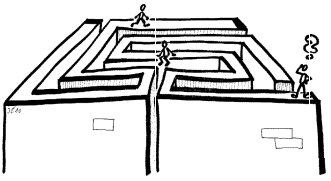
INFORMATIKA

Zeď

REDAKCE



V článku [2] jsme si před několika lety ukázali řešení Bludiště jako příklad průchodu grafem do šířky, viz např. [1]. V jedné z informatických soutěží studentů v Bosně a Hercegovině byla zadána úloha také o pohybu v bludišti, ale tentokrát se nemělo chodit volnými chodbami, ale nahoře po zdi. Uveďme si nejprve její znění:

V textovém souboru Zdi.dat je zadána matice velikosti nejvýše 50×50, jejímiž prvky jsou jen 0 a 1. Prvek 0 znamená „prázdný prostor“ a prvek 1 znamená „zeď“. Napište program, který zjistí a sdělí, jestli je horní levý roh spojen zdí s dolním pravým rohem matice, tj. lze-li se z horního levého vrcholu dostat po zdi do pravého dolního tak, aby se přecházelo vždy na sousední pole s hodnotou 1.

Při analýze úlohy zjistíme, že jde o velmi příbuzný problém (v podstatě týž) jako při hledání cesty z bludiště, resp. jako při řešení problému, zda leží dva zvolené uzly grafu ve stejné komponentě souvislosti. Nejprve uveďme program jednoho úspěšného soutěžícího.

```
program Zed;

{Zda je levy horni roh spojen s pravym dolnim}

var
  N, M, I, K: Integer;
  A: array [0..51,0..51] of Integer;
```

```
  C: Char;
  F: Text;
  Konec: Boolean;

function DalsiKroky(S: Integer): Integer;
var
  R, I, K: Integer;
begin
  R := 0;
  for I := 0 to N do
    for K := 0 to M do
      if A[I,K] = S - 1 then
      begin
        if A[I-1,K] < 0 then
        begin
          A[I-1,K] := S; R := R + 1
        end;
        if A[I+1,K] < 0 then
        begin
          A[I+1,K] := S; R := R + 1
        end;
        if A[I,K-1] < 0 then
        begin
          A[I,K-1] := S; R := R + 1
        end;
         if A[I,K+1] < 0 then
        begin
          A[I,K+1] := S; R := R + 1
        end
      end;
  DalsiKroky := R
end;

begin {program}
  for I := 0 to 51 do
    for K := 0 to 51 do
      A[I,K] := 0;
  assign(F,'Zdi.dat');
```

```
  reset(F);
  I := 1; K := 0;
  while not Eof(F) do
  begin
    Read(F, C);
    if C in ['0','1'] then
    begin
      K := K + 1; N := I; M := K;
      A[I, K] := Ord('0') - Ord(C)
    end;
    if Eoln(F) then
    begin
      I := I + 1; K := 0
    end
  end;
  A[1,1] := -A[1,1];
  I := 2;
  repeat
    Konec := true;
    if A[N,M] > 0 then
      WriteLn('Cesta existuje')
    else
      if DalsiKroky(I) = 0 then
        WriteLn('Cesta neexistuje')
      else
      begin
        I := I + 1;
        Konec := false
      end
  until Konec;
  ReadLn
end.  {program}
```

Program načte vstupy s opačným znaménkem a kolem načtené matice A o M řádcích a N sloupcích nechává „ohradu z nul“, aby při testování okolí prvků matice nedošlo k „opuštění prostoru“ matice a tedy k chybám. Program pak první prvek vezme s opačným (kladným) znaménkem, tj. jako 1 a dále se postupuje po krocích. V každém kroku se příslušné „sousední“

prvky -1 změní na číslo kroku, tj. po k-tém kroku je číslo k na těch místech zdi, kam se lze od počátečního bodu dostat k-tým krokem. Program má časovou složitost $O(M^2 \cdot N^2)$ a nevýhodně při každém kroku prochází celou maticí A. Z drobnějších opomenutí lze doporučit ve funkci *DalsiKroky* začínat oba for-cykly až od 1 a upravit práci s proměnnou R. Ta v každém kroku říká, kolika směry lze v cestě po zdi pokračovat, ale této informace se nijak nevyužívá, takže je zbytečná. Stačilo by hodnoty funkce *DalsiKroky* definovat jako Boolean a v programu pak skončit při hodnotě *false*.

