Algoritmų sudarymo metodai

Raimondas Čiegis

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

Rugsėjo 12 d., 2022

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.

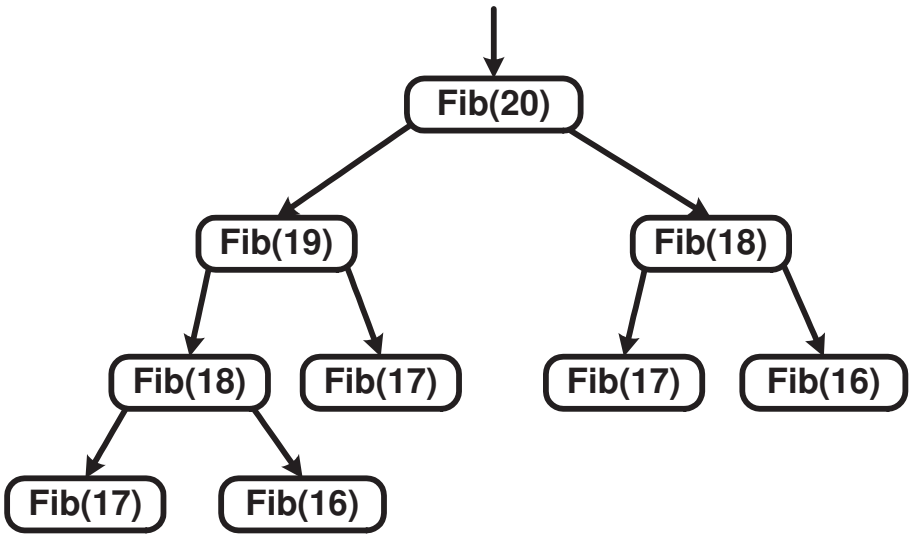
Prisiminkime rekursinį Fibonačio skaičių algoritmą

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

Prisiminkime rekursinį Fibonačio skaičių algoritmą

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

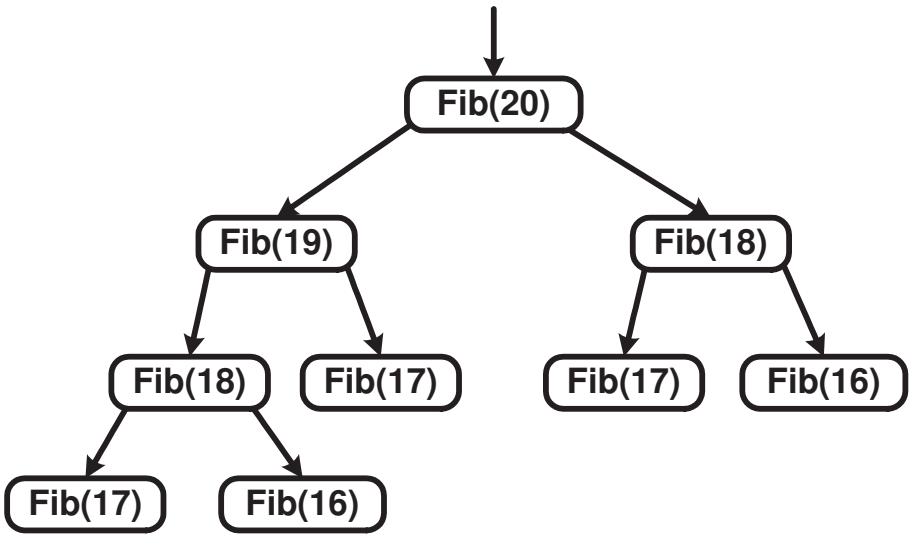
Paveiksle pavaizduota rekursinio algoritmo vykdymo eiga, kai $n = 20$.



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.

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.
- Uždavinys tenkina Belmano sąlygą, kad optimalus užduoties sprendinys yra sudarytas iš atskirų mažesnių užduočių optimalių sprendinių. Šią sąlygą užrašome rekurentinės lygybės forma, ji smarkiai sumažina nagrinėjamų variantų skaičių.

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

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

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

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

Dinaminio programavimo metode uždavinį pradedame 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 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 dvi $n \times m$ ir $m \times k$ dydžio matricas gauname $n \times k$ dydžio matricą $C = AB$

$$c_{ij} = \sum_{l=1}^{m} a_{il} b_{lj}, \quad 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant k\,.$$

atliekame $2nmk$ aritmetinių veiksmų.

