Dinamično programiranje in problem nahrbtnika

Ines Meršak

→ **Dinamično programiranje je metoda reševanja določenih problemov, ko iščemo najboljši rezultat, npr. najkrajšo pot ali največjo nagrado. Problemi, ki jih lahko rešimo tako, da jih razbijemo na manjše, poiščemo najboljšo rešitev le-teh in te rešitve združimo v rešitev večjega problema, so zelo primerni za dinamično programiranje. Če se nekateri podproblemi prekrivajo, potem uporaba dinamičnega programiranja močno pospeši algoritem za reševanje.**

Zgodovina in poimenovanje

Izvor imena *dinamično programiranje* je nenavaden in zanimiv. Kot bomo videli, ni pri dinamičnem programiranju nič preveč dinamičnega, metoda pa tudi ni neposredno povezana s programiranjem, saj gre pravzaprav za matematično tehniko. Dinamično programiranje je v petdesetih letih prejšnjega stoletja razvil ameriški matematik Richard Bellman in v svoji avtobiografiji [1] opisal razlog za čudno poimenovanje. Petdeseta leta niso bila najboljša za financiranje matematičnih raziskav. Da bi Bellman pridobil financiranje ministrstva za obrambo, je moral s primerno izbranim imenom zakriti, kaj bo v resnici počel. S svojo tehniko je želel reševati probleme optimalnega načrtovanja. A besedo načrtovanje je zamenjal z besedo programiranje. Ker je želel opisati, da se metoda dogaja v več korakih, je izbral še besedo dinamično, delno zato, ker že ima močan fizikalen pomen, in delno zato, ker jo je težko uporabiti v zaničevalnem smislu. Z besedno zvezo dinamično programiranje je bil zadovoljen, saj je bila dovolj generična, da poslanci in ministri niso ugovarjali; financiranje raziskav je bilo tako zagotovljeno.

Ilustrativni primer

Osnovni primer dinamičnega programiranja, ki kaže strukturo ponavljajočih podproblemov, je izračun n-tega Fibonaccijevega števila. Spomnimo se, da so Fibonaccijeva števila definirana z zvezo

- $F_0 = 1,\ F_1 = 1,\ F_n = F_{n-1} + F_{n-2},$

torej je vsako naslednje število vsota prejšnjih dveh, pri čemer začnemo z 1, 1. Če bi funkcijo za izračun n-tega števila napisali kot

```
def fib(n):
    if n <= 1: return 1
    else: return fib(n-1) + fib(n-2)
```

in izračunali `fib(5)`, bi dobili zaporedne klice:

- $\begin{aligned}\texttt{fib}(5) &= \texttt{fib}(4) + \texttt{fib}(3)\\ &= (\texttt{fib}(3) + \texttt{fib}(2)) + (\texttt{fib}(2) + \texttt{fib}(1))\\ &= ((\texttt{fib}(2) + \texttt{fib}(1)) + (\texttt{fib}(1) +\\ &\quad + \texttt{fib}(0))) + ((\texttt{fib}(1) + \texttt{fib}(0)) + \texttt{fib}(1))\\ &= (((\texttt{fib}(1) + \texttt{fib}(0)) + \texttt{fib}(1)) +\\ &\quad + (\texttt{fib}(1) + \texttt{fib}(0))) + ((\texttt{fib}(1) +\\ &\quad + \texttt{fib}(0)) + \texttt{fib}(1))\end{aligned}$

V zadnjem izračunu, vidimo, da se kar trikrat ponovi izračun `fib(2)`, kot označeno z zvezdico:

- $(((\underbrace{\texttt{fib}(1)+\texttt{fib}(0)}_{*})+\texttt{fib}(1)) + \underbrace{(\texttt{fib}(1)+\texttt{fib}(0))}_{*})+ + ((\underbrace{\texttt{fib}(1) + \texttt{fib}(0)}_{*}) + \texttt{fib}(1)).$

