INFORMATIKA

Optimalizace cesty

(Úlohy z MO – kategorie P, 15. část)

PAVEL TÖPFER

Matematicko-fyzikální fakulta UK, Praha

V dnešním pokračování série článků o zajímavých úlohách kategorie P matematické olympiády se zastavíme u jedné úlohy domácího kola z jubilejního 50. ročníku soutěže (školní rok 2000/2001). V této úloze se řešil problém nalezení nejvhodnější trasy pro skupinu turistů. Jednalo se vlastně o dva samostatné úkoly, které se lišily kritériem, podle něhož se optimalizace prováděla. V prvním případě bylo úkolem nalézt trasu s nejmenším možným počtem přenocování, ve druhém bylo cílem zaplatit za přenocování ve vybraných kempech co nejméně peněz, případně i za tu cenu, že bude třeba přenocovat vícekrát. Zde je celé původní zadání:

⋆ ⋆ ⋆

Skupina přátel se rozhodla, že v létě podniknou společný výlet na kolech. Většina zvolené trasy však vede přírodní rezervací, a proto mohou nocovat pouze v kempech. Kempy, ve kterých na svém výletě přespí, ještě nevybrali.

Celková délka naplánované trasy je L ($1 \leq L \leq 1000000000$). Maximální vzdálenost, kterou naši přátelé mohou urazit za jeden den, je K, tj. ve dvou po sobě následujících dnech musí přespat v kempech vzdálených nejvýše o K. Na naplánované trase se nachází celkem N kempů ($0 \leq N \leq 10000$); i-tý kemp je ve vzdálenosti l_i od začátku jejich výletu a cena za přespání v něm je c_i ($1 \leq c_i \leq 20000$). Čísla L, K, l_i a c_i jsou celá kladná; všechna l_i jsou navzájem různá a platí $0 < l_1 < l_2 < \cdots < l_N < L$.

Vaším úkolem je rozhodnout, zda skupina může naplánovanou trasu projet. Pokud lze trasu takto projet, pak určete, ve kterých kempech mají přespat tak, aby:

a) jejich výlet trval co nejmenší počet dní,

b) celková cena za přespání v kempech byla co nejmenší.

Úlohy a) a b) řešte zvlášť; v případě, že jednu z těchto úloh neumíte vyřešit, řešte pouze druhou z nich.

Formát vstupu:

Vstupní soubor se jmenuje `vylet.in`. Na prvním řádku jsou čísla L, K a N oddělená mezerou. Na dalších N řádcích následují dvojice čísel l_i a c_i oddělených mezerou, postupně pro $i = 1$ až $i = N$.

Formát výstupu:

Výstupní soubor se jmenuje `vylet-a.out` pro úlohu a) a `vylet-b.out` pro úlohu b). Na prvním řádku je věta „`Trasu nelze projet.`“, pokud výlet nelze uskutečnit tak, aby naši přátelé nikdy neujeli za den vzdálenost větší než K. V opačném případě první řádek obsahuje dvě čísla M a C. První z nich, M $(0 \leq M)$, je počet kempů, ve kterých skupina přespí, druhé z nich C je cena, kterou za přespání v těchto kempech zaplatí. Druhý řádek souboru obsahuje M mezerou oddělených čísel kempů, v nichž naši přátelé budou nocovat. Kempy jsou číslovány od jedné. Pokud je $M = 0$, nemusí být druhý řádek vůbec uveden. V případě, že existuje více řešení splňujících podmínku a) nebo b), program může vypsat libovolné jedno z nich.

Příklad 1:

```
vylet.in
25 5 9
4 2
5 8
8 2
10 8
12 2
15 8
16 2
20 8
24 2
```

```
vylet-a.out
4 32
2 4 6 8

vylet-b.out
5 16
1 3 5 7 8
```

Příklad 2:

```
vylet.in
15 10 2
2 11
4 12

vylet-a.out, vylet-b.out
Trasu nelze projet.
```

Příklad 3:

```
vylet.in
8 10 1
7 11

vylet-a.out, vylet-b.out
0 0
```

⋆ ⋆ ⋆

Nejprve se zamyslíme nad řešitelností úlohy, tzn. nad tím, zda je vůbec možné ze startu do cíle dojet podle stanovených pravidel. Pokud $K \geq L$, existuje zjevně triviální řešení obou částí úlohy, neboť je možné projet celou trasu za jediný den bez nocování v kempu, a tedy zcela zdarma. Jestliže $K < L$ a všechny kempy dělí celou trasu délky L na $N + 1$ úseků, potom úloha má řešení právě tehdy, když žádný z takto vzniklých úseků není delší než K. Pravdivost tohoto tvrzení je zřejmá – úsek delší než K nelze za jeden den přejet ani se v něm nedá nikde přespat, takže je pro nás

