Navlékání korálků
(Úlohy z MO – kategorie P, 38. část)

PAVEL TÖPFER

Matematicko-fyzikální fakulta UK, Praha

V našem seriálu článků o úlohách z Matematické olympiády kategorie P (programování) se tentokrát zastavíme u jedné praktické úlohy z ústředního kola loňského 67. ročníku MO (školní rok 2017/18). Jedná se o zajímavou modifikaci poměrně známé úlohy určit délku maximální rostoucí podposloupnosti vybrané ze zadané posloupnosti čísel. V domácím kole 67. ročníku MO soutěžící řešili právě tuto základní úlohu. V krajském kole na ni navázala o něco těžší varianta, kdy se ve vybrané rostoucí podposloupnosti připouští jeden pokles hodnoty. S uvedenou dvojicí úloh jsme se v Matematické olympiádě kategorie P již v dávné minulosti jednou setkali. Bylo to před dvaceti lety ve 47. ročníku soutěže. Úloze byl věnován také 20. díl našeho seriálu nazvaný Pokleslá podposloupnost [1].

Soutěžní úloha ústředního kola o navlékání korálků přirozeným způsobem navázala na úlohy z nižších kol soutěže. Seznámíme se nejprve s přesným zněním zadání. Úplné znění i původní autorské řešení úlohy můžete nalézt také na webu v archívu MO-P [2].

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

Ještě nedávno měla Natálka krásný náhrdelník, na kterém bylo n korálků. Korálky měly různé odstíny růžové barvy, postupně přecházely od tmavé do světlé. Náhrdelník se ale přetrhl a korálky se rozkutálely po zemi. Natálka teď drží prázdnou šňůrku a Petr (který se na přetržení náhrdelníku maličko podílel) klečí na zemi, sbírá korálky a jeden po druhém je Natálce podává. Ta může každý korálek buď navléknout na šňůrku (a to z libovolného konce), nebo ho může odložit do krabice s šitím.

Natálku při sbírání korálků napadla následující úloha: Co kdyby korálky zkusila navlékat tak, aby nakonec byly na šňůrce seřazené od nejtmavšího po nejsvětlejší? Možná se jí nepodaří takto navléknout všechny korálky, ale určitě bude zábavné zkusit navléknout jich co nejvíce.

Soutěžní úloha

Na vstupu je dána posloupnost $f_1, \ldots, f_n$: barvy korálků v tom pořadí, v jakém je Petr podával Natálce. Všechna f_i jsou *navzájem různá* přirozená čísla. Čím větší číslo, tím světlejší korálek představuje. Pro každý korálek si Natálka vybrala jednu ze tří možností: buď ho navlékla na levý konec šňůrky, nebo ho navlékla na pravý konec šňůrky, nebo ho odložila pryč. Víme, že na konci celého procesu měla šňůrku, na níž zleva doprava barva korálků přecházela z tmavé do světlé – tedy jim odpovídající čísla rostla.

Napište program, který určí, kolik korálků mohla mít nakonec Natálka na šňůrce nejvýše.

Formát vstupu a výstupu

Na prvním řádku vstupu je kladné celé číslo n. Na druhém řádku je n různých kladných celých čísel $f_1, \ldots, f_n$. Vypište jeden řádek s jedním číslem: maximální počet korálků, které mohly skončit na šňůrce.

Omezení a hodnocení

Je připraveno 10 sad testovacích vstupů, za každou můžete získat 1 bod. Ve všech sadách pro každé i platí $1 \leq f_i \leq 10^9$. Jednotlivé sady mají různou maximální hodnotu n: 6, 12, 20, 60, 100, 5 000, 17 000, 150 000, 250 000 a 500 000.

Příklady.

Vstup:	Výstup:
5 30 10 50 20 40	3

Jedno optimální řešení: Korálek 30 navleče Natálka na šňůrku (je jedno, z které strany), korálek 10 nepoužije, korálek 50 navleče zprava, korálek 20 navleče zleva a korálek 40 nepoužije.

Vstup:	Výstup:
6 10 20 9 21 8 22	6

Všechny korálky můžeme navléknout na šňůrku.

Vstup:	Výstup:
7 1 7 4 3 5 2 6	5

Zde je optimálním řešením nepoužít první dva korálky a z ostatních sestavit na šňůrce posloupnost 2, 3, 4, 5, 6.

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

Úlohu můžeme řešit hrubou silou, tedy prostým zkoušením všech možností. Podle zadání máme pro každý korálek tři možnosti, co s ním uděláme: buď ho na šňůrku navlékneme zleva, nebo ho navlékneme zprava, nebo ho vynecháme. Snadno nahlédneme, že vzhledem k požadovanému vzestupnému uspořádání korálků v náhrdelníku máme ve skutečnosti vždy nejvýše dvě možnosti: buď korálek navlékneme na šňůrku, nebo ho vynecháme. Dokud není na šňůrce žádný korálek, nemá smysl rozlišovat mezi navléknutím zleva a zprava – výsledek bude v obou případech stejný. Když už je na šňůrce navlečen první korálek a jeho hodnotu označíme P, pak následující korálky s hodnotou menší než P lze navléknout jedině zleva a následující korálky s hodnotou větší než P jedině zprava. Pro n korálků tak vyzkoušíme až 2^n možností a jako výsledek určíme maximální dosaženou délku náhrdelníku. Časová složitost tohoto řešení je proto $O(2^n)$.

