40. ročník matematické olympiády na středních školách

Kategorie P

In: Leo Boček (editor); Jiří Binder (editor); Karel Horák (editor); Václav Sedláček (editor); Pavel Töpfer (editor): 40. ročník matematické olympiády na středních školách. Zpráva o řešení úloh ze soutěže konané ve školním roce 1990/1991. 32. mezinárodní matematická olympiáda. 3. mezinárodní olympiáda v informatice. (Czech). Praha: Státní pedagogické nakladatelství, 1993. pp. 91–129.

Kategorie P

Texty úloh

P – I – 1

V rovině je dáno N bodů očíslovaných od 1 do N. Dvojice čísel $(X[i], Y[i])$ pro $1 \leqq i \leqq N$ reprezentuje kartézské souřadnice bodu i. Robot projde všemi body v pořadí jejich očíslování podle těchto pravidel:

1. Na začátku stojí robot v bodě 1 a dívá se k bodu 2.
2. Robot se pohybuje vždy přímo tím směrem, kterým se dívá.
3. V bodě i pro $1 \leqq i \leqq N$ se robot otočí ve směru pohybu hodinových ručiček o úhel α, $0° \leqq \alpha < 360°$, tak, aby se díval k bodu $(i + 1) \bmod N$.
4. Robot skončí svůj pohyb v bodě 1 tak, že se opět dívá k bodu 2.

Během svého pohybu se robot celkem d-krát úplně otočí kolem své osy. Navrhněte co nejlepší algoritmus, který pro zadaná celočíselná pole $X[1..N]$, $Y[1..N]$ vypočítá hodnotu d. Je povoleno používat jen celočíselné proměnné. Zdůvodněte správnost algoritmu.

P – I – 2

Je dáno celočíselné pole $P[1\,..\,M]$, které obsahuje permutaci čísel 1, 2, ..., M, tj. pro každé i, $1 \leqq i \leqq M$, existuje právě jedno h, $1 \leqq h \leqq M$, takové, že $P[h] = i$. Dále je dáno celočíselné pole $X[1\,..\,N]$. Navrhněte co nejlepší algoritmus, který určí, kolikrát se permutace P vyskytuje v poli X, tj. kolik existuje různých rostoucích celočíselných posloupností $R[1\,..\,M]$ takových, že pro všechna i, $1 \leqq i \leqq$ $\leqq M$,)platí

$$1 \leqq R[i] \leqq N, \qquad P[i] = X\big[R[i]\big].$$

Zdůvodněte správnost navrženého algoritmu.

P – I – 3

Orientovaný graf je dvojice $G = (V, E)$, kde V je konečná množina, jejíž prvky se nazývají vrcholy nebo uzly grafu, a E je binární relace na množině V. Je-li $(x, y) \in E$, říkáme, že v orientovaném grafu vede hrana z vrcholu x do vrcholu y, y se nazývá následník uzlu x a x je předchůdce uzlu y. Cesta délky k v orientovaném grafu je posloupnost v_0, v_1, ..., v_k vrcholů, $k \geqq 0$, taková, že pro všechna i, $0 \leqq i <$ $< k$ je v_i předchůdce v_{i-1}. Říkáme, že vrchol y je dosažitelný z vrcholu x, jestliže existuje cesta z x do y. Poznamenejme, že každý vrchol je dosažitelný sám ze sebe cestou délky 0.

Graf G budeme reprezentovat dvěma celočíselnými poli $B[1\,..\,N+1]$ a $E[1\,..\,M]$, kde N je počet vrcholů a M počet hran v grafu G. Vrcholy jsou očíslovány od 1 do N. Uzel j má $B[j+1] - B[j]$ následníků; jsou zachyceni v prv-

cích $E[B[j]+1], \ldots, E[B[j+1]]$ pole E. Položíme $B[1] = 0$, $B[N+1] = M$.

Cyklus v orientovaném grafu je cesta, jejíž délka je aspoň 1 a v níž je první uzel totožný s posledním uzlem. Graf, který neobsahuje žádný cyklus, se nazývá acyklický.

Navrhněte co nejlepší algoritmus, který pro zadaná celočíselná pole B, E reprezentující orientovaný graf G zjistí, zda graf G je acyklický. Při řešení nepoužívejte rekurzi. Zdůvodněte správnost navrženého algoritmu.

P – I – 4

a) Navrhněte Turingův stroj, který počítá součet dvou přirozených čísel.

b) Navrhněte Turingův stroj, který počítá funkci zdvojení vstupního slova nad abecedou $\{a, b\}$. Pro vstupní slovo P bude tedy výsledkem výpočtu slovo PP.

Turingovy stroje

Abecedou nazveme konečnou neprázdnou množinu Σ. Její prvky nazýváme symboly. Konečnou posloupnost symbolů abecedy Σ nazveme slovem nad abecedou Σ. Prázdnou posloupnost symbolů abecedy, značenou ε, nazveme prázdné slovo. Turingův stroj M nad abecedou Σ má řídicí jednotku, která se může dostávat do konečně mnoha různých stavů a pracuje nad páskou rozdělenou na jednotlivá pole. Na pásce existuje nejlevější pole, směrem doprava je však nekonečná. Každé pole obsahuje vždy jeden symbol páskové abecedy. Pásková abeceda Π je tvořena jednak symboly vstupní abecedy Σ, dále symboly pomocné abecedy V, $\Sigma \cap V = \emptyset$

a speciálním prázdným symbolem $\triangle$, $\triangle \notin \Sigma \cup V$. Tedy $\Pi = \Sigma \cup V \cup \{\triangle\}$.

V každém okamžiku je řídicí jednotka Turingova stroje v právě jednom ze svých možných vnitřních stavů a stroj má přístup k právě jednomu poli pásky prostřednictvím čtecí a zapisovací hlavy. V této situaci stroj provede krok výpočtu následovně: hlava zapíše nový symbol páskové abecedy na pole, nad kterým je umístěna hlava, a tím nahradí původně zapsaný symbol; po tomto zápisu přejde na levé nebo pravé sousední pole. Současně řídicí jednotka může změnit svůj stav. Všechny tyto změny závisí jednak na vnitřním stavu řídicí jednotky, jednak na obsahu čteného pole pásky.

Činnost Turingova stroje je zadána dvojrozměrnou *tabulkou*, která obsahuje pro každý vnitřní stav řídicí jednotky jeden řádek a pro každý symbol páskové abecedy jeden sloupec. Jeden zvolený stav řídicí jednotky je označen jako počáteční a žádný, jeden nebo více stavů jako koncové. Položka tabulky odpovídající stavu i a symbolu s páskové abecedy je buď prázdná, nebo je tvořena trojicí (α, β, γ), kde α je stav, β je symbol páskové abecedy, $\gamma \in \{L, P\}$.

Každá trojice popisuje jeden možný krok *výpočtu* Turingova stroje M: je-li během výpočtu stroj M ve stavu i a hlava je umístěna nad pole pásky se symbolem s, pak stroj přejde do stavu α, hlava zapíše na pásku symbol β do pole, nad nímž se nachází čtecí hlava, a posune hlavu na pásce o jedno pole doleva nebo doprava podle toho, zda $\gamma = L$, nebo P. Na počátku výpočtu je dané vstupní slovo w nad abecedou Σ umístěno zcela vlevo na začátku pásky a všechna zbývající pole vpravo od slova w obsahují prázdný symbol $\triangle$. Hlava je na počátku výpočtu umístěna

nad nejlevějším polem a řídicí jednotka je v počátečním stavu. Výpočet probíhá podle tabulky. Dosáhne-li stroj některého koncového stavu, výpočet končí. Pokud výpočet dojde do nekoncového stavu i, přičemž ve čteném poli pásky je symbol s a položka tabulky odpovídající stavu i a symbolu s je prázdná, pak výpočet rovněž končí. Řekneme, že funkce $f(w_1, w_2, \ldots, w_n)$, $n \geq 1$, zobrazující množinu n-tic slov nad abecedou Σ do množiny slov nad Σ, je počítána Turingovým strojem M nad $\Sigma \cup \{*\}$, kde $* \notin \Sigma$, jestliže pro každé vstupní slovo $w_1 * w_2 * \cdots * w_n$ zadané na pásce se stroj M chová následovně:

1. Je-li funkční hodnota $f(w_1, w_2, \ldots, w_n)$ definována a rovna slovu w, pak výpočet stroje M končí a po skončení je na pásce od začátku zapsáno slovo w, následované pouze prázdnými symboly.
2. Není-li funkční hodnota $f(w_1, w_2, \ldots, w_n)$ definována, pak výpočet stroje M neskončí.

