[dokumenty-10] 40 let matematické olympiády (v Československu)

Pavel Töpfer
Kategorie P

In: Karel Horák (editor): [dokumenty-10] 40 let matematické olympiády (v Československu). (Czech). Praha: Jednota českých matematiků a fyziků, 1993. pp. 18–22.

Persistent URL: http://dml.cz/dmlcz/405379

Kategorie P

Pavel Töpfer
(MFF UK Praha)

Od 35. ročníku byla matematická olympiáda obohacena o novou soutěžní kategorii, která byla nazvána kategorie P (programování). Kategorie P byla vytvořena v době, kdy v celé naší společnosti a zejména mezi mládeží výrazně sílil zájem o počítače a programování a kdy vznikala i řada jiných programátorských soutěží. Nová kategorie MO si kladla za cíl stát se vrcholnou soutěží pro talentované studenty středních škol, kteří se zajímají právě o matematiku a programování. Programování přitom chtěla ukázat ne ze stránky technického zvládnutí práce s počítačem a používání programovacích jazyků, jak je to obvyklé u jiných soutěží, ale zaměřit se na samotnou podstatu věci. Její náplní se proto staly úlohy na analýzu a tvorbu algoritmů, úlohy, pro jejichž úspěšné vyřešení nestačí pouze běžné praktické programátorské dovednosti, ale které navíc vyžadují od řešitele kus matematického a algoritmického myšlení. Ohlas, který kategorie P rychle získala, a stále rostoucí zájem o účast v soutěži svědčí o tom, že se tyto cíle úspěšně podařilo splnit.

V současné době řeší úlohy MO kategorie P každoročně téměř 500 studentů všech ročníků středních škol z celé republiky. Zájem o soutěž začínají projevovat i nejlepší žáci základních škol, kteří někdy dosahují překvapivě dobrých výsledků, a to i v celostátním kole. Po odborné stránce je kategorie P zajišťována vysokoškolskými pedagogy z kateder informatiky. Vedle tradičních tří odborných center působících od samého vzniku soutěže na matematicko-fyzikální fakultě Univerzity Karlovy v Praze, přírodovědecké fakultě Masarykovy Univerzity v Brně a na matematicko-fyzikální fakultě Univerzity Komenského v Bratislavě se v současné době úspěšně jedná i o zapojení dalších vysokých škol.

Z celé řady zajímavých soutěžních úloh, které se v historii kategorie P objevily, jsme zde pro vás vybrali alespoň tři. Společným rysem všech tří úloh je to, že jsou zdánlivě velmi snadné. Skutečně, nalézt nějaký algoritmus řešící daný problém vám asi nedá mnoho práce. Zde je však úkolem sestrojit algoritmus co možná nejlepší a nejrychlejší, a to již vyžaduje značně delší a hlubší přemýšlení.

Úlohy

1. Největší čísla (MO–P 39–II–1)

Je dáno pole $A[1..n, 1..n]$ obsahující n^2 navzájem různých kladných celých čísel. Navrhněte co nejrychlejší algoritmus, který vytiskne n největších čísel uložených v poli

A. Původní obsah pole A nemusí být po ukončení výpočtu zachován.

2. Jedničkový obdélník (MO–P 38–II–2)

Je dáno dvojrozměrné pole A (matice) velikosti $n \times m$, jehož prvky jsou pouze čísla 0 nebo 1. Navrhněte algoritmus, který v daném poli A nalezne maximální „obdélník" obsahující samé jedničky (maximální ve smyslu „obsahující co nejvíc jedniček"). Výsledkem práce algoritmu bude čtveřice čísel i, j, k, l takových, že $A_{i,j}$ je prvek v levém horním rohu a $A_{k,l}$ prvek v pravém dolním rohu nalezeného maximálního obdélníka.

3. Nejdelší rostoucí podposloupnost (MO–P 39–II–2)

Je dána konečná posloupnost celých čísel délky n, $n \geq 1$. Prvky této posloupnosti označíme po řadě $X(1), X(2), \ldots, X(n)$. Podposloupností délky k vybranou ze zadané posloupnosti budeme rozumět libovolnou konečnou posloupnost tvaru $X(i_1), X(i_2), \ldots, X(i_k)$, kde $1 \leq i_1 < i_2 < \ldots < i_k \leq n$ (tzn., že ze zadané posloupnosti je vybráno libovolných k čísel, přičemž je zachováno jejich pořadí).

