

1 skyrius

Algoritmų sudarymo būdai

1.1. Dinaminio programavimo metodas

Pilno variantų perinkimo algoritmai dažnai yra neefektyvūs dėl labai didelio nagrinėjamų variantų skaičiaus. Analizuodami tokius algoritmus matome, kad skirtingų variantų yra daug mažiau, nes skaičiavimuose generuojamos persidengiančios užduočių aibės. Šio trūkumo neturi *dinaminio programavimo* metodas, kurį pasiūlė Belmanas (*R. Bellman*). Jis naudotinas tada, kai tenkinamos tokios sąlygos:

- Skirtingų variantų yra daug mažiau nei perrinkimo algoritme nagrinėjamų variantų skaičius. Algoritmo vykdymo metu generuojamos užduočių aibės esmingai persidengia, todėl daug kartų sprendžiame tas pačias užduotis. Dinaminio programavimo metode įsimeiname išspręstų užduočių sprendinius ir sprendžiame tik naujus uždavinius.
- Uždavinys tenkina *Belmano* sąlygą, kad optimalus sprendinys yra sudarytas iš atskirų mažesnių užduočių optimalių sprendinių. Ši sąlyga užrašoma rekurentinės lygybės forma ir esminiai sumažina nagrinėjamų variantų skaičių.

Skaldyk ir valdyk algoritme užduotys generuojamos nuo viršaus į apačią, t.y. pradinis uždavinys skaidomas į kelias mažesnes užduotis, kurios toliau dalijamos į mažesnes. Dinaminio programavimo metode uždavinį pradeda spręsti nuo mažiausių ir lengvai išsprendžiamų užduočių, jų rezultatus naudojame spręsdami didesnes užduotis ir taip surandame viso uždavinio sprendinį. Realizuodami dinaminio programavimo metodą nagrinėjame tik tuos variantus, kurių gali prireikti sudarant optimalią strategiją.

Dinaminio programavimo metodo pagrindinius žingsnius aptarsime nagrinėdami tokį svarbų uždavinį. Reikia sudauginti n matricių $\mathbf{A}_1 \mathbf{A}_2 \cdots \mathbf{A}_n$, čia \mathbf{A}_i yra $p_{i-1} \times p_i$ dydžio matrica. Nors galutinis rezultatas nepriklauso nuo matricių dauginimo tvarkos, atliekamų veiksmų skaičius gali labai smarkiai skirtis. Priminsime, kad dauginami dvi $n \times m$ ir $m \times k$ dydžio matricas atliekame $2nmk$ aritmetinių veiksmų.

Nagrinėkime pavyzdį, kai dauginame 10×200 , 200×4 ir 4×80 dydžio matricas $\mathbf{A}_1 \mathbf{A}_2 \mathbf{A}_3$. Šią užduotį galime įvykdyti dviem skirtingais būdais:

1. $(\mathbf{A}_1 \mathbf{A}_2) \mathbf{A}_3$ atliekami $16000 + 6400 = 22400$ veiksmų,
2. $\mathbf{A}_1 (\mathbf{A}_2 \mathbf{A}_3)$ atliekami $128000 + 320000 = 448000$ veiksmų,

t.y. antruoju būdu atliekame dvidešimt kartų daugiau veiksmų. Skirtingų daugybos variantų skaičius labai greitai didėja, teisingas toks asimptotinis jų įvertis $\Omega(4^n/n^{3/2})$.

