37. ročník matematické olympiády na středních školách

Texty a řešení úloh kategorie P

In: Leo Boček (editor); Luboš Brim (editor); Tomáš Hecht (editor); Karel Horák (editor): 37. ročník matematické olympiády na středních školách. Zpráva o řešení úloh ze soutěže konané v školním roce 1987/88. 29. mezinárodní matematická olympiáda. (Czech). Praha: Státní pedagogické nakladatelství, 1990. pp. 114–145.

P - I - 1

Je dáno celé číslo T a dvě konečné celočíselné posloupnosti $A_1, A_2, \ldots, A_N$ a $B_1, B_2, \ldots, B_N$ takové, že $N \geqq 1$, $A_1 < A_2 < \ldots < A_N$ a pro všechna i, $1 \leqq i \leqq N$, je $B_i > 0$. Na poště jsou tři přepážky poskytující stejné služby. Přepážky otvírají v čase T. Poštu navštíví během dne N zákazníků. Zákazník i přichází v čase A_i a požaduje službu v délce B_i časových jednotek. Zákazníci jsou obsluhováni v pořadí jejich příchodu na poštu. Přepážka je ihned po obsloužení jednoho zákazníka k dispozici pro obsluhu dalšího. Přepážky zavírají po obsloužení všech zákazníků.

Navrhněte a dokažte algoritmus, který určí celkovou dobu, po kterou na poště čeká na obsluhu alespoň jeden zákazník.

Řešení

Hledaná doba F je délka sjednocení intervalů $[A_i, C_i]$ pro $1 \leqq i \leqq N$, kde C_i je okamžik, kdy zákazník i přijde na řadu. Platí $A_i \leqq C_i$ pro všechna i, $C_1 \leqq C_2 \leqq \ldots \leqq C_N$.

Dále budeme hodnoty C_i počítat postupně pro zákazníky $i = 4, 5, \ldots, N$.

Označme d_i okamžik, kdy je zákazník i obsloužen. Platí $d_i = C_i + B_i$. Pro první tři zákazníky je určení okamžiku, kdy přijdou na řadu, snadné. Je to buď ihned v čase T, nebo, přijdou-li až po otevření pošty, ihned v čase A_i. Tj. $C_i = \max(T, A_i)$, $i = 1, 2, 3$.

Zákazník i ($i > 3$) přijde na řadu buď v čase A_i, je-li v době jeho příchodu alespoň jedna přepážka volná, nebo v opačném

případě (všechny přepážky jsou obsazené) v okamžiku, kdy se jedna z nich uvolní. V situaci, kdy jsou všechny přepážky obsazené, jsou u nich obsluhováni tři zákazníci z množiny $\{1, 2, \ldots, i-1\}$. Pro všechny z nich jsme již stanovili, kdy přišli na řadu (hodnoty C_j), a známe tedy i kdy byli nebo budou obslouženi (hodnoty d_j). Můžeme tedy i určit, kteří zákazníci jsou u přepážek: jsou to zákazníci x, y, z, pro něž platí $d_x, d_y, d_z > d_k$ pro všechna k z $\{1, 2, \ldots, i-1\} - \{x, y, z\}$.

Jedna z přepážek se uvolní, jakmile první ze zákazníků x, y, z odejde, a to bude v čase $\min(d_x, d_y, d_z)$. Z provedených úvah je vidět, že tento čas je dán třetí největší hodnotou z $\{d_1, d_2, \ldots, d_{i-1}\}$.

Shrneme-li, pak $C_i = \max(\max 3(d_1, d_2, \ldots, d_{i-1}), A_i)$, kde max 3 je třetí největší hodnota z hodnot uvedených v závorkách. V algoritmu budeme předpokládat, že prvky posloupností A a B jsou uloženy v polích $A[1..N]$ a $B[1..N]$, hodnoty C_i budeme ukládat do pole $C[1..N]$. Tři největší z doposud spočtených hodnot d_i budou průběžně uloženy v poli $D[1..3]$. Napišme proceduru max3, která ve svém výstupním parametru určí index j nejmenší hodnoty z $D[1]$, $D[2]$, $D[3]$, což znamená, že v $D[j]$ je uložena třetí největší hodnota z doposud spočtených hodnot d_i.

```
procedure max3 (var j: integer);
begin    if D[1] < D[2] then j := 1 else j := 2;
         if D[j] > D[3] then j := 3
end;
```

První část algoritmu počítá hodnoty C_i:

```
for i := 1 to min (3, N) do begin C[i] := max (T, A[i]);
```

```
                    D[i] := C[i] + B[i];
                end;
for i := 4 to N do
    begin max3 (j);
        C[i] := max(D[j], A[i]);
        D[j] := C[i] + B[i];
end;
```

Nyní máme N intervalů $[A_i, C_i]$, $i = 1, 2, \ldots, N$. Platí $A_1 < A_2 < \ldots < A_N$, $C_1 \leqq C_2 \leqq \ldots \leqq C_N$. Máme určit délku jejich sjednocení. Předpokládejme, že známe délku sjednocení prvních i intervalů. Mohou nastat dva případy:
1. Je-li $A_i > C_{i-1}$, pak interval $[A_i, C_i]$ má prázdný průnik se všemi intervaly $[A_j, C_j]$ pro $j < i$ (neboť $C_{i-1} \geqq C_{i-2} \geqq \geqq \ldots \geqq C_1$) a délku je nutno zvětšit o $C_i - A_i$.
2. Je-li $A_i \leqq C_{i-1}$, pak interval $[A_i, C_{i-1}]$ je průnikem intervalů $[A_i, C_i]$ a $[A_{i-1}, C_{i-1}]$, a protože $C_{i-1} \geqq C_{i-2} \geqq \ldots \geqq \geqq C_1$, je délku nutno zvětšit o $C_i - C_{i-1}$.
Shrneme-li oba případy, pak dosavadní délku je nutno zvětšit o $C_i - \max(A_i, C_{i-1})$. Protože délku prvního intervalu známe, s využitím právě popsaného postupu dokončíme algoritmus:

```
F := C[1] - A[1];
for i := 2 to N do
    F := F + C[i] - max(A[i], C[i - 1]);
writeln(F).
```

Důkaz správnosti algoritmu byl prováděn současně s jeho návrhem, složitost je lineární.

Je dána konečná celočíselná posloupnost $X_1, \ldots X_N$ taková, že $N \geqq 2$ a $X_1 < \ldots < X_N$. Navrhněte a dokažte (co nejlepší) algoritmus, který určí celkový počet konečných celočíselných posloupností $V_0, \ldots V_N$ takových, že $V_0 \leqq \ldots \leqq V_N$ a pro všechna i, $1 \leqq i \leqq N$, platí $V_{i-1} + V_i = 2^* X_i$.

