A-III-1 Oplotenie záhradiek

Na žiadnom mieste sa nebudú stretať štyri konce plotov (dva vodorovné a dva zvislé). Keby sa tak stalo, môžeme napríklad tie dva vodorovné spojiť do jedného a ušetriť tak jeden poplatok.

Predstavme si, že si všetky poplatky zoradíme podľa ceny od najvyššieho po najnižšiu, a v prípade rovnosti si zvolíme nejaké jedno konkrétne poradie. Riadok alebo stĺpec, ktorý je skôr vo zvolenom poradí, nazveme dôležitejší. Rovnakú terminológiu použijeme aj pre ploty v týchto riadkoch.

Všimneme si ľubovoľnú križovatku riadku a stĺpca. Kľúčové pozorovanie je, že sa nikdy neoplatí, aby touto križovatkou prechádzal plot v menej dôležitom smere. Ak by sme totiž takýto plot rozdelili na dva a naopak spojili dva ploty v dôležitejšom smere do jedného, dostaneme aspoň rovnako dobré riešenie.

Uvažujme teraz optimálne riešenie, v ktorom vyššie uvedené pravidlo nikde nie je porušené. Je zjavné, že týmto je vzhľad optimálneho riešenia jednoznačne určený: v každom riadku/stĺpci je plot prerušený presne na tých miestach, kde križuje dôležitejšie stĺpce/riadky.

No a podľa tohto pozorovania už vieme efektívne zistiť celkovú cenu poplatkov. Potrebujeme len pre každý riadok/stĺpec zistiť, koľko od neho dôležitejších stĺpcov/riadkov ho križuje. No a toto vieme spraviť práve tak, že si ich všetky podľa vyššie popísaného kľúča usporiadame a potom ich spracúvame, pričom si pamätáme, koľko riadkov a koľko stĺpcov sme už spracovali.

Pri dobrej implementácii takto dostaneme riešenie s časovou zložitosťou $O((d+s)\log(d+s))$. Na asymptotickú časovú zložitosť nemá vplyv to, či usporiadame ceny pre každý rozmer zvlášť alebo všetky dokopy.

Listing programu (C++)