Dabar šį uždavinį spręsimė dinaminio programavimo metodu. Pirmiausia reikia rasti sprendinio optimalumo sąlygą ir įsitikinti, kad viso uždavinio optimalų sprendinį galime sudaryti iš mažesnių užduočių optimalių sprendinių. Pažymėkime

$$A_{i\dots j} = \mathbf{A}_i \mathbf{A}_{i+1} \cdots \mathbf{A}_j.$$

Tada optimalią sandaugos skaičiavimo tvarką apibrėžiame lygybe

$$\mathbf{A}_1 \mathbf{A}_2 \cdots \mathbf{A}_n = A_{1\dots k} A_{(k+1)\dots n},$$

ir paskutiniame algoritmo žingsnyje, dauginami matricas $A_{1\dots k}$ ir $A_{(k+1)\dots n}$ atliekame $2p_0 p_k p_n$ veiksmus. Prieš tai reikėjo apskaičiuoti pačias matricas $A_{1\dots k}$ ir $A_{(k+1)\dots n}$, šias sandaugas vėl reikia skaičiuoti optimaliu būdu. Taigi parodėme, kad viso uždavinio optimalaus sprendinio skaičiavimas suvedamas į dviejų mažesnių matricių sekų sandaugos skaičiavimo uždavinį.

Sudarysime rekurentinę lygtį, apibrėžiančią optimalų uždavinio sprendinį. Pažymėkime $m(i, j)$ aritmetinių veiksmų skaičių optimaliu būdu dauginant matricas $\mathbf{A}_i \mathbf{A}_{i+1} \cdots \mathbf{A}_j$. Tada, remiantis pateikta analize, egzistuoja toks k , kad šių matricių sandaugą išskaidome į matricių $A_{i\dots k}$ ir $A_{(k+1)\dots j}$ sandaugą

$$m(i, j) = m(i, k) + m(k + 1, j) + 2p_{i-1} p_k p_j.$$

Kadangi iš anksto nežinome, kuris k turi būti pasirinktas, tai gauname variacinę rekurentinę lygtį

$$m(i, j) = \begin{cases} 0, & i = j, \\ \min_{i \leq k < j} (m(i, k) + m(k + 1, j) + 2p_{i-1}p_k p_j), & i < j. \end{cases} \quad (1.1)$$

Aišku, kad $m(1, n)$ apibrėžia visų n matricių daugybos optimalaus algoritmo kaštus. Tačiau tiesioginis rekursijos algoritmas yra labai neefektyvus (netgi eksponentinio sudėtingumo), nes vėl daug kartų skaičiuojame tuos pačius koeficientus $m(i, j)$. Nesunku įvertinti, kad skirtingų koeficientų yra tik

$$n + (n - 1) + \dots + 1 = \frac{n(n + 1)}{2}.$$

Juos saugosime matricoje $\mathbf{M}(i, j)$, $1 \leq i \leq j \leq n$. Matricoje $\mathbf{P}(i, j)$ įsiminsime su kokia k reikšme radome mažiausią reikšmę (1.1) formulėje. Masyvo $p[i]$ elementuose saugome dauginamų matricių dydžius.

Skliaustų išdėstymo tvarkos skaičiavimas

```

void Skliaustai (int n, int p, int M, int P)
(1) for ( i=1; i<= n; i++ )
    (2) M(i, i) = 0;
    end for
(3) for ( r=2; r<= n; r++ )
    (4) for ( i=1; i<= n-r+1; i++ )
        (5) j = i + r - 1;
        (6) M(i, j) = ∞;
        (7) for ( k=i; k<= j-1; k++ )
            (8) m = M(i, k) + M(k+1, j) + 2 p(i-1) p(k) p(j);
            (9) if ( m < M(i, j) )
                (10) M(i, j) = m;
                (11) P(i, j) = k;
            end if
        end for
    end for
end for
end Skliaustai

```

Nesunku įrodyti, kad šio algoritmo sudėtingumas yra $\mathcal{O}(n^3)$, nes algoritmą sudaro trys įterpti ciklai, kurių indeksai perbėga aibes, turinčias ne daugiau kaip n elementų.

Pateiksime rekursinį algoritmą, kuris realizuoja matricių $\mathbf{A}_1\mathbf{A}_2\cdots\mathbf{A}_n$ daugybą. Informacija apie optimalią daugybos tvarką yra saugoma \mathbf{P} matricėje. Matricias dauginame vykdydami procedūrą $\text{MatricųDaugyba}(\mathbf{A}, \mathbf{P}, 1, n)$.