Řešení

Podle zadání platí:

$$V_{i-1} \leqq V_i \quad \text{pro} \quad i = 1, 2, \ldots, N \tag{1}$$
$$V_{i-1} + V_i = 2X, \quad i = 1, 2, \ldots, N \tag{2}$$

Využitím (1) a (2) dostáváme:

$$2X_i = V_{i-1} + V_i \geqq V_{i-1} + V_{i-1} = 2V_{i-1} \Rightarrow X_i \geqq V_{i-1}\,,$$
$$i = 1, \ldots, N \tag{3}$$
$$2X_i = V_{i-1} + V_i \leqq V_i + V_i = 2V_i \Rightarrow X_i \leqq V_i$$
$$i = 1, 2, \ldots, N \tag{4}$$

Vztahy (3) a (4) dohromady dávají:

$$X_i \leqq V_i \leqq X_{i+1} \quad i = 1, 2, \ldots, N - 1 \tag{5}$$
$$V_0 \leqq X_1, \; X_N \leqq V_N \tag{6}$$

Každá z posloupností $V_0, V_1, \ldots, V_N$ je jednoznačně určena svým libovolným prvkem V_i (ostatní prvky se vypočítají z rovnic $V_{j-1} + V_j = 2X_j, j = 1, 2, \ldots, N$).

Celkový počet posloupností $V_0, V_1, \ldots, V_N$ je proto roven počtu celých čísel, která mohou být i-tým prvkem (pro libovolné pevné i, $0 \leqq i \leqq N$) nějaké z takových posloupností. Řekneme, že celé číslo c má vlastnost $v(i)$, $1 \leqq i < N$, jestliže $X_i \leqq c \leqq X_{i+1}$ a jestliže existuje celočíselná posloupnost

$V_0, V_1, \ldots, V_i$ taková, že $V_0 \leqq V_1 \leqq \ldots \leqq V_i = c$, a pro všechna j, $1 \leqq j \leqq i$ je $V_{j-1} + V_j = 2X_j$.
Označme jako $p(i)$ počet čísel s vlastností $v(i)$.
Potom $p(N-1)$ je rovno hledanému počtu posloupností $V_0, V_1, \ldots, V_N$. Platí: Jestliže $p(i) = 0$ pro nějaké i, pak $p(i+1) = p(i+2) = \ldots = p(N-1) = 0$. Dále platí $p(1) = X_2 - X_1 + 1$ (čísla s vlastností $v(1)$ jsou totiž právě jen všechna celá čísla splňující nerovnost $X_1 \leqq c \leqq X_2$).
Jsou-li c_1 a c_2 čísla s vlastností $v(i)$, $i = 1, 2, \ldots, N-1$ taková, že $c_1 = c_2 - k$, $k \geqq 0$, pak pro čísla b_1, b_2 taková, že $c_1 + b_1 = 2X_{i+1}$, $c_2 + b_2 = 2X_{i+1}$, z těchto rovnic dostaneme $b_1 = b_2 + k$.
Je-li c rovněž číslo s vlastností $v(i)$ takové, že $c_1 \leqq c \leqq c_2$, pak číslo b takové, že $c + b = 2X_{i+1}$, vyhovuje nerovnosti $b_1 \geqq b \geqq b_2$. Z těchto úvah a z definice vlastnosti $v(i)$ vyplývá, že jsou-li všechna čísla s vlastností $v(i)$, $i = 1, 2, \ldots, N-2$, právě všechna celá čísla c splňující nerovnost $D \leqq c \leqq H$ pro nějaké D a H, pak všechna čísla b s vlastností $v(i+1)$ musí splňovat nerovnosti $2X_{i+1} - H \leqq b \leqq 2X_{i+1} - D$ a $X_{i+1} \leqq b \leqq X_{i+2}$ a zároveň každé celé číslo splňující tyto nerovnosti má vlastnost $v(i+1)$.
Protože čísla s vlastností $v(1)$ jsou všechna celá čísla vyhovující nerovnosti $X_1 \leqq c \leqq X_2$, jsme tedy schopni počítat počty $p(i)$ postupně pro $i = 2, 3, \ldots, N-1$, jak je zachyceno v algoritmu: {Předpokládáme, že prvky posloupnosti X jsou uloženy v poli $X[1..N]$}

$D := X[1]$; $H := X[2]$; počet $:= H - D + 1$;
{počet $= p(1)$}
$i := 2$;

```
while i < N and počet > 0 do
    begin {počet = p (i − 1)}
        D 1 := D;
        D := max (X [i], 2 * X [i] − H);
        H := min (X [i + 1], 2 * X [i] − D 1);
        if D ≦ H then počet := H − D + 1
                 else počet := 0;
        {počet = p (i)}
        i := i + 1
    end;
writeln (počet).
```

Algoritmus je lineární, důkaz správnosti je obsažen v návrhu algoritmu.

P - I - 3

Je dán následující program:

```
10 LET Y = 1
20 IF X > 100
THEN 60
30 LET X = X + 11
40 LET Y = Y + 1
50 GOTO 20
60 IF Y = 1
THEN 100
70 LET X = X − 10
80 LET Y = Y − 1
90 GOTO 20
100 STOP
```

```
Y := 1;
while (Y <> 1) or (X < = 100)
do if  X < = 100
  then
  begin X := X + 11;  {30}
        Y := Y + 1
  end
  else
  begin X := X − 10;  {70}
        Y := Y − 1
  end;
```

Nechť m je libovolná celočíselná hodnota vstupní proměnné X. Označme $A(m)$, resp. $B(m)$ počet provedení příkazu 30, resp. 70 při výpočtu pro m. Určete a zdůvodněte vztah mezi hodnotami $A(m)$ a $B(m)$.

Řešení

Současně s příkazem 30, resp. 70 se vždy provádí i příkaz bezprostředně následující, tj. $Y := Y + 1$, resp. $Y := Y - 1$. Na počátku je proměnné Y přiřazena hodnota 1, tj. před cyklem platí $Y = 1$. Cyklus končí při splnění podmínky $Y = 1$ **and** $X > 100$. Proto v případě ukončení cyklu je hodnota proměnné Y před jeho provedením a po jeho ukončení stejná, a tedy počet provedení příkazu 30 se rovná počtu provedení příkazu 70, tj. $A(m) = B(m)$.

Nyní dokážeme, že pro každé m cyklus končí. Je-li $m > 100$, pak se cyklus přímo ukončí. Předpokládejme proto dále, že $m \leqq 100$. Nejprve doplníme program o speciální pomocné proměnné i, j, n, tzv. čítače, které »počítají« počet skutečných průchodů cyklem:

```
Y := 1; i := 0; j := 0; n := 0;
while (Y ≠ 1) or (X ≧ 100) do
      begin {A} n := n + 1;
            if X ≦ 100 then
            begin X := X + 11;  {30}
                  Y := Y + 1;
                  i := i + 1
            end
            else
            begin X := X - 10;  {70}
```

```
        Y := Y − 1;
        j := j + 1
    end
end
```

Ukážeme, že hodnota čítače n je shora omezena konstantou, a tedy že cyklus končí. Je zřejmé, že v bodě A vždy platí

$$n = i + j. \tag{1}$$

Tento vztah umožňuje omezit n za pomoci omezení pro i a j. Poněvadž Y je zvětšeno o 1 vždy společně s i a zmenšeno o 1 vždy společně se zvětšením j o 1, platí v bodě A vždy podmínka

$$Y = i - j + 1. \tag{2}$$

Podobně platí i

$$X = 11i - 10j + m. \tag{3}$$

Dále si všimněme, že proměnné Y je na počátku přiřazena hodnota 1 a že Y je zmenšeno jen tehdy, když $Y > 1$. Proto v A vždy platí

$$Y \geqq 1, \tag{4}$$

což společně se vztahem (2) dává, že v A vždy platí

$$i \geqq j. \tag{5}$$

Užitím (5) a (3) dostáváme

$$X \geqq 11i - 10i + m = i + m \tag{6}$$

a

$$X \geqq 11j - 10j + m = j + m. \tag{7}$$

Sečtením posledních dvou nerovností dostaneme $2X \geqq i + j + 2m$ a užitím (1) získáme v bodě A podmínku

$$X \geqq n/2 + m. \tag{8}$$

Omezením hodnot proměnné X se nám tedy podaří omezit i hodnoty čítače n.