Turingův stroj M může počítat i funkce booleovské, a to takto:

1. Je-li funkční hodnota $f(w_1, w_2, \ldots, w_n) = \mathit{true}$ (pravda), pak stroj M skončí v koncovém stavu. Říkáme také, že vstupní slovo je přijímáno strojem M.
2. Je-li funkční hodnota $f(w_1, w_2, \ldots, w_n) = \mathit{false}$ (nepravda), pak stroj M skončí v nekoncovém stavu. Vstupní slovo není přijímáno.
3. Není-li funkční hodnota $f(w_1, w_2, \ldots, w_n)$ definována, pak výpočet stroje M neskončí.

Turingovy stroje můžeme používat i k práci s přirozenými

čísly. Čísla reprezentujeme např. v unární soustavě v abecedě $\Sigma = \{1\}$ takto:

Číslo n reprezentujeme slovem $\overline{n}$, které je definováno takto:

$$\overline{0} = 1, \qquad \overline{n+1} = \overline{n}1.$$

P – II – 1

Je dáno pole $A[1\,..\,N]$ celých kladných čísel. Je-li hodnota $j = A[i] \leqq N$, $1 \leqq i \leqq N$, znamená odkaz (nepřímou adresu) na prvek $A[j]$ (pro který může platit opět tato podmínka), v opačném případě jde o již uloženou hodnotu. Dále je dáno pole $ZAC[1\,..\,M]$ obsahující hodnoty z $1, \ldots, N$, kde M je mnohem menší než N.

Řekneme, že i-tý prvek pole A je dosažitelný, jestliže se v poli ZAC buď vyskytuje číslo i, anebo j-tý prvek pole A je dosažitelný a $A[j] = i$.

Navrhněte co nejlepší algoritmus, který vypíše indexy všech prvků pole A, které nejsou ze ZAC dosažitelné. Všechny dosažitelné prvky pole A musí zůstat po skončení výpočtu beze změny.

P – II – 2

V rovině je dáno N bodů očíslovaných od 1 do N. Dvojice čísel $(X[i], Y[i])$, $1 \leqq i \leqq N$, reprezentuje kartézské souřadnice bodu i. Žádné tři body neleží v jedné přímce. Napište co nejlepší program, který určí nějakou permutaci $(k_1, k_2, \ldots, k_N)$ čísel 1, 2, ..., N takovou, že N úseček

s krajními body

$$(X[k_i], Y[k_i]),\ (X[k_{i+1}], Y[k_{i+1}])$$

pro $1 \leqq i \leqq N-1$ a

$$(X[k_N], Y[k_N]), (X[k_1], Y[k_1])$$

se navzájem neprotíná.

P – II – 3

Počet všech možností, jak rozdělit N navzájem různých prvků do právě M neprázdných skupin pro $N \geqq M$, je dán hodnotami Stirlingových čísel $\left\{ {N \atop M} \right\}$, pro která platí

$$\left\{ {N \atop 0} \right\} = 0 \quad \text{pro } N \geqq 0, \qquad \left\{ {N \atop N} \right\} = 1 \quad \text{pro } N > 0,$$

$$\left\{ {N \atop M} \right\} = M \cdot \left\{ {N-1 \atop M} \right\} + \left\{ {N-1 \atop M-1} \right\} \quad \text{pro } N > M > 0.$$

a) Navrhněte co nejlepší algoritmus, který pro zadané hodnoty celých čísel N, M, kde $N \geqq M \geqq 0$, vypočte hodnotu Stirlingova čísla $\left\{ {N \atop M} \right\}$.

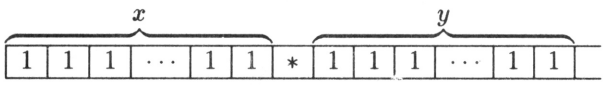
b) Určete minimální počet operací sčítání a násobení mezivýsledků nezbytných pro výpočet čísla $\left\{ {N \atop M} \right\}$ pro dané hodnoty N, M (pro $N \geqq M \geqq 0$).

P – II – 4

a) Navrhněte Turingův stroj nad abecedou $\{1, *\}$, který počítá následující funkci dvou přirozených čísel:

$$x \dot{-} y = \begin{cases} x - y, & \text{je-li } x > y, \\ 0, & \text{je-li } x \leqq y. \end{cases}$$

Čísla x, y jsou zapsána na pásce v unární soustavě a oddělena hvězdičkou, tj.



b) Navrhněte Turingův stroj nad abecedou $\{a, b\}$, který zjistí, zda vstupní slovo na pásce je palindrom.

Poznámka: Palindrom je slovo, které se čte stejně zleva doprava i zprava doleva. Např. řetězce

KOBYLAMAMALYBOK,
NÁNABALILABANÁN

jsou palindromy.

P – III – 1

Je dáno pole $A[1\,..\,N]$ celých kladných čísel. Je-li $j = A[i] \leqq \leqq N$, $1 \leqq i \leqq N$, znamená tato hodnota odkaz (nepřímou adresu) na prvek $A[j]$ (pro ten může platit stejná podmínka), v opačném případě jde o již uloženou hodnotu. Dále je dáno pole $ZAC[1\,..\,M]$ obsahující hodnoty z $\{1, \ldots, N\}$, kde M je mnohem menší než N.

Navrhněte co nejlepší algoritmus, který každý prvek pole ZAC nahradí již přímo indexem prvku pole A s původně referencovanou hodnotou, pokud je možné takovou hodnotu najít. V opačném případě bude prvek pole ZAC obsahovat nulu.

P – III – 2

Legendreův polynom N-tého stupně P je definován vztahy

$$P_0(x) = 1, \qquad P_1(x) = x,$$
$$P_{N+1}(x) = \frac{2N+1}{N+1} P_N(x) - \frac{N}{N+1} P_{N-1}(x)$$

pro všechna reálná x a celá kladná čísla N.

a) S použitím pouze celočíselných proměnných navrhněte co nejlepší algoritmus, který pro zadanou hodnotu N vypočte koeficienty Legendreova polynomu N-tého stupně. Koeficienty musí být vypočteny přesně.

b) Jaký je minimální počet koeficientů polynomů (do stupně N) nezbytných pro výpočet koeficientů polynomu P?

P – III – 3

Je dáno N letišť očíslovaných 1 až N a jejich vzdálenosti v matici $V[1..N, 1..N]$ (prvek $V[i, j]$ udává přímou vzdálenost mezi letišti i a j). Matice V je symetrická. Dále je dána hodnota D určující dolet letadla a přirozené číslo L, $1 \leqq L \leqq N$, označující vybrané letiště.

Navrhněte co nejlepší algoritmus, který sestaví letové trasy z letiště L do všech ostatních, pokud existují. Ze všech možných tras nás však zajímají pouze ty nejkratší, a pokud existuje více tras minimální délky, zajímají nás z nich jen trasy s nejmenším počtem mezipřistání.

P – III – 4

a) Navrhněte Turingův stroj nad abecedou $\{a, b, c\}$, který rozpoznává množinu všech slov $M = \{a^n b^n c^n \mid n > 0\}$.

b) Navrhněte Turingův stroj, který dané slovo nad abecedou $\{a, b\}$ doplní zprava nejkratším způsobem na palindrom.

Příklad:

aba	$\rightarrow$	aba
abaa	$\rightarrow$	abaaba
aab	$\rightarrow$	aabaa

Poznámka. Palindrom je slovo, které se čte stejně zleva doprava i zprava doleva. Např. řetězce NÁNABALILABANÁN, JELENOVIPIVONELEJ jsou palindromy.

