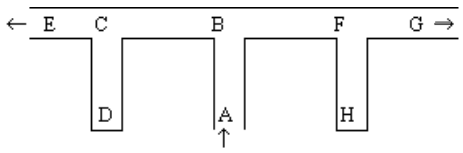
DVYLIKTAS SKYRIUS

Grįžimo metodas

Dauguma knygose, žurnaluose bei laikraščiuose pateikiamų galvosūkių įdomūs tuo, kad jiems spręsti nereikia daug matematikos ar kitų mokslų žinių. Tai kelio paieška sudėtingame labirinte, teksto perskaitymas šachmatų žirgo ėjimu, šachmatų figūrų išdėstymas lentoje taip, kad būtų tenkinamos tam tikros sąlygos. Tokius ir panašius uždavinius galima išspręsti, išmokus sistemingai nagrinėti ir įvertinti visus galimus uždavinio sprendimo variantus. Kantriai padirbėjus, randamas teisingas uždavinio sprendimo kelias, nors jis dažnai būna gerokai užmaskuotas. Čia mums gali daug pagelbėti kompiuteris.

Pateiksime gana universalų galvosūkių sprendimo būdą, kuris dažnai vadinamas grįžimo metodu. Jį aptarsime pasinaudoję tokiu pavyzdžiu.



Piešinyje pateiktas labai paprastas labirintas. Į jį patenkame taške A. Reikia rasti kelią nuo taško A iki kurio nors išėjimo kitame labirinto gale (tai gali būti išėjimai, pažymėti raidėmis E ir G). Kuris išėjimas atviras ir kaip jį pasiekti, būdami taške A mes nežinome. Todėl kiekvienoje čia pateikto labirinto kryžkelėje galima pasirinkti bet kurį iš dviejų kelių. Sutarkime, kad pirma išbandysime kairįjį kelią. Tokiu atveju iš taško A patenkame į tašką B, iš B į C, iš C į D. Iš čia kelio nebėra. Tada grįžtame atgal iki artimiausio išsišakojimo C ir iš jo bandome eiti nauju (dešiniuoju) keliu į tašką E. Čia jau randame išėjimą. Uždavinys išspręstas. Jo sprendinys – kelias ABCE.

Atkreipsime dėmesį, kad tada, kai sužinome, kad nuėjome klaidingu keliu, uždavinį spręsti pradedame ne iš naujo, o darome tiktai vieną žingsnį atgal ir iš čia vėl bandome eiti. Jeigu žengus žingsnį atgal pasirodo, kad čia nebėra naujų nenagrinėtų kelių, tai žengiamas dar vienas žingsnis atgal ir t. t., kol pasiekiamas taškas, iš kurio dar galime eiti nauju nenagrinėtu keliu. Jeigu grįžtant pasiekiamas pradinis taškas ir iš jo nebėra naujų kelių, tai reiškia, kad uždavinys sprendinio neturi.

Pavyzdys

Sudarysime algoritmą, pagal kurį šachmatų žirgo ėjimu būtų pereinami visi lentos langeliai taip, kad kiekviename langelyje žirgas pabuvotų tik vieną kartą. Pradinė žirgo padėtis – apatinis kairysis langelis. Pirmiausia pateiksime algoritmo eskizą. Procedūros *eiti* veiksmus detalizuosime vėliau.

```
const n = 8;        { lentos dydis }
type koord = 1..n;
     lenta = array [koord, koord] of integer;

procedure žirgas (var len: lenta);
  var i, j: koord;
      viskas: boolean;

  procedure eiti (i: integer;       { ėjimo numeris }
                  x, y: koord;      { pradinė žirgo padėtis }
                  var len: lenta;
                  var viskas: boolean   { užpildyta visa lenta });
```

Ėjimai surašomi į lentą (t. y. į masyvą *len*). Jie pradedami nuo langelio (*x*, *y*) – laikoma, kad ten jau yra žirgas. Ėjimai numeruojami pradedant skaičiumi *i*. Kintamojo *viskas* reikšmė *true*, jeigu sprendinys egzistuoja (t. y. po vieną kartą žirgas apsilankė visuose langeliuose) ir *false* priešingu atveju.

```
begin { žirgas }
  viskas := false;
  for i := 1 to n do
    for j := 1 to n do
      len[i, j] := 0;
  len[1, 1] := 1;   { pradinė žirgo padėtis }
  eiti (2, 1, 1, len, viskas)
end;
```

Algoritme užrašyti veiksmai pakankamai aškūs. Visa uždavinio sprendimo esmė slypi procedūroje *eiti*, kuri turi užpildyti ėjimų eilės numeriais visus likusius lentos langelius. Detalizuokime šią procedūrą.

