

1. Tiesiniai rikiavimo algoritmai (8.2 sk. 194-196 psl. 8.3 sk. 198 psl. 8.4 sk. 201 psl.). Kodėl jie lenkia asimptotiškai optimalius rikiavimo algoritmus. (paaiškinti remiantis 8 sk.)

Tiesiniai rikiavimo algoritmai yra efektyvūs algoritmai, kurie naudojami rikiuoti sąrašus, kurių dydis yra žinomas iš anksto. Jie veikia su laiko sudėtingumu $O(n)$, kur n yra sąrašo elementų skaičius.

- Rikiavimas skaičiuojant (sveiki ne neigiami skaičiai);
- Pozicinis rikiavimas (turi turėti pozicijas ir vienodą jų skaičių);
- Kišeninis rikiavimas (tolygiai pasiskirstę intervale nuo 0 iki 1).

Rūšiavimas skaičiuojant (counting sort)

Apribojimai: Rūšiuojami elementai gali turėti sveikas reikšmes iš intervalo $0 \dots k$, čia k -teigiama sveika konstanta. Jei rūšiuojama n elementų rūšiavimas trunka $k=O(n)$ tada $T(n)=O(n)$.

Counting_sort(A,B,k)

1. for $i \leftarrow 0$ to k
2. do $C[i] \leftarrow 0$
3. for $j \leftarrow 1$ to $\text{length}[A]$
4. do $C[A[j]] \leftarrow C[A[j]] + 1$
5. $C[i]$ išsaugoma kiekis elementų lygių i .
6. for $i \leftarrow 1$ to k
7. do $C[i] \leftarrow C[i] + C[i-1]$
8. $C[i]$ -išsaugoma elementų skaičių neviršinančių i .
9. for $j \leftarrow \text{length}[A]$ down to 1
10. do $B[C[A[j]]] \leftarrow A[j]$
11. $C[A[j]] \leftarrow C[A[j]] - 1$

Pvz.:

- 1) A: $2^1 5^2 3^3 0^4 2^5 3^6 0^7 3^8$
C: $2^0 0^1 2^2 3^3 0^4 1^5$ (3,4 eil)
- 2) C: $2^0 2^1 4^2 7^3 7^4 8^5$ (6,7 eil)
- 3) A: 2 5 3 0 2 3 0 3
B: 1 2 3 4 5 6 3 8
C: 2 2 4 6 7 8

Sudėtingumas, jei $h=(n)$, $T(n)=O(k+1)+O(n)+O(k)+O(n)=O(n+k)=O(n)$.

RADIX-SORT(A, d)

- 1 for $i \leftarrow 1$ to d
- 2 do Stabilus rikiavimas A masyvo pagal i poziciją (skaitmenį).

Sudėtingumas $T(n) = \Theta(d(n+k))$, kai rikiuojama n d pozicijų skaičių, kurių kiekvienoje pozicijoje gali būti k reikšmių.

BUCKET-SORT(A)

```
1   $n \leftarrow \text{length}[A]$ 
2  for  $i \leftarrow 1$  to  $n$ 
3      do įterpti  $A[i]$  į  $B[\lfloor nA[i] \rfloor]$  sąrašą
4  for  $i \leftarrow 0$  to  $n - 1$ 
5      do surikiuojame  $B[i]$  sąrašą įterpimo būdu
6  Sujungiame sąrašus  $B[0], B[1], \dots, B[n - 1]$ 
```

Idėja sudėti masyvo A elementus į n sąrašų taip, kad elementai iš $B[i]$ sąrašo būtų mažesni už kiekvieną $B[j]$ sąrašo elementus, jei tik $i < j$. Kadangi skaičiai tolygiai pasiskirstę tai tikėtina kad tie sąrašai bus „maždaug“ vienodi ir po to atlikus $B[0] \dots B[n-1]$ sąrašų rūšiavimą suliejimas tiesiog nuoseklus surašymas. Tarkime, kad n_i -elem. Skaičius i -tajam sąrašė. 1-6 išskyrus 5

vykdoma per $O(n)$ laikų, o 5 per $O(n^2)$, nes rūšiavimas atliekamas su įterpimu. $T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$ Vidurkis

$$E(T(n)) = E\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right] = \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2]) = \Theta(n)$$

Tiesiniai rikiavimo algoritmai lenkia asimptotiškai optimalius algoritmus, nes jų veikimo laikas labai greitai auga didinant duomenų kiekį. Pvz., jei turime 1000 elementų sąrašą, tiesinio rikiavimo algoritmas gali atlikti iki 1 000 000 operacijų, o asimptotiškai optimalus algoritmas tik apie 10 000 operacijų. Taigi, dideliuose duomenų rinkiniuose asimptotiškai optimalūs algoritmai bus žymiai efektyvesni už tiesinius rikiavimo algoritmus.

