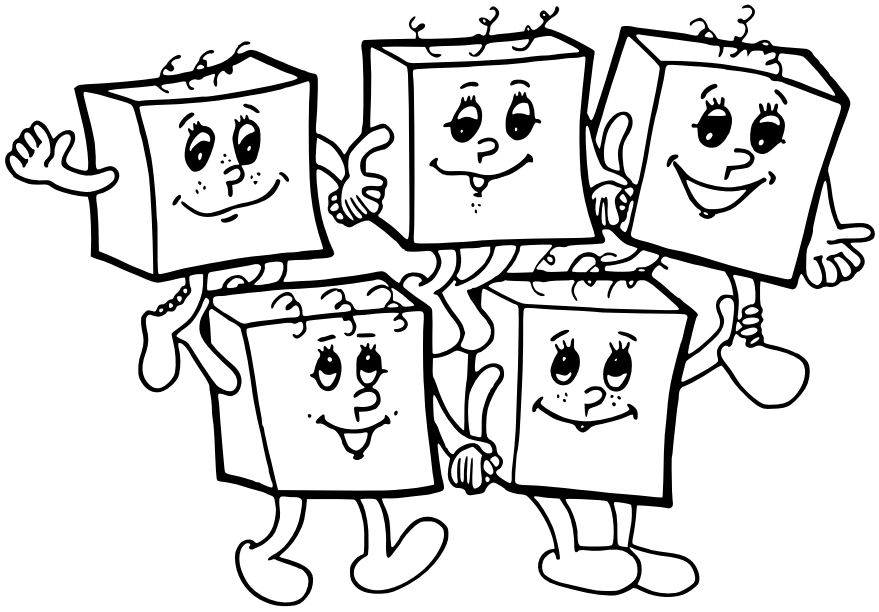
OLYMPIÁDA V INFORMATIKE
NA STREDNÝCH ŠKOLÁCH



dvadsiaty druhý ročník
školský rok 2006/07

riešenia celoštátneho kola
kategória A

1. súťažný deň

- **Olympiáda v informatike** je od školského roku 2006/07 samostatnou súťažou. Predchádzajúcich 21 ročníkov tejto súťaže prebiehalo pod názvom **Matematická olympiáda, kategória P** (programovanie).
- Oficiálnu **webstránku** súťaže nájdete na `http://www.ksp.sk/oi/`.

Riešenia prvého súťažného dňa

A-III-1 Pizza vracia úder

Efektívne riešenia tejto úlohy budú založené na metóde dynamického programovania. Zaujímať nás budú hodnoty $D(s,p)$ definované nasledovne: $D(s,p)$ je 1 (true, pravda), ak sa dá dosiahnuť výsledná suma s použitím prvých p položiek zo vstupu. Riešením zadanej úlohy je zjavne najmenšie nezáporné s také, že $D(s,N)$ je pravda.

Základné riešenie

Nech S je súčet všetkých položiek a_i zo vstupu. Zjavne žiadny dosiahnuteľný výsledok nebude v absolútnej hodnote väčší ako S. Preto sa stačí obmedziť na hodnoty $D(s,p)$, kde $s \in \{-S,\ldots,S\}$.

Hodnoty $D(s,p)$ vieme počítať pomocou nasledujúcich vzťahov:

$$\begin{aligned} D(s,0) &\stackrel{def}{=} [s=0] \\ D(s,p) &\stackrel{def}{=} D(s-a_p,p-1) \ \vee \ D(s+a_p,p-1) \end{aligned}$$

Slovne: Pomocou 0 položiek vieme vyrobiť len sumu 0. Pomocou p položiek vieme sumu s vyrobiť práve vtedy, ak vieme pomocou $p-1$ položiek vyrobiť aspoň jednu zo súm $s-a_p$ a $s+a_p$.

Takéto riešenie má časovú zložitosť $O(SN)$. Keďže každé a_i je najviac K, vieme, že $S \leq KN$, a teda časová zložitosť tohoto algoritmu je $O(KN^2)$.

V nasledujúcich častiach budeme toto riešenie postupne zlepšovať.

Kompresia vstupných údajov

Všimnime si, že výsledok nezáleží na poradí položiek na vstupe, iba na tom, koľkokrát sa ktoré číslo z intervalu 1 až K vyskytuje v Marcovej postupnosti. Označme n_i počet výskytov čísla i. Úlohu si teraz môžeme preformulovať nasledovne: hľadáme K čísel x_i (kde $0 \leq x_i \leq n_i$) takých, že pokiaľ x_i výskytov

čísla i prehlásime za príjem a $n_i - x_i$ za výdaj, zisk bude nezáporný a najmenší možný.

Veľkosť výsledku

Riešenie úlohy je menšie ako $2K$. Ak by totiž mal byť výsledný zisk $2K$ alebo ešte viac, môžeme zmeniť niektoré + na −, čím zmenšíme zisk a ešte neklesneme do záporu.

Ako na veľké počty položiek

Predstavme si napríklad situáciu, kedy $K = 5$ a Marco má na zozname milión položiek s hodnotou 3. Intuícia hovorí, že nič nepokazí, ak zhruba polovicu z nich dá ako prijmy a druhú polovicu ako výdaje.

