Kaip panešti milijoną?

Uždavinys sprendžiamas dinaminio programavimo metodu.

Šiam, kaip ir daugeliui kitų uždavinių, sprendžiamų dinaminio programavimo metodu, kyla pagunda pritaikyti *Godujį algoritmą* – išrikiuoti brangenybes jų verčių ir svorio santykio (t.y. rasti kiekvienos brangenybės svorio vieneto vertę) mažėjimo tvarka ir iš eilės imti vertingiausias brangenybes, kol dar galima panešti. Žemiau pateiktas pavyzdys parodo, kad toks algoritmas šio uždavinio nesprendžia:

Tarkime, kad vaikinas gali panešti maišą, sveriantį ne daugiau 50 svorio vienetų. Karalius turi 3 brangenybes, kurių svoriai ir vertės:

Nr.	*Svoris*	*vertė*	*svorio vieneto vertė*
1	10	60	6
2	20	100	5
3	30	120	4

Remiantis *Godžiuoju* algoritmu reikėtų imti 1 ir 2 brangenybes. Jų suminė vertė 160. Tačiau paėmę 2 ir 3 brangenybes, gausime didesnę suminę vertę – 220.

Uždavinį sprendžiame dinaminio programavimo metodu. Pirmasis žingsnis – jį suskaidyti į keletą mažesnių uždavinių.

Pažymėkime V(i, sv) (, $1 \le i \le n$, $1 \le sv \le s$) maksimalią vertę siūlomų imti brangenybių, jei renkamasi tik iš brangenybių, kurių numeriai neviršija i, o visas svoris neviršija sv. Pavyzdžiui, V(40, 2) = 160 (žr. pavyzdį). Mat galime rinktis tik iš dviejų pirmųjų brangenybių, o išrinktų brangenybių svoris neturi viršyti 40.

Taigi V (n, s) yra ieškoma brangenybių, kurias reikėtų imti, maksimali vertė, t.y. pradinis uždavinys. Su kiekviena brangenybe galimi du atvejai – ji dedama į maišą arba ne.

Pradėkime nagrinėti nuo n-osios brangenybės. Jeigu ją dedame į maišą, tai

$$V(n, s) = V(n - 1, s - sv_n, n - 1) + ve_n;$$

čia sv_n – n-osios brangenybės svoris, o ve_n – jos vertė.

Jei tos brangenybės nededame į maišą, tai

$$V(n, s) = V(n - 1, s).$$

Kaip reikėtų pasielgti su *n*-ąja brangenybe? Akivaizdu, kad reikia rinktis tą atvejį, kai gaunama didesnė vertė, t. y.

$$V(n, s) = \max (V(n - 1, s - sv_n) + ve_n, V(n - 1, s).$$

Pradinį uždavinį suskaidėme į du mažesnius. Norėdami rasti V(*n*, *s*), turime rasti V reikšmes su mažesniais parametrais. Turėdami šią rekursinę formulę, galėtume ir grįžimo metodu ieškoti sprendinio. Tačiau tai bus neefektyvu, nes bus ilgai perrenkama.

Taikykime *Dinaminį metodą*, kuris laiko sąnaudas pakeičia atminties sąnaudomis. Skaidant du gautus uždavinius į dar mažesnius, gali atsitikti taip, kad gausime bendrų dalių. Todėl vieną kartą apskaičiavę reiškinio V(*i*, *sv*) reikšmę ją įsimename. Jei tos pačios reikšmės prireiks antrą kartą – jos nebereikės skaičiuoti iš naujo. Tereikės pasiimti iš lentelės.

Sąlygoje prašoma ne tik surasti imamų brangenybių bendrą vertę, bet ir pačių brangenybių numerius. Tam kiekvienam lentelės elementui V(*i*, *sv*) įvedame dar vieną atributą, kuris parodo imti, ar neimti brangenybės duotoje situacijoje. Atkreipiame dėmesį, kad esant vienokiam svoriui ta pati brangenybė imama, kitokiam – neįmanoma.

Žemiau pateikta lentelė sudaryta ankstesniam pavyzdžiui.

V(*sv*, *i*)	1	2	3
0	0	0	0
1	0	0	0
10	60, imti	60, neimti	60, neimti
11	60, imti	60, neimti	60, neimti
…	…	…	…
20	**60, imti**	**100, imti**	100, neimti
…	…	…	…
30	**60, imti**	100, imti	160, neimti
…	…	…	…
50	**60, imti**	**160, imti**	**220, imti**

Šią lentelę užpildėme visą (iš apačios į viršų), kad geriau iliustruotume pavyzdį. Algoritme lentelė pildoma rekursiškai iš viršaus į apačią, todėl užpildomi tik reikalingi lentelės elementai. Aukščiau esančioje lentelėje jie pažymėti pajuodintu šriftu.

```
const max_s = 100;     { maksimalus maišo svoris }
      max_b = 200;     { maksimalus brangenybių skaičius }
type lentelė = array [1..max_b, 1..max_s] of
                record
                  r: integer;    { maksimali vertė }
                  imti: boolean  { ar imti brangenybę }
                end;
     duom = array [1..max_b] of integer;
var V: lentelė; { globalieji kintamieji }
    sv,   { brangenybių svoriai }
    ve: duom;   { brangenybių vertės }

procedure skaičiuoti(n, s: integer; var vertė: integer);
  var i, j: integer;
  function ieškoti (n, s: integer): integer;
 { rekursinė funkcija, pildanti lentelę V, iš karto randa maksimalią brangenybių, }
 { kurias galima panešti, vertę }
    var vert1, { maksimali vertė, jei neimame n-osios brangenybės }
        vert2: integer;{ maksimali vertė, jei tą brangenybę imame }
  begin
    if s = 0
    then begin    { jei svoris lygus nuliui }
           ieškoti := 0;
           exit
         end
       else if V[n, s].r <> -1   { arba jei ieškoma reikšmė apskaičiuota }
               then begin
                      ieškoti := V[n, s].r;
                      exit
                    end;
    { toliau nagrinėjame du atvejus: }
    { 1) jei n-osios brangenybės neimame }
    if n = 1
       then  vert1 := 0
       else  vert1 := ieškoti(n - 1, s);
```

```
    { 2) jei brangenybę imame }
    if (sv[n] <= s) { ją galime paimti, jei neviršysime svorio limito }
    then if n = 1
         then vert2 := ve[n]
         else vert2 := ve[n] + ieškoti(n-1, s-sv[n])
    else vert2 := -1;
    { pasirenkame vieną iš dviejų variantų }
    if vert1 >= vert2
       then begin
              ieškoti := vert1;
              V[n, s].r := vert1;
            end
       else begin
              ieškoti := vert2;
              V[n, s].r := vert2;
              V[n, s].imti := true
            end
  end; { ieškoti }
begin
  { užpildome lentelę pradinėmis reikšmėmis }
  for i := 1 to n do
    for j := 1 to s do
        begin
          V[i, j].imti := false;
          V[i, j].r := -1
        end;
  vertė := ieškoti (n, s)
end; { skaičiuoti }
```