INFORMATIKA

Vybrané podposloupnosti
(Úlohy z MO – kategorie P, 30. část)

PAVEL TÖPFER

Matematicko-fyzikální fakulta UK, Praha

V našem seriálu putujícím po zajímavých úlohách Matematické olympiády - kategorie P se tentokrát zastavíme v krajském kole aktuálního 62. ročníku (školní rok 2012/2013). Objevila se zde jedna poměrně snadná úloha věnovaná problematice hledání vybraných podposloupností. Autorem úlohy i původního autorského řešení je doc. Zdeněk Dvořák z Matematicko-fyzikální fakulty UK v Praze. Primitivní řešení této úlohy vymyslí jistě každý z vás, je ale zajímavé zamyslet se nad možnostmi zlepšit jeho časovou složitost. Nejprve si jako obvykle ukážeme úplné zadání úlohy, které si pro potřeby našeho článku jen velmi mírně upravíme, aniž bychom tím ovšem změnili smysl původní úlohy.

⋆ ⋆ ⋆

Král Filip vyráží na cestu po svém království. S ohledem na státnické povinnosti si už vybral, která města a v jakém pořadí navštíví. Výpravu ale komplikuje skutečnost, že král důsledně dodržuje dietu předepsanou královskými lékaři a astrology. Ti mu doporučili, kolik dní má cesta trvat, a předložili mu několik možných plánů, co má jíst v jednotlivých dnech cesty. V každém městě chce král jíst pouze místní specialitu. Ta je v každém městě právě jedna, nicméně stejná specialita může být k dostání ve více městech. Král se ovšem nemusí zastavit na jídlo v každém z navštívených měst. Pomozte mu určit, které z navržených dietních plánů lze realizovat.

Doprava mezi městy je velmi rychlá, není tedy žádné omezení na vzdá-

lenost mezi městy, v nichž se král Filip zastaví na jídlo. Města ale musí navštívit v předem určeném pořadí, nemůže se nikdy vracet ani dvakrát po sobě jíst ve stejném městě.

Popis vstupu

První řádek vstupu obsahuje přirozená čísla N, M, K a L, kde N udává počet navštívených měst, M je celkový počet druhů nabízených jídel, K je počet dietních plánů předložených k posouzení a L je jejich délka (počet jídel v každém z nich). Druhy jídel jsou očíslovány přirozenými čísly od 1 do M. Druhý řádek vstupu obsahuje N přirozených čísel udávajících, jakou specialitu nabízejí v jednotlivých městech v tom pořadí, v němž je král během své cesty navštíví. Na i-tém z K následujících řádků je L přirozených čísel popisujících i-tý dietní plán; j-té z těchto čísel určuje jídlo, které Filip musí jíst při své j-té zastávce na cestě.

Popis výstupu

Výstup se skládá z K řádků. Na i-tý z nich vypište slovo `ano`, lze-li během cesty realizovat i-tý dietní plán, v opačném případě vypište slovo `ne`. Dietní plán lze realizovat, jestliže vznikne z posloupnosti specialit nabízených v navštívených městech odstraněním některých čísel.

Příklad

Vstup:

```
10 4 4 3
1 2 1 3 4 1 2 1 3 4
1 4 2
4 3 2
1 1 1
4 4 4
```

Výstup:

```
ano
ne
ano
ne
```

Hodnocení

Plných 10 bodů získáte za řešení, které úlohu efektivně vyřeší pro N i $K.L$ řádově statisíce. Až 7 bodů získáte za řešení, které bude navíc předpokládat $M \leq 10$. Až 4 body získáte za řešení, které úlohu efektivně vyřeší za předpokladu, že $K.(N + L) \leq 100\,000$.

⋆ ⋆ ⋆

Nejdříve se budeme zabývat situací, že zkoumáme pouze jeden dietní plán a chceme zjistit, zda ho lze zrealizovat. Máme tedy zadánu posloupnost specialit nabízených v jednotlivých městech královy cesty $s = (s_1, s_2, \ldots, s_N)$ a posloupnost jídel tvořících zkoumaný dietní plán $d = (d_1, d_2, \ldots, d_L)$. Naším úkolem je zjistit, zda d je *vybranou posloupností* z posloupnosti s, tzn. zda lze posloupnost d získat vynecháním některých vhodných prvků z posloupnosti s při zachování vzájemného pořadí zbývajících prvků posloupnosti.