Pavyzdžiui, greitojo rikiavimo algoritmas veikia su laiko sudėtingumu $O(n \log n)$. Palyginus su tiesiniais rikiavimo algoritmais, kurie veikia su laiko sudėtingumu $O(n)$, asimptotiškai optimalūs algoritmai gali žymiai sumažinti veikimo laiką, ypač didelių sąrašų atveju.

2. Dinaminis programavimas (15 sk. 358 psl.). Algoritmų sudarymo metodika (15.3 sk. 379-389 psl.) šios metodikos taikymas sprendžiant Bendro ilgiausio posekio radimo (15.4 sk. 390-395 psl.)

Dinaminis programavimas – programavimo metodas, paremtas uždavinio skaidymu į mažesnes susijusias problemas, bei tų problemų sprendimų įsiminimu. Taigi laiko sąnaudos pakeičiamos atminties sąnaudomis.

Algoritmo sudarymo metodika




Uždavinio sprendimas diniminiu 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

Taikant šią metodiką bendro ilgiausio posekio radimui, mes galime efektyviai rasti ilgiausią bendrą posekį dviejų sekų. Algoritmas veikia taip: sukuriame dvimatį masyvą, kurio vienas išmatavimas atitinka vienos sekos elementus, o kitas - kitos sekos elementus. Tada, naudodami rekursyvines lygtis ir jau apskaičiuotus rezultatus, užpildome masyvą, palaipsniui didindami ilgiausią bendrą posekį. Galiausiai, naudodamiesi masyvu, galime atkurti ilgiausią bendrą posekį.

Bendro ilgiausio posekio radimas

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

	A	B	C	D	G	H
A	 1	←1	←1	←1	←1	←1
E	↑ ₁	←1	←1	←1	←1	←1
D	↑ ₁	↑ ₁	↑ ₁	 2	←2	←2
F	↑ ₁	↑ ₁	↑ ₁	↑ ₂	↑ ₂	↑ ₂
H	↑ ₁	↑ ₁	↑ ₁	↑ ₂	↑ ₂	 3
R	↑ ₁	↑ ₁	↑ ₁	↑ ₂	↑ ₂	↑ ₃

Nuo apatinio dešiniojo kampo, sekame rodyklėse ir įtraukiame į „string“ eilutę tik raudonai pažymėtas eilutes (t.y. atitikmenis). 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 $n*m$.

3. Dinaminis programavimas (15 sk. 358 psl.). Algoritmų sudarymo metodika (15.3 sk. 379-389 psl.) šios metodikos taikymas sprendžiant Strypo pjaustymo uždavinį (15.1 sk. 360-369 psl.).

- 1) Nustatome, kaip galime išskaidyti strypo pjaustymo uždavinį į mažesnes subproblemas. Vienas būdas yra dalinti plotą į strypus ir bandyti išpjauti kiekvieną strypą naudodami mažesnius objektus.
- 2) Sukuriame rekursyvines lygtis, kurios nurodo subproblemų sprendimus ir jų sąsajas su pradine problema. Šiuo atveju, galime apibrėžti lygtį, kuri apskaičiuoja maksimalų mažesnių objektų skaičių, kurį galima išpjauti iš kiekvieno strypo.
- 3) Naudodami dinaminio programavimo optimizavimą, išsaugome jau apskaičiuotas subproblemų reikšmes, kad išvengtume dubliuotų skaičiavimų ir optimizuotume algoritmo veikimą. Tai gali būti atliekama naudojant lentelę ar masyvą, kurioje saugomos jau išspręstos subproblemų reikšmės.
- 4) Pagal jau apskaičiuotas subproblemų reikšmes ir išsaugotas reikšmes, galime konstruoti galutinį sprendimą pradinei problemai. Tai gali būti maksimalus mažesnių objektų skaičius, kurį galima išpjauti iš visų strypų, arba konkretūs išpjautų objektų deriniai.