nepřekonatelnou překážkou. Jsou-li naopak všechny vzniklé úseky kratší než K, jistě existuje nějaká přípustná cesta ze startu do cíle – například taková, že budeme přespávat úplně ve všech kempech. V programu řešícím úlohu tedy nejdříve ověříme, zda $K < L$ a pokud ano, zda pro nějaké i od 1 do $N+1$ platí $l_i - l_{i-1} > K$ (položíme zde pracovně $l_0 = 0$, $l_{N+1} = L$ – fiktivní kempy na začátku a na konci trasy). Jestliže je některá tato nerovnost splněna, program oznámí, že úloha nemá řešení, a skončí.

Nadále předpokládejme, že úloha je řešitelná. Části a) a b) musíme řešit samostatně, obecně není možné hledat cestu, která by byla optimální zároveň podle obou stanovených kritérií. Tuto skutečnost můžeme snadno doložit jednoduchým příkladem. Nechť $L = 10$, $K = 5$ a máme tři kempy s následujícími parametry:

$l_1 = 3, \quad c_1 = 1$
$l_2 = 5, \quad c_2 = 5$
$l_3 = 7, \quad c_3 = 1.$

Celou trasu je možné zvládnout nejrychleji za dva dny s jedním přespáním v kempu číslo 2, a to za cenu 5. Jestliže ale dáme přednost minimální ceně, postačí nám zaplatit celkovou částku 2 za dvě přespání v kempech číslo 1 a 3, doba jízdy se nám ale prodlouží na tři dny.

Řešení úlohy a) je o dost snadnější než řešení části b), a proto se mu budeme věnovat jen velmi krátce. Chceme-li projet celou trasu za co nejmenší počet dní a nezáleží nám přitom na zaplacené ceně, budeme se snažit dojet každý den co nejdále, jak jen to bude možné. Opět budeme uvažovat, že máme také fiktivní kempy číslo 0 a $N+1$ na začátku a na konci celé trasy. Jestliže tedy některý den vyjíždíme z kempu číslo j, hledáme jako cíl pro tento den co největší číslo kempu h takové, že $l_h - l_j \leq K$. Tento postup lze snadno realizovat během jednoho postupného čtení vstupních údajů o kempech, údaje o kempech dokonce ani není třeba ukládat v programu do nějaké datové struktury. Průběžně počítáme potřebná přespání v kempech, zaplacené poplatky za přespání a zároveň si ukládáme čísla kempů, v nichž jsme přespali. Popsaný algoritmus má tedy lineární časovou složitost.

```
program Vylet;
{řešení úlohy a) -- nejrychlejší výlet}

const MaxN=10000;
```

```
var Vstup, Vystup: text;
    L, K: longint;   {celková délka, denní maximum}
    N: integer;      {počet kempů}
    V, C: longint;   {vzdálenost kempu od startu a cena za
                      přespání}
    V1, C1: longint; {údaje o bezprostředně předchozím
                      kempu}
    V2: longint;     {poloha předchozího přenocování}
    E: longint;      {průběžná cena celého výletu}
    I, J: integer;
    Trasa: array[1..MaxN] of integer;

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

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

I:=0;             {ukazatel v poli Trasa}
E:=0;             {průběžná cena výletu}
V2:=0;            {poloha předchozího přenocování}
V1:=0; C1:=0;     {předchozí kemp}

for J:=1 to N do  {zpracování jednotlivých kempů}
  begin
  readln(Vstup, V, C);
  if V-V1 > K then
    begin
```

```
    {kemp je příliš vzdálen od předchozího}
    writeln(Vystup, 'Trasu nelze projet.');
    close(Vstup);
    close(Vystup);
    halt
    end;
  if V-V2 > K then
    begin
    {do tohoto kempu už nedojedeme, musíme přespat
     v předchozím}
    V2:=V1;
    E:=E+C1;
    I:=I+1; Trasa[I]:=J-1
    end;
  V1:=V; C1:=C
  end;

if L-V1 > K then  {dojezd do cíle trasy}
  begin
  {cíl je příliš vzdálen od posledního kempu}
  writeln(Vystup, 'Trasu nelze projet.');
  close(Vstup);
  close(Vystup);
  halt
  end;
if L-V2 > K then
  begin
  {do cíle nedojedeme, musíme přespat v posledním kempu}
  E:=E+C1;
  I:=I+1; Trasa[I]:=J
  end;

{výpis trasy po úspěšném dosažení cíle:}
writeln(Vystup, I, ' ', E);
for J:=1 to I do
  write(Vystup, Trasa[J], ' ');

close(Vstup);
```