Uvedený problém vyřešíme snadno a přímočaře algoritmem s časovou složitostí $O(N+L)$. Budeme brát postupně jednotlivé prvky posloupnosti d a budeme je vyhledávat v posloupnosti s vždy od místa, kde jsme ukončili předchozí hledání. Začneme s číslem d_1. Pokud se vůbec nevyskytuje v posloupnosti s, pak dietní plán d zjevně nelze realizovat. V opačném případě se může král zastavit na jídlo d_1 v prvním městě, kde ho nabízejí, a nic tím nezkazí. Na jídlo d_2 se pak může král zastavit až v nějakém následujícím městě. Opět nic nezkazí, když to bude hned nejbližší takové město. Stejně postupujeme i pro další prvky posloupnosti d. Jestliže takto dojdeme až na konec dietního plánu d a všechny jeho prvky se nám podaří v posloupnosti s vyhledat, pak lze tento dietní plán realizovat. Pokud tímto způsobem některý prvek posloupnosti d nenajdeme v s, potom plán realizovat nelze.

Popsané řešení snadno naprogramujeme:

```
function Nejblizsi(odkud: integer; hodnota: integer):
integer;
{hledá nejbližší pozici výskytu "hodnota" v N-prvkovém
poli "s", která následuje za pozicí "odkud"; když
nenajde, vrací N+1}
var i: integer;
begin
  i:=odkud+1;
```

```
  while (i <= N) and (s[i] <> hodnota) do inc(i);
  Nejblizsi:=i
end;

function JedenPlan: boolean;
{řeší úlohu pro jeden dietní plán}
var kde: integer;  {vybraná pozice v "s"}
    i: integer;
begin
  kde:=0;
  for i:=1 to L do
    begin
    kde:=Nejblizsi(kde, d[i]);
    if kde = N+1 then
      begin JedenPlan:=false; exit end
  end;
  JedenPlan:=true
end;
```

Od popsaného řešení pro jeden dietní plán zbývá udělat už jenom malý krůček k vyřešení celé úlohy – stačí postupně aplikovat uvedený postup na všechny zadané dietní plány. Funkce *JedenPlan* tedy bude volána z hlavního programu nikoliv jednou, ale v cyklu K-krát, kde K označuje počet zkoumaných dietních plánů. Dostáváme tak primitivní řešení úlohy s časovou složitostí $O(K.(N + L))$.

Řešení tohoto typu nevyžaduje žádné hlubší znalosti ani velké přemýšlení. V krajském kole 62. ročníku MO kategorie P byla řešení navržená tímto způsobem nejčastější ze všech. Podle podmínek uvedených v zadání úlohy jste za řešení s uvedenou časovou složitostí mohli v soutěži dostat nejvýše 4 body z celkových 10.

Při zkoumání každého dietního plánu zcela samostatně a odděleně od ostatních žádné efektivnější řešení zřejmě nevymyslíte. Časovou složitost algoritmu ale dokážeme snížit, když budeme mít hned od začátku na paměti, že potřebujeme prozkoumat velké množství dietních plánů. Vyplatí se nám proto vykonat předem (a tedy jenom jednou) tu práci, která na konkrétním dietním plánu nezávisí a kterou pak budeme moci pro jednotlivé dietní plány opakovaně využívat.

V našem případě si můžeme ušetřit práci spojenou s opakovaným procházením posloupnosti s a vyhledáváním různých hodnot v ní, tzn. mů-

žeme optimalizovat činnost, kterou vykonává pomocná funkce *Nejblizsi*. V různých dietních plánech se totiž opakovaně vyskytují stále stejná jídla a my tudíž během řešení úlohy v posloupnosti s vyhledáváme mnohokrát opakovaně tytéž hodnoty. Nabízí se proto provést vhodný předvýpočet, příslušné pozice nejbližších následujících výskytů různých hodnot jídel si spočítat dopředu a uložit si je v pomocném poli. Můžeme použít například dvojrozměrné pole P, které bude obsahovat přesně tytéž údaje, jaké výstupní hodnoty vrací funkce *Nejblizsi*. Při zkoumání každého dietního plánu pak stačí namísto opakovaného volání funkce *Nejblizsi* vždy jen vyzvednout příslušnou hodnotu z pole P v konstantním čase. Tím snížíme časovou složitost zpracování jednoho dietního plánu na $O(L)$ a řešení celé úlohy s K dietními plány na $O(K.L)$.

