Vybalancovaný úsek (Úlohy z MO kategorie P, 41. část)

PAVEL TÖPFER

Matematicko-fyzikální fakulta UK, Praha

V dnešním pokračování našeho dlouhodobého seriálu věnovaného zajímavým problémům z Matematické olympiády kategorie P (programování) se budeme věnovat jedné poměrně snadné úloze z celostátního kola 43. ročníku MO (školní rok 1993/94). Patří do velmi oblíbené a často využívané kategorie úloh typu „práce s posloupnostmi čísel“. Ukážeme si, že i takto jednoduchý problém lze řešit různými způsoby a s různou časovou složitostí. Nejprve se jako obvykle seznámíme s přesným zadáním úlohy.

* * * * * * * * * * * *

Souvislý úsek v posloupnosti celých čísel nazveme *vybalancovaným úsekem*, jestliže počet kladných a počet záporných čísel v tomto úseku se sobě rovnají.

Je dáno celé číslo N ($1 \leq N \leq 1\,000\,000$) a posloupnost N celých čísel. Napište program, který určí délku maximálního vybalancovaného úseku v zadané posloupnosti čísel. Při návrhu programu se zaměřte na dosažení co největší rychlosti výpočtu.

Příklad: Pro $N = 10$ a posloupnost čísel 8 6 4 7 −5 −3 2 0 −1 9 bude výsledkem číslo 7, neboť nejdelší vybalancovaný úsek 4 7 −5 −3 2 0 −1 (případně jiný stejně dlouhý vybalancovaný úsek 7 −5 −3 2 0 −1 9) je tvořen sedmi čísly.

* * * * * * * * * * * *

Prvky zadané posloupnosti si označíme $a_1, a_2, \ldots, a_N$. Pro snazší popis algoritmů si zavedeme pojem *balance* úseku. Balance úseku $a_i, \ldots, a_j$ udává, o kolik více je v tomto úseku posloupnosti kladných čísel než záporných. Je-li v nějakém úseku posloupnosti záporných čísel více než kladných, potom balance tohoto úseku má zápornou hodnotu. Všimněte si, že případné nulové prvky posloupnosti se do balance nijak nezapočítávají, ovlivňují ovšem délku úseku. Vybalancovaným úsekem tedy rozumíme takový úsek, který má nulovou balanci.

Pokud si celou posloupnost čísel uložíme do pole, abychom se mohli k jejím prvkům opakovaně vracet, úloha má primitivní řešení spočívající v otestování všech souvislých úseků dané posloupnosti. Takových úseků existuje $O(N^2)$, neboť N způsoby můžeme zvolit začátek úseku i a pro každou volbu začátku můžeme $O(N)$ způsoby zvolit konec úseku j. Projít úsek $a_i, \ldots, a_j$ a spočítat jeho balanci představuje práci $O(N)$, takže celková časová složitost tohoto triviálního algoritmu je $O(N^3)$.

```
program Vybalancovany_Usek_1;
{kubická časová složitost}

const MaxN = 1000000;
var N: integer;          {počet čísel v posloupnosti}
    A: array[1..MaxN] of integer;    {zadaná posloupnost}
    b: integer;          {balance úseku i..j}
    Max: integer;        {délka max. vybalancovaného úseku}
    i, j, k: integer;

begin
Max:=0;
read(N);
for i:=1 to N do
  read(A[i]);
for i:=1 to N do         {začátek úseku}
  for j:=i to N do       {konec úseku}
    begin
    b:=0;
    for k:=i to j do
      if A[k] > 0 then
        b:=b+1
      else if A[k] < 0 then
        b:=b-1;
    if (b = 0) and (j-i+1 > Max) then
      Max:=j-i+1
    end;

if Max = 0 then
  writeln('Vybalancovany usek neexistuje!')
else
  writeln('Delka maximalniho vybalancovaneho useku: ', Max)
end.
```

