INFORMATIKA

Sudoku

STANISLAV TRÁVNÍČEK

Přírodovědecká fakulta UP, Olomouc

Sudoku, rébus či hlavolam snad japonského původu, se stal v posledním roce doslova hitem v oblasti hádanek a křížovek. Čtvercové plány Sudoku se na nás dívají ze stránek časopisů, z oken novinových stánků i z pultů knihkupectví, neboť na výběr je i mnoho knižních vydání. Představme Sudoku i zde.

Čtvercové schéma (obr. 1) je rozděleno na 3×3 buněk a každá buňka je rozdělena na 3×3 elementů (políček), takže celkem je ve schématu 81 elementů. Do některých z nich jsou vepsány číslice 1 až 9. Úkolem řešitele je vyplnit i zbývající elementy číslicemi 1 až 9 tak, aby byla splněna *základní vlastnost* Sudoku, tj. aby každá číslice byla (právě jednou)

a) v každém řádku,

b) v každém sloupci,

c) v každé buňce.

Ptáme-li se v našem schématu na obr. 1, co může být v elementu $a = (1, 1)$, tj. v prvním řádku a v prvním sloupci, vidíme, že pro tento element jsou zakázány číslice 1, 3, 4 a 8, protože se už v dané buňce vyskytují, a také číslice 2 a 6, které jsou tomtéž řádku, a také číslice 7, která je v tomtéž sloupci. Přípustné jsou tedy jen 5 a 9, ovšem zatím nevíme, která z nich. Vidíme, že každý element ovlivňuje 20 dalších elementů. Jsou to ty, které jsou s daným elementem ve stejném řádku nebo sloupci nebo ve stejné buňce; nazvěme je „pole působnosti“ daného elementu. V poli působnosti elementu a jsou tedy číslice 1, 2, 3, 4, 6, 7 a 8.

			2		3		4	6
3	8	1	4			y	x	7
4				7				9
			6	2		4		5
7		4		5			2	1
	2	5			4			
	4		1	3		5		
2	5				6		1	4
		3	7					2

Obr. 1

Analýzou zjistíme, že do prázdného elementu můžeme dopsat číslici ve dvou situacích:

Pravidlo 1: Je-li v poli působnosti daného prázdného elementu 8 různých číslic (tj. všechny kromě číslice c), dopíšeme do tohoto elementu číslici c.

Pravidlo 2: Jestliže k danému prázdnému elementu mají všechny ostatní prázdné elementy v tomtéž řádku nebo v tomtéž sloupci nebo v téže buňce zakázánu číslici c, dopíšeme do tohoto elementu číslici c.

Nabízí se tedy tento postup řešení: Začneme pravidlem 1 a vyhledáme tedy takový prázdný element, aby se v jeho poli působnosti vyskytovalo 8 různých číslic; po jeho nalezení do něj vepíšeme chybějící devátou číslici a pokračujeme v hledání nového prázdného elementu s touž vlastností. Když se to nepodaří, zkusíme na prázdné elementy pravidlo 2 a pak opět pravidlo 1, atd. Po zaplnění Sudoku hra končí. Např. na schématu na obr. 1 jsou v poli působnosti elementu x všechny číslice kromě 5. Proto sem zapíšeme 5. Díky tomu jsou v poli působnosti elementu y všechny číslice kromě 2, takže sem zapíšeme 2, atd.

Tato hra nás může podnítit k tomu, abychom vytvořili program, který bude Sudoku řešit (nabídnutý postup řešení je totiž velmi vhodný pro počítač – přičemž pro „ruční řešení“ je lépe postup méně formalizovat) Vytvoření programu pro řešení Sudoku se ukazuje jako vhodný podnět pro studenty, kteří již znají základy programování. Ukážeme si jeden z možných přístupů.

