42. ročník matematické olympiády na středních školách

Kategorie PCategory P

In: Leo Boček (editor); Karel Horák (editor); Jozef Moravčík (editor); Václav Sedláček (editor); Jaromír Šimša (editor); Pavel Töpfer (editor): 42. ročník matematické olympiády na středních školách. Zpráva o řešení úloh ze soutěže konané ve školním roce 1992/1993. 34. mezinárodní matematická olympiáda. 5. mezinárodní olympiáda v informatice. (Slovak). Praha: Jednota českých matematiků a fyziků, 2002. pp. 68–102.

Persistent URL: http://dml.cz/dmlcz/404974

Kategorie P

Texty úloh

P – I – 1

Úloha o výmene podreťazcov

Je daný reťazec znakov dĺžky N uložený v poli A. (V jazyku PASCAL by sme to mohli vyjadriť deklaráciou **var** A: **array** $[1\,..\,N]$ **of** *char*.) Hodnota N je pritom tak veľká, že A zaberá skoro celú operačnú pamäť; predstavte si napríklad, že A slúži ako pracovná pamäť editora a práve editujeme obrovský súbor.

Blok je súvislý podreťazec reťazca A začínajúci znakom s indexom z a končiaci znakom s indexom k ($1 \leqq z \leqq k \leqq N$); takýto blok budeme označovať $A[z\,..\,k]$. Bloky $A[z_1\,..\,k_1]$ a $A[z_2\,..\,k_2]$ sa *neprekrývajú*, ak $k_1 < z_2$ alebo $k_2 < z_1$. Napríklad podreťazce $A[4\,..\,5] = ak$ a $A[8\,..\,10] = abr$ sú neprekrývajúce sa bloky reťazca $A = abrakadabra$.

Napíšte a zdôvodnite algoritmus (program v programovacom jazyku), ktorý *vymení* dva neprekrývajúce sa bloky reťazca A, t. j. pre zadané indexy $z_1 \leqq k_1 < z_2 \leqq k_2$ premení reťazec

$$A = A[1\,..\,(z_1-1)]A[z_1\,..\,k_1]A[(k_1+1)\,..\,(z_2-1)]A[z_2\,..\,k_2]A[(k_2+1)\,..\,N]$$

na reťazec

$$A[1\,..\,(z_1-1)]A[z_2\,..\,k_2]A[(k_1+1)\,..\,(z_2-1)]A[z_1\,..\,k_1]A[(k_2+1)\,..\,N]$$

Napríklad výmenou vyznačených blokov v reťazci $A = abrakadabra$ by mal vzniknúť reťazec $A = abrabradaka$.

Algoritmus smie použiť len pomocnú pamäť konštantnej veľkosti (operačná pamäť je skoro plná, nemusí sa do nej zmestiť ani kópia menšieho z blokov) a počet krokov algoritmu by mal byť úmerný N (editor nesmie byť pomalý).

P – I – 2

Úloha o priemere mnohouholníka

Priemer rovinného útvaru S je najväčšia vzájomná vzdialenosť medzi dvoma bodmi S:

$$\operatorname{diam}(\mathsf{S}) = \max_{\substack{(x,y)\in\mathsf{S}\\(x',y')\in\mathsf{S}}} \sqrt{(x-x')^2+(y-y')^2}.$$

Body $X_1 = (x_1, y_1), \ldots, X_n = (x_n, y_n)$ $(n \geqq 3)$ tvoria (v tomto poradí) vrcholy konvexného n-uholníka M v rovine. Napíšte a zdôvodnite program, ktorý určí diam(M) — priemer mnohouholníka M!

Poznámka. Existuje algoritmus s lineárnou časovou zložitosťou.

P – I – 3

Multiplikatívna zložitosť

Lineárny algoritmus je algoritmus nasledujúceho tvaru:

Vstup: $x_1, x_2, \ldots, x_n$ (vstupné premenné)

$f_1 := g_1 \odot h_1$

$f_2 := g_2 \odot h_2$

$\vdots$

$f_i := g_i \odot h_i$

$\vdots$

$f_k := g_k \odot h_k$

Výstup: $y_1, y_2, \ldots, y_m$ (výstupné premenné)

pričom

(i) pre každé $i = 1, \ldots, k$ je $g_i, h_i \in \{x_1, x_2, \ldots, x_n\} \cup \{f_1, f_2, \ldots, f_{i-1}\}$ a operácia $\odot$ je jedna z operácií $+$, $-$, $*$ (sčítanie, odčítanie, násobenie).

(ii) pre každé $i = 1, \ldots, m$ je $y_i \in \{f_1, f_2, \ldots, f_k\}$.

Zložitosť lineárneho algoritmu je počet operácií $\odot$, **multiplikatívna zložitosť** lineárneho algoritmu je počet operácií $*$ (násobenie).

Napríklad nasledujúci lineárny algoritmus

Vstup: a, b, c, d

$f_1 := a * b$

$f_2 := c * d$

$f_3 := f_1 - f_2$

Výstup: f_3

počíta hodnotu výrazu $ab - cd$ a má multiplikatívnu zložitosť 2.

Úlohy

(a) Nájdite lineárny algoritmus, ktorý pre vstup x vypočíta hodnotu mnohočlena $1x+9x^2+9x^3+2x^4$ a má najmenšiu možnú multiplikatívnu zložitosť! Svoje tvrdenie zdôvodnite (ukážte, že neexistuje lineárny algoritmus s menšou multiplikatívnou zložitosťou než váš algoritmus!)

(b) Ukážte, že hodnota výrazu $ab - cd$ sa nedá vypočítať lineárnym algoritmom s multiplikatívnou zložitosťou 1!

P – I – 4

McCullochov stroj

McCullochov stroj je opísaný v študijnom texte.

1. Nájdite také číslo M, ktoré vytvára samo seba, t. j. $M \vdash M$!
2. Existuje N, ktoré vytvára svoje zdvojenie, t. j. $N \vdash NN$?
3. Existuje číslo P, ktoré vytvára číslo $P2$, t. j. $P \vdash P2$?
4. Existuje číslo Q, ktoré vytvára číslo $6Q$, t. j. $Q \vdash 6Q$?
5. (*McCullochov zákon*) Pre každé číslo A existuje také číslo X, že X vytvára AX, t. j. $X \vdash AX$. Dokážte!
6. Existuje také R, že $R \vdash R6$?

McCullochov stroj (študijný text)

McCullochov stroj slúži na spracovanie čísel. Má *vstup* a *výstup*; ak na vstup podáme *číslo*, t. j. konečnú neprázdnu postupnosť cifier 1, 2, 3, 4, 5, 6, 7, 8, 9, po konečnom čase sa na výstupe môže (ale nemusí) objaviť (iné) číslo — *odpoveď* stroja na vstupné číslo. V prvom prípade (t. j. ak sa odpoveď objaví), vstupné číslo sa nazýva *prijateľným*. Odpoveď stroja je vstupným číslom jednoznačne určená:

– ak vstupné číslo je prijateľné, tak sa odpoveď objaví vždy a je vždy rovnaká,

– ak vstupné číslo nie je prijateľné, nikdy sa neobjaví odpoveď a stroj sa po konečnom čase zastaví (t. j. môžeme zistiť, že vstupné číslo nie je prijateľné).

Stroj sa riadi určitými pravidlami — najprv ale uvedieme niekoľko definícií. Ako už vieme, čísla sú pre nás kladné celé čísla, ktorých desiatkový zápis neobsahuje nulu.

Ak X, Y sú čísla, tak XY označuje ich *spojenie*: číslo, ktoré dostaneme napísaním zápisov čísel X, Y za sebou. Napr. ak X je 53 a Y je 728, tak XY je 53728 a XYX je 5372853. Číslo $\underbrace{XX\ldots X}_{k\text{-krát}}$ zapíšeme ako X^k.

Zdvojením čísla X je číslo XX. Napr. zdvojením čísla $X = 12345$ je číslo $XX = 1234512345$.

Hovoríme, že číslo X *vytvára* číslo Y, ak X je prijateľné a po podaní X na vstup stroja odpoveďou je Y. Tento vzťah budeme skrátene zapisovať $X \vdash Y$.

McCullochov stroj sa riadi nasledujúcimi pravidlami:

Pravidlo P_1. *Pre ľubovoľné číslo* X, $2X2$ *je prijateľné a vytvára* X. *Skrátene,* $2X2 \vdash X$.

Pravidlo P_2. *Pre ľubovoľné čísla* X, Y, *ak* X *vytvára* Y, *tak* $7X$ *vytvára číslo* $Y2$. *Skrátene, z* $X \vdash Y$ *vyplýva* $7X \vdash Y2$.

Pravidlo P_3. *Pre ľubovoľné čísla* X, Y, *ak* X *vytvára* Y, *tak* $5X$ *vytvára zdvojenie čísla* Y. *Skrátene, z* $X \vdash Y$ *vyplýva* $5X \vdash YY$.

Pravidlo P_0. *Číslo* X *je prijateľné len vtedy, ak to vyplýva z pravidiel* P_1, P_2, P_3.

Napríklad, $2532 \vdash 53$ podľa P_1, $72532 \vdash 532$ podľa P_2, $572532 \vdash \vdash 532532$ podľa P_3, ale 4253 nie je prijateľné podľa P_0.

P – II – 1

Úloha o nákladných autách

Mestá A a B sú spojené rovným úsekom cesty. Na tejto ceste ležia mestá $A_1, \ldots, A_n, B_1, \ldots, B_n$ (nie nutne v tomto poradí, ale mesto A_i je bližšie k A než mesto B_i $(1 \leqq i \leqq n)$). Nákladné auto premáva medzi A a B a má n *objednávok*: previezť náklad z každého A_i do príslušného B_i; pritom naraz môže viezť len jeden náklad a po naložení v A_i ho môže vyložiť len v B_i.

Jazdou nákladného auta budeme rozumieť cestu auta z mesta A do mesta B (s prípadným splnením niektorých objednávok) a späť; počas jazdy smie zmeniť smer len v meste B.

Napíšte a zdôvodnite program, ktorý zostaví *plán* nákladného auta (t.j. určí počet jázd a splnené objednávky pre každú jazdu) tak, aby splnilo všetky objednávky minimálnym počtom jázd!

P – II – 2

Úloha o pokrytí štvorcom

Body $X_1 = (x_1, y_1)$, ..., $X_n = (x_n, y_n)$ $(n \geqq 3)$ tvoria (v tomto poradí) vrcholy konvexného n-uholníka M v rovine. Napíšte a zdôvodnite program, ktorý určí dĺžku strany štvorca S s nasledujúcimi vlastnosťami:

(a) S *pokrýva* M: $\mathsf{M} \subseteq \mathsf{S}$,

(b) aspoň jedna strana M leží na strane S,

(c) S má najmenší možný obsah.

Poznámka. V programe môžete využiť vzorce z analytickej geometrie (napr. pre vzdialenosť dvoch bodov alebo vzdialenosť bodu od priamky) ako funkcie; tieto funkcie nemusíte programovať.

