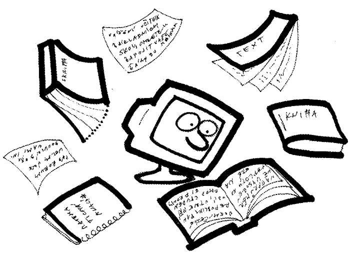
INFORMATIKA

Vyhledávání slov na stránce (Úlohy z MO – kategorie P, 28. část)

PAVEL TÖPFER
Matematicko-fyzikální fakulta UK, Praha



V tomto pokračování seriálu článků o zajímavých soutěžních úlohách z Matematické olympiády – kategorie P se budeme věnovat vyhledávání slov v textu. Budeme muset vyřešit otázku vhodného uložení seznamu slov tak, abychom k nim měli rychlý přístup, a efektivního vyhledávání těchto slov na zadané stránce textu. Tato úloha byla zadána v ústředním kole 55. ročníku Matematické olympiády – kategorie P (školní rok 2005/2006) jako úloha praktická. Jejími autory jsou pracovníci Fakulty matematiky, fyziky a informatiky Univerzity Komenského v Bratislavě, kteří se o programátorskou olympiádu středoškoláků na Slovensku starají. Česká i slovenská olympiáda stále velmi úzce spolupracují, obě soutěže mají stejné termíny jednotlivých soutěžních kol a stejné soutěžní úlohy, v jejichž přípravě se organizátoři z obou zemí pravidelně střídají.

Nejprve se seznámíme s úplným zadáním dnešní úlohy, které si pro potřeby našeho článku trochu upravíme, aniž bychom tím ovšem změnili smysl původní úlohy.

⋆ ⋆ ⋆

Rozhodli jsme se, že začneme konkurovat světoznámým vyhledávačům, jako jsou například Google a Yahoo. Hlavním klíčem k úspěchu bude samozřejmě prezentace nalezených stránek uživateli. Přesněji, chtěli bychom z každé nalezené stránky ukázat co nejkratší úsek obsahující všechna slova, která uživatel hledal. Vaší úlohou bude napsat program, který takový úsek v dané stránce nalezne.

Soutěžní úloha

Je dáno N slov, která uživatel hledá. Dále je dán text stránky obsahující M slov. Napište program, který najde nejkratší úsek stránky, v němž se vyskytují všechna zadaná slova (každé alespoň jednou). Pokud takových úseků existuje více, program vypíše ten z nich, který je nejblíže k začátku stránky.

Úsek stránky tvoří několik po sobě jdoucích slov. Délka úseku je rovna součtu jejich délek plus jejich počet minus 1 (za mezery mezi nimi). Tedy například úsek „Toto je úsek“ má délku 12.

Formát vstupu

Na prvním řádku vstupu je jediné celé číslo N – počet vyhledávaných slov. Následuje N řádků, na každém z nich je jedno vyhledávané slovo. Všechna tato slova jsou navzájem různá. Na dalším řádku vstupu se nachází jedno celé číslo M – počet slov na stránce. Následuje M řádků, na každém z nich je jedno slovo textu stránky, a to v pořadí, v jakém jsou slova na stránce uvedena.

Každé slovo je znakovým řetězcem, který je tvořen 1 až 100 malými písmeny anglické abecedy. Pro počet N vyhledávaných slov platí $1 \leq N \leq 5000$, pro počet M slov na stránce platí $1 \leq M \leq 200000$.

Formát výstupu

Program vypíše na první řádek výstupu dvě celá čísla určující začátek a konec nalezeného úseku. Začátek a konec úseku je určen pořadovým číslem příslušného slova v textu. Na druhý řádek výstupu program vypíše jedno celé číslo – délku tohoto úseku.

Pokud se některé vyhledávané slovo v textu stránky vůbec nenachází, program vypíše jediný řádek s textem „Chybna stranka!“ (bez uvozovek).

Příklad

Vstup:

3

nasi vasi prisli

hledaná slova – každé bude na vstupu uvedeno na samostatném řádku

20

poslali me nasi k vasim aby prisli vasi k nasim kdyz neprijdou vasi k nasim tak neprijdou nasi k vasim

text – každé slovo bude uvedeno na vstupu na samostatném řádku

Výstup:

3 8
28

Nejkratší úsek textu obsahující všechna tři hledaná slova má délku 28 znaků. Jedná se o úsek od třetího do osmého slova včetně, tzn. úsek ve znění „nasi k vasim aby prisli vasi“.

