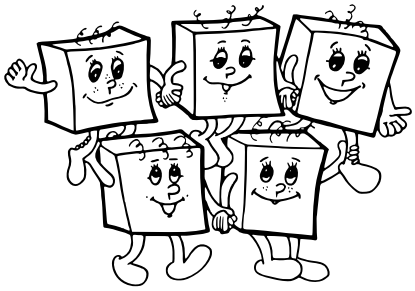
OLYMPIÁDA V INFORMATIKE

NA STREDNÝCH ŠKOLÁCH

http://oi.sk/



tridsiaty piaty ročník
školský rok 2019/2020

zadania celoštátneho kola, deň 1
kategória A

Priebeh celoštátneho kola

Celoštátne kolo 35. ročníka Olympiády v informatike, kategórie A, sa koná v dňoch 25.-28. marca 2020. Na riešenie úloh prvého, teoretického dňa majú súťažiaci 4,5 hodiny čistého času. Rôzne úlohy riešia súťažiaci na samostatné listy papiera. Akékoľvek pomôcky okrem písacích potrieb (napr. knihy, výpisy programov, kalkulačky) sú zakázané.

Čo má obsahovať riešenie úlohy?

- Slovne popíšte algoritmus.
  Slovný popis riešenia musí byť jasný a zrozumiteľný i bez nahliadnutia do samotného algoritmu/programu.
- Zdôvodnite správnosť vášho algoritmu.
- Uveďte a zdôvodnite jeho časovú a pamäťovú zložitosť.
- Podrobne uveďte dôležité časti algoritmu, ideálne vo forme programu v nejakom bežnom programovacom jazyku.
- V prípade, že používate vo svojom programovacom jazyku knižnice, ktoré obsahujú implementované dátové štruktúry a algoritmy (napr. STL pre C++), v popise algoritmu stručne vysvetlite, ako by ste napísali program s rovnakou časovou zložitosťou bez použitia knižnice.

Hodnotenie riešení prvého (teoretického) dňa

Za každú úlohu môžete získať od 0 do 10 bodov.
Pokiaľ nie je v zadaní povedané ináč, najdôležitejšie dve kritériá hodnotenia sú v prvom rade **správnosť** a v druhom rade **efektívnosť** navrhnutého algoritmu. Na výslednom počte bodov sa môže prejaviť aj kvalita popisu riešenia a zdôvodnenie tvrdení o jeho správnosti a efektívnosti.
Efektívnosť algoritmu posudzujeme vypočítaním jeho časovej zložitosti – funkcie, ktorá hovorí, ako dlho vykonanie algoritmu trvá v závislosti od veľkosti vstupných parametrov. Nezávisí pri tom na konštantných faktoroch, len na rádovej rýchlosti rastu tejto funkcie.
V zadaní úlohy môžu byť uvedené limity na veľkosť premenných. Tieto môžete použiť na odhad toho, ako dobré vaše riešenie je. Na počítači, ktorý vykoná miliardu inštrukcií za sekundu, vyrieši vzorové riešenie ľubovoľný povolený vstup nanajvýš za niekoľko sekúnd.

A-III-1 Oplotenie záhradiek

Obdĺžnikovú oblasť pôdy sa dediči pozemkového magnáta Pišišvora rozhodli rozdeliť plotmi na štvorcové záhradky so stranou dĺžky jeden meter a rozpredať. Problém je však v tom, že v tejto oblasti je stavanie plotov prísne regulované, a to nasledovne:

- Plot musí byť rovný.
- Plot musí byť rovnobežný s jednou zo strán pozemku a mať od nej celočíselnú vzdialenosť v metroch.
- Ploty musia byť stavané postupne jeden po druhom.
- Plot nesmie križovať iné, skôr postavené ploty.
- Za každý plot je potrebné zaplatiť registračný poplatok. Tento poplatok je jednoznačne určený súradnicou, na ktorej celý plot leží. Poplatok nezávisí od dĺžky plotu.
- Každý plot musí byť čo najdlhší – vo chvíli, keď je postavený, sa ho nesmie dať predĺžiť do žiadnej strany.