K tomu je třeba přičíst ještě čas potřebný na výpočet údajů uložených v poli P – ten se ovšem provádí pouze jednou a zvládneme ho vykonat v čase $O(N.M)$. Uvažované dvojrozměrné pole má totiž velikost $(N+1).M$ – je tvořeno $N+1$ řádky odpovídajícími indexům *odkud* (od 0 do N) a M sloupci odpovídajícími číslům jídel *hodnota* (od 1 do M). Každý prvek pole přitom spočítáme v konstantním čase, když budeme obsah pole vyplňovat po sloupcích, a to vždy zdola nahoru. Jestliže totiž $s_{odkud+1} = hodnota$, pak $P[odkud, hodnota] = odkud + 1$, jinak $P[odkud, hodnota] = P[odkud + 1, hodnota]$.

```
procedure VytvorPole;
{vytváří pole P na základě údajů posloupnosti "s"}
var i, j: integer;
begin
  for j:=1 to M do
    begin
    P[N,j]:=N+1;
    for i:=N-1 downto 0 do
      if s[i+1]=j then P[i,j]:=i+1
                  else P[i,j]:=P[i+1,j]
    end
end;
```

Celková časová složitost tohoto řešení s polem je $O(N.M+K.L)$. Oproti původnímu jednoduchému řešení bude rychlejší za předpokladu, že počet nabízených jídel M je výrazně menší, než počet dietních plánů K. Pokud bude celkový počet jídel M naopak velký, pak nám řešení s polem P moc

nepomůže a jeho časová náročnost může být dokonce vyšší, než délka výpočtu úvodního primitivního řešení. Postup výpočtu využívající pole P každopádně představuje použití určité netriviální úvahy s předvýpočtem a v soutěži jste za něj proto mohli získat až 7 bodů z 10. Je zajímavé, že tento způsob řešení úlohy se mezi řešiteli olympiády téměř neobjevil – většina soutěžících skončila u primitivního řešení uvedeného na začátku článku, ti lepší se naopak dostali ještě dále, až k jednomu z efektivnějších postupů, které budou následovat.

Zkusme nyní urychlit naše výchozí řešení jiným způsobem. Při hledání každého jídla ze zkoumaného dietního plánu d jsme v něm museli postupně procházet posloupnost s a určovat v ní první další pozici města takovou, kde se toto jídlo nabízí. Abychom toto hledání urychlili, aniž bychom si museli vytvářet rozsáhlé pole P, můžeme si předem vytvořit M seznamů pozic měst, kde vaří jednotlivé speciality. Pro všechna j od 1 do M si označme $sezn[j]$ takový seznam odpovídající jídlu j. Tyto seznamy snadno získáme jedním průchodem posloupností s, kdy pro každý prvek si přidáme do příslušného seznamu $sezn[s_i]$ jeho pozici i. Časová složitost této fáze výpočtu bude $O(N)$ a všechny seznamy pozic navíc získáme rovnou vzestupně uspořádané.

```
procedure VytvorSeznamy;
{vytváří seznamy pozic jednotlivých hodnot v posloupno-
sti "s"}
var i, j, hodnota: integer;
begin
  for j:=1 to M do delka[j]:=0;
  for i:=1 to N do
    begin
    hodnota:=s[i];
    delka[hodnota]:=delka[hodnota] + 1;
    sezn[hodnota, delka[hodnota]]:=i
    end
end;
```

Pomocí vytvořených seznamů pozic dokážeme významně urychlit výpočet funkce *Nejblizsi* z úvodního řešení. Údaj *Nejblizsi*(*odkud, hodnota*) je roven nejmenšímu prvku seznamu $sezn[hodnota]$ většímu než *odkud*. Pokud v seznamu $sezn[hodnota]$ neexistuje prvek větší než odkud, pak $Nejblizsi(odkud, hodnota) = N + 1$. Hledaný údaj můžeme v přísluš-

ném seznamu nalézt půlením intervalů v logaritmickém čase vzhledem k délce seznamu, přitom délka všech seznamů je shora omezena délkou posloupnosti s. Každé volání funkce $Nejblizsi$ tedy můžeme provést v čase $O(\log N)$. Celkové časová složitost tohoto řešení je proto $O(N + K.L. \log N)$. Upravenou funkci $Nejblizsi$ si zapíšeme pouze schematicky:

```
function Nejblizsi(odkud: integer; hodnota: integer):
integer;
{s využitím vytvořených seznamů hledá nejbližší pozici
výskytu "hodnota" v posloupnosti "s", která následuje
za pozicí "odkud"; když nenajde, vrací N+1}
var i: integer;
begin
  if sezn[hodnota, delka[hodnota]] <= odkud then
    Nejblizsi:=N+1
  else
    Nejblizsi := nejmenší prvek ze vzestupně uspořádaného
    seznamu sezn[hodnota,1], sezn[hodnota,2], ...,
    sezn[hodnota,delka[hodnota]] který je větší než
    "odkud" - vyhledat půlením intervalů
end;
```

Programem s touto časovou složitostí již zvládneme efektivně vyřešit každá vstupní data až do velikosti uvedené v zadání úlohy. Řešení tohoto typu se v soutěži objevila vícekrát a byla hodnocena až plným počtem 10 bodů.