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

$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i})$$

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 $\Theta(n^2)$.

$$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \dots, r_{n-1} + r_1)$$

arba

$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i})$$



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

4. Dinaminis programavimas (15 sk. 358 psl.). Algoritmų sudarymo metodika (15.3 sk. 379-389 psl.) ir šios metodikos taikymas sprendžiant Konvejerio (Surinkimo linijos planavimo)

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 būtų tokia, kad mums reikėtų tikrinti kiekvieną su kiekvienu (brute-force), todėl sudėtingumas $\Omega(2^n)$. 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 $O(n)$.

Konvejerio atveju pagalbinių uždavinių skaičius priklauso nuo darbo vietų skaičiaus, t. y. $\Theta(n)$, o pasirinkimo variantai tik 2, todėl sudėtingumas $\Theta(n)$.

5. 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 (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 algoritmo pagalba, sužinotume kaip tiksliai reikėtų dauginti matricas, kad sunaudotume mažiausią procesoriaus.

Rekurentinė lygtis:
$$M[i,j] = \begin{cases} 0 & \text{if } i=j \\ \min_{i \leq k < j} \{M[i,k] + M[k+1,j] + p_{i-1}p_kp_j\} \end{cases}$$

Pavyzdžiui, kai A_i yra matrica, o apačioje nurodytas jos dydis. Ir turime tokia daugybą. Reikia rasti būdą, kaip sudėti matricas, kad išnaudotume mažiausią kiekį daugybos.

$$\begin{matrix} A_1 & \times & A_2 & \times & A_3 \\ 4 \times 10 & & 10 \times 3 & & 3 \times 12 \end{matrix}$$

$$M(1,2) = M[1,1] + M[2,2] + p_0 * p_1 * p_2 = 0 + 0 + 4 * 10 * 3 = 120$$

$$M(2,3) = M[2,2] + M[3,3] + p_1 * p_2 * p_3 = 360$$

$$M(1,3) = \min \{ (k=1) M[1,1] + M[2,3] + p_0 * p_1 * p_3 = 740;$$

$$k(2) M[1,2] + M[3,3] + p_0 * p_2 * p_3 = 264 \} = 264$$

Suskaiciavę kur geriausiai dėti skliaustus ir gauname:

$$\left(\begin{matrix} A_1 & \times & A_2 \\ 4 \times 10 & & 10 \times 3 \end{matrix} \right) \times \begin{matrix} A_3 \\ 3 \times 12 \end{matrix}$$

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

Daugiagijai matricų dauginimo algoritmai yra naudojami siekiant efektyviai padauginti du ar daugiau matricų.

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 **$O(n^{\ln 7} / \lg^2 n)$**

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

7. Daugiagijai rikiavimo algoritmai (rikiavimo suliejimu pavyzdys) ir jų vykdymo laikų bei lygiagretinimo koeficientų įvertinimas. (27.3 sk. 797-804 psl. Amortizacinė algoritmų analizė. (17 sk. 451-462 psl.)

Daugiagijų rikiavimo algoritmų veikimo laikas priklauso nuo įvesties dydžio. Rikiavimo suliejimo algoritmo veikimo laiko sudėtingumas yra $O(n \log n)$, kur n yra sąrašo elementų skaičius. Tai reiškia, kad algoritmas yra efektyvus didelių sąrašų rikiavimui.

Lygiagretinimas daugiagijų rikiavimo algoritmų atveju gali būti naudingas, ypač dideliems sąrašams, leidžiant padalinti rikiavimo operacijas į kelias skirtingas gijas ar procesus, kurie gali vykdyti skaičiavimus tuo pačiu metu. Tačiau lygiagretinimas gali sukelti papildomų koordinavimo ir sinchronizavimo iššūkių, ypač kai reikia sujungti rikiuotas dalis. Šie iššūkiai gali turėti įtakos bendram lygiagretinimo koeficientui ir veikimo laikui.

Amortizacinė algoritmų analizė leidžia įvertinti vidutinį veikimo laiką per visą algoritmo vykdymą, o ne tik vienos operacijos atveju. Tai suteikia geresnį supratimą apie algoritmo efektyvumą ir leidžia įvertinti algoritmo našumą praktinėje situacijoje.