Obvod pozemku je už oplotený, v jeho vnútri sa zatiaľ žiadne ploty nenachádzajú.

Formát vstupu a výstupu

V prvom riadku vstupu sú celé čísla d, s udávajúce dĺžku a šírku obdĺžnikového pozemku.
V druhom riadku je $d-1$ celých čísel $z_1, \ldots, z_{d-1}$, kde z_i je poplatok za postavenie každého „zvislého“ plotu, teda plotu, ktorého každý bod je vzdialený i metrov od ľavého okraja pozemku.
V treťom riadku je $s-1$ celých čísel $v_1, \ldots, v_{s-1}$, udávajúcich ceny za „vodorovné“ ploty podľa ich vzdialenosti od horného okraja pozemku.

Vypočítajte a na výstup vypíšte minimálnu celkovú cenu, za ktorú vieme postaviť ploty tak, aby každé políčko veľkosti meter krát meter bolo zo všetkých strán oplotené.

Obmedzenia a hodnotenie

Všetky poplatky sú kladné. Cena ľubovoľného korektného oplotenia sa pohodlne zmestí do bežnej celočíselnej premennej.
Za ľubovoľné korektné riešenie môžete získať aspoň 3 body.
Za riešenie dostatočne efektívne pre $d, s \leq 50$ môžete získať 6 bodov.
Obmedzenia pre optimálne riešenie úmyselne neuvádzame. Upozorňujeme však, že u riešení, ktoré majú ambíciu získať viac ako 6 bodov, je veľmi dôležitou súčasťou dôkaz správnosti použitého algoritmu.

Príklad

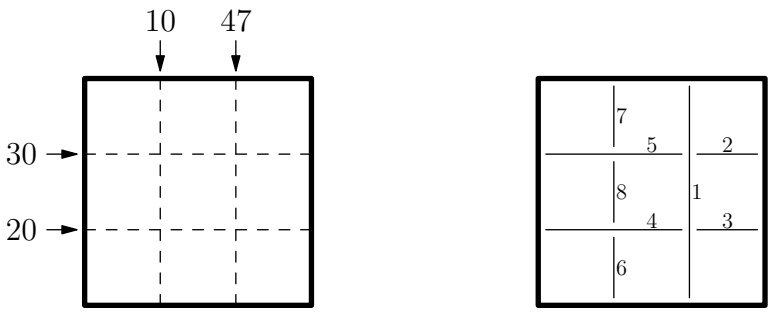
vstup

```
3 3
10 47
30 20
```

výstup

```
177
```

Na obrázku vľavo je znázornené zadanie. Čiarkované čiary predstavujú miesta, kde treba postaviť ploty. Čísla udávajú ceny za postavenie každého plota v dotyčnom riadku/stĺpci. Na obrázku vpravo je znázornené jedno možné optimálne riešenie. Čísla predstavujú poradie stavania plotov.



A-III-2 Horolezci

Horolezecký klub chystá n-dňovú expedíciu. V každý deň expedície sa jeden z jej členov pokúsi dosiahnuť jeden z okolitých vrcholov. Vrchol plánovaný na i-ty deň je vysoký v_i metrov.
Každý horolezec, ktorý pôjde dosiahnuť aspoň jeden vrchol, potrebuje špeciálny výstroj. Cena takéhoto výstroja je priamo úmerná maximálnej výške, po ktorú je certifikovaný. (Pre jednoduchosť budeme predpokladať, že výstroj certifikovaný po výšku x stojí x eur.)
Horolezec môže počas expedície svoj výstroj použiť ľubovoľne veľakrát. Kvôli bezpečnosti členov expedície ale platia nasledovné obmedzenia:

- Horolezec nesmie ísť do výšky, na ktorú nemá certifikovaný výstroj.
- Každý horolezec musí používať svoj vlastný výstroj, nesmú si ich požičiavať.
- Po každom pokuse o vrchol musí horolezec aspoň deň oddychovať.

