45. ročník matematické olympiády na středních školách

Kategorie P

In: Leo Boček (editor); Karel Horák (editor); Richard Kollár (editor); Václav Sedláček (editor); Jaromír Šimša (editor); Pavel Töpfer (editor): 45. ročník matematické olympiády na středních školách. Zpráva o řešení úloh ze soutěže konané ve školním roce 1995/1996. 37. mezinárodní matematická olympiáda. 8. mezinárodní olympiáda v informatice. (Czech). Praha: Jednota českých matematiků a fyziků, 1997. pp. 81–107.

Persistent URL: http://dml.cz/dmlcz/405000

P – I – 1

Trojúhelníková síť se skládá z rovnostranných trojúhelníků jednotkové velikosti. Soustavu souřadnic si nyní trochu pozměníme: x-ová osa bude orientována stejně, jako jsme zvyklí z kartézské soustavy souřadnic, y-ová osa svírá s x-ovou 60stupňový úhel. Například vrcholy jednoho z jednotkových trojúhelníků mají souřadnice $(0,0)$, $(1,0)$, $(0,1)$.

V této síti zadáme N-úhelník — jeho vrcholy leží ve vrcholech sítě, jsou navzájem různé, přičemž každé dva sousední vrcholy v N-úhelníku mají **jednotkovou vzdálenost**. N-úhelník je zadán na vstupu tak, že první řádek vstupu obsahuje číslo N a dalších N řádků obsahuje souřadnice vrcholů, přičemž tyto vrcholy jsou zadány postupně po obvodu.

Úloha. Napište a odlaďte program, který

a) vykreslí tento N-úhelník na obrazovku,

b) vypíše zprávu o tom, zda je konvexní nebo ne (N-úhelník je konvexní, jestliže úsečky spojující libovolné dva vrcholy leží celé uvnitř, resp. na hranici N-úhelníku.).

Poznámka: Minimalizujte paměťovou a časovou složitost algoritmu. Můžete předpokládat, že vstupní data jsou zadána korektně.

P – I – 2

Pro danou konstantu N (N je prvočíslo, např. 991) máme vyhrazeno N-prvkové celočíselné pole P s indexy 0 až $N-1$. Prvky pole jsou na začátku inicializovány na nulu. Do tohoto pole budeme postupně zařazovat L celočíselných hodnot (navzájem různých a různých od nuly), kde $2 < L < N$, pomocí procedury *Zarad*:

```
procedure Zarad(X:integer);
var i:integer;
begin
```

```
    i := (x div 10 + x mod 10) mod N;
    while P[i] <> 0 do
      i := (i+7) mod N;
    P[i] := x;
  end;
```

Tato procedura nejprve ze zařazované hodnoty X vypočítá předpokládanou hodnotu indexu v poli P, a pokud je tento prvek pole již obsazen (je tam nenulová hodnota), postupně hledá nejbližší volnou pozici v poli (pole je přitom „zacyklené" — za posledním $(N-1)$-tým prvkem následuje nultý). Jestliže je při hledání volného místa nutné postupně prohlížet následující prvky, budeme tuto situaci nazývat **kolize** a budeme počítat počet vzniklých kolizí. Počet kolizí při zařazování j-té hodnoty budeme označovat $K[j]$ ($K[j]$ bude tedy rovno počtu průchodů cyklem **while** při zařazování j-té hodnoty), celkový počet kolizí označíme C_K.

Úloha. Navrhněte algoritmus, který pro daná tři čísla L, G a C_K najde takovou vstupní posloupnost hodnot H (délky L), aby celkový počet kolizí pro tuto posloupnost byl C_K a poslední hodnota byla $H[L] = G$, případně zjistí, že taková posloupnost neexistuje.

P – I – 3

Máme kreslicí zařízení schopné kreslit různé obrázky. Jejich popis však musí být zadán ve speciálním tvaru (jako znakový řetězec):

- je to posloupnost parametrů uzavřená v ‚[' a ‚]' závorkách;
- parametrem je buď číslo nebo opět posloupnost parametrů;
- posloupnost parametrů je vždy sudé délky, přičemž na lichých pozicích je vždy číslo.

Tuto posloupnost bude kreslicí zařízení interpretovat následovně:

- je-li posloupnost prázdná, nekreslí se nic;
- je-li neprázdná, zřejmě první parametr (P_1) je číslo a druhý (P_2) je buď číslo, nebo opět posloupnost parametrů:
  - pokud je P_2 číslo, kreslicí pero přejde (se spuštěným perem) v momentálním směru natočení vzdálenost P_1 a potom se otočí o úhel P_2 vpravo (úhel je zadán ve stupních);
  - pokud je P_2 posloupnost, kreslicí zařízení P_1-krát zopakuje posloupnost P_2.

Kreslicí pero je stále natočeno nějakým směrem (na začátku algoritmu na sever) a podle parametrů posloupnosti buď mění toto natočení, nebo

se v aktuálním směru pohybuje dopředu nebo dozadu (kreslí čáru) podle toho, zda je tato vzdálenost kladná nebo záporná.

Například posloupnost `'[4[100 90]]'` nakreslí čtverec se stranou 100. (Všimněte si, že číselné parametry jsou oddělené mezerou.)

Úloha. Napište a odlaďte program, který přečte vstupní posloupnost zadanou ve tvaru znakového řetězce a zinterpretuje ji na grafické ploše obrazovky. Můžete předpokládat, že vstupní posloupnost je zadána korektně, obsahuje jen celočíselné hodnoty a že řetězec není delší než 255 znaků.

P – I – 4

D0L systémy

Abecedou nazýváme libovolnou konečnou neprázdnou množinu. Prvky této množiny nazýváme **znaky**. Konečnou posloupnost znaků z nějaké abecedy nazýváme **slovo**. Znaky označujeme zpravidla písmeny ze začátku abecedy $(a, b, c, \ldots)$, slova písmeny z konce abecedy $(u, v, w, \ldots)$ a abecedy velkými řeckými písmeny $(\Sigma, \Gamma, \ldots)$.

Délkou slova w rozumíme počet znaků, z nichž se slovo skládá, označujeme ji $|w|$. **Zřetězením slov** $v = a_1 a_2 \ldots a_n$ a $u = b_1 b_2 \ldots b_m$ rozumíme slovo $v * u = a_1 a_2 \ldots a_n b_1 b_2 \ldots b_m$. Množinu všech slov, která se dají vytvořit ze znaků abecedy Σ, označujeme Σ^*. Tato množina obsahuje také **prázdné slovo**, tj. slovo nulové délky, které označujeme ε.

Pravidlem nad abecedou Σ nazýváme uspořádanou dvojici (a, v), kde $a \in \Sigma$ a $v \in \Sigma^*$. Pravidlo zapisujeme ve tvaru $a \to v$.

Deterministický Lindermayerův systém bez interakce (**D0L systém**) je uspořádaná trojice (Σ, P, w), kde

- Σ je abeceda,
- P je množina pravidel, která pro každý znak $a \in \Sigma$ obsahuje právě jedno pravidlo nad abecedou Σ tvaru $a \to u$,
- w je slovo ze Σ, které nazýváme **axiom**.

Takovýto D0L systém produkuje posloupnost slov $w_1, w_2, w_3, \ldots$, která začíná axiomem a pokračuje vždy slovem, jenž dostaneme z předcházejícího slova současným nahrazením všech znaků za slova podle pravidel. Tedy

1. $w_1 = w$,
2. jestliže $w_i = a_1 a_2 \ldots a_n$, potom $w_{i+1} = u_1 * u_2 * \ldots * u_n$, kde $a_j \to u_j \in P$ pro $j = 1, 2, \ldots, n$.

Uvědomte si, že pro každý znak máme právě jedno pravidlo, a tedy právě jedno slovo, kterým budeme tento znak nahrazovat. Posloupnost produkovaná D0L systémem je proto jednoznačně určena.

Růstovou funkcí D0L systému nazýváme takovou funkci $f: \mathbb{N} \to \mathbb{N}_0$ ($\mathbb{N} = \{1, 2, \ldots\}$, $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$), která pro pořadí slova v posloupnosti produkované D0L systémem udává délku tohoto slova. Přesněji $f(i) = |w_i|$.