Matricių sekos daugybos algoritmas

```

matrica MatricųDaugyba (A, P, int i, int j)
(1) if ( j > i )
    (2) X = MatricųDaugyba (A, P, i, P(i,j));
    (3) Y = MatricųDaugyba (A, P, P(i,j)+1, j);
    (4) return XY;
else
    (5) return  $\mathbf{A}_i$ ;
end if else
end MatricųDaugyba

```

1.1 pavyzdys. Šešių matricių sandaugos optimali tvarka.

Skaičiuosime šešių matricių sandaugą $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_3\mathbf{A}_4\mathbf{A}_5\mathbf{A}_6$, kai šių matricių dydžiai yra tokie: 40×50 , 50×20 , 20×4 , 4×15 , 15×25 , 25×35 . Procedūros Skliaustai skaičiavimo rezultatai pateikti 1.1 paveiksle. Vykdydami algoritmo (3) ciklą paeiliui skaičiuojame matricių \mathbf{M} ir \mathbf{P} įstrižainių elementus. Paveiksle kiekvienos įstrižainės laukelis pažymėtas skirtinga spalva.

					0
				0	26250
			0	3000	10000
		0	2400	7000	15600
	0	8000	14000	21000	32000
0	80000	24000	28800	35000	45200

a)

					5
				4	5
			3	3	3
		2	3	3	3
	1	1	3	3	3

b)

1.1 pav. Matricių sandaugos optimalios tvarkos skaičiavimas dinaminio programavimo algoritmu: a) matrica \mathbf{M} , b) matrica \mathbf{P} .

Parodysime, kaip skaičiuosime koeficiento $M(2, 5)$ reikšmę. Algoritmo Skliaustai (3) cikle $r = 3$, po to vykdėme (7) ciklą:

$$k = 2 :$$

$$M(2, 2) + M(3, 5) + 2p_1p_2p_5 = 0 + 7000 + 2 \cdot 50 \cdot 20 \cdot 25 = 57000,$$

$$k = 3 :$$

$$M(2, 3) + M(4, 5) + 2p_1p_3p_5 = 8000 + 3000 + 2 \cdot 50 \cdot 4 \cdot 25 = 21000,$$

$$k = 4 :$$

$$M(2, 4) + M(5, 5) + 2p_1p_4p_5 = 14000 + 0 + 2 \cdot 50 \cdot 15 \cdot 25 = 51500.$$

Matome, kad mažiausia reikšmė yra $M(2, 5) = 21000$, ją gauname imdami $k = 3$, todėl $P(2, 5) = 3$.

Imdami informaciją, pateiktą matricioje \mathbf{P} ir pritaikę procedūrą Matricių Daugyba gauname, kad matricas reikia dauginti taip

$$(\mathbf{A}_1(\mathbf{A}_2\mathbf{A}_3))((\mathbf{A}_4\mathbf{A}_5)\mathbf{A}_6),$$

tada atliksime tik 45200 aritmetinius veiksmus.

1.2. Godieji algoritmai

Dažnai sprendžiame optimizavimo uždavinius, kai leistinų sprendinių aibėje ieškome sprendinio, minimizuojančio pasirinktą tikslo funkciją. Nagrinėjamų variantų skaičius gali būti toks didelis (nepolinominio sudėtingumo funkcija nuo uždavinio parametrų skaičiaus), kad jų visų negalime perrinkti net ir šiuolaikiniais superkompiuteriais.

Tokių uždavinių sprendinio paiešką dažniausiai išskaidome į n etapų ir kiekviename žingsnyje renkamės iš nedidelio baigtinio skaičiaus variantų m . *Godieji* algoritmai rekomenduoja rinktis lokaliai geriausią variantą duotojo žingsnio metu. Tada lokalaus pasirinkimo sudėtingumas $\mathcal{O}(m)$, o viso algoritmo sudėtingumas - tik $\mathcal{O}(nm)$ veiksmų.

