Rozklady na součet
(Úlohy z MO – kategorie P, 32. část)

PAVEL TÖPFER

Matematicko-fyzikální fakulta UK, Praha

Náš dlouhodobý seriál o úlohách z Matematické olympiády – kategorie P se dnes zastaví ve 39. ročníku MO, který se konal ve školním roce 1989/90. Ačkoliv termín na vypracování úloh domácího kola časově kolidoval s bouřlivými revolučními událostmi listopadu 1989, soutěž se normálně konala a našlo se i dost studentů, kteří v listopadu řešení úloh odevzdali. Jednu z úloh tohoto domácího kola si nyní předvedeme. Rozebereme nejen původní soutěžní úlohu, v níž bylo třeba nalézt a vypsat všechny možné rozklady zadaného přirozeného čísla na součet přirozených sčítanců, ale ukážeme si i její modifikaci, v níž máme určit pouze počet těchto rozkladů, aniž bychom museli jednotlivé rozklady vypisovat. Při řešení této modifikace můžeme totiž použít zcela odlišný postup výpočtu, který je efektivnější a určí výsledek podstatně rychleji.

Začneme zadáním původní soutěžní úlohy, které uvádíme s určitými formulačními úpravami, aniž bychom však změnili smysl úlohy.

Úloha

Vytvořte program, který vypíše všechny různé rozklady zadaného přirozeného čísla N na součet přirozených čísel. Dva rozklady nepovažujeme za různé, jestliže se liší pouze pořadím sčítanců. Rozklady vypište v libovolném pořadí a s libovolným pořadím sčítanců v jednotlivých rozkladech.

Příklad: Číslo $N = 6$ je možné rozložit na součet přirozených sčítanců těmito jedenácti způsoby:

$1+1+1+1+1+1$
$2+1+1+1+1$
$2+2+1+1$
$2+2+2$
$3+1+1+1$
$3+2+1$
$3+3$
$4+1+1$

$4 + 2$
$5 + 1$
6

Ze zkušenosti snadno odhadneme, že s rostoucí hodnotou N počet existujících rozkladů velmi rychle narůstá. Budeme je proto jistě chtít vytvářet v nějakém předem stanoveném pořadí, abychom je mohli rovnou vypisovat. Bylo by nešikovné, kdybychom museli všechny nalezené rozklady uchovávat v nějaké rozsáhlé datové struktuře a každý nový rozklad pak nejprve kontrolovat, zda se už v minulosti stejný rozklad neobjevil (neboť nechceme opakovaně vypisovat stejné rozklady ani rozklady lišící se pouze pořadím sčítanců). Ukážeme si tedy, jak na to. V našem řešení zvolíme takové pořadí vytvářených rozkladů a takové pořadí sčítanců v rámci jednotlivých rozkladů, jaké je použito v příkladu u zadání úlohy.

V první řadě potřebujeme zabránit tomu, aby se vypisovaly rozklady lišící se pouze pořadím sčítanců – tedy rozklady ve skutečnosti stejné, ale na první pohled vypadající odlišně. Pro $N = 6$ se jedná například o rozklady

$$4 + 1 + 1, \quad 1 + 4 + 1, \quad 1 + 1 + 4.$$

V každé skupině rozkladů, které se liší pouze vzájemným pořadím sčítanců, je vždy obsažen právě jeden takový, v němž jsou sčítanci uspořádány v nerostoucím pořadí. Tento rozklad vybereme jako „reprezentanta“ příslušné skupiny a toho jediného z celé skupiny vždy vypíšeme. Bylo by ovšem zbytečně pomalé vytvářet všechny podobné rozklady a až následně je kontrolovat, zda jsou opravdu nerostoucí. Namísto toho budeme v našem algoritmu rovnou generovat pouze nerostoucí rozklady. To znamená, že již při sestavování jednotlivých rozkladů budeme dodržovat zásadu, že další sčítanec může být roven nejvýše hodnotě dosud posledního sčítance v tomto rozkladu.

Když už jsme se omezili na rozklady s nerostoucím pořadím sčítanců, budeme je chtít vytvářet systematicky v nějakém jasně stanoveném pořadí. Přirozeným řešením tohoto požadavku je zvolit lexikografické uspořádání, neboli stejné řazení, jaké se používá u hesel ve slovníku. Začneme s rozklady začínajícími jedničkou, následují rozklady začínající dvojkou, atd. Mezi rozklady se stejným prvním sčítancem určuje jejich vzájemné pořadí druhý sčítanec. Pokud se shoduje i ten, vzájemné pořadí se určí podle třetího sčítance, atd.