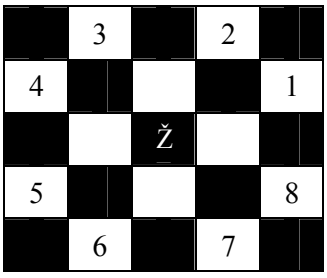
Sudaryti procedūrą, kuri iš karto išspręstų visą uždavinį (t. y. užpildytų visus lentos langelius žingsnių numeriais), sunku. Todėl uždavinį suprastinsime panaudodami rekursiją. Sudarysime procedūrą, kuri darytų tik vieną žirgo ėjimą ir užpildytų tik vieną lentos langelį. Tam, kad būtų daromas naujas ėjimas, procedūra turi vėl kreiptis pati į save. Kreipiniai tęsiasi tol, kol užpildoma visa lenta arba prieinama aklavietė, t. y. lenta dar neužpildyta, o eiti nėra kur. Tada grįžtama atgal ir bandoma eiti nauju keliu.

Kai žirgas nėra lentos krašte, galimi ėjimo variantai, parodyti paveiksle. Taigi, jeigu pradinė žirgo padėtis – masyvo *len* langelis *len*[*x*, *y*], tai bet kuri (viena iš aštuonių) nauja padėtis bus tokia:

```
len[x + cx[k], y + cy[k]]
```

Čia *k* – varianto eilės numeris, o masyvų *cx* ir *cy* reikšmės nustatomos iš pateikto paveikslo.

```
cx[1] := 2;   cy[1] := 1;
cx[2] := 1;   cy[2] := 2;
cx[3] := -1;  cy[3] := 2;
cx[4] := -2;  cy[4] := 1;
cx[5] := -2;  cy[5] := -1;
cx[6] := -1;  cy[6] := -2;
cx[7] := 1;   cy[7] := -2;
cx[8] := 2;   cy[8] := -1;
```



Kai žirgas yra arčiau lentos krašto, tai variantų skaičius mažesnis – reikia atmesti tuos variantus, kai bent vienas (arba abu) masyvo *len* indeksas nepatenka į atkarpą 1..*n* (t. y. nurodo langelį už lentos ribų).

Pateikiame procedūros *eiti* schemą.

```
procedure eiti (i: integer;            { ėjimo numeris }
                x, y: koord;           { pradinė žirgo padėtis }
                var len: lenta;
                var viskas: boolean    { užpildyta visa lenta });
  const nn = 64;    { lentos dydis n×n }
  var k: 0..8;          { varianto numeris }
      u, v: integer;    { nauja žirgo padėtis }
begin { eiti }
```

priskiriamos reikšmės masyvams *cx* ir *cy*

```
  k := 0;
  repeat
    k := k + 1;
```

```
    u := x + cx[k];
    v := y + cy[k];
    if (u >= 1) and (u <= n) and (v >= 1) and  (v <= n)
       then       { naujos koordinatės dar patenka į lentą }
         if len[u, v] = 0
           then   { langelis dar tuščias }
             begin
               len[u, v] := i;
               if i < nn then
                 begin
                   eiti (i + 1, u, v, len, viskas);
                   if not viskas then len[u, v] := 0
                 end
                 else viskas := true
              end
    until viskas or (k = n);   { užpildyta visa lenta arba peržiūrėti visi variantai }
end;   { eiti }
```

Panašiai rekursinės programos sudaromos ir kitiems uždaviniams, kuriuos sprendžiant variantai peržiūrimi tol, kol gaunamas bent vienas sprendinys arba nustatoma, jog sprendinys neegzistuoja.