Schéma hry si v programu zobrazíme jako třírozměrné pole H[R, S, T]; kromě indexu řádku a indexu sloupce bude vhodné v poli zaznamenat:

- T= 0: která číslice se nachází v daném elementu (R, S); když žádná, zapíšeme $H[R, S, 0] := 0$;
- T= 1 až 9: které z číslic $1 - 9$ nesmějí být zapsány v daném elementu (R, S), protože se už vyskytují v jeho poli působnosti; pokud je vyloučena číslice T, zapíšeme H[R, S, T]:= 1; v opačném případě je zde 0);
- T= 10: kolik číslic je pro daný element (R, S) zakázaných; jestliže H[R, S, T] = 8, lze do tohoto elementu doplnit chybějící devátou.

Vstup dat zvolíme co nejjednodušší. Programem Notepad si vytvoříme vstupní schéma jako textový soubor; pro výše uvedený příklad máme:

```
000203046
381400007
400070009
000620405
704050021
025004000
040130500
250006014
003700002
```

a jde tedy o 9 textových řetězců délky 9.

Úvod a deklarace programu:

```
program Sudoku;
uses Crt;

type
    TStr9 = string[9];
    TStr20 = string[20];
var
  JmS: TStr20;
  Radek: TStr9;    {pri cteni ze souboru}
  H: array [1..9,1..9,0..10] of Byte;
  BylaZmena: Boolean;
  SouDat: Text;
```

V programu je zvolen tento postup: Načtou se vstupní data a zkontrolují se, zda jsou v pořádku (když ne, program skončí). Pak se doplní hodnoty H[R, S, T] pro T = 1 až 10. V cyklu kroků se pak zjišťuje, pro která R, S lze použít pravidlo 1 nebo pravidlo 2. Pro tato R, S se najde T, pro něž H[R, S, T] = 0 (takové T je jediné), a položí se H[R, S, 0] := T. Pro všechny elementy v odpovídajícím poli působnosti, pokud pro toto T platí H[R, S, T] = 0, položí se H[R, S, T] := 1 (tj. v poli působnosti daného elementu se zakáže právě nastavená číslice) a H[R, S, 10] se zvětší o 1 (to provede procedura *Dopln*). Je vhodné přitom použít několika funkcí a procedur.

```
function LHR(X: Byte): Byte;
begin {LHR}
  LHR := ((X - 1) div 3) * 3 + 1
end;  {LHR}
```

Funkce LHR počítá levý horní roh aktuální buňky; k danému elementu (R, S) (indexy řádku a sloupce) má levý horní roh příslušné buňky souřadnice LHR(R), LHR(S).

```
function Zaplneno: Boolean;
var R, S: Byte;
begin {Zaplneno}
  Zaplneno := true;
  for R := 1 to 9 do
    for S := 1 to 9 do
      if H[R, S, 0] = 0 then Zaplneno := false
end;  {Zaplneno}
```

Funkce *Zaplneno* zjišťuje, zda je v některém elementu ještě nula. Pokud ano (tj. hodnota funkce je *false*), práce ještě není u konce a pokračuje se dalším krokem, v opačném případě cyklus jednotlivých kroků končí.

```
function VPoradku: Boolean; {nova verze}
var I, J, K1, K2, K3, R, S, R1, S1, T: Byte;
    VP: Boolean;
begin {VPoradku}
  VP := true;
```

```
  for R := 1 to 9 do
    for S := 1 to 9 do
      if VP then
      begin
       T := H[R, S, 0];
       if T > 0 then
       begin
         K1 := 0; K2 := 0; K3 := 0;
         for I := 1 to 9 do
         begin
           if H[I, S, 0] = T then Inc(K1);
           if H[R, I, 0] = T then Inc(K2);
          end;
          R1 := LHR(R); S1 := LHR(S);
       for I := R1 to R1 + 2 do
          for J := S1 to S1 + 2 do
            if H[I, J, 0] = T then Inc(K3);
        if (K1 > 1) or (K2 > 1) or (K3 > 1) then VP := false
       end
       else if H[R, S, 10] = 9 then VP := false
     end;
  VPoradku := VP
end;  {VPoradku}
```