Ke generování všech rozkladů čísla N v lexikografickém uspořádání použijeme přirozeným způsobem rekurzivní proceduru. Ta bude dostávat

v prvním parametru hodnotu, kterou ještě zbývá rozložit, a ve druhém parametru informaci, kolikátý sčítanec rozkladu má procedura vytvořit (označme tuto pozici P). Hlavní program zavolá proceduru s parametry N (na začátku výpočtu je třeba rozložit celé N) a 1 (rozklad začínáme prvním sčítancem). Procedura bude ukládat jednotlivé členy rozkladu do pole a kdykoliv bude celý rozklad v poli vytvořen, procedura ho vypíše. Tento stav se pozná podle toho, že už nebývá nic rozkládat (tzn. první parametr při zavolání procedury má hodnotu 0). Pokud ještě zbývá něco rozložit, procedura se pokusí všemi přípustnými způsoby stanovit hodnotu P-tého sčítance a pro každou takovou možnost provede rekurzivní volání s patřičně upravenými parametry, jak je vidět níže v naší programové ukázce. Na pozici P ve vytvářeném rozkladu procedura zkouší postupně dát hodnoty 1, 2, 3, ... atd. Tento P-tý sčítanec může být roven nejvýše hodnotě, kterou ještě zbývá rozložit, a také nejvýše hodnotě předchozího sčítance (aby byl sestavený rozklad nerostoucí). Obě uvedená omezení musíme dodržet zároveň, takže z nich musíme vzít minimum.

Popsané řešení si ukážeme zapsané v programovacím jazyce Pascal:

```
program RozkladyCisla;
{Zadané kladné celé číslo N rozloží všemi způsoby na součet
 kladných celých čísel. Na pořadí rozkladů ani sčítanců
 v rozkladu nezáleží. Např. pro N=5:
 1+1+1+1+1 = 2+1+1+1 = 2+2+1 = 3+1+1 = 3+2 = 4+1 = 5}

const MaxN = 125;                        {maximální přípustné N}
var   N: integer;                        {rozkládané číslo}
      A: array[0..MaxN] of integer;      {uložení rozkladů}

function Min(A, B: integer): integer;
  {pomocná funkce na výpočet minima ze dvou celých čísel A, B}
  begin
    if A > B then Min := B else Min := A
  end; {function Min}

procedure Rozloz(Zbytek, P: integer);
{Zbytek = kolik zbývá rozložit, P = kolikátý sčítanec vytváříme}
  var I: integer;
  begin
  if Zbytek = 0 then    {rozklad je hotov}
    begin
    for I:=1 to P-1 do write(A[I]:3);
    writeln
    end
  else  {přidat další člen rozkladu - v pořadí P-tý}
    begin
```

```
    for I:=1 to Min(Zbytek, A[P-1]) do
      begin
      A[P] := I;   Rozloz(Zbytek-I, P+1)
      end;
    end;
  end; {procedure Rozloz}

begin
write('Rozkladane cislo (1-', MaxN, '): ');
readln(N);
A[0] := MaxN + 1; {technický trik}
Rozloz(N, 1);  {chceme rozložit celé N, začínáme 1. sčítancem}
end.
```

Můžete si vyzkoušet, že pro vyšší hodnoty N je výpočet uvedeného programu velmi pomalý. To není způsobeno tím, že by byl program navržen nevhodně či nešikovně, ale skutečností, že velká N se dají rozložit na součet přirozených sčítanců mnoha různými způsoby (počet existujících rozkladů roste exponenciálně vzhledem k N) a výstup programu proto nutně musí být rozsáhlý. Například pro $N = 101$ existuje více než 200 milionů různých rozkladů a pro $N = 121$ jsou už těchto rozkladů více než 2 miliardy.

Podívejme se nyní, co by se na řešení naší úlohy změnilo, kdybychom měli za úkol určit pro dané přirozené číslo N pouze počet všech jeho rozkladů na součet přirozených sčítanců, ale jednotlivé rozklady bychom nemuseli vypisovat. Jedna možná cesta, jak lze takto upravenou úlohu vyřešit, spočívá v použití výše uvedeného rekurzivního postupu. Jedinou změnou bude nahrazení výpisu pole A v proceduře *Rozloz* zvýšením počítadla nalezených rozkladů o 1. Po skončení výpočtu pak stačí vypsat hodnotu tohoto počítadla.