Nagrinėtas grįžimo metodas yra universalus, bet nelabai ekonomiškas. Mat reikia išbandyti visus galimus kelius, nors ir ne kiekvienu keliu einama nuo pat pradžios. Atskiriems uždaviniams ar uždavinių grupėms galima rasti ir efektyvesnių sprendimo metodų.

Visuotinių (globaliųjų) kintamųjų vartojimas rekursiniuose algoritmuose

Spręsdami uždavinius *grįžimo* metodu (kartais jis dar vadinamas *paieškos į gylį* metodu), turime vartoti rekursiją. Jei įprastuose algoritmuose visuotinių kintamųjų vengiame, tai šiuo atveju be jų išsiversti negalime.

Kai procedūra kreipiasi į save, tai atmintinėje išlieka visų joje aprašytų kintamųjų (taip pat ir parametrų) reikšmės, o naujai sukurto procedūros egzemplioriaus kintamiesiems paskiriama nauja vieta – kiekvienas kreipinys pareikalauja naujos atmintinės porcijos, kuri išlaisvinama tik grįžtant atgal. Kintamųjų reikšmėms laikyti skiriama atmintinės sritis vadinama dėklu (anglų k. *stack*). Dėklo dydis ribotas. Todėl rekursinėse procedūrose reikia vengti didesnių duomenų struktūrų, ypač masyvų. Didesnių masyvų aprašus geriau iškelti iš procedūros – juos padaryti visuotiniais.

Visuotinius kintamuosius pagal galimybes derėtų aprašyti vienoje vietoje, atskirti nuo kitų kintamųjų ir pakomentuoti.

Kyla klausimas: ką daryti su nedaug vietos (vieną ar kelis baitus) užimančiais kintamaisiais?

Kintamųjų statusą visuomet reikia parinkti logiškai. Jei labai siekiama taupyti atmintinę, tuomet visi procedūroje naudojami kintamieji (pradiniai duomenys, rezultatai, vidiniai kintamieji), kuriuos tik įmanoma padaryti visuotiniais, ir turėtų būti visuotiniai.

Sprendžiant JPM uždavinius paprastai vartojamos nedidelės duomenų struktūros, todėl galima netaupyti atmintinės.

Uždaviniai

Nurodymas. Kai kuriems jums pateiktiems uždaviniams spręsti galima parašyti žymiai geresnį algoritmą, nenaudojant grįžimo metodą. Tačiau egzistuoja didelė uždavinių klasė, kur visų variantų perrinkimas yra neišvengiamas (pavyzdžiui, šachmatų žirgo uždaviniai). Kadangi vienas šio skyriaus tikslų – išmokti taikyti grįžimo metodą, o ne išspręsti duotus uždavinius bet kokiu būdu, *būtina visus* šio skyriaus *uždavinius* spręsti grįžimo metodu.

1. Duotas sveikas skaičius *m*. Skaitmenų eilutėje 123456789 tarp kai kurių skaitmenų reikia įterpti ženklus „+“ arba „–“ taip, kad gaunami reiškiniai būtų lygūs *m*. Pavyzdžiui, jei *m* = 1, tai reiškinys 1 + 2 – 34 + 56 – 7 – 8 – 9 tinka.

Parašyta programa, kuri atspausdina visus tokius reiškinius:

```
program ŽenklųIšdėstymas;
  var t : array[2..9] of string;  {išdėstytų ženklų masyvas}
      m: integer;
  procedure BandauDėtiŽenklą (skaitmuo, reiškinys : longint;
                              ženklas : char;
                              ženklo_vieta : integer);
  var naujas_reiškinys : longint;
      i: 2..9;
  begin
    if ... = '+'
      then naujas_reiškinys := ...
      else naujas_reiškinys := ...;
    if ženklo_vieta ...
      then
        begin
          t[ženklo_vieta] := ...; ...;
          t[ženklo_vieta] := ...; ...;
          t[ženklo_vieta] := ...; ...
        end
      else {lygybės tikrinimas}
        if ... = m
          then
            begin
              for i := 2 to 9  do ...;
              writeln('9');
            end;
  end;
begin
  readln(m);
  fillchar(t, sizeof(t), '');
  BandauDėtiŽenklą(1, 0, '+', 2)
end.
```

Užpildykite praleistas vietas (pažymėtos daugtaškiais).