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

Řešení úlohy b) je mnohem zajímavější. Jak jsme si již ukázali, je-li hlavním kritériem volby cesty minimální celková výše poplatků za přespání ve vybraných kempech, nemusí se vždy vyplatit dojet každý den co nejdále – výhodnější může být přespat vícekrát v levnějších kempech. K nalezení správného řešení lze použít více různých postupů, které se liší svou časovou složitostí. Ukážeme si proto postupně několik způsobů, jimiž lze tuto úlohu řešit. Z praktických důvodů pro snadnější popis algoritmů se nám bude hodit opět předpokládat, že máme navíc ještě dva fiktivní kempy umístěné na začátku (kemp číslo 0) a na konci (kemp číslo $N+1$) celé trasy, v nichž „přespání" nic nestojí. Toto doplnění nijak neovlivní výsledek.

1. Zkoušení všech možností

Začneme přímočarým, ale dosti neefektivním řešením. Vyjdeme z výchozího bodu cesty a prozkoumáme všechny kempy, do nichž můžeme dojet během prvního dne, tzn. kempy vzdálené nejvýše K od startu. Pro každý z nich podobně vyzkoušíme další postup cesty, tedy všechny možnosti, do kterého kempu bychom mohli dojet během následujícího dne. Takto postupně projdeme všechny přípustné cesty ze startu (kemp číslo 0) až do cíle (kemp číslo $N+1$). K realizaci toho postupu se nám bude nejlépe hodit rekurzivní procedura, jejímž parametrem bude číslo právě zkoumaného kempu. Pokud toto číslo ještě není rovno $N+1$, procedura zavolá sama sebe na všechny kempy vzdálené od zkoumaného místa nejvýše K.

Metoda „vyzkoušení všech možností" algoritmem backtrackingu je teoreticky správná a vede k nalezení požadovaného řešení, pro svoji exponenciální časovou složitost je však reálně použitelná jen pro velmi malá N. Podle zadání naší úlohy však může být N rovno až 10000, pro takto velká N je použití exponenciálního algoritmu naprosto nemyslitelné.

2. Klasické polynomiální řešení

Základem řešení s polynomiální časovou složitostí bude postupné procházení jednotlivých kempů v tom pořadí, jak po sobě následují na trase výletu. Pro každý kemp j při tom budeme počítat hodnotu e_j, která udává nejnižší možnou cenu, jakou musíme zaplatit za všechna přespání v kempech během co nejvýhodnější cesty od startu až do kempu j (včetně poplatku v kempu j samotném). Hodnotu e_j určíme vždy na základě již

spočtených a zaznamenaných hodnot e_0, e_1, e_2, ... , e_{j-1}. Je zřejmé, že výslednou optimální cenou za celý výlet bude hodnota v cíli e_{N+1}.

Jednotlivé metody řešení se budou lišit podle toho, jak budeme údaje e_j počítat a ukládat. Nejjednodušší možností je prosté hledání minima z cen předchozích kempů. Již spočítané hodnoty jednotlivých kempů e_i společně s údajem o jejich poloze l_i si budeme postupně ukládat do seznamu (realizovaného například jednorozměrným polem, příp. spojovým seznamem). Nově získané záznamy (e_i, l_i) ukládáme vždy na konec seznamu, takže náš seznam je uspořádán vzestupně podle hodnot l_i. Výpočet začíná vložením záznamu $(e_0 = 0, l_0 = 0)$ do seznamu. V okamžiku, když začínáme zpracovávat kemp číslo j, můžeme ze začátku seznamu zrušit všechny záznamy o kempech, které jsou blíže ke startu a kemp j je od nich vzdálen více než K, tzn. pro něž platí $l_j - l_i > K$. Z takových kempů se do aktuálního kempu j nedá dojet během jednoho dne, takže tyto údaje nejsou použitelné pro výpočet hodnoty e_j. Ze stejného důvodu nebudou rušené údaje zapotřebí ani při zpracování dalších kempů, neboť ty jsou na trase výletu ještě více vzdáleny. I po zrušení těchto vzdálených kempů nám ale v seznamu vždy alespoň jeden záznamu zůstane, neboť předpokládáme, že úloha má řešení. Při výpočtu nové hodnoty e_j pro kemp j poté projdeme všechny záznamy o kempech uložené v seznamu (jedná se tedy o ty kempy, z nichž se dá dojet do kempu j během jednoho dne, tzn. pro které platí $l_j - l_i \leq K$). Hodnotu e_j spočítáme jako minimum z příslušných údajů e_i zvětšené o cenu přespání v j-tém kempu c_j.