Ak by sme takéto niečo vedeli poriadne sformulovať a dokázať, dosť silne by to nášmu riešeniu pomohlo.

Ako by takéto tvrdenie mohlo vyzerať? Nejako takto: „Nech je nejaké n_i **dosť veľké**, potom existuje optimálne riešenie, kde nemajú všetky položky veľkosti i rovnaké znamienka.“

Ak takéto tvrdenie dokážeme, vieme ho nasledovne využiť: „Nech je nejaké n_i **dosť veľké**. Potom existuje optimálne riešenie, v ktorom je aspoň jeden výskyt i ako príjem a jeden ako výdaj. Tieto dva výskyty môžeme teda vynechať. To znamená, že ak v pôvodnej úlohe zmenšíme n_i o 2, optimálne riešenie sa tým nezmení.“

Zostáva prísť na to, čo máme dosadiť namiesto „**dosť veľké**“, aby to platilo.

Predstavme si, že už máme nejaké optimálne riešenie. Nech p je veľkosť položky, ktorá nás zaujíma. Jediné zaujímavé situácie sú keď $x_p = 0$ alebo $x_p = n_p$, teda keď majú všetky výskyty položky p rovnaké znamienko. Rozoberieme situáciu, keď sú všetky kladné, pre záporné to bude vyzerať podobne.

Rozdeľme položky na príjmy a výdaje. To, čo teraz chceme dosiahnuť, je nájsť niekoľko výdajov, ktoré budú mať dokopy súčet p, $2p$, ..., alebo $(n_p - 1)p$. Ak sa nám toto podarí, môžeme tieto výdaje zmeniť na príjmy a zodpovedajúci počet príjmov veľkosti p na výdaje. A tým vyhráme, lebo dosiahneme, že niektoré výskyty položky p budú príjmy a niektoré budú výdaje.

Nech P je súčet príjmov. Teraz využijeme, že vieme ohraničiť výsledok: Keďže výsledok je menší ako $2K$, musí byť súčet výdajov $V > P - 2K$. Keďže každý výdaj je najviac K, bude výdajov aspoň $\lceil V/K \rceil$.

Z toho dostávame, že ak $P \geq K(p+1)$, bude výdajov aspoň p.

Keď teraz máme aspoň p výdajov, vyberme si z nich presne p a označme ich $v_1, \ldots, v_p$. Všimneme si teraz tieto súčty: 0, v_1, $v_1 + v_2$, $\ldots$, $v_1 + \cdots + v_p$. Toto je $p+1$ rôznych čísel. Podľa Dirichletovho princípu niektoré dve z nich musia dávať po delení p rovnaký zvyšok. Nech sú to súčty po v_i a po v_j.

Potom číslo $v_{i+1} + \cdots + v_j$ je násobkom p. Presnejšie, je to číslo z množiny $\{p, 2p, \ldots, Kp\}$.

Platí teda tvrdenie: „Nech pre nejaké p platí $n_p > 2K$. Potom optimálne riešenie je rovnaké ako keby n_p bolo o 2 menšie.“

Dôkaz: Zoberme ľubovoľné optimálne riešenie. Ako sme už povedali, jediné dva zlé prípady sú, ak v ňom berieme všetky položky p ako príjmy / všetky ako výdaje.

V prvom prípade ale vieme, že $P \geq n_p p \geq (2K+1)p = Kp + (Kp + p) > Kp + K$, a teda bude aspoň p výdajov. Potom ale vieme nájsť niekoľko výdajov, ktorých súčet je násobkom p a je menší ako $n_p p$. Tieto výdaje zmeníme na príjmy a niekoľko príjmov veľkosti p na výdaje.

Druhý prípad vyzerá analogicky. Ak by všetky p_i boli výdaje, príjmov musí byť aspoň toľko ako v prvom prípade výdajov. Preto tentokrát medzi príjmami vyberieme vhodnú množinu, ktorú vymeníme s výdajmi veľkosti p.

Prvý lepší algoritmus

Pozrieme sa na hodnoty n_i a každú z nich upravíme, aby bola nanajvýš rovná $2K$. (Ak teda nejaké n_i bolo viac ako $2K$, po úprave bude buď $2K$, alebo $2K - 1$, podľa jeho parity.)

Teraz zoberieme takto zmenšenú množinu príjmov a výdajov a pre ne použijeme pôvodný algoritmus s časovou zložitosťou $O(NS)$.

Teraz už ale vieme povedať, že (upravený) počet všetkých položiek je $N = O(K^2)$ a súčet všetkých položiek je $S = O(K^3)$, a teda náš algoritmus má

časovú zložitosť $O(K^5)$. Presnejšie $O(N + K^5)$, lebo v $O(N)$ musíme načítať vstup.

Veľkosť priebežných výsledkov

Zatiaľ sme sa nijako nezaoberali tým, či nám naozaj treba uvažovať všetky možné priebežné súčty, ktoré sa dajú dosiahnuť. Podobne ako pri obmedzení počtov položiek budeme chcieť ukázať, že (bez ohľadu na usporiadanie položiek) existuje optimálne riešenie, v ktorom žiaden priebežný zisk nie je (v absolútnej hodnote) príliš veľký.