8. NP sudėtingumas. 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 $O(n^k)$ - čia k tam tikra konstanta.

P ir NP klasės

- P uždavinių klasė – uždaviniai, kurie sprendžiami per polinominę laiko trukmę $O(n^k)$. Čia k – konstanta, n – įvedamų duomenų kiekis.
- NP uždavinių klasė – uždaviniai, kurių sprendimą galima patikrinti per polinominį laiką. $P \subseteq NP$

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 vienas NP uždaviniui polinominis sprendimo algoritmas, tai bet kuriam kitam šios klasės uždaviniui egzistuoja toks algoritmas.

Jau nagrinėjome kelis tokius uždavinius:

- 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.
- Hamiltono ciklas. Reikia patikrinti ar duotajame grafe $G = (V, E)$ egzistuoja ciklas, jungiantis visas jo viršūnes.
- Diskretusis kuprinės užpildymo uždavinys. Turime n daiktų, kurių tūriai yra v_1, v_2, \dots, v_n , o kaina p_1, p_2, \dots, p_n . Reikia rasti tokį daiktų rinkinį, kuris tilptų į V tūrio kuprinę ir krovinio vertė būtų didžiausia.

- Dėžių užpildymo uždavinys. Turime keletą vienetinio tūrio dėžių ir n daiktų, kurių dydžiai v_1, v_2, \dots, v_n , čia $0 < v_j < 1$. Šiuos daiktus reikia sudėti į kuo mažesnę skaičių dėžių

9. Genetinis optimizavimas [paskaitų medžiaga].

Klasikinis genetinis algoritmas remiasi kandidatų sprendinių rinkiniu, kuris yra norimos išspręsti optimizavimo problemos sprendimas. Sprendinys yra potencialus kandidatas į optimizavimo uždavinio optimumą. Jo atvaizdavimas atlieka svarbų vaidmenį, nes nuo atvaizdavimo priklauso genetinių operatorių pasirinkimas. Atstovybės paprastai yra verčių sąrašai, o bendriau - simbolių rinkiniai. Jei jie yra tolydūs, vadinami vektoriais, jei sudaryti iš bitų, vadinami bitų eilutėmis. Kombinatorinių uždavinių atveju sprendinius dažnai sudaro sąrašė esantys simboliai. Pavyzdžiui, keliaujančio pardavėjo uždavinio atveju kelionė vaizduojama kaip maršrutas.

Kryžminimas yra operatorius, leidžiantis sujungti dviejų ar daugiau sprendimų genetinę medžiagą. Kai kurios išimtys nežino skirtingų lyčių, todėl turi tik vieną tėvą. Genetiniuose algoritmuose kryžminimo operatorius galime išplėsti iki daugiau nei dviejų tėvų.

Mutacijos operatoriai keičia sprendinį jį trikdydami. Mutacija pagrįsta atsitiktiniais pokyčiais. Šio trikdymo stiprumas vadinamas mutacijos greičiu. Tęstinėse sprendinių erdvėse mutacijos sparta dar vadinama žingsnio dydžiu.

10. Grafų algoritmų sudėtingumo įvertinimas: Paieškos į plotį algoritmas (22.2 sk. 594- 597 psl.); Paieškos į gylį algoritmas; Kruskalo algoritmas (23.2 sk. 631-633 psl.); Prima algoritmas (23.2 sk. 634-636 psl.); Trumpiausi keliai iš vienos viršūnės (24 sk. 641-650 psl.); Belmano – Fordo algoritmas (24.1 sk. 651 psl.) Deikstra algoritmas (24.3 sk. 658-659 psl.).

Struktūra nusakanti grafą

Grafą $G = (V, E)$ sudaro dvi aibės:

1. V viršūnių aibė;
2. E lankų (briaunų) aibė.

Šių aibių elementų skaičiumi išreikšime naudojamo algoritmo grafe sudėtingumą.

Paieška į plotį (BFS - Breadth-First Search):

- a) Laiko sudėtingumas: $O(V + E)$, kur V yra viršūnių skaičius, o E - briaunų skaičius.

Paieška į gylį (DFS - Depth-First Search):

- a) Laiko sudėtingumas: $O(V + E)$, kur V yra viršūnių skaičius, o E - briaunų skaičius.