Správnost uvedeného postupu plyne z toho, že jsme při určování každé hodnoty e_j vybírali minimum z hodnot e_i všech kempů, které na trase předcházely kempu j ve vzdálenosti nejvýše K. Z žádného jiného kempu jsme ovšem do kempu j nemohli přijít. Popsaný algoritmus má kvadratickou časovou složitost $\mathrm{O}(N^2)$. Výpočet probíhá v $N+1$ krocích, v každém z nich spočteme jednu hodnotu e_j na základě nalezení minima z nejvýše N hodnot e_i uložených momentálně v seznamu, tzn. v čase lineárním v N.

Pokud by nás zajímala pouze minimální cena, s níž dokážeme absolvovat celou trasu výletu, byli bychom v tuto chvíli hotovi – výsledkem je hodnota e_{N+1}. Jestliže chceme nalézt a vypsat také čísla kempů, v nichž budeme nocovat, je třeba řešení ještě maličko doplnit. Pro každý kemp j si budeme evidovat nejen minimální cenu e_j, za kterou se do kempu j dokážeme dostat, ale navíc ještě číslo toho kempu, ze kterého jsme do kempu j s touto minimální cenou přijeli, tedy bezprostředního předchůdce kempu j na vybrané trase. Tento údaj si označíme p_j a budeme ho snadno určo-

vat zároveň s hodnotou e_j. Je roven číslu toho kempu, pro který se získá minimum při výpočtu uvedeném v předchozím odstavci. Pokud má více takových kempů stejnou hodnotu e_i, za p_j můžeme položit číslo jednoho libovolného z nich (znamená to, že do kempu j lze přijet více různými způsoby za stejnou minimální cenu e_j). Údaje p_j o jednotlivých kempech si budeme ukládat nejlépe do jednorozměrného pole indexovaného přímo čísly příslušných kempů. Kempy vybrané k přespání na cenově optimální trase se na závěr výpočtu určí „zpětným chodem“ obdobně, jako se to dělá třeba při hledání nejkratší cesty v grafu. Procházení začneme ve fiktivním cílovém kempu $N+1$. Jeho údaj p_{N+1} nám říká, ze kterého kempu jsme do cíle přišli po cenově optimální trase. U tohoto kempu je zase zaznamenáno, odkud jsme přišli do něj, atd. Stejně postupujeme, dokud se nedostaneme až do výchozí bodu (tj. do kempu číslo 0).

Z hlediska časové složitosti algoritmu se uvedeným doplněním nic nezměnilo, výpočet údajů p_j se provádí souběžně s výpočtem hodnot e_j a celé řešení má nadále asymptotickou časovou složitost $O(N^2)$. Dodatečně přidaný zpětný chod pro určení vybraných kempů má zjevně složitost lineární a ve výsledné časové složitosti celého řešení se tedy neprojeví.

3. Využití haldy

Předcházející postup řešení můžeme vylepšit tím, že si dvojice údajů (e_i, l_i) o již zpracovaných kempech budeme ukládat nikoliv do lineárně řazeného seznamu, ale do datové struktury zvané halda.

Halda je známá a v programování dosti užívaná datová struktura, která umožňuje evidovat datové záznamy a podle jejich zvolené položky (tzv. klíče) s nimi efektivně provádět tyto operace:

- určení záznamu s minimální hodnotou klíče,
- přidáni nového záznamu do haldy,
- odebrání záznamu s minimální hodnotou klíče z haldy.

První z uvedených operací se provede v konstantním čase, přidání a odebrání záznamu má časovou složitost $O(\log N)$, kde N je počet záznamů momentálně uložených na haldě. Podrobnější popis způsobu realizace haldy a jejich operací se nám do tohoto příspěvku nevejde. Najdete ho například v učebnici [1] nebo ve stručnější podobě na Internetu ve studijním textu Korespondenčního semináře z programování MFF UK [4]. Popis haldy je také součástí řešení této úlohy, které bylo uvedeno v ročence 50. ročníku MO [2] a ve webovém archívu úloh MO kategorie P [3].