P – II – 3

Multiplikatívna zložitosť

Lineárny algoritmus je algoritmus nasledujúceho tvaru:

Vstup: $x_1, x_2, \ldots, x_n$ (vstupné premenné)

$$f_1 := g_1 \odot h_1$$
$$f_2 := g_2 \odot h_2$$
$$\vdots$$
$$f_i := g_i \odot h_i$$
$$\vdots$$
$$f_k := g_k \odot h_k$$

Výstup: $y_1, y_2, \ldots, y_m$ (výstupné premenné)

pričom

(i) pre každé $i = 1, \ldots, k$ je $g_i, h_i \in \{x_1, x_2, \ldots, x_n\} \cup \{f_1, f_2, \ldots, f_{i-1}\}$ a operácia $\odot$ je jedna z operácií $+$, $-$, $*$ (sčítanie, odčítanie, násobenie);

(ii) pre každé $i = 1, \ldots, m$ je $y_i \in \{f_1, f_2, \ldots, f_k\}$.

Zložitosť lineárneho algoritmu je počet operácií $\odot$, **multiplikatívna zložitosť** lineárneho algoritmu je počet operácií $*$ (násobenie).

Úlohy

(a) Zistite, akú činnosť vykonáva nasledujúci lineárny algoritmus:

Vstup: $a_{11}, a_{12}, a_{21}, a_{22}, b_{11}, b_{12}, b_{21}, b_{22}$,

$$
\begin{aligned}
f_1 &= a_{11} + a_{22} & f_{14} &= a_{22} * f_5 \\
f_2 &= b_{11} + b_{22} & f_{15} &= f_6 * b_{22} \\
f_3 &= a_{21} + a_{22} & f_{16} &= f_7 * f_8 \\
f_4 &= b_{12} - b_{22} & f_{17} &= f_9 * f_{10} \\
f_5 &= b_{21} - b_{11} & f_{18} &= f_{11} + f_{14} \\
f_6 &= a_{11} + a_{12} & f_{19} &= f_{15} - f_{17} \\
f_7 &= a_{21} - a_{11} & f_{20} &= f_{11} + f_{13} \\
f_8 &= b_{11} + b_{12} & f_{21} &= f_{12} - f_{16} \\
f_9 &= a_{12} - a_{22} & f_{22} &= f_{18} - f_{19} \\
f_{10} &= b_{21} + b_{22} & f_{23} &= f_{12} + f_{14} \\
f_{11} &= f_1 * f_2 & f_{24} &= f_{13} + f_{15} \\
f_{12} &= f_3 * b_{11} & f_{25} &= f_{20} - f_{21} \\
f_{13} &= a_{11} * f_4
\end{aligned}
$$

Výstup: $f_{22}, f_{23}, f_{24}, f_{25}$

(b) Nájdite lineárny algoritmus s čo najmenšou multiplikatívnou zložitosťou, ktorý pre vstup $x_1, \ldots, x_n, y_1, \ldots, y_n, u_1, \ldots, u_n, v_1, \ldots, v_n$ vypočíta

$$
\begin{aligned}
&x_1u_1 + \ldots + x_nu_n, \\
&x_1v_1 + \ldots + x_nv_n, \\
&y_1u_1 + \ldots + y_nu_n, \\
&y_1v_1 + \ldots + y_nv_n.
\end{aligned}
$$

P – II – 4

McCullochov stroj

Operačné číslo je číslo zložené z cifier 5,7. Každé operačné číslo M *určuje operáciu* $M()$ v nasledujúcom zmysle: z $X \vdash Y$ vyplýva $MX \vdash M(Y)$.

Napríklad 5 určuje operáciu zdvojenia ($5(Y) = YY$), lebo $X \vdash Y \Rightarrow$ $\Rightarrow 5X \vdash YY$; 75 určuje operáciu pripísania 2 k zdvojeniu ($75(Y) =$ $= YY2$), lebo $X \vdash Y \Rightarrow 75X \vdash YY2$; $757(89) = 8928922$, lebo $7572892 \vdash$ $\vdash 8928922$.

Úlohy

1. (*Craigov zákon*) Pre každé operačné číslo M a pre každé číslo A existuje číslo X vytvárajúce $M(AX)$, t. j.

$$(\forall \text{ operačné } M)(\forall A)(\exists X)(X \vdash M(AX)).$$

Dokážte!

2. Nájdite čísla $X \neq Y$ také, že $X \vdash Y$ a zároveň $Y \vdash X$!

3. (*Fergussonov zákon*) Pre ľubovoľné A, B existujú čísla X, Y také, že $X \vdash AY$ a zároveň $Y \vdash BX$. Dokážte!

P – III – 1

Úloha o viditeľnosti mnohouholníka

Body $X_1 = (x_1, y_1)$, ..., $X_m = (x_m, y_m)$ ($m \geqq 3$) sú (v tomto poradí, proti smeru hodinových ručičiek) vrcholy konvexného m-uholníka M v rovine.

Mnohouholník M leží vo vnútri kruhu K so stredom $S = (x_S, y_S)$ a polomerom r (to znamená, že každé X_i je vnútorným bodom K).

Hovoríme, že bod u ležiaci na obvode (hranici) M je *viditeľný* z bodu v, ak úsečka uv neobsahuje vnútorné body M. (Všimnite si, že podľa tejto definície sú všetky body strany $x_i x_{i+1}$ viditeľné z každého bodu priamky $x_i x_{i+1}$; na druhej strane, obvod M je „neviditeľný“ pre vnútorné body M!)

Napíšte a zdôvodnite čo najrýchlejší program, ktorý

a) vypíše také údaje, z ktorých bude možné pre ľubovoľný bod v ležiaci na obvode kruhu K určiť množinu tých bodov z obvodu M, ktoré sú viditeľné z bodu v,

b) určí minimálny počet bodov ležiacich na obvode K tak, aby každý bod na obvode M bol viditeľný aspoň z jedného z týchto bodov.

Poznámka. V programe môžete využiť vzorce z analytickej geometrie (napr. pre vzdialenosť dvoch bodov alebo priesečník dvoch priamok, priamky a kružnice apod.) ako funkcie; tieto funkcie nemusíte programovať.

P – III – 2

Multiplikatívna zložitosť

Lineárny algoritmus je algoritmus nasledujúceho tvaru:

$$\begin{aligned}
&\textbf{Vstup: } x_1, x_2, \ldots, x_n \qquad \text{(vstupné premenné)}\\
&f_1 := g_1 \odot h_1\\
&f_2 := g_2 \odot h_2\\
&\vdots\\
&f_i := g_i \odot h_i\\
&\vdots\\
&f_k := g_k \odot h_k\\
&\textbf{Výstup: } y_1, y_2, \ldots, y_m \qquad \text{(výstupné premenné)}
\end{aligned}$$

pričom

(i) pre každé $i = 1, \ldots, k$ je $g_i, h_i \in \{x_1, x_2, \ldots, x_n\} \cup \{f_1, f_2, \ldots, f_{i-1}\}$ a operácia $\odot$ je jedna z operácií $+$, $-$, $*$ (sčítanie, odčítanie, násobenie);

(ii) pre každé $i = 1, \ldots, m$ je $y_i \in \{f_1, f_2, \ldots, f_k\}$.

Zložitosť lineárneho algoritmu je počet operácií $\odot$, **multiplikatívna zložitosť** lineárneho algoritmu je počet operácií $*$ (násobenie).

Zistite, akú činnosť vykonáva nasledujúci lineárny algoritmus:

$$\begin{aligned}
&\textbf{Vstup: } a,\ b,\ c,\ d\\
&f_1 = a + b \qquad f_5 = f_1 * f_2\\
&f_2 = c + d \qquad f_6 = f_5 - f_3\\
&f_3 = a * c \qquad f_7 = f_3 - f_4\\
&f_4 = b * d \qquad f_8 = f_6 - f_4\\
&\textbf{Výstup: } f_7,\ f_8
\end{aligned}$$

a rozhodnite, či existuje lineárny algoritmus počítajúci f_7 a f_8 s lepšou multiplikatívnou zložitosťou!

P – III – 3

Úloha o platení

Dané sú mince v hodnote $q_1, q_2, \ldots, q_n$ korún, kde $1 \leqq q_1 < q_2 < \ldots < q_n$ sú prirodzené čísla a q_{i+1}/q_i je nepárne prirodzené číslo pre $1 \leqq i < n$.

Platiteľ aj príjemca majú k dispozícii neobmedzený počet mincí každej hodnoty. Platiteľ ma zaplatiť obnos X korún príjemcovi. Platba je *optimálna*, ak sa pri nej vymení najmenší možný počet mincí; t. j. pre každé $1 \leqq i \leqq n$ dá platiteľ x_i ($\geqq 0$) mincí hodnoty q_i, príjemca vydá y_i ($\geqq 0$) mincí hodnoty q_i tak, že $\sum_{i=1}^{n} (x_i - y_i)q_i = X$, a pritom $\sum_{i=1}^{n}(x_i + y_i)$ je minimálne.

Napríklad pri minciach s hodnotou 1, 3 a 9 korún ($n = 3$) sa dá 17 korún vyplatiť tak, že platiteľ dá 1 deväťkorunáčku, 2 trojkorunáčky a 2 jednokorunáčky; vymení sa 5 mincí. Pri optimálnej platbe ovšem dá platiteľ 2 deväťkorunáčky a príjemca vráti 1 jednokorunáčku; vymenia sa 3 mince.

Napíšte a zdôvodnite program, ktorý pre vstupné celé kladné číslo X určí optimálny spôsob platby X korún!

P – III – 4

McCullochov stroj

Chovanie McCullochovho stroja je popísané v študijnom texte! Nájdete tam aj zhrnutie známych vlastností dokázaných v minulých kolách.

Číslo X_0 je *nesmrteľné*, ak nasledujúci proces je nekonečný (t. j. nevyskytne sa v ňom neprijateľné číslo):

Zadám stroju číslo X_0;
Ak sa objaví odpoveď X_1, zadám stroju číslo X_1;
Ak sa objaví odpoveď X_2, zadám stroju číslo X_2,
Ak sa objaví odpoveď X_3, zadám stroju číslo X_3,
... ; atď.

Otázky:

1. Nájdite 4 nesmrteľné čísla!
2. Nájdite 1993 nesmrteľných čísel!
3. *McCullochov algoritmus zisťovania nesmrteľnosti čísel*
   Základom algoritmu je číslo H také, že (X je nesmrteľné) $\Leftrightarrow$ (HX nie je nesmrteľné). Použijeme dva stroje — **S1** a **S2**.
   VSTUP: Číslo X.
   OTÁZKA: Je X nesmrteľné?
   ALGORITMUS

1. $Y := X$; $U := HX$;
2. Stroju **S1** zadáme číslo Y; stroju **S2** zadáme číslo U;
3. AK stroj **S1** vydá odpoveď Z, TAK $Y := Z$

INAK STOP; Číslo X nie je nesmrteľné!

4. AK stroj **S2** vydá odpoveď V, TAK $U := V$
INAK STOP; Číslo X je nesmrteľné!

5. SKOK na **2**.

Je tento algoritmus správny?

