INFORMATIKA

Známosti na večírku (Úlohy z MO – kategorie P, 19. část)

PAVEL TÖPFER
Matematicko-fyzikální fakulta UK, Praha

Dnešní úlohu jsme vybrali z krajského kola 47. ročníku Matematické olympiády – kategorie P (školní rok 1997/1998). Pod vnější slupkou povídání o známostech na večírku snadno odhalíme úlohu z oblasti teorie grafů. Při řešení takovýchto úloh se nám obvykle vyplatí nevymýšlet celé řešení od základu sami, ale znát a využívat některé standardní algoritmy. Není přitom ovšem zapotřebí ovládat podrobně celou rozsáhlou teorii, často vystačíme se zkušeností s řešením úloh podobného typu. Nejinak tomu bude i v našem případě. Jak uvidíte, zde se bude jednat o úlohu nalezení největšího párování v bipartitním grafu. Algoritmus řešení tohoto problému lze nalézt v odborných publikacích o teorii grafů, je založen na převedení tohoto problému na hledání maximálního toku v síti. My se však této teorii vyhneme a podíváme se na problém očima řešitele, který je sice dobrým programátorem, ale žádné algoritmy na hledání toků v síti nezná. Nejprve se ale seznámíme s přesným zadáním úlohy:

⋆⋆⋆

Na večírku se sešlo několik hostů. Víme, kteří z nich se vzájemně znají a kteří ne. Vztah „znát se“ uvažujeme zásadně jako symetrický, tzn. o libovolné dvojici lidí platí, že buď se navzájem znají, nebo ani jeden z nich nezná toho druhého. Hosté se rozhodli vytvořit takové taneční páry, aby

spolu tančili vždy muž se ženou, kteří se spolu znají. Navrhněte algoritmus, který určí, kolik párů nejvýše může tančit zároveň.

Vstupem programu je počet mužů M, $M \leq 100$, a počet žen Z, $Z \leq \leq 100$, a dále seznam těch dvojic hostí, kteří se spolu znají. Pro jednodušší zadávání vstupu si jednotlivé muže označíme kladnými celými čísly od 1 do M a ženy zápornými celými čísly od -1 do $-Z$, takže na vstupu bude zadán seznam dvojic celých čísel. Výsledkem bude jediné číslo udávající maximální počet současně tančících párů.

Příklad: Pro vstupní data $M = 2$, $Z = 2$ a dvojice známých $(1, -1)$, $(1, 2)$, $(2, -1)$, $(-1, -2)$ bude výsledkem číslo 1, neboť tančit může vždy jen jediný pár – buď $(1, -1)$ nebo $(2, -1)$.

⋆⋆⋆

Jak jsme již naznačili v úvodu, jedná se o standardní úlohu teorie grafů nalézt co největší párování v bipartitním grafu. V grafu může existovat více různých párování téže maximální velikosti, výsledkem úlohy však je pouze velikost tohoto párování, která je určena jednoznačně. Zmínili jsme zde pojmy bipartitní graf a párování, tak si je nejprve trochu vysvětlíme.

Bipartitním nazýváme takový neorientovaný graf, ve kterém můžeme rozdělit všechny jeho vrcholy do dvou disjunktních množin A, B tak, že každá hrana v grafu spojuje nějaký vrchol z množiny A s nějakým vrcholem z množiny B. Vrcholy grafu v naší úloze budou představovat jednotlivé hosty na večírku. Do množiny A zařadíme vrcholy odpovídající mužům a do množiny B vrcholy odpovídající ženám. Hrany v grafu představují známosti mezi muži a ženami. Vrchol reprezentující nějakého muže je spojen hranou s vrcholem reprezentujícím nějakou ženu právě tehdy, pokud se tento muž a žena znají a mohou tedy spolu tančit. Podle zadání úlohy je vztah „znát se“ symetrický, takže všechny hrany v grafu budou neorientované. Jsou-li na vstupu zadány také dvojice mužů, kteří se navzájem znají, jsou tyto vstupní údaje ignorovány a v grafu se nijak neprojeví, neboť pro řešení úlohy nemají žádný význam. Totéž se týká i dvojic žen, které se navzájem znají. Společnost na večírku je takto reprezentována bipartitním grafem, ve kterém muži tvoří jednu skupinu vrcholů grafu a ženy druhou.

