Pirma klausimų grupė (4 balai):

## **1. Dinaminis programavimas (15 sk. 358 psl.). Algoritmų sudarymo metodika (15.3 sk. 379-389 psl.) 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** – rekursinių algoritmų taikymo metodika išsaugant dalinius rezultatus. Taip išvengiama pakartotinio to paties uždavinio sprendimo. Jei tas pats mažesnis uždavinys sprendžiamas daug kartų, mažesnio uždavinio sprendimas pakeičiamas jau išspręsto uždavinio rezultato paieška lentelėje. Taikant dinaminį programavimą paprastai reikia daugiau atminties rezultatų saugojimui, tačiau galutinis rezultatas gaunamas žymiai greičiau nei taikant įprastą rekursiją.

Taikant dinaminį programavimą uždavinys paprastai sumažinamas pašalinant vieną elementą, išsprendžiamas mažesnis uždavinys ir jo sprendinys panaudojamas didesnio uždavinio sprendiniui gauti. Dinaminis programavimas vs Skaldyk ir valdyk Strategijoje „Skaldyk ir valdyk“ uždavinys padalinamas į keletą (dažniausiai dvi) beveik lygių nepriklausomų dalių, kurios sprendžiamos atskirai, o jų apjungti sprendiniai suformuoja pradinio uždavinio sprendinį.

**Algoritmų sudarymo metodika**  
*1. Nusakyti optimalią uždavinio sprendinio struktūrą. (optimali struktūra vadinama tada, kai jos optimalus sprendinys apima pagalbinių uždavinių optimalius sprendinius)*

*2. Rekursiškai apibrėžti optimalų uždavinio sprendinį*

*3. Apskaičiuoti optimalaus sprendinio reikšmę*

*4. Rasti galutinį sprendinį*

**Bendro ilgiausio posekio radima uždavinys**

*1 etapas. Optimali uždavinio sprendinio struktūra*

Duomenys seka 𝑿 = (𝑥1, 𝑥2, … , 𝑥𝑚) (ilgis 𝑚)

seka 𝒀 = (𝑦1, 𝑦2, … , 𝑦𝑛) (ilgis 𝑛)

Tarkime, seka 𝒁 = (𝒛𝟏, 𝒛𝟐, …, 𝒛𝒌) yra bet kuris ilgiausias sekų 𝑋 ir 𝑌 posekis. Tada galioja:

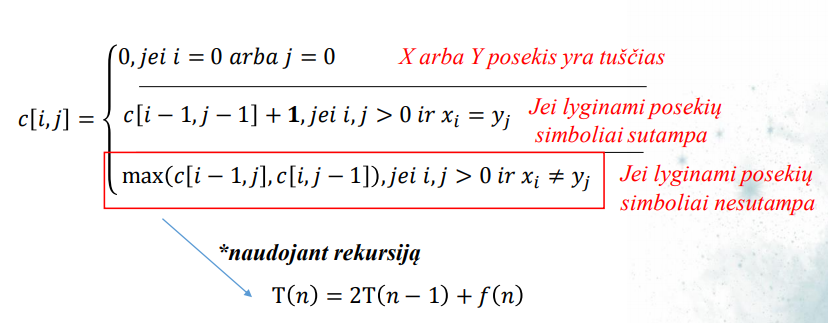
1. Jei 𝑥𝑚 = 𝑦𝑛, tai 𝑧𝑘 = 𝑥𝑚 = 𝑦𝑛 ir 𝑍𝑘−1 yra ilgiausias bendras sekų 𝑋𝑚−1 ir 𝑌𝑛−1 posekis.

2. Jei 𝑥𝑚 ≠ 𝑦𝑛, tada kai 𝑧𝑘 ≠ 𝑥𝑚, tai 𝑍 yra ilgiausias bendras sekų 𝑋𝑚−1 ir 𝑌 posekis.

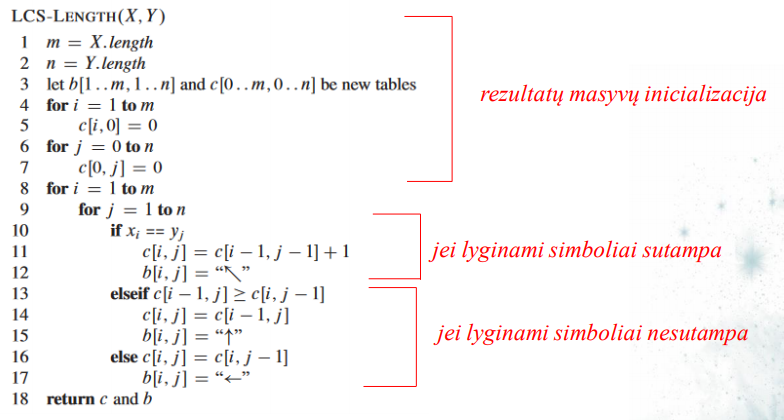
3. Jei 𝑥𝑚 ≠ 𝑦𝑛, tada kai 𝑧𝑘 ≠ 𝑦𝑛, tai 𝑍 yra ilgiausias bendras sekų 𝑋 ir 𝑌𝑛−1 posekis.

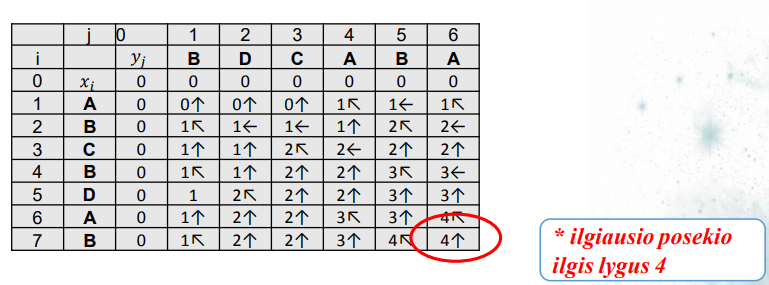
*2 etapas. Rekursiškai apibrėžiamas optimalaus uždavinio sprendinys*

Optimalus sprendinys užrašomas rekurentiniu ryšiu



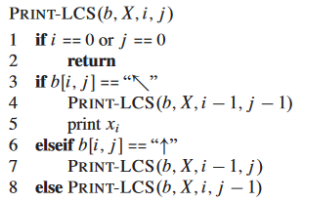
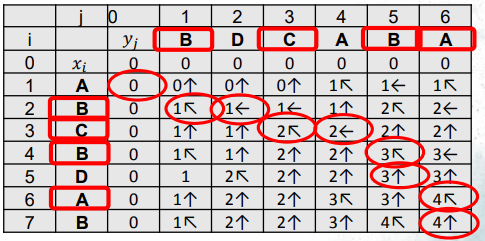
*3 etapas. Apskaičiuojama optimalaus sprendinio reikšmė*

****

****

Sudėtingumas: **(nm)**

*4 etapas. Rasti galutinį sprendinį*

**** ****

ilgiausias posekis: „BCBA“

## **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…).**

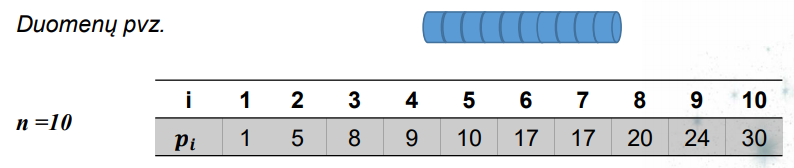
*1. Nusakyti optimalią uždavinio sprendinio struktūrą. (optimali struktūra vadinama tada, kai jos optimalus sprendinys apima pagalbinių uždavinių optimalius sprendinius)*

Duomenys : Turima n ilgio strypas ir kainų lentelė, kurioje nurodoma kiek kainuoja i ilgio strypas.

• pjaustymas nekainuoja

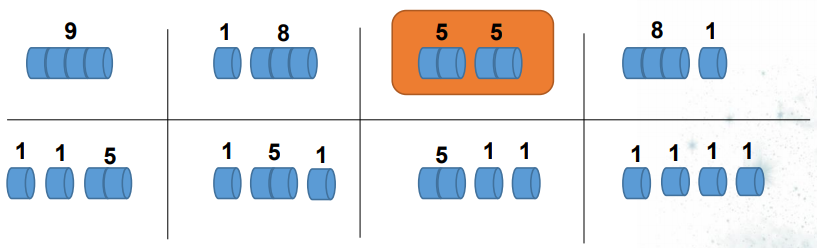
• prapjovos plotis - 0

Siekiamas rezultatas: Padalinti (supjaustyti) strypą taip, kad jo kainą būtų didžiausia (gautos didžiausios pajamos / pelnas).

