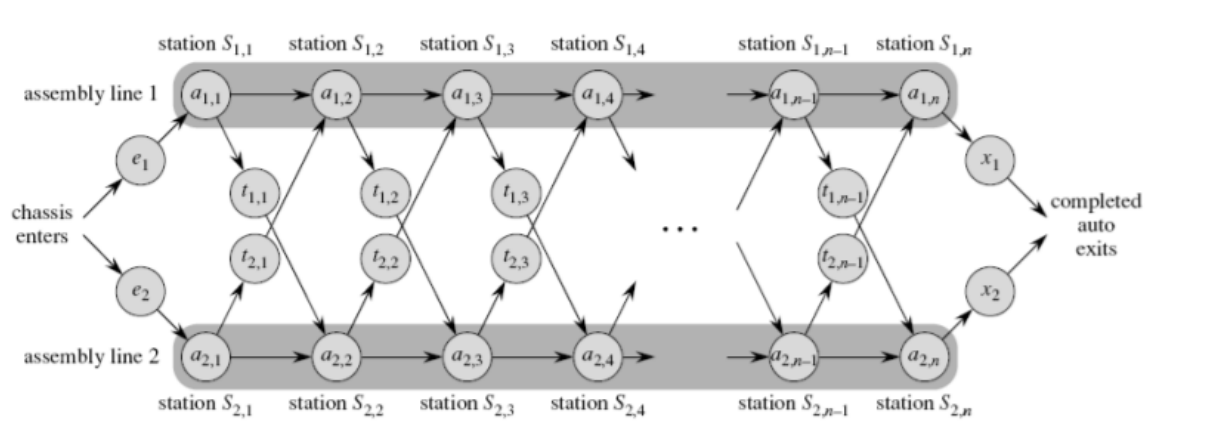
1. **Pirma grupė (4 balai)**
   1. **Dinaminis programavimas (15 sk. 358 psl.). Algoritmų sudarymo metodika (15.3 sk. 379-389 psl.) šios metodikos taikymas sprendžiant Konvejerio (Surinkimo linijos planavimo) (15.1 sk. Antras knygos leidimas) arba Bendro ilgiausio posekio radimo (15.4 sk. 390-395 psl.) uždavinį (rekursinės lygtys, optimalaus sprendinio reikšmės ir struktūros radimas, algoritmo sudėtingumo radimas…)**

**Dinaminis programavimas,** tai kaip turime rekursinį algoritmą, jį realizuojame be rekursijos, išskaidydami uždavinį į mažesnius sprendimus. Taikant dinaminį programavimą, sunaudosime daugiau atminties, bet viskas vyks greičiau.



**Algoritmo sudarymo metodika**

Uždavinio sprendimas dinaminiu programavimu susideda iš 4 žingsnių

1. Nusakyti optimalią uždavinio sprendinio struktūrą

2. Rekursiškai apibrėžti optimalų uždavinio sprendinį

3. Apskaičiuoti optimalaus sprendinio reikšmę

4. Rasti galutinį sprendinį

**Konvejeris**, tai galime įsivaizduoti kaip grafą, kur turime kainą nuvykti į salą bei keliaudami per liniją, taip pat mokame kainą. Algoritmo tikslas, rasti pigiausią maršrutą. Sprendžiant šį uždavinį paprasčiausiai, realizacija būtu tokia, kad mums reikėtu tikrinti kiekvieną su kiekvienu (brute-force), todėl sudėtingumas Ω (2^n). Pritaikant dinaminį programavimą, mes galime į vieną FOR ciklą įvesti tikrinimą, kuris tikrina tiek viršutinę liniją, tiek apatinę ir visos jos būdus, taip gauname sudėtingumą O(n).