Příklad:

- $\{a, b\}$ je abeceda obsahující dva znaky: a a b.
- Délka slova $aabab$ je 5.
- Zřetězením slov ab a aab je slovo $ab * aab = abaab$.
- Nad touto abecedou můžeme vytvořit nekonečně mnoho slov: ε, a, b, aa, ab, bb, aaa, ...
- Pravidlem je například $a \to aa$.
- D0L systémem je například $(\{a, b\}, \{a \to aa, b \to ab\}, ab)$.
- Posloupnost produkovaná tímto systémem je

$$ab, aaab, aaaaaaab, aaaaaaaaaaaaaaab, \ldots$$

(Když slovo obsahuje více stejných písmen za sebou, můžeme nahradit tato písmena jedním písmenem, u kterého uvedeme počet písmen, která zastupuje. Zkráceně můžeme tedy psát posloupnost tohoto D0L systému: ab, a^3b, a^7b, $a^{15}b$, ...).

- Růstová funkce tohoto D0L systému je $f(n) = 2^n$.

Poznámka: Při konstrukci D0L systému dáváme přednost takovým systémům, které mají co nejmenší počet pravidel.

Úloha. Vytvořte D0L systém, jehož růstová funkce je

a) $f(n) = 4$,
b) $f(n) = n$,
c) $f(n) = 3n + 2$,
d) $f(n) = n^2$,
e) $f(n) = kn^3$, kde k je libovolné přirozené číslo.

P – II – 1

Učební text čtěte v úloze P–I–1.

Úloha. Napište program, který spočítá obsah zadaného N-úhelníku a vyjádří ho počtem jednotkových trojúhelníků.

Poznámka: Minimalizujte paměťovou a časovou složitost algoritmu. Můžete předpokládat, že vstupní data jsou zadána korektně.

P – II – 2

Pro danou konstantu N (N je prvočíslo, např. 991) máme vyhrazeno N-prvkové celočíselné pole P s indexy 0 až $N-1$. V tomto poli máme uloženo M různých kladných čísel ($M < N$). Neobsazené prvky pole jsou inicializovány na nulu.

Prvky jsou v poli P uloženy tak, aby bylo možné použít funkci *Vyhledej* na zjištění, kde se v poli P nachází prvek X (funkce vrátí polohu prvku X v poli P, pokud se X v poli nachází, v opačném případě vrátí hodnotu -1):

```
function Vyhledej(X:integer):integer;
var i:integer;
begin
  i:=(X div 10 + X mod 10) mod N;
  while P[i]>X do
    i:=(i+7) mod N;
  if P[i]=X then Vyhledej:=i
           else Vyhledej:=-1;
end;
```

```
int vyhledej(int X) {
  int i;
  i=(X/10 + X  while(P[i]>X) i=(i+7) if(P[i]==X) return(i);
  else return(-1);  }
```

Úloha.

a) Napište, jak musí být prvky uloženy v poli P, aby bylo možné k jejich vyhledávání použít funkci *Vyhledej*.

b) Napište co nejefektivnější proceduru *Zarad*, pomocí níž bude možné zařadit prvek X do pole P tak, aby mohla být k vyhledávání použita funkce *Vyhledej*. Předpokládejte, že před použitím procedury *Zarad* byly prvky v poli P takto uspořádány a že prvek X se v poli P dosud nenachází.

P – II – 3

Mějme kreslicí zařízení popsané v úloze P–I–3.

Úloha. Nalezněte a dokažte algoritmus, který pro danou vstupní posloupnost (ve tvaru znakového řetězce) určí, jak nejdále může pero star-

šího kreslicího zařízení skončit od místa, kde mělo skončit podle zadané posloupnosti.

P – II – 4

Vytvořte D0L systém (nebo dokažte, že to není možné), jehož růstová funkce je

a) $f(n) = n^n$,

b) $f(n) = n^2 \cdot 2^n$.

Studijní text o D0L systémech najdete u zadání úlohy P–I–4.

P – III – 1

Učební text čtěte v úloze P–I–1.

Úloha. Na vstupu je zadán N-úhelník a M-úhelník. Napište program, který spočítá obsah jejich průniku a vyjádří ho počtem jednotkových trojúhelníků.

Poznámka: Minimalizujte paměťovou a časovou složitost algoritmu. Můžete předpokládat, že vstupní data jsou zadána korektně.

P – III – 2

Pro danou konstantu N (N je prvočíslo větší než 10, např. 991) máme vyhrazeno N-prvkové celočíselné pole P s indexy 0 až $N-1$. Prvky pole jsou na začátku výpočtu inicializovány na nulu. Do tohoto pole budeme postupně zařazovat různá kladná celá čísla pomocí procedury *Zarad*:

```
procedure Zarad(x:integer);
var i:integer;
begin
  i := x mod N;
  while P[i] <> 0 do
    if P[i]>x then
      i := (i+7) mod N
    else
      i := (i+3) mod N;
  P[i] := x;
end;
void Zarad(int x) {
  int i;
```

```
    i = x
      while (P[i]!=0)
      if (P[i]>x) i = (i+7)
      else i = (i+3)
    P[i] = x;
}
```

Úloha.

a) Napište co nejefektivnější funkci *Vyhledej* s jedním celočíselným parametrem x, která zjistí, zda se prvek x nachází v poli P. Pokud ano, funkce vrátí hodnotu indexu prvku x v poli P. Jestliže číslo x v poli P není, funkce vrátí hodnotu -1.

b) Rozhodněte, zda je výpočet procedury *Zarad* v případě, že je v poli P alespoň jedno volné místo (tj. aspoň jeden prvek pole P má hodnotu 0), vždy konečný. Odpověď dokažte.

P – III – 3

a) Sestrojte D0L systém s nejvýše dvěma pravidly, jehož růstová funkce je $f(n) = n^2$, nebo dokažte, že takový systém neexistuje.

b) Sestrojte D0L systém s růstovou funkcí $f(n) = \lfloor \log_{k+1}(n+1) \rfloor + +1$, kde k je počet symbolů abecedy, nebo dokažte, že takový systém neexistuje.

Poznámka: Zápisem $\lfloor X \rfloor$ rozumíme dolní celou část z hodnoty výrazu X, tzn. hodnotu výrazu X zaokrouhlenou dolů na nejbližší celé číslo. Například $\lfloor 4{,}789 \rfloor = 4$, $\lfloor 5 \rfloor = 5$.

Studijní text o D0L systémech najdete u zadání úlohy P–I–4.

P – III – 4

Program: `LAMPY.PAS / LAMPY.CPP`
Vstup: `LAMPY.IN`
Výstup: `LAMPY.OUT`

Veřejná prostranství ve městě jsou osvětlena N pouličními lampami. Každá z lamp má jednoznačně přiřazeno číslo od 1 do N. K zapínání a vypínání veřejného osvětlení slouží v řídicím středisku M přepínačů. Každý z přepínačů přepne najednou několik lamp. Přepnout lampu znamená zapnout ji, pokud zrovna nesvítí, a vypnout ji, jestliže momentálně svítí. Přepínač číslo i přepíná všechny lampy s čísly od a_i do b_i,

tzn. lampy s čísly ležícími v uvedeném intervalu. Můžete předpokládat, že $0 < M < 100$, $0 < N < 100$.

Úloha. Napište program, který zjistí, zda je možné zapnout pomocí přepínačů v řídicím středisku všechny lampy, jsou-li na začátku všechny lampy vypnuté.

Vstupní soubor: Vstupní soubor obsahuje několik zadání. První řádek vstupního souboru je tvořen jediným číslem, které udává počet zadání v souboru. Pro každé zadání obsahuje vstupní soubor blok údajů v tomto tvaru: Na prvním řádku bloku jsou uvedeňa čísla N a M, kde N je počet lamp ve městě a M je počet přepínačů v řídicím středisku. Dalších M řádků obsahuje pro jednotlivé přepínače vždy dvojici čísel a_i, b_i (pro každý přepínač interval čísel lamp, které se pomocí něho přepnou). Jednotlivé bloky údajů jsou ve vstupním souboru odděleny vždy jedním prázdným řádkem.

Výstupní soubor: Pro každé zadání obsažené ve vstupním souboru obsahuje výstupní soubor jeden řádek s jednou z následujících zpráv:

„Lze“ — pokud je možné rozsvítit všechny lampy pomocí přepínačů v řídicím středisku,

„Nelze“ — jestliže to není možné.

Příklad:

Soubor `LAMPY.IN`
2
5 3
1 3
2 5
2 3
10 5
1 9
2 10
3 10
4 10
5 10

Soubor `LAMPY.OUT`
Lze
Nelze

P – III – 5

Program: `MANHATAN.PAS / MANHATAN.CPP`
Vstup: `MANHATAN.IN`
Výstup: `MANHATAN.OUT`

Ve městě Manhatan vedou všechny ulice buď ze severu na jih, nebo ze západu na východ. Předpokládejte, že se ulice táhnou oběma směry dostatečně daleko. V průsečíku každých dvou ulic různých směrů je křižovatka.