Základná myšlienka bude podobná ako pri zmenšovaní počtov položiek:

Predstavme si, že máme optimálne riešenie, pri ktorom je nejaký medzisúčet M veľmi veľký. Ako by sa dalo takéto riešenie upraviť na iné, rovnako dobré, pri ktorom bude tento medzisúčet menší?

Keďže M je veľké, znamená to, že z položiek, ktoré dávajú tento medzisúčet, sme veľa prehlásili za príjmy. A naopak, keďže vieme, že výsledok je malý, bude medzi zvyšnými položkami veľa takých, ktoré sme prehlásili za výdaje.

Ak by sme teraz vybrali niekoľko zo spominaných príjmov a niekoľko zo spomínaných výdajov tak, aby dali dokopy súčet nula, vieme medzisúčet M zmenšiť tak, že zmeníme znamienko všetkým nájdeným príjmom a výdajom.

Ešte raz a poriadne. Dokážeme, že platí nasledovné tvrdenie: „Existuje optimálne riešenie, v ktorom je absolútna hodnota každého čiastočného súčtu menšia ako $2K^2 + 3K$.“

Aby sme toto dokázali, ukážeme, že ak máme väčší čiastočný súčet, vieme ho zmenšiť, a pritom nezväčšiť žiaden iný čiastočný súčet a nepokaziť optimálnosť riešenia. (Pre záporné čiastočné súčty dôkaz neuvádzame, vyzerá analogicky.)

Majme teda optimálne riešenie, v ktorom súčet prvých x položiek je $M \geq 2K^2 + 3K$. Teraz pôjdeme od x-tej položky k prvej a budeme vyberať položky, ktoré sú v našom optimálnom riešení príjmami. Prestaneme, keď súčet vybraných položiek prekročí K^2.

Ďalej pôjdeme od $(x + 1)$-ej položky k N-tej a budeme vyberať tie, ktoré sú v našom riešení výdaje. Prestaneme tesne pred tým, ako by súčet všetkých

vybraných položiek klesol pod nulu.

Zjavne takto vyberieme aspoň K príjmov a aspoň K výdajov.

Všimnime si, že všetky čiastočné súčty, ktoré zodpovedajú úseku, z ktorého sme práve vybrali príjmy, sú väčšie alebo rovné $K^2 + 2K$. A teda ak by sme aj všetkých K vybratých položiek zmenili z príjmov na výdaje, budú všetky tieto čiastočné súčty naďalej kladné, a teda sa ich absolútne hodnoty zmenšia.

Teraz potrebujeme ukázať, že existuje nejaká neprázdna podmnožina nami vybraných príjmov a výdajov, ktorá má súčet 0.

To spravíme nasledovne: Usporiadajme vybrané príjmy a výdaje do postupnosti $p_1, p_2, \ldots$ tak, aby všetky čiastočné súčty $p_1 + \cdots + p_q$ boli z množiny $\{0, \ldots, 2K - 1\}$. (Rozmyslite si, že toto vieme vždy spraviť.) Keďže má naša postupnosť aspoň $2K$ členov, a teda aspoň $2K+1$ čiastočných súčtov, musia byť dva čiastočné súčty rovnaké, a teda má zodpovedajúci úsek postupnosti súčet nula.

Ak teraz zoberieme zodpovedajúce príjmy a výdaje v optimálnom riešení a zmeníme im znamienka, dostaneme nové optimálne riešenie, pričom sme zmenšili niektoré čiastočné súčty (vrátane toho vybraného veľkého) a ostatné čiastočné súčty zostali nezmenené.

Tento proces môžeme opakovať, až kým nedostaneme optimálne riešenie, kde sú všetky čiastočné súčty malé. Analogicky sa vieme zbaviť príliš záporných čiastočných súčtov.

Druhý lepší algoritmus

Vieme už, že stačí hľadať také riešenia, v ktorých sú všetky čiastočné súčty medzi $-2K^2 - 3K$ a $2K^2 + 3K$. Keď pridáme toto ohraničenie do predchádzajúceho riešenia, dostávame algoritmus s časovou zložitosťou $O(N + K^4)$.

Šikovnejšie počítanie dosiahnuteľných medzisúčtov

Položky s rovnakou hodnotou budeme spracúvať všetky naraz.

Teraz nás teda budú zaujímať nové hodnoty $E(s, p)$ definované nasledovne: $E(s, p)$ je 1 (true, pravda), ak sa dá dosiahnuť výsledná suma s použitím všetkých položiek, ktoré majú hodnoty 1 až p. Riešením zadanej úlohy je zjavne

najmenšie nezáporné s také, že $E(s, K)$ je pravda.

Pre $p > 0$ platí: $E(s, p)$ je pravda práve vtedy, ak je pravda aspoň jedna z hodnôt $E(s - pn_p, p - 1)$, $E(s - p(n_p - 2), p - 1)$, ..., $E(s + pn_p, p - 1)$.

Slovne: pomocou položiek s veľkosťami 1 až p vieme sumu s vyrobiť vtedy a len vtedy, ak vieme pomocou položiek menších ako p vyrobiť nejakú sumu s' takú, že $s - s'$ vieme vyrobiť pomocou položiek veľkosti p.