Řešení úloh

P – I – 1

Aby mělo zadání úlohy smysl, učiníme dvě dodatečná omezení a jednu změnu:

- $N \geqq 2$
- pro $1 \leqq i \leqq N$ platí
  $(X[i], Y[i]) = (X[i \bmod N + 1], Y[i \bmod N + 1])$
- z bodu i se bude robot dívat do bodu $i \bmod N + 1$

Při určování, kolikrát se robot otočí kolem své osy, není nutné počítat jeho úhel natočení, stačí počítat, kolikrát tento úhel překročí kladný směr (nebo dojde do kladného směru) osy x. Tento počet se zřejmě nezmění, začíná-li robotem v bodě 1 obrácený do kladného směru osy x, neboť při otáčení z tohoto směru do směru k bodu 2 nedojde k překročení kladné osy x.

Směr, ve kterém přišel robot do bodu, v němž se zrovna nachází, reprezentujeme jako vektor, tj. dvojicí celočíselných proměnných (s,t), (pro bod 1 bude na začátku $(s,t) = (1,0)$). Směr, do kterého se v tomto bodě má natočit, reprezentujeme podobně dvojicí (u,v).

Podmínku vyjadřující, zda při otáčení ze směru (s,t) do směru (u,v) mine pohled robota kladný směr osy x, označíme $passx\,(s,t,u,v)$. Pokud (s,t) a (u,v) směřují do stejného kvadrantu, je $passx\,(s,t,u,v) \equiv uv < sv$. Pokud směřují do různých kvadrantů, závisí podmínka *passx* na tom, o kterou kombinaci kvadrantů jde. Rozepsáním všech možností lze ověřit, že *passx* lze obecně vyjádřit takto (za předpokladu $(s,t) \neq (0,0) \neq (u,v)$):

$$\begin{aligned} passx\,(s,t,u,v) \equiv t > 0 \wedge (v \leqq 0 \vee ut < sv) \\ \vee\, t \leqq 0 \wedge v \leqq 0 \wedge ut < sv \\ \vee\, s < 0 \wedge u > 0 \wedge v = 0 \end{aligned}$$

Vlastní algoritmus je jednoduchý:

- Začínáme v bodě 1 s robotem natočeným v kladném směru osy x a vynulujeme si počitadlo d.
- Pro $1 \leqq i \leqq N+1$ provádíme krok: otočíme robota stojícího v bodě $(i-1) \bmod N+1$ tak, aby se díval do bodu $i \bmod N+1$, a pokud přitom došlo k přechodu přes směr (nebo natočení do směru) $(1,0)$, připočteme tento přechod k počitadlu d. Tím považujeme robota za došlého do bodu $i \bmod N+1$.
- Hodnotou proměnné d je hledaný počet úplných otočení robota okolo jeho osy.

Uvedený algoritmus je lineární vzhledem k N.

```
{Algoritmus v pascalu – předpokládáme, že jsou dány
 konstanty N, X, Y}
var s,t,u,v,d,i,j: integer;

function passx (s,t,u,v: integer): boolean;
{passx (s,t,u,v) ⟺ směr (s,t) je různý od směru (1,0)
 a při otáčení ze směru (s,t) do směru (u,v) v záporném
 smyslu přejdeme směr (1,0); předpoklad: (s,t) ≠ (1,0) ≠
 ≠ (u,v)}
begin
  passx := (t > 0) and ((v <= 0) or (u*t < s*v))
           or (t <= 0) and (v <= 0) and (u*t < s*v)
           or (s < 0) and (u > 0) and (v = 0)
end;

begin
  s := 1; t := 0;          {směr, kam hledí robot před túrou}
  d := 0;                                  {nastavení počitadla}
  i := 1; j := 2;                          {počáteční body}
  repeat
    u := X[j] - X[i]; v := Y[j] - Y[i];
                                  {(u,v) je směr k bodu j}
    if passx (s,t,u,v) then d := d + 1;
                          {dokončená otočka ⇒ zvýšíme d}
    {invariant: d je počet úplných otoček robota ko-
     lem své osy od začátku po doražení do bodu i
     a obrácení se k bodu j}
    s := u; t := v;              {nový směr se stane starým}
    i := j; j := j mod N + 1          {neboť robot popoleze}
  until j = 2;   {až už se zas dívá do bodu 2, tak skončil}
  {d je počet úplných otoček robota kolem své osy od
   začátku po návratu do bodu 1 a obrácení se k bodu 2}
```

writeln ('počet obrátek =',*d*)
end.

P – I – 2

Zavedeme pomocné pole $S[0, M]$ a definujeme podmínku $C(j)$: $C(j) \equiv \forall i,\ 1 \leqq i \leqq M$: $S[i]$ je počet výskytů i-tice $(P[1], \ldots, P[i])$ v úseku $X[1], \ldots, X[j-1]$. Jestliže je $S[1] = S[2] = \cdots = S[M] = 0$, pak triviálně platí $C(1)$. Mějme j, $1 \leqq j < N$, a předpokládejme, že $S[0] = 1$ a ostatní prvky pole S mají takové hodnoty, že je splněna podmínka $C(j)$.

Potom, pokud $X[j] \notin \{1, \ldots, M\}$, je splněno také $C(j)$, neboť v úseku $X[1], \ldots, X[j]$ se vyskytuje právě tolik různých částí permutace P jako v úseku $X[1], \ldots, X[j-1]$.

Nechť $X[j] \in \{1, \ldots, M\}$. Tedy $X[j]$ se vyskytuje v permutaci P, řekněme na k-tém místě, tj. $X[j] = P[k]$. Nyní když $l \neq k$, pak počet výskytů l-tic $(P[1], \ldots, P[l])$ v úseku $X[1], \ldots, X[j]$ je tentýž jako v úseku $X[1], \ldots, X[j-1]$, tj. je roven $S[l]$. Všechny výskyty k-tice $(P[1], \ldots, P[k])$ v úseku $X[1], \ldots, X[j]$ se skládají jednak z k-tic, které se celé vyskytují v kratším úseku $X[1], \ldots, X[j-1]$ (těch je $S[k]$), jednak z k-tic $(P[1], \ldots, P[k])$, které vzniknou připojením prvku $X[j]$ ke $(k-1)$-ticím z úseku $X[1], \ldots, X[j-1]$, a těch je $S[k-1]$. (To platí i pro $k = 1$ díky tomu, že jsme položili $S[0] = 1$.) Odtud dostáváme, že když v poli S nahradíme prvek $S[k]$ hodnotou $S[k] + S[k-1]$, pak bude splněna podmínka $C(j+1)$.

Odtud již dostáváme algoritmus. Používá dvě pomocná pole délky M a jeho časová složitost je lineární vzhledem k N.

```
{Algoritmus v pascalu – předpokládáme, že jsou dány
 konstanty M, N, P, X}
var IP: array [1..M] of 1..M;
    {pole pro inverzní permutaci}
    S: array [0..M] of integer;
    {pole pro počítání výskytů prefixů permutace P}
    i, k: 1..M;
    j: 1..N;
begin
  if N < M then writeln ('počet výskytů: 0')
  else begin
    for i := 1 to M do IP[P[i]] := i; {IP je inverzní k P}
    S[0] := 1;
    for i := 1 to M do S[i] := 0;              {inicializace}
    for j := 1 to N do
      {invariant: pro 1 ≦ i ≦ M: S[i] je počet
       výskytů i-tice (P[1],...,P[i]) v posloup-
       nosti X[1], ..., X[j - 1]}
      if (X[j] >= 1) and (X[j] <= M) then begin
        k := IP[X[j]]; S[k] := S[k] + S[k - 1]
      end;
    {pro 1 ≦ i ≦ M: S[i] je počet výskytů
     i-tice (P[1],...,P[i]) v poli X, tj. zejména S[M]
     je rovno počtu výskytů P v X}
    writeln ('počet výskytů: ',S[M])
  end
end.
```

P – I – 3

Nechť $G = (V, E)$ je orientovaný graf, $V' \subseteq V$, $SG(V')$ značí podgraf grafu G, který je indukovaný V, tj. graf, který má V' jako množinu uzlů, a hrana mezi dvěma uzly je, právě když je tato hrana v G. Uvažujme následující podmínku.

$$(SG(U) \text{ je acyklický}) \equiv (G \text{ je acyklický})$$

Tato podmínka platí pro $U = V$. Následující postup tedy inicializujeme pro $U = V$.