Křižovatku označíme dvojicí čísel (i, j), jestliže se jedná o křižovatku v pořadí i-té západovýchodní ulice počítáno ze severu a j-té severojižní ulice počítáno od západu. Křižovatky jsou tedy očíslovány od $(1, 1)$ do (M, N), kde M je počet západovýchodních ulic a N je počet severojižních ulic. Můžete předpokládat, že $0 < M < 100$, $0 < N < 100$.

Na některých křižovatkách se pracuje na opravě vozovky, a proto přes ně není možné přejet. Na křižovatkách $(1, 1)$ a (M, N) se nepracuje.

Arpád vyjíždí každé ráno z křižovatky $(1, 1)$ a potřebuje se dostat do firmy, která sídlí na opačném konci města, tzn. na křižovatce (M, N).

Úloha. Napište program, který zjistí, kolika různými cestami se může Arpád dostat do své firmy, jestliže pojede vždy jen ve směru na jih nebo na východ. Program dále vypíše nejmenší počet křižovatek, přes které se může takovouto cestou do firmy dostat (včetně křižovatek $(1, 1)$ a (M, N)).

Vstupní soubor: Vstupní soubor obsahuje několik zadání. První řádek vstupního souboru je tvořen jediným číslem, které udává počet zadání v souboru. Pro každé zadání obsahuje vstupní soubor blok dat. Na prvním řádku každého bloku jsou uvedena čísla M a N, kde M je počet východozápadních ulic a N je počet severojižních ulic. Na dalším řádku se nachází číslo K — počet křižovatek ve městě, na nichž se opravuje vozovka. Na každém z následujících K řádků jsou uvedena vždy dvě čísla určující polohu křižovatky, kde se pracuje. Jednotlivé bloky vstupních údajů jsou odděleny vždy jedním prázdným řádkem.

Výstupní soubor: Pro každé zadání ve vstupním souboru obsahuje výstupní soubor jeden řádek. Na něm jsou

- buď dvě čísla A a B, kde A je počet možných různých cest a B je počet křižovatek na nejkratší z nich (včetně křižovatek $(1, 1)$ a (M, N)), a to v případě, že existuje aspoň jedna cesta požadovaného typu,
- nebo zpráva „Cesta neexistuje“, jestliže cesta požadovaného typu neexistuje.

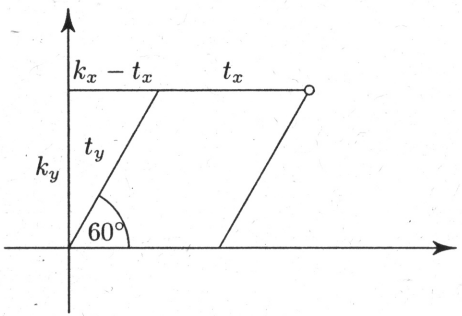
Příklad:

Soubor MANHATAN.IN
2 4 3 2 2 2 3 2 2 4 2 1 2 2 3

Soubor MANHATAN.OUT
2 6 Cesta neexistuje

P – I – 1

a) Nejprve si ukážeme, jak lze přepočítat zadané souřadnice do pravoúhlé souřadnicové soustavy. Označme (t_x, t_y) souřadnice v naší trojúhelníkové soustavě a (k_x, k_y) souřadnice v pravoúhlé souřadnicové soustavě.



Obr. 29

Z vlastností pravoúhlého trojúhelníku na obr. 29 vyplývá

$$\frac{k_y}{t_y} = \cos 30°, \quad \frac{k_x - t_x}{t_y} = \sin 30°,$$

a tedy pro souřadnice (k_x, k_y) platí:

$$k_y = \frac{1}{2}\sqrt{3}t_y, \quad k_x = \frac{1}{2}t_y + t_x$$

Pro účely přepočítání na souřadnice na obrazovce zavedeme následující proměnné:

- zvětšení k (počet pixelů na obrazovce na jednotku délky)
- souřadnice počátku souřadnicové soustavy na obrazovce (o_x, o_y)

Ve vzorovém řešení je k konstanta a o_x, o_y se počítají tak, aby bod $(0, 0)$ ležel ve středu obrazovky.

Shrneme výsledky předcházejících úvah. Bod zadaný v pravoúhlé soustavě souřadnicemi (k_x, k_y) se zobrazí na obrazovce na souřadnice $(o_x + k \cdot k_x, o_y - k \cdot k_y)$. Je-li bod zadán v trojúhelníkové soustavě souřadnicemi (t_x, t_y), potom jeho souřadnice v pravoúhlé soustavě jsou $k_y = \frac{1}{2}\sqrt{3}t_y$, $k_x = \frac{1}{2}t_y + t_x$ a na obrazovce se tedy zobrazí na souřadnice $\left(o_x + k \cdot t_x + k\frac{1}{2}t_y, o_y - k\frac{1}{2}\sqrt{3}t_y\right)$. Jestliže nyní označíme $j_{xx} := k$,

$j_{yx} := \frac{1}{2}k$, $j_{yy} := -\frac{1}{2}\sqrt{3}k$ dostáváme, že se bod se souřadnicemi (t_x, t_y) v trojúhelníkové soustavě zobrazí na obrazovce do bodu

$$(o_x + j_{xx}t_x + j_{yx}t_y, o_y + j_{yy}t_y).$$

b) Jelikož všechny vrcholy našeho N-úhelníku musí ležet v mřížových bodech a vzdálenost následujícího vrcholu musí být vždy 1 od předcházejícího, přicházejí pro vrchol se souřadnicemi (t_x, t_y) v úvahu tyto vrcholy jako následující (odpovídající směry, v nichž vrcholy leží vzhledem k (t_x, t_y) označme od 0 po 5 v pořadí, v jakém jsou zde vypsány): (t_x, t_y+1), (t_x+1, t_y), (t_x+1, t_y-1), (t_x, t_y-1), (t_x-1, t_y), (t_x-1, t_y+1).

Nechť jsme nyní do vrcholu (t_x, t_y) přišli ze směru s_s a odcházíme z něho ve směru s_n. Vidíme, že pokud $s_n = s_s$, potom jsme se v bodě neotočili, pokud $s_n = (s_s + 1) \bmod 6$ nebo $s_n = (s_s + 2) \bmod 6$ potom jsme se otočili doprava, jinak doleva (případ $s_n = (s_s + 3) \bmod 6$ nemůže nastat).

Budeme obcházet náš N-úhelník po obvodu (tzn. v pořadí, v jakém byly zadány jeho vrcholy — připomeňme si, že pořadí bodů může být zadáno buď ve směru, nebo proti směru hodinových ručiček) a budeme sledovat, v jakém směru se otáčíme v jednotlivých vrcholech (směr otáčení je dán menším z dvou úhlů u vrcholu). Je-li náš N-úhelník konvexní, potom se zřejmě musíme otáčet stále stejným směrem a naopak, jestliže konvexní není, musíme se během obcházení otočit aspoň jednou doprava a aspoň jednou doleva.

Tato jednoduchá úvaha bude základem našeho algoritmu. Obcházíme postupně N-úhelník, přičemž když se otočíme doprava, nastavíme proměnnou *doprava*, když se otočíme doleva, nastavíme proměnnou *doleva*. Jestliže jsou na konci obě proměnné nastavené, N-úhelník není konvexní, v opačném případě je konvexní.

Časová i paměťová složitost tohoto algoritmu je lineární ($O(N)$), správnost vyplývá z výše uvedené úvahy.

P – I – 2

Má-li existovat posloupnost hodnot $H[1], \ldots, H[L]$, musí především platit, že $L < N$. Dále platí, že když do pole zařazujeme i-tou hodnotu, počet kolizí $K[i]$ je nejvýše $i - 1$. ($K[i] = i - 1$ tehdy, pokud dojde ke kolizi se všemi už zařazenými prvky). Z toho ale vyplývá, že jestliže $C_K > 0 + 1 + \cdots + (L - 1) = \frac{1}{2}L(L - 1)$, posloupnost opět neexistuje.

Označme $a \oplus b := (a + 7b) \bmod N$. Vezmeme nyní libovolný index nějakého prvku v našem poli j. Potom jestliže procházíme prvky $j \oplus 1, j \oplus 2, \ldots, j \oplus N$, přejdeme každým prvkem našeho pole právě jednou (protože N je prvočíslo).