Source: <https://www.geeksforgeeks.org/assembly-line-scheduling-dp-34/>

**Bendro ilgiausio posekio radimas**

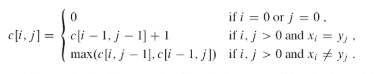
Tarkim turime žodžius ABCDGH ir AEDFHR, o atsakymas yra matosi, jog yra ADH. Patikrinsime naudodami algoritmą. Algoritmiškai pildome lentelę iš kairęs į dešinę bei palaipsniui leidžiamas į apačią.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | G | H |
| A | [U+2196.svg](https://en.wikipedia.org/wiki/File:U%2B2196.svg) 1 | [U+2190.svg](https://en.wikipedia.org/wiki/File:U%2B2190.svg)1 | [U+2190.svg](https://en.wikipedia.org/wiki/File:U%2B2190.svg)1 | [U+2190.svg](https://en.wikipedia.org/wiki/File:U%2B2190.svg)1 | [U+2190.svg](https://en.wikipedia.org/wiki/File:U%2B2190.svg)1 | [U+2190.svg](https://en.wikipedia.org/wiki/File:U%2B2190.svg)1 |
| E | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)1 | [U+2190.svg](https://en.wikipedia.org/wiki/File:U%2B2190.svg)1 | [U+2190.svg](https://en.wikipedia.org/wiki/File:U%2B2190.svg)1 | [U+2190.svg](https://en.wikipedia.org/wiki/File:U%2B2190.svg)1 | [U+2190.svg](https://en.wikipedia.org/wiki/File:U%2B2190.svg)1 | [U+2190.svg](https://en.wikipedia.org/wiki/File:U%2B2190.svg)1 |
| D | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)1 | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)1 | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)1 | [U+2196.svg](https://en.wikipedia.org/wiki/File:U%2B2196.svg) 2 | [U+2190.svg](https://en.wikipedia.org/wiki/File:U%2B2190.svg)2 | [U+2190.svg](https://en.wikipedia.org/wiki/File:U%2B2190.svg)2 |
| F | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)1 | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)1 | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)1 | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)2 | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)2 | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)2 |
| H | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)1 | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)1 | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)1 | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)2 | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)2 | [U+2196.svg](https://en.wikipedia.org/wiki/File:U%2B2196.svg) 3 |
| R | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)1 | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)1 | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)1 | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)2 | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)2 | [U+2191.svg](https://en.wikipedia.org/wiki/File:U%2B2191.svg)3 |

Nuo apatinio dešniojo kampo, sekame rodyklytes ir įtraukiame į „string“ eilutę tik raudonai pažymėtas eilutes (t.y. atitikmetnis)

Gauname **ADH**, tada apverčiame ir gauname **ADH** t.y. atsakymą.

Sudėtingumas yra **O(n\*m),** todėl, nes mes vieną kartą apeiname per visą 2D lentelę, kurios visų elementų kiekis yra n\*m.

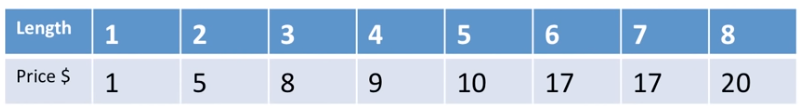
Rekurentinė lygtis: 

* 1. **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…).**

Strypų pjaustymo algoritmas pritaikomas kaip tarkime turime kažkokio tai ilgio dalyką (tarkim 8) ir skirtingai jį supjausčius, kiekviena dalis bus skirtingai vertinga.

Rekurentinė lygtis: 

Uždavinio pavyzdys:



Algoritmo tikslas, supjaustyti tą dalyką taip, kad gautumėme didžiausią kainą.

C(length) = price

C(1) = 1

C(2) = MAX(

V(1) + C(2-1) = 1 + 1 = 2 ;

