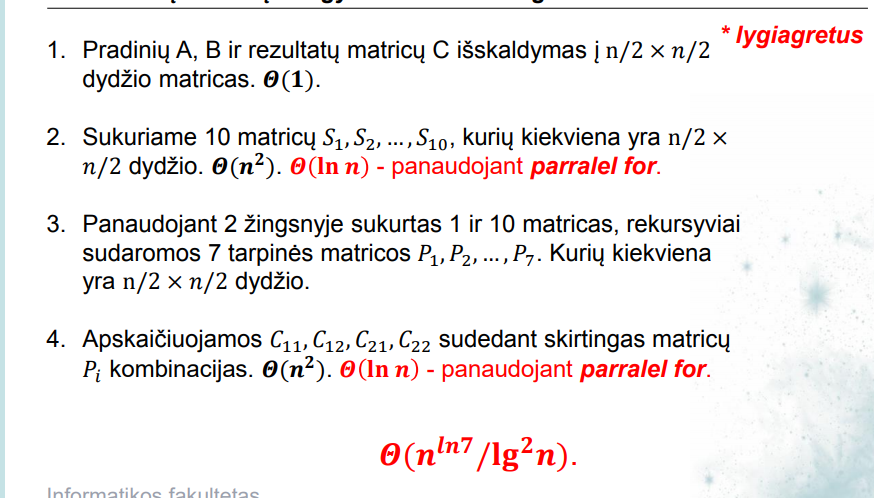


Neverta išmokti apačioje esančios ir viršuje esančios foto, nes čia rodo kaip veikia.



Same here, neverta mokytis.(Jei reiks gyvenime yra puslapis [www.google.com](http://www.google.com), labai zjbs šaltinis, tačiau abidna, kad nepadėjo man išmokti rašyti be klaidų :( )

Lygiagretus:

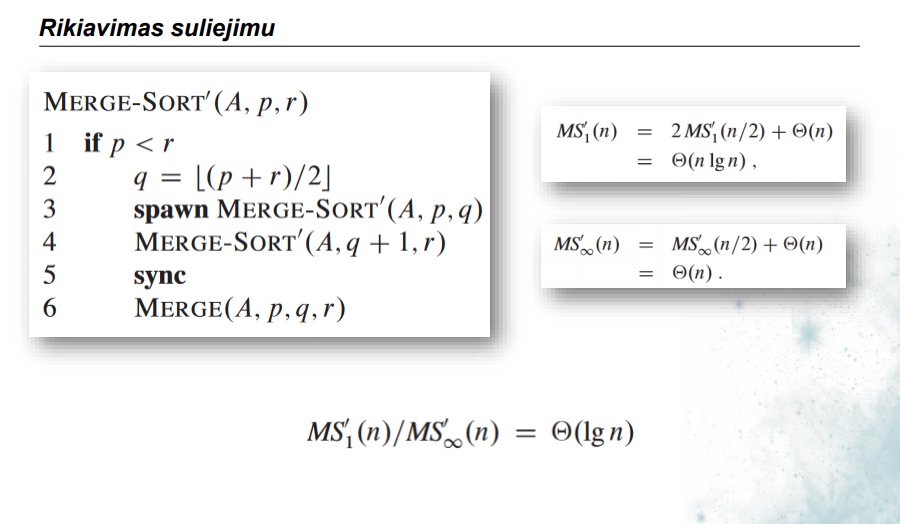


Nu Čia irgi iškalt mintinai, tik apačioje parašytas teisingas sudėtingumas.

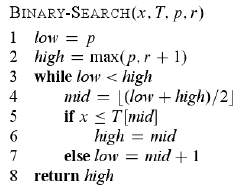
Parallel for duoda lg(n), o ne ln(n). (Nesugeba net nurašyti nuo knygos gerai :( )

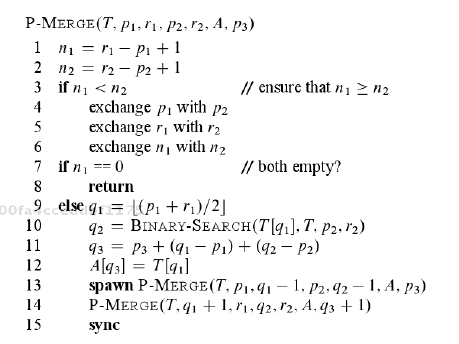
**Jei daskaitėt iki tiek, tai zjbs, jei ne tai bbz. Klauskit ko nesupratot iš šito uždavinio, nes geriausiai mokėjau iš jo.(Rašo žmogus, kuris įkėlė šį konspektą į FB)**