Máme-li v poli obsazeno k prvků s indexy j_0, $j_0 \oplus 1$, $j_0 \oplus 2$, ..., $j_0 \oplus (k-1)$, pak pro libovolné $K[k+1]$, kde $K[k+1] \in \{0, 1, \ldots, k\}$, umíme nalézt vhodný prvek $H[k+1]$ takový, že se zařadí do pole na index $j_0 \oplus k$. Označme[1] $i_{k+1} := (H[k+1] \operatorname{div} 10 + H[k+1] \bmod 10) \bmod N$ první index vypočítaný pro prvek $H[k+1]$ v proceduře *Zarad*. Potom když vezmeme

$$i_{k+1} = j_0 \oplus (k - K[k+1]),$$

tak se nám prvek $H[k+1]$ zařadí na pozici $j_0 \oplus k$.

Pro libovolný počáteční index j_0 a posloupnost K ($K[i] \leqq i-1$ pro $i = 1, 2, \ldots, L$) tedy umíme tímto způsobem nalézt příslušnou posloupnost prvních indexů $j_0 = i_1, i_2, \ldots, i_L$.

Posloupnost K však není zadána, známe jen její součet C_K. Proto si ji můžeme zvolit libovolně, pokud možno co nejjednodušším způsobem. Položme například $K[i] := i - 1$ pro $i = 1, 2, \ldots, m$ a jestliže $m \neq L$ $K[m+1] := C_K - \frac{1}{2}m(m-1)$ a $K[i] := 0$ pro $i = m+2, \ldots, L$, přičemž m vezmeme největší takové, že $\frac{1}{2}m(m-1) \leqq C_K$. Řešením kvadratické rovnice lehce zjistíme, že $m = \left\lfloor \frac{1}{2}\left(1 + \sqrt{1 + 8C_K}\right) \right\rfloor$.

Dále je třeba vyřešit problém, jak zvolit j_0 tak, aby se při výše popsané volbě posloupnosti K první index posledního prvku posloupnosti H nalezený pomocí vztahu $i_L = j_0 \oplus (L - 1 - K[L])$ skutečně rovnal hodnotě vypočítané v proceduře *Zarad* $((H[L] \operatorname{div} 10 + H[L] \bmod 10) \bmod N)$. Pro operaci $\oplus$ platí $a \bmod N = (a \oplus b) \oplus (-b)$, a tedy pokud $i_L = j_0 \oplus (L - 1 - K[L])$, potom

$$j_0 = i_0 \oplus (-(L - 1 - K[L])).$$

Zbývá určit, jak z hodnoty prvního indexu i_j vypočítat hodnotu prvku posloupnosti $H[j]$. Je zřejmé, že bude-li mít prvek hodnotu $10i_j$, určitě bude jeho index i_j. Prvky posloupnosti H však mají být nenulové a navzájem různé. Proto k j-tému prvku ještě připočítáme číslo jN. Kdyby se takto vypočítaná hodnota náhodou rovnala číslu G, odečteme od něj

[1] Viz proceduru *Zarad*.

číslo 9 (tím bude $H[j] \bmod 10 = 1$, ale hodnota $H[j] \operatorname{div} 10$ klesne o 1, takže součet se zachová).

Shrňme předcházející úvahy do funkce *Prvek* (i, j), která pro daný první index i a pořadové číslo j určí hodnotu $H[j]$

Funkce *Prvek* (i, j)
- Jestliže $j = L$ vrať G, jinak
- jestliže $10i + jN = G$ vrať $10i + jN - 9$
- jinak vrať $10i + jN$

Na závěr uveďme přehledný zápis celého algoritmu tak, jak byl popsán výše:

1. Jestliže $L > N$ nebo $C_K > \frac{1}{2}L(L-1)$, posloupnost neexistuje, jinak pokračuj bodem 2.
2. $m := \lfloor \frac{1}{2}(1 + \sqrt{1 + 8C_K}) \rfloor$, podle hodnoty m urči $K[L]$:
   - pokud $m = L, K[L] := L - 1$
   - pokud $m + 1 = L, K[L] := C_K - \frac{1}{2}m(m-1)$
   - pokud $m + 1 < L, K[L] := 0$

   a $j_0 := i_0 \oplus (-(L - 1 - K[L]))$.
3. $K[i] :=$ *Prvek* $(0, i)$ pro $i = 1, 2, \ldots, m$
4. jestliže $m + 1 < L$, potom $K[m+1] :=$ *Prvek* $(m - C_K + \frac{1}{2}m(m-1), m+1)$
5. $K[i] :=$ *Prvek* $(j_0 \oplus (i-1), i)$ pro $i = m+2, \ldots, L$

Časová složitost algoritmu je lineární ($O(N)$), paměťová konstantní ($O(1)$), neboť hodnoty $K[i]$ můžeme přímo vypisovat.

P – I – 3

Řešení nepředstavuje prakticky žádné vážnější problémy.

P – I – 4

a) Řešením je D0L systém $(\{a\}, \{a \to a\}, a^4)$.

Tvrzení: V n-tém kroku je tvar slova a^4, a tedy $f(n) = 4$.

Důkaz: Pro $n = 1$ tvrzení zřejmě platí.

Nechť dále $w_n = a^4$. Potom se uplatněním pravidla změní každé písmeno a na a, a tedy $w_{n+1} = a^4$, čímž je tvrzení dokázáno.

b) Řešením je D0L systém $(\{a, b\}, \{a \to a, b \to ab\}, b)$.

Tvrzení: V n-tém kroku je tvar slova $a^{n-1}b$, a tedy $f(n) = n$

Důkaz: Pro $n = 1$ tvrzení zřejmě platí.

Nechť dále $w_n = a^{n-1}b$. Potom se uplatněním pravidel změní každé písmeno a na a a písmeno b na ab, a tedy $w_{n+1} = a^{n-1}ab = a^nb$, čímž je tvrzení dokázáno.

c) Řešením je D0L systém $(\{a,b\}, \{a \to a, b \to a^3b\}, a^4b)$.

Tvrzení: V n-tém kroku je tvar slova $a^{3n+1}b$, a tedy $f(n) = 3n + 2$.

DŮKAZ: Pro $n = 1$ tvrzení zřejmě platí.

Nechť dále $w_n = a^{3n+1}b$. Potom se uplatněním pravidel změní každé písmeno a na a a písmeno b na a^3b, a tedy $w_{n+1} = a^{3n+1}a^3b = a^{3(n+1)+1}b$, čímž je tvrzení dokázáno.

d) Řešením je D0L systém $(\{b,c,d\}, \{b \to b, c \to bc, d \to bc^2d\}, d)$.

Tvrzení: V n-tém kroku je tvar slova $b^{(n-1)^2}c^{2(n-1)}d$, a tedy $f(n) = $ $= (n-1)^2 + 2(n-1) + 1 = n^2$.

DŮKAZ: Pro $n = 1$ tvrzení zřejmě platí.

Nechť dále $w_n = b^{(n-1)^2}c^{2(n-1)}d$. Potom po uplatnění pravidel bude $w_{n+1} = b^{(n-1)^2}b^{2(n-1)}c^{2(n-1)}bc^2d = b^{n^2}c^{2n}d$, čímž je tvrzení dokázáno.

Poznámka. Ukážeme ještě postup, jak je možné tento D0L systém sestrojit. Platí:

$$(n+1)^2 - n^2 = 2n + 1.$$

To znamená, že při přechodu od n-tého k $(n+1)$-mu slovu musí „přibýt" $2n+1$ písmenek. Kdybychom tedy našli systém, který má růstovou funkci $f(n) = 2n + 1$, potom by stačilo, aby každé jeho písmenko v každém kroku vygenerovalo nějaké „neutrální" písmenko (tj. písmenko, které se v dalších krocích mění už jen samo na sebe). Takový stroj ale lehce sestrojíme: $(\{c,d\}, \{c \to c, d \to c^2d\}, d)$ — důkaz tu nebudeme dělat, neboť je naprosto analogický s důkazy předcházejících tvrzení. Potom náš stroj bude vypadat takto: $(\{b,c,d\}, \{b \to b, c \to bc, d \to bc^2d\}, d)$.

Tento postup je možné rovněž použít jako důkaz správnosti, v mnoha případech je však jednodušší vzniklý D0L systém dokázat indukcí (viz výše). Ve skutečnosti ke konstrukci D0L systému v případě e) byl použit obdobný postup.

e) Řešením je D0L systém $(\{b,c,d,e,f\}, \{b \to b, c \to bde^5f, d \to$ $\to bd, e \to bde, f \to bde^6f\}, c^k)$.