Následující program byl vyhotoven ve stylu [1] a podle pravidel průchodu grafu do šířky vytváří frontu prvků, z nich může cesta po zdi pokračovat. Oproti předchozímu řešení jsme tedy vyměnili čas za paměť – program potřebuje navíc paměť na uložení fronty, ale s jejím využitím pak pracuje o dost efektivněji.

```
program ZedX;
{Zda mezi levym hornim a pravym dolnim rohem je souvisla zed}

type
  Tady = record
           Ra, Sl: Integer
         end;
const
  MaxInd = 50;
  Tah: array [1..4] of
         record
           Ra, Sl: Integer
         end
         = ((Ra: 0; Sl: 1), (Ra: -1; Sl: 0),
            (Ra: 0; Sl: -1), (Ra: 1; Sl: 0));
  Start: Tady = (Ra: 1; Sl: 1);

var
  Zed: array [1..MaxInd, 1..MaxInd] of Integer;
  Fronta: array [1..MaxInd * MaxInd] of
            record
              Misto: Tady;
              Krok: Integer
            end;
```

```
  ZacFr, KonFr, Krok, Smer, M, N, I, J: Integer;
  Tu1, Tu2: Tady;
  SouvislaZed: Boolean;

function JeVArea(Je: Tady): Boolean;
begin
  if ((Je.Ra >= 1) and (Je.Ra <= M) and
    (Je.Sl >=1) and (Je.Sl <= N)) then
    JeVArea := true
  else
    JeVArea := false
end;

procedure VstupZdi;
var
  DatS: Text;
  C: Char;
begin
  assign(DatS, 'Zdi.dat');
  reset(DatS);
  M := 0;
  repeat
    Inc(M); N := 0;
    repeat
      Inc(N); Read(DatS, C);
      Zed[M,N] := Ord('0') - Ord(C);
    until EoLn(DatS);
    ReadLn(DatS)
  until Eof(DatS);
  close(DatS)
end;

procedure NajdiCestu;
begin
  Zed[Start.Ra,Start.Sl] := 1;
  ZacFr := 1; KonFr := 1;
  Fronta[1].Misto := Start; Fronta[1].Krok := 1;
  while ((ZacFr <= KonFr) and (not SouvislaZed)) do
```

```
  begin
    Tu1 := Fronta[ZacFr].Misto;
    Krok := Fronta[ZacFr].Krok + 1;
    Inc(ZacFr);
    for Smer := 1 to 4 do
    begin
      Tu2.Ra := Tu1.Ra + Tah[Smer].Ra;
      Tu2.Sl := Tu1.Sl + Tah[Smer].Sl;
      if (JeVArea(Tu2) and (Zed[Tu2.Ra, Tu2.Sl] = -1)) then
      begin
        Zed[Tu2.Ra, Tu2.Sl] := Krok; Inc(KonFr);
        Fronta[KonFr].Misto := Tu2;
        Fronta[KonFr].Krok := Krok;
        if ((Tu2.Ra = M) and (Tu2.Sl = N)) then
        begin
          SouvislaZed := true; exit
        end
      end
    end
  end
end;  {NajdiCestu}

begin  {program}
  VstupZdi;
  SouvislaZed := false;
  if Zed[Start.Ra,Start.Sl] = -1 then
    NajdiCestu;
  if SouvislaZed then
    WriteLn('Existuje souvisla zed.')
  else
    WriteLn('Souvisla zed neexistuje.');
  ReadLn
end.  {program}
```

Kdybychom si vytiskli matici A ze souboru Zdi.dat a pak změněnou matici na konci programu, dostali bychom například

```
1  1  1  0  1  0      1  2  3  0 -1  0
1  0  1  1  0  1      2  0  4  5  0  9
1  0  0  1  1  1      3  0  0  6  7  8
1  1  1  0  1  0      4  5  6  0  8  0
0  1  0  1  1  0      0  6  0 10  9  0
1  0  0  1  0  1     -1  0  0 11  0 -1
1  1  0  1  1  1     -1 -1  0 12 13 14
```

Hledání cesty končí po dosažení cílového bodu (proto prvek [6,6] má hodnotu -1 a nikoli 15). Všimněme si, že omezení okolí u „hraničních prvků“ není řešeno pásmy nul, jako u předchozího programu, ale funkcí *JeVArea*.

Také tento druhý program by bylo možné ještě vylepšit. Ve frontě se zde vedle souřadnic polí naprosto zbytečně ukládá vždy i číslo kroku, v němž jsme příslušné pole dosáhli. Číslo kroku je přitom uloženo i v poli *Zed*, odkud ho můžeme snadno na základě známých souřadnic pole získat. Funkce *JeVArea* je v programu zapsána maximálně srozumitelně. Kratší zápis a efektivnější kód ovšem získáme, když v jejím těle namísto podmíněného příkazu použijeme přímé dosazení hodnoty logického výrazu do návratové hodnoty funkce.