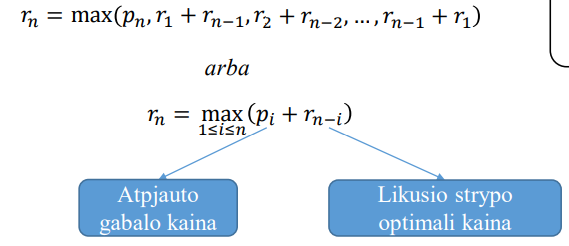


n ilgio strypą galima supjaustyti 𝟐 𝒏−𝟏 būdais

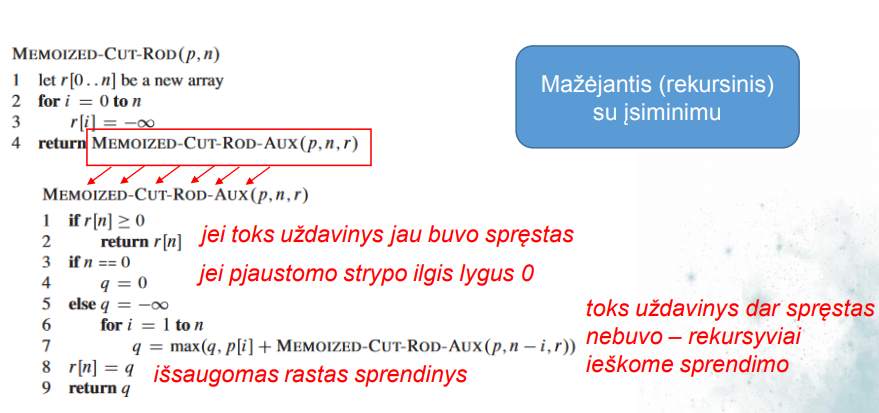
Kai n = 4

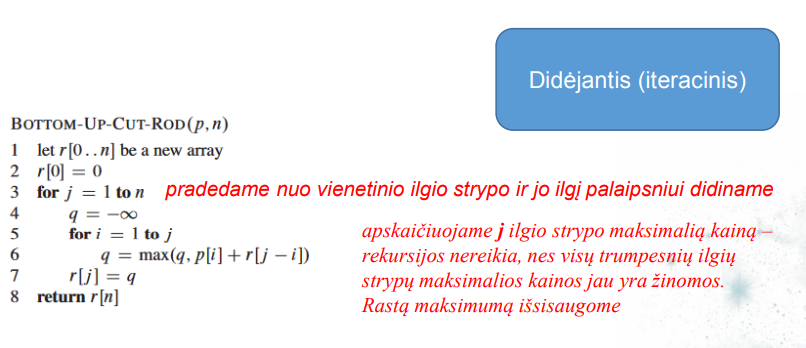


*2. Rekursiškai apibrėžti optimalų uždavinio sprendinį*

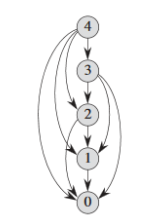


*3. Apskaičiuoti optimalaus sprendinio reikšmę*



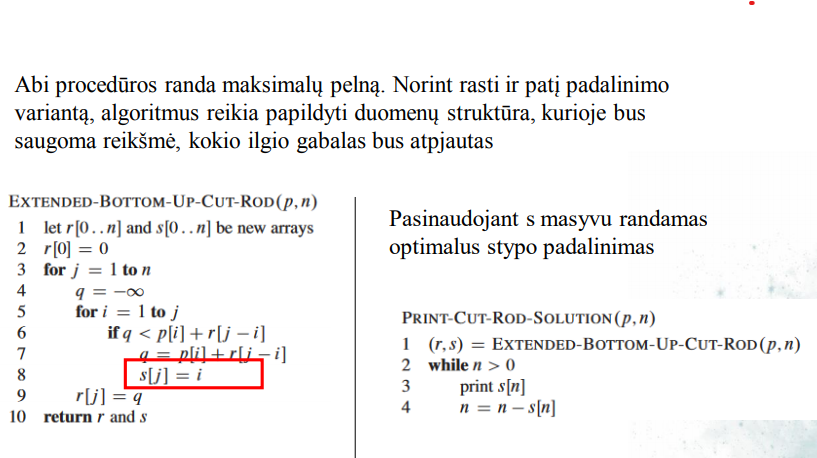


Abiem atvejais sudėtingumas **(n)**

****

*4. Rasti galutinį sprendinį*

Abi procedūros randa maksimalų pelną. Norint rasti ir patį padalinimo variantą, algoritmus reikia papildyti duomenų struktūra, kurioje bus saugoma reikšmė, kokio ilgio gabalas bus atpjautas



## **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…).**

Turim seką A1, A2, … An n matricų, kurias norėsime sudauginti ir norime įvertinti produktą A1A2 …An

Pvz.: jei turim matricų grandinę A1,A2,A3,A4, tai produktui A1A2A3A4 skliaustai gali būti sudėti keliais būdais, pora paminėsiu: (((A1 A2) A3) A4). ((A1 (A2 A3)) A4) ir t.t. Mūsų tikslas yra sudėti skliaustus taip, kad veiksmų skaičius būtų minimalus.

MATRIX-MULTIPLY(*A*, *B*)

1 **if** *columns*[*A*] ≠ *rows*[*B*]

2 **then error** "incompatible dimensions"

3 **else for** *i* ← 1 **to** *rows*[*A*]

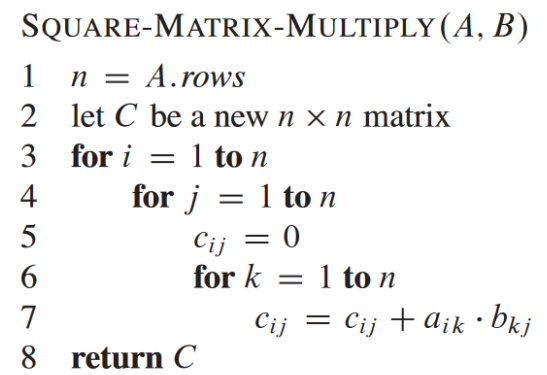
4 **do for** *j* ← 1 **to** *columns*[*B*]

5 **do** *C*[*i*, *j*] ← 0

6 **for** *k* ← 1 **to** *columns*[*A*]

7 **do** *C*[*i*, *j*] ← *C*[*i*, *j*] + *A*[*i*, *k*] · *B*[*k*, *j*]

8 **return** *C*



Galima sudauginti matricas A ir B tik jei jos yra suderinamos: stulpelius skaičius A matricoje turi būti lygus B eilučių skaičiuj. Jei A yra p x q matrica ir B yra q x r matrica, tada rezultate gausim C matricą p x r.

Tarkim turim trijų matricų grandinę A1,A2,A3. Atitinkamai jų dydžiai yra 10 × 100, 100 × 5, ir 5 × 50. Jei dauginsim pagal pimą skliaustų sudėjimo variantą ((A1 A2) A3), tai atliksim 10 · 100 · 5 = 5000 skaliarinių daugybų apskaičiuoti 10 × 5 matricos produktą A1 A2, plius kitus 10 · 5 · 50 = 2500 skaliarinės daugybos, kad sudauginti šią matricą iš A3, bendroj sumoj 7500 skaliarinių daugybų. Jei dauginsim pagal (A1 (A2 A3)), tai atliksim 100 · 5 · 50 = 25,000 skaliarinių daugybų suskaičiuoti 100 × 50 matricos produktą A2 A3, plius kitas 10 · 100 · 50 = 50,000 skaliarinių daugybų iš A1, sumoj gaunam 75,000 skaliarinių daugybų. Taigi, skaičiuojant pirmu būdu gaunam rezultatą 10 kartų greičiau.

Tačiau tikrinimas visų galimų skliaustų sudėjimo variantų neduos efektyvaus algoritmo. Pažymim skaičių alternativių variantų skliaustų sudėjimo n sekų kaip P(n). Tada gaunam rekurentinę formulę:



panašios formulės augimas yra Ω(4*n*/*n*3/2)

Rekursyvinio apibrėžimo minimaliam laikui skliaustų sudėjimo produktui Ai Ai+1 Aj yra:



The m[i, j] values give the costs of optimal solutions to subproblems

Sudėtingumas: **Θ(𝑛3 )**

## **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.).**

Godūs algoritmai – tai greitesni ir efektyvesni algoritmai lyginant su dinaminio programavimo uždaviniais. Jų esmė: kiekvienu momentu daromas sprendimas, kuris tuo momentu yra geriausias.

Godūs algoritmai dažniausiai naudojami NP uždavinių sprendimui (uždavinio sprendinio „gerumas“ apibrėžiamas skaitine reikšme. Skaitinė reikšmė, atitinkanti minimumą arba maksimumą tam tikrai uždavinio duomenų imčiai – nurodo optimalų uždavinio sprendiny. Pavyzdys : keliaujančio pirklio uždavinys)

**Procesų pasirinkimo uždavinys**

*1. Randama uždavinio optimali struktūra*

Sąlyga: Reikia paskirstyti resurso prieigą keliems „konkuruojantiems“ procesams (vienu metu resursu gali naudotis tik vienas procesas). Kuriems procesams priskirti resurso prieigą, kad atliktų procesų (užduočių) skaičius būtų maksimalus