Ak by sme hodnoty $E(s, p)$ počítali priamo podľa tejto definície, dostali by sme rovnako efektívne riešenie ako v predchádzajúcom prípade.

Na to, aby sme dosiahli lepšiu časovú zložitosť, všimnime si nasledujúcu vec: Ak nás zaujíma, či platí $E(s + 2p, p)$, budeme sa pozerať na skoro tie isté hodnoty $E(?, p - 1)$ ako keď sme zisťovali, či platí $E(s, p)$.

Ako si túto prácu ušetriť? Označme
$M(s, p) = E(s - pn_p, p - 1) + E(s - p(n_p - 2), p - 1) + \cdots + E(s + pn_p, p - 1)$.
Slovne, $M(s, p)$ je počet takých súm, ktoré sa dajú naskladať z položiek menších ako p, a z ktorých vieme položkami veľkosti p vyrobiť sumu s.

$M(s, p)$ je takmer to isté ako $E(s, p)$, len namiesto $\vee$ sme použili $+$. Zjavne teda $E(s, p)$ je pravda práve vtedy, keď $M(s, p)$ je kladné.

Čo sme tým získali? To, že vieme, že platí nasledujúci vzťah:
$M(s + 2p, p) = M(s, p) - E(s - pn_p, p - 1) + E(s + 2p + pn_p, p - 1)$

Takže „hrubou silou“ stačí spočítať prvých $2p$ hodnôt $M(?, p)$, každú ďalšiu dostaneme v konštantnom čase.

Na výpočet jednej hodnoty hrubou silou potrebujeme čas $O(n_p)$, čo vieme zhora odhadnúť ako $O(K)$. Výpočet $2p$ hodnôt hrubou silou teda bude trvať $O(K^2)$. Ostatných hodnôt, ktoré budeme počítať, je $O(K^2)$.

Celkovo teda vieme pre konkrétne p spočítať všetky hodnoty $E(?, p)$ v čase $O(K^2)$. Tým dostávame algoritmus s časovou zložitosťou $O(N + K^3)$.

Listing programu:

```
program zisk;
const MAXK = 100;                              { maximální velikost příjmu či výdaje }
      MAXZ = 2 * MAXK * (MAXK + 2);                    { maximální velikost prvku Z_t }
```

```
type mnozina = record       { true je nastaveno na pozicích všech prvku množiny }
       prvky : array[-MAXZ .. MAXZ] of boolean;
    end;
var m : array[1 .. 2] of mnozina;                          { množiny Z_t a Z_(t+1) }
    predchozi : integer;       { Z_t je m[predchozi], Z_(t+1) je m[3-predchozi] }
    k : integer;
    amaxz : integer;                                                  { 2k(k+2) }
    ni : array[1 .. MAXK] of integer;                              { čísla n_i }
    mz : array[-MAXZ .. MAXZ] of integer;                       { čísla m_z ... }
                    { ve skutečnosti by stačilo si pamatovat jen posledních 2t }

{ Inicializuje MN na prázdnou množinu }
procedure smaz (var mn : mnozina);
var i : integer;
begin for i := -amaxz to amaxz do mn.prvky[i] := false; end;

{ Vrátí true pokud X je prvkem množiny MN }
function je_prvek (var mn : mnozina; x : integer) : boolean;
begin je_prvek := (abs (x) <= amaxz) and mn.prvky[x]; end;

{ Z množiny Z_(T-1) (P) vytvoří množinu Z_T (MN) }
procedure dalsi_zisky (var p, mn : mnozina; t : integer);
var yt, i, z, max_yt : integer;
begin
  max_yt := t * ni[t];
  { spočteme čísla m_z pro z v rozmezí -amaxz ... -amaxz + 2t-1 }
  for z := -amaxz to -amaxz + 2 * t - 1 do begin
    i := 0;
    yt := -max_yt;
    while yt <= max_yt do begin
      if je_prvek (p, z + yt) then inc (i);
      inc (yt, 2 * t);
    end;
    mz[z] := i;
  end;
  { a nyní zbývající hodnoty m_z }
  for z := -amaxz + 2 * t to amaxz do begin
    i := mz[z - 2 * t];
    if je_prvek (p, z + max_yt) then inc (i);
    if je_prvek (p, z - 2 * t - max_yt) then dec (i);
    mz[z] := i;
  end;
  { do množiny mn dáme čísla z taková, že m_z není nula }
  for z := -amaxz to amaxz do if mz[z] <> 0 then mn.prvky[z] := true;
end;

{ Vrátí nejmenší nezáporné číslo v množině MN }
function minimum (var mn : mnozina) : integer;
var i : integer;
begin
  for i := 0 to amaxz do
    if mn.prvky[i] then begin minimum := i; break; end;
end;

{ Načte seznam příjmů či výdajů }
procedure nacti;
```

```
var i, n, a : integer;
begin
  readln (n, k);
  amaxz := 2 * k * (k + 2);
  for i := 1 to k do ni[i] := 0;
  for i := 1 to n do begin read (a);  inc (ni[a]);  end;
end;

{ Omezí čísla n_i na nanejvýš 2k+1 }
procedure omez_ni;
var i : integer;
begin
  for i := 1 to k do
    if ni[i] > 2 * k + 1 then begin
      if odd(ni[i]) then ni[i] := 2 * k + 1 else ni[i] := 2 * k;
    end;
end;

var t : integer;
begin
  nacti;
  omez_ni;
  smaz (m[1]);   smaz (m[2]);
  predchozi := 1;
  { Z_0 = (0) }
  m[predchozi].prvky[0] := true;
  for t := 1 to k do begin
    dalsi_zisky (m[predchozi], m[3 - predchozi], t);
    smaz (m[predchozi]);
    predchozi := 3 - predchozi;
  end;
  writeln (minimum (m[predchozi]));
end.
```