V proceduře *NajdiCestu* se zbytečně opakovaně vyhodnocuje složená podmínka v hlavním while-cyklu. Postačovala by zde přitom jednoduchá podmínka `ZacFr <= KonFr`, neboť při dosazení hodnoty *true* do proměnné *SouvislaZed* je tato procedura ihned ukončena voláním standardní procedury *exit*. Z hlediska celkového návrhu je vhodné zlepšit dekompozici a uspořádání programu. V současném programu jsou použity procedury *VstupZdi* a *NajdiCestu* bez parametrů, které se svým okolím komunikují prostřednictvím globálních proměnných. Takové řešení sice dobře funguje a programátorovi se u takto malé úlohy také snadno zapíše, jde ovšem proti duchu strukturovaného programování. Při správné dekompozici by měly mít všechny procedury jasně definováno rozhraní ve formě parametrů a ke komunikaci by měly používat právě jenom své parametry. Každá procedura by navíc měla pracovat pouze s lokálními pomocnými proměnnými.

Na závěr si tedy ukážeme ještě jednou program založený na procházení do šířky pomocí fronty, ve kterém oproti předchozímu řešení patřičně upravíme všechny zmíněné nedostatky.

```
program ZedX;
{Zda mezi levym hornim a pravym dolnim rohem je souvisla zed}

const
  MaxInd = 50;

type
  Bludiste = array [1..MaxInd, 1..MaxInd] of Integer;
  Tady = record
           Ra, Sl: Integer
         end;

var
  Zed: Bludiste;
  M, N: Integer;

function JeVArea(Je: Tady; M, N: Integer): Boolean;
begin
  JeVArea := (Je.Ra >= 1) and (Je.Ra <= M) and
    (Je.Sl >=1) and (Je.Sl <= N)
end;

procedure VstupZdi(var Zed: Bludiste; var  M, N: Integer);

var
  DatS: Text;
  C: Char;
begin
  assign(DatS, 'Zdi.dat');
  reset(DatS);
  M := 0;
  repeat
    Inc(M); N := 0;
    repeat
      Inc(N); Read(DatS, C);
      Zed[M,N] := Ord('0') - Ord(C);
    until EoLn(DatS);
```

```
    ReadLn(DatS)
  until Eof(DatS);
  close(DatS)
end;

function NajdiCestu(var Zed: Bludiste; M, N: Integer):
  Boolean;
const
  Tah: array [1..4] of
         record
           Ra, Sl: Integer
         end
         = ((Ra: 0; Sl: 1), (Ra: -1; Sl: 0),
            (Ra: 0; Sl: -1), (Ra: 1; Sl: 0));
  Start: Tady = (Ra: 1; Sl: 1);
var
  Fronta: array [1..MaxInd * MaxInd] of Tady;
  ZacFr, KonFr, Krok, Smer: Integer;
  Tu1, Tu2: Tady;
begin
  NajdiCestu := false;
  if Zed[Start.Ra,Start.Sl] = 0 then exit;
  Zed[Start.Ra,Start.Sl] := 1;
  ZacFr := 1; KonFr := 1;
  Fronta[1] := Start;
  while ZacFr <= KonFr do
  begin
    Tu1 := Fronta[ZacFr];
    Krok := Zed[Tu1.Ra,Tu1.Sl] + 1;
    Inc(ZacFr);
    for Smer := 1 to 4 do
    begin
      Tu2.Ra := Tu1.Ra + Tah[Smer].Ra;
      Tu2.Sl := Tu1.Sl + Tah[Smer].Sl;
      if (JeVArea(Tu2,M,N) and (Zed[Tu2.Ra,Tu2.Sl]=-1)) then
      begin
        Zed[Tu2.Ra, Tu2.Sl] := Krok;
        Inc(KonFr);
```

```
        Fronta[KonFr] := Tu2;
        if (Tu2.Ra = M) and (Tu2.Sl = N) then
        begin
          NajdiCestu := true; exit
        end
      end
    end
  end
end;  {NajdiCestu}

begin  {program}
  VstupZdi(Zed, M, N);
  if NajdiCestu(Zed, M, N) then
    WriteLn('Existuje souvisla zed.')
  else
    WriteLn('Souvisla zed neexistuje.');
  ReadLn
end.  {program}
```

Literatura

[1] *Töpfer, P.*: Algoritmy a programovací techniky. Prometheus, Praha 1995 a 2007.
[2] *Trávníček, S.*: Nejkratší cesta z bludiště. MFI, 15 (2005/06), č. 8, str. 484–491.