Aišku, dažniausiai negalime garantuoti, jog taip spęsdami uždavinį randame tikslų sprendinį, tokie godieji algoritmai yra euristikos, leidžiančios labai sparčiai apskaičiuoti tikslaus sprendinio artinius. Euristikos pavyzdį sukonstravome ieškodami žirgo maršruto šachmatų lentoje.

Bet egzistuoja daug svarbių taikomųjų uždavinių, kai godieji algoritmai apibrėžia tikslų sprendinį. Tokius pavyzdžius nagrinėsime spęsdami grafų teorijos uždavinius, o šiame skirsnyje tik suformuluosime teorines sąlygas, kai godusis algoritmas tiksliai sprendžia optimizavimo uždavinį.

Pirmiausia parodysime, kaip sudaromi godieji algoritmai. Čia spęsimė uždavinius, kurių sudėtingumas yra eksponentinė duomenų skaičiaus funkcija, taigi godusis algoritmas negali garantuoti tikslaus sprendinio radimo.

1.2 pavyzdys. Gražos atidavimo uždavinys. Automatas gražą atiduoda monetomis, kurių nominalai yra:

$$V_1 > V_2 > \dots > V_m.$$

G vertės sumą reikia sudaryti taip, kad monetų skaičius būtų mažiausias. Taigi sprendžiame uždavinį:

$$\min_{(n_1, \dots, n_m) \in D} (n_1 + n_2 + \dots + n_m),$$

$$D = \{n_1 V_1 + n_2 V_2 + \dots + n_m V_m = G, \quad n_j \geq 0\}.$$

Tarkime, kad $V_m = 1$, tada visada galime parinkti bent vieną monetų kombinaciją, kurios suma lygi G . Algoritmo idėja labai paprasta: pirmiausia stengiamės gražą atiduoti didžiausio nominalo monetomis,

tokių monetų skaičius yra $n_1 = \lfloor G/V_1 \rfloor$, po to likusios sumos $G_1 = G - n_1 V_1$ didžiąją dalį atiduome V_2 nominalo monetomis $n_2 = \lfloor G_1/V_2 \rfloor$ ir t.t. Jeigu kuriame nors algoritmo žingsnyje $G_j < V_{j+1}$, tai V_{j+1} nominalo monetų nenaudojame.

Imkime monetų rinkinį

$$V_1 = 25, \quad V_2 = 11, \quad V_3 = 5, \quad V_4 = 1.$$

Godžiuoju algoritmu apskaičiuojame, kad 63 centų gražą reikia atiduoti taip

$$63 = 2 \cdot 25 + 1 \cdot 11 + 2 \cdot 1,$$

taigi naudojame penkias monetas. Nesunku patikrinti, kad toks sprendinys yra optimalus (išnagrinėkite atvejus, kai $n_1 < 2$), bet tada paaiškėja, kad egzistuoja ir dar vienas penkių monetų rinkinys, leidžiantis atiduoti tokią gražą

$$63 = 1 \cdot 25 + 3 \cdot 11 + 1 \cdot 5.$$

Jei $G = 15$, tai godžiuoju algoritmu gražą sudarome taip

$$15 = 1 \cdot 11 + 4 \cdot 1,$$

t.y. vėl naudojame penkias monetas. Tačiau egzistuoja geresnis sprendinys, kai klientui atiduome tris penkių centų monetas.

1.3 pavyzdys. Diskretusis kuprinės užpildymo uždavinys.

(Angl. *0 – 1 knapsack problem*). 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 jų vertė būtų didžiausia.

