Je vračanje kovancev res preprosto?

ANDREJ TARANENKO

→ V vsakdanjem življenju se pogosto srečamo z vračanjem kovancev, pa naj bo to na avtomatu za kavo ali v trgovini. Običajno plačamo večji znesek, kot je potrebno, potem pa dobimo vrnjeno razliko. Vprašanje, ki si ga bomo zastavili je, kako dosežemo, da dobimo vrnjeno najmanjše možno število kovancev. Definirajmo problem bolj splošno.

Za podano množico vrednosti kovancev $K = \{k_1, \ldots, k_d\}$ želimo določiti, koliko kovancev potrebujemo, da izplačamo znesek, recimo znesek n. *Problem vračanja kovancev* zahteva, da za podani znesek izplačamo najmanjše možno število kovancev. V vseh situacijah v nadaljevanju predpostavimo, da imamo vedno na voljo dovolj kovancev vsake vrste.

Primer. Izplačati moramo znesek $n = 4$, na voljo imamo kovance naslednjih vrednosti $\{1, 2, 3\}$. Vse možne rešitve so naslednje: $\{1, 3\}$, $\{2, 2\}$, $\{1, 1, 2\}$ in $\{1, 1, 1, 1\}$. Vidimo lahko, da je najmanjše število kovancev, ki jih v tem primeru potrebujemo, enako 2.

Pri evrih imamo na voljo kovance naslednjih vrednosti (v centih): $\{1, 2, 5, 10, 20, 50, 100, 200\}$. Kako bi s temi kovanci vrnili naslednje zneske? Koliko kovancev bi vrnili vi?

- [25 centov] (rešitev: 20 + 5, dva kovanca)
- [98 centov] ?

Prodajalčev algoritem

Kako ste se v prejšnjem primeru odločili, katere kovance boste izbrali? Verjetno ste razmišljali na enak način, kot razmišlja prodajalec, ko vrača denar: na vsakem koraku izberemo kovanec najvišje vrednosti, ki ga še smemo izbrati.

Postopek računa za 98 centov, bi bil torej naslednji:

- Izberi kovanec za 50 centov. Ostane še 48 centov.
- Izberi kovanec za 20 centov. Ostane še 28 centov.
- Izberi kovanec za 20 centov. Ostane še 8 centov.
- Izberi kovanec za 5 centov. Ostanejo še 3 centi.
- Izberi kovanec za 2 centa. Ostane še 1 cent.
- Izberi kovanec za 1 cent. Znesek smo izplačali.

Na ta način smo torej vrnili šest kovancev.

Metoda, ki smo jo uporabili, se v računalništvu imenuje *požrešna metoda*. Za to metodo je značilno, da rešitev gradimo korakoma iz množice kandidatov, na vsakem koraku pa izberemo kandidata, ki trenutno največ doprinese k optimalni rešitvi.

S požrešno metodo pa ne dobimo nujno najboljše rešitve (v našem primeru najmanjše število vrnjenih kovancev). Poglejmo si primer, ko se to zgodi.

Primer. Recimo, da imamo na voljo kovance vrednosti $\{1, 3, 4, 5\}$. Vrniti želimo znesek $n = 7$.

S požrešno metodo bi prišli do naslednje rešitve:

- Izberemo kovanec vrednosti 5. Ostane še 2.
- Izberemo kovanec vrednosti 1. Ostane še 1.
- Izberemo kovanec vrednosti 1. Znesek smo izplačali.

S požrešno metodo smo izplačali tri kovance. Vendar obstaja boljša rešitev: izplačamo samo dva kovanca (kovanca vrednosti 4 in 3).

Tudi v splošnem velja, da moramo pri uporabi požrešne metode biti previdni, saj lahko na prvi pogled mislimo, da bomo dobili najboljšo možno rešitev, potem pa se izkaže, da temu ni tako. Zaradi tega je potrebno za vsak problem dokazati, ali je pristop s požrešno metodo resnično najboljši.

V zgornjem primeru, ko s požrešno metodo ne dobimo najboljše rešitve, smo spremenili množico vrednosti kovancev, glede na te, ki jih uporabljamo mi. Izkaže pa se, da za vrednosti kovancev, kot jih poznamo pri evrih ($\{1, 2, 5, 10, 20, 50, 100, 200\}$), s požrešno metodo vedno vrnemo najmanjše možno število kovancev. Poskusimo sedaj to tudi dokazati. Dokaz bomo naredili za nekaj vrednosti kovancev, saj so za ostale vrednosti ideje in razmišljanja enaka.

