Prvé domáce kolo 16. sezóny súťaže Palma sa konalo dňa `05.12.2017` v čase `15:00-18:00`. Zúčastnilo sa ho 13 súťažných tímov (celkovo 21 súťažiacich), ktorí riešili 4 úlohy - jedna bola rozdelená na ľahšiu a ťažšiu verziu.

1 IFK - Kalendar

Úloha Kalendar bola pripravená ako priamočiara simulácia. Na jej riešenie stačí generovať postupne ďalšie dni podľa spomenutého kalendára a kontrolovať, či majú magický tvar. Ročný a priestupný deň nie je potrebné vôbec riešiť, keďže tieto dni určite nie sú magické.

2 P - Priamkovač

Úloha Priamkovač sa dá triviálne riešiť skúšaním všetkých možných dvojíc bodov ako priamok a následne počítaním bodov na priamke v zložitosti $O\left(\binom{n}{2}.n\right) = O(n^3)$, čo samozrejme nezbehne pre veľký vstup $n = 2000$.
Efektívnejšie riešenie využíva zametaciu priamku (SweepLine). Pre daný jeden fixný bod vieme efektívne spočítať počty bodov na všetkých priamkach prechádzajúcich týmto bodom. Môžeme si usporiadať zvyšné body podľa uhla (smernice priamky s daným fixným bodom) v čase $O(n.\log n)$ a pozametať okolo fixného bodu v čase $O(n)$. Celková zložitosť algoritmu je teda $O(n^2.\log n)$.
Iné riešenie je prejsť všetky zvyšné body a počítať si v množine uhlov početnosť (na to je možné použiť `unordered_map` v C++, resp. `HashMap` v jazyku Java). V priemernom prípade takto dostaneme zložitosť $O(n^2)$, aj keď asymptotická zložitosť zostáva $O(n^3)$.
Pozor si treba dať na okrajové prípady - pri počítaní smernice sú to zvislé priamky, ďalej viacnásobné body (aj pre prípad všetkých n bodov rovnakých).
Aby sme sa vyhli nepresnostiam pri práci s reálnymi číslami (pri použití funkcií `atan` alebo `atan2` na zistenie uhla), tak namiesto uhlu sa bude porovnávať iba smernica priamky v tvare zlomku, t.j. základný tvar $\frac{y_2-y_1}{x_2-x_1}$, resp. usporiadaná dvojica $\{citatel, menovatel\}$. Do základného tvaru upravíme zlomok predelením čitateľa aj menovateľa ich najväčším spoločným deliteľom.

3 S - Štafeta

Keďže v úlohe Štafeta nepoznáme poradie, môžeme vyskúšať všetkých 6 možných poradí a vybrať z nich to najlepšie. To je možné urobiť v C++ použitím funkcie `next_permutation` alebo vymenovaním.
Priamočiare riešenie pre dané poradie vyskúša všetky možné indexy posledných úloh prvého študenta `i` a druhého študenta `j`, $0 < i < j < N$. Následne spočíta všetky obtiažnosti pre danú dvojicu i, j: $\sum_{x=1}^{i} OBT[1,x] + \sum_{x=i+1}^{j} OBT[2,x] + \sum_{x=j+1}^{N} OBT[3,x]$ a vyberie minimum. Takéto riešenie má zložitosť $O(\binom{n}{2}.n) = O(n^3)$, čo nestačí ani na ľahšiu verziu úlohy.
Lepšie riešenie nepočíta súčty stále znova - buď použije prefixové súčty v riadkoch alebo jednoducho stále len pripočíta/odpočíta ďalšiu hodnotu (ak sa j zmení na $j+1$, tak v druhej sume treba jednu hodnotu pripočítať a v tretej odpočítať oproti predchádzajúcemu výsledku). To už je riešenie ľahšej úlohy v čase $O(n^2)$.
Na riešenie ťažšej verzie je potrebné použiť dynamické programovanie. Počítať budeme minimálnu obtiažnosť $OPT[i,j]$, $1 \leq i \leq 3$, $1 \leq j \leq N$, že študent i práve vyriešil úlohu j. Pozrieme, aký mohol byť stav predtým, teda ako vypočítať aktuálne optimum z predchádzajúcich

už vypočítaných hodnôt OPT. Predchádzajúcu úlohu $j-1$ buď vyriešil tiež študent i alebo študent $i-1$. Vzorec na výpočet dynamického programovania teda bude

$$OPT[i,j] = OBT[i,j] + \min\{OPT[i-1,j-1], OPT[i,j-1]\}$$

. Takto vieme vypočítať hodnoty v čase $O(3.N)$, pričom výsledok je $OPT[3,N]$. Celková zložitosť je spolu so zarátaním kombinácií poradia je $O(6.3.N) = O(N)$, teda lineárna.
Pre vstup zo zadania (vľavo), je možné takto vypočítať tabuľku OPT (vpravo):

```
2 4 1 5 1 X X      02 06 07 12 13 XX XX
X 3 2 5 3 2 X  ⇒   XX 05 07 12 15 15 XX
X X 5 4 3 3 3      XX XX 10 11 14 17 18
```

Prvý riadok OPT je prefixový súčet, ďalšie sú dopočítané dynamickým programovaním. Výsledkom je, že pri riešení danej úlohy v poradí študentov 1,2,3 je minimálna obtiažnosť 18.

4 Z - OO

Úlohu ZOO je možné reprezentovať grafom[1], pričom zvieratá budú reprezentovať vrcholy a vzťah ublíženia si označíme hranou. Cieľom je teda zistiť, či ich vieme rozdeliť na dve množiny (ofarbiť vrcholy dvoma farbami) tak, aby hrany boli iba medzi jednou a druhou množinou (nie v rámci jednej množiny). Graf, ktorý sa takto dá zafarbiť, sa nazýva `párny`, resp. `bipartitný`. Zistiť, či je graf párny sa dá vyskúšaním. BUNV[2] môžeme prvý vrchol dať do prvej množiny. Všetky s ním susedné vrcholy musia ísť do druhej. A susedné vrcholy takto pridaných vrcholov musia ísť zase do prvej množiny, atď. Ak vznikne spor, teda že raz priradíme vrchol do prvej množiny a neskôr do druhej, tak graf nie je párny, a teda pre uvedený vstup je odpoveď `NIE`. Ak ešte zostali nespracované vrcholy, tak pokračujeme ďalej. Takto implementované riešenie má zložitosť $O(N+M)$ alebo $O(N^2)$ podľa použitej reprezentácie grafu.

Java

Viacero riešiteľov si neprečítalo poznámky k jazyku Java a nenazvali triedu `source`, a teda odovzdané riešenie skončilo chybou kompilácie.

[1] z teórie grafov, teda majúcim vrcholy a hrany - nie grafom funkcie
[2] Bez Ujmy Na Všeobecnosti