⋆⋆⋆

Při řešení úlohy se musíme věnovat dvěma relativně samostatným problémům. Prvním z nich je vhodná reprezentace množiny hledaných slov. Ta nám umožní o každém slově na stránce rychle zjistit, zda patří mezi hledaná slova, nebo ne. Určitě se nám také vyplatí každé slovo přitom nahradit nějakým jednoznačným kódem, nejlépe celým číslem. Pro řešení naší úlohy bude stačit označit si zvláštními kódy každé z N hledaných slov, zatímco všechna ostatní slova textu můžeme nahradit třeba nulou. Celý text stránky si po překódování do posloupnosti čísel můžeme uložit do pole (známe horní omezení pro počet slov M), pole celých čísel má výrazně nižší paměťová nároky než pole znakových řetězců (jednotlivá slova mají až 100 znaků!). Opakované porovnávání celých čísel je také významně rychlejší než porovnávání dlouhých znakových řetězců. Celočíselné kódy přiřazené hledaným slovům navíc ještě výhodně využijeme pro indexování pole. Jak uvidíme později, budeme potřebovat pro jednotlivá hledaná slova evidovat jejich počty výskytů v aktuálním zkoumaném úseku textu. Druhým problémem pak bude řešení vlastní úlohy, tzn. určení takového úseku slov, který je co nejkratší a přitom obsahuje všechna hledaná slova.

Začneme tedy s naším prvním problémem a všechna hledaná slova označíme celými čísly od 1 do N. Na přiřazení čísel jednotlivým slovům vůbec nezáleží, ale pro jednoznačnost můžeme zvolit očíslování v tom pořadí, v jakém jsou slova uvedena na vstupu. Pokud označíme nulou všechna slova, která nás nezajímají, můžeme takto převést text stránky uvedený v příkladu v zadání úlohy do posloupnosti čísel 0, 0, 1, 0, 0, 0, 3, 2, 0, 0, 0, 0, 2, 0, 0, 0, 0, 1, 0, 0. S tou se nám již bude pracovat lépe a hlavně rychleji, než s původním textem. Uložíme si ji jednoduše do pole celých čísel indexovaného od 1 do M. Poznamenejme ještě, že vedle posloupnosti kódů jednotlivých slov na stránce si musíme pamatovat také jejich délky, abychom dokázali určit výsledný nejkratší úsek. K tomu nám poslouží druhé pole indexované rovněž od 1 do M. Pro ilustraci si opět ukážeme, jak bude vypadat toto druhé pole pro text stránky z příkladu v zadání úlohy: 7, 2, 4, 1, 5, 3, 6, 4, 1, 5, 4, 9, 4, 1, 5, 3, 9, 4, 1, 5. Uvedená dvě pole obsahují úplnou informaci o řešeném problému. Převedeme-li si původní vstupní data do podoby těchto dvou polí, nebudeme již nadále muset vůbec pracovat s žádnými znakovými řetězci a při řešení úlohy již nebudeme ani potřebovat evidenci hledaných slov.

Popsané přiřazení čísel slovům můžeme získat nejsnáze tak, že si jednotlivá hledaná slova uložíme za sebou do pole znakových řetězců. Toto pole bude indexováno od 1 do N, index určuje kód přiřazený příslušnému slovu. Takové řešení by nám ale práci s textem příliš neurychlilo, jelikož při zpracování každého slova zkoumaného textu bychom museli pole opakovaně sekvenčně procházet a jednotlivé znakové řetězce v něm pracně vyhledávat. Převedení zadaného textu do podoby pole čísel popsaného v předchozím odstavci by vyžadovalo práci $O(M \cdot N \cdot L)$, kde L označuje maximální délku slova.

Výpočet můžeme urychlit, pokud pole hledaných slov setřídíme. To sice vyžaduje vykonat jednorázově $O(N \cdot L \cdot \log N)$ operací, slova textu pak ale můžeme v poli vyhledávat půlením intervalů, tedy v logaritmickém čase. Časová složitost zpracování zadaného textu se nám tím sníží na $O(M \cdot L \cdot \log N)$. To je ovšem pořád zbytečně mnoho. Nabízejí se dvě jiná, mnohem vhodnější řešení. Prvním z nich je použít pro uložení množiny hledaných slov místo obyčejného pole datovou strukturu trie, česky označovanou jako *písmenkový strom*. Druhou, podobně dobrou možností je využít *hešování*.

Písmenkový strom slouží k reprezentaci nějaké množiny slov (v našem případě všech N vyhledávaných slov). Každý uzel stromu představuje počáteční úsek nějakého slova. Uzel může mít až 26 synů, které odpovídají prodloužení začátku slova o některé z písmen 'a' až 'z'. Kořen stromu představuje prázdné slovo. V každém listu stromu nějaké slovo končí, slovo ale může končit i ve vnitřním uzlu stromu (je-li počátečním úsekem nějakého jiného, delšího slova). Ve všech uzlech trie proto musí být zaznamenána informace, zda tam nějaké uložené slovo končí, nebo ne. V některých úlohách nám k tomu postačí příznak typu boolean, většinou ale chceme také určit kód nalezeného slova, takže použijeme údaj vhodného celočíselného typu.

