PRASK
II. ročník, 2015/16
Katedra základov a vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

Vzorové riešenia 2. kola zimnej časti

vzorák napísal Maja, Žaba a Baklažán
(max. 15 b za riešenie)

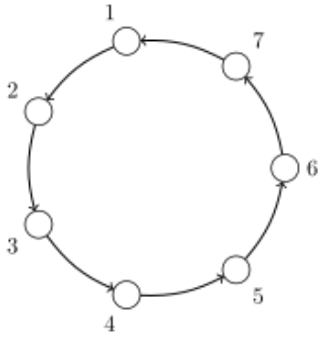
1. Preblšený cirkus

Táto úloha vyzerá ako úloha o kreslení obrázkov, ale nakoniec sa ukáže, že je to celé o niečom inom...

Podúloha a)

Najjednoduchšie riešenie je kruh so 7 blchami ako na obrázku.

Je zjavné, že každá blcha musí prejsť celý kruh, kým sa vráti na svoje pôvodné miesto a bude jej to trvať presne 7 krokov. Situácia je pre každú blšku úplne rovnaká, takže po siedmych krokoch budú všetky blchy súčasne na svojich pôvodných miestach.

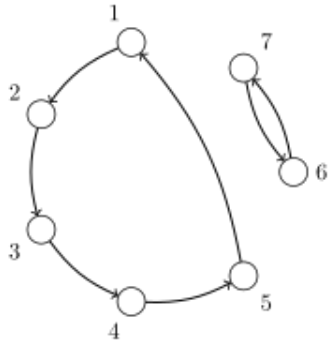


Takýto obrázok budeme volať *cyklus dĺžky* 7. V ďalších podúlohách budeme používať cykly rôznych dĺžok.

Podúloha b)

Po chvíľke kreslenia môžeme prísť na to, že funguje choreografia pozostávajúca z dvoch cyklov, jedného dĺžky 5 a druhého dĺžky 2. Ale prečo je to tak? Blšky, ktoré sú súčasťou cyklu dĺžky 2 sa na svoje miesto vrátia po $2, 4, 6, 8, 10\ldots$ krokoch (zakaždým musia prejsť celým kruhom dĺžky 2). Je jasné, že sú to všetko násobky dvojky.

Takéto tvrdenie si môžeme zovšeobecniť. Blcha v cykle dĺžky n sa na svoje miesto vráti každých n krokov, teda vždy, keď je počet krokov od začiatku násobok čísla n. Blška, ktorá je súčasťou cyklu dĺžky 5 sa teda na svoje miesto vráti po $5, 10, 15\ldots$ krokoch. Tým pádom sa blšky v oboch cykloch prvýkrát naraz vrátia na svoje miesta po 10 krokoch.



Podúloha c)

V tejto podúlohde hľadáme opakovaciu choreografiu, ktorá obsahuje 10 blšiek. Bolo by preto na mieste sa zamyslieť, ako môžu vyzerať opakovacie choreografie. Z každého krúžka, ktorý predstavuje začiatočnú pozíciu

nejakej blšky, vychádza jedna šípka do iného (alebo aj toho istého) krúžka. To znamená, že ak máme n krúžkov, bude medzi nimi viesť aj n šípiek. Uvedomme si teraz, že do každého krúžku **musí viesť práve jedna šípka**.

Ak by do nejakého krúžku neviedla žiadna šípka, blška, ktorá začína v tomto krúžku sa sem nikdy nevie vrátiť, lebo nemá ako. Takáto choreografia by teda nebola opakovacia. A ak máme n krúžkov a n šípiek a do každého krúžku musí viesť aspoň jedna šípka, neostáva nám iná možnosť, ako že do každého krúžku vedie práve jedna šípka. A naozaj, jediné obrázky, ktoré vieme nakresliť dodržiac toto pravidlo sú cykly rôznych dĺžok. Každá opakovacia choreografia sa teda skladá z niekoľkých (možno rôzne dlhých) cyklov.

V podúlohe b) sme videli, že choreografia z dvoch cyklov dĺžok 2 a 5 sa prvýkrát zopakuje po 10 krokoch. To je totiž najmenšie číslo, ktoré je zároveň násobkom čísla 2 aj čísla 5. Inými slovami, 10 je najmenším spoločným násobkom týchto dvoch čísel. Uvedomme si, že toto platí aj všeobecne. Ak budeme mať niekoľko cyklov, tak sa všetky blšky prvýkrát naraz vrátia na svoje pôvodné miesto po najmenšom spoločnom násobku ich dĺžok, lebo to je prvé číslo, ktoré je násobkom dĺžok všetkých cyklov.