Tvrzení: Pro $n \geqq 2$ je tvar slova $b^{k(n-1)^3}d^{k(3n^2-9n+7)}e^{k(6n-7)}f^k$. Jelikož $f(1) = k$, je $f(n) = kn^3$ $(k((n-1)^3 + 3n^2 - 9n + 7 + 6n - 7 + 1) = kn^3$ pro $n \geqq 2)$.

DŮKAZ: Pro $n = 1$ a $n = 2$ tvrzení zřejmě platí.

Nechť tvrzení platí pro nějaké $n \geqq 2$. Potom

$$w_n = b^{k(n-1)^3}d^{k(3n^2-9n+7)}e^{k(6n-7)}f^k$$

a po uplatnění pravidel

$$\begin{aligned} w_{n+1} &= b^{k(n-1)^3} b^{k(3n^2-9n+7)} d^{k(3n^2-9n+7)} b^{k(6n-7)} \\ &\quad d^{k(6n-7)} e^{k(6n-7)} b^k d^k e^{6k} f^k = \\ &= b^{kn^3} d^{k(3n^2-3n+1)} e^{k(6n-1)} f^k = \\ &= b^{kn^3} d^{k(3(n+1)^2-9(n+1)+7)} e^{k(6(n+1)-7)} f^k, \end{aligned}$$

čímž je tvrzení dokázáno.

P – II – 1

Uvažujme v naší soustavě souřadnic pás trojúhelníků, který je vymezen x-ovými souřadnicemi x a $x+1$. Vezmeme průnik tohoto pásu s hranicí našeho n-úhelníku. Každé hraně n-úhelníku uvnitř pásu přiřadíme směr podle toho, v jakém pořadí byly zadány její krajní body. Dostaneme tak soustavu hran jednotkové délky, přičemž

- Počet hran je sudý (pokud projdeme přes tento pás doprava, potom abychom n-úhelník uzavřeli, musíme přes tento pás přejít také doleva a naopak).
- Je-li směr některé hrany doleva, směr nejbližší nižší a vyšší hrany je doprava a naopak.
- Nejvyšší a nejnižší hrana vede ve všech pásech (tzn. pro všechna x) stejným směrem, přičemž nejvyšší hrana vede opačným směrem než nejnižší hrana.

Plocha průniku pásu a našeho n-úhelníku je zřejmě rovna součtu ploch ohraničených nejvyšší hranou a druhou nejvyšší hranou, třetí a čtvrtou nejvyšší hranou, ..., druhou a první nejnižší hranou. Předpokládejme nyní, že nejnižší hrana vede doleva. Potom plochu průniku n-úhelníku a našeho pásu spočítáme tak, že plochu v pásu „pod" hranou vedoucí doprava vždy přičítáme a plochu „pod" hranou vedoucí doleva odčítáme.

Toto však nemusíme provádět po jednotlivých pásech. Nejprve položíme *plocha* := 0. Nechť y je výška aktuálního bodu n-úhelníku. Jestliže další bod leží napravo dole (relativní souřadnice $(1,-1)$), připočítáme k ploše $2y-1$, pokud je napravo (relativní souřadnice (1,0)), přičteme $2y$. Jestliže je další bod nalevo nahoru (relativní souřadnice $(-1,1)$), odečteme od plochy $2y+1$, pokud je nalevo (relativní souřadnice $(-1,0)$), odečteme $2y$. V ostatních případech (relativní souřadnice (0,1) a $(0,-1)$) nepřipočítáváme nic, neboť tyto hrany neprotínají žádný pás.

Co se stane, jestliže byla orientace hran opačná, než jsme předpokládali? Potom dostaneme záporné číslo a jeho absolutní hodnota je rovna ploše n-úhelníku.

Poznámka: V předcházející úvaze jsme předpokládali, že se celý n-úhelník nachází jen nad osou x. Je však jasné, že algoritmus zůstává beze změny, i když bude n-úhelník ležet celý nebo částečně pod osou x.

Správnost algoritmu je zřejmá z předcházejících úvah. Časová složitost algoritmu je $O(n)$, paměťová $O(1)$.

P – II – 2

a) Označme $h(x) := (x \bmod 10 + x \operatorname{div} 10) \bmod N$. Funkce *Vyhledej* při hledání prvku x postupně prohlíží prvky pole P s indexy $h(x), (h(x)+7) \bmod N, (h(x)+14) \bmod N, \ldots, (h(x)+7l) \bmod N$. Skončí, když nastane jedna z těchto tří možností:

- najde hledaný prvek x
- najde prvek menší než x
- najde prázdné políčko (prvek s hodnotou 0).

Pokud nastala první možnost, prvek se v poli nachází, jinak v poli není. N a 7 jsou nesoudělná čísla. Z toho vyplývá, že pro každé $i \in \{0,1,\ldots,N-1\}$ existuje $j \in \{0,1,\ldots,N-1\}$ takové, že $i = (h(x)+7j) \bmod N$. V poli je aspoň jedno prázdné políčko ($M < N$), a proto l bude vždy menší než N.

Nechť se v poli P prvek x nachází na místě s indexem i. Nechť $j \in \{0,1,\ldots,N-1\}$ je takové číslo, že $i = (h(x)+7j) \bmod N$. Aby funkce *Vyhledej* prvek x našla, musí platit, že prvky s indexy $h(x), (h(x)+7) \bmod N, (h(x)+14) \bmod N, \ldots, (h(x)+7(j-1)) \bmod N$ budou větší než x.

b) Pokud do tabulky chceme uložit prvek x, postupujeme opět po posloupnosti indexů $h(x), (h(x)+7) \bmod N, (h(x)+14) \bmod N, \ldots, (h(x)+7l) \bmod N$ tak dlouho, až najdeme prvek menší než x nebo prázdné políčko. Jestliže jsme našli prázdné políčko, prvek x tam můžeme uložit a funkce *Vyhledej* ho jistě najde, neboť před ním bude prohledávat jen větší prvky. Pokud však je na tomto místě prvek s hodnotou y_1 ($y_1 < x$), uložíme sem prvek x. Při hledání prvku y_1 se předtím funkce *Vyhledej* zastavila na indexu $(h(x)+7l) \bmod N$, ale tam je nyní prvek x větší než y_1, takže funkce bude pokračovat dále po indexech $(h(x)+7(l+1)) \bmod N, \ldots, (h(x)+7l_1) \bmod N$. Prvek y_1 uložíme na místo $(h(x)+7l_1) \bmod N$. Jestliže tam předtím bylo prázdné políčko,

skončíme, jestliže tam byl prvek y_2 ($y_2 < y_1$), opět postupujeme dále po indexech $(h(x) + 7(l_1 + 1)) \bmod N, \ldots, (h(x) + 7l_2) \bmod N$. Toto opakujeme, dokud nenajdeme prvok y_k, který se již uloží na prázdné políčko. Na základě tohoto postupu můžeme sestavit proceduru *Zarad*:

```
procedure Zarad(X:integer);
var i:integer;
begin
  i:=(X mod 10 + X div 10) mod N;       {i:=h(X)}
  while P[i]<>0 do begin
    if P[i]<X then      {našli jsme menší prvek}
      swap(P[i],X);     {vymění hodnoty P[i] a X}
    i:=(i+7) mod N;     {posuneme se dále}
  end;
  P[i]:=X;              {na volné políčko uložíme X}
end;
```

Procedura *Zarad* vždy skončí, neboť $M < N$, a tedy v poli je nějaké políčko $P[i]$, které je prázdné, a pro i existuje j tak, že $(h(x) + 7j) \bmod N = i$. Proto nejpozději po j průchodech cyklus while skončí. Po provedení procedury všechny prvky, které v poli P byli, v něm zůstanou (i když možná na jiných místech) a přibude pouze prvek x. Současně pro každý index i ($0 \leqq i < N$) platí, že hodnota $P[i]$ před provedením procedury $Zarad(x)$ je menší nebo stejná jako hodnota $P[i]$ po provedení procedury. Nechť tedy po provedení procedury $Zarad(x)$ spustíme funkci $Vyhledej(y)$. Mohou nastat tyto možnosti:

- Prvek y se v poli P nenachází. Funkce $Vyhledej(y)$ nemůže nalézt něco, co v poli P není, a proto vrátí správný výsledek.
- $y = x$. V tomto případě už z popisu algoritmu vyplývá, že funkce $Vyhledej$ prvek x najde.
- $y \neq x$, y se nachází v poli P a jeho poloha se během výpočtu procedury $Zarad(x)$ nezměnila. Předtím funkce $Vyhledej$ prohledávala indexy $h(y), (h(y) + 7) \bmod N, \ldots, (h(y) + 7l) \bmod N$. To znamená, že pro každé i, $0 \leqq i < l$, platilo $P[(h(y) + 7i) \bmod N] > y$. Protože se hodnota žádného prvku z P během provádění $Zarad(x)$ nesnížila, platí to i nadále a proto $Vyhledej(y)$ bude prohledávat stejné indexy a úspěšně y najde.
- $y \neq x$, y se nachází v poli P a jeho poloha se během výpočtu procedury $Zarad(x)$ změnila, tj. je to jeden z prvků $y_1, y_2, \ldots, y_k$.