Výpočet můžeme urychlit lepší organizací práce. V předchozím řešení jsme ověřovali balanci každého úseku zvlášť. Známe-li však balanci nějakého úseku $a_i, \ldots, a_j$ a chceme určit balanci úseku o jeden prvek delšího $a_i, \ldots, a_{j+1}$, je zbytečné procházet tento nový úsek celý od začátku. Jednodušší a rychlejší je využít předchozí známou hodnotu balance a pouze ji v konstantním čase upravit podle znaménka přidaného posledního prvku a_{j+1}. V programu tedy budeme postupně N způsoby volit začátek úseku i a pro každou volbu začátku zvolíme $\mathrm{O}(N)$ způsoby konec úseku j. Výpočet balance úseku $a_i, \ldots, a_j$ provádíme průběžně vždy s využitím známé balance předchozího úseku, jak je popsáno výše. Celková časová složitost takto upraveného algoritmu se tím sníží na $\mathrm{O}(N^2)$.

```
program Vybalancovany_Usek_2;
{kvadratická časová složitost}

const MaxN = 1000000;
var N: integer;          {počet čísel v posloupnosti}
    A: array[1..MaxN] of integer;    {zadaná posloupnost}
    b: integer;          {balance úseku i..j}
    Max: integer;        {délka max. vybalancovaného úseku}
    i, j, k: integer;

begin
Max:=0;
read(N);
for i:=1 to N do
  read(A[i]);
for i:=1 to N do         {začátek úseku}
  begin
  b:=0;
  for j:=i to N do       {konec úseku}
    begin
    if A[j] > 0 then
      b:=b+1
    else if A[j] < 0 then
      b:=b-1;
    if (b = 0) and (j-i+1 > Max) then
      Max:=j-i+1
    end
  end;
```

```
if Max = 0 then
  writeln('Vybalancovany usek neexistuje!')
else
  writeln('Delka maximalniho vybalancovaneho useku: ', Max)
end.
```

To ovšem stále ještě není časově optimální řešení. Úlohu dokážeme vyřešit s lineární časovou složitostí vzhledem k délce posloupnosti N, postup práce nyní ale bude poněkud odlišný a lišit se bude také způsob uložení dat. Pro každé číslo a_i si nejprve spočítáme balanci počátečního úseku posloupnosti délky i, tzn. úseku $a_1, a_2, \ldots, a_i$. Tento údaj označíme e_i a dodefinujeme hodnotu $e_0 = 0$ (počáteční úsek nulové délky má balanci 0). Všechny hodnoty $e_1, e_2, \ldots, e_N$ dokážeme snadno spočítat při jednom průchodu posloupností v čase $\mathrm{O}(N)$, postup jsme si ukázali v předchozím řešení úlohy. Pro každé pořadové číslo prvku posloupnosti i od 1 do N bude jistě platit nerovnost $-i \leq e_i \leq i$.

Všimněte si, že úsek posloupnosti $a_i, \ldots, a_j$ je vybalancovaný právě tehdy, když prvky a_{i-1}, a_j mají stejnou e-hodnotu, tj. když platí rovnost $e_{i-1} = e_j$. Hledáme-li maximální vybalancovaný úsek v posloupnosti, stačí tedy nalézt mezi e-hodnotami dvě stejná čísla $e_{i-1} = e_j$ $(1 \leq i \leq j \leq N)$ co nejvíce od sebe vzdálená. Jejich vzájemná vzdálenost v posloupnosti pak přímo udává délku maximálního vybalancovaného úseku. Jestliže neexistuje žádná dvojice stejných hodnot $e_{i-1} = e_j$, jsou všechna čísla v posloupnosti kladná nebo všechna záporná. Taková posloupnost tudíž neobsahuje žádný vybalancovaný úsek.

Bylo by jistě možné uložit si všechna čísla e_i do pole a v tomto poli následně vyhledávat dvojice stejných hodnot. Takový postup je ale zbytečně pomalý. Vyhledávání dvojic stejných čísel v poli má kvadratickou asymptotickou časovou složitost, bylo by to vlastně jenom jinak realizované předcházející řešení úlohy. Výhodnější bude ukládat si informace o jednotlivých e-hodnotách „inverzně“. Vytvoříme si pomocné pole F, pro všechna i od 1 do N budeme postupně počítat čísla e_i a pro každou nově získanou e-hodnotu si uložíme do pole F příslušný index i, tzn. položíme $\mathrm{F}[e_i] = i$. Údaj $\mathrm{F}[m]$ nám tedy říká, kde jsme se setkali poprvé s e-hodnotou m (na kterém indexu posloupnosti). Vzhledem k podmínce $-i \leq e_i \leq i$ musí být pole F indexováno od $-N$ do N. Využívat z něj budeme pouze jistý souvislý úsek s indexy od X do Y. Během zpracovávání prvků posloupnosti se velikost tohoto úseku může zvětšovat. V každém okamžiku zpracování posloupnosti platí, že všechna celá čísla z in-