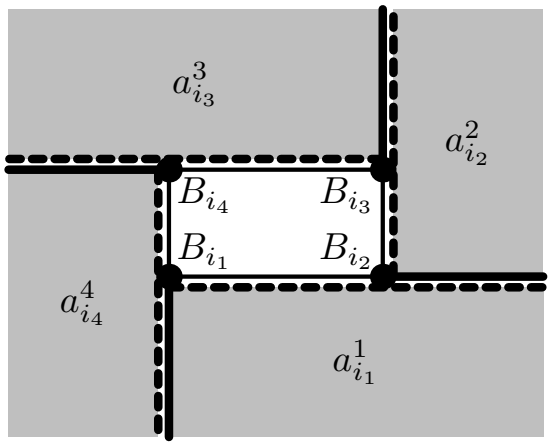
A-III-2 Obdĺžnik

Ukážeme si riešenie, ktorého časová zložitosť je $O(N^2)$ a jeho pamäťové nároky sú lineárne od počtu daných bodov.

Najprv vymyslíme, ako pre daný obdĺžnik $A_1A_2A_3A_4$ rýchlo určiť počet bodov, ktoré ležia vnútri tohto obdĺžnika. Pre každý bod B_i spočítame počet bodov $B_{i'}$, ktorých x-ová súradnica je väčšia alebo rovná x-ovej súradnici bodu B_i a zároveň y-ová súradnica je ostro menšia než y-ová súradnica bodu B_i. Tento počet označíme a_i^1. Podobne, a_i^2 bude počet bodov s x-ovou súradnicou ostro väčšou a y-ovou súradnicou väčšou alebo rovnou, a_i^3 počet bodov s x-ovou súradnicou menšou alebo rovnou a y-ovou ostro väčšou a konečne a_i^4 bude počet bodov s x-ovou ostro menšou a y-ovou menšou alebo rovnou. Každé z

čísel a_i^1, a_i^2, a_i^3 a a_i^4 je ľahké vypočítat v čase $O(N)$ (pre jeden bod B_i). Teda na spočítanie všetkých $4N$ čísel a_i^1, a_i^2, a_i^3 a a_i^4, $1 \leq i \leq n$, spotrebujeme čas $O(N^2)$.

Ľahko nahliadneme, že pokiaľ vrchol A_j obdĺžnika $A_1A_2A_3A_4$ je bod B_{i_j}, potom počet bodov ležiacich vnútri obdĺžnika $A_1A_2A_3A_4$ je $N - a_{i_1}^1 - a_{i_2}^2 - a_{i_3}^3 - a_{i_4}^4$ (viď obrázok). Pre jeden obdĺžnik $A_1A_2A_3A_4$ môžeme teda určiť počet bodov, ktoré ležia vnútri tohto obdĺžnika, v konštantnom čase.



(Poznámka: Existujú aj iné možné postupy, napr. pre vhodné body si spočítame počet bodov, ktoré ležia v ľavom hornom kvadrante od daného bodu, a z týchto informácií poskladáme v konštantnom čase počet bodov v ľubovoľnom obdĺžniku. Väčšina alternatívnych postupov ale potrebuje pamäť $O(N^2)$.)

Teraz si popíšeme, ako môžeme rýchlo najsť všetky obdĺžniky $A_1A_2A_3A_4$, ktorých hrany sú rovnoběžné s osami a ktorých vrcholy sú v niektorých zo zadaných bodov. Najprv si všetky body zotrieďme vzostupne podľa x-ovej súradnice a tie body, ktoré majú zhodnú x-ovú súradnicu, zotriedime medzi sebou vzostupne podľa y-ovej súradnice. Pre každý bod B_i teraz môžeme vypísať tie body, ktoré majú rovnakú x-ovú súradnicu ako B_i a väčšiu y-ovú súradnicu, v čase lineárnom od ich počtu. Podobne si zotriedime všetky body podľa y-ovej súradnice a v prípade zhody podľa x-ovej, aby sme mohli ľahko nájsť body s rovnakou y-ovou a väčšou x-ovou súradnicou. Na zotriedenie bodov potrebujeme čas $O(N \log N)$ (použijeme napr. triediaci algoritmus heapsort alebo quicksort)

a na uloženie zotriedeného zoznamu bodov vrátane odkazov doňho potrebujeme pamäť lineárnu od N.