Právě popsané řešení si nyní ukážeme zapsané v programovacím jazyce Pascal:

```
program Koralky1;
{zkoušení všech možností - exponenciální složitost}
const MaxN = 500000; {maximální počet korálků}
var n: integer;       {počet korálků}
    max: integer;     {maximální délka náhrdelníku}
    F: array[1..MaxN] of integer;  {hodnoty korálků}
    i: integer;

procedure Koralek(i, n, prvni, posledni, delka: integer;
                  var max: integer);
{pořadové číslo korálku, počet všech korálků, hodnota
prvního a posledního korálku v náhrdelníku, aktuální
délka náhrdelníku, maximální dosažená délka náhrdelníku}
var fi: integer;   {hodnota i-tého korálku}
begin
if i > n then      {hotovo}
  begin
  if delka > max then max := delka;
                   {nalezeno nové maximum}
  exit
  end;
```

```
fi := F[I];
if delka = 0 then
  {první navlečený korálek}
  Koralek(i+1, n, fi, fi, 1, max)
else if fi < prvni then
  {navlékneme zleva}
  Koralek(i+1, n, fi, posledni, delka+1, max)
else if fi > posledni then
  {navlékneme zprava}
  Koralek(i+1, n, prvni, fi, delka+1, max);
{korálek vynecháme}
Koralek(i+1, n, prvni, posledni, delka, max)
end; {procedura Koralek}

begin
read(n);
for i := 1 to n do read(F[i]);
max := 0;
Koralek(1, n, 0, 0, 0, max);
writeln(max)
end.
```

Popsané řešení jsme implementovali pomocí rekurzivní procedury Koralek. Význam jejích parametrů je uveden přímo v programu v komentářích. Nebudeme zde popisovat možnosti, jak lze zlepšit časovou složitost tohoto algoritmu pomocí techniky kešování (memoizace, ukládání známých výsledků) – zájemci si tato možná vylepšení jistě rozmyslí samostatně nebo si je mohou vyhledat ve [2]. Místo toho se budeme raději podrobněji věnovat efektivnějšímu postupu řešení.

Základem vzorového řešení naší úlohy je efektivní postup, jak v zadané posloupnosti čísel určit délku maximální vybrané rostoucí podposloupnosti. Pro posloupnost délky n dokážeme tuto úlohu vyřešit v čase $O(n \cdot \log n)$. Již v úvodu jsme uvedli, že s tímto algoritmem se soutěžící setkali v domácím kole soutěže, kde ho buď sami vymysleli, nebo se s ním seznámili následně ve vzorovém řešení. Naši čtenáři si mohou algoritmus přečíst v archívu úloh MO-P [2] ve vzorovém řešení úlohy 67-I-2, podrobnější rozbor najdete také v [3].

Ukážeme si zde alespoň hlavní myšlenku tohoto postupu. Budeme postupně procházet zadanou posloupnost čísel $f_1, \ldots, f_n$ a pro každý její prvek f_i určíme délku maximální rostoucí podposloupnosti vybrané z počátečního úseku délky i. Jak vzápětí uvidíme, výpočet každé z těchto hod-

not má časovou složitost $O(\log n)$ a protože ho provádíme n-krát, bude celková časová složitost popsaného algoritmu skutečně $O(n \cdot \log n)$.

V průběhu výpočtu si budeme v pomocném poli D udržovat následující informaci: číslo D_j udává, jakou nejmenší hodnotou může končit rostoucí podposloupnost délky j vybraná z úseku posloupnosti $f_1, \ldots, f_i$. Sporem snadno dokážeme, že hodnoty v poli D jsou vždy vzestupně uspořádané. V i-tém kroku výpočtu přidáme do uvažované posloupnosti prvek f_i, který sníží jednu vhodnou hodnotu uloženou v poli D – vždy tu, která je nejbližší vyšší než f_i. Díky uspořádání pole D dokážeme tuto hodnotu určit v logaritmickém čase binárním vyhledáváním.

Výsledné řešení dnešní úlohy získáme složením některých myšlenek uvedených výše v našem prvním řešení a algoritmu na určení délky maximální rostoucí vybrané podposloupnosti. Je jasné, že některý korálek navlékneme jako první. Nevíme který, ale můžeme vyzkoušet všech n takových možností. Pro každou volbu prvního navlečeného korálku (jeho hodnotu jsme si označili P) již víme, že následující korálky s hodnotou větší než P lze navlékat jedině zprava, zatímco korálky s hodnotou menší než P jedině zleva. Chceme navléknout co nejvíce korálků, takže mezi těmi většími než P chceme nalézt co nejdelší rostoucí podposloupnost a mezi těmi menšími než P co nejdelší klesající podposloupnost (což představuje prakticky stejný postup). To umíme provést v čase $O(n \cdot \log n)$, takže celkově dostáváme řešení s asymptotickou časovou složitostí $O(n^2 \cdot \log n)$. Stačí postupně vyzkoušet všech n možností volby prvního navlečeného korálku a pro každou z nich dvakrát vykonat algoritmus na určení délky maximální rostoucí (resp. ve druhém případě klesající) podposloupnosti, kterou ke zvolenému prvnímu korálku můžeme připojit.