My se nyní budeme raději věnovat tomu, jak lze haldu výhodně využít při řešení naší konkrétní úlohy. Jak již bylo řečeno, na haldě si budeme uchovávat dvojice údajů (e_i, l_i) o již zpracovaných kempech a budeme je tam řadit podle klíče e_i. Začneme s jednoprvkovou haldou obsahující jediný záznam $(e_0 = 0, l_0 = 0)$. Při výpočtu nové hodnoty e_j pro nějaký kemp j nebudeme nyní muset procházet seznam předcházejících kempů a hledat v něm minimální e_i, místo toho si záznam (e_i, l_i) o již zpracovaném kempu s minimální hodnotou e_i jednoduše vyzvedneme v konstantním čase z haldy. Musíme ale ještě zkontrolovat, zda kemp j není od tohoto kempu příliš vzdálen, tzn. zda platí $l_j - l_i \leq K$. Pokud ano, spočítáme snadno hodnotu $e_j = e_i + c_j$ a nový záznam (e_j, l_j) vložíme do haldy. Jestliže ale uvedená nerovnost neplatí, můžeme kemp s minimální hodnotou e_i (naposledy nalezený na haldě) z haldy vypustit – je příliš vzdálen od kempu j, takže bude jistě příliš vzdálen i od všech následujících kempů a již ho tedy nebudeme moci využít. Poté z haldy vyzvedneme nový záznam s minimální hodnotou e_i a provedeme s ním stejný výpočet. Toto opakujeme podle potřeby vícekrát, dokud nezískáme výslednou hodnotu e_j a nevložíme ji do haldy. Celý výpočet opět končí ve chvíli, když určíme e_{N+1}, což je hledaný výsledek.

Zamyslíme se nyní, co nám tato změna způsobu práce se záznamy (e_i, l_i) přinesla z hlediska časové složitosti algoritmu. Výpočet opět probíhá v $N+1$ krocích, v každém z nich spočteme jednu hodnotu e_j (což se nám povede buď v čase konstantním, nebo po několika nezbytných vypuštěních vzdálených kempů z haldy) a následně tuto spočítanou hodnotu do haldy vložíme. Hodnotu každého kempu do haldy právě jednou vložíme a nejvýše jednou ji z haldy někdy později vypustíme, přitom každé vložení i vypuštění má časovou složitost $\mathrm{O}(\log N)$. Celková asymptotická časová složitost algoritmu je proto $\mathrm{O}(N \log N)$.

Zbývá ještě doplnit určení čísel kempů, ve kterých budeme přenocovat. To ale provedeme naprosto stejným způsobem jako v předchozí variantě řešení úlohy – v okamžiku určení hodnoty e_j pro kemp j si zároveň zaznamenáme číslo kempu p_j, díky němuž jsme tuto hodnotu získali (tzn. předchůdce kempu j na trase s minimální cenou). Hodnoty p_j si budeme opět ukládat do samostatného pole indexovaného čísly kempů. Výpočet hodnot p_j nám nijak neovlivní celkovou asymptotickou časovou složitost algoritmu. Seznam kempů vybraných k přenocování následně opět určíme v lineárním čase zpětným chodem.

(*Pokračování*)

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

Posloupnosti a řady v Excelu

VÁCLAV MATYÁŠ

Purkyňovo gymnázium, Strážnice

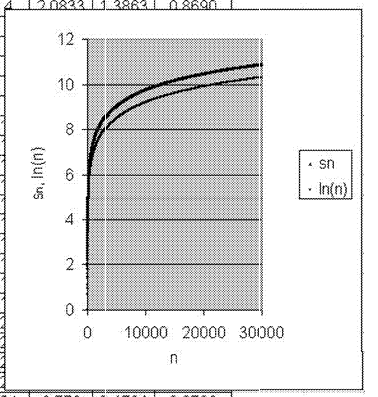
(*Dokončení*)

Základní harmonická řada $\sum_{n=1}^{\infty} \frac{1}{n}$

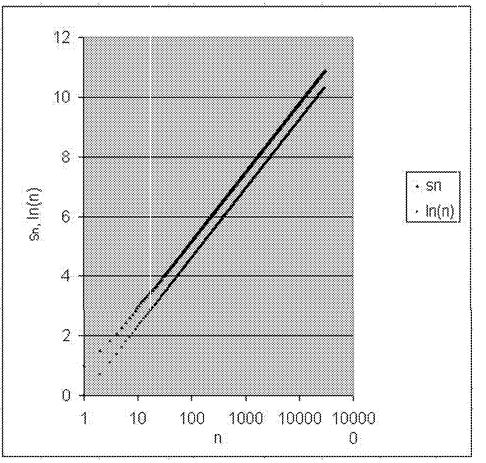
Vykreslíme-li posloupnost částečných součtů s_n harmonické řady $\sum_{n=1}^{\infty} \frac{1}{n}$, dostaneme výše položenou křivku na obrázku 8. Tato křivka zobrazuje hodnoty buněk ve sloupci C v závislosti na indexech členů ve sloupci B tabulky obrázku 8. Ve sloupci C jsou hodnoty počítány podle vztahu $s_n = s_{n-1} + \frac{1}{n}$, prvá hodnota $s_1 = 1$ je v C3 je zadána. Tabulka Excelu může mít až asi 60 000 řádků při dostatečné kapacitě operační paměti. Nám pro další zobrazení a úvahy určitě postačí tabulky s asi 30 000 řádky, do nichž Excel vypočítává pro naši potřebu dostatečně rychle.