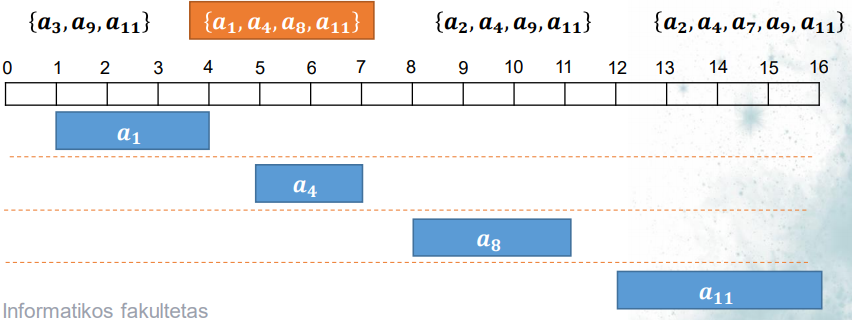
Duomenys

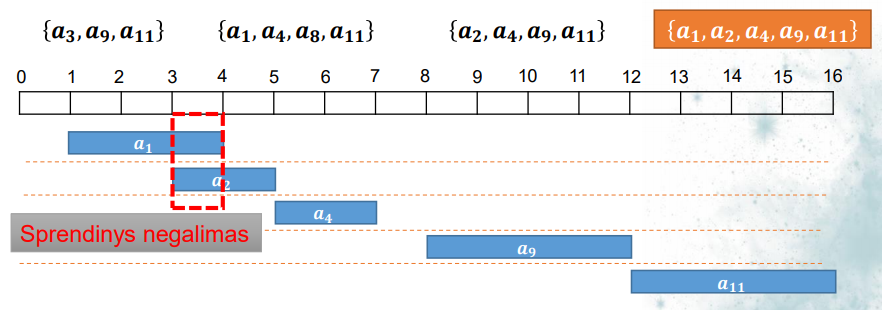
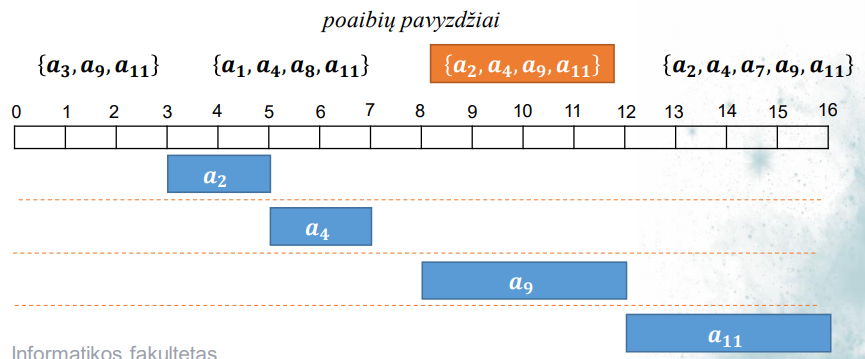
𝑺 = 𝒂𝟏,𝟐,… ,𝒂𝒏 – procesų (užduočių) aibė

𝒔𝒊 ir 𝒇𝒊 – 𝒊-tojo proceso pradžios ir pabaigos laikas, 0 ≤ 𝒔𝒊 < 𝒇𝒊 < ∞

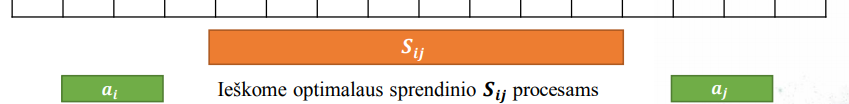
Procesai nesikerta jei intervalai [𝒊 , 𝒇𝒊 ) ir (𝒔𝒋 , 𝒇𝒋 ]nepersidengia.

Procesai pateikti surikiuoti didėjimo tvarka pagal pabaigos laiką: 𝒇𝟏 ≤ 𝒇𝟐 ≤ ⋯ ≤ 𝒇n



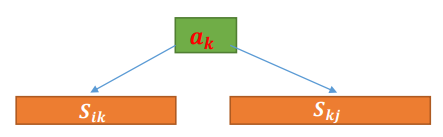
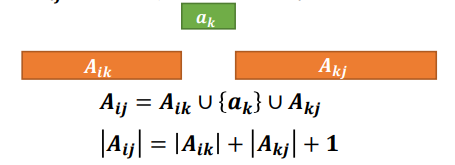


*2. Sudaromas rekursinis sprendimas*

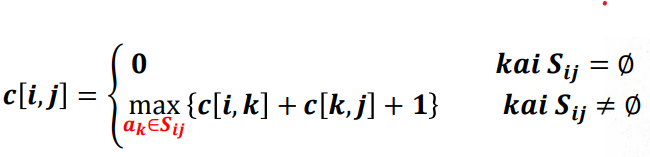


Tarkime 𝑺𝒊𝒋 - procesų poaibis, kuris prasideda baigusis procesui 𝒂𝒊 ir baigiasi dar neprasidėjus procesui 𝒂𝒋

Ieškome optimalaus sprendinio 𝑺𝒊𝒋 procesams tai tarkime optimalus sprendinys poaibis 𝑨𝒊𝒋, kuriame yra užduotis 𝒂𝒌

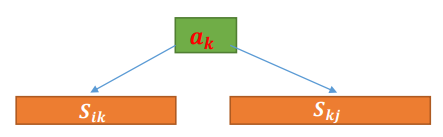


Iš optimalios sprendinio struktūros seka: 𝒄 [𝒊,] = 𝒄 [𝒊,𝒌] + 𝒄 [𝒌,𝒋] + 1

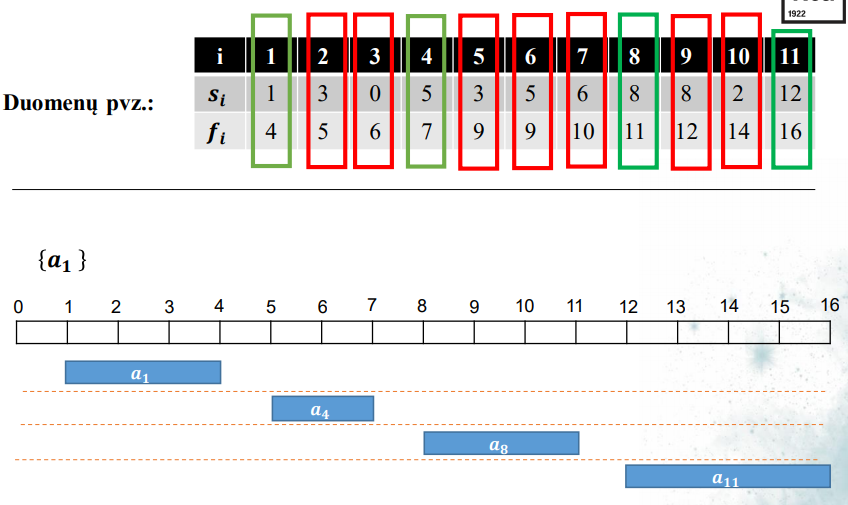
Tačiau 𝒌 nežinome ir reikia patikrinti visus galimus variantus: 

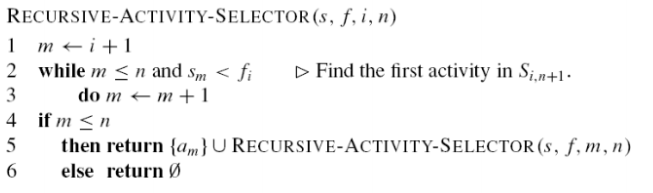
*3. Parodoma, kad esant godžiai strategijai lieka tik vienas pagalbinis uždavinys*

godūs algoritmai priima sprendimą, kuris tuo metu yra / atrodo „geriausias“



*4. Parodoma (įrodoma), kad godus pasirinkimas yra saugus*



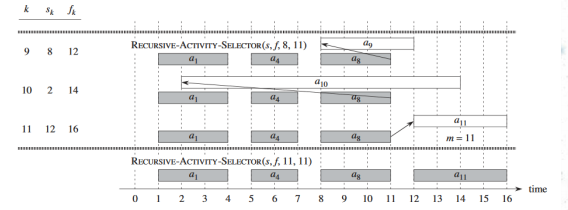
*5. Sudaromas rekursinis algoritmas, realizuojantis godžią strategiją*

𝑠 – užduočių pradžios laikai

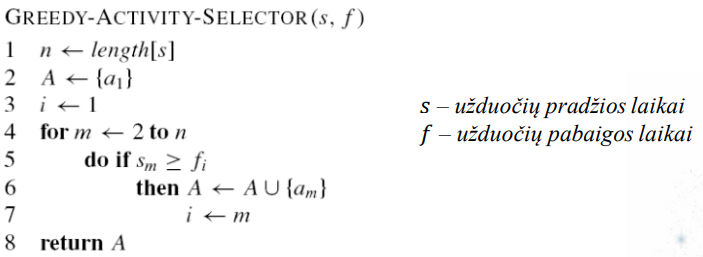
𝑓 – užduočių pabaigos laikai

𝑛 – viso užduočių

𝑖 – paskutinė įtraukta užduotis



*6. Rekursinis algoritmas transformuojamas į iteracinį*

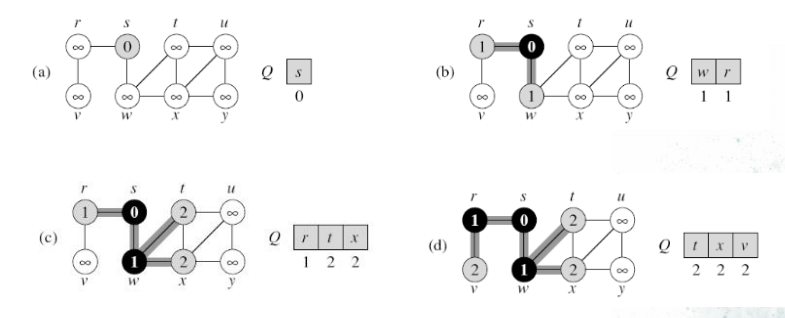


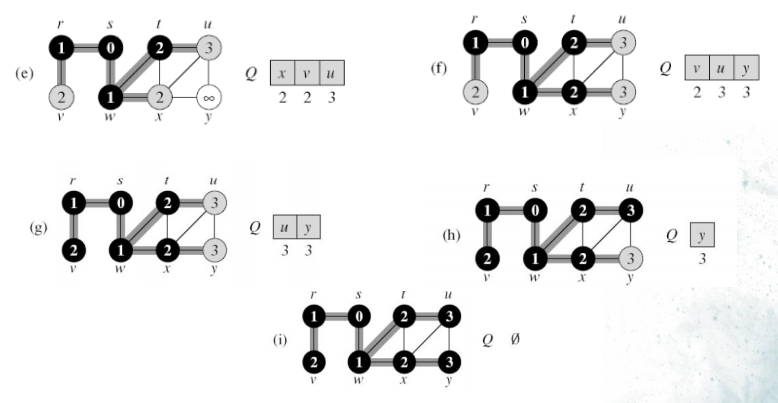
## Antra klausimų grupė (2 balai): **1. Paieškos į plotį algoritmas (22.2 sk. 594-597 psl.) ir sudėtingumo įvertinimas.**