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int D, S;
    cin >> D >> S;
    vector<int> zvisle(D-1), vodorovne(S-1);
    vector< pair<int,int> > moznosti;
    for (int d=0; d<D-1; ++d) {
        cin >> zvisle[d];
        moznosti.emplace_back(zvisle[d],0);
    }
    for (int s=0; s<S-1; ++s) {
        cin >> vodorovne[s];
        moznosti.emplace_back(vodorovne[s],1);
    }
    sort( moznosti.begin(), moznosti.end() );
    reverse( moznosti.begin(), moznosti.end() );
    vector<int> spracoval(2,0);
    int odpoved = 0;
    for (auto moznost : moznosti) {
        int cena = moznost.first, pocet = 1 + spracoval[1-moznost.second];
        odpoved += cena * pocet;
        ++spracoval[moznost.second];
    }
    cout << odpoved << endl;
}
```

Dodatočné komentáre

Na vyššie popísané riešenie v skutočnosti nemá žiadny vplyv posledná podmienka zo zadania: tá, že ploty treba stavať tak, aby sa žiadny z nich nedal predĺžiť. Naše riešenie ju zjavne splní, a zároveň z dôkazu jeho správnosti vyplýva, že táto podmienka je zbytočná. Keby sme ju v zadaní nemali, optimálne riešenie by stále malo rovnakú cenu.

Vďaka tejto dodatočnej podmienke sa však aj bez vyššie uvedeného dôkazu dá spraviť 6-bodové riešenie s časovou zložitosťou $O(d^2s^2(d+s))$. Pri tomto riešení si uvedomíme, že prvý plot, ktorý postavíme, musí nutne byť niektorý celý riadok alebo niektorý celý stĺpec. Vyskúšame postupne všetky možnosti. Každá nám problém rozdelí na dva menšie obdĺžikové podproblémy toho istého typu, ktoré treba vyriešiť každý zvlášť. To spravíme rekurzívne, pričom výsledky memoizujeme. Iný, ekvivalentný pohľad je postup zdola hore (iteratívne dynamické programovanie): postupne nájdeme optimálne riešenie pre každú z $O(d^2s^2)$ obdĺžnikových podoblastí vstupu, od najmenších po najväčšie.

Iný, komplikovanejší ale tiež korektný dôkaz tvrdenia, na ktorom je založené vzorové riešenie sa dá spraviť matematickou indukciou. Presnejšie, indukciou podľa obsahu obdĺžnika dokážeme, že pre hocijaký obdĺžnik s

daným obsahom platí, že ako prvý môžeme postaviť plot s maximálnym poplatkom. Totiž ak by bolo optimálne ako prvý postaviť iný plot kolmý na ten, ktorý chceme, z indukčného predpokladu vieme, že v každej polovici je optimálne ako prvý postaviť kus nášho plota, no a teraz vieme tieto prvé tri kroky bez zhoršenia celkovej ceny nahradiť postavením nášho plota a dvoch častí toho iného. Rozmyslite si, ako tento argument dokončiť pre riešenia začínajúce postavením iného plota rovnobežného s tým, ktorý chceme.

A-III-2 Horolezci

Triviálne pozorovanie: Cena výstroja každého z horolezcov bude rovná niektorej z výšok na vstupe. (Presnejšie, maximu z výšok vrcholov, na ktoré pôjde.)

Základné netriviálne pozorovanie: Vždy je optimálne použiť nanajvýš troch horolezcov.
Dôkaz: Uvažujme ľubovoľný plán expedície, ktorý používa viac ako troch horolezcov. Vyberme si toho horolezca H, ktorý má najlacnejší výstroj. Potom vieme vyrobiť lacnejší plán expedície nasledovne: Horolezca H necháme doma a výstroj mu vôbec nekúpime. Každý deň, kedy mal horolezec H liezť, priradíme inému horolezcovi (nie nutne všetky dni tomu istému). Toto vždy vieme urobiť, lebo:

1. Každý iný horolezec má aspoň tak dobrý výstroj ako H, teda vie ísť na vrchol, na ktorý mal ísť H.
2. Máme aspoň troch iných horolezcov a nanajvýš dvaja z nich lezú v predchádzajúci a nasledujúci deň, čiže ostáva aspoň jeden, ktorý si tento deň vie pridať medzi svoje.

Pomerne priamočiaro teraz dostávame riešenie s časovou zložitosťou $O(n2^n)$: predpokladáme, že máme presne troch horolezcov, vyskúšame všetky možnosti, ktorý deň priradiť ktorému horolezcovi (pričom pre každý deň máme len dve možnosti, lebo ho nemôžeme priradiť horolezcovi, ktorý mal predchádzajúci deň) a pre každú možnosť spočítame, kto potrebuje ako dobrý výstroj (pričom ak horolezca nepoužijeme vôbec, ostane mu cena výstroja nula).

Úloha má viacero rôznych riešení s polynomiálnou časovou zložitosťou, väčšinou založených na vhodnom použití dynamického programovania. Základné riešenie je založené na pozorovaní, že keď sa postupne rozhodujeme, kto pôjde na ktorý vrchol, tak existuje len $O(n^4)$ zaujímavých stavov: potrebujeme vedieť, koľko vrcholov sme už spracovali ($n+1$ možností), pre každého horolezca jeho doteraz najvyšší pridelený vrchol (do n^3 možností) a číslo horolezca, ktorý išiel na predchádzajúci vrchol (3 možnosti).

Overenie výstrojov

Zamyslime sa nad jednoduchšou úlohou: Niekto nám povedal, aké výstroje naši traja horolezci majú. Našou úlohou už je len overiť, či s takýmito výstrojmi vedia zvládnuť celú expedíciu.
Toto vieme pomerne ľahko overiť v lineárnom čase: postupne pre každé i od 1 po n a pre každého horolezca zistíme, či sa dá naplánovať prvých i dní tak, aby všetko sedelo a v i-ty deň liezol zrovna on. Použijeme na to jednoduché pozorovanie: naplánovať prvých i dní a skončiť horolezcom x vieme práve vtedy, ak x vie vyliezť do výšky v_i a vieme naplánovať prvých $i-1$ dní tak, aby sme skončili horolezcom iným ako x.

Kubické riešenie

Jeden z horolezcov zjavne musí mať výstroj na najvyšší vrchol na vstupe. Vyskúšame všetkých $O(n^2)$ možností pre zvyšných dvoch horolezcov: pre každého z nich je to buď výška niektorého iného vrcholu zo vstupu, alebo nula ak ho vôbec na expedíciu nevezmeme. Každú z možností vyššie popísaným postupom v čase $O(n)$ skontrolujeme a vyberieme najlacnejšiu z tých, ktoré vyhovujú.

Kvadratické riešenie

Horolezcov si označíme 1, 2 a 3. Horolezec 1 bude mať výstroj na najvyšší vrchol na vstupe. Vyskúšame všetkých $O(n)$ možností pre cenu výstroja horolezca 2. Keď vieme, akú výstroj majú títo dvaja, môžeme použiť podobný postup ako vyššie popísané overovanie, s jedným rozdielom: namiesto toho, aby sme si len pamätali, ktoré

situácie sa dajú dosiahnuť a ktoré nie, si budeme pamätať, pre akú najmenšiu cenu výstroja horolezca 3 je ktorá situácia dosiahnuteľná.
Toto riešenie teda vyskúša $O(n)$ možností, každú z nich v čase $O(n)$, a teda je dokopy jeho časová zložitosť kvadratická od počtu dní expedície.

Listing programu (C++)