Nechť předtím *Vyhledej* prohledávala indexy $h(y), (h(y)+7) \bmod N, \ldots, (h(y)+7l) \bmod N$. Ze stejných důvodů jako v předcházejícím případě ani nyní neskončí dříve, ale na místě s indexem $(h(y)+7l) \bmod N$ je nyní prvek větší než y, a proto funkce bude pokračovat dále. Z popisu algoritmu však vyplývá, že prvek, na kterém se prohledávání zastaví, je právě hledaný prvek y.

P – II – 3

Celkový posun pera je součtem jednotlivých vektorů určených vstupní posloupností. Když starší kreslicí zařízení udělá pohyb o jednotku delší nebo kratší, je to to totéž, jako kdyby se kromě určeného vektoru posunulo ještě o jednotkový vektor ve směru nebo proti směru svého natočení. Protože sčítání vektorů je komutativní, představme si, že se tyto posuny o jednotkové vektory provedou všechny až nakonec. Nechť množina A je množina obsahující ke každému vektoru ze vstupní posloupnosti dva navzájem opačné jednotkové vektory (ve směru a proti směru tohoto vektoru). Naší úlohou tedy je vybrat z množiny A podmnožinu, jejíž součet je co nejdelší. Vybraná podmnožina může být libovolná (zařízení se sice vždy posune nejvýše o jeden z dvojice navzájem opačných vektorů, ale pokud bychom vybrali oba, je to totéž, jako kdybychom nevybrali ani jeden z nich).

Podmnožina s největším součtem jistě obsahuje z každé dvojice opačných vektorů právě jeden. To lehce dokážeme sporem: nechť vektor (x, y) je součtem podmnožiny s největším součtem, která neobsahuje ani jeden z jednotkových vektorů (x_1, y_1) a $(-x_1, -y_1)$. Potom ale jeden z vektorů $(x+x_1, y+y_1)$, $(x-x_1, y-y_1)$ je určitě delší než vektor (x, y). Platí, že $x_1^2+y_1^2 = 1$ a jedno z čísel $1+2(xx_1+yy_1)$, $1-2(xx_1+yy_1)$ je určitě kladné. Délka vektoru $(x+x_1, y+y_1)$ je $\sqrt{x^2+y^2+x_1^2+y_1^2+2(xx_1+yy_1)}$ a délka vektoru $(x-x_1, y-y_1)$ je $\sqrt{x^2+y^2+x_1^2+y_1^2-2(xx_1+yy_1)}$. Jedna z těchto délek je tedy jistě větší než délka vektoru (x, y), která je rovna $\sqrt{x^2+y^2}$. Proto vektor (x, y) není vektor s největší možnou délkou. Podobně se dá zdůvodnit, že pokud A obsahuje více stejných dvojic vektorů, do podmnožiny s největším součtem se vybere z každé dvojice stejný vektor.

Nyní dokážeme následující větu: Hledané řešení obsahuje všechny vektory z A ležící v jedné polorovině určené některou přímkou vedoucí bodem (0,0). Jsou-li některé dvojice vektorů rovnoběžné s přímkou, vyberou se vektory v jednom směru.

DŮKAZ: Nechť (x, y) je součet podmnožiny s nejdelším součtem, nechť přímka p je kolmá na (x, y) a prochází bodem $(0, 0)$. Nechť (x_1, y_1) je libovolný vektor z poloroviny, v níž se nachází (x, y). Potom úhel vektorů (x_1, y_1) a (x, y) je nejvýše 90 stupňů. Proto délka vektoru $(x + x_1, y + y_1)$ je větší než délka (x, y) (podle kosinové věty). Pokud by naše podmnožina neobsahovala některý z vektorů z této poloroviny, její součet se přidáním tohoto vektoru tudíž prodlouží, takže by nemohla být hledaným řešením. Jestliže by naopak obsahovala některý vektor z opačné poloroviny, jeho ubrání je totéž jako přidání vektoru k němu opačného (který je už z naší poloroviny), a to opět prodlouží součet.

Nechť $a_1, a_2, \ldots, a_{2n}$ jsou jednotkové vektory z množiny A uspořádané podle směru (vektor a_{i+n} je opačný k vektoru a_i pro $1 \leqq i \leqq n$). Vezměme dva sousední vektory z takto setříděné posloupnosti. Poloroviny určené všemi přímkami procházejícími „mezi" těmito dvěma vektory obsahují stejnou podmnožinu vektorů z A, a proto stačí uvažovat jen přímky se směry vektorů z A. Polorovina určená přímkou ve směru vektoru a_i obsahuje vektory $a_i, a_{i+1}, \ldots, a_{i+n-1}$, $1 \leqq i \leqq n$. Stačí tedy určit všechny takovéto součty a vybrat z nich největší. To lze provést pro již setříděnou posloupnost vektorů v čase $O(n)$.

Vzhledem k tomu, že směry vektorů určených vstupní posloupností jsou celé čísla od 0 do 359, můžeme je uspořádat v čase $O(n)$ přihrádkovým tříděním, tj. pro každý ze směrů spočítáme, kolik vektorů je v tomto směru. Máme-li l jednotkových vektorů téhož směru, můžeme je považovat za jeden vektor délky l. Stačí dokonce použít jen pole od 0 do 179, neboť vektor se směrem j a vektor k němu opačný se směrem $j + 180°$ mají stejnou délku.

P – II – 4

a) Dokážeme sporem, že není možné sestrojit požadovaný D0L systém. Nechť existuje D0L systém s růstovou funkcí n^n. Vezmeme k rovné maximální hodnotě z délek pravých stran pravidel. Z toho vyplývá, že máme-li v i-tém kroku slovo délky i^i, v $i + 1$-ním kroku může náš D0L systém vygenerovat slovo délky nejvýše ki^i. Uvažujme nyní slovo w_k. Jeho délka je k^k. Pro délku slova w_{k+1} potom musí platit

$$|w_{k+1}| < k|w_k| = kk^k = k^{k+1} < (k+1)^{k+1},$$

což je spor, neboť délka slova w_{k+1} má být $(k + 1)^{k+1}$.

b) řešením je D0L systém $(\{b,c,d\},\{b \to b^2, c \to b^2c^2, d \to \to b^2c^4d^2\}, dd)$.

Tvrzení: V n-tém kroku je tvar slova $(b^{(n-1)^2}c^{2(n-1)}d)^{2^n}$, a tedy $f(n) = 2^n((n-1)^2 + 2(n-1) + 1) = 2^n n^2$.

DŮKAZ: Pro $n = 1$ tvrzení zřejmě platí.

Nechť dále $w_n = b^{2^n(n-1)^2}c^{2^n 2(n-1)}d^{2^n}$. Potom po uplatnění pravidel bude $w_{n+1} = (b^{(n-1)^2}b^{2(n+1)}c^{2(n-1)}bc^2d = b^{n^2}c^{2n}d)^{2^{n+1}}$, čímž je tvrzení dokázáno.

