Informácie a pravidlá

Pre koho je súťaž určená?

Do **kategórie B** sa smú zapojiť len tí žiaci základných a stredných škôl, ktorí ešte ani v tomto, ani v nasledujúcom školskom roku nebudú končiť strednú školu.
Do **kategórie A** sa môžu zapojiť všetci žiaci (základných aj) stredných škôl.

Odovzdávanie riešení domáceho kola

Riešitelia domáceho kola odovzdávajú riešenia sami, v elektronickej podobe, a to priamo na stránke olympiády: http://oi.sk/. Odovzdávanie riešení bude spustené niekedy v septembri.

Riešenia kategórie A je potrebné odovzdať najneskôr **15. novembra 2019**.
Riešenia kategórie B je potrebné odovzdať najneskôr **30. novembra 2019**.

Priebeh súťaže

Za každú úlohu domáceho kola sa dá získať od 0 do 10 bodov. Na základe bodov domáceho kola stanoví Slovenská komisia OI (SK OI) pre každú kategóriu bodovú hranicu potrebnú na postup do **krajského kola**. Očakávame, že táto hranica bude približne rovná **tretine maximálneho počtu bodov**.

V krajskom kole riešitelia riešia štyri teoretické úlohy, ktoré môžu tematicky nadväzovať na úlohy domáceho kola. V kategórii B súťaž týmto kolom končí.

V kategórii A je približne najlepších 30 riešiteľov krajského kola (podľa počtu bodov, bez ohľadu na kraj, v ktorom súťažili) pozvaných do **celoštátneho kola**. V celoštátnom kole účastníci prvý deň riešia teoretické a druhý deň praktické úlohy. Najlepší riešitelia sú vyhlásení za víťazov. Približne desať najlepších riešiteľov následne SK OI pozve na týždňové výberové sústredenie. Podľa jeho výsledkov SK OI vyberie družstvá pre Medzinárodnú olympiádu v informatike (IOI) a Stredoeurópsku olympiádu v informatike (CEOI).

Ako majú vyzerať riešenia úloh?

V praktických úlohách je vašou úlohou vytvoriť program, ktorý bude riešiť zadanú úlohu. Program musí byť v prvom rade korektný a funkčný, v druhom rade sa snažte aby bol čo najefektívnejší.

V kategórii B môžete použiť ľubovoľný programovací jazyk.

V kategórii A musíte riešenia praktických úloh písať v jednom z podporovaných jazykov (napr. C++, Pascal alebo Java). Odovzdaný program bude automaticky otestovaný na viacerých vopred pripravených testovacích vstupoch. Podľa toho, na koľko z nich dá správnu odpoveď, vám budú pridelené body. Výsledok testovania sa dozviete krátko po odovzdaní. Ak váš program nezíska plný počet bodov, budete ho môcť vylepšiť a odovzdať znova, až do uplynutia termínu na odovzdávanie.

Presný popis, ako majú vyzerať riešenia praktických úloh (napr. realizáciu vstupu a výstupu), nájdete na webstránke, kde ich budete odovzdávať.

Ak nie je v zadaní povedané ináč, riešenia teoretických úloh musia v prvom rade obsahovať **podrobný slovný popis použitého algoritmu, zdôvodnenie jeho správnosti** a diskusiu o efektivite zvoleného riešenia (t. j. posúdenie časových a pamäťových nárokov programu). Na záver riešenia uveďte program. Ak používate v programe netriviálne algoritmy alebo dátové štruktúry (napr. rôzne súčasti STL v C++), súčasťou popisu algoritmu musí byť dostatočný popis ich implementácie.

Usporiadateľ súťaže

Olympiádu v informatike (OI) vyhlasuje *Ministerstvo školstva SR* v spolupráci so *Slovenskou informatickou spoločnosťou* (odborným garantom súťaže) a *Slovenskou komisiou Olympiády v informatike*. Súťaž organizuje *Slovenská komisia OI* a v jednotlivých krajoch ju riadia *krajské komisie OI*. Na jednotlivých školách ju zaisťujú učitelia informatiky. Celoštátne kolo OI, tlač materiálov a ich distribúciu po organizačnej stránke zabezpečuje IUVENTA v tesnej súčinnosti so Slovenskou komisiou OI.