```
#include <bits/stdc++.h>
using namespace std;

const int NEKONECNO = INT_MAX/3;

int N;
vector<int> V;

int dopocitaj_tretieho(int prvy, int druhy) {
    vector< vector<int> > opt(N, vector<int>(3,NEKONECNO));
    // opt[i][j] == optimalna cena vystroja tretieho horolezca, pri ktorej sa da
    // naplanovat dni 0..i tak, aby posledny den liezol horolezec j+1

    if (prvy >= V[0]) opt[0][0] = 0;
    if (druhy >= V[0]) opt[0][1] = 0;
    opt[0][2] = V[0];

    for (int n=1; n<N; ++n) {
        if (prvy >= V[n])  opt[n][0] = min( opt[n-1][1], opt[n-1][2] );
        if (druhy >= V[n]) opt[n][1] = min( opt[n-1][0], opt[n-1][2] );
        opt[n][2] = max( V[n], min( opt[n-1][0], opt[n-1][1] ) );
    }
    return min( opt[N-1][0], min( opt[N-1][1], opt[N-1][2] ) );
}

int main() {
    cin >> N;
    V.resize(N);
    for (int &v : V) cin >> v;
    int prvy = *max_element( V.begin(), V.end() );
    int najlepsie = prvy + dopocitaj_tretieho(prvy,0);
    for (int druhy : V) najlepsie = min( najlepsie, prvy + druhy + dopocitaj_tretieho(prvy,druhy) );
    cout << najlepsie << endl;
}
```

Ešte lepšie riešenie

Až na poslednú chvíľu ale predsa sme objavili ešte lepšie riešenie tejto úlohy. (Obmedzenia v zadaní sme už nezvládli upraviť podľa neho.) Ak niekto odovzdá takéto riešenie, bude hodnotené 11 bodmi. Nie je to prvýkrát v histórii celoštátnych kôl, ale nestáva sa to často :)

Vylepšíme ďalej predchádzajúce riešenie. Rovnako ako v ňom si povieme, že horolezec 1 musí mať výstroj na najvyšší vrchol na vstupe a vyskúšame všetky možnosti pre horolezca 2. Tie ale tentokrát budeme skúšať v špecifickom poradí: postupne zhora dole.
Keď sme si zvolili cenu výstroja pre horolezca 2, je tým jednoznačne určené, že na všetky vyššie vrcholy musí ísť horolezec 1. (Tieto budeme volať vynútené.) Ako znižujeme cenu výstroja horolezca 2, vynútené vrcholy len pribúdajú. Akonáhle už sú dva vedľa seba, riešenie neexistuje a môžeme prestať skúšať.
Všimnime si, že horolezec 1 vie ísť na všetky vrcholy a horolezec 2 na všetky nie-vynútené.
V ľubovoľnom okamihu počas skúšania nám vynútené vrcholy rozdelia vstup na niekoľko nezávislých úsekov. U každého z nich nás zaujíma parita jeho dĺžky. Ak je nepárna (alebo má aspoň jeden okraj na začiatku alebo konci vstupu), vedia tento úsek vyriešiť prví dvaja horolezci na striedačku bez potreby tretieho, čo je zjavne optimálne. Ak je párna, potrebujeme aspoň raz použiť aj tretieho. A zjavne ho stačí použiť práve raz, a to na najnižšiu horu v danom úseku. Zvyšok už potom opäť vedia vyriešiť prví dvaja horolezci.
Aby sme vyhodnotili konkrétnu cenu výstroja druhého horolezca, potrebujeme teda poznať pre každý úsek párnej dĺžky jeho minimum. Maximum z týchto hodnôt nám udáva optimálnu cenu výstroja pre tretieho horolezca.

Niečo vhodné (napríklad intervalový strom) si predpočítame, aby sme vedeli povedať minimum ľubovoľného úseku v $O(\log n)$.
Na začiatku má druhý horolezec cenu výstroja ako prvý, a teda nie je žiaden vynútený vrchol a všetky vrcholy tvoria jeden úsek (ktorí vedia prví dvaja horolezci vyriešiť bez pomoci). V usporiadanej multimnožine (na

začiatku prázdnej) si budeme pamätať minimá všetkých úsekov párnej dĺžky.