Sprendžiame optimizavimo uždavinį:

$$\max_{(n_1, \dots, n_m) \in D} (n_1 p_1 + n_2 p_2 + \dots + n_m p_m),$$

$$D = \{n_1 v_1 + n_2 v_2 + \dots + n_m v_m \leq V, \quad n_j \in (0, 1)\}.$$

Taigi optimizavimo uždavinio parametrai n_j gali būti lygūs tik vienetui arba nuliui. Pirmiausia apibrėžiame santykinę kiekvieno daikto vertę $s_j = \frac{p_j}{v_j}$. Visus daiktus rūšiuojame šios vertės mažėjimo tvarka, tarsime, kad

$$s_1 \geq s_2 \geq \dots \geq s_n.$$

Kuprinę stengiamės užpildyti didžiausios santykinės vertės daiktais. Juos įmame vieną po kito ir tikriname ar daikto tūris yra nedidesnis už likusią laisvą kuprinės dalį. Jei sąlyga yra išpildyta, tai daiktą talpiname į kuprinę, jei ne – įmame sekantį daiktą.

Nagrinėkime tokį pavyzdį. Turime aštuonis daiktus (v_j, p_j) :
 $(25, 50)$, $(20, 80)$, $(20, 50)$, $(15, 45)$, $(30, 105)$, $(35, 35)$, $(20, 10)$, $(10, 45)$.
 Apskaičiuojame jų santykinės vertes

$$S = \left\{ 2, 4, 2\frac{1}{2}, 3, 3\frac{1}{2}, 1, \frac{1}{2}, 4\frac{1}{2} \right\}$$

ir surūšiuojame daiktus verčių didėjimo tvarka
 $(10, 45)$, $(20, 80)$, $(30, 105)$, $(15, 45)$, $(20, 50)$, $(25, 50)$, $(35, 35)$, $(20, 10)$.
 Imkime kuprinę, kurios tūris $V = 80$. Naudodami godųjų algoritimą į kuprinę įdedame pirmuosius keturis daiktus, jų bendras tūris 75, o vertė 275 litai. Tai nėra geriausias sprendinys, nes imdami pirmuosius tris ir penktąjį daiktus užpildome visą kuprinę, o tokio krovinio vertė 280 litų. Taigi godusis algoritmas vėl yra tik euristicas.

Šiame pavyzdyje į kuprinę galėjome įdėti tik visą daiktą. Jei galima imti ir dalį daikto, tai įsitikinsime, kad godusis algoritmas visada apibrėžia tikslų uždavinio sprendinį.

1.4 pavyzdys. Tolydusis kuprinės užpildymo uždavinys.

(Angl. *fractional knapsack problem*). Užduoties sąlyga tokia pati kaip ankstesniame pavyzdyje, tik dabar daiktai yra dalūs ir galime imti jų dalį. Tai gali būti aukso plytelės ir aukso smėlis, arba supakuotas ir birus cukrus. Tada vykdydami godųjų kuprinės pildymo algoritimą imsime visą kiekį vertingų daiktų, o mažiausiai vertingo, bet dar tilpusio į kuprinę produkto imsime tik tiek, kiek liko laisvos vietos kuprinėje.

Imkime daiktų rinkinį, pateiktą ankstesniame pavyzdyje ir surūšiuotą jų santykinių verčių didėjimo tvarka $(10, 45)$, $(20, 80)$, $(30, 105)$, $(15, 45)$, $(20, 50)$, $(25, 50)$, $(35, 35)$, $(20, 10)$. Tada į kuprinę, kurios tūris $V = 80$, įsidėsime pirmuosius keturis daiktus (jų tūris 75, o vertė 275 litai) ir ketvirtadalį penktojo produkto (jo tūris 5, o vertė 12,5 litų). Taigi krovinys užpildo visą kuprinę, o jo vertė 287,5 litai.