Písmenkový strom pro zadanou množinu N hledaných slov postavíme v čase $O(N \cdot L)$, kde L opět značí maximální délku slova. Každé z N slov do něj totiž vložíme v čase $O(L)$. Vyjdeme od „prázdného“ stromu tvořeného pouze kořenem a do stromu postupně vkládáme slova – každé vždy písmeno po písmenu. Dokud to jde, sestupujeme stromem podle jednotlivých písmen vkládaného slova od kořene směrem dolů po již existujících uzlech. Jakmile pro některé písmeno slova nevede cesta do již existujícího uzlu, vytvoříme nové uzly pro toto písmeno a také pro všechna další písmena za ním následující ve slově a zapojíme je do stromu na patřičné místo. Uzel, do něhož přijdeme pomocí posledního písmena slova, označíme ještě číselným kódem tohoto slova.

Při hledání slova v písmenkovém stromě postupujeme od kořene směrem k listům podle jednotlivých písmen hledaného slova. Pokud se při tom dostaneme do situace, že příslušný přechod neexistuje, znamená to, že hledané slovo není v trii uloženo. Jestliže takto úspěšně zpracujeme všechna písmena hledaného slova a skončíme v nějakém uzlu, ze záznamu v tomto uzlu zjistíme, zda je hledané slovo v trii uloženo a pokud ano, jaký má přiřazen kód. Vyhledání jednoho slova v trii má zřejmě časovou složitost $O(L)$, takže zpracování celého textu délky M zvládneme touto metodou provést v čase $O(M \cdot L)$.

Alternativní způsob uložení zadaných N slov pomocí hešování zde již nebudeme rozebírat, můžete se o něm dočíst třeba v Programátorské kuchařce na webových stránkách Korespondenčního semináře z programování MFF UK: http://ksp.mff.cuni.cz/tasks/21/cook4.html

Sestrojení hešovací tabulky s N slovy i následné vyhledávání všech M slov tvořících text bude mít přibližně stejnou časovou složitost, jaké jsme dosáhli při použití trie.

Nyní se budeme věnovat druhé části řešení úlohy. Předpokládejme, že máme připraveno M-prvkové pole s kódy slov tvořících zpracovávaný text v původním pořadí a druhé M-prvkové pole s délkami těchto slov. V textu budeme hledat co nejkratší úsek obsahující všech N hledaných slov. Primitivní řešení může vypadat tak, že postupně zvolíme všemi možnými způsoby souvislý úsek textu, určíme jeho délku a ověříme, zda obsahuje všech N slov. Z vyhovujících úseků vybereme ten nejkratší. Tento postup sice vede k nalezení správného řešení, je ale značně pomalý. V textu tvořeném M slovy lze vybrat souvislý úsek $O(M^2)$ způsoby, ověření správnosti jednoho zkoumaného úseku a určení jeho délky má pracnost $O(M)$, takže celková časová složitost algoritmu je $O(M^3)$.

Zbývá ještě vysvětlit, jakým způsobem budeme ověřovat, zda zkoumaný úsek obsahuje všech N hledaných slov. Využijeme skutečnost, že jsme si všechna hledaná slova překódovali do čísel od 1 do N. Vytvoříme si pole příznaků indexované od 1 do N, zkoumaný úsek budeme procházet slovo po slově a každému slovu ze zkoumaného úseku nastavíme v tomto poli příznak, že se již vyskytlo. V pomocné proměnné si přitom budeme počítat, kolik příznaků jsme nastavili. Po projití celého úseku nám počítadlo příznaků přímo určuje, zda jsou nastaveny příznaky u všech N hledaných slov.

Výše popsaný algoritmus je značně neefektivní proto, že v něm mnohokrát opakovaně procházíme a zapracováváme tytéž úseky textu. Hledání nejkratšího úseku obsahujícího všech N slov zrychlíme, když budeme všechny úseky textu se stejným začátkem zkoumat pohromadě. Začátek úseku můžeme zvolit M způsoby. Pro každou volbu začátku budeme do zkoumaného úseku postupně po jednom přidávat po sobě následující slova textu tak dlouho, dokud v úseku nebude obsaženo všech N hledaných slov. To budeme sledovat pomocí stejného pole příznaků, jako v předchozím případě. Průběžně si při tom budeme počítat také délku zkoumaného úseku. Nejkratší úsek textu, který obsahuje všech N hledaných slov a začíná na pevně zvoleném místě, takto najdeme v čase $O(M)$, resp. v čase $O(M)$ zjistíme, že pro zvolený začátek takový úsek neexistuje. Celková časová složitost uvedeného postupu řešení je proto $O(M^2)$.