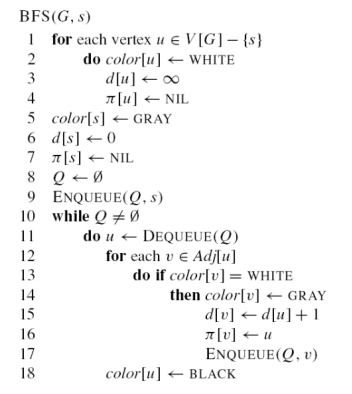
Kiekvienai viršūnei u priskirsime spalvos atributą color[u], kuris gali įgyti šias reikšmes:

* Balta – nenagrinėta
* Pilka – pažymėta kaip kaimyninė viršūnė nagrinėtos viršūnės, bet kaimynystėje gali turėti baltų viršūnių
* Juoda – išnagrinėta viršūnė

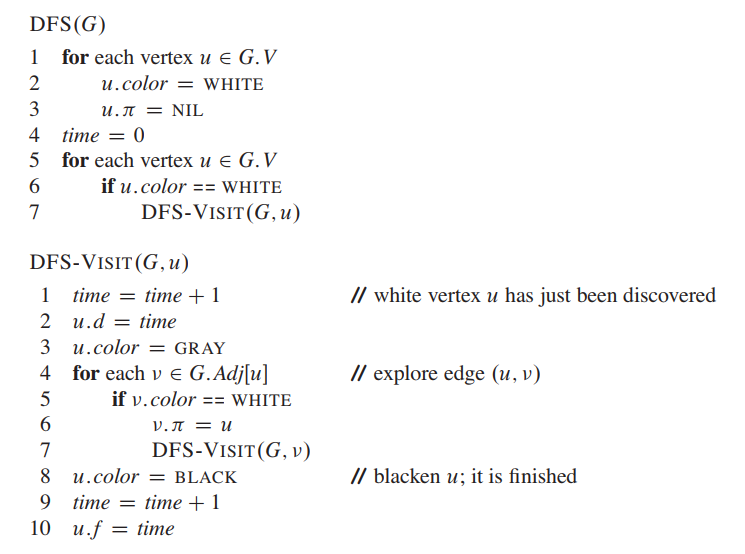
Šis algoritmas sudaro paieškos į plotį medį, kuris pradiniu laiko momentu turi tik šakninę viršūnę s, be to, kiekvienai viršūnei u priskiriamas atributas π[u], nurodantis kokia viršūnė eina prieš ją, ir atributas d[u], nurodantis kokiu atstumu yra nutolusi viršūnė nuo pradinės s

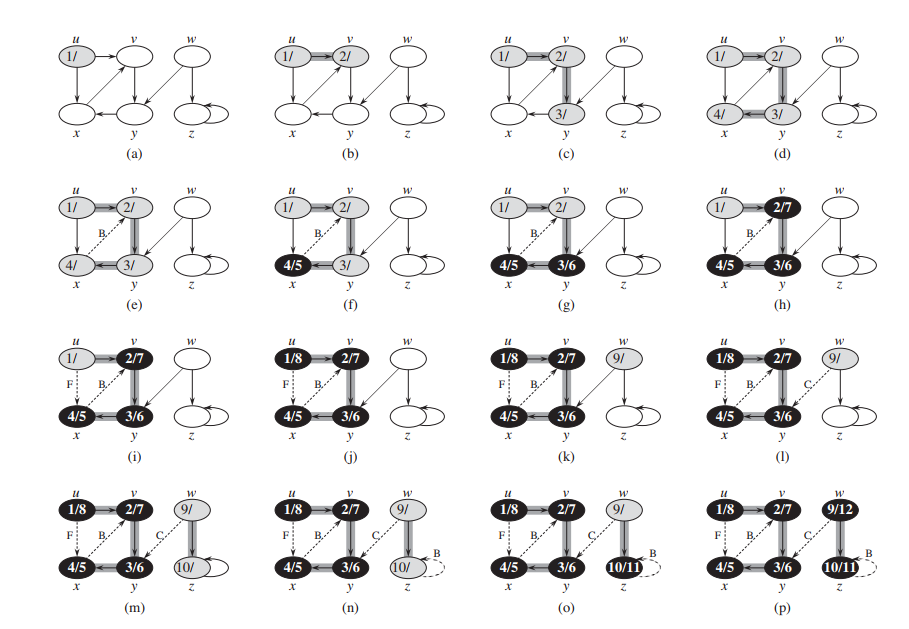




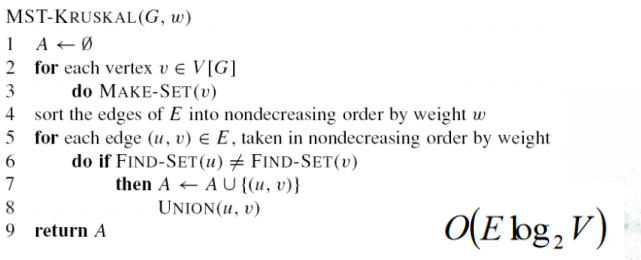
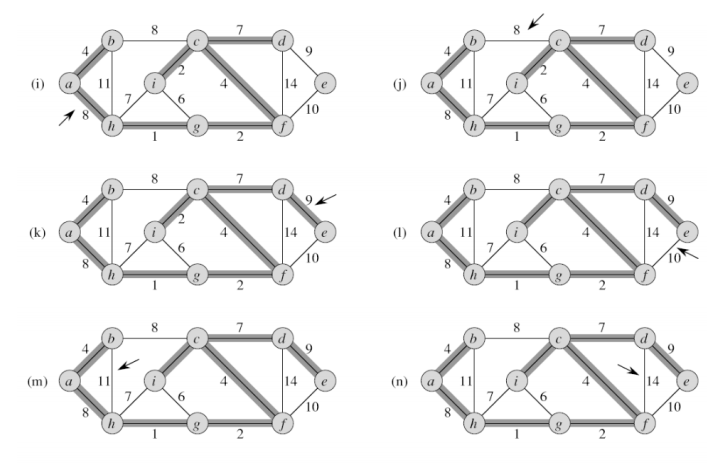
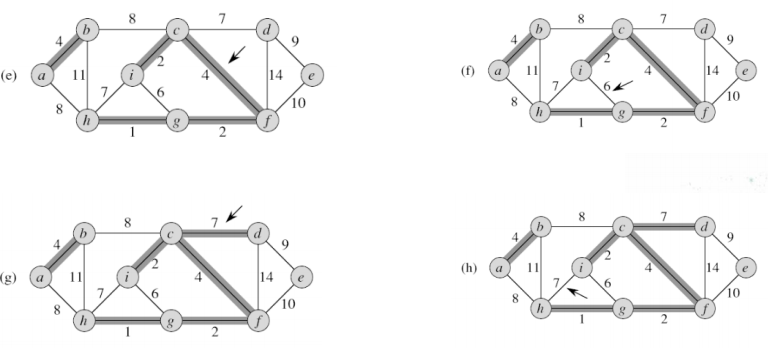
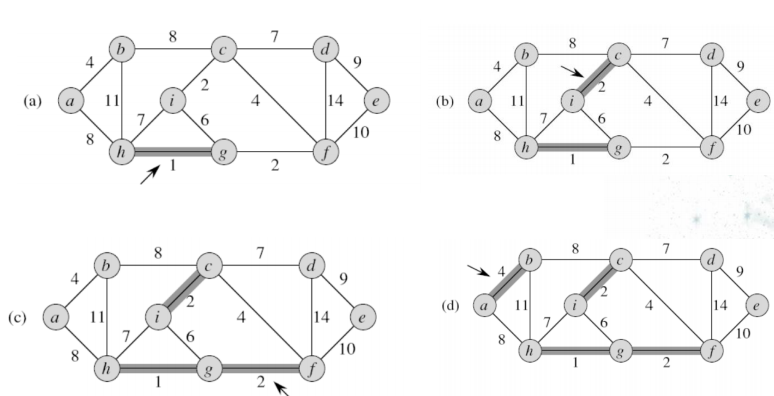
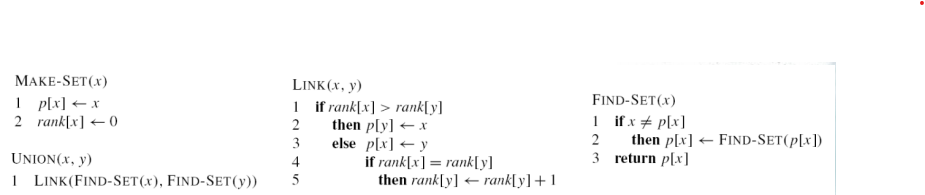
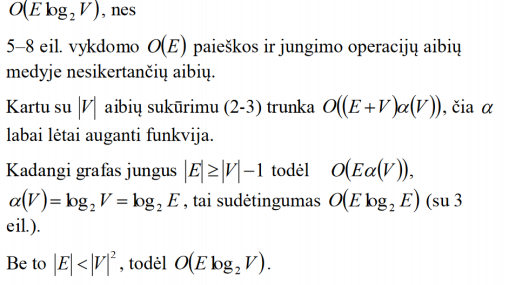