Vzhled křivky s_n na obrázku 8 připomíná průběh logaritmické závislosti, jak se potvrdí, vyneseme-li do stejného obrázku hodnoty funkce $\ln n$, kterou zobrazuje níže položená křivka na obr. 8 (hodnoty ze sloupce D vynášené v závislosti na nezávisle proměnné n ze sloupce B). Tato podobnost je ještě zřejmější, jestliže obě křivky porovnáme na obrázku s logaritmickým měřítkem na ose n (obr. 9). Posloupnost $\{\ln n\}$ je divergentní a má limitu $+\infty$ a nám takto Excel napověděl, že základní harmonická řada je také divergentní a má součet $+\infty$. Z pohledu integrálního počtu

	B	C	D	E
2	*n*	s_n	*ln(n)*	$ds_n/dln(n)$
3	1	1	0	
4	2	1,5	0,6931	0,7213
5	3	1,8333	1,0986	0,8221
6	4	2,0833	1,3863	0,8690
26	24	3,776	3,1781	0,9790
27	25	3,816	3,2189	0,9799
28	26	3,8544	3,2581	0,9806



Obr. 8



Obr. 9

to není nijak překvapivé, když si uvědomíme geometrický význam známého vztahu $\int_1^n \frac{\mathrm{d}x}{x} = \ln n$. Podle obrázků 8 a 9 je také vidět, že můžeme odhadnout $\sum_{n=n_1}^{n_2} \frac{1}{n} \approx \ln \frac{n_2}{n_1}$, $(n_2 > n_1)$ tím lépe, čím větší je n_1. V tabulce Excelu (obr. 8) zjistíme, že (s přesností na 3 desetinná místa) např. $\sum_{n=100}^{10000} \frac{1}{n} = 4{,}640$ a vypočtené $\ln \frac{10\,000}{100} = \ln(100) = 4{,}605$. Obě hodnoty se tedy liší přibližně s chybou asi 0,5 % hodnoty součtu.

Řady $\sum_{n=1}^{\infty} \frac{1}{n(n+1)}$ a $\sum_{n=1}^{\infty} \frac{1}{n^2}$

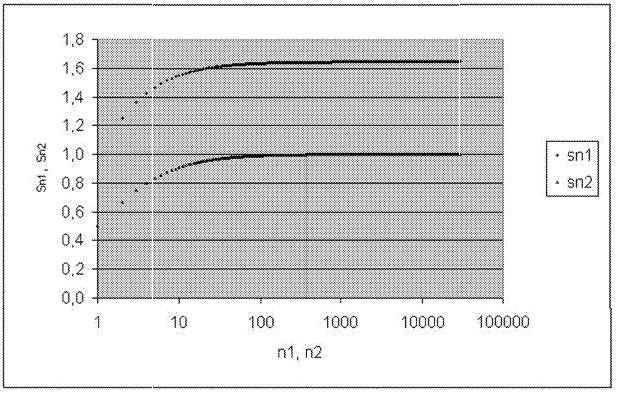
Pro první tři částečné součty první řady platí: $s_1 = \frac{1}{2}$, $s_2 = \frac{2}{3}$, $s_3 = \frac{3}{4}$. Jelikož $\frac{1}{n(n+1)} = \frac{1}{n} - \frac{1}{n+1}$, plyne odsud $s_n = 1 - \frac{1}{n+1}$ z čehož plyne $\lim_{n\to\infty} s_n = 1$, řada je konvergentní.

Pro druhou řadu platí

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = 1 + \sum_{n=1}^{\infty} \frac{1}{(n+1)^2}. \tag{2}$$

Dále platí pro všechna n: $\frac{1}{(n+1)^2} < \frac{1}{n(n+1)}$. Z konvergence řady $\sum_{n=1}^{\infty} \frac{1}{n(n+1)}$ tedy vyplývá konvergence řady $\sum_{n=1}^{\infty} \frac{1}{(n+1)^2}$ a podle (2) i konvergence řady $\sum_{n=1}^{\infty} \frac{1}{n^2}$.

Na obrázku 10 jsou vykresleny částečné součty s_{n_2} řady $\sum_{n=1}^{\infty} \frac{1}{n(n+1)}$ (dolní křivka) a s_{n_1} řady $\sum_{n=1}^{\infty} \frac{1}{n^2}$ (horní křivka). I v grafu s logaritmickým měřítkem na ose x, kde řada ukazuje lineární růst (obr. 9), nabývají obě vykreslené řady podle vzhledu konstantní hodnoty přibližně pro $n > 100$. Srovnání průběhu obou křivek (obr. 10) může vést k odhadu součtu řady $\sum_{n=1}^{\infty} \frac{1}{n^2}$.