Chceme vypouštět uzly z U, aby zůstala zachována platnost podmínky. Jestliže $j \in U$ nemá předchůdce v $SG(U)$, pak nemůže patřit do cyklu v $SG(U)$, a tedy

$$\big(SG(U \setminus \{j\}) \text{ je acyklický}\big) \equiv \big(SG(U) \text{ je acyklický}\big).$$

Totéž platí samozřejmě i o uzlech bez následníků. K odstranění těchto uzlů potřebujeme program pro zjištění předchůdců každého uzlu.

Pro množinu uzlů W a uzel $j \in W$ označíme $P(j, W)$ množinu předchůdců j v $SG(W)$ a $S(j, W)$ množinu následníků j v $SG(W)$. Je-li v P nebo S vynechán argument W, je míněna celá množina V, tj. množina uzlů G. Podmnožina U, která sestává z uzlů j: $P(j, U) = \emptyset$, je množina kandidátů pro vypuštění z U. Označme tuto množinu U_1 a $U_0 = U \setminus U_1$. Pak lze naše podmínky přepsat na $P_0 \wedge P_1$, kde

P_0: $\big(SG(U_0 \cup U_1) \text{ je acyklický}\big) \equiv (G \text{ je acyklický})$
P_1: $\big(\forall j : j \in U_0 : P(j, U_0 \cup U_1) = \emptyset\big) \wedge \big(\forall j : j \in U_1 : P(j, U_0 \cup \cup U_1) = \emptyset\big)$.

Protože postup inicializujeme pro V a jediné uzly, které vypouštíme z $U_0 \cup U_1$, jsou ty bez předchůdců v $U_0 \cup U_1$, platí též P_2:

P_2: $(\forall j : j \in U_0 \cup U_1 : S(j) \subseteq U_0)$.

Po vypuštění uzlu j z U_1 dostáváme $\forall k \in S(j, U_0 \cup U_1)$, že je také $j \in P(k, U_0 \cup U_1)$, a tedy $P(k, U_0 \cup U_1)$ se zmenší o jeden prvek. Každý prvek k, pro který se $P(k, U_0 \cup U_1)$ stane $\emptyset$, je přesunut z U_0 do U_1 (podle P_2 je $k \in U_0$). Dále z P_2 plyne, že pro $j \in U_1$ je $S(j, U_0 \cup U_1) = S(j)$. Opakování ukončíme pro $U_1 = \emptyset$. Z toho dostáváme

$$(SG(U_0) \text{ je acyklický}) \equiv (G \text{ je acyklický}).$$

Z $U_1 = \emptyset$ a P_1 dostáváme

$$(\forall j : j \in U_0 : P(j, U_0) \neq \emptyset),$$

tj. každý uzel v $SG(U_0)$ má předchůdce. A tedy $SG(U_0)$ je acyklický, pouze je-li $U_0 = \emptyset$.

Tato diskuse nás vede k následující struktuře programu:

$U_0 := \{j \mid P(j) \neq \emptyset\}$; $U_1 := \{j \mid P(j) = \emptyset\}$;
while $U_1 \neq \emptyset$ **do begin**
  „*nechť* $j \in U_1$“; $U_1 := U_1 \setminus \{j\}$; „$\forall k \in S(j)$“;
  if $P(k, U_0 \cup U_1) = \emptyset$ **then begin**
    $U_0 := U_0 \setminus \{k\}$; $U_1 := U_1 \cup \{k\}$;
  end
end
acykl $:= (U_0 = \emptyset)$;

Zavedeme celočíselné pole $t(j : 1 \leqq j \leqq N)$, aby platilo:

$$(\forall k : k \in U_0 \cup U_1 : t(k) = |\{j \in U_0 \cup U_1 : j \in P(k)\}|)$$

Pak platnost $P(k, U_0 \cup U_1) = \emptyset$ lze ověřit jako $t(k) = 0$. Množinu U_1 budeme reprezentovat pomocí dvou proměnných v_1 a nv_1, kde v_1 $(j: 0 \leqq j < nv_1)$ je celočíselné pole a nv_1 je celočíselná; kde nv_1 je počet uzlů U_1 a uzly U_1 jsou v poli v_1 $(0 \leqq j < nv_1)$. Pro množinu U_0 je důležitý pouze počet jejích prvků. Budeme ji reprezentovat pomocí proměnné nv_0.

Inicializaci t, v_1, nv_1, nv_0 lze zapsat takto:

```
procedure INIT;
var i, j: integer;
begin
  for j := 1 to N do t[j] := 0;
  for i := 1 to M do t[e[i]] := t[e[i]] + 1;
  nv0 := 0;
  nv1 := 0;
  for j := 1 to N do
    if t[j] = 0 then begin
      v1[nv1] := j; nv1 := nv1 + 1
    end
    else nv0 := nv0 + 1
end;
```

Celá procedura pak vypadá takto:

```
    ⋮
type UZLY:= array [1 .. N + 1] of integer;
     NASL:= array [1 .. N] of integer;
    ⋮
function ACYKL(b: UZLY; e: NASL):boolean;
var nv0, nv1, i, j, k: integer;
```

```
    v1 : array[0 .. N − 1] of integer;
    t: array [1 .. N] of integer;
begin
  INIT;
  while nv1 <> 0 do begin
    j := v1[nv1 − 1]; nv1 := nv1 − 1;
    for i := b[j] + 1 to b[j + 1] do begin
      k := e[i]; t[k] := t[k] − 1;
      if t[k] = 0 then begin
        v1[nv1] := k; nv0 := nv0 − 1; nv1 := nv1 + 1
      end;
    end
  end;
  ACYKL := (nv0 = 0)
end;
```

Správnost algoritmu plyne z postupu na začátku popisu.

Složitost algoritmu. Pokud byl uzel j z U_1 vypuštěn, tak již do U_1 znovu nebude přidán (do U_1 jsou přidávány následníci uzlů a j nemá předchůdce). Tedy každý uzel se testuje maximálně jednou. Celkový počet následníků všech uzlů je celkem M. Vnořené cykly se tedy provedou maximálně M-krát. Celková složitost je $O(M)$.

P – I – 4

a) Čísla x, y budeme reprezentovat v unární soustavě oddělená $*$. Pak se řešení ztotožní s výpočtem zřetězení dvou slov nad abecedou $\{1\}$ a s ubráním jednoho symbolu z výsledného slova. Popis Turingova stroje je tedy ana-

logický jako ve vzorovém příkladu v zadání. Uvedeme proto pouze tabulku:

T	1	$*$	$\triangle$
START 1	$(1,1,P)$	$(1,*,P)$	$(2,\triangle,L)$
2	$(3,\triangle,L)$	$(5,\triangle,L)$	
3	$(4,\triangle,L)$	$(5,\triangle,L)$	
4	$(4,1,L)$	$(5,1,L)$	
STOP 5			

b) Užijeme pomocnou abecedu $V = \{A, B, \alpha, \beta\}$. Pokud je vstupní slovo prázdné, stroj se zastaví. Jinak si zapamatuje první symbol a přepsáním na velké písmeno označí již zkopírovanou část. Dále stroj projde zbytek slova a na jeho konec připíše zapamatovaný symbol, ale v řecké abecedě $\{\alpha, \beta\}$. Dále se stroj vrací vlevo, dokud nenarazí na velké písmeno, to přepíše na malé a pravého souseda si opět zapamatuje, přepíše na velké písmeno a pokračuje předchozím způsobem. Tímto postupem stroj pokračuje do té doby, než při návratu vlevo narazí na sousedící velké a řecké písmeno. V tom okamžiku bylo již zkopírováno celé vstupní slovo a stroj již pouze přepíše toto velké písmeno a všechna řecká písmena na malá latinská. Dále uveďme tabulku pro stroj T.

Správnost činnosti stroje plyne z postupu před tabulkou.