Kruskalo algoritmas (Minimalaus išlaikančio medžio radimo algoritmas):

- a) Laiko sudėtingumas: $O(E \log E)$, kur E yra briaunų skaičius.

Prima algoritmas:

Laiko sudėtingumas: $O(E \log V)$, kur V yra viršūnių skaičius, o E - briaunų skaičius. Šis įvertinimas taikomas, kai naudojama mininė eilė arba Fibonacci eilė prioritetinei eilei.

Belmano-Fordo algoritmas (Trumpiausių kelių radimo algoritmas su neigiamomis briaunomis):

- a) Laiko sudėtingumas: $O(V * E)$, kur V yra viršūnių skaičius, o E - briaunų skaičius.

Trumpiausi keliai iš vienos viršūnės (Dijkstra algoritmas):

- a) Laiko sudėtingumas: $O((V + E) \log V)$, kur V yra viršūnių skaičius, o E - briaunų skaičius, naudojant mininę eilę arba Fibonacci eilę.

Paieška į plotį (breadth - first search)

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

- balta – nenagrinėta,
- pilka – pažymėta, kaip kaimyninė viršūnė nagrinėtos viršūnės, bet gali turėti kaimynystėje ir baltų viršūnių,
- juoda – išnagrinėta viršūnė.

Paieška į plotį algoritmas sudaro paieškos į plotį medį, kuris pradinio laiko momentu turi tik šakninę viršūnę s , be to kiekvienai viršūnei u priskiriamas atributas $\pi[u]$, nurodantis kokia viršūnė eina prieš ją ir atributas $d[u]$ nurodantis koku atstumu yra nutolusi viršūnė nuo pradinės s .

Kruskalo algoritmas randa saugų kraštą, kurį galima įtraukti į augantį mišką, iš visų bet kuriuos du miško medžius jungiančią briauną, mažiausio svorio briauną (u,v) .

Kruskalo algoritme aibė A yra miškas, kurio viršūnės yra visos duotojo grafo viršūnės. Į A pridėta saugi briauna visada yra mažiausio svorio grafo briauna, jungianti dvi skirtingas sudedamąsias dalis. Primo algoritme aibė A sudaro vieną medį. Į A pridėta saugi briauna visada yra mažiausio svorio briauna jungianti medį su viršūne, kurios nėra medyje.

Paieška į gylį (Depth-first search)

DFS(G)

```

1  for each vertex  $u \in V[G]$ 
2      do  $color[u] \leftarrow WHITE$ 
3       $\pi[u] \leftarrow NIL$ 
4   $time \leftarrow 0$ 

```

$\Theta(V)$

```

5  for each vertex  $u \in V[G]$ 
6      do if  $color[u] = WHITE$ 
7          then DFS-VISIT( $u$ )

```

Vykdoma $|V|$ kartų.

DFS-VISIT(u)

```

1   $color[u] \leftarrow GRAY$       ▷ White vertex  $u$  has just been discovered.
2   $time \leftarrow time + 1$ 
3   $d[u] \leftarrow time$ 
4  
5   $color[u] \leftarrow BLACK$       ▷ Blacken  $u$ ; it is finished.
6   $f[u] \leftarrow time \leftarrow time + 1$ 

```

```

4  for each  $v \in Adj[u]$       ▷ Explore edge  $(u, v)$ .
5      do if  $color[v] = WHITE$ 
6          then  $\pi[v] \leftarrow u$ 
7              DFS-VISIT( $v$ )

```

Vykdoma $|Adj[u]|$ kartų.

$O(1)$

$$\sum_{v \in V} |Adj[v]| = \Theta(E)$$

$$\text{Bendras laikas } \Theta(V) + |V|O(1) + \Theta(E) = \Theta(V + E)$$

Trumpiausias kelias iš vienos viršūnės

Duotas grafas $G = (V, E)$ ir svorių funkciją:

$$w: E \rightarrow R,$$

čia V – viršūnių aibė, E – lankų (briaunų) aibė.

Kelias $p = \langle v_0, v_1, \dots, v_k \rangle$, o jo svoris $w(p) = \sum_{i=1}^k w(v_{i-1}, v_i)$.