Předvedeme si teď ještě lepší postup, který naši úlohu vyřeší dokonce v čase $O(M)$. Začneme úplně stejně jako v předchozím řešení – pro úsek začínající prvním slovem textu najdeme konec úseku tak, aby byl co nejkratší a přitom obsahoval všechna hledaná slova. Namísto pole příznaků určujících, které z hledaných slov se již ve zkoumaném úseku vyskytlo, budeme tentokrát využívat podobné pole indexované od 1 do N, které bude ovšem obsahovat čítače, kolikrát se v úseku které slovo vyskytlo. Zatímco v předchozí variantě řešení jsme poté celý výpočet zopakovali pro úsek začínající druhým slovem textu (a pak třetím slovem, čtvrtým slovem, atd.), nyní další postup změníme a z úseku nalezeného v prvním kroku výpočtu odebereme jeho první slovo. Tím posuneme začátek zkoumaného úseku na druhé slovo textu, ale zachováme si momentální konec zkoumaného úseku a jemu odpovídající nastavené čítače počtu slov (s jedinou změnou za právě odebrané slovo). Z hodnot čítačů snadno zjistíme, zda zkoumaný úsek i po odebrání svého prvního slova stále ještě obsahuje všech N hledaných slov. Pokud ano, máme nový (a dokonce kratší) vyhovující úsek.

Pokud ne, musíme zkoumaný úsek prodloužit a začneme do něj opět po jednom přidávat následující slova textu tak dlouho, dokud je to zapotřebí. Tím získáme vyhovující úsek, který začíná druhým slovem textu a končí co nejblíže, jak jen je to možné. Popsaný proces se bude stále opakovat, postupně při něm vyhledáme všechny vyhovující úseky minimální délky a z nich vybereme ten, který je nejkratší.

Značky vymezující začátek a konec právě zkoumaného úseku se během popsaného výpočtu střídavě (ale ne nutně pravidelně) posouvají textem zleva doprava. Při celém výpočtu jednou projdou od začátku do konce textu, takže postupně vyhodnocujeme pouze $O(M)$ úseků. Zpracování každého z nich má díky využití pole čítačů konstantní časovou složitost – nový úsek se liší od předchozího vždy jen o jedno slovo (přidané na konci nebo odebrané na začátku úseku). Celková časová složitost výpočtu je proto opravdu rovna slibované hodnotě $O(M)$.

Jak jsme si ukázali, řešení úlohy se skládá ze čtyř kroků. Těmi jsou

- postavení trie (příp. hešovací tabulky) se všemi N hledanými slovy v čase $O(N \cdot L)$,
- překódování textu tvořeného M slovy v čase $O(M \cdot L)$,
- vyhledání nejkratšího úseku obsahujícího všechna hledaná slova v čase $O(M)$,
- výpis výsledku v konstantním čase.

Jelikož N je jistě menší než M, můžeme celkovou časovou složitost řešení shora odhadnout jako $O(M \cdot L)$.