Při prvním vstupu výpočtu do bodu A platí $X = m$ **and** $Y = 1$ **and** $n = 0$. Podle našeho předpokladu je $m \leqq 100$, a proto v bodě A vždy platí $X \leqq 111$. Tedy v bodě A vždy platí

$$(X = m \textbf{ and } Y = 1 \textbf{ and } n = 0) \textbf{ or } X \leqq 111. \tag{9}$$

Z (8) a (9) nyní dostáváme

$$X \geqq n/2 + m \textbf{ and } [(X = m \textbf{ and } Y = 1 \textbf{ and } n = 0) \textbf{ or } X \leqq 111] \tag{10}$$

tedy

$$n = 0 \textbf{ or } n/2 + m \leqq 111.$$

Hodnota čítače n je shora omezena a cyklus musí skončit.

P - I - 4

(a) Navrhněte schéma S, které počítá součin dvou přirozených čísel.

(b) Navrhněte schéma S takové, že pro každé vstupní slovo X, tvořené znaky a, b, c, je výsledek Y výpočtu dán takto:

— $S(X) = X$, je-li délka X sudá, a

— $S(X)$ vznikne z X vynecháním každého třetího znaku b zleva, je-li délka slova X lichá.

Poznámka: Markovovy algoritmy

Konečnou posloupnost znaků nazveme slovem (například, *aba*, 01*AB*, 1234 jsou slova). Prázdnou posloupnost, značenou ε, nazveme prázdné slovo. Počet znaků ve slově nazveme délkou slova (slovo ε je délky 0, slovo *abaa* má délku 4). Zřetězení slova $P = a_1 \ldots a_n$ se slovem $Q = b_1 \ldots b_m$ je slovo $PQ = a_1 \ldots a_n, b_1 \ldots b_m$ (např.: Je-li $P = aabb$, $Q = bba$, pak $PQ = aabbbba$; je-li $Q' = \varepsilon$, pak $PQ' = aabb$).

Jsou-li P a Q slova, řekneme, že slovo P se vyskytuje v Q, jestliže existují slova U a T taková, že $Q = UPT$. Nejlevější výskyt P v Q je takový, pro který má slovo U nejmenší délku.

Nechť P a Q jsou slova. Výraz tvaru $P \to Q$ nazýváme obyčejné pravidlo, výraz tvaru $P \to .Q$ nazýváme koncové pravidlo. Zápis $P \to (.)Q$ znamená obyčejné nebo koncové pravidlo. Konečnou posloupnost pravidel

S: $P_1 \to (.)\, Q_1$

$\ldots$

$P_n \to (.)Q_n$

nazveme schéma. Každé schéma zadává (tzv. Markovův) algoritmus následujícím způsobem. Výpočet podle schématu S pro dané vstupní slovo X probíhá takto: Nalezneme první pravidlo $P_m \to (.)Q_m$ takové, že P_m se vyskytuje v X. Nejlevější výskyt slova P_m v X nahradíme slovem Q_m. Nechť R_1 je výsledek tohoto nahrazení (stručně budeme psát $X \vdash R_1$). Bylo-li použito koncové pravidlo $P_m \to .Q_m$, pak výpočet končí a výsledek výpočtu je slovo R_1. Bylo-li použito obyčejné pravidlo $P_m \to Q_m$, použijeme na R_1 stejný postup, který byl aplikován na X (schéma prohledáváme opět od

začátku) atd. Jestliže někdy nastane situace, kdy se žádné ze slov $P_1, \ldots, P_n$ nevyskytuje v právě zpracovaném slově R_i, pak výpočet končí s výsledkem R_i. Při tom je možné, že popsaný proces nikdy neskončí. Pak řekneme, že výsledek výpočtu není pro slovo X definován. Je-li výsledek výpočtu pro X definován, označíme ho $S(X)$.

Příklad 1

Navrhněte schéma S, které připíše na konec každého slova X, tvořeného ze znaků 1, 0, slovo 10. Řešením je schéma

S: $c1 \rightarrow 1c$
$c0 \rightarrow 0c$
$c \rightarrow .10$
$\varepsilon \rightarrow c$

Schéma S používá pomocný znak c. Jako první bude aplikováno pravidlo $\varepsilon \rightarrow c$, neboť předchozí pravidla obsahují na svých levých stranách znak c, a ten se nevyskytuje v žádném vstupním slově. Navíc, prázdné slovo se vyskytuje na začátku každého slova. Následuje opakované užití vždy jednoho z prvních dvou pravidel, a to tak dlouho, dokud se c »neposune« až na konec. Poté je aplikováno třetí pravidlo a výpočet končí, neboť se jedná o koncové pravidlo. Například pro vstupní slovo $X = 1011$ probíhá výpočet takto:

$R_1 = c1011$, $R_2 = 1c011$, $R_3 = 10c11$, $R_4 = 101c1$, $R_5 = 1011c$, $R_6 = 101110$. Stručně zapsáno $1011 \vdash c1011 \vdash 1c011 \vdash 10c11 \vdash 101c1 \vdash 1011c \vdash 101110$.

Poznamenejme ještě, že uvedené schéma S lze zapsat také ve zkrácené podobě ve tvaru:

S: $cx \rightarrow xc$

$c \rightarrow .10$

$\varepsilon \rightarrow c$

kde $x \in \{0, 1\}$, ve kterém jsou první dvě pravidla původního schématu zapsána jako jedno pravidlo.

Příklad 2

Navrhněte schéma S, které »zdvojuje« vstupní slovo X, tvořené ze znaků a, b, tj. $S(X) = XX$. Řešením je schéma

S: $za \rightarrow aAz$ (1)

$zb \rightarrow bBz$ (2)

$Aa \rightarrow aA$ (3)

$Ab \rightarrow bA$ (4)

$Ba \rightarrow aB$ (5)

$Bb \rightarrow Bb$ (6)

$z \rightarrow t$ (7)

$At \rightarrow ta$ (8)

$Bt \rightarrow tb$ (9)

$t \rightarrow \varepsilon$ (10)

$\varepsilon \rightarrow z$ (11)

Schéma S používá pomocné znaky A, B jako dvojníky pro znaky a, b a další pomocné znaky z, t.