Funkce *Vporadku* testuje, zda v některém řádku (sloupci, buňce) nejsou dvě stejné číslice. Jestliže ano, pak hodnota funkce je *false*. Tato funkce se použije pro ověření, že je zadání korektní a také pak ke kontrole výsledku, že splňuje pravidla Sudoku.

```
procedure VystupCtverce;
var R, S: Byte;
begin {VystupCtverce}
  WriteLn;
  for R := 1 to 9 do
  begin
    for S := 1 to 9 do Write(H[R, S, 0], '  ');
    WriteLn; WriteLn
  end
```

```
end;  {VystupCtverce}
```

Procedura *VystupCtverce* zobrazí schéma Sudoku na displeji – zadání a výsledek.

```
procedure Zadani;
var R, S, T: Byte;

procedure Prevezmi;
var
  I, J: Byte;
  K: Integer;
begin
  Reset(SouDat); I := 1;
  repeat
    ReadLn(SouDat, Radek);
    for J := 1 to 9 do Val(Radek[J], H[I, J, 0], K);
    Inc(I)
  until I = 10;
  Close(SouDat)
end;  {Prevezmi}

begin {Zadani}
  for R := 1 to 9 do
    for S := 1 to 9 do
      for T := 0 to 10 do H[R, S, T] := 0;
  Write('Jmeno souboru: ');
  ReadLn(JmS);
  JmS := 'DATA\' + JmS + '.txt';
  Assign(SouDat, JmS); Prevezmi; WriteLn;
  VystupCtverce;
  Write('Dale klavesou Enter..'); WriteLn;
  ReadLn;
  for R := 1 to 9 do
    for S := 1 to 9 do
       if H[R, S, 0] > 0 then
       begin
         for T := 1 to 9 do H[R, S, T] := 1;
```

```
       H[R, S, 10] := 9
      end
end;  {Zadani}
```

Procedura *Zadani* organizuje vstup dat (procedura *Prevezmi* umísťuje přečtené číslice do H[R, S, 0]) a v případě, že H[R, S, 0]) > 0 doplňuje jedničkami H[R, S, T] pro T $= 1$ až 9 a položí H[R, S, 10] $:= 9$ (což značí: tento element je již vyplněn).

```
procedure Dopln(R, S, T: Byte);
var I, J, K, R1, S1: Byte;
begin
  for I := 1 to 9 do
  begin
    if (H[R, I, T] = 0) then
    begin
      H[R, I, T] := 1; Inc(H[R, I, 10])
    end;
    if (H[I, S, T] = 0) then
    begin
      H[I, S, T] := 1; Inc(H[I, S, 10])
    end;
    R1 := LHR(R); S1 := LHR(S);
    for J := R1 to R1 + 2 do
      for K := S1 to S1 + 2 do
        if (H[J, K, T] = 0) then
        begin
          H[J, K, T] := 1; Inc(H[J, K, 10])
        end
  end
end;  {Dopln}
```

Procedura *Dopln* pro daný element provede zákaz použití dané číslice T v poli působnosti daného elementu.

```
procedure Zaznamenej;
var R, S, T: Byte;
begin {Zaznamenej}
```

```
  for R := 1 to 9 do
    for S := 1 to 9 do
    begin
      T := H[R, S, 0];
      if T > 0 then Dopln(R, S, T)
    end
end;  {Zaznamenej}
```

Procedura *Zaznamenej* organizuje použití procedury *Dopln* po načtení vstupních dat.

```
procedure VlozNove(R, S, T: Byte);
var I: Byte;
begin {VlozNove}
  if H[R, S, 0] <> T then
  begin
    H[R, S, 0] := T;
    for I := 1 to 9 do H[R, S, I] := 1;
    H[R, S, 10] := 9; Dopln(R, S, T)
  end
end;  {VlozNove}
```

