INFORMATIKA

Opakované úseky posloupnosti
(Úlohy z MO – kategorie P, 31. část)

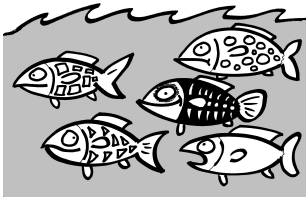
PAVEL TÖPFER

Matematicko-fyzikální fakulta UK, Praha

V našem cyklu zajímavých úloh z Matematické olympiády – kategorie P tentokrát zamíříme do daleké historie. Navštívíme 36. ročník MO, který se konal ve školním roce 1986/87, tedy v době, kdy kategorie P teprve vznikala. Na ukázku jsme pro vás vybrali jednu ze soutěžních úloh tehdejšího celostátního kola. Jako obvykle začneme zadáním této úlohy, které uvádíme téměř v původní podobě, pouze s drobnými formulačními úpravami. Sami vidíte, že text úlohy je o dost kratší a jednodušší oproti tomu, jak obvykle vypadají zadání soutěžních úloh v současné době.

Úloha

Nalezněte a dokažte co nejlepší algoritmus, který pro libovolnou konečnou posloupnost celých čísel délky N a pro zadanou dvojici kladných celých čísel K, L určí, zda daná posloupnost obsahuje souvislý úsek délky K, který se v ní vyskytuje alespoň L-krát. Jednotlivé výskyty se mohou částečně překrývat.



Příklad. Pro posloupnost 1, 2, 1, 2, 1, 2, 1 a $K = 3$, $L = 3$ program odpoví ANO, protože se v ní třikrát vyskytuje úsek 1, 2, 1.

Úlohu můžeme snadno vyřešit primitivním přímočarým způsobem. Postupně budeme brát všechny souvislé úseky délky K a pro každý z nich zvlášť spočítáme všechny jeho výskyty v posloupnosti. Pro každý takový

úsek délky K tedy uvažujeme všechny polohy, kde by mohl v posloupnosti začínat, a pro každou volbu jeho polohy pak úsek procházíme a testujeme, zda se jednotlivá čísla zkoumaného úseku shodují s odpovídajícími čísly v posloupnosti. Pokud se shodují všechna čísla tvořící zkoumaný úsek, našli jsme další výskyt tohoto úseku v posloupnosti.

Zamyslíme se nad časovou složitostí uvedeného algoritmu. Počet souvislých úseků délky K vybraných z posloupnosti délky N je přesně $N-K+1$. Pro každý z nich zkoumáme $N-K+1$ možných umístění v rámci posloupnosti. Kontrola každého umístění úseku pak představuje sekvenční průchod tímto úsekem a vyžaduje tedy až K porovnání. Uvážíme-li, že délka úseku K může být rovna třeba $N/2$, tedy $O(N)$, můžeme časovou složitost našeho algoritmu odhadnout jako $O(N^3)$.

Řešení založené na popsaném postupu velmi snadno naprogramujeme. Předpokládejme, že na standardním vstupu programu je nejprve zadána trojice údajů v pořadí N, K, L a za nimi následuje N členů posloupnosti. Výsledkem výpočtu bude buď zpráva ANO, nebo zpráva NE vypsaná na standardní výstup.

```
program Useky1;
const MaxN = 10000;  {maximální délka posloupnosti}
var A: array[1..MaxN] of integer;  {zkoumaná posloupnost}
    N, K, L: integer;
    Pocet, i, j, d: integer;
    Shoda: boolean;
begin
read(N, K, L);
for i:=1 to N do read(A[i]);
for i:=1 to N-K-L+2 do  {začátek zkoumaného úseku}
  begin
  Pocet:=0;
  for j:=i to N-K+1 do  {začátek možného umístění}
    begin
    Shoda:=true;
    for d:=0 to K-1 do  {porovnávání znaků}
      if A[i+d] <> A[j+d] then
        begin Shoda:=false; break end;
    if Shoda then inc(Pocet);  {máme další výskyt}
    if Pocet = L then break
    end;
  if Pocet = L then break
  end;
if Pocet = L then writeln('ANO')
             else writeln('NE')
end.
```