První bude aplikováno pravidlo (11), kterým se na začátek slova X připíše znak z. Pak jsou opakovaně aplikována pravidla (1) a (2) připisující za každý znak jeho dvojníka. Přitom se znak z posouvá doprava tak dlouho, dokud se nestane posledním znakem. Dále jsou aplikována pravidla (3)—(6), kterými se přesunou všechny znaky A, B doprava. Přitom zůstává zachováno vzájemné pořadí znaků a, b i vzájemné pořadí znaků A, B. Tím vznikne slovo tvořené vstupním

slovem X, následovaným jeho »dvojníkem«. Pravidlem (7) se znak z změní na znak t. Pravidly (8) a (9) jsou pak pomocné znaky A, B nahrazeny svými protějšky a, b. Výpočet končí užitím pravidla (10), které odstraní pomocný znak t ze středu slova. Například pro vstupní slovo $X = abb$ probíhá výpočet takto:

$X = abb \vdash zabb \vdash aAzbb \vdash aAbBzb \vdash aAbBbBz \vdash$
$\vdash abABbBz \vdash abAbBBz \vdash abbABBz \vdash abbABBt \vdash$
$\vdash abbABtb \vdash abbAtbb \vdash abbtabb \vdash abbabb.$

Markovovy algoritmy můžeme používat i k práci s přirozenými čísly. Pro její usnadnění se často přirozené číslo n reprezentuje slovem n, které je definováno induktivně takto: $0 = a$, $n + 1 = na$. Každou k-tici přirozených čísel $(n_1, \ldots, n_k)$ reprezentujeme slovem $(n_1, \ldots, n_k) = n_1 b n_2 b \ldots$ $\ldots b n_k$. Například $(3, 1, 2) = aaaabaabaaa$. Poznamenejme, že jsou možné i jiné reprezentace.

Příklad 3

Navrhněte schéma S, které počítá součet tří přirozených čísel. Vstupním slovem bude výše uvedená reprezentace této trojice. Řešením je schéma

$$S: ba \to \varepsilon$$

Každý výpočet podle schématu S končí po dvou aplikacích jediného pravidla schématu. Všimněte si, že na rozdíl od příkladů 1 a 2, kde výpočet končil aplikací koncového pravidla, zde končí výpočet tím, že pravidlo již nelze více aplikovat. Např. pro součet $1 + 1 + 2$ je vstupní slovo $X =$ $= aabaabaaa$ a výpočet proběhne takto: $aabaabaaa \vdash$ $\vdash aaabaaa \vdash aaaaa$. Výsledek reprezentuje číslo 4.

Řešení

a) Dvojici m, n přirozených čísel $(m, n \geqq 0)$ reprezentujeme

slovem $a^{m+1}ba^{n+1}$. Algoritmus je založen na postupném »sčítání«. Řešením je schéma

$$
\begin{aligned}
\text{S:}\quad & FXX \to FX && (1)\\
& FX \to a && (2)\\
& aX \to Xa && (3)\\
& EX \to XaE && (4)\\
& E \to D && (5)\\
& aDb \to bE && (6)\\
& Db \to F && (7)\\
& aD \to Da && (8)\\
& XD \to DX && (9)\\
& bD \to Db && (10)\\
& XC \to CX && (11)\\
& bC \to Db && (12)\\
& Ba \to XB && (13)\\
& B \to C && (14)\\
& Aaba \to bB && (15)\\
& Aa \to aA && (16)\\
& \varepsilon \to A && (17)
\end{aligned}
$$

Schéma S používá pomocné symboly A, B, C, D, E, F a X. Algoritmus má 4 části. Nejprve je vstupní slovo $a^{m+1}ba^{n+1}$ převedeno pravidly (15) – (17) na tvar a^mbBa^n. Ve druhé části (pravidla (11)–(14)) je toto slovo převedeno na tvar a^mDbX^n. Slovo X^n slouží k zapamatování druhého činitele pro potřeby sčítání. Třetí část (pravidla (3)–(10)) realizuje vlastní násobení. Pro každý symbol a vlevo od b je za X^n připsáno slovo a^n tak, že X^n je »zkopírováno«. Přesněji můžeme jeden krok činnosti algoritmu v této části popsat takto: $a^{m-i}DbX^na^{i\cdot n} \vdash a^{m-i-1}bEX^na^{i\cdot n} \vdash \ldots \vdash$

$a^{m-i-1}bX^n a^n Da^{i \cdot n} \vdash \ldots \vdash a^{m-i-1}DbX^n a^{(i+1) \cdot n}$ (pro $i > 0$). Tato část končí aplikací pravidla (6). Poslední část (pravidla (1)—(2)) pak již jen vymaže pomocné X a doplní jeden symbol a, čímž dostaneme výsledný tvar $a^{m \cdot n+1}$.

b) Řešením je schéma (ve zkráceném zápisu, kde $x, y \in \{a, b\}$)

S: Ba	$\to aB$	(1)
Bb	$\to bC$	(2)
Ca	$\to aC$	(3)
Cb	$\to bD$	(4)
Da	$\to aD$	(5)
Db	$\to B$	(6)
B	$\to .\varepsilon$	(7)
C	$\to .\varepsilon$	(8)
D	$\to .\varepsilon$	(9)
Axy	$\to xyA$	(10)
xE	$\to Ex$	(11)
E	$\to B$	(12)
Ax	$\to Ex$	(13)
A	$\to .\varepsilon$	(14)
ε	$\to A$	(15)

Schéma používá pomocné symboly A, B, C, D, E. Nejprve je pravidlem (10) pomocí A po dvojicích znaků procházeno vstupní slovo X. Je-li slovo sudé délky, pak je výsledek čtení tvaru YA, je-li liché délky, pak YAa nebo YAb. Pro lichou délku je pravidly (11)—(13) toto slovo převedeno na tvar BX a tím připraveno k provedení požadované modifikace. Následuje postupné procházení slovem pravidly (1)—(6), přičemž pomocný symbol určuje, kolikáté b (mod 3) je hledáno: B - první, C - druhé, D - třetí. Třetí symbol b

je pravidlem (6) vymazán. Výpočet končí aplikací jednoho z pravidel (7), (8), (9) po zpracování celého slova. Pro slovo sudé délky je pomocné A pravidlem (14) vymazáno a výpočet bezprostředně končí s $S(X) = X$.

P - II - 1

Jsou dána dvě celočíselná pole A, B $(1..N)$ taková, že $N > 2$, a pro všechna i, $1 \geqq i \leqq N$, platí $A(i) \leqq N$, $B(i) \leqq N$, $A(i) \neq B(i)$, $A(i) > i$ nebo $A(i) = 0$, $B(i) > i$ nebo $B(i) = 0$.

Mezi N městy, očíslovanými postupně od 1 do N, je vybudována jednosměrná silniční síť podle těchto pravidel:

1. Pro všechna i, $1 \leqq i \leqq N$, vede přímá silnice z města i do města $A(i)$, jestliže $A(i) > 0$, a z města i do města $B(i)$, jestliže $B(i) > 0$.