Ak teda hľadáme najdlhšiu opakovaciu choreografiu pre 10 krúžkov, musíme tieto krúžky rozdeliť na niekoľko cyklov, pričom súčet ich dĺžok bude 10. Pritom chceme maximalizovať najmenší spoločný násobok týchto čísel. Nebude trvať dlho a zistíme, že najlepšie riešenie je spraviť choreografiu skladajúcu sa z troch cyklov dĺžok 2, 3 a 5 a táto choreografia sa prvýkrát zopakuje po 30-tich krokoch.

V ďalších podúlohách budeme využívať algoritmus hľadania najmenšieho spoločného násobku čísel pomocou ich prvočíselných rozkladov. Ak ste sa s týmto algoritmom (prípadne ani s rozkladmi na prvočísla) ešte nestretli, odporúčame vám najprv si o tom niečo naštudovať z iných zdrojov.

Podúloha d)

V tejto podúlohe bolo treba nájsť čo najmenšiu opakovaciu choreografiu, ktorá sa zopakuje po 3 960 skokoch. Z predchádzajúcej podúlohy vieme, že opakovacia choreografia sa musí skaldať z niekoľkých cyklov. A tiež vieme, že najmenší spoločný násobok dĺžok týchto cyklov musí byť nami požadované číslo 3 960.

Samozrejme, môžeme spraviť jeden cyklus dĺžky 3 960. Použiť ale takmer štyritisíc krúžkov asi nebude najlepšie.

Prvočíselný rozklad čísla 3 960 je $3\,960 = 2^3 \cdot 3^2 \cdot 5 \cdot 11$. Keďže 3 960 má byť najmenší spoločný násobok dĺžok cyklov, všetky tieto prvočísla musia byť niekde v prvočíselných rozkladoch týchto dĺžok. Niektorá z dĺžok teda musí byť deliteľná jedenástimi, niektorá (možno tá istá a možno iná) musí byť deliteľná piatimi. Ďalej musí byť niektorá z dĺžok cyklov deliteľná 3^2 – nestačí, aby sme mali dve dĺžky deliteľné 3, lebo jedna z týchto troják by sa pri hľadaní najmenšieho spoločného násobku škrtla. Podobne musí byť niektorá z dĺžok deliteľná 2^3.

Najmenší súčet dĺžok cyklov (a teda aj najmenší počet krúžkov v našej choreografii) za splnenia týchto podmienok dostaneme, keď zvolíme štyri cykly s dĺžkami 5, 8, 9 a 11 krúžkov. Ľahko overíme, že najmenší spoločný násobok týchto štyroch čísel je naozaj 3 960, teda choreografia sa naozaj prvýkrát začne opakovať po 3 960 zapískaniach. Spolu sme pri tom použili iba 33 krúžkov.

Podúloha e)

Výsledok v podúlohe d) je pomerne malý. Je však najmenší možný? Ako vieme robiť niečo takéto pre všeobecné n? Budeme postupovať podobne, ale všeobecne. Nech prvočíselný rozklad čísla n je

$$n = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k},$$

kde $p_1, p_2, \ldots, p_k$ sú navzájom rôzne prvočísla, $\alpha_1, \alpha_2, \ldots, \alpha_k$ sú prirodzené čísla a k je počet rôznych prvočísel v rozklade. Nenechajte sa zmiasť tým, ako zápis s troma bodkami vyzerá a nemyslite si, že k musí byť najmenej 4. Pokojne to môže byť aj menej, napríklad 1 (vtedy by rozklad mal tvar $n = p_1^{\alpha_1}$). Dôležité je, že každé prirodzené číslo n väčšie ako 1 sa dá zapísať takýmto spôsobom. Napríklad pre číslo 3 960 by sme mali $k = 4$, $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, $p_4 = 11$, $\alpha_1 = 3$, $\alpha_2 = 2$, $\alpha_3 = 1$ a $\alpha_4 = 1$.

Predstavme si, že za nami prišla víla Amálka a dala nám hotové najlepšie riešenie našej úlohy, teda opakovaciu choreografiu, ktorá sa prvý krát zopakuje po n zapískaniach a má najmenší možný počet krúžkov. Aké vlastnosti musí táto choreografia mať?