Procedura *VlozNove* vkládá do elementu nově zjištěnou číslici a zařídí zákaz této číslice v poli působnosti daného elementu procedurou *Dopln*.

```
procedure DalsiKrok;
var K, M, N, R, S, T, R1, S1, U, V, W: Byte;
begin {DalsiKrok}
  if H[R, S, 10] = 8 then
  begin
    for K := 1 to 9 do
    if H[R, S, K] = 0 then
    begin
      T := K;
      VlozNove(R, S, T);
      BylaZmena := true
    end
  end;
```

```
  for R := 1 to 9 do
    for S := 1 to 9 do
    begin
      for T := 1 to 9 do
      begin
        U := 0; V := 0; W := 0;
        if H[R, S, T] = 0 then
        begin
          for K := 1 to 9 do
          begin
            if K <> S then U := U + H[R, K, T];
            if K <> R then V := V + H[K, S, T]
          end;
          R1 := LHR(R); S1 := LHR(S);
          for M := R1 to R1 + 2 do
            for N := S1 to S1 + 2 do
             if (M <> R) or (N <> S) then W := W + H[M, N, T]
        end;
        if (U = 8) or (V = 8) or (W = 8) then
        begin
          VlozNove(R, S, T); BylaZmena := true
        end
      end
    end
end;  {DalsiKrok}
```

Procedura *DalsiKrok* zjišťuje, zda lze použít pravidlo 1 a pravidlo 2; v kladném případě je procedurou *VlozNove* doplněna nová číslice do schématu, včetně jejího zákazu v poli působnosti daného elementu. Současně je nastaven příznak, že proběhla ve schématu změna (`BylaZmena :=true`), což znamená, že základní cyklus doplňování číslic může pokračovat dalším krokem. Pokud by procedura *DalsiKrok* skončila s nastaveným příznakem `BylaZmena := false`, cyklus doplňování číslic by se přerušil.

Jestliže k funkcím a procedurám uvedeným v této části článku připojíme vlastní program, můžeme řešit jedno Sudoku za druhým.

```
begin {program}
   ClrScr; WriteLn('SUDOKU'); WriteLn;
```

```
  Zadani;
  if not VPoradku then
  begin
    WriteLn('Vadne SUDOKU..');
    ReadLn; Exit
  end;
  Zaznamenej;
  repeat
    BylaZmena := false; DalsiKrok
  until Zaplneno or (not BylaZmena);
  if Zaplneno and VPoradku then
    VystupCtverce;
  ReadLn
end.
```

Ale tu se někdy ukáže, že Sudoku zůstane nedořešeno a v závěrečném schématu se objeví několik nul. To se stane tehdy, když u každého prázdného elementu zůstávají ještě dvě nebo i více možností (nebo také žádná, když zadané Sudoku nemá řešení, ale to zatím nechejme stranou). Pravidlo 1 nevede k doplnění další číslice a současně nám neporadí ani pravidlo 2. Měli bychom tedy vyzkoušet, která *volba* číslice vede k řešení. Může se dokonce stát, že (nepozorně zadaná) úloha má více řešení a s tím si náš program neporadí.

Chceme-li řešit i takováto Sudoku, musíme program doplnit tak, aby dovedl najít všechna řešení.

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

(*Pokračování*)

Optimalizace cesty

(Úlohy z MO – kategorie P, 15. část)

PAVEL TÖPFER

Matematicko-fyzikální fakulta UK, Praha

(*Dokončení*)

4. Časově optimální řešení

Ke zvýšení efektivity základního polynomiálního řešení můžeme přistoupit také jiným způsobem. Tentokrát nepoužijeme haldu a dvojice údajů (e_i, l_i) ponecháme v jednoduchém seznamu jako v případě původního řešení, ze seznamu však budeme průběžně odstraňovat všechny takové záznamy o kempech, které nemohou ovlivnit výsledek.