A-I-1 Po schodoch

*Toto je **praktická úloha**. Pomocou webového rozhrania odovzdajte **funkčný, odladený program**.*

Na horu Inari vedie dlhočizné nepravidelné schodisko. Schody má očíslované zdola nahor od 1 po n. Výšku schodu i označíme h_i.
Mních Takeši každé ráno nesie vedro vody hore schodiskom na horu Inari. Začína pri studni tesne pred schodom 1 a končí až pri svätyni stojacej na vrchu schodu n.
Takeši každým krokom stúpi aspoň o schod vyššie, občas ale spraví dlhší krok a nejaké schody vynechá. Najväčší výškový rozdiel, ktorý ešte zvláda jedným krokom prekonať, je d. (Ľubovoľný takýto krok vie spraviť, bez ohľadu na to, koľko schodov pri tom vynechá.)
Pre daný popis schodiska vypočítajte, koľkými rôznymi spôsobmi vie Takeši vyjsť na horu.

Formát vstupu a výstupu

V prvom riadku vstupu je počet schodov n a maximálna výška kroku d.
V druhom riadku vstupu sú celé čísla $h_1, \ldots, h_n$ udávajúce výšky schodov.

Na výstup vypíšte jeden riadok a v ňom jedno celé číslo: zvyšok, ktorý dáva hľadaný počet spôsobov po delení číslom $10^9 + 7$.

Obmedzenia a hodnotenie

Odovzdané riešenie bude otestované na piatich sadách vstupov, za každú sú dva body. V každej sade platí $\forall i : 1 \leq h_i \leq 10^9$ a tiež $(\max h_i) \leq d \leq 10^9$. V jednotlivých sadách platia nasledovné ďalšie obmedzenia:

- Sada #1: $n \leq 16$.
- Sada #2: $n \leq 50$ a v každom vstupe existuje nanajvýš 10^6 rôznych spôsobov výstupu na horu.
- Sada #3: $n \leq 1000$.
- Sada #4: $n \leq 100\,000$ a $d \leq 10$.
- Sada #5: $n \leq 500\,000$.

Príklad

vstup	výstup
4 100 20 30 50 30	6

Číslom i označíme stav, v ktorom Takeši práve stojí na i-tom schode – špeciálne 0 označuje stav, kedy ešte nezačal stúpať do schodov. Existuje šesť rôznych spôsobov, ktorými vie Takeši vyjsť na túto horu. Zodpovedajú im nasledujúce postupnosti stavov: 01234, 0124, 0134, 0234, 024 a 034.

A-I-2 Nová bankovka

*Toto je **praktická úloha**. Pomocou webového rozhrania odovzdajte **funkčný, odladený program**.*

Kocúrkovo má systém platidiel veľmi podobný tomu nášmu. Najmenšou nominálnou hodnotou je 1 toliar. Existujú mince a bankovky s nasledujúcimi peknými, systematicky zvolenými nominálnymi hodnotami: 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10 000, 20 000 a 50 000 toliarov.
Kocúrkovská centrálna banka sa ale rozhodla, že vydá novú bankovku v hodnote presne x toliarov.
Obyvatelia Kocúrkova si teraz kladú veľa otázok, a to hlavne nasledovného tvaru: „Ak budem chcieť zaplatiť presne t_i toliarov, koľko najmenej kusov platidiel stačí použiť?“

Formát vstupu a výstupu

V prvom riadku vstupu je číslo x: nominálna hodnota novej bankovky. V druhom riadku je číslo q: počet otázok, ktoré majú obyvatelia Kocúrkova. V treťom riadku sú medzerou oddelené čísla $t_1, \ldots, t_q$: sumy pre jednotlivé otázky.

Na výstup vypíšte q riadkov: postupne pre každú otázku jeden riadok s odpoveďou na ňu.

Obmedzenia a hodnotenie

Odovzdané riešenie bude otestované na piatich sadách vstupov, za každú sú dva body.
Je zaručené, že x **bude vždy rôzne** od nominálnych hodnôt existujúcich platidiel.
V jednotlivých sadách platia nasledovné obmedzenia na veľkosť premenných:

sada	x	q	všetky t_i
#1	≤ 20	≤ 50	≤ 20
#2	$= 100\,000$	≤ 50	$\leq 10^5$
#3	$\leq 10^5$	≤ 50	$\leq 10^5$
#4	$\leq 10^7$	≤ 50	$\leq 10^7$
#5	$\leq 2 \cdot 10^9$	≤ 50	$\leq 2 \cdot 10^9$

Príklady

vstup

```
4700
4
53 9400 9401 30000
```

výstup

```
3
2
3
2
```

Nová bankovka má hodnotu 4700 toliarov.