T	a	b	△	A	B	α	β
START 1	$(2, A, P)$	$(3, B, P)$					
2	$(2, a, P)$	$(2, b, P)$	$(4, \alpha, L)$			$(2, \alpha, P)$	$(2, \beta, P)$
3	$(3, a, P)$	$(3, b, P)$	$(4, \beta, L)$			$(3, \alpha, P)$	$(3, \beta, P)$
4	$(5, a, L)$	$(5, b, L)$		$(6, a, P)$	$(6, b, P)$	$(4, \alpha, L)$	$(4, \beta, L)$
5	$(5, a, L)$	$(5, b, L)$		$(1, a, P)$	$(1, b, P)$		
6			$(7, \triangle, P)$			$(6, a, P)$	$(6, b, P)$
STOP 7							

P – II – 1

Algoritmus je jednoduchý. Nejdříve si označíme všechny dosažitelné prvky pole A. Potom projdeme polem A a vypíšeme indexy všech neoznačených, tj. nedosažitelných prvků.

Postup označování dosažitelných prvků je následující: Procházíme pole ZAC a pro každý jeho prvek $ZAC[i]$ označujeme všechny prvky pole A dosažitelné ze $ZAC[i]$. Když při tom narazíme na už označený prvek $A[j]$, (tj. už dříve jsme poznali, že $A[j]$ je dosažitelný), přestaneme další prvky, na něž $A[j]$ případně odkazuje (tj. $A[A[j]]$ atd.), označovat — ty už označené zaručeně jsou — a přejdeme k dalšímu začátku $ZAC[i+1]$.

Dosažitelných prvků je nejvýše N, u nejvýše M z nich testujeme označení dvakrát, u ostatních jen jednou. Algoritmus je tedy lineární vzhledem k N.

Při programové realizaci označování prvků pole A nemusíme zavádět další pole, ale můžeme využít jednak toho, že A je programová proměnná, do níž lze zapisovat, jednak toho, že prvky A jsou celá kladná čísla. Označení prvku $A[j]$ pak realizujeme jeho nahrazením číslem $-|A[j]|$. (Aby byla dodržena poslední podmínka v zadání úlohy, je nutno na konci všechna označení zrušit, tj. nahradit dosažitelné prvky jejich absolutní hodnotou.)

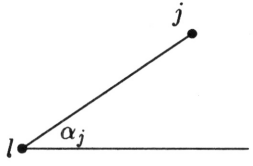
```
{Algoritmus v pascalu}
{předpokládáme, že jsou dány konstanty M, N a naplněna
 pole ZAC, A}
var ZAC: array[1..M] of 1..N;
    A: array[1..N] of integer;
    i, j, K: integer;

begin
  for i := 1 to M do
  {invariant: jsou označeny (tj. záporné) všechny prvky
   pole A dosažitelné ze ZAC[1], ..., ZAC[i - 1]}
  begin
    K := ZAC[i];
    while (1 <= A[K]) and (A[K] <= N) do begin
      A[K] := -A[K];     {označení dosažitelného prvku}
      K := -A[K]                    {posunutí na další}
    end;
    if A[K] >= 0 then A[K] := -A[K]
    {označení dosažitelného prvku, který už není
     ukazatelem}
  end;
  for j := 1 to N do
  if A[j] >= 0 then writeln(j)
                          {výpis nedosažitelných prvků}
  else A[j] := -A[j]          {restaurace dosažitelných}
end.
```

P – II – 2

Při řešení úlohy použijeme následující úvahu. Nechť bod $(X[l], Y[l])$ je takový, že $X[l] = \min\{X[i], i =$

$= 1, \ldots, n\}$, a označíme α_j $(j \neq l)$ úhel určený body $(X[l], Y[l])$, $(X[l] + 1, Y[l])$, $(X[j], Y[j])$ (obr. 25). (Pokud $Y[j] < Y[l]$, pak úhel má znaménko $-$.)



Obr. 25

Nyní setřídíme úhly podle velikosti takto: $\alpha_{j_1} < \cdots < \alpha_{n-1}$. Ostrou nerovnost můžeme předpokládat, protože žádné tři body neleží v jedné přímce. Hledaná permutace bodů je potom takováto:

$$\{l, j_1, \ldots, j_n\}$$

Ukážeme, že pro tuto posloupnost se určené úsečky neprotínají. Nechť se protínají úsečky určené body j_i, j_{i+1}, j_k, j_{k+1}, pak např. $\alpha_{j_i} < \alpha_{j_{i+1}} < \alpha_{j_k} < \alpha_{j_{k+1}}$. Označme s průsečík těchto úseček. Protože žádné tři zadané body neleží na jedné přímce, je bod s vnitřní bod obou úseček. Označíme-li α_s úhel určený analogicky jako úhly α_j, pak platí:

$$\alpha_{j_i} < \alpha_s < \alpha_{j_{i+1}}$$
$$\alpha_{j_k} < \alpha_s < \alpha_{j_{k+1}}$$

To je ale spor. Tedy úsečky se neprotínají.

V programu použijeme následující vlastnost. Abychom nemuseli přímo počítat velikost úhlu, stačí počítat pouze podíl $(Y[l] - Y[i])/(X[l] - X[i])$. Je zde však problém, že jeden bod může mít stejnou x-ovou souřadnici jako $X[l]$. To odstraníme tak, že bod $X[l]$ „zanedbatelně posuneme" doleva (např. 0.0001 pro zadání s přesností 0.1).

Algoritmus má následující postup:

1. Načtení zadání.
2. Zjištění $X[l] = \min\{X[i],\ i = 1, \ldots, n\}$.
3. Výpočet $(Y[i] - Y[l])/(X[i] - X[l] + 0.0001)$.
4. Setřídění uzlů $i = 1, \ldots, n;\ i \neq l$.

Pro paměťové zvýhodnění je hodnota vypočtená v bodě 3 ukládána znovu do pole X. Pro třídění je použito pomocné pole Z. Třídění je vykonáno metodou *Quicksort*. (Viz např. ročenka 36. ročníku MO na středních školách, příklad P–I–2.)

Složitost algoritmu

Body 1, 2, 3 mají složitost $O(n)$, bod 4 $O(n \log n)$. Celková složitost je tedy $O(n \log n)$ za použití jednoho pomocného pole velikosti n.

Zápis algoritmu (v jazyce Pascal)

```
program mnoh;
const u = −1000;
var X, Y : array[1 .. 10] of real;
    Z : array[1 .. 10] of integer;
    i, j, k, l, n : 0 .. 11;
    m : real;
procedure SORT(l, r : integer);
```

```
var i, j, k, w: integer;
begin
  i := l; j := r; k := (l + r) div 2;
  repeat
    while x[z[i]] < x[z[k]] do i := i + 1;
    while x[z[k]] < x[z[j]] do j := j - 1;
    if i <= j then begin
      w := z[i]; z[i] := z[j]; z[j] := w; i := i + 1; j := j - 1
    end
  until i > j;
  if l < j then SORT(l, j);
  if i < r then SORT(i, r)
end;
begin
  writeln ('Zadej počet bodů 1-10'); readln (n);  {načtení}
  writeln ('Zadej body pomocí souřadnic');
  for i := 1 to n do readln (x[i], y[i]);
  m := x[1]; l := 1;
  for i := 2 to n do
    if x[i] < m then begin
      m := x[i];
            {nalezení bodu s nejmenší x-ovou souřadnicí.}
      l := i
    end;
  for i := 1 to n do begin
    x[i] := (y[i] - y[l])/(x[i] - m + 0.0001);
                                          {výpočet hodnot}
    z[i] := i                               {pro srovnání}
  end;
  SORT (1, n);                                {setřídění}
```

```
  writeln ('Výsledná permutace je následující:');
  write (l : 3);
  for i := 1 to n do
    if z[i] <> l then write (z[i] : 3);
  writeln
end.
```

P – II – 3

Nástin algoritmu: Jeden z možných postupů výpočtu $\binom{N}{M}$ je následující (předpokládáme $N \geqq M > 0$, jinak je výpočet triviální).

Označíme $k = N - M$ a zavedeme pomocné pole $P[0..k]$, do něhož na začátku uložíme hodnoty $\binom{1}{1}, \binom{2}{1}, \ldots, \binom{k+1}{1}$, tj. samé jedničky.