(Autorkou úvodní ilustrace je *Mgr. Jaroslava Čermáková* ze Suchých Lazců.)

Úlohy nového typu na IOI

PAVEL TÖPFER

Matematicko-fyzikální fakulta UK, Praha

Předposlední, tj. 21. ročník Mezinárodní olympiády v informatice (IOI – International Olympiad in Informatics) přinesl jednu zajímavou novinku. Vedle tří tradičních, poměrně obtížných soutěžních úloh, byla v každém soutěžním dnu zadána navíc jedna úloha výrazně snadnější. Cílem této úpravy bylo, aby i soutěžící s nižšími programátorskými schopnostmi a znalostmi měli reálnou šanci v soutěži částečně uspět a vyřešit vždy alespoň tuto jednu úlohu. Pro ty nejlepší bylo naopak vyřešení této úlohy samozřejmostí a příliš je nezdržela od práce nad náročnějšími problémy. Přidaná lehčí úloha byla navíc vyhodnocována ve zvláštním režimu ihned v průběhu soutěže. Zatímco u tradičních úloh nemají soutěžící okamžitou odezvu a výsledky se dozvědí až po skončení soutěžního dne, tuto novou úlohu mohou odevzdat kdykoliv během soutěže a v krátké době dostanou od vyhodnocovacího systému zprávu, jak uspěli. Podle výsledku testů pak mají možnost svoje řešení opravit a odevzdat ho k vyhodnocení znovu, třeba i postupně vícekrát. Tím se úloha stává ještě snadnější ve srovnání s úlohami, kde přímá odezva chybí. Obdobným způsobem se odevzdávají a vyhodnocují rovněž praktické úlohy domácího kola naší Matematické olympiády – kategorie P.

Změna skladby soutěžních úloh se na výsledcích 21. ročníku IOI projevila příznivě. Na rozdíl od minulých let bylo tentokrát jen velmi málo účastníků, kteří v soutěži nevyřešili ani jednu ze zadaných úloh. Vedení Mezinárodní olympiády v informatice proto rozhodlo, že tuto úpravu zachová v platnosti i do příštích let.

Tradiční úlohy zadávané na IOI jsou určeny pro nejtalentovanější studenty, bývají značně náročné a nelze je proto využít v běžné výuce programování na střední škole. Naproti tomu tyto nové úlohy žádné zvláštní znalosti nebo zkušenosti nevyžadují a může je bez problémů řešit každý středoškolák, který zvládl základy programování. Seznámíme vás proto nyní s oběma úlohami nového typu, které se objevily na 21. ročníku Mezinárodní olympiády v informatice (Bulharsko, Plovdiv, srpen 2009). Uve-

deme zde úplné zadání každé z úloh a rovněž komentář, jakým způsobem je možné úlohy řešit. Jednu z úloh si navíc předvedeme i s úplným ukázkovým řešením.

⋆⋆⋆

Plovdivská olympiáda v informatice

Místní Plovdivská olympiáda v informatice (POI) má následující neobvyká pravidla. Soutěží v ní N účastníků a mají za úkol řešit T soutěžních úloh. Každé odevzdané řešení úlohy je testováno s jedinými vstupními daty, takže za každou úlohu získá každý soutěžící buď plný počet bodů, nebo nic. Žádné částečné hodnocení neexistuje.

Počet bodů přidělených za správně vyřešenou úlohu se stanoví až po skončení soutěže a je roven počtu soutěžících, kteří tuto úlohu nevyřešili. Výsledné hodnocení soutěžícího je stanoveno jako součet bodů, které soutěžící získal za všechny jím vyřešené úlohy.

Filip se zúčastnil soutěže, ale je zmaten složitými pravidly bodování a nedokáže určit svoje umístění v celkovém pořadí POI. Pomozte Filipovi a napište mu program, který určí jeho celkové hodnocení a výsledné umístění v soutěži.

Před zahájením soutěže obdrželi účastníci jednoznačná identifikační čísla od 1 do N. Filip má číslo P. Ve výsledkové listině jsou soutěžící uspořádáni sestupně podle počtu získaných bodů. V případě shody bodů je ve výsledkové listině umístěn dříve ten soutěžící, který vyřešil více úloh. Pokud nerozhodne ani toto kritérium, soutěžící se stejnými výsledky budou uspořádáni vzestupně podle svých identifikačních čísel.

