Algoritmų sudarymo metodai

Raimondas Čiegis

Matematinio modeliavimo katedra, e-paštas: rc@vgtu.lt

Rugsėjo 1 d., 2023

Dinaminio programavimo metodas

Pavadinimas skamba gana sudėtingai, bet šie raktiniai žodžiai klaidina – metodo esmė pakankamai paprasta ir juo remiantis pavyksta sukonstruoti labai efektyvius algoritmus.

Dinaminio programavimo metodas

Pavadinimas skamba gana sudėtingai, bet šie raktiniai žodžiai klaidina – metodo esmė pakankamai paprasta ir juo remiantis pavyksta sukonstruoti labai efektyvius algoritmus.

Aptarkime, kodėl variantų perrinkimo algoritmai dažnai yra neefektyvūs. Taip atsitinka ne tik todėl, kad variantų skaičius yra labai didelis, bet ir todėl, kad skirtingų variantų yra daug mažiau nei generuojame spręsdami uždavinį. Tie patys variantai yra tikrinami/perskaičiuojami daug kartų.

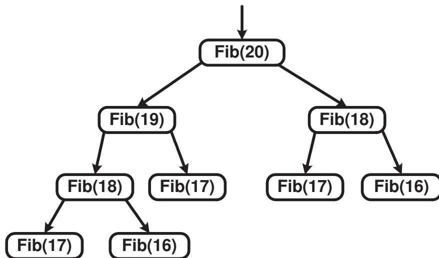
Prisiminkime rekursinį Fibonačio skaičių algoritmą

$$F(n) = F(n-1) + F(n-2).$$

Prisiminkime rekursinį Fibonačio skaičių algoritmą

$$F(n) = F(n-1) + F(n-2).$$

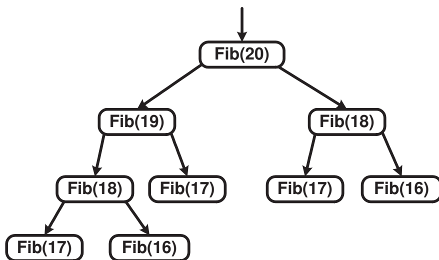
Paveiksle pavaizduota rekursinio algoritmo vykdymo eiga, kai $n = 20$.



Prisiminkime rekursinį Fibonačio skaičių algoritmą

$$F(n) = F(n-1) + F(n-2).$$

Paveiksle pavaizduota rekursinio algoritmo vykdymo eiga, kai $n = 20$.



Variantai kartojasi, todėl skaičiavimų apimtis greitai didėja.

Toks Fibonačio skaičių radimo algoritmas atitinka principą "iš viršaus į apačią" – pradedame nuo didžiausio argumento $F(n)$ užduoties ir ją skaičiuojame išreikšdami per dvi mažesnio argumento užduotis.

Toks Fibonačio skaičių radimo algoritmas atitinka principą "iš viršaus į apačią" – pradedame nuo didžiausio argumento $F(n)$ užduoties ir ją skaičiuojame išreikšdami per dvi mažesnio argumento užduotis.

Algoritmo vykdymą galime smarkiai pagreitinti pasitelkdami tarpinių rezultatų išaugojimo strategiją. Aišku, tada turime naudoti papildomus atminties resursus.

Toks Fibonačio skaičių radimo algoritmas atitinka principą "iš viršaus į apačią" – pradedame nuo didžiausio argumento $F(n)$ užduoties ir ją skaičiuojame išreikšdami per dvi mažesnio argumento užduotis.

Algoritmo vykdymą galime smarkiai pagreitinti pasitelkdami tarpinių rezultatų išaugojimo strategiją. Aišku, tada turime naudoti papildomus atminties resursus.

Kiekvieną kartą, kai suskaičiuojame kurią nors reikšmę $F(m)$, ją įsimename ir kai, vykdant algoritmą, kitą kartą tenka naudoti $F(m)$ reikšmę, ją tiesiog skaitome iš saugyklos. Taigi kiekvienas uždavinys yra sprendžiamas tik vieną kartą.

Toks Fibonačio skaičių radimo algoritmas atitinka principą "iš viršaus ¿ apačią" – pradedame nuo didžiausio argumento $F(n)$ užduoties ir ją skaičiuojame išreikšdami per dvi mažesnio argumento užduotis.

Algoritmo vykdymą galime smarkiai pagreitinti pasitelkdami tarpinių rezultatų išaugojimo strategiją. Aišku, tada turime naudoti papildomus atminties resursus.

Kiekvieną kartą, kai suskaičiuojame kurią nors reikšmę $F(m)$, ją ¿simename ir kai, vykdant algoritmą, kitą kartą tenka naudoti $F(m)$ reikšmę, ją tiesiog skaitome iš saugyklos. Taigi kiekvienas uždavinys yra sprendžiamas tik vieną kartą.

Per pratybas palyginkite abiejų algoritmo variantų vykdymo laikus. Taip pat realizuokite ir iteracin¿ algoritmą, kur¿ apibūdina principas "iš apačios ¿ viršų".