V(2) + C(2-2) = 5 ) = 5

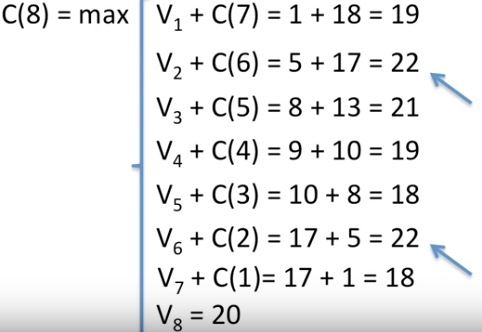
C(3) = MAX(

V(1) + C(3-1) = 1 + 5 = 6

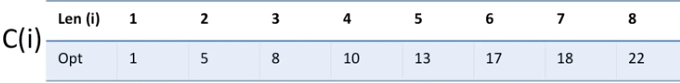
V(2) + C(3-2) = 5 + 1 = 6

V(3) + C(3-3) = 8 ) = 8

Ir taip toliau, kol gauname paskutinį



Tada atrenkame didžiausius ir gauname mūsų padalinimą. T.y. 8 ilgio strypą, reiktu skelt į 2 ir 6 ilgio.



Be dinaminio programavimo, šio algoritmo sudėtingumas būtu O(2^n), nes reikėtu tikrinti kiekvieną su kiekvienu. Pritaikius dinaminį programavimą, gauname sudėtingumą O(n^2), kadangi realizacijos atvėju, gautumėme ciklą cikle.

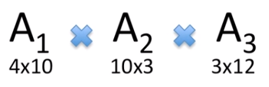
Source: <https://www.youtube.com/watch?v=ElFrskby_7M>

* 1. **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…).**

**Matricų sekos optimalaus dauginimo skaičiavimas** – tarkime turime įvairaus dydžio penkias matricas ir norime jas sudauginti. Ir priklausomai nuo jų išsidėstymo, mums gali daugybos veiksmų būti labai daug. Šio atgoritmo pagalba, sužinotumėme kaip tiksliai reikėtu dauginti matricas, kad sunaudotumėme mažiausiai procesoriaus.

Rekurentinė lygtis:  (knygoje yra kitokia lygtis, bet tolimesnis pavyzdis įspręstas taikant šią rekurentinę)

Pavyzdžiui, kai Ai yra matricą, o apačioje nurodytas jos dydis. Ir turime tokia daugybą. Reikia rasti būda, kaip sudėti matricas, kad išnaudotumėme mažiausią kiekį daugybos.



|  |  |  |  |
| --- | --- | --- | --- |
| i/j | 1 | 2 | 3 |
| 1 | 0 | 120 | 264 |
| 2 | X | 0 | 360 |
| 3 | x | X | 0 |

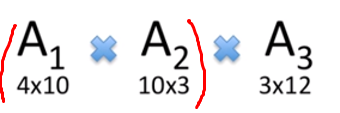
M(1,2)=M[1,1]+M[2,2]+p0\*p1\*p2=0+0+4\*10\*3=120

M(2,3)=M[2,2]+M[3,3]+p1\*p2\*p3=360

M(1,3)=min { (k=1) M[1,1]+M[2,3] + p0\*p1\*p3 = 740;

k(2) M[1,2]+M[3,3] + p0\*p2\*p3 = 264 } = 264

Geriausias variantas paskutiniame skaičiavime buvo, kaip k buvo lygus 2. Todėl skliaustus dedame nuo pradžios iki antrosios matricos.



Source: <https://www.youtube.com/watch?v=GMzVeWpyTN0>

* 1. **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 principai:**

* Godūs algoritmai priima sprendimą kuris tuo metų atrodo „geriausias“
* Gaunamas lokaliai geriausias sprendinys, tikintis, kad jis prives prie globaliai optimalaus sprendimo
* Bendriniu atvėju, godūs algoritmai negarantuoja optimalaus sprendimo. (Kaikuriems uždaviniams galima rasti ir optimalų sprendimą)
* Algoritmų sudarymo procedūra panaši į dinaminio programavimo algoritmų sudarymo eigą
* Privalumai: Paprastas, lengva implementuoti
* Minusai: labai dažnai tai nėra globaliai optimalus algoritmas

**Pavyzdys**. Yra žmogus kuris turi neribotą 5, 10, 20 kupiūrų kiekį. Tau reikia 35 eurų.

Greita algoritmo idėją: Aš tau duosiu didžiausią kupiūros kiekį, tol kol neperžengsiu 35 eurų sumos, jeigu peržengiu, sumažinu kupiūros dydį, tol kol pasieksiu išdalintą sumą 35 eurus.

