Erdvėlaivis

Jei kreditų skaičius būtų neribojamas, tai būtų trumpiausio kelio paieškos grafe uždavinys, kurį būtų galima spręsti pagal *Dijkstros* algoritmą ar kokį kitą dar paprastesnį algoritmą. Šiuo atveju situacija sudėtingesnė.

Kokiu būdų bespręstume uždavinį, būtinai reikės tikrinti, ar užteks kuro, esančio bake iš vienos bazės nuskristi į kitą. Todėl pradžioje apskaičiuokime atstumus tarp visų bazių. Sakykime, kad turime dvi bazes $a(x_1, y_1, z_1)$ ir $b(x_2, y_2, z_2)$. Atstumas tarp jų skaičiuojamas pagal formulę:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Na o maksimalus atstumas, kurį galima nuskristi su pilnu kuro rezervuaru lygus *v*/*san*, čia *v* – kuro rezervuaro talpa, *san* – kuro sąnaudos vienam ilgio vienetui.

```
const MAX_N = 100;  { maksimalus bazių skaičius }
type  bazės = array [1..MAX_N] of
                                record  { duomenys apie bazes }
                                  k, x, y, z: integer;
                                end;
        atstumai = array [1..MAX_N, 1..MAX_N] of real;
                   { atstumai tarp bazių }

procedure atstum (n: integer; baz: bazės;
                  var ats: atstumai);
{ suskaičiuoja atstumus tarp bazių }
  var i, j: integer;
      d: real;
begin
  for i := 1 to n do
    for j := 1 to n do
      ats[i, j] := 0;
  for i := 1 to n do
    for j := i+1 to n do
      begin
        d := sqrt(sqr(baz[i].x - baz[j].x) +
                  sqr(baz[i].y - baz[j].y) +
                  sqr(baz[i].z - baz[j].z));
```

```
        ats[i, j] := d;
        ats[j, i] := d;
    end;
end;  { atstum }
```

Uždavinį spręsime *Dinaminio programavimo* metodu. Susidarysime dvimatę lentelę, turinčią tiek eilučių kiek yra bazių ir tiek stulpelių, kiek yra kreditų. Lentelę pavadinkime *m*. Kiekvienas lentelės elementas susidės iš dviejų laukų. *m*[*a*, *kr*].*kur* rodys bazę į kurią reikia skristi iš bazės *a*, jei turime *kr* kreditų. Jei turint *kr* kreditų iš bazės *a* neįmanoma pasiekti galutinės bazės, tai *m*[*a*, *kr*].*kur* = -1. *m*[*a*, *kr*].*ats_n* yra trumpiausio kelio ilgis nuo bazės *a* iki galutinės bazės, jei turime *k* kreditų.

Nesunkiai galime užpildyti pirmąjį lentelės stulpelį ir paskutiniąją eilutę. Mat *m*[*a*, 0].*kur* = -1, nebent nereikėtų mokėti už kurą bazėje *a* ir jo užtektų kuro be sustojimų pasiekti bazę *n*. Taip pat akivaizdu kad *m*[N, *kr*].*kur* = 0, ir *m*[N, *kr*].*ats_n* = 0. Mat iš paskutiniosios bazės niekur skristi nereikia.

Imkime sąlygoje patiktą pavyzdį ir pradėkime pildyti lentelę. Su pilnu baku kuro galima nuskristi 10 ilgio vienetų.

Atstumai tarp bazių pateikti lentelėje dešinėje

bazė	1	2	3	4
1	0.00	10.00	7.07	10.20
2	10.00	0.00	7.07	2.00
3	7.07	7.07	0.00	5.83
4	10.20	2.00	5.83	0.00

kreditas /bazė	0	1	2	3	4	5	6	7	8	9	10
1	-1										
2	-1										
3	-1										
4	0	0	0	0	0	0	0	0	0	0	0
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Lentelę pildysime iš kairės į dešinę. Pirmasis lentelės stulpelis jau užpildytas. Sakysime, kad turime *k* kreditų, o kuro bako papildymas bazėje *a* kainuoja *kred* kreditų. Išskridus iš bazės a liks *k* - *kred* kreditų. Peržiūrime visas bazes iki kurių be sustojimo galime nuskristi iš *a*. Iš jų

išsirenkame tas, iš kurių turėdami *k* - *kred* kreditų galėsime pasiekti galutinę bazę. T. y. peržiūrėdami reikiamus anksčiau užpildytus stulpelius, pildome tolesnį stulpelį.

Nagrinėjamam pavyzdžiui užpildyta lentelė atrodytų taip:

kreditas /bazė	0	1	2	3	4	5	6	7	8	9	10
1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	3
											12.90
2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
3	-1	-1	-1	-1	-1	4	4	4	4	4	4
						5.83	5.83	5.83	5.83	5.83	5.83
4	0	0	0	0	0	0	0	0	0	0	0
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Keblumų sukelia atvejis, kai bazėje už kurą mokėti nereikia. Tokiu atveju *k* - kred = k. Šiuo atveju tektų pildyti *k*-tąjį stulpelį, ir nagrinėti to paties stulpelio dar neužpildytus langelius *m*[*b*, *k*].*kur*. Problema sprendžiama taip. Pildant stulpelį, pirmiausia užpildomi tų bazių, kuriose reikia mokėti už kurą langeliai. Tuomet pildomi bazių, kuriose mokėti nereikia, langeliai. Jei bazėje *b* į kurią skrendame už kurą mokėti reikia, tai naudojame jau užpildytą langelio *m*[*b*, *k*].*kur* reikšmę. Jei bazėje *b* už kurą mokėti nereikia, imame langelio *m*[*b*, *k*-1].*kur* reikšmę.