V každém dalším (i-tém, pro $2 \leqq i \leqq M$) kroku změníme obsah pole P z hodnot $\binom{i-1}{i-1}, \binom{i}{i-1}, \ldots, \binom{i+k-1}{i-1}$ na hodnoty $\binom{i}{i}, \binom{i+1}{i}, \ldots, \binom{i+k}{i}$. Po M-tém kroku pak bude platit $P[k] = \binom{N}{M}$.

Provedení i-tého kroku: Na začátku i-tého kroku pole P obsahuje prvky

$$\binom{i}{i}, \binom{i}{i-1}, \ldots, \binom{i+k-1}{i-1},$$

neboť $P[0] = \binom{i-1}{i-1} = \binom{i}{i} = 1$. Proto přičtením součinu $i \cdot P[0]$ k prvku $P[1]$ dostaneme $P[1] = \binom{i+1}{i}$. Dalším přičtením $i \cdot P[1]$ k prvku $P[2]$ dostaneme $P[2] = \binom{i+2}{i}$, atd., až nakonec $P[k] = \binom{i+k}{i}$.

Algoritmus v pascalu:

```
{předpokládáme, že jsou dány konstanty N, M, a že N ≧
 ≧ M ≧ 0}
{podmínka C(i,j) je definována níže a slouží jako
 invariant}
const k = N − M;
var i, j: integer;
    P: array[0..k] of integer;
begin
  if M = 0 then writeln ('{', N,' 0} = 0')
  else begin
    for j := 0 to k do P[j] := 1;                {inicializace}
    for i := 2 to M do                                {C(i,1)}
      for j := 1 to k do P[j] := P[j]+i*P[j−1]; {C(i,j)}
    {C(M + 1,1)}
    writeln ('{', N, ' ',M ,'} = ', P[k])
  end
end.
```

Důkaz: Pro $2 \leqq i \leqq M+1$, $1 \leqq j \leqq k+1$ definujeme podmínku

$$C(i,j) \equiv \forall r, \quad 0 \leqq r < j\colon P[r] = \left\{ \begin{matrix} i+r \\ i \end{matrix} \right\}$$

a současně

$$\forall s, \quad j \leqq s \leqq k\colon P[s] = \left\{ \begin{matrix} i+s-1 \\ i-1 \end{matrix} \right\}.$$

Nyní lze snadno ověřit tato fakta:

a) Po provedení prvního — inicializačního kroku (tj. na začátku druhého kroku) platí $C(2,1)$.

b) Platí-li $C(i,j)$ pro určité i, j, $2 \leqq i \leqq M$, $1 \leqq j \leqq k$, potom po provedení přiřazení $P[j] := P[j] + i * P[j-1]$ bude platit $C(i, j+1)$.

c) Pro $2 \leqq i \leqq M$ je $C(i, k+1) \equiv C(i+1, 1)$.

d) Užitím bodu (b) a indukce vzhledem k j máme, že platí-li $C(i,1)$ na začátku i-tého kroku, $2 \leqq i \leqq M$, (před provedením vnitřního cyklu), pak po jeho skončení bude platit $C(i, k+1)$, tedy, podle (c), bude platit $C(i+1, 1)$.

e) Užitím bodů (a), (d) a indukce vzhledem k i dostáváme, že na začátku každého i-tého kroku ($2 \leqq i \leqq M$) platí $C(i,1)$ a na konci M-tého kroku platí $C(M+1, 1)$.

Po provedení celého algoritmu tedy platí $C(M+1, 1)$. Odtud podle definice podmínky C máme $P[k] = \left\{ {N \atop M} \right\}$.

Počet aritmetických operací na mezivýsledcích: Jediné sčítání a jediné násobení mezivýsledků se provádí ve vnitřním cyklu algoritmu. Počet provedení vnitřního cyklu je k, počet provedení vnějšího cyklu je $M-1$. Vidíme tedy, že celkový počet sčítání mezivýsledků je tentýž jako počet násobení a je roven $(M-1) \cdot k = (M-1) \cdot (N-M)$.

P – II – 4

a) Čísla jsou reprezentována v unární soustavě, oddělená $*$. Nechť $|x|$, resp. $|y|$ značí počet jedniček v zápisu x, resp. y. Řešení plyne z následujících faktů. Nechť $|x| = k$, $|y| = l$, pak $|x \dot{-} y| = k - l + 1$, je-li $k \geqq l$, resp. $|x \dot{-} y| = 1$, je-li $k < l$.

Postup řešení je následující. Nejprve se vstupní slovo na pásce přepíše tak, že bude začínat 0, která bude indikovat

pravý okraj pásky. Pak se postupně jedničky v prvním čísle nahradí $*$, dokud není

a) $*$ tolik, jako bylo cifer v 2. čísle, nebo
b) se nenahradila 0.

Na závěr se přidá jedna 1 a $*$ se smažou. Postup lze zapsat do této tabulky:

T	1	$*$	$\triangle$	0
START 1	$(2, 0, P)$			
2	$(2, 1, P)$	$(3, 1, P)$		
3	$(4, *, P)$			
4	$(4, 1, P)$		$(5, 1, P)$	
5	$(6, \triangle, L)$		$(5, \triangle, L)$	
6	$(6, 1, L)$	$(7, *, L)$		
7	$(8, *, P)$	$(7, *, L)$		$(10, 1, P)$
8	$(9, 1, P)$	$(8, *, P)$	$(11, \triangle, L)$	
9	$(9, 1, P)$		$(5, \triangle, L)$	
10		$(10, \triangle, P)$		
11	$(11, 1, L)$	$(11, \triangle, L)$		$(11, 1, P)$

b) Užijeme pomocnou abecedu $V = \{A, B\}$. Pokud je vstup prázdný, odpověď je ano. Jinak si stroj zapamatuje první písmeno vstupu, změní ho na velké z důvodu již ověřeného označení. Dále se pročte zbytek slova a porovná se zapamatované písmeno s posledním. Pokud nejsou stejná, slovo není palindrom a odpověď je ne. Jinak se poslední písmeno přepíše na velké a ověřuje se platnost palindromu pro zbytek slova. Tento postup lze napsat do následující tabulky:

T	a	b	△	A	B
START 1	$(2, A, P)$	$(3, B, P)$	$(8, \triangle, P)$	$(8, A, P)$	$(8, B, P)$
2	$(2, a, P)$	$(2, b, P)$	$(4, \triangle, L)$	$(4, A, L)$	$(4, B, L)$
3	$(3, a, P)$	$(3, b, P)$	$(5, \triangle, L)$	$(5, A, L)$	$(5, B, L)$
4	$(6, A, L)$	$(7, b, L)$	$(7, \triangle, L)$	$(8, A, L)$	$(8, B, L)$
5	$(7, a, L)$	$(6, B, L)$	$(7, \triangle, L)$	$(8, A, L)$	$(8, B, L)$
6	$(6, a, L)$	$(6, b, L)$		$(1, A, P)$	$(1, B, P)$
7					
STOP 8					

P – III – 1

Zavedeme si následující pojem: Koncovým indexem prvku $A[j]$ (pro $1 \leqq j \leqq N$) nazveme číslo j, pokud $A[j] > N$; jestliže $1 \leqq A[j] \leqq N$ a prvek $A[A[j]]$ má koncový index, pak tento index bude také koncovým indexem prvku $A[j]$; v ostatních případech položíme koncový index prvku $A[j]$ roven nule.

Koncovým indexem prvku $ZAC[i]$ (pro $1 \leqq i \leqq M$, $1 \leqq \leqq ZAC[i] \leqq N$) rozumíme koncový index prvku $A[ZAC[i]]$.

Úkolem algoritmu tedy bude nahradit každý prvek pole ZAC jeho koncovým indexem. Snadno nahlédneme následující tvrzení: Jestliže v poli A libovolný prvek nahradíme jeho koncovým indexem, pak koncové indexy všech prvků polí ZAC a A zůstanou stejné jako před záměnou.