Program upravený uvedeným způsobem vykoná při výpočtu stejný počet kroků, jako původní řešení naší úlohy s výpisem všech rozkladů. Bude mít opět exponenciální asymptotickou časovou složitost a pro velká N bude tedy velmi pomalý. Pokud si ho ovšem prakticky vyzkoušíte na počítači pro menší hodnoty N, zjistíte, že se oproti původnímu řešení výpočet znatelně urychlil, neboť již není zdržován relativně pomalým vypisováním jednotlivých rozkladů na obrazovku.

K řešení upravené úlohy však můžeme přistoupit odlišným způsobem. Nebudeme již generovat a počítat jednotlivé rozklady, ale budeme počítat pouze to, co se od nás skutečně požaduje – jejich počet. Dokážeme to provést s polynomiální časovou složitostí, tedy dostatečně rychle i pro velké N. K nalezení výsledku použijeme techniku dynamického programování.

Zavedeme si dvojrozměrnou tabulku T velikosti $N \times N$, ve které celo-

číselný údaj $T_{i,j}$ určuje, kolik existuje různých rozkladů čísla i na součet přirozených sčítanců, má-li největší z těchto sčítanců hodnotu j. Logický význam mají pouze údaje v trojúhelníku tabulky T pod hlavní diagonálou, tzn. prvky $T_{i,j}$ pro $i \geq j$. Při zkoumání rozkladů čísla i totiž pochopitelně nejvyšší sčítanec v rozkladu nemůže být větší než i. Některé prvky tabulky T jsou předem známé – v prvním sloupci, na hlavní diagonále i těsně pod hlavní diagonálou budou jistě samé jedničky. Je tomu tak proto, že libovolné číslo i lze jediným způsobem rozložit na součet i sčítanců rovných 1 (hodnota $T_{i,1}$), jediným způsobem ho můžeme zapsat jedním číslem rovným i (hodnota $T_{i,i}$) a jediným způsobem ho můžeme rozložit na součet sčítanců $i-1$ a 1 (hodnota $T_{i,i-1}$). Zbývající prvky tabulky T budeme vyplňovat po řádcích níže popsaným způsobem, tzn. postupně budeme zvyšovat hodnotu rozkládaného čísla. Po vyplnění celé tabulky najdeme výsledek na jejím posledním řádku. Podle definice tabulky T je hledaný počet všech různých rozkladů čísla N na součet přirozených sčítanců roven součtu všech čísel na N-tém řádku tabulky.

Zbývá ukázat, jak při vyplňování i-tého řádku tabulky T spočítáme hodnoty $T_{i,j}$ pro $1 < j < i-1$. V rozkladu čísla i musí být obsažen aspoň jeden sčítanec rovný j, zbývající hodnotu $i-j$ pak chceme rozložit všemi způsoby na součet sčítanců rovných nejvýše j. Počet těchto rozkladů ovšem při vyplňování i-tého řádku tabulky T již známe – je roven součtu čísel uložených v tabulce T na řádku $i-j$ ve sloupcích od 1 do j. Stačí tedy těchto j čísel sečíst a tím dostaneme správnou hodnotu $T_{i,j}$.