Teraz si popíšeme postup, ako nájsť všetky obdĺžniky $A_1A_2A_3A_4$, ktoré spĺňajú podmienky zo zadania úlohy. Zafixujeme jeden bod B_i a určíme všetky body B_j také, že bod B_i je vrchol A_1 a bod B_j vrchol A_3 nejakého obdĺžniku $A_1A_2A_3A_4$ (všimnite si, že vrcholy A_2 a A_4 sú bodmi B_i a B_j jednoznačne určené).

K nájdeniu všetkých takýchto bodov B_j pre bod B_i si vytvoríme pomocné pole C, ktorého všetky zložky $C[j]$ budú na začiatku rovné nule. Potom vezmeme všetky body $B_{i'}$, ktoré majú rovnakú x-ovú súradnicu ako B_i a zároveň väčšiu y-ovú súradnicu, a pre každý takýto bod $B_{i'}$ vezmeme všetky body $B_{i''}$, ktoré majú rovnakú y-ovú a väčšiu x-ovú súradnicu než $B_{i'}$. Body $B_{i'}$ zvládneme nájsť v lineárnom čase od ich počtu a rovnako body $B_{i''}$ pre každý bod $B_{i'}$ vieme nájsť v lineárnom čase od ich počtu. Pretože body $B_{i''}$ sú rôzne pre rôzne body $B_{i'}$, trvá nájdenie všetkých bodov $B_{i''}$ čas lineárny od N. Zložku $C[i'']$ poľa C zvýšime o jedna pre každý bod $B_{i''}$, ktorý sme takto našli. Zdôrazňujeme, že teraz je každá zložka poľa C rovná buď 0 alebo 1.

Potom pre bod B_i vezmeme všetky body $B_{i'}$ s rovnakou y-ovou a väčšou x-ovou súradnicou a pre také body $B_{i'}$ nájdeme všetky body $B_{i''}$ s rovnakou x-ovou súradnicou a väčšou y-ovou súradnicou. Za každý takýto bod zvýšime hodnotu $C[i'']$ o jedna. Zložky poľa C sú teraz rovné $0, 1$ alebo 2. Ďalej platí, že $C[j]$ je rovné 2 vtedy a len vtedy, keď pre body B_i a B_j existuje bod, ktorý má rovnakú x-ovú súradnicu ako B_i a rovnakú y-ovú súradnicu ako B_j, a zároveň existuje bod, ktorý má rovnakú x-ovú súradnicu ako B_j a rovnakú y-ovou súradnicu ako B_i. Potom ale takéto dva body tvoria vrcholy A_2 a A_4 obdĺžnika, ktorého vrchol A_1 je B_i a vrchol A_3 je B_j. Pre daný bod B_i môžeme teda v čase $O(N)$ nájsť všetky body B_j, ktoré tvoria protiľahlé vrcholy A_1 a A_3 nejakého obdĺžnika $A_1A_2A_3A_4$. Určiť počet vnútorných bodov takéhoto obdĺžnika vieme v konštantnom čase, ako sme si vysvetlili na začiatku riešenia.

Ako sme si práve popísali, dá sa pre daný bod B_i v čase $O(N)$ nájsť všetky obdĺžniky s ľavým dolným rohom B_i, a pre každý z nich spočítať, koľko bodov v ňom leží. Toto spravíme pre každé i.

Výsledný algoritmus má časovú zložitosť $O(N^2)$ a pamäťovú $O(N)$.

Listing programu:

```
program obdelniky;
const MAXN=1000; { maximální počet zadaných bodů }
var N,i: longint;
    sorted_by_x, sorted_by_y, index_by_x, index_by_y: array[1..MAXN] of longint;
    Bx, By, A1, A2, A3, A4, C: array[1..MAXN] of longint;
    nejlepsi_pocet: longint;

procedure spocitej_A;
var i,j: longint;
begin
  for i:=1 to N do begin
    A1[i]:=0; A2[i]:=0; A3[i]:=0; A4[i]:=0;
    for j:=1 to N do begin
      if (Bx[j]>=Bx[i]) and (By[j]<By[i]) then inc(A1[i]);
      if (Bx[j]>Bx[i]) and (By[j]>=By[i]) then inc(A2[i]);
      if (Bx[j]<=Bx[i]) and (By[j]>By[i]) then inc(A3[i]);
      if (Bx[j]<Bx[i]) and (By[j]<=By[i]) then inc(A4[i]);
    end;
  end;
end;

{ funkcie quick_?, ktore pouzitim quicksortu utriedia pole sorted_by_?
  podla ?-ovej suradnice, z priestorovych dovodov neuvadzame }

procedure setrid;
var i:longint;
begin
  for i:=1 to N do begin sorted_by_x[i]:=i; sorted_by_y[i]:=i; end;
  quick_x(1,N);  for i:=1 to N do index_by_x[sorted_by_x[i]]:=i;
  quick_y(1,N);  for i:=1 to N do index_by_y[sorted_by_y[i]]:=i;
end;

procedure zkus_vrchol(i: longint);
var i1, i2, j: longint;
    a2_index, a4_index: array[1..MAXN] of longint;
begin
  for j:=1 to N do C[j]:=0;
  i1:=index_by_x[i]+1;
  while Bx[sorted_by_x[i1]]=Bx[i] do begin
    i2:=index_by_y[sorted_by_x[i1]]+1;
    while By[sorted_by_y[i2]]=By[sorted_by_x[i1]] do begin
      inc(C[sorted_by_y[i2]]);
      inc(i2);
      a2_index[i2]:=sorted_by_x[i1];
    end;
    inc(i1)
  end;
  i1:=index_by_y[i]+1;
  while By[sorted_by_y[i1]]=By[i] do begin
    i2:=index_by_x[sorted_by_y[i1]]+1;
    while Bx[sorted_by_x[i2]]=Bx[sorted_by_y[i1]] do begin
      inc(C[sorted_by_x[i2]]);
      inc(i2);
      a4_index[i2]:=sorted_by_y[i1];
```

```
    end;
    inc(i1)
  end;
  for j:=1 to N do
    if (C[j]=2)
       and (N-A1[i]-A2[a2_index[j]]-A3[j]-A4[a4_index[j]] > nejlepsi_pocet)
    then begin
      nejlepsi_pocet := N - A1[i] - A2[a2_index[j]] - A3[j] - A4[a4_index[j]];
    end;
end;

begin
  readln(N); for i:=1 to N do readln(Bx[i],By[i]); end;
  spocitej_A;
  setrid;
  nejlepsi_pocet:=0;
  for i:=1 to N do zkus_vrchol(i);
  if nejlepsi_pocet=0 then writeln('Nemá řešení.') else writeln(nejlepsi_pocet);
end.
```