Párováním v grafu rozumíme takovou množinu hran grafu, v níž žádné dvě nemají společný vrchol. Největším párováním v grafu nazveme párování, které je tvořeno co největším počtem hran. Velikost největšího párování (tzn. počet jeho hran) v našem grafu určuje největší počet zároveň tančících párů na večírku a je tedy hledaným výsledkem.

Úlohu bychom mohli řešit primitivním způsobem tak, že bychom ze všech hran grafu vybírali různé podmnožiny a hledali bychom mezi nimi takovou, která je tvořena co nejvíce hranami a přitom žádné dvě z nich nemají společný vrchol. Takovéto slepé zkoušení všech možností je sice teoreticky správné, opravdu najde největší párování v grafu, ale je velmi neefektivní a pro rozsáhlejší data je proto tento postup nevhodný.

Standardní algoritmus řešení naší úlohy vychází od nějakého (libovolného) výchozího párování, které se pak postupně vylepšuje. Dokud to půjde, v každém kroku výpočtu zvětšíme velikost párování o jednu hranu. Když už párování zvětšit nelze, máme výsledek. Za výchozí párování můžeme vzít jednoduše prázdnou množinu hran. Zbývá ukázat, jak lze existující párování zvětšit o jednu hranu nebo případně zjistit, že již máme párování největší možné.

V první chvíli každého jistě napadne postupně přidávat do vytvářeného párování hrany, dokud to bude možné. Vyhledáme prvního muže, který dosud nemá partnerku, a mezi ženami, které zná, se pokusíme najít takovou, která zatím nemá partnera. Když se nám to podaří, můžeme do existujícího párování přidat další hranu, v opačném případě zůstane tento muž bez partnerky. Celý výpočet postupně zopakujeme pro všechny muže. Uvedený postup je jednouchý a rychlý, neřeší však správně zadanou úlohu. Najde sice nějaké párování, které již nelze rozšířit přidáním další hrany, nemusí to ale být největší možné párování v grafu. Postup je totiž založen na tom, že hrany do párování pouze přidáváme, vůbec nepočítá s možností nevhodně spojené páry naopak zase rozdělovat. Například ve chvíli, když už nemůžeme do stávajícího párování žádnou další hranu přidat, mohlo by se vyplatit nějaký pár rozdělit (tedy jednu hranu z párování odebrat), pokud se oba bývalí partneři mohou spojit s nějakými jinými dosud nespárovanými účastníky večírku. Tím do vytvářeného párování místo jedné odebrané hrany dvě nové hrany přidáme, takže velikost párování se celkově zvýší o 1. Situace navíc nemusí být takhle jednoduchá, někdy je pro zvýšení velikosti párování zapotřebí rozdělit třeba dva stávající páry a z jejich účastníků plus z dalších dvou lidí vytvořit tři nové páry apod.

Abychom postihli všechny tyto případy, budeme při zvyšování velikosti párování v grafu vyhledávat tzv. zlepšující cesty. Zlepšující cestou nazveme takovou posloupnost navzájem různých vrcholů grafu $V_1, V_2, \ldots V_n$, v níž mezi každými dvěma sousedními vrcholy vede hrana grafu, tyto hrany střídavě jsou a nejsou zařazeny do momentálně existujícího párování, přitom

první a posledních z nich v párování nejsou a navíc z vrcholů V_1 a V_n ani nevede žádná jiná hrana, která by byla zařazena do párování. Z uvedených podmínek přímo plyne, že n je sudé a že tedy jeden z vrcholů V_1, V_n představuje muže a druhý ženu. Nalezená zlepšující cesta V_1, V_2, ... V_n má lichý počet hran (je jich $n-1$), z nichž $(n-2)/2$ je zařazeno v současném párování a $n/2$ v současném párování není. Nyní stačí u všech hran zlepšující cesty změnit jejich náležení do párování. To znamená, že ty hrany zlepšující cesty, které jsou zařazeny v párování, z párování odstraníme, a naopak všechny ostatní hrany zlepšující cesty do párování zařadíme. Z vlastností zlepšující cesty plyne, že tím získáme opět párování a to že bude obsahovat o jednu hranu více, než kolik hran mělo párování původní. Pokud žádná zlepšující cesta v grafu neexistuje, je momentální párování již největší možné.