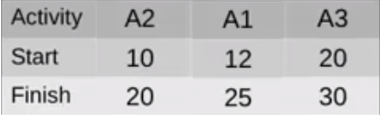
**Maksimalios procesų aibės radimo uždavinys**

**Problema**: Duota n įvykių aibė su pradžios ir pabaigos laiku. Vienu metu gali vykti tik viena užduotis. Rasti didžiausią įvykių kiekį, kurios gali būti atliktos.



**Godaus algoritmo sprendimas –** surušiuojame activity pagal finished time. Atlikti visus activities pagal surušiuotą principą.

Surušiuota lentelė



Pradedame **A2** activity, baigiame 20-ą sekundę. Todėl **A1** pradėti negalime. Pradedame **A3** ir ją baigiame. Todėl bendra activity **sumą yra 2.**

Rekurentinė lygtis: 

Jeigu turime surušiuotą lentelę, sudėtingumas **O(n),** dėl vieno ciklo.

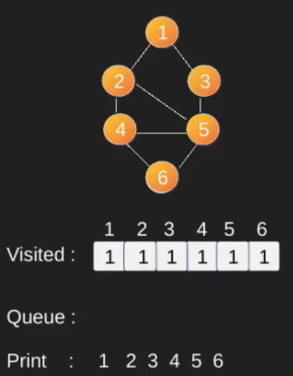
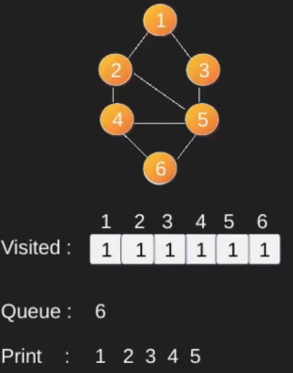
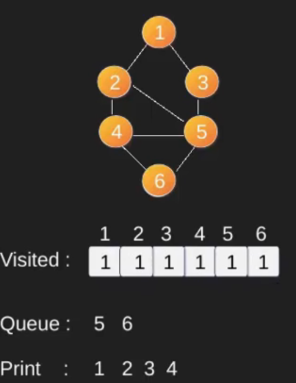
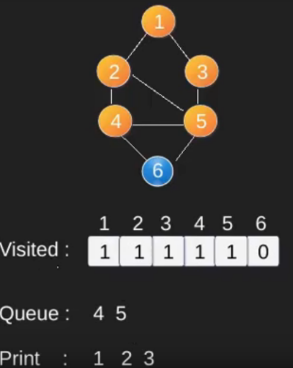
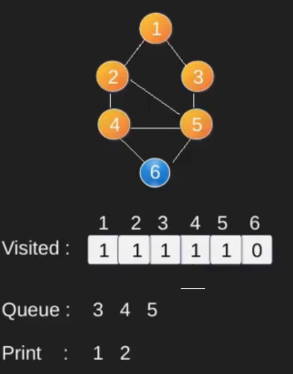
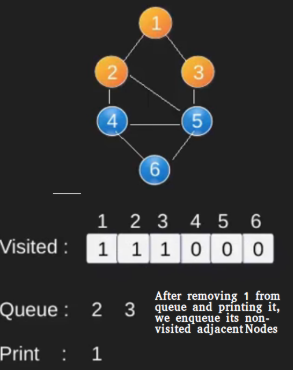
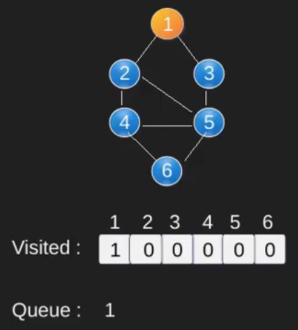
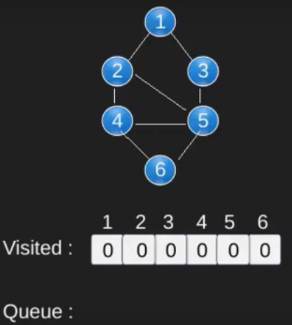
Jeigu turime nesurušiuotą lentelę, sudėtingumas yra **O(n log n),** dėl vieno ciklo bei logn yra surušiavimo greitis.