Hrubá idea algoritmu: Procházíme postupně prvky $ZAC[1], \ldots, ZAC[M]$ a v každém kroku nahradíme ty prvky pole A, jež jsou dosažitelné ze $ZAC[i]$, jejich koncovým indexem. Ten je také koncovým indexem prvku $ZAC[i]$, takže ho zapíšeme i do $ZAC[i]$.

Při zjišťování koncových indexů postupujeme podobně jako v úloze P–II–1, případné zacyklení odkazů (tj. případ, kdy koncový index je nula) si hlídáme tak, že si průběžně

označujeme (obrácením znaménka) prvky pole A dosažitelné ze $ZAC[i]$.

```
{Algoritmus v pascalu}
{předpokládáme, že jsou dány konstanty M, N a naplněna
 pole ZAC, A}
var ZAC: array[1..M] of 0..N;
    A: array[1..N] of integer;
    i, k, L, P: integer;

begin
  for i := 1 to M do                                  {„inv“}
  begin
    k := ZAC[i];
    while (1 ≦ A[k]) and (A[k] <= N) do begin
      A[k] := -A[k];                {označení prvků pole A}
      k := -A[k]                    {dosažitelných ze ZAC[i]}
    end;
    if A[k] <= 0 then P := 0
    else P := k;        {P = koncový index prvku ZAC[i]}
    k := ZAC[i]; ZAC[i] := P;
                    {nahrazení ZAC[i] koncovým indexem}
    while A[k] < 0 do                {nahrazení všech prvků}
    begin
      L := k;                  {pole A z původního ZAC[i]}
      k := -A[k];                        {dosažitelných jejich}
      A[L] := P                          {koncovým indexem}
    end
  end
end.
```

V místě označeném „inv“ platí invariant:

- prvky $ZAC[1], \ldots, ZAC[i-1]$ jsou svými koncovými indexy,
- všechny prvky pole A, které byly dosažitelné z původních hodnot $ZAC[1], \ldots, ZAC[i-1]$, jsou svými koncovými indexy,
- ostatní prvky polí mají své původní hodnoty,
- koncové indexy všech prvků polí ZAC a A jsou stejné jako koncové prvky jejich původních hodnot,

V prvním vnitřním cyklu hledáme koncový index prvku $ZAC[i]$. Jsou-li odkazy v poli A cyklické, bude koncový index roven nule (příkaz **if** za cyklem). Koncový index se zapíše do $ZAC[i]$. Přiřazení tohoto indexu prvkům pole A (druhý vnitřní cyklus) není nutné z hlediska správnosti algoritmu, ale je důležité z hlediska jeho časové složitosti.

Je-li vyšetřován nějaký prvek $A[k]$, je počet jeho zpřístupnění roven řádově číslu M. V prvním vnitřním cyklu je opakovanému procházení prvku zabráněno tím, že si hlídáme cyklení odkazů, v druhém vnitřním cyklu se prvek $A[k]$ nahradí svým koncovým indexem a díky tomu na něm každé další provedení prvního **while**-cyklu ihned skončí. Odtud vyplývá časová složitost algoritmu $O(M+N)$, což (vzhledem k podmínce $M < N$) je rovno $O(N)$. Algoritmus je tedy lineární vzhledem k N.

P – III – 2

Koeficienty Legendreových polynomů jsou racionální čísla, která v paměti můžeme reprezentovat dvojicemi celých čísel (čitatel, jmenovatel). Pro účely generování koeficientů je

vhodné čitatele uchovávat v jednorozměrném poli $A[0\,..\,N]$, v prvku $A[j]$ je čitatel koeficientu u j-té mocniny.

Polynom P je pro sudé, resp. liché i sudou, resp. lichou funkcí, tj. jeho liché, resp. sudé koeficienty jsou nulové. Proto lze v poli A uchovávat naráz čitatele koeficientů polynomů P_i, P_{i-1}. Pro každý polynom je uchován společný jmenovatel jeho koeficientů tak, aby aspoň 1 koeficient tohoto polynomu byl v základním tvaru.

Algoritmus může tedy pracovat takto:

- Do pole A umístíme čitatele koeficientů polynomů P_0, P_1 a do proměnných *ds*, *dl* umístíme hodnoty jmenovatelů.
- Pro $i = 2, \ldots, N$ opakujeme:
  - vypočteme koeficienty polynomu P_i a přepíšeme jimi koeficienty polynomu P_{i-2};
  - provedeme možné krácení hodnot vypočtených koeficientů polynomu P_i.
- Vypíšeme koeficienty polynomu P_N, tj. sudé nebo liché prvky pole A, a hodnotu *ds* nebo *dl* podle toho, zda N je sudé, nebo liché.

Pro zjednodušení algoritmu je pole A indexováno od -1, přičemž $A[-1]$ je vždy nulové.

```
{Algoritmus v pascalu – předpokládáme, že je dána konstanta N}
var A: array[−1 .. N] of integer;
    {čitatele koeficientů polynomu}
    ds, dl: integer;{jmenovatele koeficientů polynomu}
    i, j, NN1, N1, M, NSD: integer;
begin
  A[−1] := 0; A[0] := 1; A[1] := 1;
```

```
for j := 2 to N do A[j] := 0; ds := 1; dl := 1; NN1 := 1;
for i := 2 to N do
        {výpočet čitatelů koeficientů polynomu stupně i}
begin
  NN1 := NN1 + 2; N1 := i - 1; j := i;
  repeat
    if ds = dl then A[j] := NN1 * A[j - 1] - N1 * A[j]
    else A[j] := NN1 * A[j - 1] - N1 * N1 * A[j];
    j := j - 2;
  until j < 0;
  if i mod 2 = 0 then begin
    ds := dl * i; NSD := ds
  end
  else
  begin
    dl := ds * i; NSD := dl
  end;
  {zjištění NSD vypočtených hodnot}
  j := i;
  repeat
    M := abs(A[j]);
    while M <> NSD do begin
      while NSD < M do M := M - NSD;
      while NSD > M do NSD := NSD - M
    end;
    j := j - 2;
  until (J < 0) or (NSD = 1);
  {krácení koeficientů}
  if NSD <> 1 then begin
    J := 1;
```

```
        repeat
          A[j] := A[j] div NSD; j := j - 2;
        until j < 0;
        if i mod 2 = 0 then ds := ds div NSD
        else dl := dl div NSD;
      end
    end;
    { výpis koeficientů }
    if N mod 2 = 0 then writeln ('jmenovatel:', ds)
    else writeln ('jmenovatel:', dl);
    j := N;
    repeat
      writeln ('mocnina', j, ':koeficient :', A[j]); j := j - 2;
    until j < 0;
end.
```

P – III – 3

Řešení úlohy převedeme jednoduchou úvahou na řešení klasického grafového problému hledání nejkratších cest z daného uzlu grafu do všech ostatních. Pro řešení tohoto problému pak použijeme klasický Dijkstrův algoritmus. Naši úlohu převedeme tak, že z matice V vypustíme hrany delší než dolet letadla. Nyní můžeme již použít Dijkstrův algoritmus pro určení $d[v]$, nejkratší vzdálenosti z u do v pro graf G:

Pomocné proměnné

- M, množina vrcholů, pro které ještě není vzdálenost $d[v]$ definitivně určena
- V, vektor, kde pro každý uzel v je uveden předposlední uzel nejkratší cesty z u do v a neprochází M

– L, matice přímé vzdálenosti

Algoritmus

1. Inicializace. $M = V(G) \setminus \{u\}$; $d[u] = 0$; $V[u] = u$; pro $v \neq u$: $d[v] = L[u, v]$ a je-li $d[v] < \infty$, $V[v] = u$.
2. Test ukončení. Je-li M prázdná, výpočet končí.
3. Určení $d[v]$ pro další uzel. Z uzlů množiny M vybereme uzel v s minimální hodnotou $d[v]$. Je-li $d[v] = \infty$, pak výpočet končí, jinak vyjmeme v z M.
4. Aktualizace d a V. Je-li v uzel vybraný v bodě 3, pak pro každý uzel $w \in M$: pokud $d[v] + L[v, w] < d[w]$, provedeme $V[w] = v$, $d[w] = d[v] + L[v, w]$.
5. Skok do bodu 2.