To je primer manjšega podproblema enake oblike. Zato, da bi izračunali `fib(5)`, moramo izračunati `fib(3)` in `fib(4)`, za ta dva pa potrebujemo `fib(3)`, `fib(2)` (dvakrat) in `fib(1)`. Vidimo, da bi `fib(3)` in tudi `fib(2)` pri različnih podproblemih računali večkrat; to je druga lastnost, ki omogoča uporabo dinamičnega programiranja (manjši podproblemi se prekrivajo). Popolnoma nepotrebno in računsko potratno je vsakič znova ponavljati enak izračun za `fib(2)`. Dovolj je le, da ga izračunamo samo enkrat in si zapomnimo rezultat. Ko za `fib(5)` potrebujemo `fib(4)` in `fib(3)`, najprej računamo `fib(4)`. Za to potrebujemo `fib(3)` in `fib(2)` in najprej izračunamo `fib(3)`. Za to potrebujemo `fib(2)` in `fib(1)`. Zopet najprej izračunamo `fib(2)`, za kar potrebujemo `fib(1)` in `fib(0)`, ki ju oba že poznamo, ju seštejemo in dobimo `fib(2) = 2`. S tem smo izračunali prvi del izračuna za `fib(3)`, drugi del, `fib(1)`, pa poznamo po definiciji in tako dobimo `fib(3) = 3`. S tem smo dobili prvi del izračuna za `fib(4)`. Lotimo se računanja drugega dela `fib(2)`. To smo že izračunali: rezultat je enak 2, tako dobimo `fib(4) = 5`. S tem smo izračunali prvi del za izračun `fib(4)`, drugi del pa zahteva izračun `fib(3)`. Vemo, da je rezultat 3 in dobimo `fib(5) = 8`.

Alternativni način računanja, ki je morda v tem primeru lažji, je, da računamo od »spodaj«. Ker vemo, da bomo potrebovali vrednosti `fib` od 0 do 5, jih lahko računamo kar po vrsti, tako da jih imamo vedno na voljo, ko bo potrebno. Če računamo tako, dobimo

- $\texttt{fib}(0) = 1$
  $\texttt{fib}(1) = 1$
  $\texttt{fib}(2) = \texttt{fib}(1) + \texttt{fib}(0) = 1 + 1 = 2$
  $\texttt{fib}(3) = \texttt{fib}(2) + \texttt{fib}(1) = 2 + 1 = 3$
  $\texttt{fib}(4) = \texttt{fib}(3) + \texttt{fib}(2) = 3 + 2 = 5$
  $\texttt{fib}(5) = \texttt{fib}(4) + \texttt{fib}(3) = 5 + 3 = 8.$

V obeh primerih se izognemo dodatnim izračunom. Za izračun `fib(100)` bi potrebovali več ur, za izračun s pomočjo dinamičnega programiranja pa potrebujemo le nekaj milisekund.

Problem nahrbtnika

Oglejmo si še en problem, pri reševanju katerega nam uporaba dinamičnega programiranja da optimalno rešitev. Predstavljajmo si, da smo v vlogi roparja, ki ima s sabo samo en nahrbtnik z omejeno prostornino, iz zlatarne pa želi odnesti čim več predmetov. Kako izbrati predmete?

Podajmo problem malo natančneje in bolj rigorozno: dan imamo nahrbtnik, v katerega lahko damo največ W kilogramov, in n predmetov s celoštevilskimi masami $w_1, \ldots w_n$, ki so vredni $p_1, \ldots, p_n$. Naš cilj je zložiti predmete v nahrbtnik tako, da bomo v njem imeli največjo vrednost predmetov, seveda upoštevajoč, da je skupna masa predmetov v nahrbtniku manjša ali enaka W. Pri tem predmetov ne smemo deliti: za vsak predmet se odločimo, ali ga vzamemo ali ne. Tak problem nahrbtnika se imenuje tudi 0-1 nahrbtnik.