Označimo znesek, ki ga želimo izplačati, z n. Naj bo najboljša rešitev (z najmanjšim možnim številom kovancev) oblike

- $n = 200a + 100b + 50c + 20d + 10e + 5f + 2g + h.$

Očitno je $b \leq 1$, saj bi sicer lahko b zmanjšali za 2 in a povečali za 1; tako bi zmanjšali število vrnjenih kovancev. Iz podobnih razlogov velja tudi, da je $c \leq 1$, $d \leq 2$, $e \leq 1$, $f \leq 1$, $g \leq 2$ in $h \leq 1$. Še več, velja, da je $2g + h \leq 4$, sicer bi lahko zmanjšali število kovancev tako, da bi znesek 5 izplačali s kovancem z vrednostjo 5.

Zapišimo rešitev, ki jo dobimo s požrešnim algoritmom, kot

- $n = 200a' + 100b' + 50c' + 20d' + 10e' + 5f' + 2g' + h'.$

Ponovno lahko sklepamo, da je $b' \leq 1$, saj požrešna metoda, če je mogoče, izbere kovanec z vrednostjo 200. Iz podobnih razlogov dobimo: $c' \leq 1$, $d' \leq 2$, $e' \leq 1$, $f' \leq 1$, $g' \leq 2$, $h' \leq 1$ in $2g' + h' \leq 4$.

Ker je $n = 5\,(40a + 20b + 10c + 4d + 2e + f) + 2g + h$ in velja $2g + h \leq 4$, $g \leq 2$ in $h \leq 1$, dobimo pri deljenju števil n in $2g + h$ s številom 5 isti ostanek. To v matematiki zapišemo kot

- $n \equiv 2g + h \pmod 5$.

Na podoben način dobimo

- $n \equiv 2g' + h' \pmod 5$.

Zaradi $2g + h \leq 4$ in $2g' + h' \leq 4$ velja $g = g'$ in $h = h'$.

Označimo z $n_1 = (n - (2g + h))/5$. Število n_1 je naravno število, saj je $(n - (2g + h))$ deljivo s 5. Ponovno lahko dobimo

- $n_1 \equiv f \pmod 2$ in $n_1 \equiv f' \pmod 2$.

Ker je $f \leq 1$ in $f' \leq 1$, velja $f = f'$.

S podobnimi razmisleki pokažemo, da je $a = a'$, $b = b'$, $c = c'$, $d = d'$ in $e = e'$. Torej s požrešno metodo za evrske kovance vrnemo vedno najmanjše možno število kovancev.

Pravilna rešitev v vsakem primeru

Kako bi izračunali optimalno rešitev za primer s poljubnimi vrednostmi kovancev? Eden od načinov je, da se problema lotimo s pristopom *deli in vladaj*. Ta pristop k reševanju problemov je bil v Preseku že opisan (Presek 4, 2009/2010). Bistvo pristopa je, da nalogo danega problema razdelimo na manjše naloge istega problema, ki jih običajno rešimo rekurzivno.

Primer. Recimo, da želimo s kovanci vrednosti $\{1, 2, 5, 10, 20\}$ (v centih) vrniti 30 centov. Seveda bi radi ponovno vrnili najmanjše možno število kovancev. Označimo z minKovancev(n) najmanjše število kovancev za izplačano vrednost n. Iščemo torej minKovancev(30). Najmanjše možno število kovancev izračunamo tako, da izberemo najmanjšo izmed naslednjih rešitev:

- 1 + minKovancev(29) (če je v rešitvi kovanec vrednosti 1),
- 1 + minKovancev(28) (če je v rešitvi kovanec vrednosti 2),
- 1 + minKovancev(25) (če je v rešitvi kovanec vrednosti 5),
- 1 + minKovancev(20) (če je v rešitvi kovanec vrednosti 10) in
- 1 + minKovancev(10) (če je v rešitvi kovanec vrednosti 20).

Vrednosti minKovancev(29), minKovancev(28), minKovancev(25), minKovancev(20) in minKovancev(10) izračunamo na enak način, torej rekurzivno. Ustavitveni pogoj rekurzivnega klica je znesek 0, saj zanj ne vrnemo nobenega kovanca.

Kako smo v prejšnjem primeru razmišljali? Naj bo n znesek, ki ga vračamo. Predpostavimo, da izberemo kovanec vrednosti k, torej smo vrnili en kovanec, moramo jih še vrniti toliko, kot jih vrnemo za znesek $n - k$. Za znesek 0 je rešitev, da ne vrnemo nobenega kovanca. Postopek bi lahko zapisali, kot je prikazano v algoritmu 1.