Šio trūkumo neturi dinaminio programavimo metodas. Jis taikytinas tada, kai tenkinamos tokios sąlygos:

- Algoritmo vykdymo metu generuojamos užduočių aibės esmingai persidengia, todėl daug kartų sprendžiame tas pačias užduotis. Dinaminio programavimo metode įsimename jau spręstų užduočių sprendinius ir sprendžiame tik naujus uždavinius.

Šio trūkumo neturi dinaminio programavimo metodas. Jis taikytinas tada, kai tenkinamos tokios sąlygos:

- Algoritmo vykdymo metu generuojamos užduočių aibės esmingai persidengia, todėl daug kartų sprendžiame tas pačias užduotis. Dinaminio programavimo metode įsimename jau spręstų užduočių sprendinius ir sprendžiame tik naujus uždavinius.
- Svarbiausia yra antroji, Belmano sąlygą, kad optimalus viso uždavinio sprendinys yra sudarytas iš tokio pačio tipo mažesnių užduočių optimalių sprendinių. Šią sąlygą užrašome rekurentinės lygybės forma, ji smarkiai sumažina nagrinėjamų variantų skaičių.

Skaldyk ir valdyk algoritme užduotys generuojamos iš viršaus ¿ apačią, t. y. pradinis uždavinys skaidomas ¿ kelias mažesnes užduotis, kurios toliau dalijamos ¿ mažesnes.

Skaldyk ir valdyk algoritme užduotys generuojamos iš viršaus ¿ apačią, t. y. pradinis uždavinys skaidomas ¿ kelias mažesnes užduotis, kurios toliau dalijamos ¿ mažesnes.

Dinaminio programavimo metode uždavin¿ pradedame spręsti nuo mažiausių ir lengvai išsprendžiamų užduočių, jų rezultatus naudojame spręsdami didesnes užduotis ir taip surandame viso uždavinio sprendin¿.

Skaldyk ir valdyk algoritme užduotys generuojamos iš viršaus ¿ apačią, t. y. pradinis uždavinys skaidomas ¿ kelias mažesnes užduotis, kurios toliau dalijamos ¿ mažesnes.

Dinaminio programavimo metode uždavin¿ pradedame spręsti nuo mažiausių ir lengvai išsprendžiamų užduočių, jų rezultatus naudojame spręsdami didesnes užduotis ir taip surandame viso uždavinio sprendin¿.

Realizuodami dinaminio programavimo metodą nagrinėjame tik tuos variantus, kurių gali prireikti optimaliai strategijai sudaryti (Belmano sąlyga).

Matricų daugybos eiliškumas

Reikia sudauginti n matricų $A_1 A_2 \cdots A_n$, čia A_i yra $p_{i-1} \times p_i$ dydžio matrica.

Matricų daugybos eiliškumas

Reikia sudauginti n matricų $A_1 A_2 \cdots A_n$, čia A_i yra $p_{i-1} \times p_i$ dydžio matrica.

Matricų daugyba sudaro pagrindą daugelio algoritmų, kurie naudojami kompiuterinėje animacijoje, virtualios realybės modeliuose, dizaine.

Matricų daugybos eiliškumas

Reikia sudauginti n matricų $A_1 A_2 \cdots A_n$, čia A_i yra $p_{i-1} \times p_i$ dydžio matrica.

Matricų daugyba sudaro pagrindą daugelio algoritmų, kurie naudojami kompiuterinėje animacijoje, virtualios realybės modeliuose, dizaine.

Nors galutinis rezultatas nepriklauso nuo matricų dauginimo tvarkos, atliekamų veiksmų skaičius gali labai smarkiai skirtis.

Priminsime, kad, daugindami $n \times m$ ir $m \times k$ dydžio matricas AB atliekame $2nmk$ aritmetinių veiksmų.

Nagrinėkime pavyzdį, kai dauginame 10×200, 200×4 ir 4×80 dydžio matricas $A_1 A_2 A_3$.

Priminsime, kad, daugindami $n \times m$ ir $m \times k$ dydžio matricas AB atliekame $2nmk$ aritmetinių veiksmų.

Nagrinėkime pavyzdį, kai dauginame 10×200, 200×4 ir 4×80 dydžio matricas $A_1 A_2 A_3$.

Šią užduotį galime įvykdyti dviem skirtingais būdais:

Priminsime, kad, daugindami $n \times m$ ir $m \times k$ dydžio matricas AB atliekame $2nmk$ aritmetinių veiksmų.

Nagrinėkime pavyzdį, kai dauginame 10×200, 200×4 ir 4×80 dydžio matricas $A_1 A_2 A_3$.

Šią užduotį galime įvykdyti dviem skirtingais būdais:

1. $(A_1 A_2) A_3$, tada atliekame

$$2x10x200x4 + 2x10x4x80 = 16000 + 6400 = 22400$$

veiksmų,

Priminsime, kad, daugindami $n \times m$ ir $m \times k$ dydžio matricas AB atliekame $2nmk$ aritmetinių veiksmų.

Nagrinėkime pavyzdį, kai dauginame 10×200, 200×4 ir 4×80 dydžio matricas $A_1 A_2 A_3$.