- *Najlepší spôsob ako zaplatiť 53 toliarov je použiť platidlá s hodnotami 50 + 2 + 1.*
- *Najlepší spôsob ako zaplatiť 9400 toliarov je použiť dve nové bankovky.*
- *Najlepší spôsob ako zaplatiť 9401 toliarov je použiť dve nové bankovky a jednu 1-toliarovú mincu.*
- *Najlepší spôsob ako zaplatiť 30000 toliarov je 10000 + 20000.*

A-I-3 Rekonštrukcia mapy

Toto je teoretická úloha. Pomocou webového rozhrania odovzdajte súbor vo formáte PDF, obsahujúci riešenie, spĺňajúce požiadavky uvedené v pravidlách.

Cestovateľ Parko Mólo nedávno navštívil jedno ostrovné kráľovstvo. Pozostávalo z n ostrovov, ktoré sa jeden druhému podobali ako vajce vajcu. Niektoré dvojice ostrovov boli prepojené mostami, a to tak, že sa po mostoch dalo z ľubovoľného ostrova na ľubovoľný iný prejsť práve jedným spôsobom. (Mostov teda bolo práve $n - 1$ a odborne hovoríme, že kráľovstvo malo stromovú topológiu.)
Keď už Parko z kráľovstva odcestoval, spomenul si, že si vlastne úplne zabudol nakresliť jeho mapu. Jediné, čo ešte našiel v svojom notese, je zoznam, do ktorého si zapísal, z ktorého ostrova viedlo koľko mostov. Ale ani správnosti tohto zoznamu úplne nedôveruje.

Súťažná úloha

Zadané sú čísla $d_1, \ldots, d_n$. Zistite, či existuje ostrovné kráľovstvo s vyššie popísanými vlastnosťami, ktorého ostrovy sa dajú očíslovať od 1 po n tak, aby pre každé i platilo, že z ostrova i vedie presne d_i mostov.
Ak takéto kráľovstvo existuje, zostrojte jeho (jednu možnú) mapu – presnejšie, zoznam mostov v ňom.

Formát vstupu a výstupu

V prvom riadku vstupu je kladné celé číslo n. V druhom riadku vstupu sú medzerou oddelené kladné celé čísla $d_1, \ldots, d_n$. (Tieto čísla sa zmestia do bežnej celočíselnej premennej, ale nič iné o ich veľkosti nepredpokladajte.)

Ak takéto kráľovstvo neexistuje, vypíšte jeden riadok a v ňom reťazec „`neexistuje`“.
Ak takéto kráľovstvo existuje, výstup má tvoriť $n - 1$ riadkov a v každom z nich dve čísla z rozsahu 1 až n: čísla dvoch ostrovov spojených mostom. Tieto čísla majú byť zvolené tak, aby sa z každého ostrovu dalo po mostoch dostať na každý iný a navyše platilo, že z každého ostrova i vedie presne d_i mostov. (Zväčša existuje veľa vyhovujúcich sád mostov, vypísať môžete ľubovoľnú z nich.)

Obmedzenia a hodnotenie

Riešenia s optimálnou časovou zložitosťou môžu získať plný počet bodov: 10.
Iné riešenia, ktoré sú dostatočne efektívne na to, aby na bežnom počítači za pár sekúnd vyriešili vstup s $n = 10^6$, môžu získať nanajvýš 8 bodov.
Riešenia dostatočne efektívne pre $n = 5\,000$ môžu získať nanajvýš 6 bodov.
Za ľubovoľné funkčné riešenie sa dá získať aspoň 3 body, bez ohľadu na to, ako je pomalé.
Nezabudnite, že dôležitou súčasťou riešenia je dôkaz jeho správnosti!

Príklady

vstup

```
3
4 1 2
```

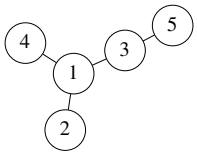
výstup

```
neexistuje
```

Rozmyslite si, že v žiadnom kráľovstve nemôžu byť dva ostrovy priamo prepojené viac ako jedným mostom. Ak teda máme len tri ostrovy, nemôže existovať ostrov, z ktorého vedú až štyri mosty.

vstup

```
5
3 1 2 1 1
```

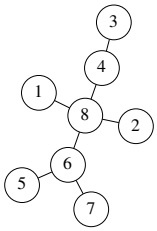


výstup

```
1 3
1 2
1 4
3 5
```

vstup

```
8
1 1 1 2 1 3 1 4
```



výstup

```
1 8
2 8
3 4
4 8
5 6
6 7
6 8
```

A-I-4 Exaktné exponenciálne algoritmy

Toto je teoretická úloha. Pomocou webového rozhrania odovzdajte súbor vo formáte PDF, obsahujúci riešenie, spĺňajúce požiadavky uvedené v pravidlách. K tejto úlohe patrí študijný text uvedený na nasledujúcich stranách. Odporúčame najskôr prečítať ten a až potom sa vrátiť k samotným súťažným úlohám.