## **2. Paieškos į gylį algoritmas (22.3 sk. 603-606 psl.) ir sudėtingumo įvertinimas.**

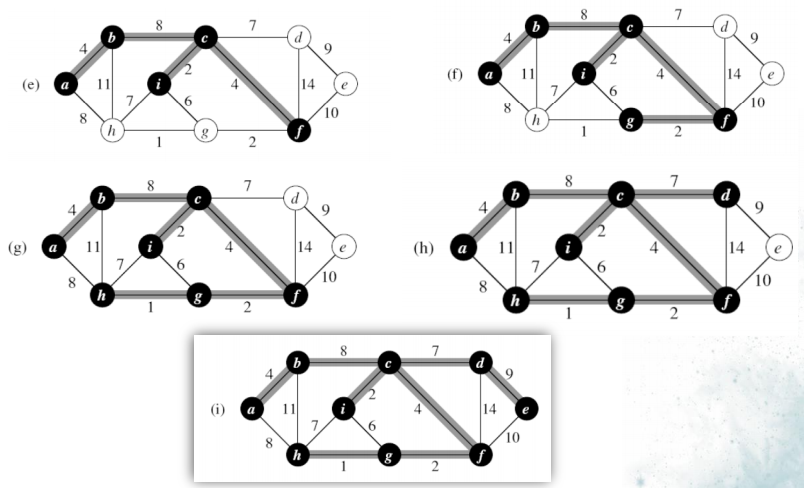


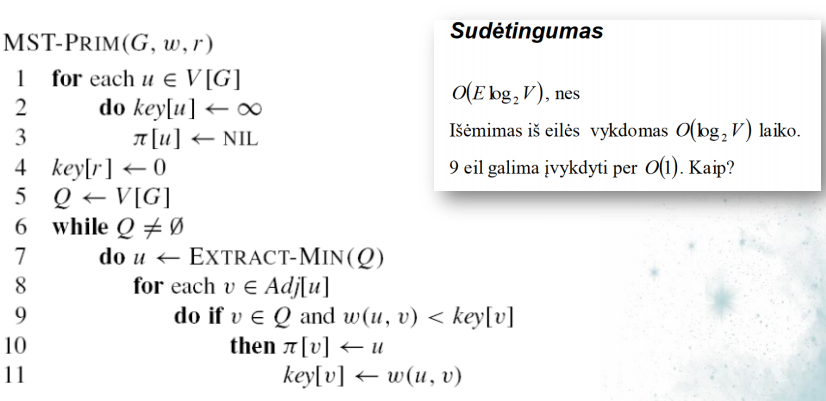


## **3. Kruskalo algoritmas (23.2 sk. 631-633 psl.) ir sudėtingumo įvertinimas.**

****

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

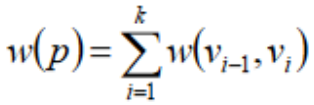




## **5. Trumpiausi keliai iš vienos viršūnės (24 sk. 641-650 psl.). Relaksacijos metodas. Belmano –Fordo** **algoritmas (24.1 sk. 651 psl.). Sudėtingumo įvertinimas.**

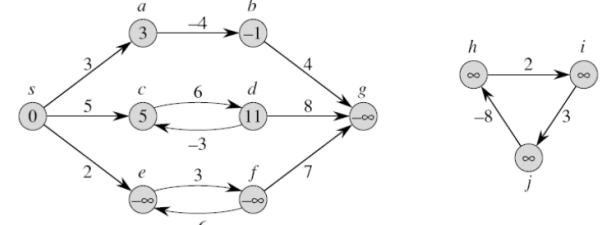
**Trumpiausi keliai iš vienos viršūnės**

Duotas grafas G = (V, E) ir svorių funkcija w : E -> R

 V – viršūnių aibė, E – lankų aibė

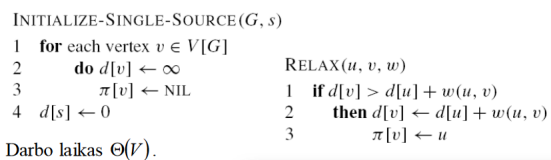
Kelias p = { v0, v1, .. , vk} , o jo svoris

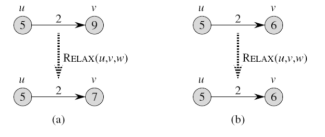
Jei kelyje yra neigiamo svorio ciklas, tai bendras kelio svoris tarp dviejų viršūnių – minus begalybė

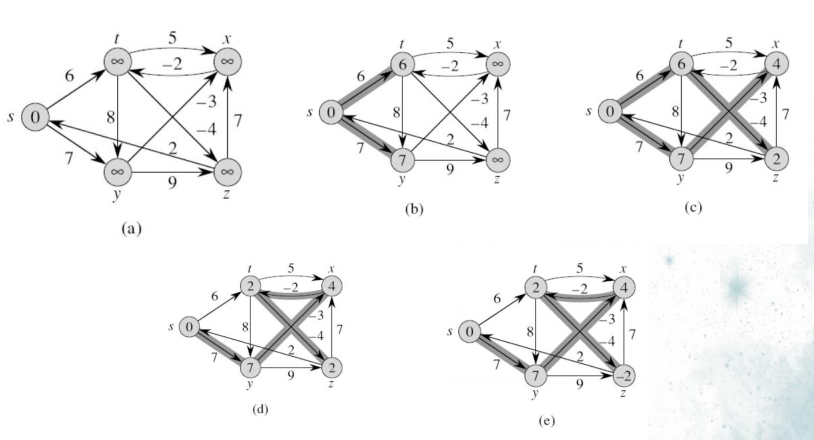


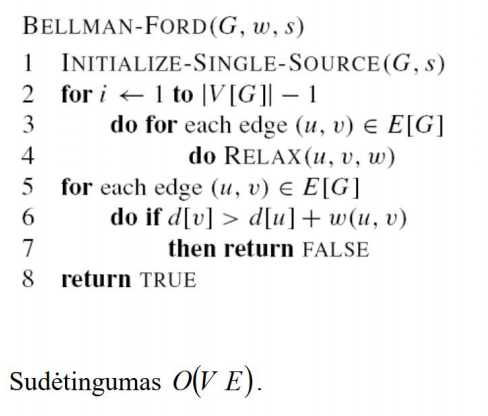
**Relaksacijos metodas**

Kiekvienai grafo G = (V, E) viršūnei v priskirsime atributą d[v], kuris nusakys viršutinę svorio ribą trumpiausio kelio is pradinės viršūnės s iki v