<u>*Pastaba*</u>. Naudoti papildomus kintamuosius bei kitaip keisti parašytą tekstą <u>*negalima*</u>.

2. Duotas galvosūkis:

$$\begin{array}{r} ABCD \\ +\ ABCD \\ \hline EFGH \end{array}$$

, čia kiekviena skirtinga raidė žymi skirtingą skaitmenį.

Reikia pakeisti raides skaitmenimis, kad būtų teisinga duota lygybė.

Parašyta programa, randanti visus galimus sprendinius:

```
program galvosūkis;
```

```
  type aibė = set of 0..9;
  var sakitmuo, m, u, h, a : 0..9;
      i, n1, n2 : integer;
      S1, S2 : aibė;
      f : boolean;
  procedure spausdink(x, y : integer);
  begin
    write(x);
    write(' + ');
    write(x);
    write(' = ');
    writeln(y);
    writeln;
  end;
begin
  S1 := [];
  for m := 0 to 9 do
    begin
      S1 := S1+[m];
      for u := 0 to 9 do
        if u in S1
          then
            begin
              S1 := S1+[u];
              for h := 0 to 9 do
                if h in S1
                  then
                    begin
                      S1 := S1+[h];
                        for a := 0 to 9 do
                          if a in S1
                            then
                              begin
                                S1 := S1+[a];
                                n1 := 1000*m+100*u+10*h+a;
                                n2 := n1;
                                f := true;
                                S2 := [];
                                for i := 0 to 9 do
                                  begin
                                    skaitmuo := n2 mod 1;
                                    n2 := n2 mod 10;
                                    f := f and skaitmuo in s2;
                                    S2 := [sakitmuo] + S2;
                                  end;
                                if (S1+S2=[ ]) and f
                                  then
                                    spausdink(n1, 2 * n1);
                                S1 := S1-[a];
                              end;
                         S1 := S1-[h];
                    end;
              S1 := S1-[u];
        end;
      S1 := S1-[m];
    end;
  readln;
end.
```

Tačiau programoje yra klaidų. Ištaisykite jas ir parašykite programą paaiškinančius komentarus.

3. Parašykite *kuprinės uždavinio* sprendimo idėjos aprašą.

Kuprinės uždavinys. Yra n daiktų, sunumeruotų nuo 1 iki n. Kiekvienas daiktas turi savo vertę k_i ir masę m_i ($1 \le i \le n$).
Raskite vertingiausią (jei yra keli, bet kokį iš jų) daiktų rinkinį, kurio bendra masė neviršija *max* (bet kuriam i $max \ge m_i$).

4. Yra n domino kauliukų. Parašykite funkciją

```
ilgiausia(n: kiek; var A: kauliukai): kiek
```

kuri rastų ilgiausios, sudėtos pagal domino taisykles (kauliukai liečiasi vienodą taškų skaičių turinčiomis pusėmis), grandinės ilgį.

Ši funkcija gali turėti vidines procedūras ir (ar) funkcijas.

Naudotini tokie visuotinių kintamųjų tipų aprašai:

```
type kiek=1..28;
     kauliukai = array [0..6, 0..6] of boolean;
```

Šio skyriaus 1–4 uždavinių sprendimai turi būti pateikti JPM interneto svetainėje (http://ims.mii.lt/jpm/) iki 2009 m. kovo 8 d. 24 val.

Failo vardas turi būti *?????12.** (čia * – *doc, txt, odt arba rtf*). Vietoj klaustukų įrašykite pirmąsias penkias savo pavardės raides (be diakritinių ženklų). Jei kartais būtų pavardžių, trumpesnių negu penkios raidės, trūkstami simboliai keičiami pabraukimo brūkšniais „_“.

Elektroninio pašto adresas JPM antrosios dalies klausytojams: jpm.2kursas@gmail.com.