Podúloha A (2 body). V úlohe o maximálnej nezávislej množine uvažujme teraz len vstupy, v ktorých platí, že každé zviera má nanajvýš dva konflikty. Nájdite algoritmus s polynomiálnou časovou zložitosťou, ktorý pre ľubovoľný takýto vstup vypočíta, koľko najviac zvierat vieme pustiť do výbehu.

Podúloha B (4 body). Ukážte, ako pomocou algoritmu z podúlohy A zlepšiť „lepší algoritmus 1“ zo študijného textu. Akú časovú zložitosť bude mať algoritmus, ktorý takto dostanete?

Podúloha C (4 body). Máme n škatúľ. Každá škatuľa má nejakú výšku (v cm), nejakú hmotnosť a nejakú nosnosť (oboje v kg). Chceme z nejakých škatúľ postaviť vežu: niektorú škatuľu položíme na zem, druhú na prvú, tretiu na druhú, a tak ďalej. Škatule môžeme na vežu klásť v ľubovoľnom poradí. Veža je stabilná, ak pre každú škatuľu platí, že súčet hmotností škatulí nad ňou je menší alebo rovný jej nosnosti.
Navrhnite algoritmus s časovou zložitosťou $\hat{O}(2^n)$, ktorý zistí, akú najvyššiu stabilnú vežu vieme z danej sady škatúľ postaviť.

Študijný text: Exaktné exponenciálne algoritmy

Pri analýze algoritmov sa často stretneme so zjednodušeným tvrdením, že algoritmy s *polynomiálnou* časovou zložitosťou považujeme za efektívne, zatiaľ čo algoritmy s *exponenciálnou* (a horšou) časovou zložitosťou považujeme za neefektívne. V tomto ročníku olympiády trochu nahliadneme do sveta exponenciálnych algoritmov a uvidíme, že toto zjednodušenie nemusí byť vždy pravdivé.

Porovnanie časových zložitostí

Pre súčasné počítače môžeme odhadnúť, že za minútu zvládnu vykonať približne 10^{10} jednoduchých logických krokov programu. Ak teda máme algoritmus s časovou zložitosťou $f(n)$ a zaujíma nás, aké veľké vstupy dokáže za minútu vyriešiť, hľadáme jednoducho najväčšie n také, že $f(n) \leq 10^{10}$. Výsledky pre niektoré zaujímavé funkcie uvádzame v tabuľke.

$f(n)$	$n \log n$	n^2	n^3	$3n^4$	2^n	1.42^n	1.1^n	$n!$
$\max n$	500 000 000	100 000	2154	240	33	66	241	13

Vidíme teda, že napríklad medzi polynomiálnou časovou zložitosťou $3n^4$ a exponenciálnou časovou zložitosťou 1.1^n nie je v praxi zase až taký veľký rozdiel: rozsahy efektívne riešiteľných vstupov majú oba skoro rovnako veľké.

Možno vám v tabuľke padlo do oka, že 66 je dvakrát 33. Toto nie je náhoda. Totiž $\sqrt{2}^n = \left(2^{1/2}\right)^n = 2^{n/2}$. No a to znamená, že ak časovú zložitosť algoritmu zlepšíme z 2^n na $\sqrt{2}^n \approx 1.42^n$, tak nový algoritmus zvládne za rovnaký čas vyriešiť približne dvakrát väčší vstup ako ten pôvodný.
Túto úvahu vieme aj zovšeobecniť. Výraz a^n môžeme upraviť nasledovne: plati $a = 2^{\log_2 a}$, a teda $a^n = \left(2^{\log_2 a}\right)^n = 2^{n \log_2 a}$. Napríklad takto dostaneme, že 1.1^n je približne to isté ako $2^{0.1375n}$, resp. $2^{n/7.27254}$. Zlepšenie časovej zložitosti z 2^n na 1.1^n teda znamená, že novým algoritmom v rovnakom čase vyriešime vyše sedemkrát väčší vstup ako pôvodným. Všeobecne teda platí, že každé zlepšenie základu exponenciálnej funkcie niekoľko*krát* zväčší rozsah vstupov, ktoré ešte vieme efektívne riešiť.