Můžeme se pokusit o zrychlení výpočtu tím, že zavedeme vhodnou pomocnou datovou strukturu S. Do ní budeme ukládat všechny zpracované úseky délky K a s každým také údaj o počtu jeho dosud nalezených výskytů. Postupně budeme brát jednotlivé úseky vstupní posloupnosti a každý nejprve vyhledáme ve struktuře S. Pokud ho najdeme, zvýšíme mu o 1 počet výskytů. Pokud ho nenajdeme, vložíme ho do S společně s informací o jeho prvním nalezeném výskytu. Pro každý zkoumaný úsek tedy již nemusíme procházet celou posloupnost, ale pouze se ho pokusíme vyhledat ve struktuře S mezi dosud prozkoumanými různými úseky délky K.

Časová složitost takovéhoto řešení závisí na typu použité struktury S. Kdybychom zpracovávali posloupnost znaků, mohli bychom na uložení všech prozkoumaných K-tic použít třeba strukturu písmenkového stromu (trie) – viz např. [1] nebo [2]. Vyhledávání bychom tím značně urychlili, pro každý z $N-K+1$ zkoumaných úseků posloupnosti bychom procházeli pouze cestu stromem délky K od kořene k listu, takže celé řešení by mělo časovou složitost $O(N \cdot K)$, resp. $O(N^2)$. V našem případě ovšem zpracováváme posloupnost celých čísel. Pokud bychom se pro ně pokusili vytvořit nějakou stromovou strukturu analogickou trii, museli bychom v každém uzlu prohledávat seznam čísel délky řádově N a oproti počátečnímu primitivnímu řešení bychom tak časově neušetřili téměř nic. Určité úspory bychom dosáhli pouze v případě, že by se čísla v posloupnosti hodně opakovala.

Přesto existuje ještě lepší řešení naší úlohy, které dosahuje i v nejhorším případě asymptotické časové složitosti $O(N^2)$. Jako zajímavost uveďme, že při soutěži celostátního kola 36. ročníku Matematické olympiády – kategorie P na řešení s kvadratickou časovou složitostí nikdo ze soutěžících studentů nepřišel. Všechna funkčně správná řešení odevzdaná v soutěži měla časovou složitost $O(N^3)$.

Vysvětlíme si nejprve základní myšlenku algoritmu. Představme si čtvercovou tabulku T o rozměrech $N \times N$, jejíž řádky i sloupce jsou postupně označeny jednotlivými čísly ze zadané posloupnosti (řádky označíme shora dolů, sloupce zleva doprava). Do tabulky zapíšeme čísla 0 a 1 následovně: pokud má políčko stejné označení řádku i sloupce, zapíšeme do něj číslo 1, v opačném případě do něj zapíšeme číslo 0. Platí tedy, že $T[i,j] = 1$, právě když $A[i] = A[j]$, jinak $T[i,j] = 0$. V takto vyplněné tabulce budou jistě na hlavní diagonále samé jedničky. Obsah tabulky je symetrický podle hlavní diagonály, dále se proto budeme zajímat pouze o trojúhelníkovou oblast tabulky nad hlavní diagonálou.

Naši pozornost nyní zaměříme na souvislé řady jedniček, které mají šikmý směr rovnoběžný s hlavní diagonálou. Každá taková řada jedniček délky D ležící mimo hlavní diagonálu představuje opakující se výskyt nějakého úseku čísel délky D v zadané posloupnosti. Jestliže tedy existuje takových K po sobě jdoucích řádků tabulky T, že jimi všemi prochází alespoň L souvislých šikmých řad jedniček, program má odpovědět ANO, v opačném případě odpoví NE. Tím je úloha vyřešena.