Šią užduotį galime įvykdyti dviem skirtingais būdais:

1. $(A_1 A_2) A_3$, tada atliekame

$$2x10x200x4 + 2x10x4x80 = 16000 + 6400 = 22400$$

veiksmų,

2. $A_1 (A_2 A_3)$, tada atliekame

$$2x200x4x80 + 2x10x200x80 = 128000 + 320000 = 448000$$

veiksmų, t. y. antruoju būdu atliekame dvidešimt kartų daugiau veiksmų.

Dabar šį uždavinį spręsime dinaminio programavimo metodu.

Dabar šį uždavinį spręsime dinaminio programavimo metodu.

Pirmiausia reikia rasti sprendinio optimalumo sąlygą ir įsitikinti, kad viso uždavinio optimalų sprendinį galime sudaryti naudodami mažesnių užduočių optimalius sprendinius.

Dabar šį uždavinį spręsime dinaminio programavimo metodu.

Pirmiausia reikia rasti sprendinio optimalumo sąlygą ir įsitikinti, kad viso uždavinio optimalų sprendinį galime sudaryti naudodami mažesnių užduočių optimalius sprendinius.

Pažymėkime mažesnio skaičiaus iš eilės einančių matricų sandaugos rezultatą

$$B_{i,j} = A_i A_{i+1} \cdots A_j.$$

Tai irgi matrica, jos dydis $p_{i-1} \times p_j$.

Dabar šį uždavinį spręsime dinaminio programavimo metodu.

Pirmiausia reikia rasti sprendinio optimalumo sąlygą ir įsitikinti, kad viso uždavinio optimalų sprendinį galime sudaryti naudodami mažesnių užduočių optimalius sprendinius.

Pažymėkime mažesnio skaičiaus iš eilės einančių matricų sandaugos rezultatą

$$B_{i,j} = A_i A_{i+1} \cdots A_j.$$

Tai irgi matrica, jos dydis $p_{i-1} \times p_j$.

Tada optimalią sandaugos skaičiavimo tvarką apibrėžiame lygybe

$$A_1 A_2 \cdots A_n = B_{1,k} B_{k+1,n}.$$

Dabar šį uždavinį spręsime dinaminio programavimo metodu.

Pirmiausia reikia rasti sprendinio optimalumo sąlygą ir įsitikinti, kad viso uždavinio optimalų sprendinį galime sudaryti naudodami mažesnių užduočių optimalius sprendinius.

Pažymėkime mažesnio skaičiaus iš eilės einančių matricų sandaugos rezultatą

$$B_{i,j} = A_i A_{i+1} \cdots A_j.$$

Tai irgi matrica, jos dydis $p_{i-1} \times p_j$.

Tada optimalią sandaugos skaičiavimo tvarką apibrėžiame lygybe

$$A_1 A_2 \cdots A_n = B_{1,k} B_{k+1,n}.$$

Paskutiniame algoritmo žingsnyje, daugindami matricas $B_{1,k}$ ir $B_{k+1,n}$ atliekame $2p_0 p_k p_n$ aritmetinius veiksmus.

Prieš tai reikėjo apskaičiuoti pačias matricas $B_{1,\,k}$ ir $B_{k+1,\,n}$, šias sandaugas vėl skaičiuojame optimaliu būdu.

Prieš tai reikėjo apskaičiuoti pačias matricas $B_{1,\,k}$ ir $B_{k+1,\,n}$, šias sandaugas vėl skaičiuojame optimaliu būdu.

Taigi parodėme, kad n matricų sandaugos optimalus skaičiavimas suvedamas į dviejų trumpesnių matricų sekų sandaugos skaičiavimo uždavinį.

Prieš tai reikėjo apskaičiuoti pačias matricas $B_{1,\,k}$ ir $B_{k+1,\,n}$, šias sandaugas vėl skaičiuojame optimaliu būdu.

Taigi parodėme, kad n matricų sandaugos optimalus skaičiavimas suvedamas į dviejų trumpesnių matricų sekų sandaugos skaičiavimo uždavinį.

Sudarysime lygtį, apibrėžiančią optimalų uždavinio sprendinį. Pažymėkime $m(i,j)$ aritmetinių veiksmų skačių optimaliu būdu dauginant matricas $A_i A_{i+1} \cdots A_j$.

Remiantis pateikta analize, egzistuoja toks k, kad šių matricų sandaugą išskaidome į dviejų matricų $B_{i,\,k}$ ir $B_{k+1,\,j}$, kurios irgi skaičiuojamos optimaliu algoritmu, sandaugą

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

Remiantis pateikta analize, egzistuoja toks k, kad šių matricų sandaugą išskaidome į dviejų matricų $B_{i,k}$ ir $B_{k+1,j}$, kurios irgi skaičiuojamos optimaliu algoritmu, sandaugą

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

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

$$m(i,j) = \begin{cases} 0, & i = j, \\ \min\limits_{i \le k < j} \big(m(i,k) + m(k+1,j) + 2p_{i-1}p_kp_j\big), & i < j. \end{cases}$$

Kiekvienam elementui $m(i,j)$ optimalaus indekso k reikšmę saugome P matricos $p(i,j)$ elemente.

Skaičius $m(1, n)$ ir apibrėžia n matricų $A_1 A_2 \cdots A_n$ daugybos optimalaus algoritmo sąnaudas.

Optimali šešių matricų sandaugos tvarka.