P – I – 1

Uvažujme o „veľkom“ bloku $A[z_1 \,..\, k_2]$, ktorý zahŕňa obidva vyznačené bloky $A[z_1 \,..\, k_1]$, $A[z_2 \,..\, k_2]$ aj nevyznačený blok $A[(k_1 + 1) \,..\, (z_2 - 1)]$ medzi nimi; tento „medziblok“ je v prípade $k_1 = z_2 - 1$ prázdny. Základom algoritmu je pozorovanie, že obrátením poradia znakov v bloku $A[z_1 \,..\, k_2]$ sa vyznačené bloky dostanú na „správne“ miesto, ale ich znaky budú v obrátenom poradí. To sa dá napraviť tým, že pred obrátením veľkého bloku obrátime jeho „podbloky“.

Aby sme to mohli presnejšie sformulovať, označme X^{R} reťazec, ktorý dostaneme obrátením poradia znakov v reťazci X. Príklad:

$$(abrakadabra)^{\mathrm{R}} = arbadakarba.$$

Potom pre ľubovoľné reťazce U, V, W, X, Y platí

$$U(V^{\mathrm{R}}W^{\mathrm{R}}X^{\mathrm{R}})^{\mathrm{R}}Y = UXWVY, \qquad (*)$$

t. j. postupnosť štyroch operácií obrátenia bloku vymení bloky V, X v reťazci $UVWXY$. (Bloky dĺžky $\leqq 1$ nemusíme obracať.) Príklad:

$$ab((raka)^{\mathrm{R}}(d)^{\mathrm{R}}(ab)^{\mathrm{R}})^{\mathrm{R}}ra = ab((akar)(d)(ba))^{\mathrm{R}}ra = ab(ab)(d)(raka)ra.$$

K dokončeniu algoritmu si stačí všimnúť, že blok sa dá obrátiť v čase úmernom jeho dĺžke s použitím pomocnej pamäti konštantnej veľkosti.

Program v PASCALe:

```
program VYMENA;
  const N = ...;
  type index = 1..N;
  var
    A : array [index] of char;
    z1, z2, k1, k2, zpom, kpom : index;
  procedure OBRAT(zac, kon : index);
      (* Ak zac ≧ kon, neurobí nič *)
    var
      pom : char;
    begin
```

```
      while (zac < kon) do begin
        pom := A[zac];
        A[zac] := A[kon];
        A[kon] := pom;
        zac := zac + 1;
        kon := kon - 1;
      end;   (* while *)
    end;   (* OBRAT *)
  begin   (* hlavný program *)
    ...   (* Načítanie reťazca A a medzí blokov: *)
    ...   (* z1 ≦ k1, z2 ≦ k2 *)
    if (z1 > z2) then begin
      zpom := z1; z1 := z2; z2 := zpom;
      kpom := k1; k1 := k2; k2 := kpom;
    end;
    if (k1 ≧ z2) then
      writeln('Bloky sa prekrývajú!')
    else
      begin
        OBRAT(z1, k1);
        OBRAT(k1 + 1, z2 - 1);
        OBRAT(z2, k2);
        OBRAT(z1, k2);
      end
  end.   (* hlavný program *)
```

Správnosť algoritmu vyplýva zo vzťahu (∗); správnosť procedúry $OBRAT$ je očividná.

Odhad časovej a pamäťovej zložitosti: Procedúra $OBRAT$ sa vyvolá štyrikrát; jedno jej prevedenie pozostáva z nanajvýš $c \cdot l$ krokov, kde l je dĺžka bloku, ktorý chceme obrátiť, a c je vhodná konštanta. Celkový počet krokov je teda nanajvýš

$$c \cdot [(k_1 + 1 - z_1) + (z_2 - k_1 - 1) + (k_2 + 1 - z_2) + (k_2 + 1 - z_1)] \leqq$$
$$\leqq 2c \cdot (k_2 + 1 - z_1) \leqq 2c \cdot N,$$

t. j. lineárny v N. Zo zápisu programu je jasné, že sme použili pomocnú pamäť konštantnej veľkosti.

Vzdialenosť bodov X, Y označíme $d(X,Y)$; vzdialenosť bodu X od priamky YZ označíme $d(X,YZ)$. V ďalšom budeme predpokladať, že vrcholy M sú usporiadané (v cyklickom zozname $X_1, X_2, \ldots, X_n, X_1$) proti smeru hodinových ručičiek, a položíme $X_0 = X_n$, $X_{n+1} = X_1$.

Kľúčovým pozorovaním je nasledujúce

Tvrdenie. $\operatorname{diam}(\mathsf{M}) = \operatorname{diam}(\{X_1, \ldots, X_n\})$.

NÁZNAK DÔKAZU. Prijmime (bez dôkazu) fakt, že existujú body X, $Y \in \mathsf{M}$, ktoré majú vzdialenosť $\operatorname{diam}(\mathsf{M})$. Tieto body zrejme ležia na hranici M (inak by vzdialenosť priesečníkov priamky XY s hranicou M bola väčšia, než $d(X,Y)$.) Keby X ležalo vnútri hrany X_iX_{i+1}, bolo by $d(X,Y) < \max\{d(X_i,Y), d(X_{i+1},Y)\}$, čo je spor s definíciou bodov X a Y. Podobne sa nahliadne, že Y nemôže byť vnútorným bodom hrany M, a teda X aj Y sú vrcholy M, čo bolo treba dokázať.

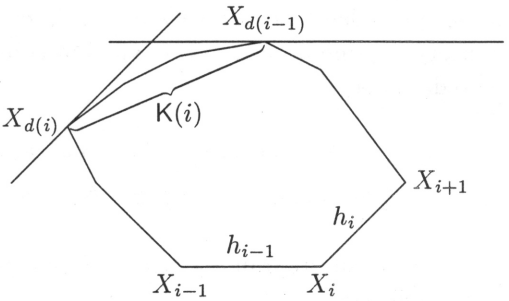
Vyzbrojení týmto aparátom ľahko navrhneme kvadratický algoritmus spočívajúci vo vyskúšaní každej dvojice vrcholov. Keby sa nám podarilo zredukovať úlohu na vyskúšanie lineárneho počtu *dvojíc* vrcholov, dostali by sme lineárny algoritmus.

Predpokladajme najprv, že M nemá dvojicu rovnobežných hrán. Pre každú hranu $h_i = X_iX_{i+1}$ $(1 \leqq i \leqq n)$ označíme $X_{d(i)}$ vrchol M najviac vzdialený od priamky X_iX_{i+1}.

Pre každý vrchol X_i definujeme množinu *kandidátov* pre vrchol $X_{d(i)}$:

$$\mathsf{K}(i) = \left\{X_{d(i-1)}, X_{d(i-1)+1}, \ldots, X_{d(i)-1}, X_{d(i)}\right\}.$$

Pozorovanie (obr. 27). *Pre* $1 \leqq i, j \leqq n$ *ak* $X_j \notin \mathsf{K}(i)$, *potom* $\operatorname{diam}(\mathsf{M}) > d(X_i, X_j)$.



Obr. 27

Intuitívne, ak X_j nie je kandidátom pre i, tak je *vylúčené*, aby sa priemer dosahoval na dvojici vrcholov X_i, X_j.

DÔKAZ. Ak totiž $x_j \notin \mathsf{K}(i)$ a $X_j \neq X_i$, tak

$$X_j \in \{X_{i+1}, \ldots, X_{d(i-1)-1}\} \quad \text{alebo} \quad X_j \in \{X_{d(i)+1}, \ldots, X_{i-1}\}.$$

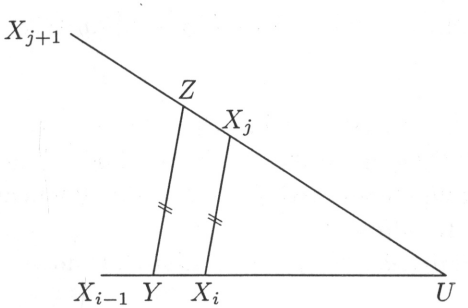
V prvom prípade

$$d(X_{j+1}, X_{i-1}X_i) > d(X_j, X_{i-1}X_i);$$

zostrojíme priamku rovnobežnú s X_iX_j pretínajúcu hrany h_{i-1}, h_j vo vnútorných bodoch Y, Z a priesečník U priamok $X_{j+1}X_j$ a $X_{i-1}X_i$. Potom (obr. 28) uhol X_iUX_j je ostrý (podľa predchádzajúcej nerovnosti), a preto

$$\operatorname{diam}(\mathsf{M}) \geqq d(Y, Z) > d(X_i, X_j),$$

čo bolo treba ukázať. V druhom prípade sa postupuje podobne.



Obr. 28

Pre každé i teda stačí vyskúšať dvojice (X_i, X_j), $X_j \in \mathsf{K}(i)$. V prípade $h_i \parallel h_j$ sa situácia zmení len v tom, že sa musia vyskúšať všetky dvojice (X_i, X_j), (X_i, X_{j+1}), (X_{i+1}, X_j), (X_{i+1}, X_{j+1}).

V každom prípade tvorí množina kandidátov vrcholu X_i „súvislý" úsek vrcholov M a „susedné" množiny kandidátov (t. j. pre X_i, X_{i+1}) majú nanajvýš dva spoločné prvky. To je základom nasledujúceho algoritmu.

Program v PASCALe:

program *PRIEMER*;

```
const
  n = ...;
type
  bod=record x, y : real end;
var
  m : array [1..n] of bod;
  i, j : 1..n;
  diam : real;
function d(i, j : index) : real;   (* vzdialenosť bodov *)
  begin d := sqrt(sqr(m[i].x − m[j].x) + sqr(m[i].y − m[j].y)); end;
function dalsi(i : index) : index;   (* ďalší bod v cykl. usporiadaní *)
  begin dalsi := i mod n + 1; end;
function max(x, y : real) : real;   (* maximum čísel *)
  begin if (x > y) then max := x else max := y; end;
function P(i, j, k : index) : real;   (* obsah trojuholníka *)
  begin
    P := ((m[j].x − m[i].x) * (m[k].x − m[i].x)
          − (m[j].y − m[i].y) * (m[k].y − m[i].y))/2;
  end;
begin
  ...   (* Načítanie súradníc zadaných bodov *)
    (* POZOR! Program neoveruje, či dané body skutočne tvoria *)
    (* (v zadanom poradí) vrcholy konvexného mnohouholníka!!! *)
  i := n;   j := 1;   diam := 0;
  while P(i, dalsi(i), dalsi(j)) > P(i, dalsi(i), j) do
    j := dalsi(j);
  while j ≠ 1 do begin
    i := dalsi(i);
    diam := max(diam, d(i, j));
    while P(i, dalsi(i), dalsi(j)) > P(i, dalsi(i), j) do begin
      j := dalsi(j);
      diam := max(diam, d(i, j));
    end;
    if P(i, dalsi(i), dalsi(j)) = P(i, dalsi(i), j) then
      diam := max(diam, d(i, dalsi(j)));
  end;
  writeln(diam);
end.
```

Odhad časovej zložitosti: Priradenie $j := dalsi(j)$ sa vykoná presne n-krát; priradenie $i := dalsi(i)$ sa vykoná nanajvýš n-krát. Telo **if**-príkazu sa vykoná nanajvýš $\frac{1}{2}n$-krát (nemôže byť viac dvojíc paralelných hrán). Každé priradenie hodnoty premennej *diam* je spojené s priradením hodnoty premennej i alebo j alebo s vykonaním tela **if**-príkazu; to znamená, že premennej *diam* sa priradí hodnota nanajvýš $\frac{5}{2}n$-krát. Každý príkaz v programe sa vykoná nanajvýš $\frac{5}{2}n$-krát, algoritmus má teda lineárnu časovú zložitosť.