Godžiųjų algoritmų naudojimo sritis

Šiame skyrelyje aptarsime, kada tikslinga naudoti godžiuosius algoritmus. Pirmiausia pastebėsime, kad tie uždaviniai, kurių tikslų sprendinį ran-

dame godžiuoju algoritmu, būtina tenkina sąlygą, kad šį sprendinį galime skaičiuoti išskaidę uždavinį į mažesnių užduočių seką ir suradę jų optimalius sprendinius. Pavyzdžiui, jei optimaliai užpildėme kuprinę (tiek diskrečiuoju, tiek ir tolydžiuoju atveju), tai išėmę iš jos vertingiausią daiktą vėl gauname panašų uždavinį: reikia užpildyti $(V - v_1)$ tūrio kuprinę likusiais daiktais.

Tokia optimalaus sprendinio priklausomybė nuo mažesnių šio uždavinio dalių optimalių sprendinių yra būdinga ir dinaminio programavimo metodu sprendžiamiems uždaviniams. Tačiau abu metodai visiškai skirtingai renkasi lokalaus uždavinio sprendinį. *Godžiajame* algoritme visada renkamės lokaliai vertingiausią sprendinį, kitame žingsnyje vėl imame vertingiausią sprendinį iš likusių variantų ir t.t. Taip rinkdamiesi lokalių uždavinių sprendinius tikimės, kad išlieka galimybė rasti globaliai optimalų sprendinį. *Dinaminio programavimo* metode prieš pasirinkdami lokalaus uždavinio sprendinį įvertiname tokio pasirinkimo pasekmes visiems tolimesniems variantams.

Algoritmo teisingumas. Godieji algoritmai yra labai efektyvūs, nes jų sudėtingumo funkcija yra polinominė (dažniausiai kvadratinė arba kubinė) duomenų skaičiaus n funkcija. Bet ne visada galime garantuoti, kad šiuo algoritmu randame optimalų uždavinio sprendinį. Norėdami tai įrodyti dažniausiai naudojame tokią schemą:

1. Įrodome, kad ir po godžiojo pasirinkimo pirmajame žingsnyje liko galimybė rasti optimalų viso uždavinio sprendinį, t.y. būtina egzistuoja bent vienas sprendinys, kuris yra suderintas su pirmajame žingsnyje atliktu pasirinkimu.
2. Įrodome, kad atlikę pirmąjį žingsnį vėl gavome tokį patį optimizacijos uždavinį tik mažesnėje pasirinkimų aibėje. Tada algoritmo teisingumo įrodymą užbaigiame matematinės indukcijos būdu.

Įrodykime, kad godžiuoju algoritmu randame optimalų tolydaus kuprinės pildymo uždavinio sprendinį. Jei į kuprinę tilpo tik vertingiausias daiktas ar jo dalis, tai godžiojo algoritmo teisingumas yra akivaizdus. Todėl nagrinėkime atvejį, kai godžiojo algoritmo sprendinys yra toks:

$$\{(v_1, p_1), (v_2, p_2), \dots, (v_{m-1}, p_{m-1}), (w_m, p_m)\}, \quad w_m \leq v_m, \quad m \geq 2..$$

a) Pirmajame godžiojo žingsnyje pasirinkome santykinai vertingiausią daiktą (v_1, p_1) . Tarkime, kad optimaliame rinkinyje šio daikto kiekis yra $w_1 < v_1$, o viso rinkinio vertė P litų. Tada išėmę iš kuprinės

$$\Delta w_1 = \min(w_m, v_1 - w_1)$$

kiekį mažiausiai vertingo produkto ir įdėję tokį patį kiekį pirmojo produkto, gausime naują rinkinį, kurio vertė yra nemažesnė už ankstesnio, nes

$$(s_1 - s_m)\Delta w_1 \geq 0.$$

Matome, kad tada, kai dalies produktų santykinės vertės sutampa, gali egzistuoti ir keli uždavinio sprendiniai.

b) Po pirmojo žingsnio vėl gavome kuprinės užpildymo vertingiausiais produktais uždavinį, tik sumažėjo kuprinės tūris ($V - v_1$) ir trumpesniu tapo daiktų sąrašas. Remdamiesi matematinės indukcijos metodu tvirtiname, kad godžiuoju algoritmu radome optimalų sprendinį.