2. Žádné jiné silnice nejsou.

Navrhněte a dokažte (nejlépe lineární) algoritmus, který určí, kolika různými způsoby se lze dostat z města 1 do města N.

Poznámka. Algoritmus je lineární, jestliže existuje přirozené číslo L takové, že žádný krok algoritmu se neprovede vícekrát než $L * N$ krát.

Řešení

Z definice silniční sítě vyplývá, že každá přímá silnice vychází z města, jehož číslo je menší než číslo města do něhož vede, a že mezi žádnými městy nevede více než jedna přímá silnice. Čísla měst, kterými projíždíme při libovolné cestě v silniční síti, tvoří tedy vzestupnou posloupnost a do žádného města, kterým jsme projeli, se nemůžeme již vrátit.

Pro všechna i, $1 \leqq i \leqq N$, označme jako $p(i)$ počet cest z města 1 do města i a jako PŘED(i) množinu čísel těch měst, z nichž vede přímá silnice do města i. Definitoricky položme $p(1) = 1$. Z výše uvedených vlastností silniční sítě zřejmě plyne, že pro všechna i, $1 < i \leqq N$, je $p(i)$ rovno součtu všech $p(k)$, pro něž k patří do PŘED(i). Protože pro všechna k z množiny PŘED(i) platí $k < i$, můžeme hodnotu $p(i)$ stanovit, jakmile známe hodnoty $p(k)$ pro všechna k, $1 \leqq k < i$. To nás vede k tomu, že hodnoty $p(i)$ budeme zjišťovat pro města v pořadí $i = 2, 3, \ldots, N$.

Výpočet hodnoty $p(i)$, která je součtem zmíněných sčítanců, budeme průběžně provádět ve složce $c[i]$ pomocného pole $c[1 \ldots N]$ a budeme přitom vycházet ze způsobu zadání silniční sítě v polích $A[1..N]$ a $B[1..N]$.

Jestliže totiž $A[k] = i$ a $i \neq 0$, pak $k \in$PŘED(i), a máme-li již vypočtenou hodnotu $p(k)$, která je jedním ze sčítanců tvořících součet $p(i)$, můžeme ji hned přičíst k průběžnému součtu v $c[i]$. Analogická úvaha platí pro pole B.

Algoritmus navrhneme tak, aby byla zachovávána platnost tvrzení $P(i)$, $1 \leqq i \leqq N$.

$P(i)$: Pro všechna j, $1 < j \leqq N$, platí, že $c[j]$ je součet všech $p(k)$ takových, že $k \in$ PŘED (j) **and** $(k < i)$.

```
for i := 2 to N do c[i] := 0;
c[1] := 1;
{Zde platí c[1] = p(1)}
for i := 1 to N − 1 do
    begin      {zde platí P(i) a c[1] = p(1)}
        if A[i] ≠ 0 then c[A[i]] := c[A[i]] + c[i];
        if B[i] ≠ 0 then c[B[i]] := c[B[i]] + c[i];
```

{zde platí $P(i + 1)$}

end;

{zde platí $P(N) \Rightarrow$ v $c(N)$ je hledaný výsledek}

writeln ($c[N]$).

Z platnosti $P(i)$ a $c[1] = p(1)$ před provedením těla cyklu a z vlastností hodnot $p(j)$ plyne, že pro všechna j, $1 \leqq j \leqq i$, je $c[j] = p(j)$. Přičteme-li tedy známou hodnotu $p(i)$ k těm hodnotám $c[k]$, pro něž $i \in$ PŘED(k), splníme podmínku $P(i + 1)$. Algoritmus je lineární.

P - II - 2

H-sekvence je konečná posloupnost tvořená z 0 a 1 podle těchto pravidel:

1. 0 je H-sekvence,
2. Jsou-li H a H' dvě H-sekvence, pak i spojení $1HH'$ je H-sekvence a
3. žádné jiné H-sekvence nejsou.

Je dáno pole $H(1..N)$ tvořené z 0 a 1, $N \geqq 1$. Navrhněte a dokažte algoritmus, který rozhodne, zda konečná posloupnost $H(1) \ldots H(N)$ je H-sekvence.

Příklad

Posloupnost 1011000 je H-sekvence, neboť vznikla spojením podle bodu 2 ze sekvencí $H = 0$ a $H' = 11000$, při tom H je H-sekvence podle bodu 1, H' je H-sekvence, která vznikla opět podle bodu 2 z $H'' = 100$ a H-sekvence 0. Konečně H'' je spojením podle bodu 2 H-sekvencí 0 a 0.

Řešení

Nejprve si všimněme, že podle druhého pravidla je každá

(netriviální) H-sekvence tvaru $1H_1H_2$. Jestliže chceme zjistit, zda je zadaná posloupnost začínající 1 H-sekvencí, musíme zjistit, zda je zbytek této posloupnosti (po odstranění první 1) zřetězením dvou H-sekvencí. Předpokládejme, že H_1 je opět tvaru $1H_3H_4$. Původní posloupnost bude H-sekvencí, jestliže je zbytek posloupnosti H_1H_2 (opět po odstranění prvního symbolu) zřetězením $H_3H_4H_2$ tří H-sekvencí. Zobecnění těchto úvah nás přivádí k závěru, že úlohu umíme vyřešit, pokud můžeme zjistit, zda je nějaká posloupnost z 0 a 1 zřetězením určitého počtu H-sekvencí. Vzhledem k tomu, že vlastní začátek H-sekvence není H-sekvencí, lze posloupnost z 0 a 1 rozložit nejvýše jedním způsobem do zřetězení H-sekvencí. Počet H-sekvencí je při tom jednoznačně určen prvním symbolem posloupnosti, a sice takto:
(1) Posloupnost tvořená jedinou 0 je H-sekvencí (pravidlo 1). Tedy posloupnost z 0 a 1, která začíná 0, je zřetězením m H-sekvencí, $m \geqq 1$, právě když je zbytek této posloupnosti zřetězením $m - 1$ H-sekvencí.
(2) Posloupnost začínající 1, která je následována dvěma H-sekvencemi, je H-sekvence (pravidlo 2). Proto posloupnost z 0 a 1, která začíná 1, je zřetězením m H-sekvencí, $m \geqq 1$, právě když je zbytek této posloupnosti zřetězením $m + 1$ H-sekvencí. Algoritmus vychází bezprostředně z provedeného rozboru. Jeho idea je následující:
Budeme postupně procházet zadanou posloupností a na základě právě zpracovávaného symbolu určovat, z kolika H-sekvencí musí být zřetězen zbytek. Původní posloupnost bude H-sekvencí, právě když po zpracování celé posloupnosti má následovat nulový počet H-sekvencí.

```
m := 1; n := 0;
while (m ≠ 0) and (n ≠ N) do
      begin n := n + 1;
      if     H(n) = 0  then m := m − 1   {viz (1)}
                       else m := m + 1   {viz (2)}
      end;
if (m = 0) and (n = N) then write ('je H-sekvence')
                       else write ('není H-sekvence').
```

Algoritmus je lineární.