O ťažkých problémoch

V teoretickej informatike poznáme značné množstvo algoritmických problémov, ktoré sú *ťažké*: nepoznáme pre ne žiadne algoritmy s polynomiálnou časovou zložitosťou a často máme dobré dôvody domnievať sa, že takáto časová zložitosť sa pre tieto problémy vôbec nedá dosiahnuť. (Exaktný dôkaz tejto domnienky pre jednu konkrétnu sadu ťažkých problémov je jedným z najvýznamnejších otvorených problémov v informatike.)
Z pozorovaní, ktoré sme spravili v predchádzajúcej časti študijného textu, však vyplýva jeden možný „smer útoku“: ak narazíme na takýto problém a potrebujeme ho vedieť exaktne riešiť, jednou z možností, ktoré máme, je snažiť sa nájsť taký exponenciálny algoritmus, ktorého základ exponenciálnej funkcie bude čo najmenší. Čím bližšie k jednotke ho dostaneme, tým väčšie vstupy ešte zvládneme v rozumnom čase vyriešiť.
Vo zvyšku tohto študijného textu si ukážeme dva takéto ťažké problémy a predvedieme si na nich dve techniky návrhu šikovných exponenciálnych algoritmov.

Zápis časovej zložitosti exponenciálnych algoritmov

Pri odhade časovej zložitosti klasických efektívnych algoritmov zvykneme zanedbávať konštanty. Namiesto exaktného vyčíslenia, že algoritmus na vstupe veľkosti n spraví nanajvýš $7n^2 - 3n + 147$ krokov, sa uspokojíme s asymptotickým odhadom „časová zložitosť algoritmu je $O(n^2)$“ – čiže „časová zložitosť je nejaká funkcia, ktorá rastie nanajvýš rádovo tak rýchlo ako funkcia n^2“.
Pri analýze exponenciálnych algoritmov niekedy budeme podobným spôsobom chcieť zanedbať aj polynomiálne faktory. Takýto horný odhad budeme zapisovať $\hat{O}$. Napríklad funkcie 1.9^n, $100 \cdot 2^n$ aj $(3n^2 + 6)2^n + n^4$ patria do $\hat{O}(2^n)$, ale funkcia $0.047 \cdot 2.01^n$ tam už nepatrí.
Formálne, funkcia f patrí do $\hat{O}(g)$ práve vtedy, ak patrí do $O(p \cdot g)$ pre nejaký polynóm p.

Časová zložitosť rekurzívnych programov

Niektoré exaktné exponenciálne algoritmy sú založené na *backtrackingu* (teda rekurzívnom prehľadávaní s návratom). Pri analýze ich časovej zložitosti budeme používať nasledujúce tvrdenie:

Veta o časovej zložitosti rekurzie. Majme rekurzívny algoritmus A, ktorý pri riešení problému postupuje nasledovne: Ak má vstup malej konštantnej veľkosti, vyrieši ho v konštantnom čase. Vo všeobecnom prípade pre vstup veľkosti n postupne spraví k rekurzívnych volaní, pričom pri i-tom z nich sa zavolá na vstup veľkosti nanajvýš $n - a_i$. (Hodnoty k aj a_i sú konštanty, ktoré ostávajú rovnaké počas celého behu algoritmu.) Okrem týchto rekurzívnych volaní už algoritmus spraví len polynomiálne veľa krokov v závislosti od n. Potom platí, že časová zložitosť tohto algoritmu je $\hat{O}(\alpha^n)$, kde α je jediné kladné reálne riešenie rovnice

$$x^n - x^{n-a_1} - \cdots - x^{n-a_k} = 0$$

Náčrt dôkazu. Keď si označíme časovú zložitosť nášho algoritmu T, z popisu algoritmu A dostávame, že T spĺňa rekurentný vzťah: $T(n) = T(n-a_1) + \cdots + T(n-a_k) + p(n)$, kde p je nejaký polynóm. Ak zanedbáme p a hľadáme čistú exponenciálnu funkciu T spĺňajúcu tento rekurentný vzťah, teda položíme $T(n) = \alpha^n$, dostaneme

pre α práve vyššie uvedenú rovnicu. Následne sa dá ukázať, že keď za α zoberieme jej kladné reálne riešenie, tak náš algoritmus skutočne spraví $O(\alpha^n)$ rekurzívnych volaní, a teda celkový čas jeho behu vieme zhora odhadnúť $O(p(n) \cdot \alpha^n)$.