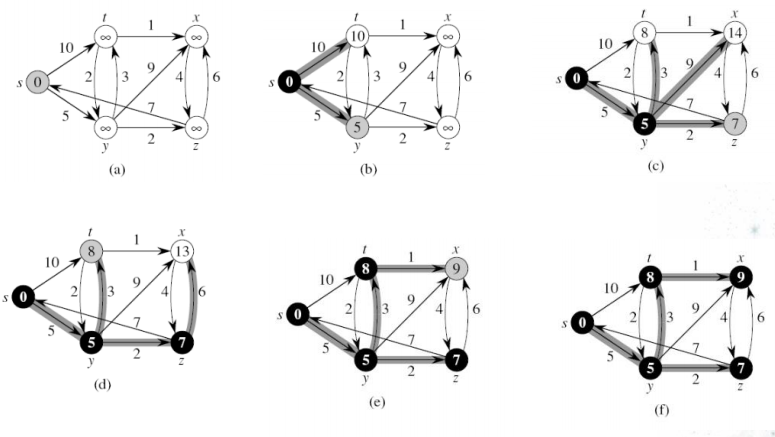


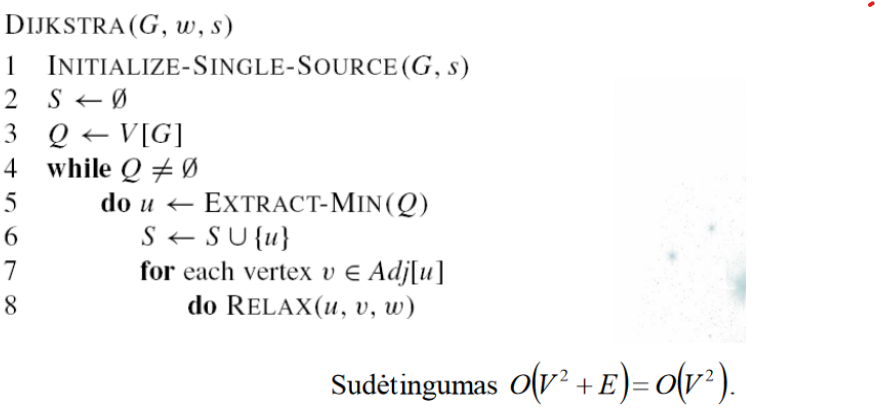
**Belmano –Fordo algoritmas**



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

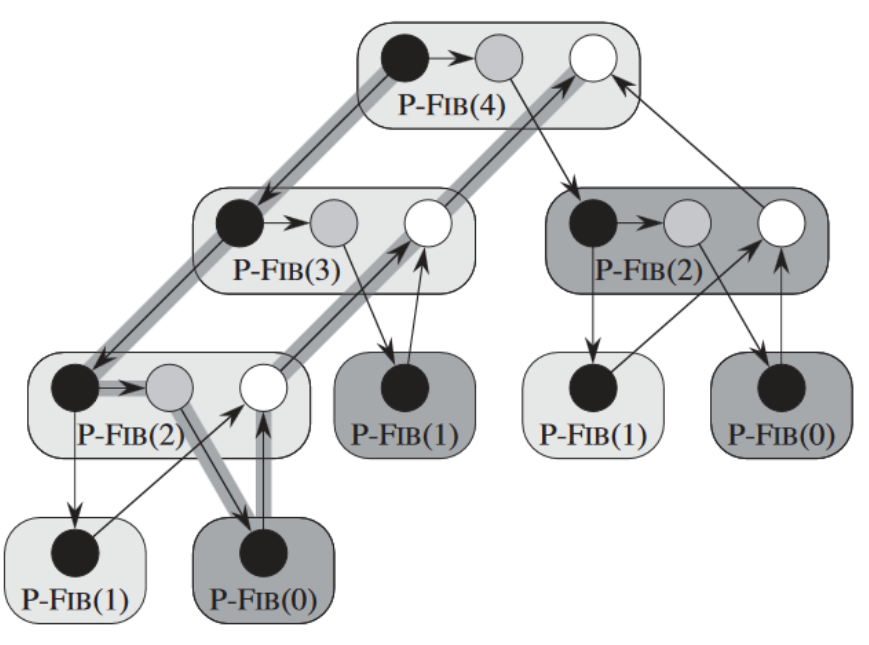
**Deikstra algoritmas**



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

Darbas (work) – kiek yra vienetinių užduočių, arba „mazgų skaičius acikliniame grafe“

Intervalas (span) – kiek užtrunka skaičiavimai, arba „mazgų skaičius acikliniame grafe“



Darbas: 17

Intervalas: 8

𝑃 – procesorių skaičius

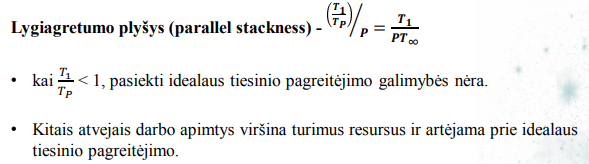
𝑇𝑝 – laikas, reikalingas atlikti užduotį su 𝑃 procesorių

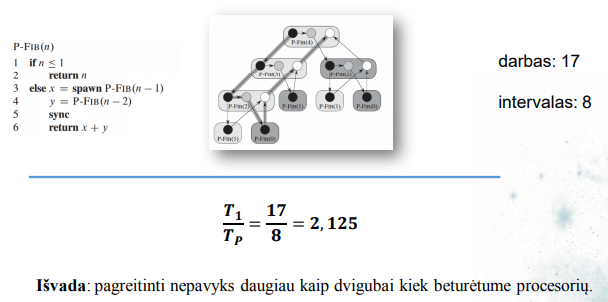
Idealus kompiuteris su 𝑷 procesorių per vieną žingsnį gali atlikti 𝑷 darbo vienetų (mazgų, jei žiūrėsime į vykdymo grafiką).

Per laiko tarpą 𝑻𝒑, gali būti atlikta 𝑷𝑻𝒑 užduočių. Kai reikalingas atlikti darbo laikas 𝑻𝟏, tada 𝑷𝑻𝒑 ≥ 𝑻𝟏. Nelygybę dalinant iš 𝑷 gaunama darbo apibrėžimo taisyklė

Idealus kompiuteris su 𝑷 procesorių negali greičiau dirbti (įvykdyti užduotį) nei kompiuteris su neribotu procesorių kiekiu, t.y.

Kai procesų skaičius viršina lygiagretumą, skaičiavimai negali pasiekti tiesinį pagreitėjimą.



spawn – naujos gijos proceso paleidimas sync – gijų / procesų sinchronizavimas

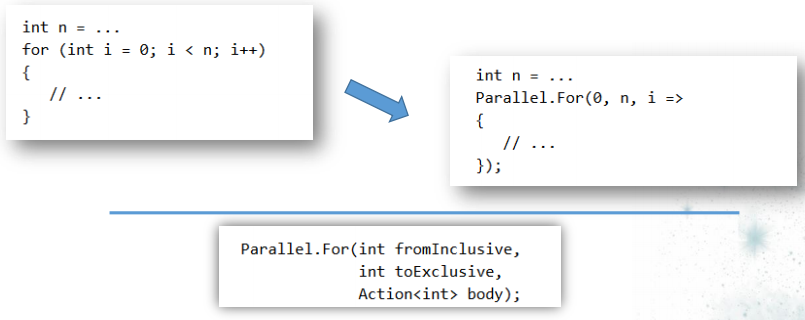
## **Trečia klausimų grupė (4 balai):**

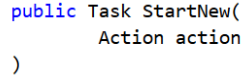
## **8. Daugiagijo dinaminio programavimo metodika. (27 sk. 772-791 psl.)**

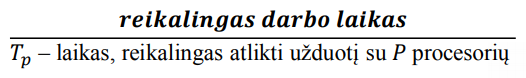
Statinėmis gijomis – visos programos gyvavimo metu išlieka gijos, t.y. mažai kinta gijų skaičius

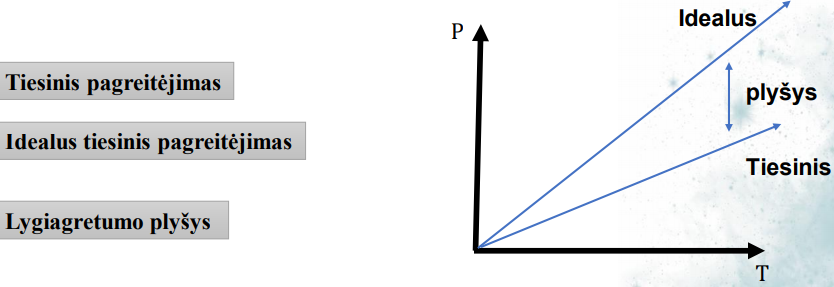
Dinaminėmis gijomis – leidžia programuotojui nurodyti lygiagretinimo lygį labai nekeičiant programos kodo ir nesirūpinant apkrovos balansavimu, komunikavimu

*parallel* – ciklo iteracijų vykdymas vienu metu

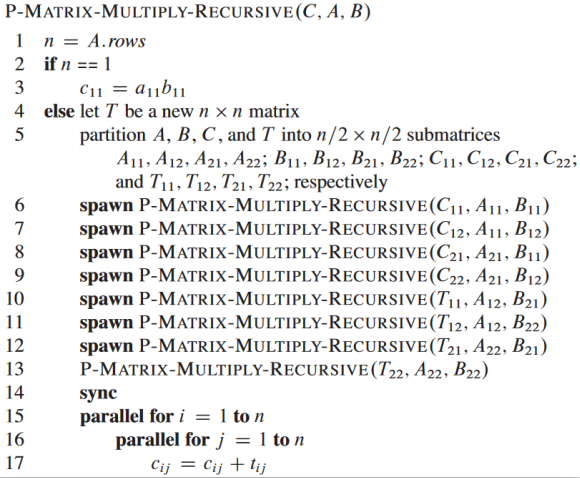
****

****

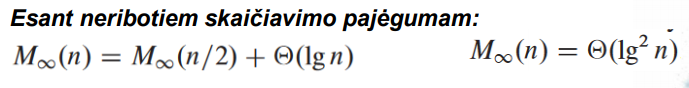
Pagreitinimas (speedup) randamas skaičiuojant santykį****

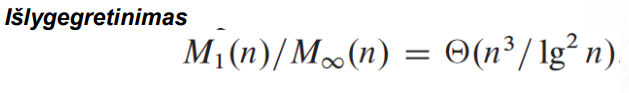
****

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



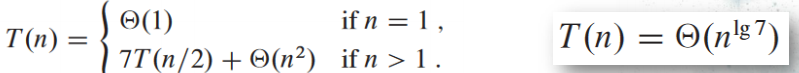






**Kvadratinių matricų daugyba – Strassen algoritmas**

* vietoje 8 rekursinių kreipinių perduodant– atlieka 7 kreipinius (matricos išlieka n/2 × 𝑛/2 dydžio)
* vienos daugybos panaikinimo kaina, kelios naujos n/2 × 𝑛/2 matricų sudėtys