Skaičiuokime šešių matricų sandaugą

$$A_1 A_2 A_3 A_4 A_5 A_6 ,$$

kai šių matricų dydžiai yra tokie: 40×50, 50×20, 20×4, 4×15, 15×25, 25×35.

Optimali šešių matricų sandaugos tvarka.

Skaičiuokime šešių matricų sandaugą

$$A_1 A_2 A_3 A_4 A_5 A_6,$$

kai šių matricų dydžiai yra tokie: 40×50, 50×20, 20×4, 4×15, 15×25, 25×35.

Vykdydami algoritmą paeiliui skaičiuojame matricų M ir 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) matrica M

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

b) matrica P

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

a) matrica M

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

b) matrica P

Matome, kad matricas reikia dauginti taip:

$$\big(A_1(A_2A_3)\big)\big((A_4A_5)A_6\big),$$

tada atliksime tik 45 200 aritmetinius veiksmus.

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

Matrica M

Suskaičiuokime elementą $m(2,4)$. Matricų A_2, A_3, A_4 dydžiai $50x20$, $20x4$, $4x15$.

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

Matrica M

Suskaičiuokime elementą $m(2,4)$. Matricų A_2, A_3, A_4 dydžiai $50x20$, $20x4$, $4x15$.
Imkime $k = 2$:

$$m(2,2) + m(3,4) + 2 \cdot 50 \cdot 20 \cdot 15 = 0 + 2400 + 30000 = 32400,$$

kitas variantas $k = 3$:

$$m(2,3) + m(4,4) + 2 \cdot 50 \cdot 4 \cdot 15 = 8000 + 0 + 6000 = 14000.$$

Mažiausia reikšmė gaunama, kai $k = 3$, dauginimo tvarka $(A_2 A_3) A_4$.

Godieji algoritmai

Dažnai patenkame į tokias situacijas, kai reikia priimti sprendimą "čia ir dabar". Šis pasirinkimas paveiks ir ateities rezultatus, pvz. firmos pelną ar kelionės trukmę. Bet įvertinti visas šio žingsnio pasekmes kitais algoritmais (skaldyk ir valdyk, dinaminio programavimo ar pilnu variantų perrinkimu) neturime galimybių (tai užtruktų per daug ilgai).

Godieji algoritmai

Dažnai patenkame į tokias situacijas, kai reikia priimti sprendimą "čia ir dabar". Šis pasirinkimas paveiks ir ateities rezultatus, pvz. firmos pelną ar kelionės trukmę. Bet įvertinti visas šio žingsnio pasekmes kitais algoritmais (skaldyk ir valdyk, dinaminio programavimo ar pilnu variantų perrinkimu) neturime galimybių (tai užtruktų per daug ilgai).

Tokių uždavinių sprendinio paiešką dažniausiai išskaidome į n etapų ir kiekvienu žingsniu renkamės iš nedidelio baigtinio skaičiaus variantų m (peržiūrime tik mažą dalį visų galimų variantų).

Godieji algoritmai rekomenduoja rinktis lokaliai geriausią variantą duotojo žingsnio metu. Tada lokalaus pasirinkimo sudėtingumas – m, o viso algoritmo sudėtingumas – tik nm veiksmų.

Godieji algoritmai rekomenduoja rinktis lokaliai geriausią variantą duotojo žingsnio metu. Tada lokalaus pasirinkimo sudėtingumas – m, o viso algoritmo sudėtingumas – tik nm veiksmų.

Aišku, dažniausiai negalime garantuoti, jog, taip spręsdami uždavinį, radome globaliai geriausią sprendinį. Tokie godieji algoritmai yra euristikos, leidžiančios labai sparčiai apskaičiuoti tikslaus sprendinio artinius.

Godieji algoritmai rekomenduoja rinktis lokaliai geriausią variantą duotojo žingsnio metu. Tada lokalaus pasirinkimo sudėtingumas – m, o viso algoritmo sudėtingumas – tik nm veiksmų.

Aišku, dažniausiai negalime garantuoti, jog, taip spręsdami uždavinį, radome globaliai geriausią sprendinį. Tokie godieji algoritmai yra euristikos, leidžiančios labai sparčiai apskaičiuoti tikslaus sprendinio artinius.

Visgi egzistuoja daug svarbių taikomųjų uždavinių, kai godieji algoritmai apibrėžia tikslų sprendinį. Tokius pavyzdžius nagrinėsime spręsdami grafų teorijos uždavinius.

Grąžos atidavimo uždavinys.

Automatas grąžą atiduoda monetomis, kurių nominalai

$$V_1 > V_2 > \ldots > V_m.$$

Grąžos atidavimo uždavinys.

Automatas grąžą atiduoda monetomis, kurių nominalai

$$V_1 > V_2 > \ldots > V_m.$$

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

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

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

Tarkime, kad $V_m = 1$, tada visada galime parinkti bent vieną monetų kombinaciją, kurios suma lygi G.

Tarkime, kad $V_m = 1$, tada visada galime parinkti bent vieną monetų kombinaciją, kurios suma lygi G.

Godžiojo grąžos atidavimo algoritmo idėja labai paprasta: pirmiausia stengiamės grąžą atiduoti didžiausio nominalo monetomis, tokių monetų skaičius $n_1 = \lfloor G/V_1 \rfloor$, paskui 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.