Morda najprej pomislimo, da bi predmete razvrstiti glede na razmerje vrednosti in mase $\frac{p_i}{w_i}$ tako, da so na začetku tisti, ki imajo največ »vrednosti na kilogram«. Taki rešitvi rečemo *požrešna*, nas pa ne pripelje vedno do optimalne odločitve. Oglejmo si primer.

Recimo, da imamo nahrbtnik, v katerega lahko zložimo $W = 5$ kilogramov, na voljo pa imamo tri predmete, katerih masa in vrednost sta prikazana v tabeli 1.

	vrednost	masa	razmerje
predmet 1	10	4	2,5
predmet 2	7	3	2,33…
predmet 3	4	2	2

TABELA 1.
Prikazan je primer 0-1 nahrbtnika z omejitvijo $W = 5$, kjer požrešna strategija ne deluje.

Če predmete razvrstimo padajoče po razmerju med vrednostjo in maso, ima prvi predmet najboljše razmerje $10/4 = 2{,}5$, nato drugi predmet z razmerjem $7/3 \approx 2{,}33$, najmanjše razmerje pa ima tretji predmet $4/2 = 2$. Če torej vzemamo predmete po vrsti, v nahrbtnik pospravimo prvi predmet, masa našega nahrbtnika je zdaj 4 kilograme, vrednost pa 10. Drugega predmeta ne moremo več vzeti, saj bi skupna masa bila $4 + 3 = 7$, kar je več kot 5 kilogramov, kar je naša omejitev.

Vendar pa to ni najboljša rešitev, ki jo lahko dosežemo s temi predmeti. Če namreč vzamemo drugi in tretji predmet, bo skupna masa našega nahrbtnika še ravno dovoljenih 5 kilogramov, vrednost pa bo 11. Požrešen pristop nas v tem primeru ni pripeljal do prave rešitve.

Ali se lahko problema lotimo z dinamičnim programiranjem? Poskusimo najti rešitev s pomočjo rešitve manjših podproblemov. Lahko se omejimo na manjše število predmetov, namesto da upoštevamo vse predmete naenkrat, in se vprašamo, kakšna bi bila optimalna vrednost, če bi imeli na voljo le en predmet, dva predmeta ipd. Poleg tega lahko kot na podproblem gledamo tudi, če je volumen nahrbtnika, ki ga imamo na voljo, manjši.

Označimo optimalno vrednost problema nahrbtnika z omejitvijo w kilogramov in upoštevajoč predmete $1, \ldots, i$ s $k(w, i)$. Za naš primer si zamislimo najbolj enostaven podproblem: recimo, da imamo na voljo le 1 kilogram prostora v našem nahrbtniku, torej $w = 1$. Vsi trije naši predmeti imajo maso večjo od 1, torej v nahrbtnik ne moremo spraviti nobenega izmed njih. Optimalne vrednosti so torej enake 0 ne glede na to, koliko predmetov upoštevamo (samo prvega, prvega in drugega, vse tri):

- $k(1,1) = k(1,2) = k(1,3) = 0.$

Povečajmo prostornino našega nahrbtnika za 1, torej $w = 2$. Če upoštevamo prvi predmet, ali pa prvi in drugi predmet, bo optimalna vrednost nahrbtnika enaka 0, saj sta oba predmeta pretežka. Če pa upoštevamo vse tri predmete, lahko v nahrbtnik spravimo le tretji predmet. Optimalna vrednost bo v tem primeru torej $p_3 = 4$:

- $k(2,1) = k(2,2) = 0, \quad k(2,3) = 4.$

Za omejitev $w = 3$ je prvi predmet še vedno prevelik. Če upoštevamo prva dva predmeta, je optimalna vrednost ravno vrednost drugega predmeta:

- $k(3,1) = 0, \quad k(3,2) = p_2 = 7.$

Kaj pa, če upoštevamo vse tri predmete? Sedaj imamo dva predmeta, ki bi šla v naš nahrbtnik (drugi predmet in tretji predmet). Vzamemo največjega, torej drugi predmet. Poglejmo še drugače: rešitev, ki upošteva vse tri predmete, lahko sestavimo iz rešitve, ki upošteva prva dva. Torej se pravzaprav odločamo, ali želimo tretji predmet dati v nahrbtnik ali ne. Če ga damo, potem je preostala prostornina nahrbtnika še 1, torej je vrednost nahrbtnika v tem primeru seštevek vrednosti tretjega predmeta in optimalne vrednosti za podproblem z omejitvijo nahrbtnika 1, upoštevajoč prvi in drugi predmet. Če pa ga ne dodamo v nahrbtnik, imamo na voljo še vse 3 kilograme, vrednost našega nahrbtnika je kar optimalna vrednost podproblema za $w = 3$, upoštevajoč prvi in drugi predmet. Optimalna vrednost bo seveda maksimum obeh dveh vrednosti:

- $$k(3,3) = \max\{\underbrace{k(1,2) + p_3}_{\text{vzamemo predmet 3}}, \overbrace{k(3,2)}^{\text{ne vzamemo predmeta 3}}\} = \max\{0 + 4, 7\} = 7.$$

Na hitro si poglejmo še optimalne vrednosti za $w = 4$. Ko upoštevamo le prvi predmet, ga seveda vzamemo, v nahrbtnik pa potem ne moremo spraviti ničesar več. Če upoštevamo prvi in drugi predmet, ugotovimo, da lahko v nahrbtnik spravimo le enega od teh dveh, torej je bolje vzeti vrednejšega, to je prvi predmet. Podobno ugotovimo, ko upoštevamo vse tri, torej

- $k(4,1) = k(4,2) = k(4,3) = p_1 = 10.$

Sedaj lahko rešimo naš primer za omejitev, ki nas res zanima, $w = 5$. Če upoštevamo le prvi predmet, ga vzamemo; vrednost našega nahrbtnika je 10. Če upoštevamo prva dva predmeta, se torej odločamo, ali bi vzeli drugega, pri čemer bi v nahrbtniku ostala 2 kilograma prostora, kar ne bi bilo dovolj za prvega, ali drugega predmeta ne bi vzeli, v tem primeru bi torej vzeli optimalno rešitev podproblema za $w = 5$, ki smo ga ravnokar izračunali. Ker ima prvi predmet precej boljšo vrednost, je maksimum dosežen pri drugi opciji. Sedaj poglejmo še, kako je, če upoštevamo vse tri predmete. Odločamo se, ali izbrati tretji predmet ali ne. Če ga izberemo, potem imamo na voljo še 3 kilograme, vrednost našega nahrbtnika je torej seštevek vrednosti tretjega predmeta p_3 in optimalne rešitve podproblema nahrbtnika z omeji-

tvijo $w = 3$, upoštevajoč prva dva predmeta:

- $k(5,1) = p_1 = 10$,
  $k(5,2) = \max\{k(2,1) + p_2, k(5,1)\}$
  $= \max\{0 + 7, 10\} = 10$,
  $k(5,3) = \max\{k(3,2) + p_3, k(5,2)\}$
  $= \max\{7 + 4, 10\} = 11$.

Optimalne rešitve vseh podproblemov lahko tabeliramo, kar je prikazano v tabeli 2. Odgovor na naše vprašanje se skriva desno spodaj, kjer je vrednost nahrbtnika upoštevajoč vse predmete in začetno podano omejitev mase.

w\i	1	2	3
1	0	0	0
2	0	0	4
3	0	7	7
4	10	10	10
5	10	10	11

TABELA 2.
Tabela optimalnih rešitev podproblemov (vrednosti $k(w,i)$) za izbrani primer 0-1 nahrbtnika.