***10. Daugiagijai rikiavimo algoritmai ir jų vykdymo laikų bei lygiagretinimo koeficientų įvertinimas. . (27.3 sk. 797-804 psl.)***

******

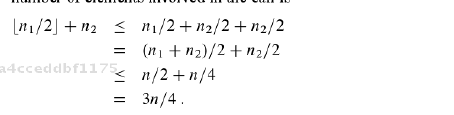
**Binarinio medzio:**

****

****

1-2 eilutėmis apskaičiuojame T1 ir T2 masyvų ilgį. 3-6 eilutės perša mintį, kad n1 turi būti didesnė už n2. 9-15 eilutės naudoja skaldyk ir valdyk principą. 9 suranda vidurio taska T1 masyve, o T2 randa toki taską, kuriame T2[p2..q2-1] yra mažiau nei T[q1] (kas yra X) ir taskus T2[q2..r2], kurie yra bent didesni už T[q1]. 11 padalina gautus masyvus į A[p3..q3-1] ir A[q3+1..r3] matricas ir 12 eilute perkopijuoja T[q1] tiesiai į A[q3]. Tada naudojama rekursiine nested *daugiagijyste.13 sprendžia viena subproblemą, o 14 sekančia subproblemą. 15 eilutė sinchronizuoja rezultatus prieš išeinant iš kodo.*

*Tad gauname:*

******

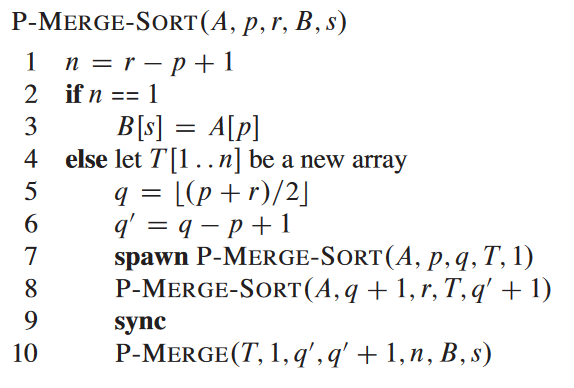
*Dar pridedant, kad kvieciant binary-search kainuoja* Ɵ(lg n) gauname (išsprendus medžio būdu):

****

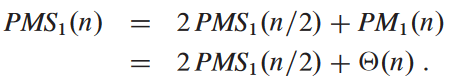
**Lygiagretumas lygus:**



Kai išanalizavom suliejimo procedūrą, dabar galime jį pritaikyti daugiagijystės suliejimo rūšiavimui. Ši versiją yra panaši į merge sort, kuria dareme, tačiau šitoje versijoje B masyvo išvedimą priima kaip argumentą, kuris palaiko surušiuotą masyvą.

****

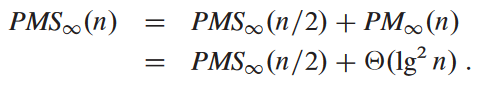
**Normalus darbas neįvertinant gijų:**

****

****

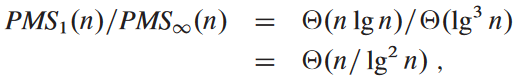
Gijų laiko įvertinimas:

(PMSinf(n/2), nėra 2PMSinf(n/2), nes naudojama gijas ir jos vyks tuo pačiu metu)

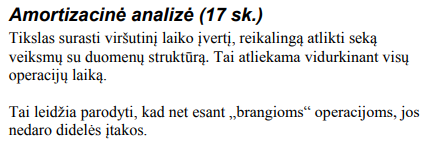
****

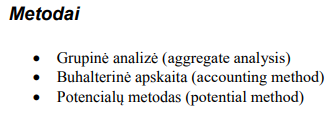
****

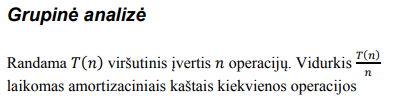
P-MERGE SORT lygiagretinimo koeficientas:

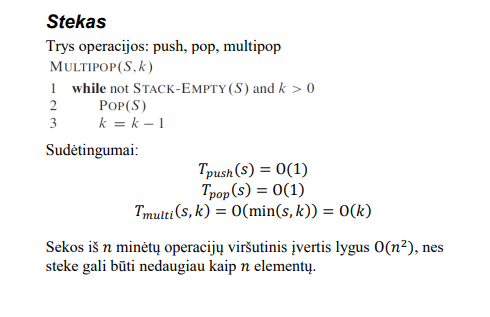
****

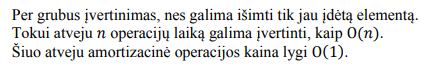
***11. Amortizacinė algoritmų analizė. (17 sk. 451-462 psl.)***

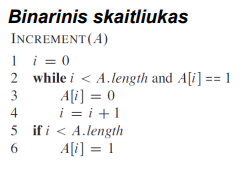


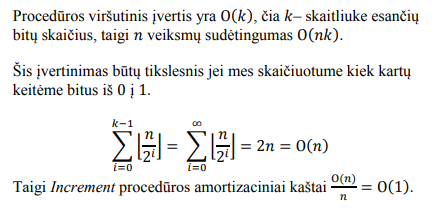












***Buhalterinė apskaita***

Agregatinis metodas tiesiogiai nustato bendrą operacijų sekos vykdymo laiką. Priešingai, apskaitos metodu siekiama rasti papildomų laiko vienetų, mokamų kiekvienai atskirai operacijai, sumokėti taip, kad mokėjimų suma būtų viršutinė riba bendrų visų išlaidų. Intuityviai galima galvoti apie banko sąskaitos tvarkymą. Pigios operacijos yra apmokestinamos šiek tiek daugiau negu jos iš tikro kainuoja, o perteklius yra pervedamas į banko sąskaitą vėlesniam naudojimui. Tada brangios operacijos gali būti apmokestinamos mažesnėmis už jų tikras išlaidas, o deficitas mokamas iš taupymo banko sąskaitoje. Tokiu būdu mes paskirstome didelių sąnaudų operacijų išlaidas per visą seka. Kiekvienos operacijos mokesčiai turi būti tokie dideli, kad banko sąskaitos balansas visada išliktų teigiamas, bet pakankamai mažas, kad nė viena operacija nebūtų apmokestinta žymiai daugiau už jo tikras išlaidas.

Mes pabrėžiame, kad papildomas operacijos laikas nereiškia, kad operacija užtrunka tiek daug laiko. Tai tik apskaitos metodas, kuris leidžia lengviau analizuoti.

**Potencialų metodas**

Tarkim, kad galime apibrėžti potencialų funkciją Φ duomenų struktūros būsenoje su tokiomis savybėmis:

Φ(h0) = 0, kur h0 yra pradinė duomenų struktūros būsena

Φ(h0) >= 0 visoms būsenoms ht duomenų struktūrai visoje skaičiavimo eigoje.

Intuityviai, potenciali funkcija seks iš anksto nustatytą laiką viso skaičiavimo metu. Ji nustato kiek sutaupyto laiko yra prieinama apmokėti brangioms operacijoms. Šis metodas yra analogiškas buhalterinei apskaitai. Įdomiausia tai, kad funkcija priklauso tik nuo dabartines duomenų struktūros būsenos, nepriklausomai nuo skaičiavimo istorijos kaip ji pateko į tą būseną.

Tada nustatome amortizuojama operacijos laiką kaip

****

Kur c - tikra operacijos kaina, o h ir h` yra duomenų struktūros būsenos prieš ir po operacijos.

Idealiai Φ turi būti nustatytas taip, kad kiekvienos operacijos amortizuojamas laikas būtų mažas.

Dinaminiams, neapibrėžto dydžio, masyvams su dvigubu padidinimu, galime naudoti potencialų funkciją



Kur n yra dabartinis elementų skaičius, o m dabartinis masyvo ilgis. Jei pradėsim nuo masyvo ilgio 0 ir nustatysim jo ilgį 1, kai pirmas elementas yra pridėtas, o vėliau padvigubinsim masyvą reikės daugiau vietos, turim Φ(ht) = 0 ir Φ(ht) >= 0 visoms t. Paskutinė nelygybė veikia, nes elementų skaičius yra visada bent pusė masyvo dydžio.