Tarkime, kad $V_m = 1$, tada visada galime parinkti bent vieną monetų kombinaciją, kurios suma lygi G.

Godžiojo grąžos atidavimo algoritmo idėja labai paprasta: pirmiausia stengiamės grąžą atiduoti didžiausio nominalo monetomis, tokių monetų skaičius $n_1 = \lfloor G/V_1 \rfloor$, paskui 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 kuriuo nors algoritmo žingsniu $G_j < V_{j+1}$, 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 grąžą reikia atiduoti taip

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

taigi naudojame penkias monetas. Nesunku patikrinti, kad toks sprendinys yra optimalus (patikrinkite).

Imkime monetų rinkinį

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

Godžiuoju algoritmu apskaičiuojame, kad 63 centų grąžą reikia atiduoti taip

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

taigi naudojame penkias monetas. Nesunku patikrinti, kad toks sprendinys yra optimalus (patikrinkite).

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

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

t. y. vėl naudojame penkias monetas.

Imkime monetų rinkinį

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

Godžiuoju algoritmu apskaičiuojame, kad 63 centų grąžą reikia atiduoti taip

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

taigi naudojame penkias monetas. Nesunku patikrinti, kad toks sprendinys yra optimalus (patikrinkite).

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

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

t. y. vėl naudojame penkias monetas.

Tačiau egzistuoja ir geresnis sprendinys, kai klientui atiduome tris penkių centų monetas – $15 = 3 \times 5$.

Diskretusis kuprinės užpildymo uždavinys

Turime n daiktų, kurių tūriai yra $v_1, v_2, \ldots, v_n$, o kaina $p_1, p_2, \ldots, p_n$. Reikia rasti tokį daiktų rinkinį, kuris tilptų į V tūrio kuprinę, o daiktų vertė būtų didžiausia.

Diskretusis kuprinės užpildymo uždavinys

Turime n daiktų, kurių tūriai yra $v_1, v_2, \ldots, v_n$, o kaina $p_1, p_2, \ldots, p_n$. Reikia rasti tokį daiktų rinkinį, kuris tilptų į V tūrio kuprinę, o daiktų vertė būtų didžiausia.

Sprendžiame optimizavimo uždavinį:

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

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

Optimizavimo parametrai n_j gali būti lygūs tik vienetui arba nuliui.

Pirmiausia apibrėžiame santykinę kiekvieno daikto vertę $s_j = p_j / v_j$. Visus daiktus rūšiuojame šios vertės mažėjimo tvarka, tarsime, kad

$$s_1 \geq s_2 \geq \ldots \geq s_n.$$

Pirmiausia apibrėžiame santykinę kiekvieno daikto vertę $s_j = p_j / v_j$. Visus daiktus rūšiuojame šios vertės mažėjimo tvarka, tarsime, kad

$$s_1 \geq s_2 \geq \ldots \geq s_n.$$

Godžioji strategija – kuprinę stengiamės užpildyti didžiausios santykinės vertės daiktais.

Juos ¡mame vieną po kito ir tikriname, ar daiktas dar telpa ¿ kuprinę, jei ne – ¡mame kitą daiktą.

Turime aštuonis daiktus, kuriuos žymėsime (v_j, p_j):

$$(25, 50), \quad (20, 80), \quad (20, 50), \quad (15, 45),$$
$$(30, 105), \quad (35, 35), \quad (20, 10), \quad (10, 45).$$

Turime aštuonis daiktus, kuriuos žymėsime (v_j, p_j):

$$(25, 50), \quad (20, 80), \quad (20, 50), \quad (15, 45),$$
$$(30, 105), \quad (35, 35), \quad (20, 10), \quad (10, 45).$$

Apskaičiuojame jų santykines vertes

$$S = \{\, 2, \quad 4, \quad 2.5, \quad 3, \quad 3.5, \quad 1, \quad 0.5, \quad 4.5 \,\}$$

ir surūšiuojame daiktus verčių didėjimo tvarka

$$(10, 45), \quad (20, 80), \quad (30, 105), \quad (15, 45),$$
$$(20, 50), \quad (25, 50), \quad (35, 35), \quad (20, 10).$$

Turime aštuonis daiktus, kuriuos žymėsime (v_j, p_j):

$$(25, 50), \quad (20, 80), \quad (20, 50), \quad (15, 45),$$
$$(30, 105), \quad (35, 35), \quad (20, 10), \quad (10, 45).$$

Apskaičiuojame jų santykines vertes

$$S = \{\, 2, \quad 4, \quad 2.5, \quad 3, \quad 3.5, \quad 1, \quad 0.5, \quad 4.5 \,\}$$

ir surūšiuojame daiktus verčių didėjimo tvarka

$$(10, 45), \quad (20, 80), \quad (30, 105), \quad (15, 45),$$
$$(20, 50), \quad (25, 50), \quad (35, 35), \quad (20, 10).$$

Imkime kuprinę, kurios tūris $V = 80$. Naudodami godųjį algoritmą į ją įdedame pirmuosius keturis daiktus, jų bendras tūris – 75, o vertė – 275.

$$(10, 45), \quad (20, 80), \quad (30, 105), \quad (15, 45),$$
$$(20, 50), \quad (25, 50), \quad (35, 35), \quad (20, 10).$$

Tai nėra geriausias sprendinys, nes, imdami pirmuosius tris ir penktąjį daiktus, užpildome visą kuprinę, o tokio krovinio vertė – 280.

$$(10, 45),\quad (20, 80),\quad (30, 105),\quad (15, 45),$$
$$(20, 50),\quad (25, 50),\quad (35, 35),\quad (20, 10).$$

Tai nėra geriausias sprendinys, nes, imdami pirmuosius tris ir penktąjį daiktus, užpildome visą kuprinę, o tokio krovinio vertė – 280.

