Šachmatai

Uždavinys priklauso tokiai uždavinių kategorijai, kuriems padaryti nereikia žinoti jokio ypatingesnio algoritmo. Užtenka pasirinkti tinkamas duomenų struktūras ir tvarkingai padaryti.

Perskaičius pradinius duomenis reikia įsiminti situaciją lentoje bei figūrų sąrašą. Apie kiekvieną lentos langelį žinome ar jame stovi figūra, ar ne. Jei stovi, įsiminsime figūros numerį. Kiekvieną figūrą apibūdina keli parametrai – jos spalva, tipas, langelis kuriame ji stovi. Žemiau pateikiame duomenų struktūras:

```
{ ---------------------------------- duomenų tipai ------------------------------- }
type koordinatės = record
                     x, y: integer;
                   end;
     spalva = (juoda, balta);
     tipas = (karalius, valdovė, bokštas, rikis,
              žirgas, pėstininkas);
     figūra = record
                k: koordinates;  { figūros koordinatės }
                s: spalva;       { spalva }
                t: tipas         { ir tipas }
              end;
     fig_sąr = array [0..64] of figūra;   { figūrų sąrašas }
                   { fig[0] – kuria figūra ir kur einama šacho atveju }
     lenta = array [1..8, 1..8] of integer;

{ ---------------------------------------- globalieji kintamieji ---------------------------------- }
var n: integer;     { figūrų skaičius }
    fig: fig_sąr;   { figūrų sąrašas }
    len: lenta;     { šachmatų lenta – kokiame langelyje kokia figūra }
```

Šį uždavinį moksleiviai sprendė olimpiadoje. Tačiau ne visi žino taisykles, nėra paprasta jas įsisavinti per trumpą laiką. Todėl sudėtingesniems atvejams (pavyzdžiui, kai pėstininkas yra pirmoje eilutėje, t. y. jau virtęs kita figūra ir pan.) nebuvo parinkti testai. Šių atvejų nenagrinėsime ir algoritme.

Pirma algoritmo dalis – tikrinimas, ar juodųjų karalius yra šachuojamas. Imamos visos baltos figūros ir paeiliui tikrinama, ar kuri nors figūra nepuola juodųjų karaliaus.

```
function ribose (x, y: integer): boolean;
{ ar koordinatės neišeina iš lentos ribų }
begin
  ribose := (x > 0) and (x < 9) and (y > 0) and (y < 9)
end; { ribose }

{ ---------------------------------- ar karalius šachuojamas ------------------------------- }
function yra_fig (x, y: integer; tp: tipas; sp: spalva)
                                                        : boolean;
{ ar pozicijoje (x, y) yra „tp“ tipo ir „sp“ spalvos figūra }
begin
  if len[x, y] = 0
  then yra_fig := false
  else yra_fig := (fig[len[x,y]].t = tp) and
                  (fig[len [x,y]].s = sp)
end; { yra_fig }

function šach_krypt (kor: koordinatės; dx, dy: integer)
                                                        : boolean;
{ tikrina, ar figūra, galinti judėti nurodyta kryptimi šachuoja karalių; }
{ jos judėjimo kryptį nusako dx ir dy – koordinačių pokyčiai }
begin
  kor.x := kor.x + dx;
  kor.y := kor.y + dy;
  while ribose(kor.x,kor.y) and (len[kor.x,kor.y] = 0) do
    begin                     { einame, kol sutinkame figūrą ar }
      kor.x := kor.x + dx;    { prieiname lentos kraštą }
      kor.y := kor.y + dy
    end;
  if ribose (kor.x, kor.y)
     then if yra_fig (kor.x, kor.y, karalius, juoda)
             then šach_krypt := true
             else šach_krypt := false { sutikome ne karalių }
     else šach_krypt := false { išėjome iš ribų nesutikę figūros }
end;  { šach_krypt }
```

```
function šach_pėst (kor: koordinates): boolean;
begin
  šach_pėst := ribose(kor.x-1, kor.y+1) and
               yra_fig(kor.x-1, kor.y+1, karalius, juoda) or
               ribose(kor.x+1, kor.y+1) and
               yra_fig(kor.x+1, kor.y+1, karalius, juoda)
end; { šach_pėst }

function šach_rikis (kor: koordinates): boolean;
begin
  šach_rikis :=
    šach_krypt(kor,-1,-1) or šach_krypt(kor,1,-1) or
    šach_krypt(kor,-1,1)  or šach_krypt(kor,1,1)
end; { šach_rikis }

function šach_bokštas (kor: koordinates): boolean;
begin
  šach_bokštas :=
   šach_krypt(kor,0,-1) or šach_krypt(kor,0,1) or
   šach_krypt(kor,-1,0) or šach_krypt(kor,1,0)
end; { šach_bokštas }

function šach_valdovė (kor : koordinates): boolean;
begin
  šach_valdovė := šach_rikis(kor) or šach_bokštas(kor)
end;   { šach_valdovė }

function šach_žirgas (kor: koordinates): boolean;
  var i, j: integer;
      šach: boolean;
begin
  šach := false;
  for i:=-2 to 2 do
   for j:=-2 to 2 do
    if (abs(i)+abs(j)=3) and ribose(kor.x+i, kor.y+j) and
       yra_fig(kor.x+i, kor.y+j, karalius, juoda)
       then šach := true;
  šach_žirgas := šach
end; { šach_žirgas }
```

```
function šachas: boolean;
{ tikrina, ar juodųjų karalius šachuojamas }
{ n, fig, len – globalieji kintamieji }
  var i: integer;
      šach: boolean;
begin
  šach := false;
  i := 1;
  while (i <= n) and not šach do
    begin
      with fig[i] do      { tikrina, ar nešachuoja kuri nors figūra }
        if s = balta
           then case t of
                  pėstininkas: šach := šach_pėst(k);
                  žirgas     : šach := šach_žirgas(k);
                  rikis      : šach := šach_rikis(k);
                  bokštas    : šach := šach_bokštas(k);
                  valdovė    : šach := šach_valdovė(k)
                end;
      i := i + 1
    end;
  šachas := šach
end; { šachas }
```