Source: <https://www.youtube.com/watch?v=poWB2UCuozA> <https://www.youtube.com/watch?v=HzeK7g8cD0Y>

Visuose pirmos dalies klausimuose algoritmo struktūra yra - **optimal substructure**

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

Algoritmo idėją yra apvaikščioti visas įmanomas galimas grafo viršūnės. Algoritmas grąžina (Print eilutė) kurios viršūnės buvo paliestos pirmos.

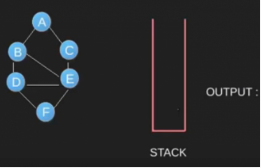
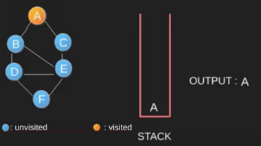
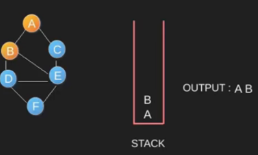
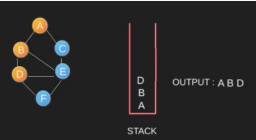
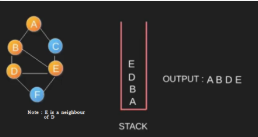
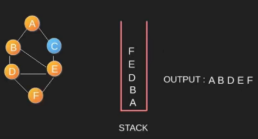
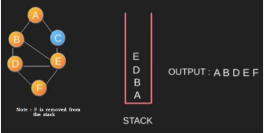
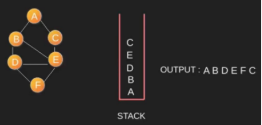
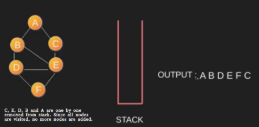


**Sudėtingumas yra: O(V+E), kaip V yra salelės, o E kampai.**

Source: <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>

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

Skirtumas nuo paieškos į plotį, tai, kad viena iteracija mes nuspalvinti tik vieną salelę galime. Visuomet judame link mažiausios viršūnės, pasiekė tai ką baigėme, grįžtame ten kur pradėjome ir tęsiame viršūnių „aplankymą“. Bendrasis algoritmo rezultatas yra, tos viršūnės kurios buvo aplankytos pirmosios (tipo taip surušiuotos)

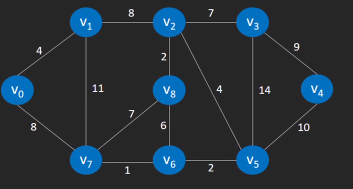
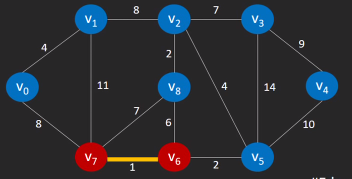
        

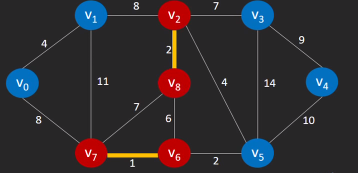
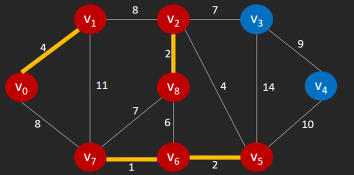
**Sudėtingumas yra: O(V+E), kaip V yra salelės, o E kampai.**