Vždy, keď znížime cenu výstroja druhého horolezca, postupne jeden po druhom pridáme nové vynútené vrcholy. Každý z nich nám rozdelí jeden z dovtedajších úsekov na dva kratšie. Vždy, keď sa toto stane, v čase $O(\log n)$ zistíme minimá starého úseku aj oboch nových úsekov (ak ich podľa parity ich dĺžky treba) a potom v $O(\log n)$ upravíme našu multimnožinu miním. Keď pridáme všetky nové vynútené vrcholy, opäť v $O(\log n)$ zistíme maximum našej multimnožiny.
Každý vrchol sa najviac raz stane vynúteným a pre každý vrchol najviac raz budeme skúšať jeho výšku ako cenu výstroja druhého horolezca. Dokopy má toto riešenie teda časovú zložitosť $O(n \log n)$. Lepšie to nejde, lebo už len na samotné usporiadanie výšok vrcholov potrebujeme takýto čas.

Obľúbené chyby

Pomerne často sa vyskytlo nesprávne „pažravé“ riešenie, ktoré sa snaží pre troch horolezcov optimálny rozvrh spraviť tak, že prvému horolezcovi kúpi výstroj na najvyšší vrchol a potom prechádza vrcholy od najvyššieho po najnižší a na každý, na ktorý vie prvého horolezca poslať, ho aj pošle.
Toto zlyhá napríklad pre výšky $(100, 90, 80, 100)$, kde vyrobíme rozvrh ABCA s cenou $100 + 90 + 80$, pričom optimálne je ABAB s cenou $100 + 100$.
A nepomôže ani ak vopred zvlášť ošetríme možnosť, že sú horolezci len dvaja. Aj potom dá toto riešenie nesprávny výstup napr. pre vstup $(1000, 1, 100, 90, 80, 100, 1, 1000)$.

Druhou kategóriou zlých riešení boli riešenia dynamickým programovaním, ktoré sa ale snažili zapamätať si pre konkrétny stav len niektoré možné spôsoby ako ho dosiahnuť, zväčša len tie lokálne optimálne. Ukážeme si teraz pre ilustráciu jednu možnú situáciu, kedy toto nevedie ku globálne optimálnemu riešeniu.
Uvažujme napríklad vstup, v ktorom má najvyšší vrchol výšku 1000 a ktorý začína $(1000, 30, 40, 100, 7)$. Existujú dva neporovnateľné spôsoby, ako dosiahnuť stav, kde na poslednú horu nejde prvý horolezec. Pri jednom z nich majú zvyšní dvaja výstroj ceny 30+40, pri druhom 100+7.
Keby teraz vstup ďalej obsahoval horu výšky 999 a tým skončil, viedla by prvá možnosť skutočne k optimálnemu riešeniu. No ani na tú druhú možnosť nemôžeme v tejto chvíli zabudnúť: ak by vstup pokračoval $(999, 800, 999)$, tak by optimálnym riešením bolo nakúpiť výstroje za $1000 + 800 + 7$.

A-III-3 Exaktné exponenciálne algoritmy

Obe súťažné úlohy sú vlastne úlohami o farbení grafov. Graf, ktorý farbíme, má vo vrcholoch zvieratká a hrany predstavujú konflikty medzi nimi. Rôzne farby predstavujú rôzne výbehy. Vrcholom grafu teda chceme priradiť farby tak, aby žiadne dva vrcholy spojené hranou nemali tú istú farbu.
V podúlohe A sa pýtame, či sa zadaný graf dá ofarbiť tromi farbami (ktoré budeme pre jednoduchosť volať červená, zelená a modrá).
V podúlohe B sa pýtame, aký najmenší počet farieb nám stačí na ofarbenie daného grafu.
(Pre zaujímavosť uvedieme, že k týmto úlohám nielen že nepoznáme žiadne efektívne algoritmy, máme aj dobrý dôvod domnievať sa, že takéto algoritmy pre ne neexistujú – ide o tzv. NP-ťažké problémy. Efektívne exponenciálne algoritmy sú teda najefektívnejším exaktným riešením týchto problémov, ktoré poznáme.)

Podúloha A, pomalšie riešenia