tervalu $\langle X, Y\rangle$ přestavují právě všechny e-hodnoty, s nimiž jsme se dosud setkali.

Na začátku výpočtu položíme $X = 0$, $Y = 0$ a $\mathrm{F}[0] = 0$, což odpovídá počátečnímu prázdnému úseku posloupnosti s balancí 0. Postupně budeme číst čísla tvořící zadanou posloupnost a zpracovávat je. Po přečtení čísla a_i nejprve spočítáme hodnotu e_i výše uvedeným způsobem na základě zaznamenané předchozí hodnoty e_{i-1}. Jestliže $e_i < X$, musí být nutně $e_i = X - 1$. Zmenšíme tedy X o 1 a zaznamenáme výskyt nové e-hodnoty $\mathrm{F}[X] = i$. Podobně pro $e_i > Y$ zvětšíme Y o 1 a uložíme hodnotu $\mathrm{F}[Y] = i$. Je-li číslo e_i z intervalu $\langle X, Y\rangle$, pak stejná e-hodnota byla již nalezena někdy dříve, a sice poprvé při zpracování prvku s indexem $\mathrm{F}[e_i]$. To ale znamená, že nejdelší vybalancovaný úsek končící prvkem a_i má délku $i - \mathrm{F}[e_i]$. Nyní už jenom zbývá porovnat tuto délku s průběžně ukládaným maximem z délek nalezených vybalancovaných úseků.

Popsaný algoritmus je jistě konečný, jediný jeho cyklus se opakuje pro každé číslo zadané posloupnosti, tedy N-krát. Zpracování každého čísla vyžaduje vykonat jen konstantní počet operací, takže celé řešení má lineární časovou složitost vzhledem k N. Oproti oběma pomalejším řešením zde navíc potřebujeme pomocné pole F. Na druhou stranu ale zase ušetříme paměť tím, že posloupnost čísel můžeme zpracovávat průběžně při čtení dat ze vstupu a nemusíme si ji vůbec ukládat do paměti. Správnost algoritmu vyplývá ze skutečnosti, že systematicky prozkoumáme maximální vybalancované úseky končící každým z prvků posloupnosti a z jejich délek vybereme maximum.

```
program Vybalancovany_Usek_3;
{lineární časová složitost}

const MaxN = 1000000;
var N: integer;          {počet čísel v posloupnosti}
    a: integer;          {právě zpracovávané číslo}
    i: integer;          {pořadí zpracovávaného čísla}
    e: integer;          {e-hodnota zpracovávaného čísla}
    X, Y: integer;       {meze rozsahu nalezených e-hodnot}
    F: array[-MaxN..MaxN] of integer;
            {indexy prvních výskytů jednotlivých e-hodnot}
    Max: integer;        {délka max. vybalancovaného úseku}

begin
X:=0; Y:=0;
```

```
F[0]:=0;
e:=0;
Max:=0;

read(N);
for i:=1 to N do
  begin
  read(a);
  if a > 0 then
    e:=e+1
  else if a < 0 then
    e:=e-1;
  if e < X then                {nová nejmenší e-hodnota}
    begin X:=X-1; F[X]:=i end
  else if e > Y then           {nová největší e-hodnota}
    begin Y:=Y+1; F[Y]:=i end
  else                         {další výskyt stejné e-hodnoty}
    if i-F[e] > Max then
      Max:=i-F[e]
  end;

if Max = 0 then
  writeln('Vybalancovany usek neexistuje!')
else
  writeln('Delka maximalniho vybalancovaneho useku: ', Max)
end.
```