Zdá se, že jsme s řešením úspěšně u konce, ale není tomu tak. Popsané řešení ještě stále provádí řadu výpočtů zbytečně vícekrát a vhodnou úpravou ho můžeme významně urychlit. Algoritmus z domácího kola můžeme snadno upravit tak, aby zpracovával zadanou posloupnost v opačném směru, tedy od konce. Pro každý prvek posloupnosti f_i, ovšem tentokrát pro i v pořadí od n dolů k 1, budeme určovat délku maximální rostoucí (resp. klesající) podposloupnosti začínající prvkem f_i. Uvedený postup opět zopakujeme dvakrát – zvlášť pro rostoucí a zvlášť pro klesající podpodsloupnost. Všechny spočítané hodnoty si uložíme do polí A, B pro další využití. Již víme, že celý tento výpočet má časovou složitost $O(n \cdot \log n)$.

Tím máme v polích A, B připraveno všechno potřebné pro vyřešení úlohy. Nyní už jen stačí jednou projít zadanou posloupnost čísel a pro

každý její prvek f_i (tj. první navlečený korálek) získáme v konstantním čase maximální délku náhrdelníku jako součet $\mathrm{A}_i + \mathrm{B}_i - 1$. Odečtení 1 je zde proto, abychom první navlečený korálek nezapočítali do výsledku dvakrát. Tato závěrečná fáze výpočtu má tedy časovou složitost $\mathrm{O}(n)$, celé řešení úlohy proto zvládneme v čase $\mathrm{O}(n \cdot \log n)$.

Popsané řešení zapíšeme opět také ve tvaru programu. Zvolili jsme takovou formu zápisu, aby byla jasná a dobře srozumitelná, i když program by bylo možné zapsat i o něco kratším a úspornějším způsobem.

```
program Koralky2;     {dynamické programování}
const MaxN = 500000; {maximální počet korálků}
var n: integer;       {počet korálků}
    max: integer;     {maximální délka náhrdelníku}
    F: array[1..MaxN] of integer;   {hodnoty korálků}
    A, B: array[1..MaxN] of integer;
    {A[i] - délka max. rostoucí podposloupnosti
                začínající korálkem "i"}
    {B[i] - délka max. klesající podposloupnosti
                začínající korálkem "i"}
    D: array[1..MaxN] of integer;
    {D[i] - jakou největší hodnotou může začínat
           rostoucí podposloupnost délky "i",
           resp. jakou nejmenší hodnotou může začínat
           klesající podposloupnost délky "i"}
    k: integer; {délka max. rostoucí,
                 resp. klesající podposloupnosti}
    i, j: integer;

begin
read(n);
for i := 1 to n do read(F[i]);
{Zpracujeme F[n] - úsek délky 1}
k := 1;
D[1] := F[n]; A[n] := 1; B[n] := 1;
{Rostoucí podposloupnost začínající korálkem číslo "i":}
for i := n-1 downto 1 do
  if F[i] < D[k] then
    begin  {lze prodloužit}
    k := k + 1;
    D[k] := F[i]; A[i] := k
    end
  else
```

```
    begin
    {Mezi D[1], D[2], ..., D[k] vyhledáme největší
     hodnotu D[j] takovou, že D[j] < F[i]. Hodnoty
     v poli D jsou uspořádané sestupně, takže to lze
     provést v logaritmickém čase binárním vyhledáváním.
     My zde pro jednoduchost zápisu použijeme sekvenční
     průchod s lineární časovou složitostí:}
    j := 1;
    while D[j] > F[i] do j := j + 1; {máme potřebné D[j]}
    D[j] := F[i];  A[i] := j
    end;
{Klesající podposloupnost začínající
 korálkem číslo "i" - analogicky:}
k := 1;
for i := n-1 downto 1 do
  if F[i] > D[k] then
    begin  {lze prodloužit}
    k := k + 1;
    D[k] := F[i];  B[i] := k
    end
  else
    begin
    j := 1;
    while D[j] < F[i] do j := j + 1; {máme potřebné D[j]}
    D[j] := F[i];  B[i] := j
    end;
{Nejdelší náhrdelník:}
max := 0;
for i := 1 to n do
  if A[i] + B[i] - 1 > max then max:= A[i] + B[i] - 1;
writeln(max)
end.
```

Literatura

[1] *Töpfer, P.*: Pokleslá podposloupnost (Úlohy z MO – kategorie P, 20. část). MFI, roč. 17 (2007–2008), č. 9, s. 553–556.

[2] http://mo.mff.cuni.cz/p/archiv.html

[3] *Libicher, I., Töpfer, P.*: Od problému k algoritmu a programu. Grada, Praha, 1992.