Príklady použitia. Ak algoritmus pri riešení problému veľkosti n spraví dve rekurzívne volania na problémy veľkosti $n-1$, dostávame rovnicu $x^n - x^{n-1} - x^{n-1} = 0$. Keďže hľadáme kladný reálny koreň, môžeme obe strany vydeliť nenulovým výrazom x^{n-1} a dostávame $x - 1 - 1 = 0$, čiže $x = 2$. Tento algoritmus má teda časovú zložitosť $\hat{O}(2^n)$.
Ak však algoritmus spraví jedno rekurzívne volanie na problém veľkosti $n-1$ a jedno na problém veľkosti $n-3$, dostaneme rovnakou úvahou rovnicu $x^3 - x^2 - 1 = 0$. Tej jediný kladný reálny koreň je $x \approx 1.4656$. Platí teda, že takýto algoritmus má časovú zložitosť $\hat{O}(1.4656^n)$. (Technický detail: všetky približné číselné konštanty uvádzame zaokrúhlené *nahor*.)

Maximálna nezávislá množina

V zoologickej záhrade práve postavili nový výbeh. Majú n zvierat, ktoré by doň chceli vypustiť. Problém je ale v tom, že niektoré dvojice zvierat nemôžu byť spolu vo výbehu, lebo by sa hrýzli. Na vstupe dostanete zoznam všetkých m takýchto dvojíc. Navrhnite algoritmus, ktorý zistí, koľko najviac zvierat môže skončiť vo výbehu.

Skôr, než sa pustíme do lepších riešení, podotknime, že túto úlohu vieme ľahko riešiť s časovou zložitosťou $O(m2^n)$: Existuje presne 2^n rôznych podmnožín zvierat. Postupne každú z nich vygenerujeme a zakaždým prejdeme celý zoznam dvojíc a pozeráme sa, či sme náhodou nevybrali obe zvieratá z niektorej dvojice.

Maximálna nezávislá množina: lepší algoritmus 1

Náš algoritmus bude mať podobu rekurzívnej funkcie, ktorá dostane na vstupe nejakú množinu zvierat a na výstupe vráti číslo hovoriace koľko najviac spomedzi týchto zvierat môžeme dať do prázdneho výbehu.
Všimnime si nejaké zviera z. Optimálne riešenie, ktoré **neobsahuje** z, nájdeme tak, že sa rekurzívne zavoláme na všetky zvieratá okrem z. Ako nájsť optimálne riešenie, ktoré **obsahuje** z? Označme $N(z)$ množinu tých zvierat, ktoré nemôžu byť vo výbehu spolu so zvieraťom z. Ak sa rozhodneme do výbehu pustiť zviera z, vieme, že zvieratá z $N(z)$ do výbehu pustiť nemôžeme. Optimálne riešenie obsahujúce z teda vieme získať tak, že sa rekurzívne zavoláme na všetky zvieratá okrem z a zvierat z $N(z)$, a následne do takto získaného riešenia pridáme zviera z. Na výstup naša funkcia vráti väčšie z oboch vyššie popísaných riešení.
Je zjavné, že za zviera z sa oplatí voliť také zviera, ktoré má *čo najviac* konfliktov s inými – aby sme pri druhom rekurzívnom volaní dostali čo najmenšiu množinu zvyšných zvierat. Toto pozorovanie nás privádza k nasledujúcemu algoritmu:

1. Ak má každé zvieratko najviac jeden konflikt: Zober všetky bezkonfliktné zvieratká. Z každej dvojice, čo je v konflikte, zober ľubovoľné jedno. Hotovo.
2. Inak si vyber zvieratko z ktoré má najviac konfliktov s inými.
3. Rekurzívne nájdi najlepšie riešenie pre všetky zvieratká okrem z.
4. Rekurzívne nájdi najlepšie riešenie pre všetky zvieratká okrem z a $N(z)$, a pridaj doň z.
5. Na výstup vráť lepšie z týchto dvoch riešení.

Pre veľké vstupy tento algoritmus vždy spraví dve rekurzívne volania. Prvé je na vstup veľkosti $n-1$. Keďže vybrané zvieratko z má aspoň dva konflikty, druhé rekurzívne volanie je na problém veľkosti nanajvýš $n-3$. Z vety o časovej zložitosti rekurzie teda vyplýva, že toto riešenie má časovú zložitosť $\hat{O}(1.4656^n)$.

Maximálna nezávislá množina: lepší algoritmus 2

Aj tento algoritmus bude mať podobu rekurzívnej funkcie, ktorá dostane na vstupe nejakú množinu zvierat a na výstupe vráti číslo hovoriace koľko najviac spomedzi týchto zvierat môžeme dať do prázdneho výbehu.
Pripomeňme si, že optimálne riešenie, ktoré **obsahuje** zviera z, vieme nájsť tak, že nájdeme optimálne riešenie pre všetky zvieratá okrem z a $N(z)$, a potom doň pridáme z. Tentokrát budeme pokračovať trochu inou myšlienkou. Tvrdíme, že v optimálnom riešení, ktoré z **neobsahuje**, musí byť vo výbehu aspoň jedno zo zvierat

v $N(z)$. Toto je dosť zjavné: riešenie, v ktorom nie je vo výbehu ani z ani žiadne zviera z $N(z)$, nemôže byť optimálne, lebo ho vieme zlepšiť pustením z do výbehu.
Uvažujme teda nasledujúci algoritmus (slovo „najmenej“ v kroku 1 vysvetlíme nižšie):