Taigi šiam uždaviniui godusis algoritmas yra tik euristika.

Jau susipažinome su pagrindiniais metodais, kurie padeda sudaryti efektyvius algoritmus:
variantų perrinkimas, rekursija, skaldyk ir valdyk,
dinaminis programavimas, godieji algoritmai.

Jau susipažinome su pagrindiniais metodais, kurie padeda sudaryti efektyvius algoritmus:
variantų perrinkimas, rekursija, skaldyk ir valdyk,
dinaminis programavimas, godieji algoritmai.

Juos ir naudosime viso algoritmų kurso metu, detaliai susipažinsime su jų galimybėmis.

Dabar panagrinėsime paprastą, bet informatyvų pavyzdį, kurio sprendimui galime sudaryti įvairius algoritmus. Juos sukursime imdami vieną ar kitą bendrą metodą.

Turime vienmatį masyvą A, kurio elementai $A[i]$, $i = 1, \ldots, n$ yra teigiami sveikieji skaičiai.

Turime vienmatį masyvą A, kurio elementai $A[i]$, $i = 1, \ldots, n$ yra teigiami sveikieji skaičiai.

Sakysime, kad elementas $A[j]$ apibrėžia piką, jei

$$A[j-1] \le A[j] \le A[j+1], \quad 1 < j < n.$$

Atitinkamai patiksliname apibrėžimą masyvo galuose, pvz. $A[1]$ yra pikas, jei $A[1] \ge A[2]$.

Taigi, elementas yra pikas, jeigu jo kaimynai yra nedidesni. Matome, kad užtenka patikrinti tik lokalią informaciją.

Turime vienmatį masyvą A, kurio elementai $A[i]$, $i = 1, \dots, n$ yra teigiami sveikieji skaičiai.

Sakysime, kad elementas $A[j]$ apibrėžia piką, jei

$$A[j-1] \le A[j] \le A[j+1], \quad 1 < j < n.$$

Atitinkamai patiksliname apibrėžimą masyvo galuose, pvz. $A[1]$ yra pikas, jei $A[1] \ge A[2]$.

Taigi, elementas yra pikas, jeigu jo kaimynai yra nedidesni. Matome, kad užtenka patikrinti tik lokalią informaciją.

Mūsų uždavinys yra surasti kurį nors piką. Aišku, turime būti tikri, kad bent vienas toks elementas egzistuoja. Tai nesunku įrodyti (padarykite tai).

Algoritmas 1. Panaudokime variantų perrinkimo metodą. Paiešką pradėkime nuo $A[1]$ elemento, jeigu $A[2] > A[1]$, tai $A[1]$ nėra pikinis elementas.

Algoritmas 1. Panaudokime variantų perrinkimo metodą. Paiešką pradėkime nuo $A[1]$ elemento, jeigu $A[2] > A[1]$, tai $A[1]$ nėra pikinis elementas.

Tęskime paiešką tikrindami, ar $A[2] \geq A[3]$. Jeigu sąlyga vėl neišpildyta, tai tikriname elementą $A[3]$ ir šį procesą kartojame tol, kol surasime piką.

Algoritmas 1. Panaudokime variantų perrinkimo metodą. Paiešką pradėkime nuo $A[1]$ elemento, jeigu $A[2] > A[1]$, tai $A[1]$ nėra pikinis elementas.

Tęskime paiešką tikrindami, ar $A[2] \geq A[3]$. Jeigu sąlyga vėl neišpildyta, tai tikriname elementą $A[3]$ ir šį procesą kartojame tol, kol surasime piką.

Įvertinsime tokio algoritmo sudėtingumą. Tarkime, kad su vienoda tikimybe kiekvienas elementas gali būti pirmuoju piku.

Tada blogiausiu atveju turėsime patikrinti visus elementus $T_B(n) = n$, vidutiniu atveju sudėtingumo įvertis yra panašus $T_V(n) = \frac{1}{2}n$.

Algoritmas 1. Panaudokime variantų perrinkimo metodą. Paiešką pradėkime nuo $A[1]$ elemento, jeigu $A[2] > A[1]$, tai $A[1]$ nėra pikinis elementas.

Tęskime paiešką tikrindami, ar $A[2] \geq A[3]$. Jeigu sąlyga vėl neišpildyta, tai tikriname elementą $A[3]$ ir šį procesą kartojame tol, kol surasime piką.

Įvertinsime tokio algoritmo sudėtingumą. Tarkime, kad su vienoda tikimybe kiekvienas elementas gali būti pirmuoju piku.