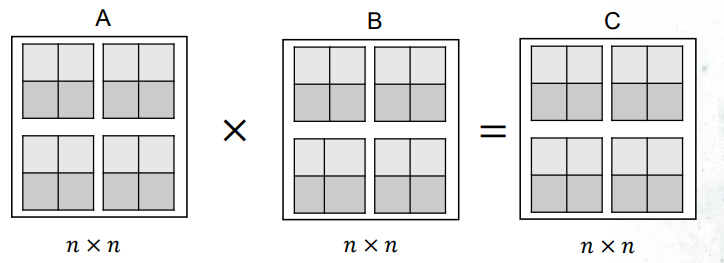
*1. Pradinių A, B ir rezultatų matricų C išskaldymas į n/2 × 𝑛/2 dydžio matricas. 𝜣(𝟏).*

*2. Sukuriame 10 matricų 𝑆1, 𝑆2, … , 𝑆10, kurių kiekviena yra n/2 × 𝑛/2 dydžio. (𝒏2 ). (𝐥𝐧 𝒏) - panaudojant parralel for.*

*3. Panaudojant 2 žingsnyje sukurtas 1 ir 10 matricas, rekursyviai sudaromos 7 tarpinės matricos 𝑃1, 𝑃2, … , 𝑃7. Kurių kiekviena yra n/2 × 𝑛/2 dydžio.*

*4. Apskaičiuojamos 𝐶11, 𝐶12, 𝐶21, 𝐶22 sudedant skirtingas matricų 𝑃𝑖 kombinacijas. (𝒏2 ). (𝐥𝐧 𝒏) - panaudojant parralel for.*

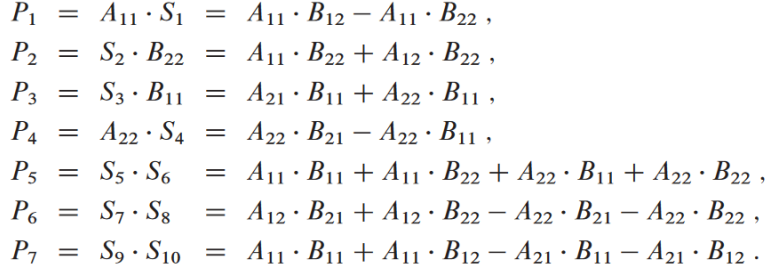
1. Pradinių A, B ir rezultatų matricų C išskaldymas į n/2 × 𝑛/2 dydžio matricas. 𝜣(𝟏).



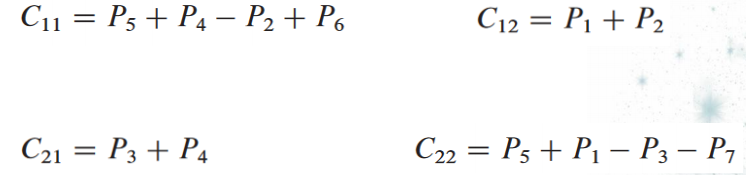
2. Sukuriame 10 matricų 𝑆1, 𝑆2, … , 𝑆10, kurių kiekviena yra n/2 × 𝑛/2 dydžio. 𝜣(𝒏𝟐 ).



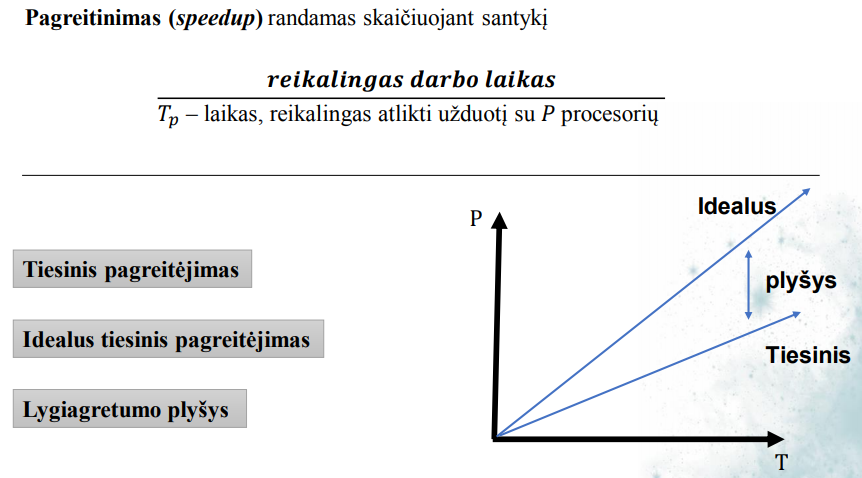
3. Panaudojant 2 žingsnyje sukurtas 1 ir 10 matricas, rekursyviai sudaromos 7 tarpinės matricos 𝑃1, 𝑃2, … , 𝑃7. Kurių kiekviena yra n/2 × 𝑛/2 dydžio.

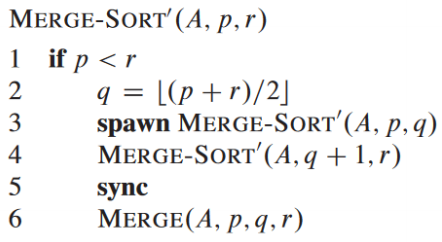


4. Apskaičiuojamos 𝐶11, 𝐶12, 𝐶21, 𝐶22 sudedant skirtingas matricų 𝑃𝑖 kombinacijas. 𝜣(𝒏𝟐 ).



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

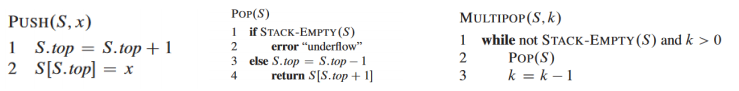
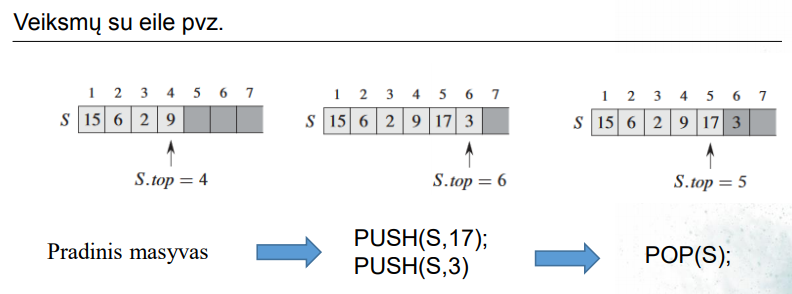
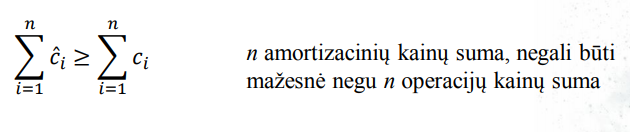


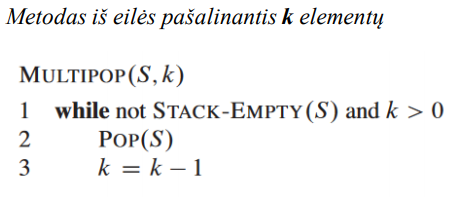


## 

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

Siekiama įvertinti ne atskirų funkcijų sudėtingumą, bet gauti bendrą programos / algoritmo sudėtingumo įvertinimą.





* ***Agregatinė analizė***

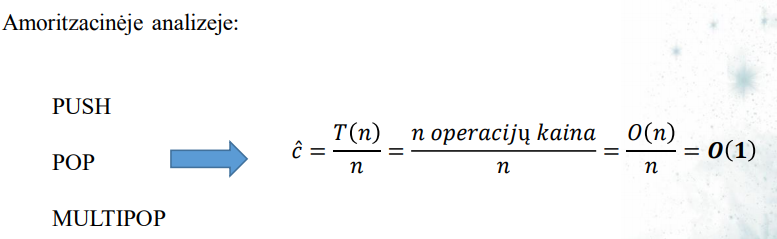
Turime tuščią eilę. Į ją įterpiama / pašalinama n elementų.

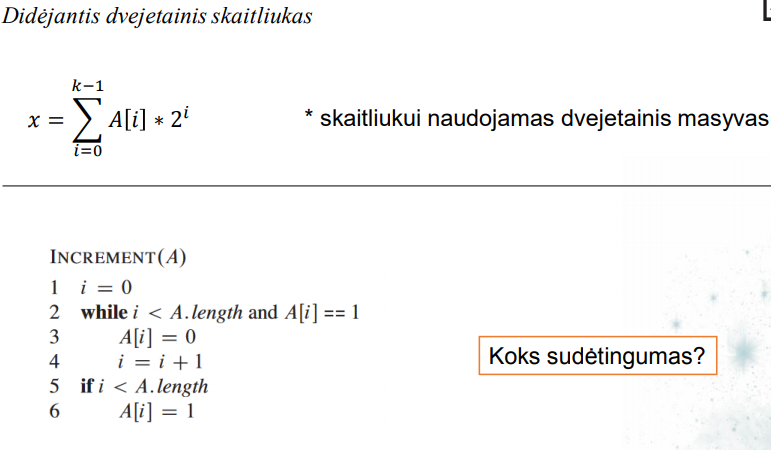
Blogiausiu atveju MULTIPOP procedūros sudėtingumas (𝑛), čia 𝑛 – eilės ilgis.