Všetkých možností, ktoré zviera dať do ktorého výbehu, je 3^n. Ak teda chceme časovú zložitosť $\hat{O}(2^n)$, potrebujeme robiť niečo šikovnejšie. Ukážeme si dva možné prístupy.

Prvý je založený len na šikovnejšom skúšaní možností. Začneme tým, že si graf rozdelíme na komponenty súvislosti. Zjavne stačí skontrolovať každý z nich zvlášť. Keď teraz máme súvislý graf, všetky možnosti jeho ofarbenia budeme skúšať backtrackingom. Vrchol, kde začneme, môžeme bez ujmy na všeobecnosti rovno ofarbiť červenou. Každý ďalší vrchol na skúšanie si vyberieme tak, aby susedil s aspoň jedným vrcholom, ktorý sme už ofarbili. Takto dosiahneme, že pre každý vrchol budeme mať na výber najviac dve možné farby (lebo mu

nesmieme dať tú istú farbu ako už má jeho sused). Dokopy teda aj v najhoršom prípade vyskúšame nanajvýš 2^{n-1} rôznych ofarbení.

Druhý spôsob je založený na pozorovaní, že tú istú otázku pre dve farby namiesto troch už vieme zodpovedať efektívne (t.j. v polynomiálnom čase). Ide vlastne o otázku, či je graf bipartitný. S efektívnym riešením tejto úlohy ste sa stretli v krajskom kole. Úlohu pre tri farby teraz môžeme vyriešiť tak, že vyskúšame všetkých 2^n možností, ktoré vrcholy sú červené. Pre každú z nich v polynomiálnom čase overíme, či sa zvyšné vrcholy dajú ofarbiť zelenou a modrou.

Podúloha A, vzorové riešenie

Vzorové riešenie dostaneme tak, že vylepšíme druhé z vyššie uvedených pomalších riešení. Budeme pri tom potrebovať ešte jeden pojem zo študijného textu: nezávislé množiny. Keď sa pozrieme na korektne ofarbený graf, je zjavné, že každá farba nám určuje jednu nezávislú množinu jeho vrcholov. Vyššie uvedené riešenie by sme teda vedeli vylepšiť tak, že namiesto všetkých 2^n podmnožín vrcholov budeme skúšať len tie podmnožiny, ktoré sú nezávislé. Tých však stále môže byť rádovo 2^n, takže samo o sebe toto pozorovanie nestačí. Sme ale na dobrej ceste.

Nezávislú množinu vrcholov X nazveme *nezväčšiteľná*, ak do nej už nevieme pridať žiadny ďalší vrchol grafu tak, aby ostala nezávislou.
(Pozor, nezväčšiteľné nezávislé množiny nie sú nutne všetky rovnako veľké. Každá najväčšia nezávislá množina je z pochopiteľných dôvodov nezväčšiteľná, naopak to však platiť nemusí.)

Tvrdenie 1: Ak sa zadaný graf dá korektne ofarbiť tromi farbami, tak sa dá ofarbiť tak, aby množina červených vrcholov bola nezväčšiteľná.
Dôkaz: Zoberme ľubovoľné platné ofarbenie. Postupne sa pozrime na každý vrchol, ktorý nie je červený. Ak ani žiadny jeho sused nie je červený, prefarbíme ho na červeno. Keď tento proces skončí, máme naďalej platné ofarbenie, a zároveň je zjavné, že nová množina červených vrcholov je už nezväčšiteľná.

Tvrdenie 2: Všetkých možných nezväčšiteľných nezávislých množín je nanajvýš $3^{n/3}$ a v čase $\hat{O}(3^{n/3})$, čiže $\hat{O}(1.4423^n)$, ich vieme všetky vygenerovať.
Dôkaz: „Lepší algoritmus 2“ zo študijného textu v skutočnosti rieši presne túto úlohu.
(V študijnom texte sme ním síce riešili o čosi inú úlohu – nájdenie najväčšej nezávislej množiny – no ľahko nahliadneme, že tento algoritmus v skutočnosti dokonca vygeneruje každú nezväčšiteľnú množinu.)