Keďže ide o opakovaciu choreografiu, určite bude pozostávať z niekoľkých cyklov. Navyše vieme, že najmenší spoločný násobok ich dĺžok musí byť n.

Pozrime sa teraz na prvočíselný rozklad dĺžky d jedného cyklu. Mohlo by v ňom byť viac ako jedno prvočíslo? Povedzme, že áno. Potom by sme túto dĺžku vedeli zapísať ako $d = q_1^{\beta_1} \cdot q_2^{\beta_2} \cdot \ldots \cdot q_x^{\beta_x}$, kde $q_1, q_2, \ldots, q_x$ sú navzájom rôzne prvočísla, ich počet x je aspoň 2 a $\beta_1, \beta_2, \ldots, \beta_x$ sú prirodzené čísla. Ak by sme náš cyklus nahradili dvoma cyklami s dĺžkami $a = q_1^{\beta_1}$ a $b = q_2^{\beta_2} \cdot \ldots \cdot q_x^{\beta_x}$, najmenší spoločný násobok dĺžok všetkých cyklov by sa nezmenil – v rozkladoch dĺžok by boli stále tie isté prvočísla v tých istých mocninách a škrtli by sa nám tie isté prvočísla, čo predtým. Zároveň by sme však takouto výmenou znížili počet krúžkov v choreografii,

keďže platí $a + b < a \cdot b = d$[1]. Keďže ale vieme, že Amálka nám už dala najmenšiu možnú choreografiu, tento prípad nemohol nastať. To znamená, že dĺžka nášho cyklu musí mať v rozklade iba jedno prvočíslo (ktoré ale môže byť v ľubovoľne vysokej mocnine). Toto musí platiť pre každý cyklus.

Keďže n musí byť najmenším spoločným násobkom dĺžok našich cyklov, dĺžka niektorého z cyklov musí mať vo svojom prvočíselnom rozklade $p_1^{\alpha_1}$. Podobne musí existovať cyklus, ktorý má v rozklade $p_2^{\alpha_2}$, atď. až po $p_k^{\alpha_k}$. Všetky tieto cykly musia byť rôzne, nakoľko dĺžka žiadneho cyklu v Amálkinej choreografii nemôže mať v prvočíselnom rozklade dve rôzne prvočísla. V Amálkinom riešení teda určite bude aspoň k cyklov, ktorých dĺžky budú najmenej $p_1^{\alpha_1}, p_2^{\alpha_2}, \ldots, p_k^{\alpha_k}$. Zároveň platí, že v riešení už nič ďalšie netreba (lebo najmenší spoločný násobok týchto čísel je n), teda optimálne riešenie bude pozostávať s k cyklov s dĺžkami $p_1^{\alpha_1}, p_2^{\alpha_2}, \ldots, p_k^{\alpha_k}$.

Podúloha f)

Tu existuje viacero ľahkých postupov, ako vytvoriť takúto neurčitú choreografiu. Napríklad si môžeme zobrať ľubovoľnú opakovaciu a ľubovoľnú stretávaciu choreografiu s aspoň dvoma krúžkami, nakresliť ich do jedného obrázka a krúžky očíslovať tak, aby žiadne dva nemali rovnaké číslo.

Alebo môžeme zobrať ľubovoľne dlhý cyklus, napríklad cyklus dĺžky tri a pridať ešte jeden krúžok, do ktorého nevedie žiadna šípka a vychádza z neho šípka do nejakého krúžku na cykle. Blška začínajúca v krúžku mimo cyklu sa nebude vedieť vrátiť späť (takže choreografia nie je opakovacia) a blšky na cykle sa nikdy nestretnú (takže nie je ani stretávacia).

Podúloha g)

Veľmi ľahko nájdeme choreografiu, v ktorej bude trvať 46 krokov, kým sa všetky blšky stretnú na jednom mieste. Nakreslíme cyklus dĺžky 47, vyberieme si jeden krúžok a zmažeme šípku, ktorá z neho vedie. Následne mu dokreslíme slučku, teda šípku, ktorá pôjde z neho späť doňho. V podstate takto vytvoríme dlhú cestu, kde sa blšky budú stretať práve v tomto pozmenenom krúžku. A blška začínajúca na krúžku, do ktorého nevedie žiadna šípka, bude musieť skočiť 46 krát, aby sa stretla s ostatnými.