Popsané řešení má paměťovou složitost $O(N^2)$, která je dána velikostí vyplňované tabulky T. Z popisu algoritmu vidíme, že postupně počítáme řádově N^2 prvků tabulky T a každý z nich spočítáme v čase $O(N)$. Asymptotická časová složitost popsaného řešení je proto $O(N^3)$. To je dostatečně rychlý výpočet i pro vyšší hodnoty N. Pamatujte ovšem na to, že s rostoucím N počet existujících rozkladů čísla na součet přirozených sčítanců velmi rychle narůstá. Již jsme uvedli, že pro $N = 121$ existuje těchto rozkladů více než dvě miliardy, což je zhruba rozsah 4-bytového celočíselného typu se znaménkem (typ longint v Turbo Pascalu, integer ve FreePascalu). Při použití takovéhoto datového typu proto dojde již pro N větší než 121 k aritmetickému přetečení. Efektivní řešení upravené úlohy určit počet rozkladů zadaného přirozeného čísla na součet přirozených sčítanců si na závěr naprogramujeme:

```
program PocetRozkladuCisla;
{Určí počet různých rozkladů čísla N na součet
 přirozených sčítanců}
```

```
const MaxN = 125;          {maximální přípustné N}
var   N: integer;          {rozkládané číslo}
      T: array[1..MaxN, 1..MaxN] of longint;
      {T[i,j] = počet rozkladů čísla "i",
       v nichž nejvyšší sčítanec je "j"}
      i, j, k: integer;
      v: longint;
      {výsledný počet rozkladů - může jich být hodně}

begin
write('Rozkladane cislo (1-', MaxN, '): ');
readln(N);
{Známé hodnoty v tabulce T:}
T[1,1]:=1;
for i:=2 to N do
  begin T[i,1]:=1; T[i,i]:=1; T[i,i-1]:=1 end;
for i:=1 to N-1 do
  for j:=i+1 to N do T[i,j]:=0;
{Dynamickým programováním počítáme po řádcích
 zbývající hodnoty tabulky T:}
for i:=4 to N do
  for j:=2 to i-2 do
    begin
    {počítáme T[i,j: v rozkladu čísla "i" použijeme sčítanec
     "j", zbývá ještě rozložit "i-j" a v tomto rozkladu mohou
     být další sčítanci z rozmezí hodnot 1 do "j"}
    v:=0;
    for k:=1 to j do v:=v + T[i-j, k];
    T[i,j]:=v
    end;
v:=0;
for j:=1 to N do v:=v + T[N,j];
writeln('Pocet ruznych rozkladu: ', v);
end.
```

Poznámka k úlohám o směsích

V článku [1] z roku 2011 jsem v informatické části časopisu MFI uvedl program na řešení různých typů slovních úloh o směsích. Většinou jde o hmotnosti a ceny komponent a výsledné směsi, jejichž souvislost je dána vztahy

$$\begin{aligned} m_1 + m_2 &= m_3 \\ m_1 c_1 + m_2 c_2 &= m_3 c_3. \end{aligned} \tag{1}$$

Pro ilustraci použití programu jsem uvedl zadání několika úloh, jaké bývají v učebnicích matematiky a sbírkách příkladů, a tak se tam objevila i tato úloha o slitinách.

Úloha 2

Ze dvou kovů s hustotami 7,4 g/cm^3 a 8,2 g/cm^3 máme připravit 0,5 kg slitiny s hustotou 7,6 g/cm^3. Kolik gramů každého kovu je k tomu zapotřebí?

Podobnou ilustrativní úlohu jsem použil i v knize [2]:

Úloha 21.2.7

Určete množství a hustotu slitiny dvou kovů, když prvního máme 300 g a jeho hustota je 7,2 g/cm^3, druhého 250 g a má hustotu 7,8 g/cm^3?

Při své návštěvě Olomouce v dubnu 2015 mě upozornil kolega doc. RNDr. Josef Polák, CSc. z FAV ZČU v Plzni, že u slitin není použití druhé rovnice (1) na místě (c_i jsou hustoty kovů), protože obecně objem slitiny není roven součtu objemů komponent. O upřesnění tohoto faktu jsem požádal doc. Mgr. Pavla Banáše, Ph.D., z Katedry fyzikální chemie na Přírodovědecké fakultě UP. Jeho vyjádření lze shrnout takto:

„Druhá rovnice (1) je pro slitiny jen aproximativní a platila by pouze v ideálním případě, tj. kdyby oba kovy i jejich slitina měly stejné krystalografické parametry (symetrii i mřížkové parametry), a tedy i stejné molární objemy. Obecně tomu tak není, ale odchylka od takové zidealizované varianty (kdy by platilo, že objem slitiny je roven objemu jejích komponent) je relativně malá, pro většinu případů kolem několika málo procent. Čili s vědomím všech těchto limitací to pořád může být užitečný postup pro hrubý odhad."

Prosím tedy čtenáře [1] a [2], aby uvedené úlohy chápali tímto způsobem a oběma jmenovaným kolegům děkuji za upozornění a za vysvětlení. Vyučujícím matematiky lze doporučiti, aby v případě, že se chtějí slitinami zabývat, upozornili žáky na to, že slitiny fungují chemicky jinak než směsi a že v tomto případě je výsledek jen přibližný.

Literatura

[1] *Trávníček, S.*: Úlohy o směsích. MFI, 20 (2010/11), č. 6, s. 361–369.

[2] *Trávníček, S.*: Pojďme na to s matematikou (a někdy i s počítačem). Vydavatelství UP, Olomouc, 2013.

Stanislav Trávníček