Jak už víme z řešení číslo 2, informace (e_i, l_i) o zpracovaných kempech se do seznamu ukládají v pořadí jejich zpracování, tzn. podle rostoucích hodnot l_i. Nyní můžeme učinit důležité pozorování: Máme-li v seznamu uloženy údaje o dvou kempech u, v takových, že $l_u < l_v$ a přitom $e_u \geq e_v$, pak záznam o kempu u je zcela zbytečný, výsledek stejně neovlivní a ze seznamu ho můžeme rovnou vypustit. Rozhodně se nám totiž nemůže vyplatit přenocovat v kempu u, který je blíže ke startu, když se za stejnou nebo dokonce nižší finanční částku můžeme dostat do nějakého od startu vzdálenějšího kempu v. To znamená, že než na konec seznamu vložíme nový záznam (e_j, l_j) o právě zpracovaném kempu j, můžeme nejprve vypustit z konce seznamu všechny záznamy o kempech s hodnotou e_i vyšší nebo rovnou hodnotě e_j. V důsledku toho bude seznam v každém okamžiku vzestupně uspořádán nejen podle hodnot l_i, ale také podle hodnot e_i. Je-li však seznam vzestupně uspořádaný také podle hodnot e_i, nepotřebujeme při výpočtu e_j celý seznam procházet a hledat v něm minimum (v lineárním čase), minimální hodnotu e_i máme přímo k dispozici na začátku seznamu (získáme ji tedy v konstantním čase).

Zopakujeme si tedy ještě jednou celý algoritmus výpočtu. Postupně budeme procházet všechny kempy v pořadí od startu do cíle cesty a budeme počítat, jak nejlevněji se do nich můžeme dostat. Přitom si budeme udržovat seznam údajů (e_i, l_i) o již zpracovaných a zároveň ne příliš vzdálených kempech. Výpočet začíná vložením záznamu $(e_0 = 0, l_0 = 0)$ do seznamu.

Zpracování kempu j (pro $j > 0$) zahájíme tím, že ze začátku seznamu vypustíme všechny záznamy o těch kempech, od nichž jsme se příchodem ke kempu j vzdálili o více než K, tzn. pro něž platí $l_j - l_i > K$. Po tomto případném vypouštění určitě v seznamu nějaký záznam zbude, neboť předpokládáme, že úloha má řešení. (Kdyby pro zadaná vstupní data úloha neměla řešení a nebylo by vůbec možné dojet ze startu do cíle podle stanovených pravidel, při zpracování některého kempu by se nám stalo, že by se uvažovaný seznam zcela vyprázdnil.) Hodnotu e_j pak spočítáme jako součet hodnoty e_i momentálně prvního záznamu uloženého v seznamu a ceny c_j za přespání v kempu j. Po určení e_j vypustíme z konce seznamu všechny záznamy o dřívějších kempech, do kterých se dokážeme dostat jedině za stejnou nebo vyšší cenu než e_j, tzn. pro něž platí $e_i \geq e_j$. Následně na konec seznamu vložíme záznam o zpracovávaném kempu j, tedy (e_j, l_j). Výpočet končí stanovením výsledné hodnoty e_{N+1}.

Je dost překvapující, že touto drobnou modifikací původního řešení 2 se nám podařilo získat řešení s lineární časovou složitosti $\mathrm{O}(N)$, tedy řešení, které je z hlediska asymptotické časové složitosti optimální. Rychlejší řešení nemůže existovat, neboť jenom samotná vstupní data mají velikost N a pro správné vyřešení úlohy je nezbytné přečíst a zpracovat zadané údaje o všech kempech. Výpočet probíhá v $N+1$ krocích, v každém z nich je nejprve několik (tj. žádný, jeden nebo více) záznamů ze seznamu vypuštěno, pak se v konstantním čase spočítá hodnota nového záznamu a nakonec je tento jeden nový záznam do seznamu vložen. Každé jedno vypuštění záznamu i vložení záznamu se provádí s konstantní časovou složitostí (provádí se vždy na některém konci seznamu, v případě vypouštění na základě jednoduchého testu). Záznam o každém kempu je do seznamu právě jednou vložen a nejvýše jednou vypuštěn, takže celkový počet všech vykonaných vkládání i vypouštění je skutečně lineární v N.