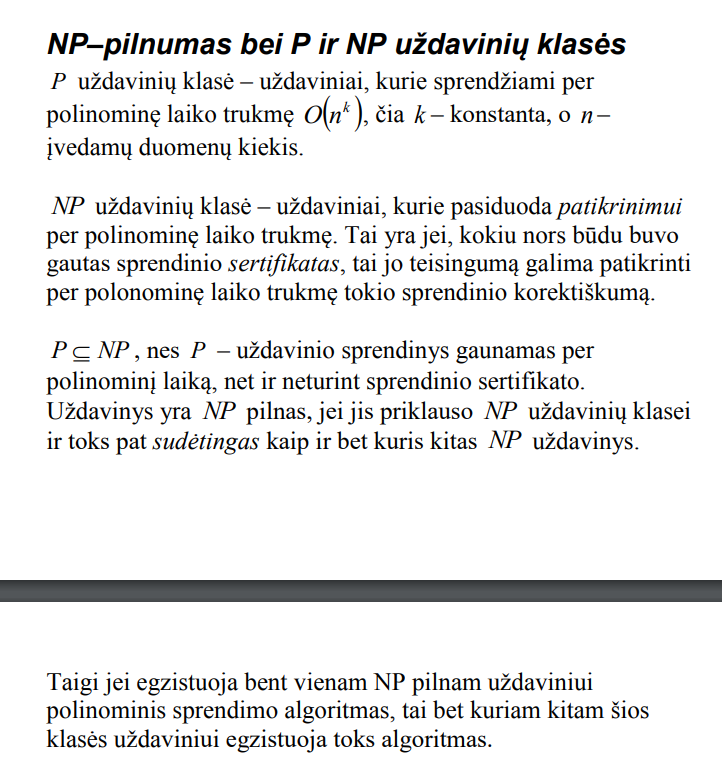
Pridedant elementus yra naudojama amortizuojamo laiko konstanta. Yra 2 atvejai:

* Jei n < m, tada tikra kaina yra 1, n padidėja per 1, o m nesikeičia. Tada potencialas padidėja 2, o amortizuojamas laikas yra 1 + 2 = 3
* Jei n = m, tada masyvas yra padvigubintas, tada tikras laikas yra n+1. Bet potencialas sumažėjo nuo n iki 2, todėl amortizavimo laikas yra n + 1 + (n-2) = 3.

Abejais atvejais, amortizavimo laikas yra O(1).

Esminis dalykas potencialų metode yra nustatyti teisingą potencialų funkciją. Potencialų funkcija turi sutaupyti pakankamai laiko tolimesniam naudojimui, kai to reikia. Bet ji neturi sutaupyti tiek laiko kad amortizuojamas laikas dabartinei operacijai būtų per didelis.

***12. NP sudėtingumas. Ir P ir NP klasės. Uždavinių pavyzdžiai. (34 sk. 1048-1053 psl.)***

******

*Jau nagrinėjome kelis tokius uždavinius:  
1. Keliaujančio pirklio uždavinys. Duotas svertinis grafas G = (V,E),  
reikia rasti trumpiausią pirklio maršrutą, kai jis po vieną kartą aplanko  
visas grafo viršūnes ir grįžta į pradinę viršūnę.  
2. Hamiltono ciklas. Reikia patikrinti ar duotajame grafe G = (V,E)  
egzistuoja ciklas, jungiantis visas jo viršūnes.  
3. Diskretusis kuprinės užpildymo uždavinys. Turime n daiktų, kurių  
tūriai yra v1,v2,... ,vn, o kaina p1,p2,... ,pn. Reikia rasti tokį daiktų  
rinkinį, kuris tilptų į V tūrio kuprinę ir krovinio vertė būtų didžiausia.  
4. Dėžių užpildymo uždavinys. Turime keletą vienetinio tūrio dėžių ir n  
daiktų, kurių dydžiai v1, v2,..., vn, čia 0 < vj < 1. Šiuos daiktus  
reikia sudėti į kuo mažesnį skaičių dėžių.*