Jei juodųjų karalius šachuojamas, reikia nustatyti, ar juodiesiems matas, ar mato dar galima išvengti. Paeiliui imamos visos juodos figūros ir su kiekviena figūra atliekami visi galimi ėjimai bei kirtimai. Atlikus ėjimą, tikrinama, ar karalius tebėra šachuojamas, jei taip – figūra gražinama į ankstesnę padėtį, atstatoma nukirsta figūra (jei buvo kertama) ir bandoma daryti kitą ėjimą su ta pačia figūra, kol nebelieka galimų ėjimų. Tada bandoma eiti kita juoda figūra, kol išbandomi visi variantai.

```
{ ------------------------------------- tikrinama, ar matas ---------------------------------- }
function yra (x, y: integer; sp: spalva): boolean;
{ ar pozicijoje (x, y) yra bet kokia „sp“ spalvos figūra }
begin
  if len[x, y] = 0
     then yra := false
     else yra := fig[len [x, y]].s = sp
end; { yra }
```

```
function pasiseka (f, x, y: integer): boolean;
{ ar pasiseks išvengti mato, paėjus figūra f į langelį (x, y) }
  var buv_f: integer;
      buv_k: koordinates;
      nukirsta: figūra;
begin
  { patikriname, ar nebandoma kirsti savo spalvos figūros }
  buv_f := len[x, y]; { sužinome, kokia figūra stovi kertamame }
  if (buv_f = 0) or (fig[buv_f].s = balta)   { langelyje }
  then with fig[f] do
         begin
           buv_k := k;            { įsimename buvusias koordinates }
           len[k.x, k.y] := 0;  { atliekame ėjimą }
           k.x := x; k.y := y;
           len[x, y] := f;
           if buv_f <> 0          { bei nukertame figūrą }
              then begin
                     nukirsta := fig[buv_f];
                     fig[buv_f] := fig[n];
                     n := n - 1
                   end;
           if not šachas
              then begin  { šacho išvengta }
                     pasiseka := true;
                     fig[0] := fig[f];
                   end
              else begin  { grąžiname atgal figūras }
                     k.x := buv_k.x; k.y := buv_k.y;
                     len[k.x, k.y] := f;
                     len[x,y] := buv_f;  {atstatome nukirstą figūrą}
                     if buv_f <> 0
                        then begin
                               n := n + 1;
                               fig[n] := fig[buv_f];
                               fig[buv_f] := nukirsta
                             end;
                     pasiseka := false
                   end
         end
  else pasiseka := false
end;  { pasiseka }
```

```
function eik_krypt (f, dx, dy: integer): boolean;
{ figūra f, galinti juda nurodyta kryptimi, kol apgina karalių arba nebegali toliau eiti }
{ jos judėjimo kryptį nusako dx ir dy – koordinačių pokyčiai }
  var saugu: boolean;
      kor: koordinatės;
begin
  saugu := false;
  kor.x := fig[f].k.x + dx;
  kor.y := fig[f].k.y + dy;
  while ribose(kor.x,kor.y) and (len[kor.x,kor.y]=0) and
        not saugu do
    begin                                  { einame, kol sutinkame figūrą ar }
      saugu := pasiseka(f,kor.x,kor.y);  { prieiname lentos kraštą }