Tada blogiausiu atveju turėsime patikrinti visus elementus $T_B(n) = n$, vidutiniu atveju sudėtingumo įvertis yra panašus $T_V(n) = \frac{1}{2}n$.

Ar galima šį uždavinį išspręsti greičiau?

Algoritmas 2. Panaudokime skaldyk ir valdyk metodą. Paiešką pradedame nuo vidurinio $A[n/2]$ elemento. Jeigu

$$A[n/2-1] \leq A[n/2] \leq A[n/2+1],$$

tai $A[n/2]$ yra pikinis elementas. Užteko atlikti du palyginimus.

Algoritmas 2. Panaudokime skaldyk ir valdyk metodą.
Paiešką pradedame nuo vidurinio $A[n/2]$ elemento. Jeigu

$$A[n/2-1] \leq A[n/2] \leq A[n/2+1],$$

tai $A[n/2]$ yra pikinis elementas. Užteko atlikti du palyginimus.

Priešingu atveju renkamės tą masyvo pusę, kurios kryptimi nelygybė buvo neteisinga. Jeigu turime dvi galimybes, tai pasirenkame, kurią nors vieną iš jų.

Algoritmas 2. Panaudokime skaldyk ir valdyk metodą. Paiešką pradedame nuo vidurinio $A[n/2]$ elemento. Jeigu

$$A[n/2-1] \leq A[n/2] \leq A[n/2+1],$$

tai $A[n/2]$ yra pikinis elementas. Užteko atlikti du palyginimus.

Priešingu atveju renkamės tą masyvo pusę, kurios kryptimi nelygybė buvo neteisinga. Jeigu turime dvi galimybes, tai pasirenkame, kurią nors vieną iš jų.

Taigi po pirmo algoritmo žingsnio masyvo elementų skaičius dvigubai sumažėjo. Pavyzdžiui dabar turime nagrinėti elementus $A[i]$, $i = 1, \ldots, n/2-1$.

Algoritmas 2. Panaudokime skaldyk ir valdyk metodą. Paiešką pradedame nuo vidurinio $A[n/2]$ elemento. Jeigu

$$A[n/2-1] \le A[n/2] \le A[n/2+1],$$

tai $A[n/2]$ yra pikinis elementas. Užteko atlikti du palyginimus.

Priešingu atveju renkamės tą masyvo pusę, kurios kryptimi nelygybė buvo neteisinga. Jeigu turime dvi galimybes, tai pasirenkame, kurią nors vieną iš jų.

Taigi po pirmo algoritmo žingsnio masyvo elementų skaičius dvigubai sumažėjo. Pavyzdžiui dabar turime nagrinėti elementus $A[i]$, $i = 1, \ldots, n/2-1$.

Vėl kartojame pagrindinį žingsnį, kol surandame pikinį elementą. Čia užtenka prisiminti, kad jei masyve yra tik vienas elementas, tai jis apibrėžia piką.

Įvertinsime naujojo algoritmo sudėtingumą. Kadangi po kiekvieno žingsnio elementų skaičius sumažėja dvigubai, tai užteks atlikti ne daugiau nei $\log n$ žingsnius. Taigi net ir blogiausio atvejo sudėtingumas yra

$$T_B(n) = 2 \log n.$$

Įvertinsime naujojo algoritmo sudėtingumą. Kadangi po kiekvieno žingsnio elementų skaičius sumažėja dvigubai, tai užteks atlikti ne daugiau nei $\log n$ žingsnius. Taigi net ir blogiausio atvejo sudėtingumas yra

$$T_B(n) = 2 \log n.$$

Imkime $n = 1000000$, tada $\log n = 20$, matome, kad antrasis algoritmas yra esmingai efektyvesnis.

Dabar apibendrinkime uždavinį ir nagrinėkime dvimatį masyvą (matricą) A, kurio dimensija $m \, x \, n$.

Dabar apibendrinkime uždavinį ir nagrinėkime dvimatį masyvą (matricą) A, kurio dimensija $m \times n$.

Masyvo piku vadinsime elementą (i, j), kurio kaimynai eilutėje ar stulpelyje yra nedidesni už šį elementą

$$A[i][j] \geq A[i-1][j], \;\; A[i][j] \geq A[i+1][j],$$
$$A[i][j] \geq A[i][j-1], \;\; A[i][j] \geq A[i][j+1].$$

Apibrėžimą atitinkamai modifikuojame pasienio taškuose.

Algoritmas 1. Panaudokime godžiąją strategiją. Pasirenkame bet kurį elementą (i, j), nuo kurio pradedame paiešką.

Jeigu tai nėra pikas, tai pereiname kito elemento – didžiausios reikšmės kaimyno.

Tokį procesą tęsiame, kol surandame elementą piką.

Algoritmas 1. Panaudokime godžiąją strategiją. Pasirenkame bet kurį elementą (i, j), nuo kurio pradedame paiešką.

Jeigu tai nėra pikas, tai pereiname kito elemento – didžiausios reikšmės kaimyno.

Tokį procesą tęsiame, kol surandame elementą piką.

Tokiu algoritmu visada randame sprendinį. Įrodykite teiginį patys, tai tikrai nesudėtingas pratimas.

