INFORMATIKA

Aritmetický průměr posloupnosti (Úlohy z MO kategorie P, 37. část)

PAVEL TÖPFER
MFF UK, Praha

V našem dlouhodobém seriálu o zajímavých úlohách z Matematické olympiády kategorie P (programování) jsme se již vícekrát věnovali různým úlohám o číselných posloupnostech. Tentokrát téma posloupností obohatíme o jednu úlohu z krajského kola 63. ročníku MO (školní rok 2013/14), která velmi názorně ukazuje, jak užitečné při řešení úloh může být použití různých jednoduchých programátorských postupů, jako jsou vhodný předvýpočet, prefixové součty posloupnosti nebo třeba šikovné seřazení dat. Začneme jako obvykle přesným zadáním úlohy.

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

Dostanete číslo k a posloupnost n čísel. Napište program, který v zadané posloupnosti čísel určí nejdelší souvislou podposloupnost, jejíž aritmetický průměr je přesně roven hodnotě k.

Popis vstupu:
Na prvním řádku vstupu jsou dvě čísla n, k. Na druhém řádku je n kladných celých čísel: prvky posloupnosti. Můžete předpokládat, že vstupní posloupnost čísel vždy obsahuje alespoň jednu souvislou podposloupnost s průměrem prvků přesně rovným k. Můžete také předpokládat, že se součet všech prvků posloupnosti vejde do běžné celočíselné proměnné.

Popis výstupu:
Program vypíše dvě celá čísla – pozici začátku a pozici konce nejdelší souvislé podposloupnosti s aritmetickým průměrem k. Pozice čísel v po-

sloupnosti číslujeme od 1 do n. Pokud existuje více různých vhodných podposloupností téže maximální délky, program vypíše jednu libovolnou z nich.

Hodnocení:

Plných 10 bodů získáte za řešení, které zvládne efektivně vyřešit libovolný vstup délky $n < 200\,000$.

Až 6 bodů dostanete za řešení, které efektivně vyřeší každý vstup délky $n < 5\,000$.

Za jakékoliv funkční řešení bez ohledu na jeho efektivitu můžete získat až 4 body.

Příklady:

vstup
```
7 4
1 1 3 8 1 5 2
```

výstup
```
4 7
```

Existují tři podposloupnosti s průměrem rovným přesně 4, a to $(1, 3, 8)$, $(3, 8, 1)$ a $(8, 1, 5, 2)$. Třetí z nich je nejdelší, takže vypíšeme pozici jejího začátku a konce.

vstup
```
4 5
2 3 5 1
```

výstup
```
3 3
```

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

Ukážeme si několik způsobů řešení, které se budou lišit svou časovou složitostí. Nejsnadnějším řešením, které napadne asi každého, je vyzkoušet postupně každý souvislý úsek posloupnosti a spočítat jeho aritmetický průměr. Ze všech úseků s průměrem rovným přesně k si průběžně pamatujeme pozici toho dosud nejdelšího. V programu použijeme dva cykly určující začátek a konec úseku, ve třetím vnořeném cyklu budeme počítat součet prvků v aktuálním úseku. Spočítaný součet prvků v úseku vždy vydělíme jejich počtem a výsledek porovnáme s hodnotou k. Časová složitost tohoto řešení je $\mathrm{O}(n^3)$, neboť uvažujeme $\mathrm{O}(n^2)$ různých úseků a každý z nich procházíme v čase $\mathrm{O}(n)$.

```
program Prumer1;
var n, k: integer;
    a: array[1..200000] of integer;
    zac, kon, soucet: integer;      {zkoumaný úsek}
    zacv, konv: integer;            {výsledný úsek}
    i: integer;

begin
read(n, k);
for i:=1 to n do read(a[i]);
zacv:=1; konv:=0;
for zac:= 1 to n do
  for kon:=zac to n do
    begin
    soucet:=0;
    for i:=zac to kon do
      soucet:=soucet + a[i];
    if (soucet = (kon-zac+1)*k) and (konv-zacv < kon-zac) then
      begin zacv:=zac; konv:=kon end
    end;
writeln(zacv, konv)
end.
```

