Lobio ieškojimas

Šį uždavinį galima būtų pavadinti labiau skaitymo, nei algoritmavimo uždaviniu. Sąlyga labai ilga ir paini. Tačiau ją įsiskaitęs pamatai, kad sprendimas nėra kažkuo ypatingas. Duotas labirintas, reikia jame rasti lobį ir iš jo išeiti. Įprasta labirintų uždavinius spręsti *Bangos* arba *Paieškos į plotį metodu*. Deja, šiam uždaviniui tų metodų pritaikyti negalime. „Reikalą gadina“ dvi sienos ir sprogmenys. Galima rasti trumpiausią kelią nuo įėjimo iki lobio ir nuo lobio iki išėjimo. Tačiau, ką daryti, jei ateinant trumpiausiu keliu teko išsprogdinti dvi skyles sienose, o išeinant trumpiausiu keliu taip pat tenka sprogdinti sienas kitose vietose.

Todėl pasirenkame *Grįžimo metodą*. Labirintas yra nedidelis, taigi laiko limitas nebus viršyti.

Apeinant labirintą grįžimo metodu, žymimasi kuriuose langeliuose būta. Mes taip pat žymėsime kuriuose langeliuose būta. Tačiau radus lobį, labirinto langelių būsenas teks įsiminti, ir vėl pažymėti, kad nei viename labirinto langelyje dar nebūta: juk grįžtant gali tekti eiti tais pačiais langeliais kaip ir atėjome. Įsiminti aplankytus langelius lobio atradimo momentu reikia tam, kad galėtume grįžti atgal.

Įsimenant kurie langeliai buvo aplankyti, svarbu įsiminti, kaip į juos pakliuvome: ar atėjome patys, ar mus permetė (atnešė). Imkime langelį D_2 (žr.pav.).

D_3	S	S	S
U_2	U_3	U_4	U_5
U_1	P	S	Ž
D_1	L	D_2	S

Jei į šį langelį ateiname patys, tai iš jo mus permeta į D_3. Jei į šį langelį, pakliūname iš D_1, galime pasiekti lobį.

Sprendimą pradėkime nuo sienų formavimo. Mums būtų patogu, jei turėdami dviejų langelių koordinates iš karto galėtume pasakyti: skiria juos siena, ar ne. Tam sudarykime keturių matavimų loginę lentelę. Du pirmieji elemento indeksai reikš pirmojo langelio koordinates, du paskutinieji – antrojo.

```
type sienos = array [1..D, 1..D, 1..D, 1..D] of boolean;
```

Toks vaizdavimas užima daug atminties. Kadangi labirintas labai mažas tai sprendimui nekliudys.

```
type siena = record
                kuri,  { su kuria išorine siena liečiasi }
                n1, n2: integer;   { kuriuos stulpelius (eilutes) skiria }
              end;
     sienos = array [1..D, 1..D, 1..D, 1..D] of boolean;

procedure form_sienas (s2, s3: siena; var s: sienos);
{ suformuoja sienas }
  var y, x, u, v, i, kryptis: integer;
begin
  for y := 1 to D do
    for x := 1 to D do
      for u := 1 to D do
        for v := 1 to D do
            s[y, x, u, v] := false;
  for i := 1 to 5 do
    begin
      if i <= 2   { prie kurios labirinto sienos glaudžiasi ši siena }
         then kryptis := s2.kuri
         else kryptis := s3.kuri;
      if i in [1, 3]  { randame stulpelį (eilutę) }
         then if (kryptis = 1) or (kryptis = 4)
                 then y := 1
                 else y := 4
         else if (kryptis in [1, 4]) and (i in [2, 4]) or
                 (kryptis in [2, 3]) and (i = 5)
                 then y := 2
                 else y := 3;
      if (kryptis = 1) or (kryptis = 3)
         then if i <= 2
                 then s[y, s2.n1, y, s2.n2] := true
                 else s[y, s3.n1, y, s3.n2] := true
         else if i <= 2
                 then s[s2.n1, y, s2.n2, y] := true
```

```
                  else s[s3.n1, y, s3.n2, y] := true;
    end;
  for y := 1 to D do
   for x := 1 to D do
    for u := 1 to D do
      for v := 1 to D do
        if s[y, x, u, v]
           then s[u, v, y, x] := true
end; { form_sienas }
```