Navrhněte algoritmus, který určí délku nejdelší rostoucí podposloupnosti vybrané z dané posloupnosti. To znamená, že určí maximální k takové, že $X(i_1) < X(i_2) < \ldots < X(i_k)$ pro nějaké indexy $1 \leq i_1 < i_2 < \ldots < i_k \leq n$. Zdůvodněte správnost algoritmu.

Např. pro posloupnost 4, 2, 7, 6, 4, 5, 3, 9, 8, 5, 9 je $k = 5$, neboť maximální vybraná rostoucí podposloupnost 2, 4, 5, 8, 9 má délku 5.

Řešení

1. Algoritmus s optimální kvadratickou časovou složitostí je založen na často užívaném postupu: nejprve se provede vhodný předvýpočet, jeho výsledky se uloží do pomocného pole a poté teprve následuje vlastní výpočet výsledných hodnot s využitím předem připraveného pomocného pole.

Zavedeme pomocné pole $SlMax[1..n]$, které bude obsahovat informace o poloze maximálních hodnot v jednotlivých sloupcích pole A. Bude tedy platit $SlMax[j] = i$ právě tehdy, jestliže $A[i, j]$ je největší ze všech čísel uložených v j-tém sloupci pole A.

Nejprve provedeme počáteční zaplnění pole $SlMax$ odpovídajícími hodnotami. Výběr n největších čísel uložených v poli A potom proběhne v n krocích následujícího postupu:

– pomocí pole $SlMax$ nalezneme největší hodnotu ze sloupcových maxim; tuto hodnotu získáme jako maximum z čísel $A[SlMax[j], j]$ pro j od 1 do n; nechť je to číslo $A[i, k]$

– číslo $A[i, k]$ je tedy největším z čísel uložených v poli A; vytiskneme ho a vypustíme ho z pole A dosazením nuly do $A[i, k]$

– obnovíme informaci o poloze sloupcového maxima ve sloupci, v němž došlo ke změně, tzn. spočteme novou hodnotu $SlMax[k]$.

Správnost algoritmu přímo plyne z úvodního rozboru. V každém kroku výpočtu je nalezena a vytištěna největší hodnota z maxim v jednotlivých sloupcích, což je jistě největší číslo momentálně se nacházející v poli A. Přepsáním tohoto čísla nulou je

vytisknuté číslo z pole A vynecháno (všechna čísla v poli A jsou podle zadání kladná!) a v dalším kroku se tedy bude vyhledávat největší ze všech zbývajících čísel. Celkem po n krocích se vytiskne skutečně n největších čísel uložených původně v poli A. Výpočet je konečný, počet kroků výpočtu je předem určen hodnotou n.

Popsaný algoritmus má kvadratickou časovou složitost. Přečtení n^2 čísel ze vstupu i počáteční zaplnění pole *SlMax* jistě vyžadují řádově n^2 operací. Vlastní výpočet je pak tvořen n kroky, přičemž v každém z nich je nejprve pomocí pole *SlMax* vybráno maximum z n čísel a po jeho vypsání a smazání je opět výběrem maxima z n čísel obnoveno správné zaplnění pole *SlMax*. Počet provedených operací je tedy úměrný hodnotě n^2.

2. V první fázi řešení provedeme pomocný výpočet, při kterém určíme délky souvislých sloupců jedniček v dané matici A. Výsledky tohoto výpočtu si uložíme přímo do pole A tak, že položíme $A_{i,j} = k$, jestliže prvek $A_{i,j}$ sám a dalších přesně $k-1$ prvků pod ním mělo původně hodnotu 1, tzn. jestliže v původní matici A platilo $A_{p,j} = 1$ pro $p = i, i+1, \ldots, i+k-1$ a navíc buď $i+k-1 = n$ nebo $i+k-1 < n$ a přitom $A_{i+k,j} = 0$ (kde n je počet řádků matice A). Údaje v zadaném poli A tím pozměníme, ale pouze tak, že v případě potřeby by bylo snadné zrekonstruovat původní podobu pole A (neboť žádná nula v poli A neubyla ani nepřibyla, nenulová čísla jsou uložena na místech původních jedniček). Výsledek první pomocné fáze výpočtu si ukážeme na příkladu:

ze zadané matice:	dostaneme upravenou matici:
1 1 0 1 0	3 4 0 1 0
1 1 1 0 1	2 3 2 0 3
1 1 1 1 1	1 2 1 1 2
0 1 0 0 1	0 1 0 0 1