Užrašant algoritmą programa iškyla atmintis problema. Mat aukščiau minėtai lentelei trūktų atminties. Todėl lentelę suskaidome į dvi lenteles: bazių į kurias reikia skristi ir atstumų lentelę. Pirmajai vartojame statinę, antrajai – dinaminę atmintį.

```
const MAX_N = 100;  { maksimalus bazių skaičius }
      MAX_K = 100;  { maksimalus kreditas }
      DAUG = 1000;  { atstumas tarp dviejų bazių nebus didesnis už šį }
type  bazės = array [1..MAX_N] of record { duomenys apie bazes }
                                   k, x, y, z: integer;
                                 end;
      atstumai = array [1..MAX_N, 1..MAX_N] of real;
                          { atstumai tarp bazių }
      maršrutas = array [1..MAX_N, 0..MAX_K] of integer;
   { elementas [a, kr] parodo, į kurią bazę reikia skristi iš bazės a, turint kreditą kr }
```

```
      t_atst = array[1..MAX_N, 0..MAX_K] of real;
      atstumas_n = ^t_atst;
      { parodo, koks trumpiausias atstumas iki n-tosios bazės }
      { jei esame bazėje a ir turime kr kreditų }

procedure skrieti (v, san, { rezervuaro talpa ir kuro sąnaudos ilgio vnt. }
                   kr, n: integer;
                   { turimas kreditas ir bazių skaičius }
                   baz: bazės; var kur: maršrutas);
{ randa optimalų skridimo maršrutą }
  var ats: atstumai;
      min,
      max_kel: real;     { ilgiausias kelias, kurį galima }
                         { nuskristi su pilnu baku kuro }
      ats_n: atstumas_n; { trumpiausi atstumai iki n-tosios bazės }
      i, k, a, b, lik: integer;

  procedure inicializuoti (var kur: maršrutas;
                           var ats_n: atstumas_n);
  { užpildo lentelių pirmąjį stulpelį ir paskutinę eilutę }
  { vartoja globalius kintamuosius n, kr, max_kel, ats, baz }
  { šių kintamųjų reikšmių nekeičia }
    var a, k: integer;
  begin
    new (ats_n);
    { užpildysime kur[a, 0] }
    for a := 1 to n-1 do
      if (baz[a].k = 0) and (max_kel >= ats[a, n])
         then begin
                kur[a, 0] := n;
                ats_n^[a, 0] := ats[a, n];
              end
         else kur[a, 0] := -1; { nuskristi be mokesčio negalima }
    { užpildysime kur[n, k] }
    for k := 0 to kr do
      begin
        kur[n, k] := 0; { toliau skristi nebereikia }
        ats_n^[n, k] := 0;
      end;
  end; { inicializuoti }
```

```
procedure skristi_a (a, k: integer; { skrenda iš a su kreditu k }
                     var kur: maršrutas;
                     var ats_n: atstumas_n);
  { randa trumpiausią kelią iš bazės a į bazę n, kai turima k kreditų }
  { vartojami globalieji kintamieji max_kel, baz, ats }
  { šių kintamųjų reikšmės nekeičiamos }
  var min: real; { minimalus atstumas nuo bazės a iki bazės n }
      b,
      lik: integer; { kredito likutis }
  begin
    min := DAUG;
    kur[a, k] := -1; { kelio kol kas neradome }
    for b := 1 to n do { skrendame į bazę b }
      if (ats[a, b] <= max_kel) and (a <> b)
          { jei bazė b pasiekiama }
      then begin
             if (baz[a].k = 0) and (baz[b].k = 0)
                then lik := k - 1
                else lik := k - baz[a].k;
             if (kur[b, lik] >= 0)
                { jei iš bazės b pasieksim n-tąją bazę }
                and (min > (ats[a, b] + ats_n^[b, lik]))
                { ir naujas kelias yra trumpesnis }
                then begin
                       min := ats[a, b] + ats_n^[b, lik];
                       kur[a, k] := b;
                       ats_n^[a, k] := min;
                     end;
           end;
  end; { skristi }

begin
  { rasime atstumus tarp bazių }
  atstum (n, baz, ats);
  max_kel := v/san;
  inicializuoti (kur, ats_n);
  { užpildysime likusią lentelės dalį }
  for k := 1 to kr do
    begin { turime k kreditų }
      { pirmiausia skrendame iš bazių, kuriose reikia mokėti }
      for a := 1 to n-1 do { skrendame iš bazės a }
        if k < baz[a].k { jei iš šios bazės negalima išskristi }
          then kur[a, k] := -1
          else if baz[a].k <> 0
```

```
                    then skristi_a (a, k, kur, ats_n);
      { tuomet skrendame iš bazių, kuriose nereikia mokėti }
      for a := 1 to n-1 do
        if baz[a].k = 0
           then skristi_a (a, k, kur, ats_n);
   end;
end;  { skrieti }
```