Na závěr zapíšeme celé řešení v programovacím jazyce Pascal. Datový typ *Uzel*, procedury *Vytvor*, *Vloz* a *Vyhledej* slouží k uložení písmenkového stromu a k provádění příslušných operací v něm. Kód programu přesně odpovídá popsanému algoritmu a je názorně rozdělen do čtyř po sobě jdoucích etap uvedených v předchozím odstavci.

```
program Stranka;
const MaxN = 5000;     {maximální počet hledaných slov}
      MaxM = 200000;  {maximální počet slov v textu}

type Uk = ^Uzel;
     Uzel = record  {uzel trie}
              pismeno: array['a'..'z'] of Uk;
              kod: integer
            end;
```

```
var N, M: integer;  {počet hledaných slov a počet slov
                     v textu}
    trie: Uk;       {kořen trie}
    slova: array[1..MaxM] of integer;  {zakódování textu}
    delky: array[1..MaxM] of integer;  {délky slov v textu}
    zac, kon, delka: integer;          {zkoumaný úsek}
    min_zac, min_kon, min_delka: integer;  {nejlepší úsek}
    vyskyt: array[1..MaxN] of integer;
              {počty výskytů hledaných slov ve zkoumaném
               úseku}
    pocet: integer;
              {počet hledaných slov obsažených ve zkoumaném
               úseku}
    slovo: integer; {přidávané nebo ubírané slovo}
    s: string;
    i: integer;

procedure Vytvor(var T: Uk);
{vytvoření nového prázdného uzlu trie}
var z: char;
begin
new(T);
for z := 'a' to 'z' do T^.pismeno[z] := nil;
T^.kod := 0;
end; {Vytvor}

procedure Vloz(T: Uk; S: string; K: integer);
{vložení slova S s kódem K do trie T}
var i: integer;
begin
for i := 1 to length(S) do
  begin
  if T^.pismeno[S[i]] = nil then Vytvor(T^.pismeno[S[i]]);
  T := T^.pismeno[S[i]];
  end;
T^.kod := K
end; {Vloz}
```

```
function Vyhledej(T: Uk; S: string): integer;
{vyhledání slova S v trii T;
 vrací kód slova S resp. 0, pokud slovo S v T není}
var i: integer;
begin
for i := 1 to length(S) do
  if T^.pismeno[S[i]] = nil then
    begin Vyhledej := 0; exit end
  else
    T := T^.pismeno[S[i]];
Vyhledej := T^.kod
end; {Vyhledej}

begin  {program}
{1. Sestrojení trie hledaných slov}
Vytvor(trie);       {kořen stromu}
readln(N);
for i:=1 to N do
  begin
  readln(s);        {hledaná slova}
  Vloz(trie, s, i)
  end;

{2. Uložení vstupního textu}
readln(M);
for i := 1 to M do
  begin
  readln(s);                        {slova textu}
  slova[i] := Vyhledej(trie, s);   {kódy slov}
  delky[i] := length(s)             {délky slov}
  end;


{3. Nalezení nejlepšího úseku}
zac := 1; kon := 0;   {počáteční prázdný úsek}
delka := -1;
for i := 1 to N do vyskyt[i] := 0;
```

```
počet := 0;
min_delka := maxint;
while true do
  if počet < N then        {úsek prodloužit}
    begin
    inc(kon);
    if kon > M then break;
    slovo := slova[kon];  {přidávané slovo}
    delka := delka + delky[kon] + 1;
    if slovo > 0 then     {je to hledané slovo}
      begin
      inc(vyskyt[slovo]);
      if vyskyt[slovo] = 1 then  {dosud v aktuálním úseku
                                  nebylo}
        begin
        inc(pocet);
        if pocet = N then        {máme nový vyhovující úsek}
          if delka < min_delka then
            begin
            min_zac := zac; min_kon := kon;
            min_delka := delka
            end
        end
      end
    end
  else  {pocet = N ... zkrátit úsek zleva}
    begin
    slovo := slova[zac];  {odebírané slovo}
    delka := delka - delky[zac] - 1;
    if slovo > 0 then     {je to hledané slovo}
      begin
      dec(vyskyt[slovo]);
      if vyskyt[slovo] = 0 then dec(pocet);
      end;
    inc(zac);
    if (pocet = N) and (delka < min_delka) then
      begin
      min_zac := zac; min_kon := kon;
```

```
      min_delka := delka
      end
    end;

{4. Výpis výsledku}
if min_delka = maxint then
  writeln('Chybna stranka!')
else
  begin
  writeln(min_zac, ' ', min_kon);
  writeln(min_delka);
  end
end.  {program}
```

(Autorkou úvodní ilustrace je Mgr. Jaroslava Čermáková.)

Využitie IKT pri rozvíjaní kompetencie modelovania vo vyučovaní matematiky

STANISLAV LUKÁČ

Prírodovedecká fakulta UPJŠ v Košiciach

Modelovanie a riešenie problémov

Rozvíjanie kompetencií matematického modelovania a riešenia problémov patrí medzi základné ciele súčasného vyučovania matematiky.

Pre modelovanie reálnych problémov je východiskom dobré porozumenie reálnej situácie a identifikácia reálnych objektov a ich vzájomných väzieb. Výsledkom matematického modelovania je potom transformácia reálneho problému na problém matematický. Pri tejto transformácii prenášame objekty a vlastnosti z reálneho sveta do abstraktného sveta matematických pojmov a objektov, z ktorých je zostavený model. Pri matematickom modelovaní sa tak reálne objekty nahradzujú matematickými a vzťahy medzi nimi sa vyjadrujú najčastejšie vo forme rovníc, nerovníc a funkčných závislostí.

V procese vytvárania a riešenia matematického modelu možno rozmanitými spôsobmi využiť potenciál IKT. My sa v tomto článku sústredíme na využite tabuľkového kalkulátora a dynamického geometrického systému pri vytváraní a skúmaní rôznych typov modelov. Tabuľkový kalkulátor disponuje numerickými a grafickými nástrojmi, ktoré uľahčujú a zjednodušujú proces modelovania. Tabuľky môžu predstavovať jednoduchú formu matematického modelu a ich tvorba a modifikácia v prostredí tabuľkového kalkulátora rozvíja predstavu modelovania ako dynamického procesu. Využitím základných výpočtových operácií a zabudovaných funkcií možno vytvárať vzorce, ktoré vyjadrujú vzťahy medzi údajmi uloženými v bunkách tabuľky. Veľkou výhodou tabuľkového kalkulátora je skutočnosť, že údaje v tabuľkách sa pri manipulácii s modelom simultánne menia a automaticky prinášajú výsledky, ktorých vhodnosť vo vzťahu k riešenému problému možno následne vyhodnotiť.