Ve druhé fázi výpočtu již budeme hledat v poli A maximální obdélník tvořený jedničkami (nyní po úpravě nenulovými čísly). Postupně budeme zkoumat všechny možné pozice levého horního rohu takového obdélníka. Pro zvolený levý horní roh $A_{i,j} > 0$ musíme vyzkoušet všechny přípustné polohy pravého horního rohu $A_{i,l}$. Prvek $A_{i,l}$ může být pravým horním rohem obdélníka s levým horním rohem $A_{i,j}$, jestliže všechna čísla $A_{i,q}$ pro $q = j, j+1, \ldots, l$ jsou nenulová.

Velikost maximálního obdélníka, který je v původní matici A tvořen samými jedničkami a jehož levý a pravý horní roh mají souřadnice $[i,j]$, resp. $[i,l]$, nyní již snadno určíme pomocí hodnot, které jsme si předem připravili v první fázi výpočtu. Takový obdélník má totiž šířku $(l-j+1)$ a jeho výška je rovna minimu z hodnot $A_{i,q}$ pro $q = j, j+1, \ldots, l-1, l$.

Uvedený výpočet je možno opakovat pro všechny možné volby levého horního rohu obdélníka a přitom si v pomocné proměnné udržovat velikost maximálního již nalezeného obdélníka tvořeného v zadané matici samými jedničkami. V dalších čtyřech pomocných proměnných si musíme zaznamenávat souřadnice levého horního a pravého dolního rohu nalezeného maximálního obdélníka. Tyto proměnné budou po ukončení výpočtu udávat požadovaný výsledek úlohy.

Správnost algoritmu plyne z uvedeného rozboru. Pokud zadaná matice obsahuje samé nuly, algoritmus nenalezne žádný přípustný levý horní roh obdélníka $A_{i,j} > 0$, čímž je tato situace detekována. Jestliže matice obsahuje alespoň jednu jedničku, musí obsahovat také nějaký maximální obdélník tvořený jedničkami. Dvojice proměnných i, j během výpočtu nabyde hodnot odpovídajících souřadnicím levého horního rohu tohoto maximálního obdélníka, neboť pomocí indexů i, j algoritmus postupně prochází všechny prvky pole A. Proměnná l potom jistě nabyde také hodnoty sloupcového indexu pravého horního rohu maximálního obdélníka z jedniček a tím bude tento maximální obdélník nalezen a uloží se údaje o jeho velikosti a souřadnicích. Jestliže lze v zadané matici A nalézt více různých obdélníků ze samých jedniček této maximální velikosti, vyhledá algoritmus souřadnice rohů jednoho z nich (toho, který byl nalezen jako první).

Výpočet podle uvedeného algoritmu je jistě konečný, neboť počet průchodů každým z cyklů je předem omezen některým z rozměrů zadané matice. Načtení hodnot matice A ze vstupu a modifikace obsahu pole A v první fázi výpočtu vyžadují provedení nm operací. Ve druhé fázi výpočtu se nm způsoby volí levý horní roh zkoumaného obdélníka (indexy i, j) a pro každou takovou volbu se provádí nejvýše m voleb sloupcového indexu pravého horního rohu (proměnná l). Celkem se tedy provede řádově nm^2 výběrů horních rohů obdélníka. Kdybychom pro každou takto vybranou trojici i, j, l hledali v naší upravené matici A maximální jedničkový obdélník s levým horním rohem $A_{i,j}$ a s pravým horním rohem $A_{i,l}$ zvlášť, potřebovali bychom vykonat vždy až m operací na nalezení minima z čísel $A_{i,q}$ pro $q = j, j+1, \ldots, l$. Celý algoritmus by pak měl časovou složitost $O(nm^3)$. Tento výběr minima neboli určování velikosti maximálního jedničkového obdélníka je ovšem možné provádět zároveň s postupným výběrem indexu l, čímž dosáhneme celkové časové složitosti algoritmu $O(nm^2)$.