Takto teda dostávame riešenie podúlohy A v čase $\hat{O}(1.4423^n)$. Ešte raz si ho stručne zhrnieme: použijeme Lepší algoritmus 2 zo študijného textu na vygenerovanie všetkých možností, ktoré vrcholy budú červené. Pre každú z nich v polynomiálnom čase overíme, či sa zvyšok grafu dá ofarbiť dvomi farbami.

Podúloha B, exponenciálne riešenie

V najhoršom prípade môže byť potrebných až n farieb – ak sa žiadne dve zvieratká neznesú, potrebuje každé vlastný výbeh. Priamočiare skúšanie všetkých možností backtrackingom bude teda mať časovú zložitosť až rádovo n^n, čiže ešte horšiu ako exponenciálnu.
Aby sme dosiahli exponenciálnu časovú zložitosť, nebudeme vrcholy farbiť jeden po druhom, ale budeme jednu po druhej pridávať celé farby.
Náš algoritmus si môžeme predstaviť ako rekurzívnu funkciu, ktorá na vstupe dostane množinu ešte neofarbených vrcholov a na výstupe vráti najmenší počet ďalších farieb potrebný na ich korektné ofarbenie. Telo tejto funkcie bude vyzerať nasledovne: Ak je množina prázdna (čiže už sme ofarbili všetko), vrátime nulu. Inak zoberieme nasledujúcu farbu a vyskúšame všetky možnosti, ktoré neofarbené vrcholy ju dostanú a ktoré nie. Pre každú z týchto možností rekurzívnym volaním našej funkcie zistíme, koľko ďalších farieb ešte treba na dokončenie farbenia, a vyberieme si najlepšiu z týchto možností.
No a už si stačí uvedomiť, že v tejto chvíli môžeme použiť memoizáciu – pre každú z 2^n podmnožín vrcholov raz spočítame jej optimálne riešenie a zapamätáme si ho.

Graf, ktorý farbíme, má 2^n podmnožín vrcholov, potrebujeme teda zodpovedať 2^n otázok. Pri každej z nich vyskúšame nanajvýš 2^n možností pre to, ktoré vrcholy dostanú nasledujúcu farbu. Dokopy teda vieme časovú zložitosť zhora odhadnúť $\hat{O}(4^n)$ – úspešne sme teda zostrojili riešenie s exponenciálnou časovou zložitosťou.
Toto isté riešenie si môžeme samozrejme sformulovať aj ako iteratívne dynamické programovanie: postupne pre každú podmnožinu vrcholov zadaného grafu, od najmenších po najväčšie, zistíme optimálny počet farieb potrebných pre ňu vyskúšaním vyššie popísaných možností. V praxi by sa takéto riešenie dobre implementovalo pomocou bitových masiek.

Podúloha B, vzorové riešenie

Vyššie uvedený odhad časovej zložitosti nie je tesný a šikovnejšou úvahou ho vieme zlepšiť. Stačí si všimnúť, že vždy, keď skúšame nejakú konkrétnu možnosť, ako pridať jednu novú farbu, je každý vrchol v jednom z troch stavov: už ofarbený, práve dostáva farbu, alebo ostáva neofarbený. Dokopy pri zodpovedaní všetkých otázok teda vyskúšame len 3^n možností, čiže pri šikovnej implementácii vie mať vyššie popísané riešenie časovú zložitosť dokonca $\hat{O}(3^n)$.
Vyššie uvedený algoritmus vieme vylepšiť rovnako ako sme to spravili v podúlohe A. Stačí si uvedomiť, že aj tu platí analógia Tvrdenia 1: vždy, keď skúšame všetky možnosti pre novú farbu, stačí skúšať tie, ktoré sú nezväčšiteľné. Už priamočiary odhad nám povie, že takto pre každú z 2^n otázok vyskúšame nanajvýš 1.4423^n možností, ako pridať novú farbu, a teda budeme mať časovú zložitosť $\hat{O}(2.8846^n)$.
Presnejšou analýzou tohto algoritmu sa opäť dá ukázať aj tesnejší horný odhad jeho časovej zložitosti: $\hat{O}(2.4423^n)$.

TRIDSIATY PIATY ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Anderle, Michal Forišek
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2020