Algoritem 1: minKovancev(n)

Vhod: znesek n, vrednosti kovancev *kovanci*
Izhod: najmanjše število kovancev za izplačilo zneska n

```
1 if n = 0 then
2 |   return 0
3 end
4 v ← ∞
5 for all k iz množice kovanci, kjer
  je k ≤ n do
6 |   v ← min{v, minKovanci(n − k) + 1}
7 end
8 return v
```

Rekurzivna rešitev seveda ni učinkovita, saj pogosto večkrat računa rešitev za iste vrednosti zneskov. Temu se lahko izognemo tako, da si rešitve za že izračunane vrednosti shranimo. Algoritem najprej preveri, če je želena rešitev že bila izračunana. Če je bila, uporabi shranjeno vrednost, sicer jo izračuna. Torej v algoritmu potrebujemo polje, v katero si za vse vrednosti od 0 do n shranimo rešitev, ko je le-ta znana. Na ta način dobimo učinkovitejši algoritem, ki je zapisan z algoritmom 2. V algoritmu 2 predpostavimo, da je vrednost shranjeneVrednosti[0] = 0.

Preverite sami, da za primer zneska 7 in vrednosti kovancev $\{1, 3, 4, 5\}$ dobimo optimalno rešitev, ki je pri požrešni metodi nismo dobili. Razmislite tudi, kako bi prilagodili predstavljeno rešitev, če bi že-

Algoritem 1: minKovancev(n)

Vhod: znesek n, vrednosti kovancev *kovanci*
Izhod: najmanjše število kovancev za izplačilo zneska n

```
1 if shranjeneVrednosti[n] ni prazno then
2 |   return shranjeneVrednosti[n]
3 end
4 v ← ∞
5 for all k iz množice kovanci, kjer je k ≤ n
6 |   v ← min{v, minKovanci(n − k) + 1}
7 end
8 shranjeneVrednosti[n] = v
9 return v
```

leli poleg števila vrnjenih kovancev vedeti še, katere kovance izberemo. Predstavljeni problem lahko še bolj zaostrimo, saj smo do sedaj predpostavljali, da imamo kovancev vsake vrednosti dovolj na zalogi. Razmislite, kako bi se spremenil algoritem, če imeli podatek, koliko kovancev posamezne vrednosti imamo na razpolago.

×××

↓↓↓

REŠITEV POIŠČI MINE S STRANI 25

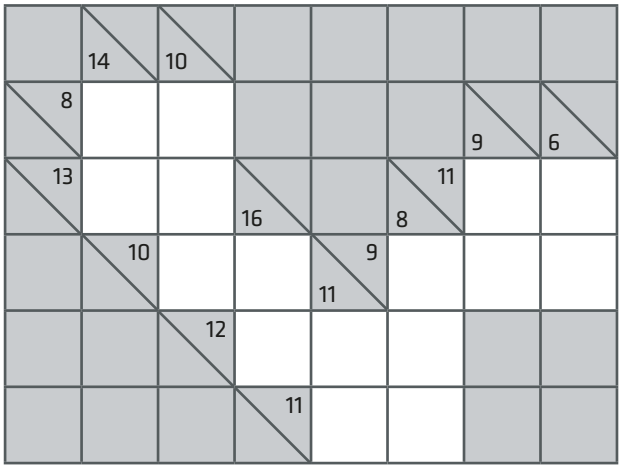
1	**M**	2	**M**	2
2	**M**			**M**
M	4	**M**	2	1
	M		**M**	1
0		0		0

×××

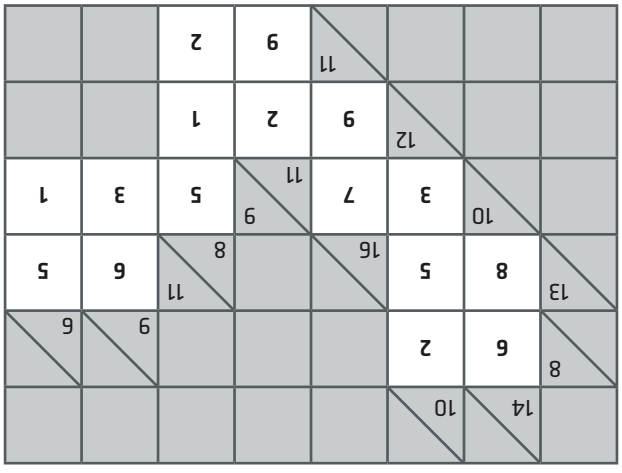
Križne vsote

↓↓↓

→ Naloga reševalca je, da izpolni bele kvadratke s števkami od 1 do 9, tako, da je vsota števk v zaporednih belih kvadratkih po vrsticah in stolpcih enaka številu, ki je zapisano v črnem kvadratku na začetku vrstice (stolpca) nad (pod) diagonalo. Pri tem pa morajo biti vse števke v posamezni vrstici (stolpcu) različne.



↓↓↓ **REŠITEV**



×××