PRESEK
List za mlade matematike, fizike, astronome in računalnikarje

ISSN 0351-6652
Letnik **30** (2002/2003)
Številka 3
Strani 138–141

Martin Juvan:

ŠTEVILSKI LABIRINT

Ključne besede: računalništvo, rekreacijska matematika, naravna števila, pravokotna tabela.

Elektronska verzija: http://www.presek.si/30/1519-Juvan.pdf

ŠTEVILSKI LABIRINT

Pred časom sem naletel na naslednjo različico iskanja izhoda iz labirinta. Namesto običajne slike labirinta dobimo pravokotno tabelo naravnih števil, na primer tako, kot je prikazana na sliki 1. Na začetku se nahajamo na enem od polj tabele, recimo v levem zgornjem vogalu. Naš cilj je priti do nekega izbranega polja tabele, ki predstavlja izhod iz labirinta. Pri tabeli s slike 1 naj bo to desni spodnji vogal. Premikanje po labirintu poteka takole: Če smo na polju, na katerem je napisano število k, potem se lahko premaknemo za k mest v desno, za k mest v levo, za k mest navzgor ali za k mest navzdol. Seveda pa pri tem ne smemo stopiti čez rob tabele.

3	2	3	4	3	3
1	3	3	2	1	3
5	3	3	2	2	4
1	3	3	3	3	3
1	4	2	3	2	5

Slika 1. Številski labirint.

Za labirint s slike 1 je ena od možnih poti do izhoda prikazana na sliki 2. Seveda pa pri vseh številskih labirintih ni moč priti do izhoda. Tako na primer labirint s slike 1 nima rešitve, če mu izhod prestavimo v desni zgornji vogal.

⓪			①		
			③		④
			②		⑤

Slika 2. Pot skozi labirint s slike 1.

In kako se lotiti iskanja izhoda iz številskega labirinta? Seveda se iskanja lahko lotimo s poskušanjem in se pri tem zanašamo na navdih ter zaupamo v srečo. Če pa želimo zanesljivo pot do rešitve, je priporočljivo, da se reševanja lotimo bolj urejeno in sistematično.

Ena od možnih poti je naslednja: Polja tabele bomo postopoma oštevilčili tako, da bo številka, prirejena posameznemu polju, povedala, koliko potez smo potrebovali, da smo z začetnega prišli do tega polja. Začetno polje označimo s številko 0. Nato poljem, na katera lahko pridemo s polja številka 0, dodelimo številko 1. V naslednjem koraku še neoštevilčenim poljem, na katera lahko pridemo s polj, ki imajo številko 1, dodelimo številko 2. Postopek ponavljamo, dokler številke ne dobi tudi polje, na katerem je izhod iz labirinta. Številka, dodeljena izhodu, pove, koliko potez potrebujemo, da pridemo iz labirinta. Vendar pa se postopek vedno ne izide. Lahko se namreč zgodi, da na nekem koraku ne moremo več priti na nobeno neoštevilčeno polje, do polja, na katerem je izhod, pa še nismo prišli. V takem primeru poti iz labirinta ni.

0	3		1	4	
	4		3		4
2	4				3
1	2		4	3	
2	3		2		4

Slika 3. Iskanje poti skozi labirint.

Če opisani postopek uporabimo na labirintu s slike 1, dobimo tabelo števil s slike 3. Številka 4 v desnem spodnjem vogalu pove, da je iz labirinta moč priti v štirih potezah, torej potezo hitreje, kot to naredimo z rešitvijo s slike 2.

In kako poiščemo poteze, ki jih moramo narediti, da pridemo iz labirinta? Verjetno je najlažje, da si jih beležimo kar sproti. Vsakič, ko neko polje oštevilčimo, si tudi zapomnimo, s katerega polja (z za eno manjšo številko) smo nanj prišli. Včasih je takih polj več. Takrat si zapomnimo katerokoli od njih. Upoštevajte nasvet in poiščite štiri poteze, ki nas pripeljejo do izhoda iz labirinta s slike 1.

Ko se reševanja s svinčnikom in papirjem naveličamo, lahko gornji postopek zapišemo tudi v obliki računalniškega programa. V nadaljevanju je zapisana le funkcija, ki opravi glavni del računanja, branje podatkov o labirintu in izpis ugotovitev pa boste morali dodati sami.

```
int resi(int lab[MAX][MAX], int m, int n, int pot[MAX][MAX][3])
{
  int i, j, poteza, konec, t;

  for (i = 0; i < m; i++)
    for (j = 0; j < n; j++)
      pot[i][j][0] = pot[i][j][1] = pot[i][j][2] = -1;
  /* začetni položaj in začetna poteza */
  poteza = pot[0][0][0] = 0;
  do {
    konec = 1;
    for (i = 0; i < m; i++)
      for (j = 0; j < n; j++)
        if (pot[i][j][0] == poteza) {
          /* gremo na levo */
          if ((t = j - lab[i][j]) >= 0 && pot[i][t][0] == -1) {
            pot[i][t][0] = poteza + 1; konec = 0;
            pot[i][t][1] = i; pot[i][t][2] = j;
          }
          /* gremo na desno */
          if ((t = j + lab[i][j]) < n && pot[i][t][0] == -1) {
            pot[i][t][0] = poteza + 1; konec = 0;
            pot[i][t][1] = i; pot[i][t][2] = j;
          }
          /* gremo navzgor */
          if ((t = i - lab[i][j]) >= 0 && pot[t][j][0] == -1) {
            pot[t][j][0] = poteza + 1; konec = 0;
            pot[t][j][1] = i; pot[t][j][2] = j;
          }
          /* gremo navzdol */
          if ((t = i + lab[i][j]) < m && pot[t][j][0] == -1) {
            pot[t][j][0] = poteza + 1; konec = 0;
            pot[t][j][1] = i; pot[t][j][2] = j;
          }
        }
    poteza++;
  } while (pot[m - 1][n - 1][0] == -1 && !konec);
  return !konec;
}
```

Funkcija vrne vrednost 1, kadar obstaja pot do izhoda iz labirinta, sicer vrne vrednost 0. Komponenti 1 in 2 v tabeli `pot` povesta, od kod smo prišli na dano polje, komponenta 0 pa, koliko potez smo potrebovali.

Funkcija ni napisana najbolj varčno. Z dodatno pomožno funkcijo bi njeno kodo lahko nekoliko skrajšali, še bolj moteča pa je potratnost izvedbe postopka. V vsaki ponovitvi zanke `do` namreč pregledamo celotno tabelo, čeprav je v njej morda le nekaj polj, iz katerih lahko naredimo nove poteze. Ta polja bi si lahko na vsakem koraku zabeležili v pomožno enorazsežno tabelo in se tako izognili vsakokratnemu potratnemu preverjanju vseh polj. Tisti, ki poznate osnove algoritmov, ste v gornjem postopku verjetno prepoznali eno od različic pregleda grafa v širino.

Končajmo z nalogo. Na sliki 4 vas čaka "še neraziskan" številski labirint. Začnete v levem zgornjem vogalu, izhod pa je v desnem spodnjem vogalu. Seveda iščete tako pot do izhoda, ki zahteva čim manj potez.

6	1	2	1	4	1	2	1	2	1
1	4	4	2	2	2	1	4	4	6
2	1	4	1	4	1	4	2	4	1
1	6	1	2	1	4	1	4	1	2
6	8	6	1	2	2	4	6	4	4
1	2	1	2	1	6	1	6	1	4
6	1	2	4	2	4	4	4	5	1

Martin Juvan