Týmto však náš dôkaz nekončí. Zatiaľ sme ukázali len to, že Marienka musí povedať **aspoň** číslo 46. Ešte musíme ukázať, že neexistuje stretávacia choreografia so 47 krúžkami, v ktorej sa blšky prvýkrát stretnú po viac ako 46 skokoch.

Uvedomme si nasledovnú vlastnosť. Ak sa blška počas vystúpenia druhýkrát ocitne v krúžku, v ktorom už niekedy bola, určite je na cykle, po ktorom bude skákať do nekonečna. Keď totiž bola v tomto krúžku prvýkrát, pokračovala po šípkach ďalej a zaviedli ju naspäť na tento krúžok. Tieto šípky sú jednoznačné, blška sa nemôže rozhodovať o tom, kam chce skočiť. Preto ju zavedú naspäť na tento krúžok aj tretí, štvrtý, ... krát.

Po 47 skokoch už každá blška bola v 48 krúžkoch (pretože v jednom krúžku začínala). Keďže krúžkov je iba 47, určite už musela v nejakom krúžku byť aspoň dvakrát. To znamená, že po 47 skokoch bude už každá blška v nejakom cykle. Ba čo viac, každá blška už tento svoj cyklus stihla aj celý prejsť (keďže v niektorom z krúžkov už bola dvakrát). To znamená, že každá blška musela byť v nejakom cykle už skôr – teda už aj po 46 skokoch.

Ak sú dve blšky v nejakom momente v rôznych krúžkoch jedného cyklu, budú po tomto cykle krúžiť dookola s nemeniacim sa odstupom a nikdy sa nestretnú. Tiež sa určite nikdy nestretnú, ak budú v rôznych cykloch. Keďže po 46 skokoch je každá blška v nejakom cykle a navyše vieme, že niekedy sa určite stretnú, jediná možnosť je, že sú všetky v tom istom krúžku, teda už sa stretli. To znamená, že v ľubovoľnej stretávacej 47-krúžkovej choreografii sa všetky blšky stretnú prvýkrát najneskôr po 46 skokoch.

Podúloha h)

Predstavme si stretávacie miesto choreografie, ktorá neobsahuje slučku. Pre každú blšku si vieme zrátať, po koľkých krokoch sa prvýkrat dostane na toto miesto. Musí existovať blška, ktorá sa na stretávacie miesto dostane po 1 kroku, nazvime si ju Zuzka. Zároveň musí byť nejaká blška, ktorá začína v stretávacom mieste, nazvime si ju Lucka. Kedže neexistujú žiadne slučky, v každom kroku sa Lucka musí posunúť na iné miesto. A kedže Zuzka sa po prvom kroku ocitne na Luckinom mieste, v druhom kroku sa Zuzka pohne tam, kam sa pohla Lucka v prvom kroku atď. Zuzka sa vždy posunie na to miesto, z ktorého Lucka práve odišla. Zuzka a Lucka sa teda nikdy neocitnú spolu v jendom bode, ale budú sa donekonečne naháňať. To znamená, že choreografia bez slučky nemôže byť stretávacia.

[1] Nerovnosť $a + b \leq a \cdot b$ platí pre ľubovoľné $a, b \geq 2$. V našom prípade vieme, že a aj b sú aspoň 2, pretože a je aspoň q_1 (a q_1 je prvočíslo) a b je aspoň q_2. Navyše, keďže q_1 a q_2 sú navzájom rôzne prvočísla, aspoň jedno z čísel a, b musí byť väčšie ako 2 a preto platí aj nerovnosť $a + b < a \cdot b$

2. Praktické regulárne výrazy 2

vzorák napísal Žaba
(max. 15 b za riešenie)

Podúloha a)

Prvou podúlohou bolo napísať regulárny výraz, ktorý akceptuje všetky reťazce predstavujúce čísla deliteľné 5. Vieme, že číslom 5 sú deliteľné tie čísla, ktoré majú na konci cifru „0" alebo „5". Podľa zadania si však máme dať pozor na to, aby nami akceptované čísla nemali na začiatku zbytočné nuly. Môžeme napísať regulárny výraz, ktorý bude mať na prvom mieste nenulovú cifru, za ktorou bude nasledovať niekoľko ľubovoľných cifier a následne cifra 0 alebo 5. Tento výraz vyzerá takto: „`[1-9][0-9]*[05]`".