Program není moc zajímavý, ale všimněte si, že jsme v něm použili jeden šikovný technický trik. Chtěli jsme se vyhnout operaci dělení, která se provádí v pomalejší reálné aritmetice a navíc může být nepřesná. Označíme-li součet prvků ve zkoumaném úseku *soucet* a jejich počet p, pak místo testování, zda $soucet/p = k$ v programu raději testujeme, zda $soucet = p * k$. Výsledek je matematicky ekvivalentní, výpočet se ale provede v celočíselné aritmetice, tedy rychleji a zaručeně přesně. Podobný obrat můžete využívat i v jiných vašich programech.

Za výše uvedené řešení s asymptotickou časovou složitostí $O(n^3)$ mohli soutěžící získat nejvýše 4 body z celkových 10 možných. Zkusíme proto výpočet programu zrychlit. Často pomůže zamyslet se, zda se některé operace neprovádějí při výpočtu zbytečně vícekrát. V právě popsaném řešení spočívá neefektivita výpočtu zejména v tom, že součet prvků v každém zkoumaném úseku počítáme zvlášť, takže opakovaně sčítáme stejná čísla. Namísto toho můžeme součet prvků ve zkoumaném úseku počítat průběžně při každém zvýšení proměnné *kon*. Zvýšením proměnné *kon* o 1 se předchozí zkoumaný úsek prodlouží o jeden prvek posloupnosti, takže součet nového úseku získáme přičtením tohoto prvku $a[kon]$ k součtu předchozího úseku. Dostáváme tak řešení, které opět projde všech $O(n^2)$ souvislých úseků zadané posloupnosti, ale každý z nich zpracuje v konstantním čase.

Časová složitost řešení se tak sníží na $O(n^2)$, což v soutěži stačilo na zisk až 6 bodů.

```
program Prumer2;
var n, k: integer;
    a: array[1..200000] of integer;
    zac, kon, soucet: integer;      {zkoumaný úsek}
    zacv, konv: integer;            {výsledný úsek}
    i: integer;

begin
read(n, k);
for i:=1 to n do read(a[i]);
zacv:=1; konv:=0;
for zac:= 1 to n do
  begin
  soucet:=0;
  for kon:=zac to n do
    begin
    soucet:=soucet + a[kon];
    if (soucet = (kon-zac+1)*k) and (konv-zacv < kon-zac) then
      begin zacv:=zac; konv:=kon end
    end
  end;
writeln(zacv, konv)
end.
```

Řešením s kvadratickou asymptotickou časovou složitostí naše úsilí ještě nekončí. Pro další zrychlení výpočtu ale budeme potřebovat přetransformovat náš problém do trochu jednodušší podoby. Je totiž obtížné ve zkoumaných úsecích posloupnosti sledovat a porovnávat zároveň dva parametry, na nichž závisí aritmetický průměr – totiž délku úseku a součet jeho prvků. Hodilo by se nám sledovat jenom jeden parametr.

Má-li mít výsledný úsek původní posloupnosti aritmetický průměr k, nabízí se možnost upravit posloupnost tak, že snížíme hodnotu všech jejích prvků právě o k. Stejný úsek v takto upravené posloupnosti pak bude mít aritmetický průměr 0. Platí to samozřejmě i naopak: úsek s průměrem 0 v upravené posloupnosti má v původní posloupnosti aritmetický průměr k. Je tomu tak proto, že když od každého prvku posloupnosti odečteme k, snížíme tím aritmetický průměr libovolného úseku přesně o k.