Vlastní program v jazyce Pascal:

```
program LET;
label 5,10;
var M: set of 1..20;
    L: array[1..20,1..20] of integer;
    i,j,k,n,u: 0..20;
    v,w: integer;
begin
  w := 0;
5: writeln ('Zadej dolet letadla');
  readln (v);
  writeln ('Zadej počet letišť [1–20]');
  readln (n);
  writeln ('Zadej matici vzdálenosti ');
  writeln ('[pokud není hrana –1]');
  for i := 1 to n do begin
    for j := 1 to n do begin
```

```
        read (L[i, j]);
        if L[i, j] > v then L[i, j] := −1;
                        {Odstranění hran přesahujících dolet}
      end;
      readln
    end;
    writeln ('Zadej letiště, ze kterého se určují trasy:');
    readln (u); M := [1 .. n];
    for i := 1 to n do
      if L[u, i] ≧ 0 then L[i, u] := u;                    {* 1 *}
    M := M − [u];
    while M <> [] do begin
      w := n * v + 1; j := 0;
      for i := 1 to n do
        if i in M then
          if (L[u, i] >= 0) and (L[u, i] <= w) then begin
            w := L[u, i]; j := i                           {* 3 *}
          end;
      if j = 0 then goto 10;                               {* 4 *}
      M := M − [j];
      for i := 1 to n do
        if i in M then
          if (L[j, i] >= 0) and
          ((w + L[j, i] < L[u, i]) or (L[u, i] < 0)) then begin
            L[i, u] := j; L[u, i] := w + L[j, i]           {* 5 *}
          end;
    end;
10: L[u, u] := 0;
    writeln (' ':27,'Trasy z letiště',u:3); writeln ;
    write (' ':10,'letiště',' ':10,'vzdálenost');
```

```
    write (' ':10,'nejkratší trasa'); writeln;

    for i := 1 to n do begin
      v := L[u, i];
      if v >= 0 then begin
        k := i; j := 1;
        while k <> u do begin
          L[j, i] := L[k, u]; k := L[k, u];                    {* 6 *}
          j := j + 1
        end;
        write (' ':11,i : 2);
        write (' ':14,v : 3,' ':14);

        for k := j - 1 downto 1 do write(L[k, i]:2,',');
        write (i : 2); writeln

      end
      else
      begin
        write (' ':11,i : 2);
        write (' ':12,'nekonečno');
        write (' ':11,'neexistuje');
        writeln
      end
    end;
    writeln ('Opakovat výpočet [0/1]'); readln (j);
    if j = 1 then goto 5;
end.
```

*** K O M E N T Á Ř E ***

* 1 * : Odstranění hran přesahujících dolet

* 2 * : Ztotožnění u-tého sloupce matice L s vektorem V a u-tého řádku a d
* 3 * : Nalezení minima $d[v]$ pro $v \in M$
* 4 * : Neexistuje zlepšení
* 5 * : Provedení zlepšení
* 6 * : Rekonstrukce letové trasy

Časová složitost: Algoritmus zpracuje vstup n letišť v čase $O(n)$.

Správnost algoritmu: Dokážeme pouze pro Dijkstrův algoritmus. Indukcí ukážeme, že pro v vybraný v bodě 3 algoritmu je $d[v] = d_{uv}$, tj. délka nejkratší cesty. Nechť předpoklad platí pro uzly z $V(G) \setminus M$. Nechť v je uzel vybraný v bodě 3 algoritmu a nechť existuje cesta P z u do v tak, že délka $d_P < d[v]$. Nechť x je první uzel z P, který leží v M a y jeho předchůdce v P. Pak $d[x] \leqq d[y] + L[y,x] = = d_{uy} + L[y,x] \leqq d_P < d[v]$, což je spor s minimalitou $d[v]$.

P – III – 4

a) Užijeme pomocnou abecedu $V = \{A, B, C\}$. Pokud je slovo prázdné, pak stroj odpoví *ano*. Jinak přečte první symbol slova. Pokud není tento symbol a, pak slovo není přijímáno. Pokud je symbol a, stroj ho přepíše na A a hledá k němu b. Pokud jej nalezne, přepíše ho na B a hledá c, které přepíše na C. Tento postup se opakuje, dokud stroj neskončí úspěšně, či neúspěšně.

T	a	b	c	△	A	B	C
START 1	$(2, A, P)$			$(5, \triangle, P)$	$(1, A, P)$	$(1, B, P)$	$(1, C, P)$
2	$(2, a, P)$	$(3, B, P)$				$(2, B, P)$	
3		$(3, b, P)$	$(4, C, L)$				$(3, C, P)$
4	$(4, a, L)$	$(4, b, L)$	$(4, c, L)$		$(1, A, P)$	$(4, B, L)$	$(4, C, L)$
STOP 5							

b) Užijeme pomocnou abecedu $V = \{A, B, \alpha, \beta\}$. Pokud je vstup prázdný, je slovo palindrom. Jinak stroj postupně testuje postfixy vstupního slova, dokud nenarazí na palindrom. Pro testování se používá stroj z příkladu v krajském kole (stavy 2, 3, 4, 5, 6, 7). Pokud postfix není palindrom vloží se jeho první znak mezi vstupní slovo a již připsanou část vpravo a ta se posune o jeden znak (stavy 8, 12, 13, 14, 15, 16, 17). Začátek testovaného postfixu a začátek již doplněné části je označen řeckými písmeny. Vždy po ukončení testování palindromu se velká písmena přepíší na malá. Po zjištění, že postfix je palindrom, se přepíší všechna písmena na malá.

T	a	b	△	A	B	α	β
START 1	$(3, \triangle, P)$	$(4, \triangle, P)$	$(9, \triangle, P)$				
2	$(3, A, P)$	$(4, B, P)$		$(9, A, P)$	$(9, B, P)$	$(9, \triangle, P)$	$(9, \triangle, P)$
3	$(3, a, P)$	$(3, b, P)$	$(5, \triangle, L)$	$(5, A, L)$	$(5, B, L)$	$(5, \triangle, L)$	$(5, \triangle, L)$
4	$(4, a, P)$	$(4, b, P)$	$(6, \triangle, L)$	$(6, A, L)$	$(6, B, L)$	$(6, \triangle, L)$	$(6, \triangle, L)$
5	$(7, A, L)$	$(8, b, L)$		$(9, A, P)$	$(9, B, P)$	$(9, \triangle, P)$	$(9, \triangle, P)$
6	$(8, a, L)$	$(7, B, L)$		$(9, A, P)$	$(9, B, P)$	$(9, \triangle, P)$	$(9, \triangle, P)$
7	$(7, a, L)$	$(7, b, L)$		$(2, A, P)$	$(2, B, P)$	$(2, \triangle, P)$	$(2, \triangle, P)$
8	$(8, a, L)$	$(8, b, L)$		$(8, A, L)$	$(8, B, L)$	$(12, \triangle, P)$	$(15, \triangle, P)$
9	$(9, a, P)$	$(9, b, P)$	$(10, \triangle, L)$	$(9, a, L)$	$(9, b, P)$	$(10, a, L)$	$(10, b, L)$
10	$(10, a, L)$	$(10, b, L)$		$(10, a, L)$	$(10, b, L)$	$(11, a, P)$	$(11, b, P)$
STOP 11							
12	$(12, a, P)$	$(12, b, P)$	$(16, \triangle, P)$	$(12, a, P)$	$(12, b, P)$	$(13, \triangle, P)$	$(14, \triangle, P)$
13	$(13, a, P)$	$(14, a, P)$	$(16, a, L)$				
14	$(13, b, P)$	$(14, b, P)$	$(16, b, L)$				
15	$(15, a, P)$	$(15, b, P)$	$(16, \triangle, P)$	$(15, a, P)$	$(15, b, P)$	$(13, \triangle, P)$	$(14, \triangle, P)$
16	$(16, a, L)$	$(16, b, L)$	$(16, \triangle, L)$			$(17, \triangle, L)$	$(17, \triangle, L)$
17	$(17, a, L)$	$(17, b, L)$		$(17, a, L)$	$(17, b, L)$	$(1, a, P)$	$(1, b, P)$