Zamyslíme se ještě nad tím, jak by bylo možné popsaný algoritmus realizovat. Nejsložitější částí celého řešení bude nalezení zlepšující cesty v grafu. To budeme provádět systematickým zkoušením možností. Aktuální stav párování si budeme evidovat tak, že si pro každou ženu zaznamenáme číslo jejího partnera, resp. 0, pokud dosud partnera nemá (v programu realizováno polem P). Začínáme s prázdným párováním, takže na začátku výpočtu nemá partnera žádná žena (v poli P jsou samé nuly). Postupně budeme procházet seznamem všech mužů a každého se vždy pokusíme zařadit do průběžně vytvářeného párování (v programu prostřednictvím rekurzivní funkce *Cesta*). Pokud uspějeme, zvýšíme počítadlo evidující počet již vytvořených párů (v programu proměnná V). Funkce *Cesta* hledá postupně mezi ženami první takovou, s níž se nově zařazovaný muž zná a která buď dosud nemá žádného partnera, nebo již nějakého partnera má (tzn. je již zařazena s někým jiným do párování), ale ten je schopen nalézt si jinou partnerku. To se zjišťuje rekurzivním voláním funkce *Cesta*. V případě úspěšného nalezení ženy uvedených vlastností s ní nově přidávaný muž vytvoří pár.

K programové realizaci algoritmu uveďme ještě jednu důležitou technickou poznámku. V rekurzivní funkci *Cesta* je třeba sledovat, aby posloupnost vnořených rekurzivních volání nevedla k zacyklení, tzn. aby se na zlepšující cestě žádný vrchol neopakoval. V naší programové ukázce jsme k tomu použili množinovou proměnnou S, jež obsahuje v každém okamžiku čísla těch žen, které se již účastní právě zkoumané zlepšující cesty. Muž, který pomocí rekurzivního volání funkce *Cesta* zkoumá možnost nalezení jiné partnerky, než jakou má dosud přiřazenou, neobrací se na ty ženy,

které jsou momentálně zařazeny do množiny S. Bez tohoto kontrolního mechanismu by pro některá vstupní data došlo k zacyklení výpočtu.

```
program Vecirek;
{Velikost největšího párování v bipartitním grafu}

const MaxM=100;  {maximální počet mužů}
      MaxZ=100;  {maximální počet žen}

var A: array[1..MaxM, 1..MaxZ] of boolean;      {známosti}
    M: 1..MaxM;                    {počet mužů}
    Z: 1..MaxZ;                    {počet žen}
    P: array[1..MaxZ] of 0..MaxM; {partneři jednotlivých žen}
    S: set of 1..MaxZ;             {ženy na zlepšující cestě}
    V: integer;                    {výsledný počet párů}
    i, j: integer;                 {pomocné}

function Cesta(C: integer): boolean;
{Určení zlepšující cesty, která začíná mužem číslo C}
{Vrací informaci, zda byla zlepšující cesta nalezena}
  var J: integer;
  begin
  Cesta:=false;
  for J:=1 to Z do
    if A[C,J] then
      if P[J]=0 then    {žena J zatím nemá partnera}
        begin           {zařadit hranu C-J do párování}
        P[J]:=C;
        Cesta:=true;
        exit
        end
      else if not (J in S) then   {žena J není na zlepšující
                                   cestě}
        begin
        S:=S+[J];
        if Cesta(P[J]) then
          begin            {partner ženy J si našel někoho
                            jiného}
```

```
          P[J]:=C;          {C bude novým partnerem ženy J}
          Cesta:=true;
          S:=S-[J];
          exit
          end
        else
          S:=S-[J]
        end
  end; {Cesta}

begin
{Vstup dat a inicializace:}
write('Počet mužů: ');
readln(M);
write('Počet žen: ');
readln(Z);
for i:=1 to M do
  for j:=1 to Z do A[i,j]:=false;
writeln('Dvojice známých -- muži kladnými  čísly',
                           ' od 1  do ', M);
writeln('                  ženy zápornými čísly',
                           ' od -1 do -', Z);
while not eof do
  begin
  read(i,j);
  if (i>0) and (j<0) then A[i,-j]:=true;
  if (i<0) and (j>0) then A[j,-i]:=true;
  end;
for j:=1 to Z do P[j]:=0;
V:=0;
S:=[];

{Hledání zlepšujících cest:}
for i:=1 to M do
  if Cesta(i) then V:=V+1;
writeln('Maximální počet současně tančících párů: ', V)
end.
```