Nyní provedeme v naší úvaze druhý krok. Jakýkoliv úsek posloupnosti má aritmetický průměr rovný 0 právě tehdy, když má součet 0. Při hledání úseků s nulovým součtem přitom už nemusíme hledět na jejich délku. Náš původní problém jsme tedy převedli na jinou, jednodušší úlohu: v upra-

vené posloupnosti nalézt nejdelší souvislý úsek se součtem 0. Pozice tohoto úseku pak přímo určuje pozici nejdelšího souvislého úseku s aritmetickým průměrem k v původní posloupnosti. Zadanou posloupnost čísel si tedy nejprve upravíme tak, že od každého prvku odečteme k. To je jednoduchý *předvýpočet* s lineární časovou složitostí $O(n)$. Od této chvíle budeme slovem „posloupnost“ označovat tuto novou posloupnost, ve které hledáme úseky se součtem 0.

Kdybychom posloupnost jednoduše procházeli a počítali v ní součty všech možných souvislých úseků, dostali bychom se opět k řešení s časovou složitostí $O(n^3)$ nebo s trochou šikovnosti $O(n^2)$. Úseky ale můžeme sčítat i jiným způsobem, využijeme tzv. *prefixové součty*. To je další z technických obratů, který se při práci s číselnými posloupnostmi často využívá. Vytvoříme si pomocné pole $P[0..n]$ a naplníme ho takovými hodnotami, aby hodnota $P[i]$ byla rovna součtu prvních i prvků naší posloupnosti. Prvkům takového pole P říkáme prefixové součty dané posloupnosti. Hodnoty pole P snadno spočítáme jedním průchodem v čase $O(n)$. Položíme $P[0]=0$, neboť úsek nulové délky má i nulový součet. Každé další $P[i]$ určíme jako součet již známé hodnoty $P[i-1]$ a i-tého prvku posloupnosti. Když potom potřebujeme zjistit součet všech čísel v souvislém úseku posloupnosti od pozice *zac* do pozice *kon* (včetně prvků z obou těchto krajních pozic), spočítáme ho v konstantním čase jako rozdíl $P[kon] - P[zac-1]$.

Vraťme se nyní k naší úloze. Už víme, že pomocí dalšího předvýpočtu s lineární časovou složitostí získáme pole P prefixových součtů naší posloupnosti. V posloupnosti hledáme souvislé úseky s nulovým součtem, tzn. úseky, pro které platí $P[kon]-P[zac-1] = 0$ neboli $P[kon] = P[zac-1]$. Jinými slovy řečeno, zajímají nás dvojice stejných hodnot v poli P. Chceme určit v poli P takovou dvojici stejných čísel, aby tato čísla byla od sebe co nejvíce vzdálena. Tím bude určen nejdelší úsek posloupnosti s nulovým součtem.

Nejjednodušší cestou řešení je využít obyčejné třídění. To je v programování opět jedna z často užívaných metod – při práci s daty se mnohdy vyplatí seřadit si je jinak. Místo pole s hodnotami prefixových součtů P[0], P[1], P[2], ..., P[n] vytvoříme pole uspořádaných dvojic (P[0],0), (P[1],1), (P[2],2), ..., (P[n], n). S každým prefixovým součtem si tedy uložíme navíc informaci, o kolikátý prefixový součet se jedná. Toto pole nyní vzestupně uspořádáme primárně podle první souřadnice, tedy podle odpovídajícího prefixového součtu, a sekundárně podle indexu, který mu odpovídá. V takto uspořádaném poli budou všechny indexy, kterým odpovídá

stejná hodnota prefixového součtu, tvořit vždy souvislý rostoucí úsek. Nejvzdálenější dvojici indexů, kterým odpovídá stejný prefixový součet, pak dokážeme snadno určit při jednom průchodu polem, tzn. s lineárním časovou složitostí $O(n)$.

Celé řešení zadané úlohy se tedy skládá ze čtyř postupně prováděných fází:

– úprava zadané posloupnosti snížením všech prvků o k,
– výpočet prefixových součtů $P[i]$ takto upravené posloupnosti,
– seřazení pole dvojic $(P[i], i)$ výše popsaným způsobem,
– nalezení výsledku průchodem seřazeného pole.