Tikslas: rasti trumpiausią kelią iš viršūnės u į viršūnę v , kuris turi mažiausią svorį

$$\delta(u, v) = \begin{cases} \min \{w(p) : u \xrightarrow{p} v\}, & \text{jei egzistuoja} \\ \infty, & \text{jei neegzistuoja.} \end{cases}$$

Šis uždavinys leidžia spręsti:

- Ieškoti trumpiausio kelio iki nurodytos viršūnės;
- Ieškoti trumpiausio kelio tarp poros viršūnių;
- Ieškoti trumpiausių kelių tarp visų grafo viršūnių.

Bellmano-Fordo algoritmas sprendžia vieno šaltinio trumpiausių kelių problemą bendruoju atveju, kai briaunų svoriai gali būti neigiami. Turint svertinį kryptinį grafą $G = (V, E)$ su šaltiniu s ir svorio funkcija $w: E \rightarrow \mathbb{R}$, tai Bellmano-Fordo algoritmas grąžina loginę reikšmę, rodančią, ar yra, ar nėra neigiamo svorio ciklas, kurį galima pasiekti iš šaltinio

Dijkstros algoritmas sprendžia vieno šaltinio trumpiausių kelių problemą svertinėje, $G = (V, E)$ atveju, kai visi briaunų svoriai yra neneigiami. Todėl šiame skyriuje darome prielaidą, kad $w(u, v) \geq 0$ kiekvienai briaunai $(u, v) \in E$. Kadangi pamatysime, kad gerai įgyvendinus Dijkstros algoritmą, jo veikimo laikas yra mažesnis nei Bellmano-Fordo algoritmo.

11. Godūs algoritmai ir jų sudarymo metodika. (16 sk. 414psl. / 16.2 sk. 423-427 psl.) ir jos taikymas sudarant maksimalios procesų aibės radimo uždavinį (16.1 sk. 415-418 psl.).

Godūs algoritmai dažniausiai naudojami optimizavimo uždavinių sprendimui

Tipinio optimizavimo uždavinio sprendime, algoritmas turi daug žingsnių, kuriuose

reikia priimti vienokį ar kitokį sprendimą

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

-
- gaunamas lokaliai geriausias sprendinys, tikintis, kad jis ves prie globaliai optimalaus sprendinio
 - bendrinio atveju, godūs algoritmai **negarantuoja** optimalaus sprendinio, nors kai kuriems uždaviniams galima rasti ir optimalų sprendinį
 - algoritmų sudarymo procedūra panaši į dinaminio programavimo algoritmų sudarymo eigą

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

Maksimalios procesų aibės radimas:

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 $\Theta(n)$, t.y. procesų kiekis masyvuose s ir f .

12. Godūs algoritmai. Algoritmų sudarymo metodika (16 sk. 414psl. / 16.2 sk. 423-427 psl.) ir jos taikymas sudarant Huffman kodą (16.3 sk. 429-435).

HUFFMAN(C)

```
1   $n = |C|$ 
2   $Q = C$ 
3  for  $i = 1$  to  $n - 1$ 
4      allocate a new node  $z$ 
5       $z.left = x = \text{EXTRACT-MIN}(Q)$ 
6       $z.right = y = \text{EXTRACT-MIN}(Q)$ 
7       $z.freq = x.freq + y.freq$ 
8       $\text{INSERT}(Q, z)$ 
9  return  $\text{EXTRACT-MIN}(Q)$     // return the root of the tree
```

Taikant gražiąją algoritmų sudarymo metodiką Huffman kodavimo atveju, pagrindinis tikslas yra sukurti efektyvų simbolių kodavimą, kuris minimizuoja naudojamą bitų skaičių. Huffman kodavimas yra beprasmių ženklų (simbolių) kodavimo metodas, kuriame simboliai yra atitinkami nulinės ir vieneto seka.

Huffman kodavimo algoritmo veikimo eiga yra ši:

- 1) Apskaičiuojamos simbolių pasikartojimo dažnumo tekstiniame įrašė arba kitame kontekste.
- 2) Sukuriamas medis, kuriame kiekvienas lapas atitinka simbolį ir jo kodas yra nustatomas remiantis dažnumu. Retai pasitaikančių simbolių kodai yra ilgesni, o dažnių simbolių kodai yra trumpesni.
- 3) Kodai naudojami simbolių užkodavimui arba atkodavimui.