      kor.x := kor.x + dx;
      kor.y := kor.y + dy
    end;
  if ribose (kor.x, kor.y)   { jei sutikome figūrą }
     then saugu := pasiseka (f, kor.x, kor.y);
  eik_krypt := saugu
end; { eik_krypt }

function ėj_rikis (f: integer): boolean;
begin
  ėj_rikis := eik_krypt(f,-1,-1) or eik_krypt(f,1,-1) or
              eik_krypt(f,-1,1)  or eik_krypt(f,1,1)
end; { ėj_rikis }

function ėj_bokštas (f: integer): boolean;
begin
  ėj_bokštas := eik_krypt(f,0,-1) or eik_krypt(f,0,1) or
                eik_krypt(f,-1,0) or eik_krypt(f,1,0)
end; { ėj_bokštas }

function ėj_valdovė (f: integer): boolean;
begin
  ėj_valdovė := ėj_rikis(f) or ėj_bokštas(f)
end; { ėj_valdovė }
```

```
function ėj_žirgas (f: integer): boolean;
  var i, j: integer;
      kor: koordinatės;
      saugu: boolean;
begin
  kor := fig[f].k; { figūros koordinatės }
  saugu := false;
  for i:=-2 to 2 do
    for j:=-2 to 2 do
      if not saugu and (abs(i)+abs(j) = 3) and
        ribose(kor.x+i,kor.y+j) and
        pasiseka f,kor.x+i,kor.y+j)
        then saugu := true;
  ėj_žirgas := saugu
end; { ėj_žirgas }

function ėj_karalius (f: integer): boolean;
  var i, j: integer;
      kor: koordinatės;
      saugu: boolean;
begin
  kor := fig[f].k;
  saugu := false;
  for i := -1 to 1 do
    for j := -1 to 1 do
      if not saugu and (abs(i) + abs(j) <> 0) and
         ribose(kor.x+i, kor.y+j) and
         pasiseka(f, kor.x+i, kor.y+j)
         then saugu := true;
  ėj_karalius := saugu
end; { ėj_karalius }

function ėj_pėst (f: integer): boolean;
  var kor: koordinates;
begin
  kor := fig[f].k;
  ėj_pėst :=
   ribose(kor.x-1,kor.y-1) and yra(kor.x-1,kor.y-1,balta)
   and pasiseka(f,kor.x-1,kor.y-1)     { kertama į dešinę }
   or
   ribose(kor.x+1,kor.y-1) and yra(kor.x+1,kor.y-1,balta)
   and pasiseka(f,kor.x+1,kor.y-1)     { kertama į kairę }
   or
   ribose(kor.x,kor.y-1) and (len[kor.x,kor.y-1] = 0)
```

```
   and pasiseka(f,kor.x,kor.y-1)  { einama pirmyn per vieną langelį }
   or
   (kor.y=7) and (len[kor.x,5]=0) and (len[kor.x,6]=0)
   and pasiseka(f,kor.x,5)  { einama pirmyn per du langelius }
end;  { ėj_pėst }

function matas: boolean;
{ tikrina, ar juodųjų karaliui matas }
{ n, len, fig – globalieji kintamieji }
  var i: integer;
      saugu: boolean;
begin
  saugu := false;
  i := 1;
  while (i <= n) and not saugu do
    begin   { tikrinsime, ar galime išsigelbėti paėję i-tąją figūra }
      with fig[i] do
        if s = juoda   { jei tai juodoji figūra, bandome ja paeiti }
        then case t of   { sėkmingas ėjimas įsimenamas fig[0] }
               pėstininkas: saugu := ėj_pėst(i);
               žirgas     : saugu := ėj_žirgas(i);
               rikis      : saugu := ėj_rikis(i);
               bokštas    : saugu := ėj_bokštas(i);
               valdovė    : saugu := ėj_valdovė(i);
               karalius   : saugu := ėj_karalius(i)
             end;
        i := i + 1;
    end;
  matas := not saugu
end;  { matas }
```