Panagrinėkime pavyzdį, kai piko paiešką pradedame nuo elemento 12:

$$\begin{pmatrix} 13 & 11 & 9 & 22 \\ 14 & 13 & \color{red}{12} & 10 \\ 15 & 9 & 11 & 17 \\ 16 & 17 & 19 & 21 \end{pmatrix}.$$

Panagrinėkime pavyzdį, kai piko paiešką pradedame nuo elemento 12:

$$\begin{pmatrix} 13 & 11 & 9 & 22 \\ 14 & 13 & \color{red}{12} & 10 \\ 15 & 9 & 11 & 17 \\ 16 & 17 & 19 & 21 \end{pmatrix}.$$

Tada algoritmo eiga bus tokia

$$\begin{pmatrix} 13 & 11 & 9 & 22 \\ \color{blue}{14} & \color{blue}{13} & \color{red}{12} & 10 \\ \color{blue}{15} & 9 & 11 & 17 \\ \color{blue}{16} & \color{blue}{17} & \color{blue}{19} & \color{red}{21} \end{pmatrix}.$$

Panagrinėkime pavyzdį, kai piko paiešką pradedame nuo elemento 12:

$$\begin{pmatrix} 13 & 11 & 9 & 22 \\ 14 & 13 & \color{red}{12} & 10 \\ 15 & 9 & 11 & 17 \\ 16 & 17 & 19 & 21 \end{pmatrix}.$$

Tada algoritmo eiga bus tokia

$$\begin{pmatrix} 13 & 11 & 9 & 22 \\ \color{blue}{14} & \color{blue}{13} & \color{red}{12} & 10 \\ \color{blue}{15} & 9 & 11 & 17 \\ \color{blue}{16} & \color{blue}{17} & \color{blue}{19} & \color{red}{21} \end{pmatrix}.$$

Algoritmo logika paprasta, bet vykdymo kaštai (vidutiniu ir blogiausiu atveju) yra proporcingi $\Theta(n\,m)$, t.y. tenka tikrinti didžiąją dalį visų elementų.

Algoritmas 2. Panaudokime Skaldyk ir valdyk strategiją.

1. Pasirenkame vidurinį stulpelį $j = m/2$.

Algoritmas 2. Panaudokime Skaldyk ir valdyk strategiją.

1. Pasirenkame vidurinį stulpelį j= m/2.

2. Šiame stulpelyje surandame maksimalų elementą (i, j).

Algoritmas 2. Panaudokime Skaldyk ir valdyk strategiją.

1. Pasirenkame vidurinį stulpelį j= m/2.

2. Šiame stulpelyje surandame maksimalų elementą (i, j).

3. Palyginame (i, j) su kaimynais $(i, j-1)$ ir $(i, j+1)$. Jeigu abu kaimynai yra nedidesni, tai (i, j) yra pikas.

Algoritmas 2. Panaudokime Skaldyk ir valdyk strategiją.

1. Pasirenkame vidurinį stulpelį j= m/2.

2. Šiame stulpelyje surandame maksimalų elementą (i, j).

3. Palyginame (i, j) su kaimynais $(i, j-1)$ ir $(i, j+1)$. Jeigu abu kaimynai yra nedidesni, tai (i, j) yra pikas.

4. Priešingu atveju pasirenkame tą masyvo pusę, kuri atitinka didesnį elementą. Taigi stulpelių skaičius sumažėjo dvigubai. Naujame masyve vėl kartojame tą patį algoritmą.

Algoritmas 2. Panaudokime Skaldyk ir valdyk strategiją.

1. Pasirenkame vidurinį stulpelį j= m/2.

2. Šiame stulpelyje surandame maksimalų elementą (i, j).

3. Palyginame (i, j) su kaimynais $(i, j-1)$ ir $(i, j+1)$. Jeigu abu kaimynai yra nedidesni, tai (i, j) yra pikas.

4. Priešingu atveju pasirenkame tą masyvo pusę, kuri atitinka didesnį elementą. Taigi stulpelių skaičius sumažėjo dvigubai. Naujame masyve vėl kartojame tą patį algoritmą.

Jeigu lieka vienintelis stulpelis, tai jo didžiausias elementas yra pikas.

Nesunku įvertinti algoritmo sudėtingumą blogiausiu atveju, tai atliekame nagrinėdami gautosios rekursijos kaštus:

$$\begin{aligned} T(n,m) &= T(n,m/2) + \Theta(n) \\ &= T(n,m/4) + 2\Theta(n) \\ &\dots \\ &= T(n,1) + \log(m)\,\Theta(n) \\ &= \log(m)\,\Theta(n). \end{aligned}$$

Nesunku įvertinti algoritmo sudėtingumą blogiausiu atveju, tai atliekame nagrinėdami gautosios rekursijos kaštus:

$$
\begin{aligned}
T(n,m) &= T(n, m/2) + \Theta(n) \\
&= T(n, m/4) + 2\Theta(n) \\
&\dots \\
&= T(n,1) + \log(m)\,\Theta(n) \\
&= \log(m)\,\Theta(n).
\end{aligned}
$$

Kokią algoritmo modifikaciją rekomenduosite, jei $m \ll n$?