Poznámky k implementácii: Vzdialenosti bodov od priamky (určenej úsečkou) porovnávame pomocou obsahov trojuholníkov určených týmito bodmi a danou úsečkou. Je to efektívnejšie, než mechanická aplikácia postupu z analytickej geometrie, hlavne ak vezmeme do úvahy, že chceme len *porovnávať* vzdialenosti, a nepotrebujeme ich explicitne spočítať.

P – I – 3

(a) Napríklad nasledujúci lineárny algoritmus

Vstup: x

$$f_1 = x * x \qquad f_4 = x + x \qquad f_7 = f_3 + f_6$$
$$f_2 = f_1 + x \qquad f_5 = f_4 + f_4 \qquad f_8 = f_2 * f_7$$
$$f_3 = f_1 + f_2 \qquad f_6 = f_4 + f_5 \qquad f_9 = f_8 + f_3$$

Výstup: f_9

má multiplikatívnu zložitosť 2.

Lineárny algoritmus pre vstup x nepoužívajúci násobenie dokáže spočítať iba polynómy tvaru kx, kde $k \in \mathbb{Z}$. Ak lineárny algoritmus používa jediné násobenie, potom činitele sú tvaru k_1x, k_2x a výsledok násobenia je $k_1k_2x^2$. Teda lineárny algoritmus dokáže spočítať iba polynómy tvaru $kx^2 + jx$, $k, j \in \mathbb{Z}$, t. j. iba kvadratické polynómy. Náš polynóm je štvrtého stupňa, a preto neexistuje lineárny algoritmus s multiplikatívnou zložitosťou 1, ktorý ho počíta.

(b) Sporom dokážeme, že $ab - cd$ sa nedá spočítať lineárnym algoritmom zložitosti 1.

Lineárny algoritmus pre vstup a, b, c, d nepoužívajúci násobenie dokáže spočítať iba polynómy tvaru $k_1a + k_2b + k_3c + k_4d$, $k_i \in \mathbb{Z}$, lineárny algoritmus používajúci práve jedno násobenie dokáže spočítať iba polynómy tvaru

$$v_1(t_1a+t_2b+t_3c+t_4d)(u_1a+u_2b+u_3c+u_4d)+v_2a+v_3b+v_4c+v_5d. \quad (*)$$

Keby $ab - cd$ bolo tvaru $(*)$, potom z porovnania máme (vzhľadom na symetriu možno bez újmy na všeobecnosti predpokladať $t_1 \neq 0$):

$$\begin{array}{llll} t_1u_1 = 0, & (1) & t_1u_3 + t_3u_1 = 0, & (5) \\ t_2u_2 = 0, & (2) & t_1u_4 + t_4u_1 = 0, & (6) \quad v_1(t_1u_2 + t_2u_1) = 1, \quad (9) \\ t_3u_3 = 0, & (3) & t_2u_3 + t_3u_2 = 0, & (7) \quad v_1(t_3u_4 + t_4u_3) = -1, \quad (10) \\ t_4u_4 = 0, & (4) & t_2u_4 + t_4u_2 = 0, & (8) \end{array}$$

$$v_1 \neq 0, \quad v_2 = v_3 = v_4 = v_5 = 0,$$

potom $u_1 = 0$ z (1), $u_3 = 0$ z (5), $t_3 \neq 0$ a $u_4 \neq 0$ z (10), $t_4 = 0$ z (4), $t_2 = 0$ z (8) a $u_2 \neq 0$ z (9). Teda $t_1, t_3, u_2, u_4 \neq 0$, $u_1, u_3, t_2, t_4 = 0$, avšak to je podľa (6) spor:

$$0 \neq t_1u_4 = t_1u_4 + 0 = t_1u_4 + t_4u_1 = 0.$$

Všimnime si, že hodnota výrazu $a \cdot a - b \cdot b = a^2 - b^2$ sa dá vypočítať lineárnym algoritmom zložitosti 1!

P – I – 4

1. Keby sme našli také H, že

$$X \vdash Y \Rightarrow HX \vdash Y2Y2, \qquad (*)$$

po dosadení $X := 2Y2$ by sme dostali $2Y2 \vdash Y \Rightarrow H2Y2 \vdash Y2Y2$, čo dáva (spolu s P_1 a po dosadení $Y := H$) $H2H2 \vdash H2H2$ a stačilo by položiť $M := H2H2$.

Podmienka $(*)$ nám vlastne hovorí, že H spôsobí pripísanie 2 na koniec čísla a následné zdvojenie. Z toho je vidieť, že $H = 57$ vyhovuje $(*)$. Riešením je teda $M = 572572$. Skutočne, $2572 \vdash 57$ (P_1), $72572 \vdash 572$ (P_2), $572572 \vdash 572572$ (P_3).

Ak $(*)$ nahradíme podmienkou $X \vdash Y \Rightarrow HX \vdash YY2$, po dosadení $Y := H2$, $X := 2Y2 = 2H22$, $H := 75$ dostaneme riešenie $M = 7527522$ (overte!).

Teraz príde malý podraz: vyriešime hneď 5. otázku.

5. Chceme nájsť X také, že $X \vdash AX$. Hľadajme ho v tvare $X = 5Y$ (prijateľné sú len čísla tvaru $2Y2$, $5Y$ a $7Y$). Potom $Y \vdash Z \Rightarrow X = 5Y \vdash ZZ$. My ale chceme, aby $ZZ = AX = A5Y$. Teraz je rozumné predpokladať, že Y je dlhšie než A, t. j. $Z = A5U$. Hľadáme teda také U,

že $UA5U \vdash A5U$. Prečo by U nemohlo byť tvaru $U = 7V$? (5 sme už mali, skúsme 7!) Teraz chceme splniť podmienku $7VA57V \vdash A57V$. Pritom $VA57V \vdash W \Rightarrow 7VA57V \vdash W2$, t.j. by malo platiť $W2 = A57V$. Túto podmienku splníme položením $W := A57$, $V := 2$. Po spätnom dosadení dostaneme $X = 572A572$. Ľahko sa overí, že toto X vyhovuje.

Keď v predchádzajúcej úvahe prehodíme úlohu 5 a 7, dostaneme druhé riešenie $X = 752A7522$ (overte!)

Poznámka. Ak dovolíme, aby A bolo „prázdne číslo", vyriešili sme 1. úlohu iným spôsobom (dostali sme ovšem rovnaké riešenia).

2. Skúsme hľadať N v tvare $5X$. Ak $X \vdash 5X$, potom $5X \vdash 5X5X$. T.j. (použijeme McCullochov zákon pre $A = 5$) $N = 55725572$ alebo $N = 575257522$ vyhovuje.

3. Hľadajme P v tvare $7X$. $X \vdash 7X \Rightarrow 7X \vdash 7X2$, teda možné riešenia sú $P = 75727572$, $P = 775277522$.

4. To je triviálna aplikácia McCullochovho zákona: $Q = 5726572$ alebo $Q = 75267522$.

6. Predpokladajme, že existuje číslo R vytvárajúce $R6$; z tohoto predpokladu odvodíme spor.

Počet cifier čísla Y budeme označovať $|Y|$, počet jeho pätiek resp. sedmičiek $\sharp_5(Y)$ resp. $\sharp_7(Y)$.

Prijateľné čísla sú tvaru $H2Y2$, kde H pozostáva z cifier 5, 7, Y je ľubovoľné, $|Y| \geqq 1$.

Pozorovanie. *Nech* $X \vdash Y$, $HX \vdash Z$. *Potom* $|Z| \geqq 2^{\sharp_5(H)}|Y| + \sharp_7(H)$.

Číslo R je tvaru $H2Y2$, lebo je prijateľné; pritom R vytvára číslo $R6 = H2Y26$. Z Pozorovania vyplýva $|H2Y26| \geqq 2^{\sharp_5(H)}|Y| + \sharp_7(H)$, t.j.

$$\sharp_5(H) + \sharp_7(H) + |Y| + 3 \geqq 2^{\sharp_5(H)}|Y| + \sharp_7(H),$$
$$\sharp_5(H) + 3 \geqq \left(2^{\sharp_5(H)}|Y| - 1\right)|Y|,$$

a teda $\sharp_5(H) \in \{0, 1, 2\}$, lebo $|Y| \geqq 1$.

Rozlíšime tri prípady:

(1) $\sharp_5(H) = 0$. V tomto prípade R je tvaru 7^s2Y2, ale $7^s2Y2 \vdash Y2^s$ a $|Y2^s| < |7^s2Y26|$ — spor.

(2) $\sharp_5(H) = 1$. V tomto prípade $R = 7^a57^b2Y2 \vdash Y2^bY2^{a+b} = R6$, t.j. $a + b = 0$, $a = b = 0$, $R = 52Y2 \vdash YY = R6$ a porovnanie dĺžok ($|R| + 1 = |R6|$) dáva $|Y| + 4 = 2|Y|$, $Y = 4$, pričom Y končí cifrou 6. Teda $Y = uvw6$ a podmienka $52uvw626 = R6 = uvw6uvw6$ vedie k sporu (porovnanie 2. cifier dáva $v = 2$, porovnanie 6. cifier $6 = v$).

(3) $\sharp_5(H) = 2$. V tomto prípade $R = 7^a57^b57^c2Y2 \vdash Y2^cY2^{b+c}Y2^{a+b+c}$ a podobne ako vyššie dostaneme $a = b = c = 0$, $R = 552Y2 \vdash YYYY$ a porovnanie dĺžok vedie k sporu: $4|Y| = |YYYY| = |R6| = |R| + 1 = |Y| + 5 \Rightarrow 3|Y| = 5$.

Zhrnutie: neexistuje R také, že $R \vdash R6$.

P – II – 1

Označme $\alpha_i = |AA_i|$, $\beta_i = |AB_i|$. Podľa zadania $0 \leqq \alpha_i < \beta_i$ pre $1 \leqq \leqq i \leqq n$. V ďalšom *objednávka j* bude označovať objednávku patriacu mestám A_j, B_j.

Myšlienka algoritmu je jednoduchá: pokiaľ sú nejaké nesplnené objednávky, pridáme ďalšiu jazdu a simulujeme cestu auta z A do B; ak v nejakom meste je auto voľné, vyberieme si ako ďalšiu tú objednávku zo zatiaľ nesplnených, ktorá začína čo najskôr.

Efektívna implementácia tejto myšlienky vedie k príliš zložitému programu. Nebudeme preto priraďovať objednávky jazdám, ale naopak. Vektory $(\alpha_1, \ldots, \alpha_n)$ a $(\beta_1, \ldots, \beta_n)$ usporiadame vzostupne a potom „prechádzame“ úsek AB. Pritom si udržiavame zatiaľ potrebný počet jázd v premennej p a zoznam „voľných jázd“ (spočiatku je prázdny). Ak narazíme na mesto A_i a zoznam je prázdny, zvýšime počet jázd na $p + 1$ a i-tu objednávku splníme v $(p + 1)$-vej jazde. Ak zoznam je neprázdny, vyberieme si z neho voľnú jazdu a i-tu objednávku splníme v tejto jazde. Ak narazíme na mesto B_i, jazdu, v ktorej sme splnili i-tu objednávku, „uvoľníme“ — pridáme do zoznamu voľných jázd.