Úloha

Napište program, který určí Filipovo výsledné hodnocení a jeho pořadí ve výsledkové listině na základě informací, který soutěžící vyřešil kterou úlohu.

Omezení

$1 \leq N \leq 2\,000$	počet účastníků soutěže
$1 \leq T \leq 2\,000$	počet řešených úloh
$1 \leq P \leq N$	Filipovo identifikační číslo

Vstup

Váš program musí přečíst ze standardního vstupu následující údaje:

- První řádek obsahuje tři celá čísla N, T, P oddělená vždy jednou mezerou.
- Dalších N řádků popisuje, které úlohy vyřešil který soutěžící. V pořadí k-tý z těchto řádků určuje, které úlohy vyřešil soutěžící s identifikačním číslem k. Každý takový řádek obsahuje T celých čísel oddělených mezerami. První z čísel na řádku udává, zda soutěžící k vyřešil první úlohu, druhé určuje, zda vyřešil druhou úlohu, atd. Každé z těchto T čísel je rovno 0 nebo 1, přičemž 1 znamená vyřešenou úlohu a 0 znamená nevyřešenou úlohu.

Výstup

Váš program musí zapsat na standardní výstup jediný řádek, který obsahuje dvě celá čísla oddělená jednou mezerou. První z nich představuje výsledný počet bodů, které Filip získal v soutěži. Druhé z čísel na výstupu určuje Filipovo pořadí ve výsledkové listině. Je to číslo z rozmezí od 1 do N, kde 1 znamená umístění na prvním místě (tj. soutěžící, který dosáhl nejvyššího bodového hodnocení) a N umístění na posledním místě (tj. soutěžící s nejnižším hodnocením).

Hodnocení

V testovacích datech odpovídajících celkovému ohodnocení 35 body nebude mít žádný soutěžící v POI stejný výsledný počet bodů jako Filip.

Příklad

Vstup	**Výstup**
5 3 2 0 0 1 1 1 0 1 0 0 1 1 0 1 1 0	3 2

První úlohu nevyřešil jediný soutěžící, takže bude hodnocena 1 bodem. Druhou úlohu nevyřešili dva soutěžící, bude tedy hodnocena 2 body. Třetí úlohu nevyřešili čtyři soutěžící, takže je hodnocena 4 body. První soutěžící tudíž získal celkem 4 body; druhý soutěžící (Filip), čtvrtý a pátý soutěžící mají všichni po 3 bodech; třetí soutěžící má 1 bod. Soutěžící s identifikačními čísly 2, 4 a 5 mají nejen shodný počet bodů, ale i stejný počet vyřešených úloh, takže jejich vzájemné pořadí bude určeno podle identifikačních čísel. Filip bude tudíž ve výsledkové listině POI na druhém místě, hned za soutěžícím s číslem 1.

⋆⋆⋆

Řešení této úlohy je zcela přímočaré, nevyžaduje použití žádných složitějších datových struktur nebo netriviálních algoritmů. Vzhledem k zadaným omezením na velikost vstupních dat si můžeme všechny vstupní údaje přečíst a uložit jednoduše do dvojrozměrného pole (řádky pole odpovídají soutěžícím, sloupce úlohám). Při prvním průchodu polem po sloupcích spočítáme pro každou z úloh, kolik soutěžících ji nevyřešilo. Tím budeme znát počty bodů, jimiž jsou jednotlivé úlohy ohodnoceny. Tyto počty si uložíme do dalšího pole (jednorozměrného, indexovaného čísly úloh). Jinou možností by bylo počítat bodová hodnocení úloh průběžně již při načítání vstupních dat. Druhý průchod polem po řádcích nám pak stačí k tomu, abychom pro každého soutěžícího určili jeho celkové bodové hodnocení a počet vyřešených úloh.

Pro určení Filipova pořadí ve výsledkové listině nemusíme sestavovat celou výsledkovou listinu, tzn. nemusíme řadit do výsledného pořadí všechny účastníky soutěže. Místo toho stačí zjistit počet soutěžících, kteří se umístili před Filipem. Tento počet určíme tak, že si nejprve spočítáme výsledky Filipa a potom spočítané výsledky každého dalšího soutěžícího ihned porovnáme s výsledky Filipa. Soutěžící X se ve výsledkové listině umístil před Filipem právě tehdy, když

- X má vyšší počet bodů, než Filip; nebo
- X má stejný počet bodů jako Filip, ale vyřešil více úloh; nebo
- X má stejný počet bodů i stejný počet vyřešených úloh jako Filip, ale má nižší identifikační číslo.

(*Pokračování*)