Pokud chceme vedle výsledné optimální ceny výletu určit také čísla kempů vybraných k přenocování, budeme postupovat naprosto stejně jako v obou předchozích variantách řešení úlohy. Zavedeme si tedy pomocné pole indexované čísly kempů, ve kterém si pro každý kemp j budeme evidovat číslo toho kempu, odkud jsme do kempu j přišli po cenově nejvýhodnější cestě. Výsledný seznam čísel vybraných kempů nakonec určíme dříve popsaným algoritmem zpětného chodu. Zavedení evidence předchůdců asymptotickou časovou složitost algoritmu neovlivní a zpětný chod má lineární časovou složitost, takže i po této úpravě bude celý výpočet probíhat s časovou složitostí $\mathrm{O}(N)$.

Výsledné optimální řešení si ještě ukážeme naprogramované v Pascalu. Program dodržuje tvar vstupních a výstupních dat podle původního zadání úlohy, zahrnuje i test řešitelnosti úlohy pro zadaná vstupní data. Pro zvýšení přehlednosti řeší tento program pouze část b) původní úlohy – tato část je algoritmicky zajímavější a věnovali jsme jí většinu článku. Bylo by samozřejmě možné doplnit do téhož programu také řešení části a) a při jednom načítání vstupních dat řešit souběžně obě části úlohy.

```
program Vylet;
{řešení úlohy b) s lineární časovou složitostí}

const MaxN=10000;

var Vstup, Vystup: text;
    L, K: longint;    {celková délka výletu, denní maximum}
    N: integer;       {počet kempů}
    V, C: longint;    {vzdálenost kempu od startu a cena za
                       přespání}
    E: longint;       {cena cesty do nového kempu}
    I, J: integer;
    Seznam: array[0..MaxN+1] of {seznam hodnot jednotlivých
                                 kempů}
              record
              I: integer;     {číslo kempu}
              E, L: longint   {minimální cena, vzdálenost od
                               startu}
              end;
    Zac, Kon: integer;        {začátek a konec seznamu}
    P: array[1..MaxN+1] of integer; {předchůdci jednotlivých
                                     kempů}
    Trasa: array[1..MaxN] of integer;  {kempy vybrané
                                        k přespání}

begin
assign(Vstup, 'vylet.in');
reset(Vstup);
assign(Vystup, 'vylet-b.out');
rewrite(Vystup);
```

```
readln(Vstup, L, K, N);  {základní vstupní údaje}
if L <= K then           {úloha má triviální řešení bez
                          přespání}
  begin
  writeln(Vystup, 0, ' ', 0);
  close(Vstup);
  close(Vystup);
  halt
  end;

Zac:=0; Kon:=0;                {inicializace seznamu}
Seznam[0].E:=0;
Seznam[0].L:=0;

for J:=1 to N do        {zpracování jednotlivých kempů}
  begin
  readln(Vstup, V, C);
  while V-Seznam[Zac].L > K do  {vypouštění na začátku
                                 seznamu}
    begin
    inc(Zac);
    if Zac > Kon then           {seznam se vyprázdnil
                                 - neexistuje řešení}
      begin
      writeln(Vystup, 'Trasu nelze projet.');
      close(Vstup);
      close(Vystup);
      halt
      end
    end;
  E:=Seznam[Zac].E+C;           {cena cesty do nového kempu}
  P[J]:=Seznam[Zac].I;          {předchůdce kempu na trase}
  while Seznam[Kon].E >= E do   {vypouštění na konci seznamu}
    dec(Kon);
  inc(Kon);                     {vložení nového záznamu do
                                 seznamu}
  Seznam[Kon].I:=J;
```

```
  Seznam[Kon].E:=E;
  Seznam[Kon].L:=V
  end;

while L-Seznam[Zac].L > K do    {dojezd do cíle trasy}
  begin
  inc(Zac);
  if Zac > Kon then  {seznam se vyprázdnil - není řešení}
    begin
    writeln(Vystup, 'Trasu nelze projet.');
    close(Vstup);
    close(Vystup);
    halt
    end
  end;
E:=Seznam[Zac].E;               {výsledná optimální cena}
P[N+1]:=Seznam[Zac].I;          {poslední přenocování}

I:=0;
J:=N+1;
while P[J] <> 0 do              {zpětný chod}
  begin
  J:=P[J];
  inc(I);
  Trasa[I]:=J
  end;

writeln(Vystup, I, ' ', E);     {výstup výsledků}
while I > 0 do                  {je třeba obrátit pořadí
                                 čísel kempů}
  begin
  write(Vystup, Trasa[I], ' ');
  dec(I)
  end;

close(Vstup);
close(Vystup);
end.
```