Samozrejme, takýto výraz neakceptuje dve výnimky – číslo 0 a číslo 5. Tieto pridáme pomocou „|", takže výsledný regulárny výraz má tvar „`([1-9][0-9]*[05])|0|5`".

Delenie číslom 5 bolo ľahké. Museli sme sa trochu pohrať s nulami na začiatku, ale inak nám stačilo splniť jedinú podmienku – 0 alebo 5 na konci čísla. Aj pre čísla deliteľné 3 platí jednoduchá podmienka: ciferný súčet týchto čísel musí byť deliteľné troma. Takáto podmienka sa však nedá zapísať pomocou regulárnych výrazov takým jednoduchým spôsobom. Nasledujúce podúlohy preto slúžili na to, aby sme postupne mohli vytvárať finálne riešenie.

Podúloha b)

V tejto podúlohe bolo treba napísať regulárny výraz, ktorý obsahuje iba cifry 1 a 2 predstavujúce chodenie po schodoch. Má pritom platiť, že tento regulárny výraz popisuje všetky cesty na schod vyšší ako šiesty, ktoré nestúpia na schod 3 ani na schod 6.

Keď zlodej stojí na schode číslo 0 (teda na spodku schodišťa), má dve možnosti. Buď spraví krok dĺžky 1 alebo dĺžky 2. Pozrime sa, čo sa stane ak najskôr spraví krok dĺžky 1. Týmto krokom sa dostane na schod 1, z ktorého ale nemôže spraviť krok dlhý 2, lebo by sa dostal na zalepidlovaný schod 3. Musí teda spraviť krok dĺžky 1, čím sa ocitne na schode číslo 2. Následne musí prekročiť schod 3 pomocou dlhšieho kroku dĺžky 2, čím sa dostane na schod 4. Odtiaľ má opäť iba jednu možnosť, krok dĺžky 1, lebo dlhším krokom by sa ocitol na schode 6. No a z piateho schodu musí spraviť krok dĺžky 2, aby prekročil zalepidlovaný schod. V tomto momente už pred ním nie sú žiadne zalepené schody a môže pokračovať ľubovoľne. Tento postup vieme zapísať ako regulárny výraz „`11212[12]*`".

Čo ak sa však na začiatku rozhodne spraviť krok dĺžky 2? Ocitne sa na schode 2, čo znamená, že opäť musí spraviť krok dĺžky 2, aby prekročil schod 3. Tak ako predtým, keď sa ocitol na schode 4, musí spraviť krok dĺžky 1 a potom dĺžky 2. Tým sa dostane na schod 7, odkiaľ je to opäť bezpečné. Tento postup popisuje regulárny výraz „`2212[12]*`".

Vidíme, že kým sa zlodej dostal na siedmy schod, rozhodoval sa iba na úplnom začiatku a zvyšok cesty bol presne určený. Výsledný regulárny výraz teda dostaneme spojením týchto dvoch regulárnych výrazov, čo vieme urobiť napríklad tak, že ich oddelíme „|". Ak sa však trochu zamyslíme, nájdeme aj kratší spôsob: „`(11|2)212[12]*`".

Podúloha c)

Všimnime si, že v podúlohe b) sme mali na výber len na úplnom začiatku, keď sme boli na schode číslo 0 a potom až keď sme sa dostali na schod 7, kde už neboli žiadne schody potreté lepidlom. Uvedomme si, že schody môžeme rozdeliť na tri skupiny podľa toho, či ich číslo dáva po delení 3 zvyšok 0, 1 alebo 2.

Takisto si všimnime, že ak stojíme na schode, ktorého zvyšok po delení 3 je 1, tak nemôžeme spraviť krok dĺžky 2, lebo by sme sa dostali na schod pokrytý lepidlom. Musíme teda spraviť krok dĺžky 1, čím sa dostaneme na schod, ktorého zvyšok po delení 3 je 2. Ak stojíme na schode so zvyškom 2, musíme spraviť krok dĺžky 2, ktorým sa dostaneme na schod so zvyškom 1.