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

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:

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 $16000 + 6400 = 22400$ 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 $16000 + 6400 = 22400$ veiksmų,
2. $A_1 (A_2 A_3)$, tada atliekame $128000 + 320000 = 448000$ 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 matricų sandaugos rezultatą (irgi matricą)

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

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 matricų sandaugos rezultatą (irgi matricą)

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

Tada optimalią sandaugos skaičiavimo tvarką apibrėžiame lygybe (parenkame pirmųjų skliaustų vietą k):

$$A_1 A_2 \cdots A_n = B_{1,k} B_{k+1,n} \equiv \left(A_1 A_2 \cdots A_k\right) \left(A_{k+1} A_2 \cdots A_n\right).$$

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 matricų sandaugos rezultatą (irgi matricą)

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

Tada optimalią sandaugos skaičiavimo tvarką apibrėžiame lygybe (parenkame pirmųjų skliaustų vietą k):

$$A_1 A_2 \cdots A_n = B_{1,\,k} B_{k+1,\,n} \equiv \left(A_1 A_2 \cdots A_k\right) \left(A_{k+1} A_2 \cdots A_n\right).$$

Paskutiniame algoritmo žingsnyje, daugindami dvi 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 mažesnių 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 mažesnių matricų sekų sandaugos skaičiavimo uždavinį.

Sudarysime lygtį, apibrėžiančią optimalų uždavinio sprendinį.

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

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 mažesnių matricų sekų sandaugos skaičiavimo uždavinį.

Sudarysime lygtį, apibrėžiančią optimalų uždavinio sprendinį.

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

Kadangi matricų išsidėstymo tvarkos keisti negalime, tai iš viso galime sudaryti tiek skirtingų variantų:

$$1 \leq i \leq n, \quad i \leq j \leq n.$$

Tokia M matrica yra apatinė trikampė.

Parodėme, kad optimaliu algoritmu skaičiuojant matricą B_{ij} egzistuoja toks k, kad šių matricų sandaugą išskaidome į dviejų matricų $B_{i,k}$ ir $B_{k+1,j}$ sandaugą

$$B_{ij} = B_{ik} B_{k+1,j}.$$

Patarosios matricos irgi skaičiuojamos optimaliu algoritmu. Taigi įvertiname aritmetinių veiksmų skaičių

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

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

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

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

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

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

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

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

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

Skaičius $m(1,n)$ ir apibrėžia n matricų daugybos $A_1 A_2 \cdots A_n$ 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 (paaiškinkite, kodėl tokia tvarka vykdomas algoritmas).

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

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

Matome, kad matricas reikia dauginti taip:

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

tada atliksime tik 45 200 aritmetinius veiksmus.

Skaičiuokime koeficiento $m(2,5)$ reikšmę (trečioji mėlynoji įstrižainė).

Skaičiuokime koeficiento $m(2,5)$ reikšmę (trečioji mėlynoji įstrižainė).

Randame tokį k, kuris atitinka mažiausią skaičiavimų apimtį

$$\min_{2\le k<5} \left(m(2,k) + m(k+1,5) + 2p_1 p_k p_5\right),$$

Skaičiuokime koeficiento $m(2,5)$ reikšmę (trečioji mėlynoji įstrižainė).

Randame tokį k, kuris atitinka mažiausią skaičiavimų apimtį

$$\min_{2\le k<5}\left(m(2,k)+m(k+1,5)+2p_1p_kp_5\right),$$

$k=2$:

$$M(2,2)+M(3,5)+2p_1p_2p_5 = 0+7000+2\cdot 50\cdot 20\cdot 25 = 57\,000,$$

$k=3$:

$$M(2,3)+M(4,5)+2p_1p_3p_5 = 8000+3000+2\cdot 50\cdot 4\cdot 25 = 21\,000,$$

$k=4$:

$$M(2,4)+M(5,5)+2p_1p_4p_5 = 14000+0+2\cdot 50\cdot 15\cdot 25 = 51\,500.$$

Matome, kad mažiausia reikšmė yra

$$M(2,5) = 21000,$$

ją gauname imdami $k = 3$, todėl

$$P(2,5) = 3.$$

Godieji algoritmai

Dažnai patenkame į tokias situacijas, kai reikia priimti sprendimą "čia ir dabar". Mūsų dabartinis 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". Mūsų dabartinis 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:

$$\max_{(n_1,\ldots,n_m)\in D}(n_1p_1 + n_2p_2 + \ldots + n_mp_m),$$

$$D = \{n_1v_1 + n_2v_2 + \ldots + n_mv_m \leq V, \quad n_j \in (0,1)\}.$$

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 įmame vieną po kito ir tikriname, ar daiktas dar telpa į kuprinę, jei ne – įmame 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.

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

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.