A-III-3 Grafomat a šamani

Na úvod si uvedomme, že stačí vedieť zadanú úlohu riešiť pre stromy. Ak totiž máme našu úlohu riešiť pre 3-graf, môžeme ho z vrcholu, ktorý je na začiatku označený, prehľadať do šírky. Podobne ako v riešení úlohy z domáceho kola si navyše v každom vrchole zapamätáme, odkiaľ sme doň prvýkrát prišli. Takto dostaneme zakorenený strom s koreňom vo vrchole, ktorý bol na začiatku označený.

Jednoduchšie ale pomalšie riešenie

Chceme rozdeliť vrcholy stromu na dve rovnako veľké množiny. Keby sme túto úlohu riešili na klasickom počítači, mohli by sme vymyslieť napríklad nasledujúce riešenie: prehľadáme celý strom do hĺbky a vrcholy v poradí, v akom ich objavujeme, farbíme striedavo na červeno a na modro. V Pascale by to vyzeralo napríklad nejako takto:

```
function prejdi_strom ( v:vrchol ; f:farba ) : farba;
var i : integer;
begin
   if (f=cervena) f:=modra else f:=cervena;
   v.farba := f;
   for i:=1 to v.pocetSynov do   f := prejdi_strom( v.syn[i], f );
end;
```

Toto riešenie vieme naprogramovať aj v grafomate. Namiesto rekurzie si budeme medzi vrcholmi odovzdávať značku, ktorá bude reprezentovať miesto, kde sa práve prehľadávanie nachádza. Po každom takte bude v grafe práve jedna značka. Vo vrchole, ktorý má značku, si navyše budeme pamätať naposledy použitú farbu a to, či sme do vrcholu práve prišli, alebo už chceme odísť z jeho podstromu.

Presnejšie to bude prebiehať takto:

1. Na začiatku dostane značku koreň, pamätáme si v ňom, že sme práve prišli a posledná použitá farba bola 0.
2. Ak vrchol má značku a práve sme doň prišli, v nasledujúcom takte zmení pamätanú farbu na opačnú a ofarbí sa ňou. Potom:
   - Ak nemá žiadnych synov, značku si nechá a zapamätá si, že chceme odísť preč.
   - Ak má nejakých synov, pošle značku prvému z nich.
3. Ak vrchol má značku a pamätáme si, že chceme z neho odísť:
   - Ak má mladšieho brata (teda vrchol, ktorý má rovnakého otca a u otca o jedno väčšie poradové číslo), pošle jemu značku a u brata si zapamätáme, že sme práve prišli.
   - Ak mladšieho brata nemá, odovzdá značku späť otcovi a zapamätáme si u otca, že budeme z neho chcieť odísť.
   - Ak už nemá ani otca, sme v koreni, celý graf je ofarbený a končíme.

Tieto pravidlá vieme priamočiaro prepísať do programu pre grafomat. Bude to fungovať tak trochu naopak. Napríklad namiesto toho, aby sme raz poslali značku mladšiemu bratovi, bude sa každý vrchol v každom takte pozerať na svojho staršieho brata, či ten nemá značku, ktorú chce poslať preč. Keď niekedy vrchol uvidí, že ju jeho starší brat má, nastaví si, že teraz je značka u neho.

Toto riešenie má časovú zložitosť úmernú počtu vrcholov v grafe.

Ide to však aj lepšie. Vzorové riešenie bude mať časovú zložitosť úmernú tomu, ako ďaleko je najvzdialenejší vrchol od toho, kde výpočet začíname – teda od hĺbky stromu.

Zložitejšie ale rýchlejšie riešenie

Podobne ako v predchádzajúcom riešení začneme tým, že zadaný 3-graf prehľadáme do šírky, čím dostaneme zakorenený strom. Aby bolo naše riešenie rýchle, budeme chcieť jeho podstromy ofarbovať paralelne. Čo na to ale potrebujeme vedieť?