Sedaj razmislimo o problemu v splošnem in poskusimo ugotoviti, kako se manjši problemi uporabijo za reševanje večjega ter poskusimo zapisati splošno rekurzivno formulo. Spomnimo se, da s $k(w,i)$ označimo optimalno vrednost problema nahrbtnika z omejitvijo w kilogramov in upoštevajoč predmete $1,\ldots,i$, pri čemer imamo skupaj N predmetov. Vrednost $k(w,i)$ želimo izračunati s pomočjo rešitve *podproblemov* za različne omejitve nahrbtnikov, kjer upoštevamo samo predmete do $i-1$. Predstavljajmo si torej, da že imamo rešitve vse podproblemov, kjer upoštevamo samo predmete do $i-1$, za različne omejitve nahrbtnikov od 1 do w.

Na tej točki se želimo odločiti, ali vzeti predmet i ali ne. Če se odločimo, da predmeta ne vzamemo, potem je naša vrednost nahrbtnika enaka vrednosti nahrbtnika $k(w,i-1)$ z omejitvijo w, upoštevajoč predmete do $i-1$. Če predmet vzamemo, potem imamo v nahrbtniku le še $w - w_i$ prostora, torej je vrednost našega nahrbtnika enaka seštevku vrednosti predmeta p_i in vrednosti nahrbtnika $k(w-w_i, i-1)$. Izberemo tisto izmed možnosti, ki da večjo vrednost nahrbtnika. Pri tem moramo upoštevati, da je možnost, da predmet vzamemo, na voljo le, če nam omejitve nahrbtnika to dopuščajo: veljati mora namreč $w \geqslant w_i$, sicer je edina možnost, da ga pustimo. Formula za izračun optimalne vrednosti je tako enaka

- $$k(w,i) = \begin{cases} \max\{k(w,i-1),\, k(w-w_i,i-1)+p_i\}, \\ \qquad \text{če } w \geqslant w_i \\ k(w,i-1), \\ \qquad \text{sicer} \end{cases} \tag{1}$$

Za popolno rešitev moramo razmisliti še o končnih pogojih. Podobno kot v primeru, ki smo ga obravnavali prej, je vrednost nahrbtnika z omejitvijo teže 0 enaka 0, saj ne moremo vanj shraniti nobenega predmeta. Prav tako je vrednost nahrbtnika enaka 0, če nimamo na voljo nobenega predmeta ne glede na kapaciteto, ki je na voljo. Zapisano s formulami so robni pogoji enaki

- $k(w,0) = 0$ za vse w od 0 do W
  $k(0,i) = 0$ za vse i od 0 do N (2)

Rešitev lahko zapišemo tudi s pomočjo programske kode, ki tesno sledi zgornji razlagi. Funkcija `knapsack` spodaj sprejme omejitev W, seznam tež predmetov `weights`, ki hrani $w_1,\ldots,w_N$, ter seznam vrednosti predmetov `prices`, ki hrani vrednosti $p_1,\ldots,p_N$. Vrednosti $k(w,i)$ izračunamo za vse $w = 0,\ldots,W$ in $i = 0,\ldots,W$. Za začetek si pripravimo prazno tabelo, v katere bomo vpisovali vrednosti $k(w,i)$. Nato v dveh zankah nastavimo robne pogoje pri $i = 0$ in $w = 0$, kot smo opisali v enačbi (2). Na koncu po vrsti izračunamo še vse ostale vrednosti $k(w,i)$, pri čemer jih računamo s pomočjo formule (1). Pri tem vrednosti računamo v pravem vrstnem redu, tako da sta pri računanju `k[w][i]` manjša podproblema `k[w][i-1]` in `k[w-w_i][i-1]` že izračunana.

```
def knapsack(W, weights, prices):
    N = len(weights)
    k = [[None for _ in range(N+1)] for _
    in range(W+1)]
    for i in range(N+1):
```

```
    k[0][i] = 0
for w in range(W+1):
    k[w][0] = 0

for w in range(1, W+1):
    for i in range(1, N+1):
        w_i, p_i = weights[i-1],
       prices[i-1]
        if w_i > w:
            k[w][i] = k[w][i-1]
        else:
            k[w][i] = max(k[w][i-1],
      k[w-w_i][i-1] + p_i)

return k[W][N]
```

Ta tehnika računanja reši problem v približno $W \cdot N$ korakih, kar je precej bolj učinkovito, kot če bi preverili vse možnosti. Bralci ste tudi spodbujeni, da preverite, ali podana programska koda izračuna enake številke, kot so dane v tabeli 2. Lahko pa poskusite rešiti primer z npr. petimi ali več predmeti in se prepričate o pravilnosti in enostavnosti postopka.