Využitie rôznych typov modelov vo vyučovaní matematiky

Matematické modelovanie môže nadobúdať vo vyučovacom procese rôzne podoby. Jeho vhodné zaradenie do výučby môže podporovať aktívne získavanie poznatkov založené na experimentovaní a výskumníckej činnosti.

Podľa matematického aparátu možno rozlišovať päť typov modelov:

- Aritmetický - vyjadrený napríklad vo forme tabuľky operácie.
- Geometrický - zostavený pomocou geometrických útvarov.
- Grafický - znázorňujúci grafy reprezentujúce funkčné závislosti.
- Algebraicko-analytický - založený na vyjadrení vzťahov medzi premennými reprezentujúcimi reálne objekty pomocou rovníc, nerovníc a funkčných závislostí.
- Kombinovaný.

Výber typu modelu pre riešenie problému je podmienený charakterom vstupných informácií a vlastnosťami modelovanej problémovej situácie.

Na prácu s numerickou reprezentáciou reálnych objektov možno využiť aritmetický model, ktorý v zjednodušenej forme vyjadruje kvantitatívne vzťahy medzi veličinami. Preskúmanie výsledkov aritmetických operácií umožňujú jednoduchšie tabuľky s dvoma riadkami alebo stĺpcami alebo aj zložitejšie tabuľky.

Príklad 1. Ako príklad na využitie zložitejšej tabuľky, pri zostavovaní ktorej je vhodné využiť tabuľkový kalkulátor, môže slúžiť nasledovná úloha od anglického matematika Jamesa Josepha Sylvestera, ktorá bola publikovaná v [3]. *Z veľkého počtu poštových známok s hodnotami 5 a 17 sa dajú skladať rôzne hodnoty. Aká je najväčšia hodnota, ktorá sa nedá vytvoriť kombináciou týchto dvoch hodnôt?* Je zrejmé, že napríklad hodnota 16 sa nedá zložiť z hodnôt známok 5 a 17. Ak budeme skúšať zostavať väčšie hodnoty, stále viac možností je potrebné preveriť pri hľadaní postupu skladania hodnôt 5 a 17. Preskúmajme čísla, ktoré možno vytvoriť kombináciou hodnôt známok 5 a 17. Detailnejší pohľad na súčty vytvárané kombináciou hodnôt známok prináša tabuľka, v ktorej sú systematicky počítané súčty násobkov hodnôt 5 a 17 (pozri tab. 1).

Záhlavie riadkov a stĺpcov tvoria čísla udávajúce, koľkokrát je v súčte započítaný násobok jednotlivých hodnôt. Na výpočet súčtu násobkov hodnôt 5 a 17 je v bunke B2 zapísaný vzorec: `=5*$A2+17*B$1`, ktorý je skopírovaný do zvyšnej časti tabuľky. V druhom riadku tabuľky sú teda za sebou idúce násobky čísla 17, ku ktorým sa v stĺpcoch tabuľky postupne pripočítava číslo 5.

	A	B	C	D	E	F	G	H	I	J	K	L
1	**5/17**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
2	**0**	0	17	34	51	68	85	102	119	136	153	170
3	**1**	5	22	39	56	73	90	107	124	141	158	175
4	**2**	10	27	44	61	78	95	112	129	146	163	180
5	**3**	15	32	49	66	83	100	117	134	151	168	185
6	**4**	20	37	54	71	88	105	122	139	156	173	190
7	**5**	25	42	59	76	93	110	127	144	161	178	195
8	**6**	30	47	64	81	98	115	132	149	166	183	200
9	**7**	35	52	69	86	103	120	137	154	171	188	205
10	**8**	40	57	74	91	108	125	142	159	176	193	210
11	**9**	45	62	79	96	113	130	147	164	181	198	215
12	**10**	50	67	84	101	118	135	152	169	186	203	220
13	**11**	55	72	89	106	123	140	157	174	191	208	225
14	**12**	60	77	94	111	128	145	162	179	196	213	230
15	**13**	65	82	99	116	133	150	167	184	201	218	235
16	**14**	70	87	104	121	138	155	172	189	206	223	240
17	**15**	75	92	109	126	143	160	177	194	211	228	245

Tab. 1

Vzhľadom na skutočnosť, že v stĺpcoch tabuľky sa čísla zväčšujú o 5, patria všetky čísla v jednotlivých stĺpcoch do rovnakej zvyškovej triedy modulo 5. V stĺpci B sú čísla zo zvyškovej triedy 0 modulo 5. V ďalších štyroch stĺpcoch tabuľky sú čísla zo zvyškových tried 2, 4, 1, 3 modulo 5.