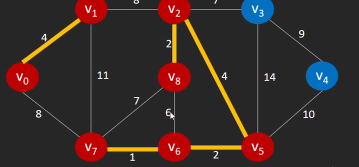
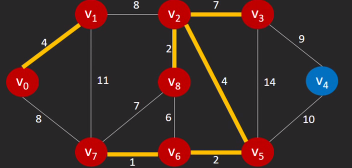
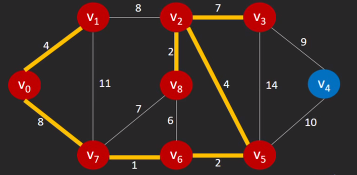
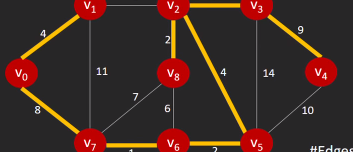
Source: <https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>

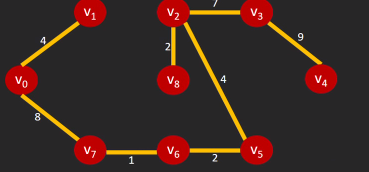
* 1. **Kruskalo algoritmas (23.2 sk. 631-633 psl.) ir sudėtingumo įvertinimas.**

**Algoritmo tikslas,** yra sužymėti visas linijas, kad nesudarytumėme ciklo. Pradžioje žymime tas linijas, kurių svoriai yra mažiausi ir žymime tik toliau nepažymėtas mažiausią svorį turinčias linijas.

Ir gaunasi atsakymas 

**Sudėtingumas:** O(ElogE) or O(ElogV), kaip V – salėlės, E – kampai

Source: <https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/>

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

**Algoritmo tikslas,** yra sužymėti visas linijas, kad nesudarytumėme ciklo. Pradžioje žymime tas linijas, kurių svoriai yra mažiausi ir žymime tik toliau nepažymėtas mažiausią svorį turinčias linijas

**Sudėtingumas**: O(V^2) nes bus ciklas cikle, kur pirmas ciklas reikš, kad turime tiesiog apeiti visą grafą, o kitas kur turėtumėme eitį iš galimų reikšmių.

Source: <https://www.youtube.com/watch?v=cplfcGZmX7I>

1. **Trečia klausimų dalis (4 balai)**
   1. **Daugiagijo dinaminio programavimo metodika. (27 sk. 772-791 psl.)**

**Daugiagijo (lygiagretaus) programavimo strategijos:**

* Kiekvienam procesoriui/branduoliai atskira atmintis. Tai reiškia, kad atskiro procesoriaus/branduolio atminties negalės TIESIOGIAI peržiūrėti kiti procesoriai.
* Visiems procesoriams/branduoliams bendra atmintis. Tai reiškia, kad kiti procesoriai/branduoliai GALĖS TIESIOGIAI valdyti bei manipuliuoti kitų procesorių/branduolių atmintimi.

**Daugiagijo programavimo principai**

* Statinės gijos – kaip gijų kiekis programos gyvavimo laikotarpyje nekinta arba jų kiekis labai mažai kinta.
* Dinaminės gijos – leidžia programuotojui nurodyti lygiagretinimo lygį labai nekeičiant kodo bei nesirūpinant balansavimu bei komunikavimu.

**Pseudo koduose naudojami žodžiai**

* Parallel – ciklo iteracijos vyksta vienu metu skirtingose gijose
* Spawn – sukuria gija
* Sync – gijų/procesų sinchronizavimas
  1. **Daugiagijai matricų dauginimo algoritmai ir jų vykdymo laikų bei išlygiagretinimo koeficientų įvertinimas. (27.2 sk. 792-797 psl.)**

**Kvadratinių matricų daugybos algoritmas LYGIAGREČIAI**

* Pradinių A ir B ir rezultatų matricos (C) išskaidymas į n/2 \* n/2 dydžio matricas. **O(1)**
* Sukuriame 10 matricų S1, S2, ... , S10 kurių kiekvieną yra n/2 \* n/2 dydžio. **O(ln n)**
* Panaudojus 2 žingsnyje sukurtas 1 ir 10 matricas, rekursyviai sudaromos 7 tarpinės matricos P1, P2, ... , P7. Kurių kiekviena yra n/2 \* n/2 dydžio.
* Apskaičiuojamos C11,C12,C21,C22 sudedant skirtingas matricų kombinacijas. **O(ln n)**

**Bendras gaunasi O(n^ln7 / lg^2n)**

Realiai, tarkim dauginant 4x4 matricą su kita 4x4 matrica, kiekvienai pirmosios matricos eilutėi galime išskirti po atskirą giją, taip dinamiškai paspartindami algoritmą.