(Autorkou úvodní ilustrace je Mgr. Jaroslava Čermáková z Hlinska.)

Přímka daná dvěma body

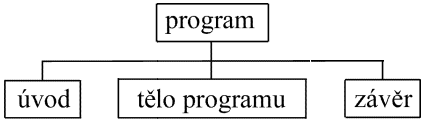
STANISLAV TRÁVNÍČEK

Přírodovědecká fakulta UP, Olomouc

V prvních fázích výuky programování řeší žáci (studenti) jako cvičení vhodné jednoduché úlohy. Patří k nim i úloha: najít obecný tvar $Ax+By+C=0$ rovnice přímky dané dvěma body (samozřejmě za předpokladu, že již z matematiky vědí, co je to rovnice přímky).

Má-li se taková úloha programově vyřešit, je třeba nejprve provést její analýzu, a to jak z hlediska matematiky, tak z hlediska programování. Je pravda, že zkušení programátoři si často provedou takovou analýzu „v duchu" a zdánlivě hned začnou psát program. Nebude však na škodu, když se u těchto úvodních fází programování malou chvíli pozdržíme (viz též [1]). Upřesněme však nejprve zadání úlohy takto: souřadnice bodů jsou celočíselné a rovněž koeficienty rovnice mají vyjít celočíselné.

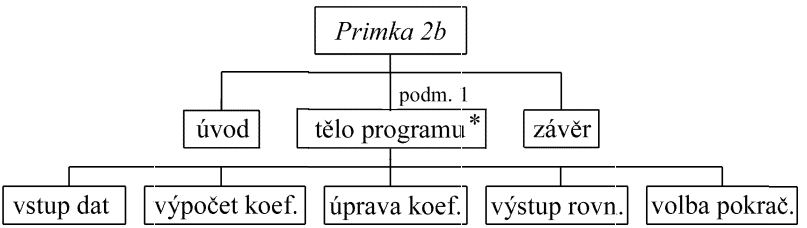
Klasickou strukturu jednoduchého školního programu, na jaký vede i naše úloha, lze v podstatě zakreslit schématem na obr. 1.



Obr. 1

Jaké jsou funkce jednotlivých částí programu: V *úvodu* program zpravidla seznámí uživatele s řešenou úlohou. *Tělo programu* řeší zadaný problém a zobrazí výstupy, uživatel zde vede dialog s programem. V *závěru* „se zamete", v našem případě vlastně jen program sdělí, že skončil svou práci (toto sdělení je užitečné, abychom věděli, že program skončil „normálně" a ne pro nějakou chybu). Analýza struktury programu obecně nezávisí na programovacích prostředcích, ale je pravda, že je praktické provádět ji už s jejich znalostí; v našem případě použijeme jazyk Pascal. (Je důležité uvědomit si toto schéma při programové analýze, protože ať už je pak struktura programu a forma jeho interakce s uživatelem jakákoli, vždy je tu třeba zařídit něco na počátku práce a něco na jejím konci.)