V ďalších stĺpcoch tabuľky sú už potom čísla, ktoré sa nachádzajú aj v stĺpcoch B až F, ale v riadkoch, ktoré už nie sú zobrazené v tabuľke 1. Na čísla, ktoré sa nedajú vyjadriť kombináciou čísel 5 a 17, poukazuje druhý riadok tabuľky. Napríklad najväčšie prirodzené číslo zo zvyškovej triedy 4 modulo 5, ktoré sa nedá vyjadriť kombináciou čísel 5 a 17 je 29. Riešením úlohy je číslo zo zvyškovej triedy 3 modulo 5, konkrétne číslo 63.

Príklad 2. Zaoberajme sa ďalej modifikáciou prvej úlohy. *Majme známky s hodnotami 5 a 10. Koľkými spôsobmi možno nalepiť do radu známky, aby sme poskladali hodnotu 25?* Začnime skladaním menších hodnôt predstavujúcich násobky čísla 5:

5: (5)
10: (5, 5), (10)
15: (5, 10), (5, 5, 5), (10, 5)
20: (5, 5, 10), (10, 10), (5, 10, 5), (5, 5, 5, 5), (10, 5, 5)

Už pri vypisovaní možností pre hodnotu 20 sme postupovali tak, že k možnostiam pre hodnotu 10 sme pridali číslo 10 a k možnostiam pre hodnotu 15 sme pridali číslo 5. Získali sme tak 5 možností. Ak tento rekurentný vzťah budeme využívať ďalej, tak pre hodnotu 25 máme 8 možností, pre hodnotu 30 je to už 13 možností, atď. Teraz už žiaci ľahko zostavia schému s počtami možností, v ktorej sa postupne sčítavajú posledné dve čísla. Získajú tak členy Fibonacciho postupnosti. Zvyčajne sa táto postupnosť vyjadruje všeobecným vzťahom: $F(n) = F(n-1)+F(n-2)$, $F(1) = 1$, $F(2) = 1$.

V ďalšom kroku môžu žiaci prehľadne zapísať počty nájdených možností v závislosti od počtu využitých známok do tabuľky 2.

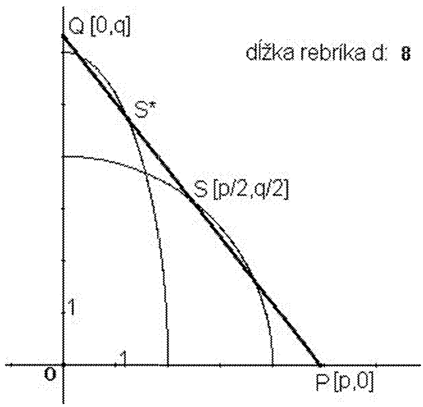
Počet/Hodnota	5	10	15	20	25
1 známka	1	1	0	0	0
2 známky	0	1	2	1	0
3 známky	0	0	1	3	3
4 známky	0	0	0	1	4
5 známok	0	0	0	0	1

Tab. 2

Úlohou žiakov by bolo objaviť vzťah, ktorý by im dovolil ľahko zväčšiť túto tabuľku v prostredí tabuľkového kalkulátora až do hodnoty 100.

Vyššie opísaný rekurentný vzťah sa v zostavenej tabuľke premietne takým spôsobom, že ak číslo neleží v prvom riadku alebo v prvých dvoch stĺpcoch výsledkovej časti tabuľky, tak sa rovná súčtu susedných dvoch čísel z predchádzajúceho riadku tabuľky, ktoré sú umiestnené od neho vľavo. Po vytvorení tabuľky by mali žiaci zbadať, že v jednotlivých riadkoch tabuľky sa nachádzajú čísla z odpovedajúcich riadkov Pascalovho trojuholníka.

Príklad 3. Zaujímavú aplikáciu geometrického modelu ponúka problém, ktorý je uvedený v publikácii [1] venovanej matematickému modelovaniu. *Rebrík dĺžky d sa posúva popri stene a padá na podlahu. V strede rebríka je farebná škvrna. Akú dráhu opisuje škvrna počas pádu rebríka?* Pri riešení tohto problému využijeme geometrický model vytvorený v prostredí dynamického geometrického systému. Základom modelu je súradnicová sústava znázorňujúca kolmú stenu a podlahu a úsečka PQ predstavujúca konkrétnu polohu rebríka (pozri obr. 1).