Obr. 10

Nechť m je přirozené číslo. Platí

$$\sum_{n=1}^{\infty}\frac{1}{n^2}=\sum_{n=1}^{m}\frac{1}{n^2}+\sum_{n=m+1}^{\infty}\frac{1}{n^2}\geq\sum_{n=1}^{m}\frac{1}{n^2}+\sum_{n=m+1}^{\infty}\frac{1}{n(n+1)}=$$

$$=\sum_{n=1}^{m}\frac{1}{n^2}+\left(\sum_{n=1}^{\infty}\frac{1}{n(n+1)}-\sum_{n=1}^{m}\frac{1}{n(n+1)}\right)=\sum_{n=1}^{m}\frac{1}{n^2}+1-\sum_{n=1}^{m}\frac{1}{n(n+1)},$$

Podobně pro horní odhad z nerovnosti

$$\sum_{n=1}^{m}\frac{1}{n^2}+\sum_{n=m+1}^{\infty}\frac{1}{n^2}\leq\sum_{n=1}^{m}\frac{1}{n^2}+\sum_{n=m}^{\infty}\frac{1}{n(n+1)}$$

dostaneme

$$\sum_{n=1}^{\infty}\frac{1}{n^2}\leq\sum_{n=1}^{m}\frac{1}{n^2}+1-\sum_{n=1}^{m-1}\frac{1}{n(n+1)}$$

Celkově tedy odhadneme:

$$\sum_{n=1}^{m}\frac{1}{n^2}+1-\sum_{n=1}^{m}\frac{1}{n(n+1)}\leq\sum_{n=1}^{\infty}\frac{1}{n^2}\leq\sum_{n=1}^{m}\frac{1}{n^2}+1-\sum_{n=1}^{m-1}\frac{1}{n(n+1)}\qquad(3)$$

Součty konečného počtu sčítanců odhadneme z tabulek Excelu. Zvolíme-li např. $m = 30\,000$, pak s přesností na 9 desetinných míst platí: $\sum_{n=1}^{m} \frac{1}{n^2} =$ $= 1{,}644900734$, $\sum_{n=1}^{m} \frac{1}{n(n+1)} = 0{,}999966668$, $\sum_{n=1}^{m-1} \frac{1}{n(n+1)} = 0{,}999966667$ a podle (3) platí

$$1{,}644934066 \leq \sum_{n=1}^{\infty} \frac{1}{n^2} \leq 1{,}644934067.$$

Již *L. Euler* odvodil, že řada $\sum_{n=1}^{\infty} \frac{1}{n^2}$ má součet $\frac{\pi^2}{6}$ a můžeme se i na kalkulačce přesvědčit, že náš výpočet tomu odpovídá.

Řada $\sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{n}$.

Jde o alternující řadu $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \ \ldots\ \frac{(-1)^{n-1}}{n} + \ \ldots$ Podle Leibnizova kritéria konvergence alternujících řad $c_1 - c_2 + c_3 - c_4 + \ldots$, $c_n > 0$, je postačující podmínkou konvergence, aby posloupnost $\{c_n\}$ monotónně konvergovala k nule, a to naše řada splňuje. (Pro řady, které nejsou alternující to postačující podmínkou konvergence není, jak je vidět na příkladu divergentní harmonické řady $\sum_{n=1}^{\infty} \frac{1}{n}$.) Leibnizova věta udává také návod pro odhad součtu alternující řady [1]: absolutní chyba $s - s_n$ výpočtu má velikost $|s - s_n| < c_{n+1}$ a znaménko $(-1)^n$.

Částečné součty řady $\sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{n}$ jsou vykresleny na obrázku 11 a jsou vypočteny ve sloupci C tabulky. Do buňky C3 je zadán první částečný součet $s_1 = 1$ a další součty jsou vypočteny podle vztahu $s_n = s_{n-1} + \frac{(-1)^{n-1}}{n}$. Obsah buňky C4 je tedy vypočten podle vzorce =C3+1/B4*(-1)^(B4-1), obsahy dalších buněk sloupce C dostaneme tažením. Ve sloupci B jsou indexy posloupnosti částečných součtů řady.

Z obrázku je vidět, že v konkrétním případě této řady můžeme se žáky provést názorné úvahy o konvergenci.