Ukážeme si najskôr na príklade približnú myšlienku riešenia, potom doplníme implementačné detaily.

Predstavme si, že máme napríklad vrchol v, ktorý má troch synov: a, b a c. Pozrime sa teraz na podstromy s koreňmi a, b, c. V niektorých z nich je párny počet vrcholov. Tieto nás netrápia, lebo sa ich dá ofarbiť presne. Trápiť nás budú tie, kde je počet vrcholov nepárny. Keď ofarbíme taký podstrom „približne správne“, bude v ňom vrcholov jednej farby o jedno viac ako druhej.

Keby sme vo vrchole v vedeli, ktoré z podstromov s koreňmi a, b, c majú párnu a ktoré nepárnu veľkosť, ľahko by sme všetko vyriešili: do párnych podstromov oznámime „ofarbite sa rovnomerne“, do približne polovice nepárnych podstromov oznámime „ofarbite sa tak, aby bolo o jedno viac červených“ a do zvyšku „ofarbite sa tak, aby bolo o jedno viac modrých“.

Teraz už k tým sľúbeným detailom. Podrobnejšie bude náš algoritmus vyzerať nasledovne:

1. Prehľadáme graf zo začiatočného vrcholu, čím vznikne strom.
2. Budeme si posielať informácie od listov do koreňa a pre každý vrchol si spočítame, či má jeho podstrom párnu alebo nepárnu veľkosť.
3. Keď sa už koreň dozvie svoju paritu (tá určite vyjde párna), začneme po hladinách ofarbovať. V prvom takte tejto časti ofarbíme koreň farbou 0.
4. Ak už je nejaký vrchol v ofarbený, postupne sa ofarbia jeho synovia. Prvý syn dostane opačnú farbu ako v. Farba každého ďalšieho syna závisí od farby predchádzajúceho a parity veľkosti jeho podstromu.

   (Presnejšie, ak bol predchádzajúci podstrom párnej veľkosti, dostane ďalší syn rovnakú farbu, inak dostane opačnú.)

Uvedomte si, že ofarbenie, ktoré tento algoritmus vyrobí, bude úplne rovnaké ako to, ktoré vyrobil pomalší algoritmus. Iba sme na zrýchlenie použili to, že

keď vieme paritu počtu vrcholov v podstrome, nemusíme čakať na to, ktorá farba v ňom bude použitá ako posledná – vieme si to vypočítať a rovno začať farbiť aj susedný podstrom.

Keďže pôvodný graf je 3-graf, nemá žiaden vrchol v strome viac ako troch synov. (Presnejšie, každý vrchol okrem koreňa má najviac dvoch synov.) Preto každú vrstvu stromu ofarbíme na najviac tri takty.

Listing programu:

```
var x: 0..1;                                    { 1=počáteční vrchol, 0=ostatní }
    y: 0..2 = 2;              { výstupní barva: 0=červená, 1=zelená, 2=neurčena }
    z: 0..4 = 0;            { hrana vedoucí k otci vrcholu, 0=neurčena, 4=kořen }
    q: 0..2 = 2;               { parita podstromu: 0=sudá, 1=lichá, 2=neurčena }
    i, k, l: 0..3 = 0;                                        { pomocné proměnné }

begin
   if z=0 then begin          { 1. fáze: prohledávání do šířky (stavění stromu) }
      if x=1 then z := 4                                 { Kořen označíme speciálně }
      else for i:=1 to 3 do           { Když máme označeného souseda, máme otce }
         if (S[i].z > 0) or (S[i].x = 1) then z := i;
   end
   else if q=2 then begin                                  { 2. fáze: počítání parity }
      l := 0;                                          { kolik synů má lichou paritu }
      k := 0;                                             { už můžeme paritu určit? }
      for i:=1 to 3 do
         if S[i].z = 0 then k := 1      { neprohledaný soused => určit nemůžeme }
         else if S[i].z = P[i] then                                   { je to syn }
            if S[i].q = 2 then k := 1               { syn nemá paritu => ani my }
            else if S[i].q = 1 then l := l+1;                        { lichý syn }
      if k = 0 then q := 1 - l mod 2;    { toto funguje i když má vrchol 0 synů }
   end
   else if y=2 then begin                                        { 3. fáze: barvení }
      if (x=1) and (q<2) then y := 0      { dopočítáme paritu => obarvíme kořen }
      else if (z>0) and (S[z].y < 2) then begin          { otec už je obarven => }
         y := 1 - S[z].y;         { => dopočítáme svou barvu podle parity bratrů }
         k := P[z];                                  { číslo hrany vedoucí z otce sem }
         for i := 1 to k-1 do           { za každého lichého staršího bratra ... }
            if S[z].S[i].q = 1 then y := 1 - y;              { ... barvu otočíme }
      end;
   end;
end.
```

SLOVENSKÁ KOMISIA OLYMPIÁDY V INFORMATIKE

DVADSIATY DRUHÝ ROČNÍK OLYMPIÁDY V INFORMATIKE

Vydala IUVENTA s finančnou podporou Ministerstva školstva SR
Náklad: 40 výtlačkov
Zodpovedný redaktor: Michal Forišek
Sadzba programom LaTeX