P - II - 3

V urně je m bílých a n černých kuliček, $m, n \geqq 0$, $m + n \geqq 2$. Náhodným způsobem vytáhneme z urny dvě kuličky. Jsou-li stejné barvy, vložíme místo nich do urny jednu černou kuličku; nejsou-li stejné barvy, vložíme místo nich do urny jednu bílou kuličku. Celý postup opakujeme tak dlouho, až v urně zbude jedna kulička. Předpokládejme při tom, že máme k dispozici neomezený počet černých kuliček.

a) Stanovte, po kolika krocích popsaný algoritmus skončí.

b) Pro všechny dvojice m, n takové, že $m + n = 4$, určete barvu poslední kuličky v urně.

c) Určete barvu poslední kuličky v urně v závislosti na m a n.

Řešení

a) Při každém kroku algoritmu odebereme z urny dvě kuličky a vrátíme jen jednu. Při každém kroku dojde tedy ke snížení počtu kuliček v urně o 1. Na počátku je v urně $m + n$ kuliček. Algoritmus končí, jestliže v urně zůstává jedna kulička. Algoritmus tedy končí po $m + n - 1$ krocích.

b), c) Zkcumejme nejprve, jaký vliv mají jednotlivé tahy na počet černých a bílých kuliček v urně:

— jsou-li taženy dvě černé, pak je vrácena jedna černá; počet bílých se nemění a počet černých se zmenší o 1;
— jsou-li taženy dvě bílé, pak se počet bílých sníží o 2 a počet černých se zvětší o 1;
— je-li tažena černá a bílá, pak po tahu je v urně o jednu bílou méně, ale po přidání další bílé (jejím vrácení) je jejich počet opět stejný jako před tahem a počet černých se zmenší o 1.

Z uvedené analýzy vyplývá, že po každém tahu je počet bílých kuliček v urně buď stejný jako před tahem, nebo o 2 menší. Proto poslední kulička v urně je černá, právě když počet bílých kuliček na počátku je sudý.
Odpověď na část b) dostáváme aplikací právě získaného závěru na případ, kdy $m + n = 4$.

P - II - 4

Nechť $P = a_0 \ldots a_n$ je neprázdné slovo. Slovo opačné ke slovu P je slovo $P' = a_n \ldots a_0$.
Navrhněte schéma S, které určí, zda vstupní slovo X, tvořené znaky A, B, C, je tvaru PP' pro nějaké slovo P, tj.

$$S(X) = \varepsilon \text{ právě když } X = PP,$$

pro nějaké slovo P (ε je prázdné slovo).
Řešení
Pokud by byl ve vstupním slovu vyznačen střed, tj. slovo by bylo tvaru PtP', $t \in \{A, B, C\}$, pak by řešením bylo schéma

$$ata \to t, \quad a \in \{A, B, C\}$$
$$t \quad \to .\varepsilon,$$

které od středu postupně maže symetrickou část. Určení středu však nemusí být snadné.

Všimněme si, že slovo X je tvaru PP' právě když slovo XX je tvaru RR', neboť $X = X'$ právě když X je tvaru PP'. Tato skutečnost dovoluje řešit nyní úlohu tak, že vstupní slovo X nejprve převedeme na slovo XtX s vyznačeným středem (»zdvojíme je«), a pak budeme zkoumat symetričnost shora uvedeným způsobem. Ke zdvojení použijeme jako pomocné symboly dvojníky A, B, C k symbolům A, B, C, symbol t k vyznačení středu a symbol z jako ukazatel používaný při procházení slovem. Řešením úlohy je schéma

S: $za \to aaz$ (1)
$ab \to ba$ (2)
$z \to t$ (3)
$at \to ta$ (4)
$ata \to t$ (5)
$t \to \varepsilon$ (6)
$\varepsilon \to z$ (7)

kde $a, b \in \{A, B, C\}$.

Nejprve je aplikováno pravidlo (7), kterým se na začátek slova X připíše symbol z (tj. $X \vdash zX$). Pak je opakovaně aplikováno pravidlo (1) připisující za každý znak jeho dvojníka. Aplikací pravidla (2) se přesunou všechny znaky A, B, C doprava za všechny znaky A, B, C. Zůstává zachováno vzájemné pořadí symbolů ve slově (tj. $zX \vdash \ldots \vdash XXz$). Pravidlo (3) mění z na t a pravidlo (4) mění dvojníky opět v původní symboly a současně vyznačí střed (tj. $XXz \vdash$

$\vdash \ldots \vdash XtX$). Pak je aplikací pravidla (5) prověřována symetričnost. Pravidlo (6) končí činnost algoritmu a právě v případě, že slovo bylo tvaru PP', je výsledkem prázdné slovo.

P - III - 1

Jsou dána dvě celočíselná pole A, $B(1..N)$ taková, že

$$N \geq 1,$$

$A(i) \geq 0$ a $B(i) > 0$ pro všechna i, $1 \leq i \leq N$,

$A(1) + \ldots + A(N) = B(1) + \ldots + B(N)$.

Podél kruhové závodní dráhy je umístěno N nádrží s benzínem, po řadě očíslovaných od 1 do N. Po nádrži s číslem N následuje opět nádrž s číslem 1. V nádrži i je $A(i)$ litrů benzínu a spotřeba benzínu při cestě od nádrže i k následující, tj. $(i \bmod N) + 1$, je $B(i)$ litrů benzínu.

Navrhněte a dokažte algoritmus (nejlépe lineární), který určí všechny nádrže, od kterých může vůz, s počátečně prázdnou a dostatečně velkou nádrží, projet celým okruhem (ve směru daném očíslováním nádrží).

Poznámka 1

$(i \bmod N)$ je zbytek po dělení čísla i číslem N.

Poznámka 2

Algoritmus je lineární, jestliže existuje přirozené číslo L takové, že žádný krok algoritmu se neprovede více než $L * N$.

Řešení

Ukažme nejprve, například matematickou indukcí vzhledem k počtu nádrží N, že existuje alespoň jedna nádrž, od které se dá objet celý okruh.

Pro $N = 1$ to zřejmě platí. Předpokládejme, že tvrzení platí pro $N - 1$ a ukažme, že platí i pro N. Ze zadání plyne, že musí existovat nádrž j, od níž se dá dojet k nádrži $j + 1$. Vyřaďme nádrž $j + 1$ z okruhu tak, že její obsah přidáme do nádrže j (k $A(j)$ přičteme $A(j + 1)$) a upravíme spotřebu benzínu při jízdě od nádrže j k následující (k $B(j)$ přičteme $B(j + 1)$). Máme nyní okruh s $N - 1$ nádržemi splňující podmínky zadání. Podle předpokladu existuje nádrž k, od níž se dá objet celý okruh. Od téže nádrže k se dá ovšem objet i původní okruh s N nádržemi.