1. Vyber si zvieratko z, ktoré má **najmenej** konfliktov s inými.
2. Pre každé zvieratko y z množiny $\{z\} \cup N(z)$:
   – Rekurzívnym volaním nájdi najlepšie riešenie pre všetky zvieratká okrem y a $N(y)$.
   – Pridaj doň y, čím dostaneš najlepšie riešenie obsahujúce y.
3. Na výstup vráť najlepšie z riešení zostrojených v predchádzajúcom kroku.

Príklad. Nech z je zebra a nech naraz s ňou nemôže byť vo výbehu kôň, srnka ani antilopa. Potom v optimálnom riešení je aspoň jedno z týchto štyroch zvierat. Postupne teda pre každé z nich nájdeme rekurzívnym volaním najlepšie riešenie, ktoré ho obsahuje.

Nech má vybrané zvieratko k konfliktov. Z toho, ako sme zvolili zvieratko z v kroku 1, vyplýva, že **každé** zvieratko má aspoň k konfliktov. Potom tento algoritmus spraví $k+1$ rekurzívnych volaní, pričom každé z nich bude na nejaký nový problém s nanajvýš $n-(k+1)$ zvieratkami.
Dá sa ukázať, že najhorší prípad nastáva pre $k=2$, teda keď má každé zvieratko presne dva konflikty. Vtedy bude mať tento algoritmus časovú zložitosť $\hat{O}(3^{n/3})$, čo vieme upraviť do podoby $\hat{O}(1.4423^n)$.

Maximálna nezávislá množina: nájdenie všetkých optimálnych riešení

„Lepší algoritmus 2“, ktorý sme si práve popísali, vieme ľahko upraviť tak, aby nie len vypočítal najväčší počet zvierat vo výbehu, ale navyše aj postupne vygeneroval a vypísal všetky **optimálne** riešenia tejto úlohy. Z toho vyplýva, že optimálnych riešení nemôže byť viac ako $\hat{O}(3^{n/3})$. Je ich teda vždy výrazne menej ako 2^n.
Ľahko nahliadneme, že tento odhad je pomerne tesný. Stačí zobrať $n=3k$ zvieratiek, rozdeliť ich do trojíc a povedať, že v každej trojici sú každé dve zvieratá v konflikte. Potom bude každé optimálne riešenie obsahovať práve jedno zviera z každej trojice a teda bude presne $3^k = 3^{n/3}$ optimálnych riešení.

Problém obchodného cestujúceho

V krajine je n miest, očíslovaných od 1 po n. Pre každú dvojicu miest (i,j) poznáme cenu $c_{i,j}$ cestovného lístku z i do j. Obchodný cestujúci Emil potrebuje precestovať celú krajinu: chce začať v meste 1, postupne navštíviť **práve raz** každé iné mesto a nakoniec sa vrátiť späť do mesta 1. Koľko peňazí mu na to stačí?

Priamočiare riešenie tejto úlohy má časovú zložitosť ešte horšiu ako exponenciálnu. Emila zaujíma, v akom poradí má navštíviť mestá 2 až n, hľadá teda ich optimálnu *permutáciu*. Toto vieme spraviť tak, že postupne vygenerujeme všetkých $(n-1)!$ permutácií miest 2 až n a pre každú spočítame, koľko by nás stálo cestovné. Takéto riešenie má teda časovú zložitosť $\hat{O}(n!)$.