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

Daugiagijų programavimu galima realizuoti MERGE SORT rikiavimo algoritmą, kaip kiekvieną atskirą uždavinį priskiriame naujajai gijai, kuri jį apdoroją. Pritaikius lygiagretinumo radimo algoritmą, gauname, kad lygiagretumas lygus **O(n/lg^2 n)**

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

**Šių amortizacinių algoritmų analizės tikslas yra surasti viršutinį laiko įvertį duomenų struktūroms.**

**Metodai:**

* **Aggregate analysis**
* **Accounting method**
* **Potential method**

**Aggregate analysis**

Tarkim pritaikius agregatine analize Dynamic Array duomenų struktūrai. Žinome, jog dinaminis masyvas prasiplečia, kaip jam trūksta vietos. Tarkim turime 4 vietų masyvą ir jį užpildome duomenimis, jis prasiplečia iki 8 vietų masyvo bei persikelia 4 turimas reikšmes. Žinome, kad įkėlimas į masyvą kainuoja **O(1)** , o perkelimas viską į naują masyvą, kainuotu **O(n).**

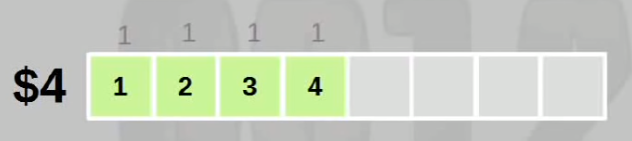


Pradėjome pildyti masyvą, pirmas 3 reikšmes įkėlėme paprastai, įkeliant ketvirtąją mums reikėjo susikurti naują masyvą bei perkelti visas reikšmes. Galime paskaičiuoti, kad užpildyti 32 vietų masyvą mums kainavo 63 operacijos, iš to galime spręsti, jog sudėtingumas viso preceso yra O(2n) -> O(n). O vieno įkėlimo vidutinė reikšmė yra O(2) -> O(1).

**Accounting method**

Principas yra toks, kad su kiekviena gauta reikšme į masyvą, mes su ja gauname 3 „dolerius“, viena perkėlimo operacija mums kainuoja 1 „dolerį“, mes negalime nueiti į skolas, turime likti su 0 arba daugiau dolerių. Pirmas doleris bus skirtas įkėlimui į duomenų struktūrą (dynamic array). Vėliau, kaip mums reikės praplėsti masyvą mums kainuos dar 1 „dolerį“ perkėlimas į kitą duomenų struktūrą.

Taigi taikant šią logiką, insertiname 4 skaičius į masyvą, jis iškarto prasiplečia ir perkeliame reikšmes (viršuje skaičiukas nurodo kiek dar dolerių turi ta reikšmė)



Toliau įdedame dar 4 reikšmes, joms lieka po 2 „dolerius“

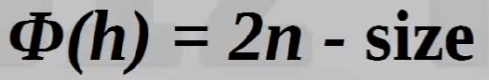


Iš karto praplečiame masyvą ir 5-8 reikšmių dolerius sunaudojame jog perkelti VISĄ masyvą.



Tada non-stop taikome šią logiką ir gauname insertinimo kainą O(3) -> O(1).

**Potential method**

Šio metodo išraiška yra 

Kaip n yra įdėtų reikšmių kiekis, o size yra visas esamo masyvo dydis. (Kadangi jis dinaminis, tai plėsis)

Šios funkcijos atsakymas reiškia duomenų struktūros potencialumą. Skaičius 0 reiškia, jog masyvo lygiai puse duomenų yra užpildyta. O pats principas veikimo yra kaip accounting method.

Source: <https://www.youtube.com/watch?v=T7W5E-5mljc>