Definujeme funkci $S(i, j)$, která udává obsah nádrže automobilu po jízdě od nádrže i k nádrži j, přičemž vozidlo s počátečně prázdnou nádrží natankuje poprvé u nádrže i, poté u každé nádrže, kolem které projíždí, a dojede k nádrži j, aniž by u ní tankovalo.

$$S(i,j) = \begin{cases} \sum_{k=1}^{j-1} (A(k) - B(k)) & \text{pro } i < j \\ \sum_{k=1}^{N} (A(k) - B(k)) + \sum_{k=1}^{j-i} (A(k) - B(k)) & \text{pro } i \geqq j. \end{cases}$$

Funkce (na rozdíl od reality) může nabývat i záporných hodnot. Platí $S(i, j) = S(i, j - 1) + A(j - 1) - B(j - 1)$. Od nádrže i, $1 \leqq i \leqq N$, se dá objet celý okruh právě tehdy, když splňuje podmínku

$d(i)$: pro všechna j, $1 \leqq j \leqq N$ je $S(i, j) \geqq 0$.

Budeme zjišťovat postupně pro nádrže $i = 1, 2, \ldots, N$, zda se od nich dá dojet k nádrži 1. Dá-li se od nádrže i dojet k nádržím $h = i + 1, i + 2, \ldots, j - 1$, kde $i \leqq j \leqq N$, tj. $S(i, h) \geqq 0$ pro všechna h, ale nedá se dojet k nádrži j, tj. $S(i, j) < 0$, pak ani jedna z nádrží h, $i \leqq h \leqq j - 1$,

nesplňuje podmínku $d(h)$ (při jízdě od i k j jsme jimi projížděli s nezáporným obsahem nádrže, a přesto jsme nedojeli) a jako následující budeme prověřovat nádrž j. Tímto způsobem najdeme minimální k takové, že od nádrže k se dá dojet k nádrži 1. Kdyby nádrž k nesplňovala podmínku d(k), nesplňovaly by podmínku d(h) ani nádrže $h = k + 1, \ldots, N$, a protože rovněž nádrže $m = 1, 2, \ldots, k - 1$ nesplňují d(m), byl by to spor s existencí alespoň jedné nádrže splňující podmínku d. Nádrž k tedy splňuje podmínku d(k) a všechny ostatní nádrže h splňující d(h) jsou ty, pro něž $S(k, h) = 0$.

Algoritmus

```
j := 1; k := 1, s: = 0;
while j ≦ N do
{zjišťuj, zda se dá od nádrže k dojet k nádrži l}
  begin s := s + A[j] − B[j]; j := j + 1;
        if s < 0 then begin k := j; s := 0 end
  end;
{od nádrže k se dá objet celý okruh, od 1, 2, ..., k − 1
nikoliv}
writeln(k); s := A[k] − B[k];
for j := k + 1 to N do
    begin if s = 0 then writeln(j);
                s := s + A[k] − B[k]
end;
```

Důkaz správnosti je obsažen v argumentaci pro návrh algoritmu. Algoritmus je lineární.

Je dáno celočíselné pole $A(1, \ldots, N)$, $N \geq 1$. P-sekvence délky k pro pole A je celočíselná posloupnost $p_1, \ldots, p_k$ taková, že $k \geq 1$, pro všechna i, $1 \leq i \leq k$, platí $1 \leq p_i \leq N$ a $p_{i-1} < p_i \leq A(p_{i-1})$.

Navrhněte a dokažte co nejlepší algoritmus, který určí maximální délku P-sekvence pro pole A.

Řešení

Jedním z mnoha způsobů řešení úlohy je nalézt pro každé j, $1 \leq j \leq N$, délku $m(j)$ nejdelší P-sekvence, jejíž poslední člen je j, a z hodnot $m(j)$ vybrat maximální.

Označme jako $P(j, i)$-sekvenci P-sekvenci, jejíž poslední člen je j a ostatní členy p vyhovují nerovnostem $1 \leq p$ a $p < i$. Označme dále jako $d(j, i)$ délku nejdelší $P(j, i)$-sekvence. Zřejmě $d(j, 1) = 1$, $d(j, k) \leq d(j, 1)$ pro $k < 1$ a $d(j, j) = m(j)$ pro všechna j, $1 \leq j \leq N$.

Hodnoty $d(j, 1)$ tedy známe, přirozeným dalším postupem je počítat hodnoty $d(j, i + 1)$ na základě znalostí hodnot $d(j, i)$. Hodnoty $d(j, i)$ budeme ukládat v pomocném poli $c[1..N]$, maximum z již vypočtených hodnot $m(j)$ v proměnné m a algoritmus budeme navrhovat tak, aby při výpočtu byla dodržována platnost tvrzení $Q(i)$:

$c[j] = d(j, i)$ pro všechna j, $\quad 1 \leq j \leq N$
a současně $m = \max\{m(1), m(2), \ldots, m(i)\}$.

První verze algoritmu může tedy vypadat takto:

for $i := 1$ **to** N **do** $c[i] := 1$;
$m := 1$;
{zde platí Q(1)}
for $i :=$ 1 **to** $N - 1$ **do**

```
    begin   {zde platí Q(i)}
        S;  {S je příkaz pro výpočet hodnot d(j, i + 1)}
            {zde platí Q(i + 1)}
    end;
{zde platí Q(N)}
writeln(m)
```

Počítáme-li hodnoty $d(j, i + 1)$ na základě již dříve vypočtených hodnot $d(j, i)$, vyjdeme z těchto faktů

— zřejmě $d(j, i + 1) = d(j, i)$ pro všechna j, $1 \leqq j \leqq i$,

— hodnota $d(j, i + 1)$ se může lišit od $d(j, i)$ jen tehdy, existuje-li P-sekvence s posledním členem j a předposledním členem i. Z definice P-sekvence plyne, že v tom případě musí být $j \leqq A[i]$.

Stačí tedy počítat jen hodnoty $d(j, i + 1)$ pro j splňující nerovnosti $i + 1 \leqq j, j \leqq N, j \leqq A[i]$.

Příkaz S můžeme částečně rozepsat:

```
for j := i + 1 to min(N, A[i]) do
    begin U;
    end;
M;
```

Jedinou $P(j, i + 1)$-sekvencí s délkou větší než $d(j, i)$ může být P-sekvence vytvořená z nejdelší $P(i, i)$-sekvence (její délka je $d(i, i)$) přidáním j jako posledního členu. Vzniklá P-sekvence je $P(j, i + 1)$-sekvencí a její délka je $d(i, i) + 1$. Toto využijeme při dokončení příkazu S.

```
for j := i + 1 to min(N, A[i]) do
    if c[i] + 1 > c[j] then c[j] := c[i] + 1;
{hodnota d(i + 1, i + 1) = m(i + 1) je nyní v c[i + 1]}
```

```
if  c[i + 1] > m then m := c[i + 1];
```

Konečná verze algoritmu:

```
for i := 1 to N do c[i] := 1;
m := 1;
{zde platí Q(1)}
for i := 1 to N − 1 do
    begin      {zde platí Q(i)}
        for j := i + 1 to min(N, A[i]) do
        if c[i] + 1 > c[j] then c[j] := c[i] + 1;
        if c[i + 1] > m then m := c[i + 1];
            {zde platí Q(i + 1)}
    end;
{zde platí Q(N)}
writeln(m).
```

Důkaz správnosti algoritmu byl prováděn současně s konstrukcí algoritmu. Uvedený algoritmus má kvadratickou složitost.