Další úpravy algoritmu jsou již jenom technického rázu a slouží ke zjednodušení a urychlení výpočtu. Tabulku T budeme vyplňovat po řádcích a místo jedniček do ní můžeme rovnou ukládat délky nalezených souvislých šikmých řad jedniček. Při stanovení hodnoty $T[i, j]$ budeme postupovat následovně:

$$\text{pokud } A[i] = A[j], \text{ položíme } T[i, j] = T[i-1, j-1] + 1,$$
$$\text{jinak } T[i, j] = 0.$$

V případě $i = 1$ se samozřejmě nemůžeme odkazovat na hodnoty z neexistujícího nultého řádku a do tabulky zapisujeme pouze jedničky a nuly. V průběhu vyplňování tabulky T budeme již zároveň kontrolovat, zda jsme do některého řádku zapsali L hodnot rovných alespoň K. Pokud se to stane, výpočet ihned ukončíme a vypíšeme odpověď ANO. Jestliže k tomu nikdy nedojde, program po vyplnění celé tabulky odpoví NE.

Z popisu algoritmu přímo plyne jeho časová složitost. Tabulka T má velikost N^2 a program v ní zaplňuje horní trojúhelník, tedy $O(N^2)$ prvků. Každou z těchto hodnot spočítáme provedením konstantního počtu operací. Časová složitost celého algoritmu je proto opravdu slibovaných $O(N^2)$ operací.

Ještě můžeme podstatným způsobem vylepšit paměťovou složitost popsaného algoritmu. Pro větší názornost jsme dosud stále hovořili o vyplňování dvojrozměrné tabulky T, při implementaci algoritmu ale ve skutečnosti dvojrozměrné pole vůbec nepotřebujeme. Tabulku vyplňujeme po řádcích a hodnoty na i-tém řádku vždy závisejí pouze na hodnotách z řádku $i - 1$. Spočítané hodnoty aktuálního řádku tabulky T také hned průběžně kontrolujeme. Při výpočtu se nikdy nepotřebujeme vracet ke starším řádkům a proto si je ani v programu nemusíme ukládat. Vystačíme tudíž s jednorozměrným polem, které bude postupně představovat jednotlivé řádky tabulky T, v i-tém kroku výpočtu budeme počítat a do pole ukládat hodnoty i-tého řádku tabulky. Jednotlivá čísla na řádku jenom musíme počítat a ukládat do pole vždy odzadu (zprava doleva), abychom

si novými hodnotami aktuálně počítaného řádku nepřepsali ty hodnoty z předchozího řádku, které ještě budeme potřebovat. Uvedenou úpravou jsme dosáhli paměťové složitosti $O(N)$.

Na závěr si opět ukážeme programovou realizaci popsaného algoritmu v Pascalu. Program má v mnohém podobnou strukturu jako počáteční primitivní řešení uvedené výše. Očekává také vstupní data ve stejném tvaru, tedy nejprve trojici údajů v pořadí N, K, L a poté N členů zkoumané posloupnosti. Jednorozměrné pole T použité v programu představuje postupně počítané jednotlivé řádky naší tabulky.

```
program Useky2;
const MaxN = 10000;  {maximální délka posloupnosti}
var A: array[1..MaxN] of integer;  {zkoumaná posloupnost}
    T: array[1..MaxN] of integer;  {řádky tabulky T}
    N, K, L: integer;
    Pocet, i, j: integer;
begin
read(N, K, L);
for i:=1 to N do read(A[i]);
for i:=1 to N do  {číslo řádku tabulky T}
  begin
  Pocet:=0;
  for j:=N downto i do  {číslo sloupce v tabulce T}
    begin
    if A[i] <> A[j] then T[j]:=0
    else if i = 1 then T[j]:=1
    else T[j]:=T[j-1] + 1;
    if T[j] >= K then inc(Pocet);  {máme další výskyt}
    if Pocet = L then break
    end;
  if Pocet = L then break
  end;
if Pocet = L then writeln('ANO')
             else writeln('NE')
end.
```

Literatura

[1] *Töpfer, P.*: Vyhledávání slov na stránce (Úlohy z MO – kategorie P – 28. část). MFI, roč. 21 (2011–2012), č. 5, s. 294–304.

[2] *Böhm, M. – Matějka, J. – Mareš, M. – Škoda, P.*: Recepty z programátorské kuchařky – Hledání v textu [online, cit. 2013–10–14]. Dostupné z: https://ksp.mff.cuni.cz/tasks/26/cook5.html.

(Autorkou ilustrace je Mgr. Jaroslava Palzerová Čermáková.)