Literatura

[1] *Töpfer, P.:* Algoritmy a programovací techniky, Prometheus, Praha, 1995.
[2] *Horák, K. a kol.:* 50. ročník matematické olympiády na středních školách, JČMF, Praha, 2001.
[3] http://mo.mff.cuni.cz/p/archiv.html
[4] http://ksp.mff.cuni.cz/tasks/16/cook1.html

ZKUŠENOSTI

Převody jednotek netradičně

Stále více žáků základních a středních škol má problémy s převody jednotek. Ty bývají příčinou neutěšených známek z přírodovědných předmětů i zvýšeného stresu učitele. A přitom se člověk bez této znalosti neobejde v celé řadě běžných zaměstnání! (Co by se asi stalo, kdyby zdravotní sestra vstříkla pacientovi do žíly stonásobek potřebné dávky?)

Zároveň však patří dotyčné procvičování k nejméně záživným činnostem ve škole. Proto jsou následující úlohy pojaty poněkud netradičně: v každé jsou „schovány" nějaké chyby, které se podle mých zkušeností vyskytují nejčastěji, a výsledkem je roztomilý nebo přímo kolosální nesmysl. To by mělo žáky motivovat k jejich nalezení. Tak je vyzvěte, aby chyby odhalili!

1. Koloseum se vejde do krosny ...

Je-li poloměr římského Kolosea 238 stop a jeho výška 120 stop, jaký je objem Kolosea?

Řešení: nejprve převedeme stopy na metry: 1 stopa = 0,30 m. A metry na decimetry: 0,30 m = 0,003 dm.

Tedy:
poloměr $r = 238$ stop $= 238 \cdot 0{,}003$ dm $= 0{,}714$ dm.
výška $v = 120$ stop $= 120 \cdot 0{,}003$ dm $= 0{,}36$ dm.

K výpočtu objemu Kolosea použijeme vzorec pro objem válce:

$$V = \pi r^2 v = \pi \cdot 0{,}714^2 \cdot 0{,}36 \text{ dm}^3 =$$

$$= 0{,}576 \text{ m}^3$$

Objem kolosea činí 0,576 dm^3. (Pro srovnání: průměrná turistická krosna má objem kolem 60 dm^3. Koloseum by se do ní podle našeho výpočtu vešlo více než 100krát!)

2. A Eiffelovka je lehká jako pírko...

Vypočtěme hmotnost Eiffelovy věže. Věž je postavena z oceli o hustotě 7 830 kg/m^3 a její objem je přibližně 900 m^3.

Hustota $\varrho = 7\,830$ $\text{kg/m}^3 =$
$= 7{,}83$ g/cm^3; objem $V = 900$ $\text{m}^3 =$
$= 0{,}000\,9$ cm^3.

Odtud dostáváme hmotnost věže: $m = \varrho V = 7{,}83 \cdot 0{,}000\,9$ kg $= 0{,}007$ kg $= 7$ g

Hmotnost Eiffelovy věže je 7 gramů (asi jako 4 listy papíru ... !!).

3. Může člověk měřit 17 centimetrů?

Výška jistého muže činí 178 cm, tedy: 178 cm = 17,8 dm = 178 mm = 0,178 m = 17,8 cm!!!