Program v PASCALe:

```
program MINJAZD;
  const n=...;
  type
    mesto=record
            vzd : real;   (* vzdialenosť od mesta A *)
            por : 1..n;   (* číslo mesta *)
          end;
    zoznam=record
             prvky : array [1..n] of 1..n;
             ukaz : 0..n;
           end;
```

```
var
  a, b : array [1..n] of mesto;
  plan : array [1..n] of 1..n;   (* jazdy priradené objednávkam *)
  volne_jazdy : zoznam;   (* zoznam voľných jázd *)
  i, j, p : 1..n;
procedure INIT(var z : zoznam);
  begin z.ukaz := 0; end;
function EMPTY(var z : zoznam) : boolean;
  begin EMPTY := (z.ukaz = 0) end;
function GET(var z : zoznam) : 1..n;
  begin GET := z.prvky[z.ukaz];   z.ukaz := z.ukaz-1; end;
procedure PUT(var z : zoznam; x : 1..n);
  begin z.ukaz := z.ukaz+1;   z.prvky[z.ukaz] := x; end ;
begin
  for i := 1 to n do begin   (* vstup *)
    read(a[i].vzd, b[i].vzd);
    a[i].por := i; b[i].por := i;
  end;   (* for *)
  INIT(volne_jazdy);
  SORT(a);
  SORT(b);
  i := 1; j := 1; p := 0;
  while i ≦ n do
    if a[i].vzd < b[j].vzd then begin
        (* začiatok objednávky a[i].por *)
      if EMPTY(volne_jazdy) then begin
        p := p+1;
        plan[a[i].por] := p;
      end else
        plan[a[i].por] := GET(volne_jazdy);
      i := i+1;
      end
      else begin   (* koniec objednávky b[j].por *)
        PUT(volne_jazdy, plan[b[j].por]);
        j := j+1;
      end;
  for i := 1 to n do   (* výsledok *)
    writeln('Objednávku č.', i, 'splníme v jazde č.', plan[i]);
end.
```

Poznámky k implementácii:

– pomocná procedúra *SORT* usporiada pole záznamov typu *mesto* vzostupne podľa hodnoty položky *vzd*. V položke *por* sa uchováva informácia o mestách — ak $a[i].por = j$, potom $|AA_j| = \alpha_j = a[i].vzd$, podobne pre mestá B_j.

– zoznam voľných jázd pripomína zásobník, ale to nie je podstatné — zvolili sme si čo najjednoduchšiu implementáciu. Tento zoznam nikdy nepretečie, lebo počet volaní procedúry *PUT* je zhora ohraničený číslom n.

DÔKAZ SPRÁVNOSTI ALGORITMU. Je jasné, že po skončení **while**-cyklu má každý prvok poľa *plan* priradenú hodnotu (všetky objednávky sú splnené). Objednávky s rovnakým číslom jazdy sú súčasne (t. j. počas jednej jazdy) splniteľné, lebo rovnosť $plan[a[i].por] = plan[a[i'].por] = q$ pre $i < i'$ znamená, že pri preberaní prvku $a[i']$ bola jazda q na zozname voľných jázd, t. j. koncový bod $\beta_{a[i].por}$ objednávky $a[i].por$ predchádza začiatočnému bodu $a[i'].vzd$ objednávky $a[i'].por$.

Dokážme teraz optimálnosť získaného plánu. Označme

$$k_0 = \max_{x \in \langle 0, |AB| \rangle} |\{i : x \in (\alpha_i, \beta_i\rangle\}|$$

maximálny počet intervalov $(\alpha_i, \beta_i\rangle$ s neprázdnym prienikom. Je jasné, že potrebujeme aspoň k_0 jázd. Ukážeme, že algoritmus vytvorí plán s presne k_0 jazdami.

Uvažujme také i, že počet jázd (premenná p) sa v programe zvyšuje z p_0 na $p_0 + 1$ pri preberaní počiatočného bodu $a[i].vzd$ objednávky $a[i].por$. Vtedy nie je k dispozícii žiadna voľná jazda, t. j. platí $EMPTY(volne\_jazdy)$. Ku každej jazde q, $1 \leqq q \leqq p_0$, priraďme index

$$i_q = \max\{i' : 1 \leqq i' < i \ \wedge \ plan[a[i'].por] = q\}.$$

Z $i_q < i$ vyplýva $\alpha_{a[i_q].por} = a[i_q].vzd \leqq a[i].vzd$; z toho, že pri preberaní $a[i].vzd$ jazda q nie je voľná, vyplýva $\beta_{a[i_q].por} > a[i].vzd$ (v prípade nerovnosti $\leqq$ by algoritmus preberal koncový bod $\beta_{a[i_q].por}$ pred počiatočným bodom $a[i].vzd$ a uvoľnil by jazdu q).

Zhrnutie: Bod $x = \alpha_{a[i].por} = a[i].vzd$ leží v p_0 intervaloch

$$(\alpha_{a[i_q].por}, \beta_{a[i_q].por}) \quad (1 \leqq q \leqq p_0);$$

bod $x + \varepsilon \in (\alpha_i, \beta_i\rangle$, kde ε je veľmi malé kladné číslo, leží v $p_0 + 1$ intervaloch, t. j. $k_0 \geqq p_0 + 1$ — algoritmus nikdy nezvýši hodnotu premennej p nad k_0.

Odhad časovej zložitosti: Program (okrem dvoch volaní procedúry *SORT*) v podstate pozostáva z jediného **while**-cyklu. V tele cyklu sa zvyšuje hodnota súčtu $i + j$, a pritom platí invariant $j \leqq i$ (v žiadnom úseku AX nemôže skončiť viac objednávok, než sa ich začalo), t. j. $i+j \leqq$ $\leqq 2i \leqq 2n$ — cyklus prebehne $\leqq 2n$-krát. Každé vykonanie tela cyklu trvá konštantný počet krokov (por. implementáciu operácií so zoznamom) — **while**-cyklus trvá $O(n)$ krokov.

Kritickým miestom programu je teda procedúra *SORT*. Nebudeme ju tu implementovať ani dokazovať, že horným odhadom jej časovej zložitosti (pri vhodnej implementácii) je *konštanta* $\times\, n \log_2 n$.

Horný odhad časovej zložitosti algoritmu je *konštanta* $\times\, n \log_2 n$.

P – II – 2

Vzdialenosť bodu X od priamky YZ označme $d(X, YZ)$, priemet rovinného útvaru X na priamku p označme $\mathrm{pr}(\mathsf{X}, p)$. Vzdialenosť bodu $C = (x_C, y_C)$ od priamky AB ($A = (x_A, y_A)$, $B = (x_B, y_B)$) určíme vzorcom

$$\mathrm{vzd}(A, B, C) = \frac{(y_C - y_A)(x_B - x_A) - (x_C - x_A)(y_B - y_A)}{\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}}.$$

Na rozdiel od $d(C, AB)$, $\mathrm{vzd}(A, B, C)$ má znamienko, a to kladné pre C ležiace naľavo od priamky AB (t. j. pre $|\sphericalangle BCA| \in (0, \pi)$) a záporné pre C ležiace napravo od tejto priamky.

V ďalšom budeme predpokladať, že vrcholy M sú usporiadané (v poli $X[1\,..\,n]$) proti smeru hodinových ručičiek, a položíme $X_{n+1} = X_1$.

Pre každé i, $1 \leqq i \leqq n$, nám stačí určiť nasledujúce dve čísla:

$$\textit{šírka}_i = \max_{1 \leqq j \leqq n} d(X_j, X_i X_{i+1}),$$

$$\textit{priemet}_i = \text{dĺžka } \mathrm{pr}(\mathsf{M}, X_i X_{i+1}).$$

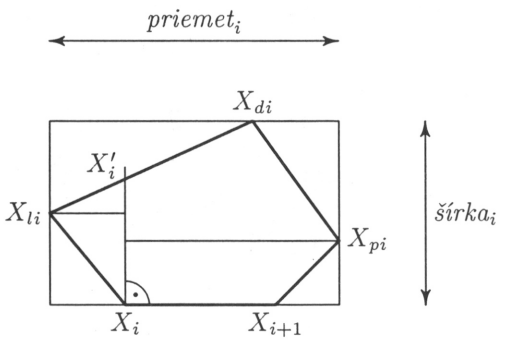
Hľadaná dĺžka strany S je potom $\min\limits_{1 \leqq i \leqq n} \max\{\textit{šírka}_i, \textit{priemet}_i\}$.

Pre každý index i určíme indexy di, li a pi nasledovne:

- X_{di} je vrchol M najviac vzdialený od priamky $X_i X_{i+1}$;
- $\mathrm{pr}(\mathsf{M}, X_i X_{i+1}) = \mathrm{pr}(X_{li} X_{pi}, X_i X_{i+1})$ a body $\mathrm{pr}(X_{li}, X_i X_{i+1})$, X_i, X_{i+1}, $\mathrm{pr}(X_{pi}, X_i X_{i+1})$ ležia na priamke $X_i X_{i+1}$ v tomto poradí (li — ľavý index, pi — pravý index).

Body X_{li}, X_{pi} sa dajú charakterizovať aj takto: položme $X_i' =$ obraz bodu X_{i+1} v otočení o $\pi/2$ okolo bodu X_i (X_i' leží vľavo od priamky X_iX_{i+1} a $X_iX_{i+1} \perp X_iX_i'$). Potom X_{li} (X_{pi}) je najvzdialenejší vrchol M ležiaci naľavo (napravo) od priamky X_iX_i' (obr. 29). Teda

$$\begin{aligned} \textit{šírka}_i &= d(X_{di}, X_iX_{i+1}) = \mathrm{vzd}(X_i, X_{i+1}, X_{di}), \\ \textit{priemet}_i &= d(X_{li}, X_iX_i') + d(X_{pi}, X_iX_i') \\ &= \mathrm{vzd}(X_i, X_i', X_{li}) - \mathrm{vzd}(X_i, X_i', X_{pi}). \end{aligned}$$



Obr. 29

Zostáva určiť, ako prejsť od indexov di, li, pi pre i k indexom di', li', pi' pre $i+1$. Predstavme si priamku, ktorá sa otáča z polohy X_iX_{i+1} do polohy $X_{i+1}X_{i+2}$. Vrchol M najvzdialenejší od tejto priamky sa postupne presúva z polohy X_{di} do polohy $X_{di'}$. Novú hodnotu di' určíme z podmienky $d(X_{di'}, X_{i+1}X_{i+2}) \geqq d(X_{di'+1}, X_{i+1}X_{i+2})$. Podobne určíme aj li' a pi'. Tieto úvahy dokazujú správnosť nasledujúceho programu, v ktorom $A = X_i$, $B = X_{i+1}$, $C = X_i'$ a $maxi = \max\{\textit{šírka}_i, \textit{priemet}_i\}$.

Odhad časovej zložitosti: Počiatočné nastavenie hodnôt di, li, pi trvá lineárny čas (prvé tri **while**-cykly). **for**-cyklus sa vykoná $(n-1)$-krát; vnorené **while**-cykly (v súčte pre všetky vykonania **for**-cyklu) trvajú lineárny čas, lebo indexy di, li, pi „obídu" mnohouholník M nanajvýš raz, ako to ukazuje vyššie uvedená úvaha s otáčajúcou sa priamkou.