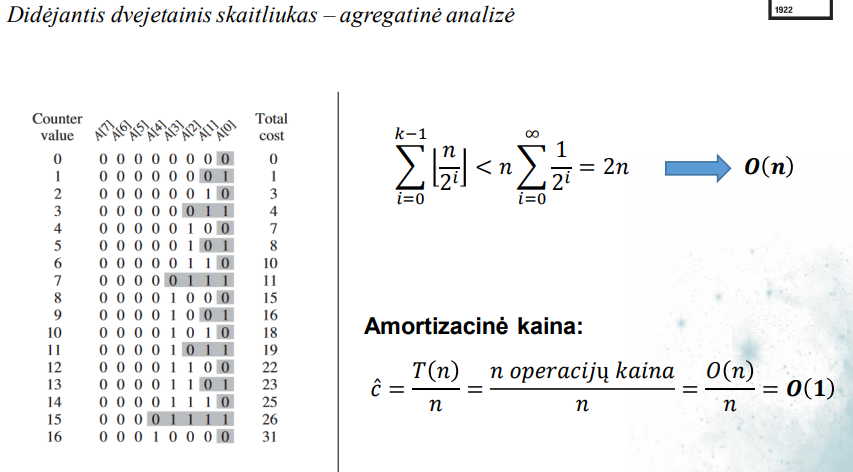
Dirbant su 𝑛 ilgio eile, MULTIPOP procedūrą gali būti iškviečiama 𝒏 kartų, todėl blogiausiu atveju įvertinimas gautųsi **O(n2)** .

Iš eilės elementas gali būti pašalintas tik vieną kartą, todėl ir POP funkcija nebus iškviečiama daugiau nei PUSH kartų.

T.y. reikšmei 𝒏 bet kokia PUSH, POP ir MULTIPOP operacijų kombinacijos sudėtingumas bus **O(n)** .





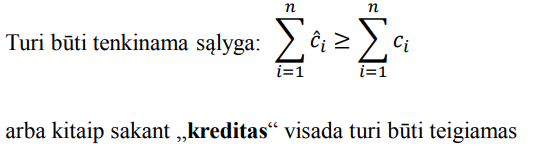


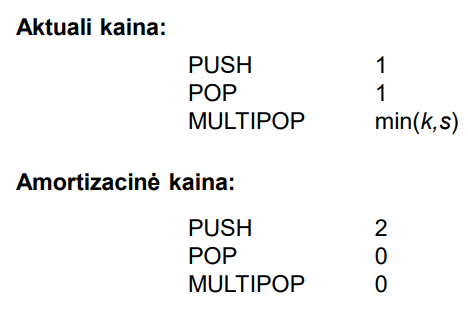
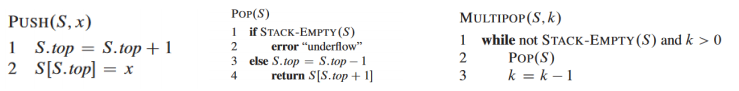
* ***Apskaitos metodas***

Priskiriami skirtingos kainos skirtingom operacijom - operacijų kainas gali būti parinktos didesnės arba mažesnės nei yra iš tikrųjų) – šias kainas vadiname amortizuojančiomis kainomis

Kai amortizuojanti kaina viršija aktualią kainą, skirtumą priskiriame „kreditui“ – kurį vėliau galėsime kompensuoti iš tų operacijų, kurių amortizuojanti kaina yra mažesnė nei aktuali.

Agregatinėje analizėje visų operacijų kaina vienoda, kai apskaitos metode gali skirtis:



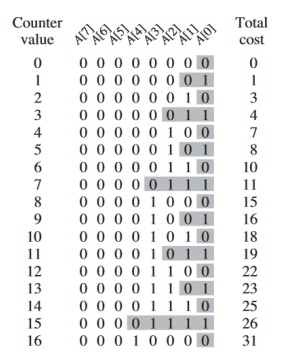
****

****** Bito priskyrimo vienetui kaina – 2 iš jų:

a) 1 – kai priskiriama vienam

b) 1 – kai grąžinamas nulis (kreditas)

Sudėtingumas: **O(n):**

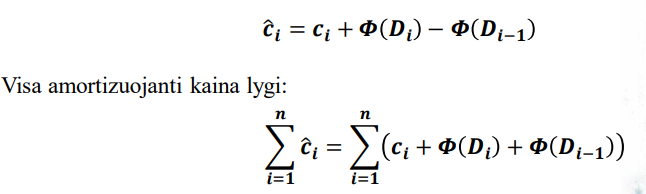
******

* ***Potencialo metodas***

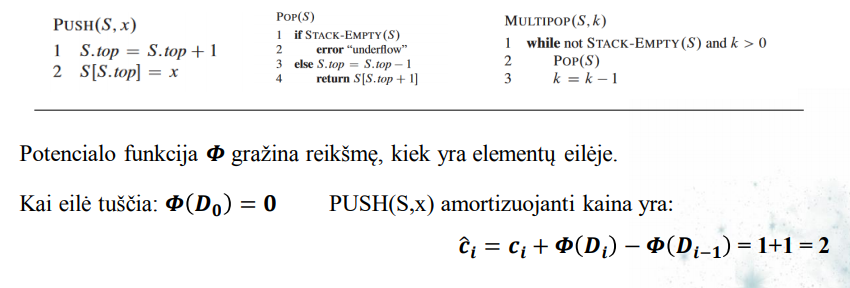
Vietoje „kreditų” naudojama „potencinės energijos“ sąvoka (arba tiesiog potencialas) susieta su duomenų struktūra.

Atliekam n iteracijų. Pradžioje, duomenų struktūra 𝐷0. Kiekvienos iteracijos kaina yra reali kaina 𝑖-tosios iteracijas 𝒄𝒊 ir 𝑫𝒊 duomenų struktūros pokytis lyginant su praėjusios iteracijos duomenų struktūra 𝑫𝒊−𝟏.

Potencialo funkcija (𝑫𝒊)gražina realią reikšmę, tam tikrai duomenų struktūrai, ir tam tikros iteracijos amortizacinė kaina išreiškiama:

******

Praktikoje, ne visuomet žinome, kiek iteracijų bus atliekama, todėl visiems 𝒊 pareikalaudami, kad (𝑫𝒊))≥ 𝜱 𝑫𝟎 - garantuojame, kad „sumokėsime į priekį“, ir reali operacijų kaina neviršys amortizuojančios kainos. Dažniausiai priimama, kad (𝑫0)= 0, o 𝜱(𝑫𝒊) ≥ 𝟎 visiems 𝒊.

****

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

Dauguma iki šiol nagrinėtų algoritmų buvo polinominio sudėtingumo uždaviniai. Blogiausiu atveju, priklausomai nuo duomenų kiekio n, jų sudėtingumas buvo 𝑶**(nk)** - čia 𝒌 tam tikra konstanta.

**P ir NP klasės**

* P uždavinių klasė – uždaviniai, kurie sprendžiami per polinominę laiko trukmę 𝑶**(nk)** . Čia 𝒌 – konstanta, 𝒏 – įvedamų duomenų kiekis.
* NP uždavinių klasė – uždaviniai, kurių sprendimą galima patikrinti per polinominį laiką. 𝑷 ⊆ 𝑵P

**Algoritmo konstravimo eiga**

1. Parodyti, kad tai „sprendimo priėmimo“ uždavinys

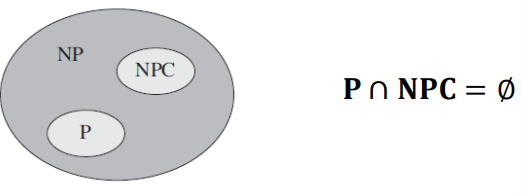
2. Parodyti, kad tai NPC klasės uždavinys

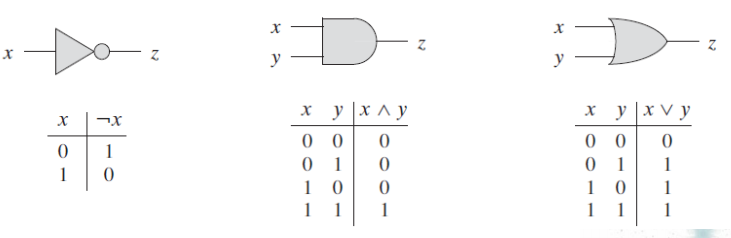
3. Pateikti polinominį sprendimo priėmimo algoritmą

4. Pateikti polinominį sprendimo optimizavimo algoritmą

**NP uždaviniai**

Uždavinys yra NP pilnasis (NPC), jei jis priklauso NP uždavinių klasei ir toks pats sudėtingas kaip ir bet kuris kitas NP uždavinys. Taigi, jei egzistuoja bent vienam NP uždaviniui polinominis sprenimo algoritmas, tai bet kuriam kitam šios klasės uždaviniui egzistuoja toks algoritmas.



* Loginių iteracijų grandinė (Circuit satisfiability) 
* Keliaujančio pirklio uždavinys - aplankyti visas viršūnes ir grįžti atgal