Žemiau pateikiame patį algoritmą:

```
const D = 4; { labirinto dydis }
type  labirintas = array [1..D, 1..D] of
         record
           t: char;        { laukelio tipas }
           nr: integer;    { laukelio numeris }
           būta_s,             { ar pakliūta į langelį savarankiškai }
           būta_n: boolean;    { ar pakliūta į langelį savarankiškai }
         end;

procedure nebūta (y, x: integer; var l: labirintas);
{ pažymi, kad nebūta nei viename labirinto laukelyje, }
{ išskyrus laukelį su koordinatėmis (y, x) }
  var i, j: integer;
begin
 for i := 1 to D do
   for j := 1 to D do
     begin
       l[i, j].būta_s := false;
       l[i, j].būta_n := false;
     end;
 l[y, x].būta_s := true;
end; { nebūta }

function ėjimai (l: labirintas; { labirintas }
                 įėjim_y, įėjim_x: integer; { įėjimas }
                 s2, s3: siena { sienos }): integer;
{ randa lobį ir išeina iš labirinto }
  var dx, dy: array [1..4] of -1..1; { iš laukelio einama keturiomis }
                                     { kryptimis }
      min: integer;   { minimalus ėjimų skaičius }
      s: sienos;
```

```
{ visi šie kintamieji globalūs rekursinei procedūrai eiti }

procedure rasti (y, x: integer; var kur_y, kur_x: integer);
{ randa langelį į kurį mus permeta (nuneša) iš langelio (y, x) }
  var tipas: char;
      ska, u, v: integer;
begin
  { rasime į kokio tipo langelį mus perkels }
  if l[y, x].t = 'D'
     then tipas := 'D'
     else if (l[y, x].nr <= 2)
             then tipas := 'U'
             else tipas := 'Ž';
  { rasime koks bus to langelio numeris }
  if tipas = 'D'
     then if l[y, x].nr <= 2
             then ska := l[y, x].nr + 1
             else ska := 1
     else if tipas = 'Ž'
             then ska := 1
             else ska := l[y, x].nr + 3;
  { rasime tą langelį }
  for u := 1 to D do
    for v := 1 to D do
      if (l[u, v].t = tipas) and (l[u, v].nr = ska)
         then begin
                kur_y := u; kur_x := v;
              end;
end; { rasti }

procedure eiti (y, x,     { langelio koordinatės }
                sprogm,   { turimų sprogmenų kiekis }
                ėj: integer;  { padarytų ėjimų skaičius }
                rasta,    { ar rastas lobis }
                pats: boolean;  { ar pats atėjo į langelį }
                var min: integer { minimalus ėjimų skaičius });
{ rekursinė procedūra, vaikščiojanti po labirintą }
  var k, u, v, kur_y, kur_x: integer;
      lab: labirintas;
      sprogo: boolean;
begin
  if (y = įėjim_y) and (x = įėjim_x) and rasta
  then begin   { radome lobį ir pasiekėme išėjimą }
```

```
          if min > ėj
             then min := ėj;
          exit;
        end;
if (l[y, x].t in ['U', 'D']) and pats
then begin { jei iš šio laukelio mus nuneš(permes) į kitą }
        rasti (y, x, kur_y, kur_x);
        if not l[kur_y, kur_x].būta_n { jei jame nebūta }
        then begin
               l[kur_y, kur_x].būta_n := true;
               eiti (kur_y, kur_x, sprogm, ėj, rasta,
                     false, min);
               l[kur_y, kur_x].būta_n := false
             end;
      end
else begin { eisime visomis keturiomis kryptimis }
        if l[y, x].t = 'L' { jei pasiekėme lobį }
           then begin
                  lab := l;        { įsimename aplankytus laukelius }
                  rasta := true;  { tikrai radome lobį }
                  nebūta (y, x, l); { vėl galime visur vaikščioti }
                end;
        for k := 1 to 4 do
          begin
            u := y + dy[k]; { paeiname atitinkama kryptimi }
            v := x + dx[k];
            if (u in [1..D]) and (v in [1..D]) and
               (ėj+1 < min)
               { jei langelis priklauso labirintui ir verta eiti }
            then if not l[u, v].būta_s { jei nebuvome šiame langelyje}
                 then begin
                        { gal reikia sprogdinti sieną }
                        sprogo := s[y, x, u, v];
                        if sprogo { jei reikia sprogdinti }
                        then begin
                               sprogm := sprogm - 1;
                               s[y, x, u, v] := false;
                               ėj := ėj + 1;
                             end;
                        if sprogm >= 0 { jei užteko sprogmenų }
```

```
                        then begin
                                l[u, v].būta_s := true;
                                eiti (u, v, sprogm, ėj+1,
                                      rasta, true, min);
                                l[u, v].būta_s := false;
                              end;
                      if sprogo { grįžtame per žingsnį atgal }
                      then begin
                              sprogm := sprogm + 1;
                              s[y, x, u, v] := true;
                              ėj := ėj - 1;
                            end;
                    end;
          end;
        if l[y, x].t = 'L'  { jei buvome radę lobį }
        then begin
                l := lab;   { atstatome buvusią labirinto padėtį }
                rasta := false;  { lobio vėl neturime }
              end;
    end;
end;  { eiti }

  var  i, j: integer;
begin { ėjimas }
  { apibrėžiame ėjimo kryptis }
  dy[1] := 0;  dx[1] := 1;
  dy[2] := 0;  dx[2] := -1;
  dy[3] := 1;  dx[3] := 0;
  dy[4] := -1; dx[4] := 0;
  { nė viename laukelyje dar nebūta }
  nebūta (įėjim_y, įėjim_x, l);
  min := maxint;  { minimalus ėjimų skaičius }
  { suformuojame sienas }
  form_sienas (s2, s3, s);
  eiti (įėjim_y, įėjim_x, 2, 1, false, true, min);
  ėjimai := min + 1;    { išėjimas iš labirinto taip pat yra ėjimas }
end;  { ėjimai }
```