3. Zadanou posloupnost čísel X budeme procházet po jednotlivých číslech odpředu dozadu. V i-tém kroku výpočtu budeme sledovat, jak mohou vypadat rostoucí podposloupnosti vybrané z počátečního úseku posloupnosti X délky i, tzn. z posloupnosti $X(1), \ldots, X(i)$. Pro dosažení co nejúspornějšího a nejrychlejšího řešení úlohy si zavedeme pomocné pole $M[1..n]$, do něhož si budeme průběžně ukládat následující informaci: prvek $M[j]$ je v každém okamžiku roven minimální dosud známé hodnotě posledního prvku vybrané rostoucí podposloupnosti délky j. Další průběžně aktualizovaná proměnná k udává délku nejdelší dosud nalezené rostoucí podposloupnosti. V poli M jsou tedy definovány hodnoty $M_1, M_2, \ldots, M_k$. Po provedení i-tého kroku výpočtu budou tudíž splněny následující podmínky:

1) $1 \leqq i \leqq n$, $1 \leqq k \leqq i$,
2) k je délka nejdelší rostoucí podposloupnosti vybrané z posloupnosti $X(1), X(2), \ldots, X(i)$,
3) pro každé $j = 1, \ldots, k$ platí

$$M_j = \min\{X(i_j);\ \text{existují indexy } 1 \leqq i_1 < i_2 < \ldots < i_j \leqq i \text{ takové, že } X(i_1) < X(i_2) < \ldots < X(i_j)\}$$

Z poslední uvedené podmínky zřejmě plyne platnost nerovnosti $M_1 < \ldots < M_k$. Pokud totiž rostoucí vybraná podposloupnost délky j může končit číslem M_j, pak

existuje také vybraná podposloupnost délky $j-1$, která vznikne z předchozí uvažované podposloupnosti vynecháním posledního členu. Její poslední člen bude ovšem jistě menší než M_j, a tedy skutečně platí $M_{j-1} < M_j$.

Po provedení všech n kroků výpočtu bude proměnná k obsahovat délku maximální rostoucí podposloupnosti vybrané z celé zadané posloupnosti $X(1), \ldots, X(n)$, a právě to je požadovaný výsledek úlohy.

Zbývá ukázat, jakým způsobem provedeme aktualizaci hodnot proměnné k a údajů uložených v poli M při jednom kroku výpočtu. Uvažujme i-tý krok výpočtu a zpracování čísla $X(i)$ ze zadané posloupnosti. Je-li $X(i)$ větší než M_k, je možné prodloužit dosud nejdelší nalezenou rostoucí podposloupnost o číslo $X(i)$. Zvětšíme tedy hodnotu proměnné k o jedničku a pro nové k definujeme údaj M_k jako hodnotu čísla $X(i)$. V opačném případě není možné dosud nejdelší vybranou podposloupnost prodloužit a hodnota k se tedy nezmění. Může se ovšem stát, že číslo $X(i)$ nám umožní snížit některou z dříve stanovených hodnot M_j. Jak jsme již uvedli, platí stále nerovnost $M_1 < \ldots < M_k$. Je tedy možné najít takový index j, že

$$\begin{array}{llll} \text{buď} & j = 1 & \text{a} & X(i) \leqq M_1, \\ \text{nebo} & 1 < j \leqq k & \text{a} & M_{j-1} < X(i) \leqq M_j. \end{array}$$

Nastane-li ostrá nerovnost $X(i) < M_j$, můžeme snížit hodnotu M_j. Existuje totiž rostoucí vybraná podposloupnost délky $j-1$ končící číslem M_{j-1}, a protože $X(i) > M_{j-1}$, číslo $X(i)$ tuto podposloupnost prodlužuje na rostoucí podposloupnost délky j. Jejím posledním prvkem je právě číslo $X(i)$.

Uvedený rozbor je zároveň zdůvodněním správnosti navrženého algoritmu. Výpočet je jistě konečný, neboť je tvořen přesně n kroky představujícími zpracování jednotlivých prvků posloupnosti X. Konečnost každého z těchto kroků hned ukážeme. Při vhodné organizaci výpočtu lze dosáhnout časové složitosti algoritmu $O(n \log n)$. V každém z n kroků se totiž kromě jednoduchých akcí s konstantní časovou složitostí musí vyhledávat v poli M index j výše uvedené vlastnosti. Pokud bychom index j hledali prostým sekvenčním průchodem polem M, potřebovali bychom provést v každém kroku výpočtu až n operací, což by vedlo k celkové časové složitosti algoritmu $O(n^2)$. Vzhledem k uspořádání prvků pole M podle velikosti je zde ovšem možné určit index j binárním prohledáváním (půlením intervalů) a tedy s časovou složitostí $O(\log n)$. Odtud plyne časová složitost celého algoritmu $O(n \log n)$.