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

Amortizacinė analizė (17 sk.)

Tikslas surasti viršutinį laiko įvertį, reikalingą atlikti seką veiksmų su duomenų struktūra. Tai atliekama vidurkinant visų operacijų laiką.

Tai leidžia parodyti, kad net esant „brangioms“ operacijoms, jos nedaro didelės įtakos.

Metodai

- Grupinė analizė (aggregate analysis)
- Buhalterinė apskaita (accounting method)
- Potencialų metodas (potential method)

Grupinė analizė

Randama $T(n)$ viršutinis įvertis n operacijų. Vidurkis $\frac{T(n)}{n}$ laikomas amortizaciniais kaštais kiekvienos operacijos

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.

Potencialų metodas

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 dabartinės duomenų struktūros būsenos, nepriklausomai nuo skaičiavimo istorijos kaip ji pateko į tą būseną.

14. Šakų ir rėžių metodas [paskaitų medžiaga].

Algoritmų sudarymo paradigma pagrinde skirta spręsti kombinatorinio / diskretaus optimizavimo problemoms (eksponentinis sudėtingumas), siekiant rasti **optimalų sprendinį**.

- 1) Taikant euristiką (godus pasirinkimas) randamas problemos sprendimas Xh . $B = f Xh$ nurodo geriausią žinomą sprendimą einamuoju metu.
- 2) Rekursyviai atliekamas pilnas perrinkimas, kai:
 - a) Nauja būseną pasirenkama taikant godų pasirinkimą (pigiausias, geriausias, ...)
 - b) Jei tikslo funkcijos reikšmė einamojoje būsenoje yra daugiau nei B (arba pažeidžiami kiti apribojimai), tolimesni rekursiniai veiksmai neatliekami (sprendiniai bus prastesni nei žinimas einamuoju metu).
 - c) Jei pasiekama pabaigos būseną, ir tikslo funkcijos reikšmė mažesnė nei B, atnaujinama B reikšmė ir išsaugomas naujas „geriausias“ sprendinys.

Privalumai:

- Dažnai pavyksta smarkiai sumažinti sprendinių aibę paieškos aibę.
- Galima rasti optimalų sprendinį.

Trūkumai:

- Laikui imlus sprendimas. Blogiausiu atveju liekama prie to pačio sudėtingumo (priklauso nuo duomenų ir problemos) kaip ir sprendžiant pilno perrinkimo uždavinį.
- Sudėtinga atlikti skaičiavimus lygiagrečiai.

15. Sprendinių medžio metodas [paskaitų medžiaga].

Skirtas sprendimo priėmimui **duomenų pagrindu**.

Sprendimų medis sudaromas remiantis kuo paprastesne klausimų seka, kuri veda prie atsakymo geriausiai atitinkančio turimus duomenis.

1 žingsnis: Sukuriama medžio šaknis su svarbiausiu kintamuoju.

2 žingsnis: Parenkamas sprendimas pagal didžiausią informacijos padalinimą.

3 žingsnis: Rekursyviai konstruojamas medis taikant 1-2 žingsnius tol, kol duomenys gali būti padalinami žemiausiame mazge.

Pranašumai:

- Nereikalauja pradinio duomenų apdorojimo.
- Gali būti naudingas duomenų analizei.
- Intuityvus, ir lengvai paaiškinamas (sprendimo priėmimo prasme).

Trūkumai:

- Laikui imlus sudarymo prasme, kai yra daug kintamųjų.
- Prisitaikymas prie duomenų.

Entropija Mažiau netvarkingam mazgui apibūdinti reikia mažiau informacijos, ir priešingai, labiau netvarkingam mazgui reikia daugiau informacijos.

$$H = - \sum_{i=1}^n (p_i \log_2 p_i) \Rightarrow \begin{array}{l} \mathbf{p_i} \text{ i-osios reikšmės pasirodymo tikimybė} \\ \mathbf{n} \text{ viso reikšmių} \\ \mathbf{H} \text{ entropija} \end{array}$$

Kauliukas su reikšmėmis 1–6

$$H = - \sum_{i=1}^6 \left(\frac{1}{6} \log \frac{1}{6} \right) \approx 2.58$$

Monetos metimas

$$H = - \sum_{i=1}^2 \left(\frac{1}{2} \log \frac{1}{2} \right) \approx 1$$