Časová zložitosť algoritmu je teda lineárna.

Program v PASCALe:

```
program STVOREC;
const
  n = ...;
```

```
type
  bod = record x, y : real end;
var
  X : array [1..n] of bod;
  i, li, pi, di : 1..n;
  sirka, priemet, minstrana, maxi : real;
  A, B, C : bod;
function vzd(A, B, C : bod) : real;
  begin
    vzd := ((C.y - A.y) * (B.x - A.x) - (C.x - A.x) * (B.y - A.y))/
      sqrt((B.x - A.x) * (B.x - A.x) + (B.y - A.y) * (B.y - A.y));
  end;
function dalsi(i : integer) : integer;
  begin
    dalsi := i mod n + 1;
  end;
function max(x, y : real) : real;
  begin
    if x > y then max := x else max := y
  end;
function min(x, y : real) : real;
  begin
    if x < y then max := x else max := y
  end;
begin
  for i := 1 to n do
    readln(X[i].x, X[i].y);
  A := X[n];
  B := X[1];
  C.x := A.x - (B.y - A.y);
  C.y := A.y + (B.x - A.x);
  li := n; pi := 1; di := 1;
  while vzd(A, C, X[li - 1]) > vzd(A, C, X[li]) do li := li - 1;
  while vzd(A, C, X[pi + 1]) < vzd(A, C, X[pi]) do pi := pi + 1;
  while vzd(A, B, X[di + 1]) > vzd(A, B, X[di]) do di := di + 1;
  sirka := vzd(A, B, X[di]);
  priemet := vzd(A, C, X[li]) - vzd(A, C, X[pi]);
  maxi := max(sirka, priemet);
  minstrana := maxi;
```

```
    for i := 1 to n − 1 do begin
      A := X[i];
      B := X[i + 1];
      C.x := A.x − (B.y − A.y);
      C.y := A.y + (B.x − A.x);
      while vzd(A, C, X[dalsi(li)]) > vzd(A, C, X[li])
            do li := dalsi(li);
      while vzd(A, C, X[dalsi(pi)]) < vzd(A, C, X[pi])
            do pi := dalsi(pi);
      while vzd(A, B, X[dalsi(di)]) > vzd(A, B, X[di])
            do di := dalsi(di);
      sirka := vzd(A, B, X[di]);
      priemet := vzd(A, C, X[li]) − vzd(A, C, X[pi]);
      maxi := max(sirka, priemet);
      minstrana := min(maxi, minstrana);
    end;
end.
```

P – II – 3

(2a)

$$f_{22} = a_{1,1}b_{1,1} + a_{1,2}b_{2,1}, \qquad f_{24} = a_{1,1}b_{1,2} + a_{1,2}b_{2,2},$$
$$f_{23} = a_{2,1}b_{1,1} + a_{2,2}b_{2,1}, \qquad f_{25} = a_{2,1}b_{1,2} + a_{2,2}b_{2,2},$$

teda matica $\begin{pmatrix} f_{22} & f_{24} \\ f_{23} & f_{25} \end{pmatrix}$ je výsledkom násobenia matíc:

$$\begin{pmatrix} f_{22} & f_{24} \\ f_{23} & f_{25} \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} * \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix}.$$

(2b) Ak n je párne, potom $\frac{1}{2}n$-krát použijeme algoritmus z časti (2a) postupne pre vstup $x_i, x_{i+1}, y_i, y_{i+1}, u_i, v_i, u_{i+1}, v_{i+1}, i = 1, 3, \ldots, \frac{1}{2}n-1$, pomocou $\frac{7}{2}n$ násobení získame

$$x_iu_i + x_{i+1}u_{i+1},$$
$$x_iv_i + x_{i+1}v_{i+1},$$
$$y_iu_i + y_{i+1}u_{i+1},$$
$$y_iv_i + y_{i+1}v_{i+1}$$

$(i = 1, 3, \ldots, \frac{1}{2}n - 1)$ a sčítaním príslušných medzivýsledkov dostaneme požadované súčty. Ak n je nepárne, potom $n - 1$ je párne a súčty

$$\sum_{i=1}^{n-1} x_i u_i, \quad \sum_{i=1}^{n-1} x_i v_i, \quad \sum_{i=1}^{n-1} y_i u_i, \quad \sum_{i=1}^{n-1} y_i v_i$$

získame popísaným spôsobom pomocou $\frac{7}{2}(n-1)$ násobení a ďalších 4 násobení $x_n u_n$, $x_n v_n$, $y_n u_n$, $y_n v_n$. Požadované súčty tak získame použitím $\frac{7}{2}(n-1)+4 = \frac{1}{2}(7n+1)$ násobení.

P – II – 4

1. Ak $Y \vdash AMY$, tak $MY \vdash M(AMY)$ podľa definície operácie $M()$. Stačí teda nájsť takéto Y (ale to je McCullochov zákon) a položiť $X := MY$.

Odpoveď: $X = M572AM572$ alebo $X = M752AM7522$.

2. Zrejme $2X2 \vdash X$ (pravidlo P_1); stačí teda nájsť také X, že $X \vdash \vdash 2X2$. Ovšem $2X2 = 7(2X)$ a môžeme použiť Craigov zákon s $M = 7$, $A = 2$: dostávame $X = 757227572$ a $X = 7752277522$.

3. Postupujme podobne ako v predchádzajúcom riešení: $2BX2 \vdash BX$, teda hľadáme také X, že $X \vdash A2BX2 = 7(A2BX)$. Craigov zákon (s $M = 7$, $A = A2B$) dáva riešenia $X = 7572A2B7572$ alebo $X = 7752A2B77522$.

P – III – 1

Obvod (hraničnú kružnicu) kruhu K označíme C. Body $u, v \in \mathsf{C}$ jednoznačne určujú oblúk $\overset{\frown}{uv}$ (od u k v proti smeru hod. ručičiek). Položíme $X_k = X_{(k \bmod m)+1}$ pre $k \in \mathbb{Z}$. Pre $i, j, k \in \{1, 2, \ldots, m\}$, *$j$ leží medzi i a k*, ak $i \leqq j \leqq k$, $k \leqq i \leqq j$ alebo $j \leqq k \leqq i$ (tento fakt v programe zistí boolovská funkcia $medzi(i, j, k)$).

Ak bod u ležiaci vnútri hrany $X_k X_{k+1}$ je viditeľný z bodu $v \in \mathsf{C}$, z definície viditeľnosti vyplýva, že celá hrana $X_k X_{k+1}$ je viditeľná z bodu v. Z toho a z konvexnosti M vyplýva, že z bodu v na kružnici C je viditeľný úsek („*množina viditeľnosti*“) $\mathsf{M}_\mathsf{v}(v) = X_i X_{i+1} \cup X_{i+1} X_{i+2} \cup \ldots \cup X_{j-1} X_j$ z obvodu M. Tento úsek je jednoznačne určený (usporiadanou) dvojicou indexov (i, j) (v ďalšom budeme rozumieť pod (i, j) buď usporiadanú dvojicu, alebo úsek obvodu M medzi X_i a X_j — z kontextu bude jasné, ktorý význam máme na mysli). Obrátene, každý vrchol X_i je viditeľný

z oblúka $\widehat{zac_i kon_i} \subseteq \mathsf{C}$, kde $zac_i = \overrightarrow{X_i X_{i-1}} \cap \mathsf{C}$ („*začiatok*“) a $kon_i = \overrightarrow{X_i X_{i+1}} \cap \mathsf{C}$ („*koniec*“). Navrhneme algoritmus, ktorý určí body zac_i, kon_i $(1 \leqq i \leqq m)$, usporiada ich na kružnici C a tým rozdelí C na $2m$ oblúkov s konštantnými množinami viditeľnosti (niektoré oblúky môžu byť pritom jednobodové). Pre každý oblúk určíme aj $\mathsf{M_v}$ — úsek (i, j). $\mathsf{M_v}$ pre koncové (t. j. styčné) body oblúkov určíme tak, že porovnáme množiny viditeľnosti susediacich oblúkov a $\mathsf{M_v}$ styčného bodu je väčšia z nich. Výsledkom časti a) bude (cyklický) zoznam koncových bodov oblúkov s príslušnými usporiadanými dvojicami (i, j) (t. j. pre v na danom oblúku je $\mathsf{M_v}(v) = (i, j)$); prvé, resp. druhé prvky týchto dvojíc budú v zozname usporiadané proti smeru hodinových ručičiek.

Aby sme nemuseli triediť, budeme prechádzať priesečníkmi proti smeru hodinových ručičiek. Ak „prejdeme“ cez zac_j, $\mathsf{M_v}$ sa rozšíri o $X_{j-1}X_j$; ak „prejdeme“ cez kon_i, z $\mathsf{M_v}$ vypadne hrana $X_{i+1}X_i$. Daný priesečník sa pritom stane koncovým bodom oblúka (pole *obluky* v programe). Cyklus naštartujeme tým, že (tentokrát výnimočne v smere hod. ručičiek) nájdeme vrchol X_i, ktorý nie je viditeľný z bodu zac_1, ale X_{i+1} ešte áno. Potom oblúk začínajúci v bode zac_1 má $\mathsf{M_v} = (i + 1, 1)$. Za zvláštnu pozornosť stojí len prípad $kon_i = zac_j$, vtedy dostaneme jednobodový „oblúk“ s $\mathsf{M_v} = \mathsf{M_v}(kon_i) = \mathsf{M_v}(zac_j) = (i, j)$. Algoritmicky to znamená, že pri usporiadaní má zac_j prednosť pred kon_i.

V časti b) stačí určiť minimálny počet oblúkov (dvojíc), ktorých množiny viditeľnosti „pokrývajú“ obvod M. Budeme hľadať takýto „minimálny zoznam“ (nemusí byť určený jednoznačne). Vstupom algoritmu bude cyklický zoznam $\mathscr{L}$ usporiadaných dvojíc z časti a); koncové body oblúkov nebudeme potrebovať. Nech $dalsi(d)$ je dvojica nasledujúca za dvojicou $d = (d.i, d.j)$ v zozname $\mathscr{L}$. (V programe sa $\mathscr{L}$ reprezentuje poľom L a $dalsi$ je inkrement indexu modulo $2m$.)

Ku každej dvojici $d \in \mathscr{L}$ určíme dvojicu $najdalej(d)$, pre ktorú platí

$$medzi(d.i, najdalej(d).i, d.j)$$
$$\textbf{and not } medzi(d.i, dalsi(najdalej(d)).i, d.j).$$

Je jasné, že ak sa v minimálnom zozname vyskytne d, tak existuje minimálny zoznam obsahujúci d a $najdalej(d)$ a neobsahujúci dvojice z $\mathscr{L}$ „medzi nimi“ — z dvojíc $dalsi(d)$, $dalsi^2(d)$, $dalsi^3(d)$, ..., $najdalej(d)$ musí byť aspoň jedna v minimálnom zozname; zoznam zostane minimálnym, ak ju nahradíme dvojicou $najdalej(d)$. V prípade $d' = dalsi^k(d)$,

$najdalej(d) = najdalej(d')$ hodnotu $najdalej(d')$ zmeníme na nedefinovanú. (V programe sa $najdalej$ reprezentuje poľom indexov; nedefinovaná hodnota je 0.)