Do tohoto obecného schématu musíme nyní dosadit náš konkrétní problém. Podívejme se tedy nejprve na to hlavní, na tělo programu. Předpokládejme, že program (dáme mu název *Primka2b*) bude sloužit pro řešení sady úloh; v tom případě je vhodné vyhotovit tělo programu jako iteraci, tj. *aby se opakovaně vykonávalo, dokud uživatel nezadá konec* (podmínka1). Jaká bude struktura těla programu, to plyne z jeho úkolu. Nejprve se musejí načíst souřadnice dvou různých bodů, pak provést výpočet koeficientů A, B, C. Abychom dostali rovnici přímky v „pěkném“, tj. v obvyklém tvaru, bude vhodné tyto koeficienty upravit tak, aby jejich největší společný dělitel byl 1 a aby první nenulový koeficient byl kladný. Poslední částí bude zobrazení rovnice a dotaz, zda máme k řešení další příklad.



Obr. 2

Na obr. 2 je znázorněna získaná podrobnější struktura našeho programu; vidíme, že tělo programu je sekvencí (posloupností) pěti činností: *Vstup dat*, *Výpočet koeficientů*, *Úprava koeficientů*, *Výstup rovnice* a *Volba pokračování* (volba podmínky 1).

Nyní se věnujme další analýze úlohy. V grafické podobě bychom tak pod každý blok zakreslili jeho strukturu až po činnosti jednoduše zvládnutelné příkazy programovacího jazyka. My se však zastavíme již na této úrovni a programovou analýzu budeme doprovázet odkazy na již hotový program *Primka2b*.

Úvod programu (tj. začátek jeho fungování) zde zařídíme tak, že se vymaže obrazovka a program informuje o tom, co má uživatel udělat (zadat dva body roviny) a co dostane (rovnici přímky).

Od úvodu programu odlišujme *úvod zdrojového textu*, který obsahuje zejména deklarace proměnných a konstant, definice procedur a funkcí. Z toho je zřejmé, že jej nelze sestavit celý hned na začátku práce, ale v průběhu celé analýzy a programování, jak se potřeba deklarací a definic postupně vynořuje.

Tělo programu – Zadání bodů. Program o zadání bodů požádá. Zde si uvědomíme, že přímka je určena jen dvěma *navzájem různými* body, tj. kdybychom zadali dvakrát jeden a tentýž bod, přímku bychom nedostali. Proto programem zajistíme, aby se tak nestalo. Tedy po zadání souřadnic 1. bodu se souřadnice 2. bodu načítají v iteraci (tj. opakovaně), dokud 2. bod není různý od 1. Samozřejmě, pokud 2. bod zadáme hned správně (tj. různý od 1.), proběhne cyklus načtení souřadnic 2. bodu jen jednou (viz zdrojový text).

Výpočet koeficientů A, B, C. Zde je třeba vykonat nejprve matematickou analýzu problému. Označme zadané body $M_1[x_1, y_1]$, $M_2[x_2, y_2]$. Dva body leží na přímce, právě když jejich souřadnice vyhovují rovnici přímky, tj. platí

$$\begin{aligned} Ax_1 + By_1 + C &= 0, \\ Ax_2 + By_2 + C &= 0. \end{aligned}$$

Dostáváme tak dvě rovnice pro tři neznámé A, B, C. Násobme 1. rovnici y_2 a odečtěme od ní y_1 násobek rovnice druhé, dostaneme po úpravě

$$A(x_1y_2 - x_2y_1) = C(y_1 - y_2).$$

Tento výsledek nezávisí na tom, jsou-li některé ze souřadnic nulové. Podobně odečteme-li od x_1 násobku 2. rovnice x_2 násobek rovnice první dostaneme

$$B(x_1y_2 - x_2y_1) = C(x_2 - x_1).$$

Označme $D = x_1y_2 - x_2y_1$. Pak předchozí dvě rovnice lze zapsat

$$\begin{aligned} AD &= C(y_1 - y_2), \\ BD &= C(x_2 - x_1). \end{aligned}$$

Je-li $D \neq 0$, lze položit

$$A = y_1 - y_2, \quad B = x_2 - x_1, \quad C = D = x_1y_2 - x_2y_1. \tag{1}$$

Je-li $D = 0$, je z předchozích dvou rovnic nutně $C = 0$ (přímka prochází počátkem a tak opět platí $C = D$), protože alespoň jeden z rozdílů $y_1 - y_2$, $x_2 - x_1$ se nerovná 0 – body M_1, M_2 jsou totiž různé. Platí tedy

$$\begin{aligned} Ax_1 + By_1 &= 0, \\ Ax_2 + By_2 &= 0, \end{aligned} \tag{2}$$

a ježto $D = 0$, je jedna rovnice násobkem druhé, neboť $D = \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix}$ je determinant této soustavy.