Dané sú výšky v_i. Navrhnite algoritmus, ktorý vypočíta minimálnu celkovú cenu potrebného vybavenia pre expedíciu. (Hľadáme teda, koľko členov má mať expedícia, aký výstroj ktorému z nich kúpiť a v ktorý deň koho z nich poslať dosiahnuť vrchol, pričom toto celé chceme spraviť tak, aby celková cena potrebných výstrojov bola čo najmenšia.)

Formát vstupu a výstupu

V prvom riadku vstupu je kladné celé číslo n. V druhom riadku vstupu sú kladné celé čísla $v_1, \ldots, v_n$.
Na výstup vypíšte minimálnu celkovú cenu horolezeckých výstrojov s ktorými sa expedícia dá zvládnuť.

Obmedzenia a hodnotenie

Súčet všetkých výšok sa pohodlne zmestí do bežnej celočíselnej premennej.
Za ľubovoľné korektné riešenie sa dajú získať aspoň 3 body. Za riešenie efektívne pre $n \leq 100$ môžete získať 6 bodov, za riešenie efektívne pre $n \leq 1000$ môžete získať 8 bodov, a za riešenie efektívne pre $n \leq 30\,000$ plných 10 bodov.

Príklady

vstup

```
5
8485 8516 8586 8611 8848
```

výstup

```
17459
```

Najvyšších päť osemtisícoviek. Optimálne riešenie je mať horolezca A s výstrojom certifikovaným po 8848 metrov a horolezca B s výstrojom po 8611 metrov. Pokusy o dosiahnutie vrcholu budú robiť v poradí ABABA.

vstup

```
7
1200 7500 1100 1500 1400 1100 7300
```

výstup

```
10000
```

Jedno možné optimálne riešenie: Horolezci A (výstroj po 1100), B (výstroj po 7500) a C (výstroj po 1400) pôjdu liezť v poradí CBABCAB.

A-III-3 Exaktné exponenciálne algoritmy

Táto súťažná úloha nadväzuje na úlohy z domáceho a krajského kola. Za samotným zadaním úlohy nájdete študijný text. Ten je totožný so študijným textom zo zadaní domáceho kola. Podúlohy sú nezávislé, môžete ich riešiť v ľubovoľnom poradí.

Podúloha A (5 bodov).
Pripomeňme si, že v zoologickej záhrade zo študijného textu majú n zvierat, pričom existuje m dvojíc zvierat, ktoré nemôžu byť spolu vo výbehu. Pre dané n, m a zoznam dvojíc zvierat by sme teraz chceli zistiť, či sa dá všetky tieto zvieratá rozmiestniť do troch výbehov. Navrhnite čo najefektívnejší algoritmus riešiaci túto úlohu.

Ak má vaše riešenie časovú zložitosť $\hat{\Theta}(b^n)$, tak môže získať nanajvýš 5 bodov ak $b < 2$, nanajvýš 3 body ak $b = 2$ a nanajvýš 1 bod ak $b > 2$.

Podúloha B (5 bodov).
Vstup je rovnaký ako v podúlohe A. Navrhnite algoritmus, ktorý vypočíta, koľko najmenej výbehov potrebujeme, ak chceme každé zviera umiestniť do nejakého výbehu.

Na zisk 5 bodov treba časovú zložitosť $\hat{\Theta}(b^n)$, kde $b < 3$. Ľubovoľné riešenie s exponenciálnou časovou zložitosťou môže dostať aspoň 3 body. Riešenia s časovou zložitosťou horšou ako exponenciálnou od n môžu získať nanajvýš 1 bod.

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 2020