P ≥ III – 1

Uvažujme v naší souřadnicové soustavě vodorovný pás trojúhelníků, který je vymezen y-ovými souřadnicemi y a $y+1$. Jestliže vezmeme stranu mnohoúhelníku s procházející tímto pásem s koncovými body $[x_1, y]$, $[x_2, y+1]$, jistě platí, že $x_1 = x_2$ nebo $x_1 + 1 = x_2$. Pokud známe součet $x_1 + x_2$, dokážeme určit x_1 a x_2: $x_1 = \lfloor \frac{1}{2}(x_1 + x_2) \rfloor$, $x_2 = \lceil \frac{1}{2}(x_1 + x_2) \rceil$. Proto můžeme každou stranu, která není vodorovná, jednoznačně popsat uspořádanou dvojicí čísel (i, j), kde $i = x_1 + x_2$ a $j = y$. Dvojici (i, j) budeme nazývat souřadnicemi strany. Mějme stranu s se souřadnicemi konců $[x_1, y]$, $[x_2, y+1]$ a stranu s' se souřadnicemi konců $[x_1', y]$, $[x_2', y+1]$, přičemž strana s je nalevo od strany s'. Potom část pásu ohraničená stranami s a s' je lichoběžník nebo v krajním případě trojúhelník a jeho obsah $S = (x_1' - x_1) + (x_2' - x_2)$ (první sčítanec určuje počet jednotkových trojúhelníků, které mají jednu ze svých stran na spodní přímce pásu, druhý počet těch, které mají jednu ze stran na horní přímce pásu). Po úpravě dostaneme vztah $S = (x_1' + x_2') - (x_1 + x_2)$, což už je vyjádření přímo pomocí souřadnic stran.

Mějme nyní dva mnohoúhelníky A a B a zkoumejme průnik A, B a našeho vodorovného pásu. Tento průnik se bude zřejmě skládat z lichoběžníků (příp. trojúhelníků) ohraničených stranami mnohoúhelníků. Bod pásu patří do průniku, je-li nalevo od něj lichý počet stran A a zároveň lichý počet stran B. Tedy začátkem nějakého takovéhoto lichoběžníku bude strana, od níž vlevo je sudý počet stran jejího vlastního mnohoúhelníku a lichý počet stran druhého mnohoúhelníku. Naopak koncem takového lichoběžníku bude strana, od níž vlevo je lichý počet stran její vlastního a lichý počet stran druhého mnohoúhelníku. Strany mnohoúhelníků, které procházejí jedním pásem, můžeme setřídit zleva doprava, tj. podle jejich první souřadnice. Potom jediným průchodem přes utříděnou posloupnost stran jednoduše spočítáme plochu průniku.

Algoritmus je tedy následující: setřídíme strany obou mnohoúhelníků podle pásu, ve kterém se nacházejí (podle jejich druhé souřadnice), přičemž vodorovné strany můžeme vynechat. Strany téhož pásu setřídíme zleva doprava (podle první souřadnice). Potom procházíme zároveň oběma seznamy setříděných stran a hledáme strany, které jsou začátky nebo konci lichoběžníků, a jejich první souřadnice odpočítáváme nebo připočítáváme k celkovému součtu.

Zbývá ještě popsat realizaci třídění. Je třeba si uvědomit, že má-li mnohoúhelník nějaké dva vrcholy s x-ovými souřadnicemi i a j $(i < j)$, má také vrcholy s x-ovými souřadnicemi $i+1, i+2, \ldots, j-1$, neboť délka každé strany je 1. Rozdíl maximální a minimální x-ové souřadnice mnohoúhelníku je tedy nejvýše N. Totéž samozřejmě platí i pro y-ovou souřadnici. Proto na třídění stran použijeme algoritmus Radixsort. Strany setřídíme nejprve podle první souřadnice a potom podle druhé, přičemž v obou případech použijeme stabilní a lineární třídění. První souřadnice stran jsou ale součtem dvou souřadnic vrcholů, takže rozdíl dvou souřadnic stran může být i dvojnásobkem rozdílu souřadnic vrcholů.

Díky lineárnímu třídění je paměťová i časová složitost algoritmu $O(N+M)$, kde N a M jsou počty stran mnohoúhelníků.

P – III – 2

b) Protože 7 a N jsou nesoudělná čísla, existuje l takové, že $7l \bmod N = N-3$ a $0 \leqq l < N$. Platí, že $l < N-1$, neboť kdyby se rovnaly, muselo by platit $(7N-7) \bmod N = N-3$, což pro žádné $N > 10$ zjevně neplatí.

Vkládejme do prázdného pole P procedurou *Zarad* postupně prvky $(l+2)N, (l+1)N, \ldots, 4N, 3N, N$. Uloží se postupně v tomto pořadí na místa $0, 7, 14, \ldots, N-3$. V poli je nyní obsazeno $l+1$ míst, a tedy aspoň jedno místo je volné. Když nyní zavoláme proceduru *Zarad* s parametrem $2N$, bude proměnná i postupně nabývat hodnot $0, 7, 14, \ldots, N-3$, neboť na pozicích $0, 7, 14, \ldots, N-10$ jsou uložena čísla větší než $2N$. Ale $P[i] = N$, a proto další hodnota indexu bude opět 0. Procedura bude proto cyklicky nabývat stále tyto hodnoty a tudíž není konečná pro každý vstup.

a) Mějme funkci *Posun*, která nám vrátí další hodnotu, jakou by získal index i v proceduře *Zarad* (tato hodnota závisí na původní hodnotě i, na prvku x a prvku $P[i]$):

```
function Posun(i:integer; x:integer):integer;
begin
  if P[i]>x then
    posun:=(i+7) mod N
  else
    posun:=(i+3) mod N;
end;
```

V určitém stavu pole P chceme vyhledat prvek x. Definujme (nekonečnou) posloupnost indexů $h_0, h_1, h_2, \ldots$, kde $h_0 = x \bmod N$ a $h_{i+1} = Posun(h_i, x)$.

Mohou nastat tři případy:

1. Prvek x se v poli P nachází. Všimněte si, že procedura *Zarad* mění jen hodnoty nulových prvků pole P. Hodnoty na indexech, přes které přecházela procedura *Zarad*(x), se tedy nezměnily. Jsou to indexy $h_0, h_1, \ldots, h_k$, kde k je nejmenší číslo takové, že $P[h_k] = x$.
2. Prvek x se v poli nenachází a procedura *Zarad* by ho zařadila na volné místo $P[i]$. Při tomto zařazení by procedura prohlédla prvky P s indexy $h_0, h_1, \ldots, h_k$, kde k je nejmenší číslo takové, že $P[h_k] = 0$ (resp. že $i = h_k$).
3. Prvek x se v poli P nenachází a procedura *Zarad* pro vstup x neskončí. Také v tomto případě by procedura prohlédla indexy $h_0, h_1, \ldots$. Nechť k je nejmenší číslo takové, že existuje $l < k$ takové, že $h_l = h_k$. Potom posloupnost $\{h_n\}$ začíná indexy $h_0, h_1, \ldots, h_{l-1}$ a potom se už stále periodicky opakují indexy $h_l, h_{l+1}, \ldots, h_{k-1}$.

Funkce *Vyhledej* tedy rozpoznává tyto tři případy, v prvním případě vrátí h_k a ve druhých dvou -1. V prvních dvou případech stačí postupně vypočítávat indexy h_n, dokud nenajdeme 0 nebo x. Jestliže ale chceme zjistit, zda nenastal třetí případ, potřebujeme ověřit, jestli se právě vypočítaný index h_n už předtím v posloupnosti nevyskytoval. Budeme se po poli posunovat se dvěma indexy i a j, každý bude postupně nabývat hodnot posloupnosti h_n. Index i ale budeme posouvat „rychleji". Když i nabyde hodnoty h_n, tak hodnota j bude $h_{\lfloor n/2 \rfloor}$. V případě, že se indexy v posloupnosti periodicky opakují, po jistém čase bude platit $i = j$. V okamžiku, kdy j nabyde poprvé hodnoty h_l, i se nachází také někde v cyklu. Po každém posunu j se vzdálenost i a j zmenší o 1, neboť index i se mezitím posunul dvakrát, a tak vlastně „dobíhá" index j. Proto se index j posune nejvýše k-krát, a jelikož i se posouvá dvakrát tak často, celkový počet posunů je nejvýše $3k$.

```
function Vyhledej(x:integer);
var i,j,vysledek:integer;
    menit_j:boolean;
begin
  vysledek:=0;                {výsledek ještě neznáme}
  i:=x mod N;                 {první index}
  j:=i;
  menit_j:=false;
  while vysledek=0 do         {dokud neznáme výsledek}
    if P[i]=x then            {našli jsme x}
      vysledek:=i
    else if P[i]=0 then       {x v poli není}
      vysledek:=-1
    else begin
      i:=posun(i,x);          {posuneme i}
      if menit_j then begin {je-li třeba, posuneme j}
        j:=posun(j,x);
        if j=i then vysledek:=-1;
                              {i a j se setkaly - cyklus}
      end;
      menit_j:=not menit_j;
    end;
  Vyhledej:=vysledek;
end;
```

P – III – 3

a) Dokážeme sporem, že není možné sestrojit požadovaný D0L systém. Nechť tedy existuje D0L systém s růstovou funkcí n^2. Nechť má tento D0L systém pouze jedno pravidlo. Potom toto pravidlo musí mít tvar $a \to a^i$ a $w_1 = a$, neboť $|w_1| = f(1) = 1$. Pro w_2 platí $w_2 = a^i$ a $|w_2| = f(2) = 4$. Z toho vyplývá, že $i = 4$. Ale pak pro w_3 platí $w_3 = a^{i^2}$ a $|w_3| = f(3) = 9$. Dostáváme tedy, že $9 = i^2 = 4^2 = 16$, což je spor.