Je-li M_1 počátek soustavy souřadnic, je první rovnice (2) splněna a ve druhé lze položit $A = -y_2$, $B = x_2$, tedy platí (1). Není-li M_1 počátek, jsou souřadnice bodu M_2 jistým k-násobkem souřadnic bodu M_1 ($D = 0$ a 1. řádek determinantu není nulový, takže 2. řádek je násobkem 1. řádku), $x_2 = kx_1$, $y_2 = ky_1$, nevyjímá se ani $k = 0$. Zvolíme-li A, B dle (1), lehce se přesvědčíme, že obě rovnice (2) jsou opět splněny.

Hledaná rovnice přímky má tedy tvar

$$(y_1 - y_2)x + (x_2 - x_1)y + (x_1y_2 - x_2y_1) = 0 \qquad (3)$$

a takto se postupuje i při výpočtech koeficientů v uvedeném programu *Primka2b*.

Úprava koeficientů. Popsaný způsob výpočtu koeficientů zaručuje, že koeficienty rovnice budou celá čísla. Mohli bychom však dostat např. rovnici $-9x + 6y - 21 = 0$, kterou bychom asi nechtěli považovat za (pěkný) výsledek. Vidíme totiž, že všechny koeficienty jsou dělitelné třemi a po dělení třemi dostaneme $-3x + 2y - 7 = 0$, která už představě výsledku vyhovuje. Přidali jsme ještě drobný požadavek navíc – aby první z nenulových koeficientů byl kladný, tj. naše rovnice se má změnit na $3x - 2y + 7 = 0$.

Nejprve tedy musíme najít největší společný dělitel čísel A, B, C, zde čísel 9, 6 a 21. Lze přitom použít vlastnosti, že největší společný dělitel $D(m, n, p) = D(D(m, n), p)$, tedy v našem případě nejprve najdeme číslo $K_1 = D(A, B)$ a pak výsledného dělitele $K = D(K_1, C)$ a tímto K všechny tři koeficienty dělíme (případ $K = 1$ je zbytečné rozlišovat). Takže je nyní problém najít největšího společného dělitele dvou čísel, na což však máme Eukleidův algoritmus, který se dá jednoduše naprogramovat. Ale pozor! Některý z koeficientů A, B (a také koeficient C) může být roven nule! V tomto případě potřebujeme, aby náš největší společný dělitel fungoval tak, že pro $n > 0$ bude platit $D(n, 0) = D(0, n) = n$ (případ $D(0, 0)$ nemůže nastat vzhledem k našemu uvedenému pořadí výpočtů).

Je více možností, jak tento problém vyřešit. V programu je zvoleno použití rekurzivní funkce $NSD(x, y)$ (vidíme, že jde vlastně o jediný složený příkaz). Pak je řešeno znaménko 1. členu v rovnici, tj. je-li $A < 0$ nebo $(A = 0) \wedge (B < 0)$, je změněno znaménko K a tím (po dělení číslem K) i znaménko koeficientů rovnice.

Zobrazení rovnice nabízí zajímavou hru, jak má počítač zvládnout naše lidské výjimky: Když je koeficient A nebo B roven 0, nepíšeme příslušný

člen, je-li roven 1, tak jedničku nepíšeme, ale člen (x nebo y) ano, před 1. členem nepíšeme +, je-li kladný, ale −, je-li záporný, píšeme vždy. Tím 1. členem může být i člen s y, tj. nenapíšeme rovnici $+2y - 3 = 0$, ale jen $2y - 3 = 0$. Je-li $C = 0$, tak tento člen nepíšeme, jinak je před C vždy znaménko. Tyto všechny problémy jsou ve zdrojovém textu vyřešeny zcela názorně, takže se jimi už zabývat nebudeme. Jen si můžeme všimnout nepodstatné změny, že místo $A \neq 0$ je použito $A > 0$ vzhledem k tomu, že po předchozí úpravě koeficientů jistě platí $A \geq 0$.