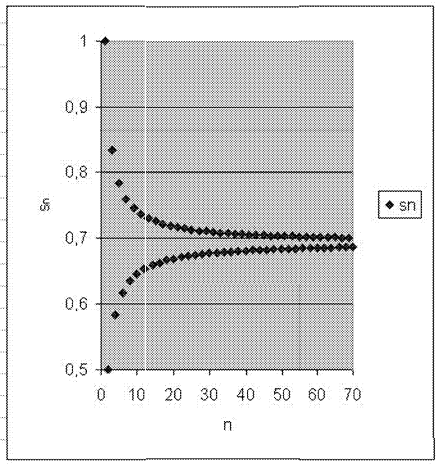
Částečné součty řady lze rozdělit na součty s lichým indexem (dále zkráceně liché): $s_1 = 1$, $s_3 = 1 - \frac{1}{2} + \frac{1}{3}$, $\ldots$,

$$s_{2k+1} = s_{2k-1} - \frac{1}{2k} + \frac{1}{2k+1} = s_{2k} + \frac{1}{2k+1}, \quad k \in \mathbb{N} \tag{4}$$

a na součty se sudým indexem (dále zkráceně sudé) $s_2 = 1 - \frac{1}{2}$, $s_4 = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4}, \ldots$,

$$s_{2k+2} = s_{2k} + \frac{1}{2k+1} - \frac{1}{2k+2}. \tag{5}$$

3	1	1
4	2	0,5
5	3	0,8333
6	4	0,5833
7	5	0,7833
8	6	0,6167
9	7	0,7595
10	8	0,6345
11	9	0,7456
12	10	0,6456
13	11	0,7365
14	12	0,6532
15	13	0,7301
16	14	0,6587
17	15	0,7254
18	16	0,6629
19	17	0,7217
20	18	0,6661
21	19	0,7188
22	20	0,6688
23	21	0,7164
24	22	0,6709
25	23	0,7144
26	24	0,6727



Obr. 11

Z obrázku 11 vidíme a podle (4) snadno dokážeme, že posloupnost lichých částečných součtů je klesající, protože $s_{2k+1} - s_{2k-1} = -\frac{1}{2k} + \frac{1}{2k+1} < 0$ a odsud $s_{2k+1} - s_{2k-1}$ pro všechna $k \in \mathbb{N}$.

Podobně podle obrázku vidíme a podle (5) dokážeme, že $s_{2k+2} - s_{2k} = \frac{1}{2k+1} - \frac{1}{2k+2} > 0$, a tedy posloupnost sudých částečných součtů je rostoucí.

Dále platí $0{,}5 = s_2 \leq s_{2k} < s_{2k-1} \leq s_1 = 1$. Posloupnost lichých součtů je tedy zdola omezená a posloupnost sudých součtů je shora omezená. Klesající a zdola omezená posloupnost lichých součtů tedy má limitu (označme $\lim_{k\to\infty} s_{2k+1} = p$) a stejně tak rostoucí a shora omezená posloupnost sudých součtů má limitu (označme $\lim_{k\to\infty} s_{2k} = s$). Protože

však $p - s = \lim\limits_{k\to\infty} s_{2k+1} - \lim\limits_{k\to\infty} s_{2k} = \lim\limits_{k\to\infty} (s_{2k+1} - s_{2k}) = \lim\limits_{k\to\infty} \frac{1}{2k+1} = 0$, tedy $p = s$. Tato společná limita posloupností částečných součtů je tedy součtem řady $\sum\limits_{n=1}^{\infty} \frac{(-1)^{n-1}}{n}$. (Tímto postupem jsme vlastně na zvláštním případě dokázali Leibnizovu větu.) Vzhledem k tomu, že posloupnost lichých částečných součtů se k součtu s řady blíží shora a posloupnost sudých částečných součtů zdola, platí pro všechna $k \in \mathbb{N}$

$$s_{2k} \leq s \leq s_{2k-1}$$

a podle této nerovnosti můžeme součet řady s odhadnout. Například pro $2k = 30\,000$ odečteme z tabulky Excelu:

$$0{,}69313 \leq s \leq 0{,}69316.$$

Na intervalu $(-1, 1\rangle$ platí

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \ \ldots\ + (-1)^{n-1}\frac{x^n}{n} + \ \ldots,$$

z čehož plyne, že řada $\sum\limits_{n=1}^{\infty} \frac{(-1)^{n-1}}{n}$ má součet $\ln 2$. Opět se pomocí kalkulačky můžeme přesvědčit, že náš výpočet tomu odpovídá.

Autor děkuje svým kolegům ing. Blance Hlávkové a RNDr. Josefu Tomešovi za užitečné informace o dalších možnostech Excelu.

Literatura byla uvedena za 1. částí článku.