K dokončeniu algoritmu si všimnime, že minimálny zoznam musí obsahovať dvojicu d, ktorá *pokrýva* X_1, t. j. platí $medzi(d.i, 1, d.j)$. (Všetky dvojice pokrývajúce X_1 sú v poli L na začiatku, lebo $obluky[1] = zac_1$.) Pre každé takéto d postupne vypočítame $najdalej(d)$, $najdalej^2(d)$, ..., pokiaľ nenájdeme najmenšie k také, že $najdalej^k(d)$ pokrýva $d.i$. Ak pritom narazíme na nedefinovaný ukazovateľ $najdalej^l(d)$, tak existujú postupnosti $d, najdalej(d), \ldots, najdalej^{l-1}(d)$ a $d', najdalej(d'), \ldots, najdalej^{l-1}(d')$ rovnakej dĺžky bez spoločných prvkov (keby mali spoločný prvok, bolo by $najdalej^{l'}(d) = 0$ už pre $l' < l$), pričom d aj d' pokrývajú X_1, d' je pred d v zozname $\mathscr{L}$ a hodnota $najdalej^l(d)$ je nedefinovaná, lebo by sa mala rovnať hodnote $najdalej^l(d')$. To znamená, že existuje minimálny zoznam neobsahujúci d: podzoznam $d, najdalej(d), \ldots$ nahradíme podzoznamom $d', najdalej(d'), \ldots$, a teda výpočet pre dané d môžeme ukončiť.

Program v PASCALe:

```
program VIDITELNOST;
const m = ...;   dvem = 2 * m;
type
  bod = record x, y : real end;
  dvojica = record i, j : 1 .. m end;
  index = 1 .. m;   index2 = 1 .. dvem;
var
  X : array [index] of bod;
  C : record S : bod; r : real end;
  zac, kon : array [index] of bod;
  L : array [index2] of dvojica;
  obluky : array [index2] of bod;
  najdalej : array [index2] of 0 .. dvem;
  k : index2;
function medzi(i, j, k : index) : boolean;
  begin medzi := (i <= j and j <= k)
          or (k <= i and i <= j) or (j <= k and k <= i) end;
function inc1(i : index) : index; begin inc1 := i mod m + 1; end;
function dec1(i : index) : index;
          begin dec1 := (i + m - 2) mod m + 1; end;
```

```
function dalsi(i : index2) : index2;
            begin dalsi := i mod dvem + 1; end;
procedure p_polpriamka_kruznica(i, j : index; var pries : bod);
  begin   (* určí priesečník X_iX_j ∩ C *) end;
function uhol(U, V : bod) : real;
  begin   (* určí veľkosť uhla USV: číslo z intervalu ⟨0, 2π) *) end;
procedure cast_a;
  var ii, jj : integer;
  begin
    for k := 1 to m do begin
      p_polpriamka_kruznica(k, dec1(k), zac[k]);
      p_polpriamka_kruznica(k, inc1(k), kon[k]);
    end;
    jj := 1;   ii := m;
    repeat ii := dec1(ii)
    until uhol(kon[m], zac[1]) < uhol(kon[m], kon[ii]);
      (* X_ii je posledný vrchol ešte neviditeľný z bodu zac[1] *)
    obluky[1] := zac[1];
    for k := 1 to dvem - 1 do begin
      L[k].i := inc1(ii);   L[k].j := jj;
      if uhol(zac[inc1(jj)], kon[ii]) <= uhol(kon[inc1(ii)], kon[ii])
          then begin jj := inc1(jj);   obluky[k + 1] := zac[jj]; end
          else  begin ii := inc1(ii);   obluky[k + 1] := kon[ii]; end;
    L[dvem].i := inc1(ii);   L[dvem].j := jj;
  end;   (* cast_a *)
procedure cast_b;
  var k, minpocet, pocet : index2;   nd : 0 .. dvem;
  begin
      (* výpočet hodnôt najdalej *)
    nd := 1;
    for k := 1 to dvem do begin
      while medzi(L[k].i, L[dalsi(nd)].i, L[k].j) do nd := dalsi(nd);
      najdalej[k] := nd
    end;
      (* vynulovanie zbytočných hodnôt najdalej *)
    nd := najdalej[dvem];
    for k := dvem downto 2 do
      if (najdalej[k] = najdalej[k - 1]) then najdalej[k] := 0;
```

```
    if (najdalej[1] = nd) then najdalej[1] := 0;
      (* určenie minimálneho počtu bodov *)
    k := 1;   minpocet := dvem;
    while medzi(L[k].i, 1, L[k].j) do begin
      nd := najdalej[k];   pocet := 2;
      while (nd <> 0) and not medzi(L[nd].i, L[k].i, L[nd].j) do
      begin
        nd := najdalej[nd];   pocet := pocet + 1;
      end;
      if (nd <> 0) and (pocet < minpocet) then minpocet := pocet;
      k := k + 1;
    end;
  end;   (* cast_b *)
```

Časová zložitosť je lineárna. Procedúra *časť_a* neobsahuje vnorené cykly. V procedúre *časť_b* sú dva vnorené cykly: v prvom prípade je to **while**-cyklus vnorený do **for**-cyklu. Pritom **while**-cyklus sa vykoná lineárny počet krát, lebo k „prebehne" zoznamom $\mathscr{L}$ práve raz, nd prebieha zoznamom $\mathscr{L}$ v cyklickom poradí $(1, 2, 3, \ldots, 2m, 1, 2, \ldots)$ a nikdy „nepredbehne" k, t. j. prebehne $\mathscr{L}$ nanajvýš dvakrát.

Druhý vnorený cyklus trvá tiež lineárny počet krokov, lebo každým nenulovým smerníkom *najdalej* sa „prejde" nanajvýš raz a nulové sa testujú tiež len raz.

P – III – 2

Pre výstup daného algoritmu platí $f_7 = ac - bd$, $f_8 = ad + bc$, teda komplexné číslo $f_7 + f_8\,\mathrm{i}$ je súčin komplexných čísel: $f_7 + f_8\,\mathrm{i} = (a + b\,\mathrm{i}) * (c + d\,\mathrm{i})$.

Lineárny algoritmus pre vstup a, b, c, d používajúci práve jedno násobenie dokáže spočítať iba polynómy tvaru

$$v_1(t_1a+t_2b+t_3c+t_4d)(u_1a+u_2b+u_3c+u_4d)+v_2a+v_3b+v_4c+v_5d \quad (*)$$

a z riešenia úlohy P–I–3 vieme, že výraz $ac - bd$ nie je tohoto tvaru, t. j. nedá sa spočítať lineárnym algoritmom (multiplikatívnej) zložitosti 1. Celkom analogicky to môžme dokázať aj o výraze $ad + bc$.

Teraz sporom dokážeme, že na súčasný výpočet $ac - bd$, $ad + bc$ potrebujeme aspoň tri násobenia. Keby sa dali spočítať pomocou dvoch násobení, nech p_1 je výsledok prvého násobenia, p_2 výsledok druhého

násobenia (pripúšťame aj možnosť, že pri výpočte p_2 sa použil súčin p_1). Výstupné hodnoty $ac - bd$, $ad + bc$ potom dostaneme pomocou operácií $+$, $-$ z p_1, p_2, a, b, c a d:

$$ac - bd = r_1p_1 + r_2p_2 + r_3a + r_4b + r_5c + r_6d,$$
$$ad + bc = s_1p_1 + s_2p_2 + s_3a + s_4b + s_5c + s_6d,$$

kde koeficienty sú celé čísla, pričom $r_2 \neq 0 \neq s_2$ (lebo na $ac-bd$ i na $ad+bc$ potrebujeme dve násobenia). Odčítaním r_2-násobku druhej rovnice od s_2-násobku prvej dostaneme

$$\begin{aligned} &s_2ac - r_2ad - s_2bd - r_2bc = \\ &\quad = (s_2r_1 - r_2s_1)p_1 + (s_2r_3 - r_2s_3)a + (s_2r_4 - r_2s_4)b + \\ &\qquad\qquad + (s_2r_5 - r_2s_5)c + (s_2r_6 - r_2s_6)d, \end{aligned}$$

vidíme teda, že výraz $s_2ac - r_2ad - s_2bd - r_2bc$ sa dá spočítať pomocou jedného násobenia, a je preto tvaru $(*)$ s $v_1 \neq 0$. Keby sa totiž v_1 rovnalo nule, pre $a = b = c = d$ by sme dostali $-2r_2a^2 = (v_2 + v_3 + v_4 + v_5)a$. Ak $v_2 + v_v + v_4 + v_5 = 0$, dostávame spor dosadením $a := 1$, v opačnom prípade dosadením $a := v_2 + v_3 + v_4 + v_5$.

Porovnaním koeficientov pri a, b, c, d, a^2, b^2, c^2, d^2, ab, ac, ad, bc, bd, cd postupne dostávame

$$v_2 = v_3 = v_4 = v_5 = 0,$$
$$t_1u_2 + t_2u_1 = 0 \qquad (5)$$
$$t_1u_1 = 0, \quad (1) \qquad v_1(t_1u_3 + t_3u_1) = s_2, \qquad (6)$$
$$t_2u_2 = 0, \quad (2) \qquad v_1(t_1u_4 + t_4u_1) = -r_2, \qquad (7)$$
$$t_3u_3 = 0, \quad (3) \qquad v_1(t_2u_3 + t_3u_2) = -r_2, \qquad (8)$$
$$t_4u_4 = 0, \quad (4) \qquad v_1(t_2u_4 + t_4u_2) = -s_2, \qquad (9)$$
$$t_3u_4 + t_4u_3 = 0, \qquad (10)$$

pričom v_1, s_2 a r_2 sa nerovnajú nule. Vzhľadom k symetrii $(*)$ môžeme predpokladať, že $u_1 = 0$ (kvôli (1)) a postupne dostaneme $t_1 \neq 0 \neq u_3$ z (6), $u_2 = 0$ z (5), $u_4 \neq 0$ z (7), $t_3 = t_4 = 0$ z (3) a (4), $t_2 \neq 0$ z (8), t. j. $t_1, t_2, u_3, u_4 \neq 0$, $u_1, u_2, t_3, t_4 = 0$, a teda $s_2 = v_1t_1u_3$, $-r_2 = v_1t_1u_4$, $-r_2 = v_1t_2u_3$, $-s_2 = v_1t_2u_4$ a získavame spor

$$0 > -s_2^2 = (v_1t_1u_3)(v_1t_2u_4) = (v_1t_1u_4)(v_1t_2u_3) = r_2^2 > 0,$$

ktorý dokazuje, že dvojica výrazov $ac - bd$, $ad + bc$ sa nedá spočítať pomocou dvoch násobení.

P – III – 3

Najprv si všimnime, že pri hľadaní optimálnej platby sa stačí obmedziť na také platby, pri ktorých pre každé i je buď $x_i = 0$, alebo $y_i = 0$, lebo nemá cenu, aby platiteľ dával mince, ktoré potom príjemca vráti. *Platbou* teda budeme rozumieť n-ticu $\boldsymbol{U} = (u_1, \ldots, u_n) \in \mathbb{Z}^n$ (kladné u_i znamená, že platiteľ dal u_i mincí hodnoty q_i; záporné u_i znamená, že príjemca vrátil $-u_i$ mincí tejto hodnoty).