Problém obchodného cestujúceho: dynamické programovanie

Ukážeme si, ako túto úlohu vyriešiť s časovou zložitosťou $\hat{O}(2^n)$, presnejšie, v čase $O(n^2 2^n)$.
Všimnime si Emila niekedy počas jeho cesty po krajine. Už navštívil nejaké mestá a zaplatil nejaké peniaze za cestovné. Položme mu teraz otázku: „Za koľko najmenej peňazí vieš svoju cestu dokončiť?“
Od čoho závisí odpoveď na túto otázku? Len od dvoch vecí: od mesta a, kde sa Emil práve nachádza, a od množiny miest B, ktoré ešte nenavštívil. Označme $d_{a,B}$ odpoveď na otázku s týmito dvoma parametrami.
Ak je množina B prázdna, je otázku ľahké zodpovedať: $d_{a,\emptyset} = c_{a,1}$, lebo už sa len potrebujeme vrátiť z aktuálneho mesta a na začiatok. Vo všetkých ostatných prípadoch sa pozrime na to, čo Emil spraví v nasledujúcom kroku: vyberie si nejaké mesto $b \in B$ a odcestuje doň. Najlepšie riešenie pre konkrétne mesto b bude Emila stáť $c_{a,b} + d_{b,B-\{b\}}$ peňazí: najskôr zaplatí $c_{a,b}$ za cestu z a do b a potom $d_{b,B-\{b\}}$ za optimálne dokončenie riešenia zo situácie, kedy stojí v meste b a ešte potrebuje navštíviť ostatné mestá z množiny B.
Hodnotu $d_{a,B}$ pre $B \neq \emptyset$ teda spočítame tak, že postupne vyskúšame všetky $b \in B$, pre každé z nich zistíme, k ako najlepšiemu riešeniu vedie, a z takto získaných hodnôt zoberieme minimum.

Otázok, ktoré si kladieme, teda rôznych hodnôt $d_{a,B}$, ktoré nás zaujímajú, je $O(n2^n)$, lebo je n možností pre a a pri konkrétnom a nanajvýš 2^{n-1} možností pre B (lebo pre každé mesto iné od a máme dve možnosti: buď v B leží alebo nie). Každú otázku vieme zodpovedať v čase $O(n)$, a teda celková časová zložitosť výpočtu všetkých hodnôt $d_{a,B}$ je $O(n^2 2^n)$. Výsledným riešením je potom hodnota $d_{1,\{2,3,\ldots,n\}}$.

Pseudokód rekurzívnej implementácie:

```
funkcia d(a, B):
    ak si už niekedy spracúval vstup (a, B):
        vráť zapamätanú odpoveď

    ak je B prázdna:
        odpoveď = C[a, 1]
    inak:
        odpoveď = minimum { C[a, b] + d(b, B okrem b)  |  b je prvok B }
    zapamätaj si že pre vstup (a, B) je výstup odpoveď
    vráť odpoveď
```

Všimnite si, že pri každom rekurzívnom volaní sa zmenší množina ešte nenavštívených miest. Vďaka tomu každá vetva rekurzie eventuálne skončí. Pre každú z $O(n2^n)$ dvojíc (a, B) sa telo tejto funkcie (výpočet konkrétnej hodnoty $d_{a,B}$) vykoná nanajvýš raz, vďaka čomu dosiahneme sľúbenú celkovú časovú zložitosť $O(n^2 2^n)$.

Pseudokód jednej možnej iteratívnej implementácie:

```
pre každé a:
    D[a, prázdna množina] = C[a, 1]

pre každú veľkosť vb množiny B od 1 až po n-1:
    pre každú množinu B veľkosti vb:
        pre každé a nepatriace do množiny B:
            D[a, B] = nekonečno
            pre každé b z množiny B:
                D[a, B] = min( D[a, B], C[a, b] + D[b, B okrem b] )
```

Všimnite si, že pri tejto implementácii pri výpočte konkrétnej $d_{a,B}$ už poznáme všetky hodnoty $d_{b,B-\{b\}}$, ktoré potrebujeme, lebo sme ich vypočítali v skoršej iterácii vonkajšieho for-cyklu: množina $B-\{b\}$ má menšiu veľkosť ako množina B.

Na záver tohto študijného textu podotkneme, že pri praktickej implementácii tohto algoritmu by sme na uloženie množín B použili tzv. bitové masky (bitmasky): množinu $\{x_1, \ldots, x_i\}$ by sme reprezentovali číslom $2^{x_1} + \cdots + 2^{x_i}$, teda číslom, ktoré má nastavené práve bity s číslami $x_1, \ldots, x_i$.
Rozmyslite si, že ak má konkrétna množina priradené nejaké číslo, tak všetky jej podmnožiny majú priradené menšie čísla (lebo keď zmažeme z množiny prvok, tak v dvojkovom zápise čísla, ktoré ju reprezentuje, zmeníme príslušnú jedničku na nulu). Namiesto vonkajších dvoch for-cyklov by sme teda mohli použiť len jeden for-cyklus cez všetky čísla predstavujúce platné kódy množín, od najmenšieho po najväčšie. Takto dostaneme iné poradie, v ktorom budeme množiny B spracúvať, ale vyššie popísaný algoritmus bude stále korektne fungovať, lebo pre každé B a b bude aj teraz platiť, že množinu $B - \{b\}$ spracujeme skôr ako množinu B.

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 2019