P - III - 3

Je dán kruh s 16 světly a vypínač. Každé světlo se nachází právě v jednom ze dvou stavů - svítí, nesvítí. Jedno otočení vypínačem vyvolá změnu stavu každého světla v závislosti na dosavadních stavech tohoto světla a světla následujícího (ve směru pohybu hodinových ručiček) podle těchto pravidel:

1. Jestliže světlo i jeho následovník svítí, pak světlo svítí dále.
2. Jestliže světlo svítí a jeho následovník nesvítí, pak světlo přestane svítit.

3. Jestliže světlo i jeho následovník nesvítí, pak se světlo rozsvítí.
4. Jestliže světlo nesvítí a jeho následovník svítí, pak světlo nesvítí dále.

Dokažte, že pro libovolné počáteční stavy světel budou nejvýše po 16 otočeních vypínačem svítit všechna světla.

Řešení

Máme dva možné stavy každého světla. Tyto stavy budeme modelovat pomocí pravdivostních hodnot (pravda, nepravda) a změnu stavu jako logickou funkci b argumentů i (číslo světla) a j (pořadové číslo otočení vypínačem), $i, j \geqq 0$, definovanou takto:

$b(i, j) \equiv$ »i-té světlo svítí po j-tém otočení«.

Uvažujme situaci po j otočeních a popišme situaci, která nastane po dalším otočení, tj. vyjádříme hodnotu $b(i, j + 1)$ pomocí hodnot $b(i, j)$ a $b(i + 1, j)$. Platí, že světlo i bude po $j + 1$ otočeních svítit, právě když světla i a $i + 1$ po j otočeních současně obě buď svítila, nebo nesvítila, což vyplývá z pravidel [1] a [2].

Formálně píšeme

$$b(i, j + 1) \equiv [b(i, j) \equiv b(i + 1, j)] . \tag{1}$$

Podobně z pravidel [3] a [4] dostáváme

$$b(i + 1, j) \equiv [b(i, j + 1) \equiv b(i, j)] . \tag{2}$$

Vztahy (1) a (2) zapíšeme (vzhledem k asociativitě logické ekvivalence) do jediného výrazu

$$b(i, j + 1) \equiv b(i, j) \equiv b(i + 1, j) . \tag{3}$$

Naším cílem je ukázat, že $b(i, j + 16)$ je pravda. Za tímto účelem dokážeme obecnější výsledek, a sice ten, že platí

$$H(k): \quad b(i, j + 2^k) \equiv b(i, j) \equiv b(i + 2^k, j) . \qquad (4)$$

Potom, protože $16 \equiv 2^4$ a $b(i, j) \equiv b(i + 16, j)$ je pravda, bude platit, že i $b(i, j + 16)$ je pravda. Vztah (4) dokážeme indukcí. Pro $k = 0$ je $2^0 = 1$ a $H(0)$ platí podle (3).

Nyní předpokládejme, že platí $H(k)$ pro $k > 0$, a dokažme platnost $H(k + 1)$. Podle indukčního předpokladu platí

$$b(i, j + 2^k + 2^k) \equiv b(i, j + 2^k) \equiv b(i + 2^k, j + 2^k) \quad (5)$$

a rovněž i

$$b(i + 2^k, j + 2^k) \equiv b(i + 2^k, j) \equiv b(i + 2^k + 2^k, j). \quad (6)$$

Spojením vztahů (5) a (6) dostáváme

$$b(i, j + 2^{k+1}) \equiv b(i, j + 2^k) \equiv b(i + 2^k, j) \equiv b(i + 2^{k+1}, j) . \quad (7)$$

Protože však

$$b(i, j + 2^k) \equiv b(i, j) \equiv b(i + 2^k, j) ,$$

platí i

$$H(k + 1): \quad b(i, j + 2^{k+1}) \equiv b(i, j) \equiv b(i + 2^{k+1}, j) .$$

P - III - 4

Libovolné přirozené číslo n je reprezentováno v unární soustavě slovem U_n, které obsahuje právě $n + 1$ znaků 1 ($U_0 = 1$, $U_1 = 11$, atd.). Označme B_n reprezentaci přirozeného čísla n v binární soustavě ($B_0 = 0$, $B_1 = 1$, $B_2 = 10$, atd.).

Navrhněte schéma S, které převádí unární reprezentaci přirozeného čísla na reprezentaci binární, tj. pro libovolné přirozené číslo n je

$S(U_n) = B_n$,
a dále navrhněte schéma T, které převádí binární reprezentaci přirozeného čísla na unární, tj. pro libovolné přirozené číslo n je
$T(B_n) = U_n$.
Řešení
a) Navrhneme schéma S, které převádí unární reprezentaci přirozeného čísla na binární. Převod bude realizován postupně, a proto budeme pracovat se slovem tvaru XBY, kde X reprezentuje doposud vytvořenou binární reprezentaci a Y zbývající část unární reprezentace. Začínáme slovem B_0BU_n a postupně pro každý symbol 1 v unární části přičteme jedničku k binární části.

S: $0A \rightarrow 1$ (1)
$1A \rightarrow A0$ (2)
$A \rightarrow 1$ (3)
$B11 \rightarrow AB1$ (4)
$B1 \rightarrow \varepsilon$ (5)
$\varepsilon \rightarrow 0B$ (6)

Schéma používá pomocné symboly A, B. Nejprve je pravidlem (6) převedeno vstupní slovo U_n na $0BU_n$. Pravidlem (4) je započato zpracování jednoho symbolu 1, které spočívá v jeho odstranění a v přičtení jedničky k levé části. Pravidla (1), (3) představují přičtení jedničky bez přenosu do vyšších řádů, (2) s přenosem. Výpočet končí vymazáním posledního symbolu 1 pravidlem (5).
b) Binární reprezentace B_n čísla n je posloupnost $a_m \ldots a_0$ cifer 0 nebo 1 taková, že platí

$$n = a_m.2^m + a_{m-1}.2^{m-1} + \ldots + a_1.2 + a_0 .$$

Po úpravě můžeme číslo n vyjádřit též ve tvaru

$$n = (\ldots(a_m.2 + a_{m-1}).2 + \ldots + a_1).2 + a_0 .$$

Řešením je proto schéma

T: $1X \rightarrow X11$ (1)
$X \rightarrow \varepsilon$ (2)
$C1 \rightarrow X1C$ (3)
$C0 \rightarrow XC$ (4)
$C \rightarrow 1$ (5)
$\varepsilon \rightarrow C$ (6)

Pravidly (3) a (4) je postupně odebírán nejlevější symbol z binární reprezentace a pravidlem (1) je vždy provedeno násobení dvěma, které končí aplikací pravidla (2). Pokud je odebírána cifra 1, pak je současně pravidlem (3) i přičtena jednička. Algoritmus končí aplikací pravidla (5), které doplňuje $(n + 1)$-ní jedničku do unární reprezentace čísla n.