Na závěr si ukážeme ještě jiný způsob řešení úlohy, který je založen na souběžném zpracování všech K zkoumaných dietních plánů. Budeme postupně procházet městy tvořícími posloupnost s a pro i-té město vždy určíme, jaký nejdelší počáteční úsek z každého z dietních plánů může mít král již realizován po projití cesty $s_1, s_2, \ldots, s_i$. Délku tohoto úseku v j-tém dietním plánu označíme jako $r_{j,i}$. Ukážeme si, jak budeme hodnoty $r_{j,i}$ počítat. Při příchodu do i-tého města prodloužíme realizované počáteční úseky těch dietních plánů, v nichž za dosud realizovaným počátečním úsekem následuje právě jídlo si. Pro takové dietní plány j tedy platí $r_{j,i} = r_{j,i-1} + 1$, zatímco pro ty zbývající je $r_{j,i} = r_{j,i-1}$. Zajímá nás, ve kterých dietních plánech bude platit po projití celé posloupnosti s, že $r_{j,N} = L$. Právě tyto dietní plány je možné realizovat.

Při implementaci popsaného algoritmu nebudeme ani potřebovat dvojrozměrné pole r. Hodnota $r_{j,i}$ totiž závisí pouze na bezprostředně předcházející hodnotě $r_{j,i-1}$, nikoliv na hodnotách starších. K uchování potřebných informací nám proto postačí jednorozměrné pole $r[j]$, v němž budeme hodnoty $r_{j,i}$ postupně přepisovat (tzn. zvětšovat je o 1, když bude třeba).

Přímočará implementace právě uvedeného postupu vede k řešení s časovou složitostí $O(N.K)$, jelikož provádíme N kroků výpočtu a v každém z nich testujeme všech K dietních plánů. Dostali bychom se tak k programu s obdobnou časovou složitostí, jakou mělo již naše výchozí primitivní řešení úlohy. Nabízí se ale lepší možnost. Do programu doplníme pro každé z nabízených jídel seznam čísel těch dietních plánů, v nichž je uvažované jídlo „na řadě“, tzn. v nichž uvažované jídlo bezprostředně následuje za již realizovaným počátečním úsekem. V i-tém kroku výpočtu pak stačí projít seznam odpovídající jídlu s_i a ten již přímo určuje čísla dietních plánů j, u nichž je třeba zvýšit hodnotu $r[j]$ o 1. Tato čísla dietních plánů je ovšem třeba zároveň přesunout do patřičného seznamu podle toho, jaké jídlo se v nich dostalo na řadu po prodloužení realizovaného úseku.

Takto upravené řešení vykoná v i-tém kroku tolik operací, jaká je aktuální délka seznamu pro s_i, neboli kolikrát se v tomto kroku zvýší některá z hodnot v poli r. Pole r má K položek a každá z nich se může zvýšit nejvýše L-krát, takže celkově se během výpočtu vykoná nejvýše $O(K.L)$ těchto operací. Do výsledné časové složitosti je třeba započítat ještě čas $O(N)$ potřebný na průchod posloupností s a čas $O(M)$ potřebný na inicializaci všech M seznamů pro jednotlivá jídla. Výsledná časová složitost je proto $O(K.L + N + M)$.

Řešení s touto časovou složitostí pracuje dostatečně efektivně pro všechna vstupní data podle zadání úlohy a v soutěži bylo hodnoceno plným počtem 10 bodů. Mezi řešeními odevzdanými v soutěži byla i řešení založená na této myšlence, byla ale dosti ojedinělá. Programovou realizaci tohoto posledního algoritmu zde již neuvádíme, podle uvedeného popisu ji jistě zvládne každý sám, nebo ji můžete nalézt na webových stránkách MO kategorie P na adrese http://mo.mff.cuni.cz/p/62/reseni-2.html