Dotaz na pokračování je volen tak, aby se odpověď nemusela potvrzovat klávesou *ENTER* a aby se skončila práce po zadání velkého nebo malého 'n'; jakoukoli jinou klávesou se pokračuje dalším příkladem.

Příklady nejsou úmyslně od sebe odděleny stránkováním, protože pro uživatele je praktické při zadávání příkladu vidět i příklady předchozí. Takže jediný přechod na novou stránku je proveden na počátku chodu programu.

Z analýzy je patrno, které proměnné je třeba deklarovat a jakou funkci je třeba definovat. Jednotka Crt je deklarována vzhledem k použití příkazů ClrScr a ReadKey.

```
program Primka2b;
uses Crt;

var
  X1,Y1,X2,Y2,A,B,C,K: Integer;
  Dale: Char;

function NSD(X, Y: Integer): Integer;
var Z: Integer;
begin {NSD}
  if X*Y = 0 then NSD := X+Y
  else
  begin
    Z := X mod Y;
    NSD := NSD(Y, Z)
  end
end;  {NSD}

begin  {program}
```

```
  {Uvod}
  ClrScr;
  WriteLn('Zadate dva body M1,M2 v rovine a program
  vypocte');
  WriteLn('rovnici primky Ax+By+C=0, ktera temito body
  prochazi.');

{Program-body}
  repeat
{Zadani bodu}
    WriteLn;
    Write('1.bod; x = '); ReadLn(X1);
    Write('       y = '); ReadLn(Y1);
    repeat
      Write('2.bod; x = ');ReadLn(X2);
      Write('       y = ');ReadLn(Y2)
    until (X1 <> X2) or (Y1 <>Y2);
{Vypocet koeficientu A, B, C}
    A := Y1-Y2; B := X2-X1;
    C := X1*Y2 - X2*Y1;
{Uprava koeficientu}
    K := NSD(Abs(A), Abs(B));
    K := NSD(K, Abs(C));
    if (A < 0) or ((A = 0) and (B < 0)) then K := -K;
    A := A div K; B := B div K; C := C div K;
{Zobrazeni rovnice}
    WriteLn;
    if A > 0 then
    begin
      if A <> 1 then Write(A);
      Write('x')
    end;
    if B <> 0 then
    begin
      if (A > 0) and (B > 0) then Write('+');
      if B < 0 then Write('-');
      if Abs(B) <> 1 then Write(Abs(B));
      Write('y')
```

```
    end;
    if C <> 0 then
    begin
      if C < 0 then Write('-')
      else Write('+');
      Write(Abs(C))
    end;
    WriteLn('=0');
    WriteLn;
{Dotaz na pokracovani}
    WriteLn('Dalsi priklad (A/N)? ');
    Dale := ReadKey;
    Dale := Upcase(Dale)
  until Dale = 'N';
{Zaver}
  WriteLn('Konec prace programu.')
end.  {program}
```

Pro studenty je zpracování předloženého tématu velmi přínosné nejen z hlediska výuky programování, ale i z hlediska mezipředmětových vztahů s matematikou. Lze také pozměňovat nebo rozšiřovat zadání, např. k daným dvěma bodům hledat vyjádření přímky ještě ve tvaru parametrickém, směrnicovém či úsekovém, což navodí i některé další problémy (rovnici se zlomky např. $y = \frac{1}{2}x - \frac{2}{3}$ lze zapsat třeba jako $y = 1/2x - 2/3$).

Literatura

[1] *Trávníček, S.:* Zpracování matematické úlohy pro počítač. MFvŠ, r. 19 (1988/89), č. 2 a 3.