Ďalej si uvedomme, že sa stačí zaoberať obnosmi deliteľnými q_1 — iné obnosy danými mincami nevyplatíme.

Označme $t_i = \frac{1}{2}(q_{i+1}/q_i - 1)$, t. j. $q_{i+1}/q_i = 2t_i + 1$, $t_i \geqq 1$ pre $1 \leqq i < n$. Predpokladáme $q_1 \mid X$ a budeme uvažovať platby $\boldsymbol{U}$ s vlastnosťou $\sum_{i=1}^{n} u_i q_i = X$.

Pozorovanie 1. *Pre ľubovoľnú platbu $\boldsymbol{U}$ sa dá zostrojiť platba $\boldsymbol{U}^* = (u_1^*, \ldots, u_n^*)$ s rovnakým zaplateným obnosom, nie vyšším celkovým počtom mincí a s vlastnosťou*

$$|u_i^*| \leqq t_i \qquad \text{pre } 1 \leqq i < n. \tag{$*$}$$

Dôkaz. Ak totiž $\boldsymbol{U}$ nesplňuje podmienku $(*)$, označíme si i_0 najmenšie $i < n$ také, že $|u_i| > t_i$. Existujú (jednoznačne určené) celé čísla p, r s vlastnosťou $u_{i_0} = p(2t_{i_0} + 1) + r$, $|r| \leqq t_{i_0}$, $p \neq 0$. Teraz $p(2t_{i_0} + 1)$ mincí hodnoty q_{i_0} zameníme p mincami hodnoty q_{i_0+1}: položíme $u'_{i_0} = u_{i_0} - p(2t_{i_0} + 1)$, $u'_{i_0+1} = u_{i_0+1} + p$; ostatné zložky vektora $\boldsymbol{U}$ necháme nezmenené. Dostali sme tak novú platbu $\boldsymbol{U}'$ s rovnakým zaplateným obnosom a s $|u'_{i_0}| = |r| \leqq t_{i_0}$, $|u'_{i_0+1}| = |u_{i_0+1} + p| \leqq |u_{i_0+1}| + |p|$. Spočítajme, ako sa zmenil počet použitých mincí!

Ak $|u_{i_0}| \geqq 2t_{i_0} + 1$, počet mincí hodnoty q_{i_0} sa znížil o $|p|(2t_{i_0} + 1)$ a počet mincí hodnoty q_{i_0+1} sa zvýšil nanajvýš o $|p|$, celkový počet sa teda určite znížil.

Ak $t_{i_0} < |u_{i_0}| \leqq 2t_{i_0}$ (v tomto prípade $|p| = 1$), počet mincí hodnoty q_{i_0} sa znížil aspoň o 1 a počet mincí hodnoty q_{i_0+1} sa zvýšil nanajvýš o $|p| = 1$, celkový počet sa teda určite nezvýšil.

Príklad: Pre $q_1 = 1$, $q_2 = 5$ sa platba $(6, -2)$ zmení na $(1, -1)$, $(-4, 2)$ na $(-1, 3)$ a $(3, 1)$ na $(-2, 2)$. Zmena počtu mincí je -6, -2 resp. 0.

Zhrnutie: dostali sme platbu $\boldsymbol{U}'$ s rovnakým obnosom, nie vyšším počtom mincí a s vlastnosťou $|u'_{i_0}| \leqq t_{i_0}$ pre $1 \leqq i \leqq i_0$. Postup môžeme opakovať s platbou $\boldsymbol{U}'$, ..., atď; po nanajvýš $n-1-i_0$ takýchto krokoch dostaneme platbu $\boldsymbol{U}^*$ splňujúcu $(*)$, čo bolo treba ukázať.

Nasledujúce pozorovanie ukazuje, že platba $\boldsymbol{U}$ s vlastnosťou $(*)$ je jednoznačne určená:

Pozorovanie 2. *Nech* $\boldsymbol{U}$, $\boldsymbol{U}'$ *sú platby s vlastnosťou* $(*)$. *Potom* $\boldsymbol{U} = \boldsymbol{U}'$.

Dôkaz. Ak totiž $\boldsymbol{U} \neq \boldsymbol{U}'$, existuje $i_0 < n$ také, že

$$u_i = u'_i \quad (1 \leqq i < i_0), \qquad u_{i_0} \neq u'_{i_0}.$$

(Zrejme nemôže byť $i_0 = n$, lebo potom by obnosy zaplatené $\boldsymbol{U}$ a $\boldsymbol{U}'$ neboli rovnaké.) Potom

$$\sum_{i=i_0}^{n} u_i q_i = \sum_{i=i_0}^{n} u'_i q_i.$$

Obe strany tejto rovnosti vydelíme číslom $q_{i_0+1} = q_{i_0}(2t_{i_0}+1)$. Dostaneme

$$(u_{i_0} - u'_{i_0}) q_{i_0} = k q_{i_0}(2t_{i_0}+1)$$

a vzhľadom k $(*)$

$$|k|(2t_{i_0}+1) = |u_{i_0} - u'_{i_0}| \leqq |u_{i_0}| + |u'_{i_0}| \leqq 2t_{i_0},$$

t.j. $k = 0$ a $u_{i_0} = u'_{i_0}$, čo je spor s definíciou i_0. Tým je pozorovanie dokázané.

Algoritmus je vlastne hotový, ak si všimneme, že obnos X sa dá vyplatiť X/q_1 mincami hodnoty q_1. Vyjdeme z tejto platby $\boldsymbol{U} = (X/q_1, 0, 0, \ldots, 0)$ a zostrojíme k nej platbu $\boldsymbol{U}^*$ s vlastnosťou $(*)$ podľa Pozorovania 1. Na druhej strane určite existuje optimálna platba $\boldsymbol{U}_{\text{opt}}$; k nej môžeme zostrojiť (podľa Pozorovania 1) optimálnu platbu $\boldsymbol{U}^*_{\text{opt}}$ s vlastnosťou $(*)$. Platby $\boldsymbol{U}^*$ a $\boldsymbol{U}^*_{\text{opt}}$ majú vlastnosť $(*)$, a preto podľa Pozorovania 2 sa musia rovnať: $\boldsymbol{U}^* = \boldsymbol{U}^*_{\text{opt}}$. To ale znamená, že platba $\boldsymbol{U}^*$, ktorú sme zostrojili, je optimálna.

Správnosť algoritmu sme už dokázali; nasledujúci program určí platbu $\boldsymbol{U}$ vyhovujúcu vzťahu $(*)$ (ktorá je optimálna). Názvy premenných v programe sa zhodujú s názvami použitými v predchádzajúcich úvahách

až na to, že v programe sa nepracuje s poľom hodnôt $q[1\,..\,n]$. V programe vystupuje len hodnota q_1 ako premenná $q1$, hodnota q_i (pre aktuálne i) ako premenná q a pole $t[1\,..\,n]$ s hodnotami t_i. Hodnota $t[n]$ je ľubovoľná a slúži len na zabránenie behovej chyby pri testovaní podmienky **while**-cyklu pre $i = n$ (keby sa testovala aj druhá časť **and**-podmienky napriek neplatnosti prvej časti). Program netestuje, či vstupné dáta spĺňajú podmienky úlohy (vstupný súbor mincí 1, 4, 8 napr. „transformuje" na 1, 5, 15).

Odhad časovej zložitosti: Pri odhade zložitosti záleží na reprezentácii čísel vystupujúcich v algoritme. Ak $q_1, q_2, \ldots, q_n$ a X sú celé čísla menšie než nejaká konštanta $INTMAX$, tak hodnota n je zhora ohraničená konštantou ($2^{n-1} \leqq q_1(2t_1+1)(2t_2+1)\ldots(2t_{n-1}+1) = q_n \leqq INTMAX$, a teda $n \leqq \log_2 INTMAX + 1$). Vykoná sa najviac n opakovaní **while**-cyklu a každý prechod cyklom trvá konštantný počet krokov (vykonávajú sa len aritmetické operácie). Algoritmus teda trvá konštantný počet krokov a použije pamäť konštantnej veľkosti. (Pri použití aritmetiky veľkých čísel by boli potrebné iné odhady.)

Program v PASCALe:

```
program PLATBA;
  const n = ...;
  var
    t : array [1..n] of integer;
    q1, q, qq : integer;
    U : array [1..n] of integer;
    X : integer;
    i : 1..n;
  begin
    read(q1);
    qq := q1;
    for i := 1 to n - 1 do begin
      read(q);
      t[i] := (q div qq) div 2;
      qq := q;
    end;
    t[n] := 0;
    read(X);
    if (X mod q1 = 0) then begin
      U[1] := X div q1;
```

```
    i := 1;   q := q1;
    while (i < n) and (abs(U[i]) > t[i]) do begin
      U[i + 1] := (abs(U[i]) + t[i]) div (2 * t[i] + 1);
      if U[i] < 0 then U[i + 1] := -U[i + 1];
      U[i] := U[i] - U[i + 1] * (2 * t[i] + 1));
      writeln(U[i], ' mincí hodnoty ', q);
      q := q * (2 * t[i] + 1);
      i := i + 1;
    end;
    writeln(U[i], ' mincí hodnoty ', q);
  end else
    writeln('Obnos ', X, ' Kčs sa nedá vyplatiť.');
end.
```

P – III – 4

1. Spomeňme si na úlohy predchádzajúcich kôl. Našli sme tam čísla 7527522, 572572 (vytvárajúce sami seba) resp. 757227572 a 27572275722 (vytvárajúce sa navzájom). Hneď vidíme, že všetky sú nesmrteľné.

2. Pre každé $n \geqq 1$ nájdeme n čísel $X_1, \ldots, X_n$ takých, že

$$X_i \vdash X_{i+1} \qquad (i = 1, \ldots, n-1),$$
$$X_n \vdash X_1.$$

Položíme $X_1 = 2^{n-1}X2^{n-1}$, ..., $X_i = 2^{n-i}X2^{n-i}$, ..., $X_n = X$ a chceme nájsť také X, že $X = X_n \vdash X_1 = 2^{n-1}X2^{n-1}$. Ale $2^{n-1}X2^{n-1} = 7^{n-1}(2^{n-1}X)$ a stačí použiť Craigov zákon s $M = 7^{n-1}$, $A = 2^{n-1}$: existuje X s vlastnosťou $X \vdash M(AX) = 2^{n-1}X2^{n-1}$.

Riešením je napr. $X = M572AM572 = 7^{n-1}5722^{n-1}7^{n-1}572$.

Pre $n = 1, 2$ sme dostali niektoré riešenia z bodu 1.

3. Neexistuje také H, aby bola splnená ekvivalencia uvedená ako základ algoritmu. To sa nahliadne pomocou McCullochovho zákona: keby také H existovalo, nájdeme X s vlastnosťou $X \vdash HX$. Potom

- ak X je nesmrteľné, HX je tiež nesmrteľné;
- ak X nie je nesmrteľné, HX tiež nie je nesmrteľné.

To je spor s uvedenou ekvivalenciou.