Literatura

[1] R. Bellman, *Eye of the Hurricane: An Autobiography*, World Scientific, 1984.

× × ×

Rešitev nagradne uganke

↓↓↓

Boštjan Kuzman

→

Bralcem smo v članku 21 aritmetičnih vprašanj o številu 2021 v prejšnji številki zastavili naslednjo uganko: *Za katera naravna števila n se število $n!$ v običajnem desetiškem zapisu konča z natanko 2021 ničlami?* Do 14. marca 2021 smo v uredništvu prejeli dve rešitvi, obe pravilni: iskano število sploh ne obstaja. Uspešna reševalca Ivan Lisac iz Kopra in Andrej Jakobčič iz Novega mesta bosta za nagrado prejela knjigo o teoriji števil iz ponudbe DMFA – založništva. Rešitev, do katere lahko pridemo tudi brez računalnika, je zapisana v nadaljevanju.

Označimo število končnih ničel števila $n!$ z oznako $t(n)$. Kot je bilo opisano že v članku, je število $t(n)$ enako eksponentu prafaktorja 5 v prafaktorizaciji števila $n!$, saj vsaka končna ničla nastane z množenjem para prafaktorjev 2 in 5. Ker je med 1 in n natanko $[\frac{n}{5}]$ večkratnikov števila 5, natanko $[\frac{n}{5^2}]$ večkratnikov števila 25 in tako dalje, lahko za dani n vrednost $t(n)$ izračunamo po De Polignacovi (oziroma Legendrovi) formuli $t(n) = \sum_{k=1}^{\infty} \left[\frac{n}{5^k}\right]$.

Da bi določili število n z lastnostjo $t(n) = 2021$, je potrebno razmišljati v obratno smer. Števila n seveda ni mogoče direktno izraziti iz enačbe. Ker pa za funkcijo celi del velja $[x] \leqslant x$, lahko z uporabo formule za vsoto geometrijske vrste dobimo oceno

- $$t(n) = 2021 < \sum_{k=1}^{\infty} \frac{n}{5^k} = \frac{n}{5} \sum_{k=0}^{\infty} \left(\frac{1}{5}\right)^k = \frac{n}{5} \cdot \frac{1}{1-\frac{1}{5}} = \frac{n}{4}$$

oziroma $n > 8084$. Za $n = 8085$ zdaj z uporabo De Polignacove formule dobimo $t(8085) = 2018$, torej je treba n nekoliko povečati. Opazimo lahko, da zaradi preštevanja večkratnikov števila 5 velja, da je $t(n) > t(n-1)$, če je n večkratnik števila 5, sicer pa je $t(n) = t(n-1)$. Zato hitro ugotovimo, da je $t(8090) = \ldots = t(8094) = 2019$ in $t(8095) = \ldots = t(8099) = 2020$, toda $t(8100) = 2022$, saj je 8100 deljivo s 25. Dokazali smo, da iskano število n ne obstaja.

Omenimo še, da lahko ročno računanje z De Polignacovo formulo precej pohitrimo z uporabo znane zveze $[n/(ab)] = [[n/a]/b]$, po kateri lahko rekurzivno izračunamo izraze oblike $[n/p^k]$. Za števili $n = 8085$ in $p = 5$ dobimo denimo $[8085/5] = 1617$, $[8085/5^2] = [1617/5] = 323$, $[8085/5^3] = [323/5] = 64$, $[8085/5^4] = [64/5] = 12$, $[8085/5^5] = [12/5] = 2$ in $[8085/5^6] = [2/5] = 0$, od koder sledi $t(8085) = 1617 + 323 + 64 + 12 + 2 + 0 = 2018$.

× × ×