Obr. 1

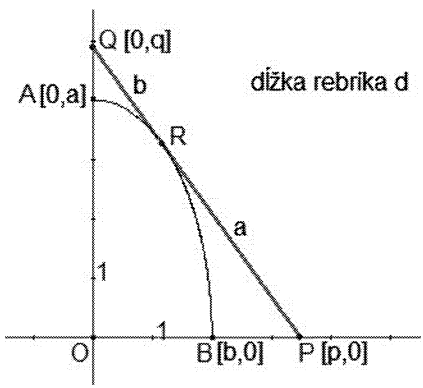
Za dĺžku rebríka sme zvolili číslo 8. Farebná škvrna v strede rebríka je reprezentovaná bodom S. Pri pohybe bodu Q po osi y sa zobrazujú rôzne polohy rebríka počas jeho pádu popri stene. Využitím nástrojov dynamického geometrického systému je zviditeľnená dráha bodu S. Po zmene dĺžky rebríka sa v konštrukcii prekreslí aj dráha bodu S. Z pozorovania zobrazenej dráhy bodu S možno vysloviť hypotézu, že stred rebríka sa pohybuje po štvrťkružnici so stredom v bode O. Túto hypotézu ľahko zdôvodníme využitím súradníc bodov a vzťahu pre vzdialenosť dvoch bodov v rovine. Ak označíme dĺžku úsečky PQ symbolom d, tak platí $p^2 + q^2 = d^2$. Druhá mocnina vzdialenosti bodu S od bodu O je určená výrazom:

$$\frac{p^2}{4} + \frac{q^2}{4}$$

Po porovnaní tohto výrazu s predchádzajúcou rovnosťou vidno, že vzdialenosť bodu S od bodu O je konštantná a má hodnotu $d/2$.

Po vyriešení problému si však možno položiť otázku: Ako by sa zmenilo riešenie, ak by sme skúmali dráhu iného bodu na rebríku? Pomocou vytvoreného geometrického modelu možno zobraziť aj dráhu iného bodu na úsečke PQ. Na obrázku 1 je zobrazený aj bod S^*, ktorý je stredom úsečky QS. Ako vidno z obrázka, získali sme krivku, ktorá už nie je častou kružnice. Pre zovšeobecnenie riešenia problému už použijeme algebraicko-analytický model odvodený z pôvodného modelu.

Na úsečke PQ dĺžky d si zvolíme ľubovoľný bod R rôzny od krajných bodov P, Q. Bod R rozdelil rebrík na dve časti, ktorých dĺžky označíme symbolmi a, b. Deliaci bod R teda rozdelil dĺžku rebríka d v pomere $a : b$. Opisovaná situácia je zobrazená na obrázku 2.



Obr. 2

Body A, B ležiace na súradnicových osiach znázorňujú polohu bodu R v dvoch medzných špeciálnych prípadoch. Na základe zvolených veličín vypočítame súradnice bodu R. Najprv vyjadríme parametrickú rovnicu úsečky PQ. Smerový vektor má súradnice $Q - P = [-p, q]$. Parametrická rovnica úsečky PQ má tvar:

$$x = p - p \cdot t$$

$$y = q \cdot t, \quad t \in \langle 0, 1 \rangle$$

Hodnotu parametra t pre súradnice bodu R určuje pomer dĺžky a k dĺžke celej úsečky PQ. Pre x-ovú súradnicu bodu R platí:

$$x = p - p \cdot \frac{a}{d} = p \cdot \left(1 - \frac{a}{d}\right) = p \cdot \frac{b}{d}$$

Podobne určíme aj y-ovú súradnicu bodu R a teda platí:

$$R\left[p \cdot \frac{b}{d}, q \cdot \frac{a}{d}\right].$$

Nech platí $b < a$. Podľa zostrojeného modelu by sa bod R mohol pohybovať po časti elipsy so stredom v bode O, hlavnou polosou a, vedľajšou polosou b. Súradnice bodov ležiacich na tejto elipse spĺňajú rovnicu:

$$\frac{x^2}{b^2} + \frac{y^2}{a^2} = 1$$

Ak platí náš predpoklad, musia potom súradnice bodu R pre ľubovoľnú polohu rebríka vyhovovať rovnici elipsy.

$$\frac{p^2 \cdot \frac{b^2}{d^2}}{b^2} + \frac{q^2 \cdot \frac{a^2}{d^2}}{a^2} = 1$$

Po jednoduchej úprave tejto rovnice dostávame rovnicu $p^2 + q^2 = d^2$, ktorá platí pre ľubovoľnú polohu rebríka počas jeho pádu. Ak by platilo $a = b$, dostávame špeciálny prípad, ktorý sme skúmali na začiatku.

(*Pokračování*)