První, druhá a čtvrtá fáze výpočtu mají asymptotickou časovou složitost $O(n)$. Časově nejnáročnější je setřídění pole ve třetí fázi, které provedeme některým ze standardních třídicích algoritmů v čase $O(n \log n)$. Výsledná časová složitost řešení úlohy je proto $O(n \log n)$. Za řešení s touto složitostí obdrželi soutěžící plný počet bodů.

```
program Prumer3;
var n, k: integer;
    a: array[0..200000] of
         record p: integer;          {prefixový součet}
                x: integer;          {index}
         end;
    zac, kon: integer;               {zkoumaný úsek}
    zacv, konv: integer;             {výsledný úsek}
    c, i: integer;

begin
{Načteme posloupnost, zároveň hned snižujeme hodnoty o "k"
a počítáme prefixové součty posloupnosti:}
read(n, k);
a[0].p:=0; a[0].x:=0;                {úsek nulové délky}
for i:=1 to n do
  begin
  read(c);                           {i-tý prvek posloupnosti}
  a[i].p:=a[i-1].p + c - k;          {i-tý prefixový součet}
  a[i].x:=i                          {jeho index je "i"}
  end;

{Seřazení pole "a" vzestupně,
primárně podle "p" a sekundárně podle "x":}
Sort(a);

{Určení výsledku:}
zacv:=1; konv:=0;
```

```
zac:=0; kon:=0;
while kon < n do
  begin
  while (kon < n) and (a[kon+1].p = a[kon].p) do kon:=kon+1;
  if a[kon].x-a[zac].x > konv-zacv then
    begin zacv:=a[zac].x; konv:=a[kon].x end;
  zac:=kon+1; kon:=kon+1
  end;
writeln(zacv+1, konv)
end.
```

V programu jsme použili drobný technický trik na úsporu paměti. Všimněte si, že po spočítání prefixových součtů upravené vstupní posloupnosti už nikdy nebudeme potřebovat původní posloupnost čísel. Nemusíme si ji proto ani ukládat. Postupně načítané prvky posloupnosti ihned snižujeme o k, průběžně z nich rovnou počítáme prefixové součty a až tyto hodnoty prefixových součtů si ukládáme do pole. Navíc si do tohoto pole ukládáme hned dvojice typu $(\mathrm{P}[i], i)$, které budeme následně třídit. V programu tak vystačíme jen s jediným polem velikosti n.

Ve výše uvedené programové ukázce jsme pro jednoduchost a pro zkrácení zápisu vynechali deklaraci třídicí procedury *Sort*. Zde by se použil některý standardní třídicí algoritmus s asymptotickou časovou složitostí $\mathrm{O}(n \log n)$, například třídění haldou nebo třídění sléváním.

Na závěr poznamenejme, že nalézt v poli P dvojici stejných hodnot s maximální vzájemnou vzdáleností můžeme i jinými způsoby. Pokud znáte některé pokročilejší datové struktury, jako jsou třeba vyvažované binární vyhledávací stromy, můžete je zde výhodně využít. Pole prefixových součtů P budeme postupně procházet a do binárního vyhledávacího stromu si budeme ukládat všechny navzájem různé hodnoty dosažených prefixových součtů. Ke každé z nich si zároveň uložíme index jejího prvního výskytu. Když tedy v i-tém kroku zpracováváme hodnotu $\mathrm{P}[i]$, zkusíme ji nejprve vyhledat ve stromě. Pokud tam ještě není, přidáme do stromu dvojici údajů $(\mathrm{P}[i], i)$. Pokud tam už je, pak rozdíl u ní uloženého indexu a aktuálního indexu určuje délku dalšího nalezeného úseku s nulovým součtem. Průběžně si zaznamenáváme maximum z takto nalezených délek, jemu příslušné indexy budou výsledkem řešení úlohy.

Časová složitost této varianty řešení je rovněž $\mathrm{O}(n \log n)$. Výpočet se provádí v n krocích a vyhledání resp. přidání každé hodnoty ve vyváženém binárním vyhledávacím stromu s n uzly má časovou složitost $\mathrm{O}(\log n)$.