Náš D0L systém musí tedy mít alespoň dvě pravidla. Nechť jsou to pravidla $a \to u$, $b \to v$, kde u, v jsou z $\{a, b\}^*$. Bez újmy na obecnosti můžeme předpokládat, že $w_1 = a$. Potom $w_2 = u$, a tudíž $|u| = f(2) = 4$. Nechť slovo u obsahuje k znaků a. Potom $w_3 = u^k v^{4-k}$ a platí, že

$|w_3| = f(3) = 9$. Ale současně platí $|w_3| = k|u| + (4-k)|v|$ a $|u| = 4$. Dostáváme tedy rovnici $4k + 4|v| - k|v| = 9$. Jestliže za k postupně dosadíme všechny možné hodnoty $(0, 1, 2, 3, 4)$, dostáváme pro $|v|$ následující rovnosti: $4|v| = 9$, $4+4|v|-|v| = 9$, $8+4|v|-2|v| = 9$, $12+4|v|-3|v| = 9$ a $16 + 4|v| - 4|v| = 9$. Ani jedna z těchto rovností však nemá nezáporné celočíselné řešení, a proto D0L systém s požadovanými vlastnostmi neexistuje.

b) D0L systém s požadovanými vlastnostmi neexistuje. Dokážeme to opět sporem. Nechť tedy existuje D0L systém s k-prvkovou abecedou a růstovou funkcí $f(n) = \lfloor \log_{k+1}(n+1) \rfloor + 1$.

Nechť $m > 1$. Označme n_1 nejmenší takové číslo n, pro které $f(n) = m$, a n_2 největší takovéto číslo. Potom $n_1 = (k+1)^{m-1} - 1$ a $n_2 = (k+1)^m - 2$. Tedy všechna slova w s indexem n_1, w s indexem $n_1 + 1, \ldots, w$ s indexem n_2 mají délku m. Jejich počet je $n_2 - n_1 + 1 = k(k+1)^{m-1}$. Ale různých slov délky m je k^m a $k^m < k(k+1)^{m-1}$. Proto určitě existují čísla $i, l > 0$ taková, že $w_i = w_{i+l}$. Když ale ze slova w_i vznikne po l krocích opět w_i, úsek $w_i, w_{i+1}, \ldots, w_{l-1}$ se bude i dále stále periodicky opakovat. Tedy D0L systém vyprodukuje nekonečně mnoho slov délky m, což je spor, neboť takových slov má být $n_2 - n_1 + 1$.

P – III – 4

Budeme nejprve řešit trochu obecnější úlohu: předpokládejme, že na začátku nemusí být všechny lampy vypnuté. Stav lamp si budeme pamatovat v poli l, kde $l[i] =$ true, pokud je i-tá lampa zapnutá a $l[i] =$ false, jestliže je vypnutá. Řešením této úlohy budeme rozumět takovou podmnožinu přepínačů, že pokud je všechny přepneme, všechny lampy budou zapnuté. Pokud $N = 0$, řešením úlohy je zřejmě prázdná množina. Nechť tedy $N > 0$. Mohou nastat dva případy:

1. $l[1] =$ false.

Jestliže žádný interval nezačíná lampou 1, úloha zjevně nemá řešení. V opačném případě najdeme nejkratší interval, který začíná lampou 1 (nechť jeho konec je b). Změníme všechna $l[i]$ z tohoto intervalu na opačná a všem intervalům, které začínají lampou 1, změníme začátek na $b + 1$ (intervaly, které se tím stanou prázdnými, už dále neuvažujeme). Dostáváme tak novou úlohu pro lampy $2, 3, \ldots, N$, která má řešení právě tehdy, má-li řešení původní úloha.

DŮKAZ: Nechť pozměněná úloha má řešení R. Potom když přepneme lampy z původní úlohy přepínači z R, budou svítit jen lampy s čísly

většími než b. Nechť R obsahuje k intervalů, kterým jsme začátek změnili z 1 na $b+1$. Nechť R' vznikne z R tak, že těmto intervalům změníme začátek zpět na 1. Jestliže je k liché, každou lampu z intervalu $\langle 1, b\rangle$ jsme oproti řešení R přepnuli ještě lichý počet krát, tedy zůstane zapnutá. V tomto případě je tedy R' řešením naší původní úlohy. Pokud je k sudé, lampy z intervalu $\langle 1, b\rangle$ zůstanou vypnuté a proto k R' je třeba ještě přidat interval $\langle 1, b\rangle$. Naopak, nechť naše původní úloha má řešení R. Počet intervalů z R, které obsahují lampu 1, je jistě lichý. Když všem těmto intervalům změníme začátek na $b+1$ a vyhodíme intervaly nulové délky, dostaneme řešení změněné úlohy.

2. $l[1] = \text{true}$

Protože lampa 1 už svítí, není třeba, aby se dala nějakým přepínačem přepnout. Jestliže neexistuje interval, který začíná lampou 1, úloha má řešení právě tehdy, má-li řešení úloha pro lampy $2, 3, \ldots, N$ a stejnou množinu intervalů. Pokud existuje interval, který začíná lampou 1, opět z nich vezmeme nejkratší, změníme začátky intervalů stejně jako v předchozím případě, ale neměníme hodnoty pole l. Takováto pozměněná úloha má opět řešení právě tehdy, když má řešení původní úloha.

Důkaz je obdobný jako v případě, kdy $l[1] = \text{true}$ (uvědomte si však, že počet intervalů, které obsahují lampu 1, musí být tentokrát samozřejmě sudý).

Naším úkolem však není nalézt řešení, ale pouze zjistit, zda nějaké řešení existuje. Je proto třeba na začátku nastavit všechny prvky pole l na false a podle uvedených úvah postupovat od první lampy až do poslední, přičemž modifikujeme pole l a začátky příslušných intervalů. Dostáváme tak algoritmus s časovou složitostí $O(NM)$ a paměťovou $O(N+M)$.

P – III – 5

Je důležité uvědomit si, že můžeme jít jen směrem na východ nebo na jih. Abychom se dostali z křižovatky $(1,1)$ na křižovatku (M,N), musíme přejít o $M-1$ ulic na jih a o $N-1$ ulic na východ. Je jedno, v jakém pořadí střídáme směry, počet křižovatek, kterými projdeme, je vždy $M+N-1$.

Počet různých cest také spočítáme jednoduše. Označme $a[i,j]$ počet různých cest z $(1,1)$ do (i,j). Pokud se křižovatka (i,j) opravuje, položíme $a[i,j] = 0$. Na libovolnou křižovatku ležící na východozápadní ulici číslo 1 (tj. na nejsevernější ulici) se můžeme dostat nejvýše jedním způsobem, a to tak, že půjdeme stále na východ. Jestliže se však opravuje

vozovka někde mezi touto křižovatkou a křižovatkou $(1,1)$, nemůžeme se tam dostat vůbec. Totéž platí i pro severojižní ulici číslo 1.

Uvažujme nyní křižovatku (i,j), kde $i,j > 1$ a tato křižovatka se neopravuje. Na tuto křižovatku můžeme přijít buď z křižovatky $(i-1,j)$, nebo z křižovatky $(i,j-1)$. Jestliže tedy známe hodnoty $a[i-1,j]$ a $a[i,j-1]$, pak $a[i,j] = a[i-1,j]+a[i,j-1]$. Na základě této úvahy můžeme sestrojit algoritmus, který bude postupně po řádcích vyplňovat pole a. Nakonec budeme mít v $a[M,N]$ počet různých cest z $(1,1)$ do (M,N). Je-li tento počet roven 0, žádná taková cesta neexistuje.

Vzhledem k tomu, že musíme vyplnit celou tabulku velikosti $M \times N$ a pro výpočet hodnoty každého políčka vykonáme konstantní počet operací, časová složitost algoritmu je $O(MN)$.