Ak teda ako prvý spravíme krok dlžky 1, následne budeme musieť striedavo robiť kroky dlžky 1 a 2, lebo iba tak nikdy nestúpime na schod deliteľný číslom 3. Ak ako prvý spravíme krok dlžky 2, budeme musieť dookola opakovať postupnosť krokov 2 a 1. Toto vieme ľahko zapísať pomocou regulárneho výrazu ako „`1(12)*1?|2(21)*2?`". Na koniec každej časti sme museli pridať ešte jeden znak s otáznikom, aby reťazce začínajúce na 1 mohli končiť 1 a reťazce začínajúce 2 mohli končiť 2. Bez nich by sme napríklad neakceptovali reťazec „`11`" alebo „`2212`", ktoré sú oba správne.

Otázkou je, či by náš regulárny výraz mal akceptovať aj prázdny reťazec. Zo zadania to nie je úplne jasné[2]. Môžeme sa teda dohodnúť, že prázdny reťazec za cestu považovať nebudeme. Ak by sme ho tam veľmi chceli mať, ľahko ho môžeme pridať jedným or-om.

[2] Ak sa zlodej nepohne, určite sa neprilepí – ale dá sa to považovať za cestu hore schodmi?

Podúloha d)

V tejto podúlohe máme napísať regulárny výraz, ktorý zodpovedá tomu, že zlodeji nejakú dobu kráčali po schodoch a potom stúpili na schod deliteľný 3, prilepili sa a ostali stáť. Všimnime si, že táto podúloha silno súvisí s podúlohou c). Keďže sa zlodeji nemohli prilepiť počas cesty, museli sa nejakú dobu vyhýbať schodom s lepidlom až kým to potom na konci nepokazili. A časť ich cesty, kým sa polepeným schodom úspešne vyhýbali, predsa popisuje regulárny výraz z podúlohy c).

Ostáva zistiť, čo musí zlodej spraviť, aby stúpil na schod, ktorého číslo je deliteľné 3. Ak stojí na schode, ktorého zvyšok po delení 3 je 1, tak ak spraví kroky „11" alebo „2", skončí na schode s lepidlom. Ak schod, na ktorom stojí má zvyšok 2, tak sa zalepí ak spraví krok „1". Predstavme si, že zlodej išiel najskôr podľa regulárneho výrazu „1(12)*". Ocitol sa teda na schode so zvyškom 1 a preto musel spraviť buď postupnosť „11" alebo „2". Ak išiel najskôr podľa „2(21)*" tak sa ocitol na schode so zvyškom 2 a teda musel spraviť krok „1".

V podúlohe c) sme však museli pridať aj reťazce „1?" a „2?" aby sme ošetrili všetky možnosti. A tu ich tiež istým spôsobom potrebujeme. V prvom prípade to nemusíme riešiť, lebo ak zlodej po ceste typu „1(12)*" spraví ešte na konci krok 1, tak potom musí spraviť opäť krok 1 aby sa prilepil, čo sme pokryli v možnosti „1(12)*11". V druhom prípade však neriešime situáciu, ak zlodej po ceste typu „2(21)*" urobil krok 2 a potom sa chce prilepiť. Okrem možnosti „1" teda musíme za „2(21)*" pridať aj možnosť „22", ktorá zodpovedá presne tomuto prípadu. Spojením tohto všetkého dostaneme regulárny výraz „1(12)*(11|2)|2(21)*(1|22)".

Podúloha e)

Asi najľahšia podúloha, ak sme vyriešili podúlohu d). Ak sa zlodej prilepí, musí sa mu to stať na schode deliteľnom troma. Ak si teda vyzuje ponožky a pokračuje ďalej, je to to isté, ako keby začínal na schode 0 (ktorý je tiež deliteľný 3). Takže celá jeho cesta je len viacero ciest, pri ktorých sa vyhýba schodom deliteľným 3, až kým sa nenalepí a potom začína akoby odznova. Toto vieme docieliť pridaním „*" na koniec výrazu z podúlohy d), teda „(1(12)*(11|2)|2(21)*(1|22))*". Takto sme povolili aj prázdny reťazec. Ak by sme ho chceli zakázať, môžeme riešenie podúlohy d) zopakovať dvakrát a „*" dať iba za druhé opakovanie.

Podúloha f)

Ak zlodej zostane chvíľu stáť na mieste, nič sa na jeho situácii nezmení. Ak by sme odstránili nuly zo vstupného reťazca, musíme dostať reťazec, ktorý je vhodný pre podúlohu e). To znamená, že chceme robiť to isté, čo v podúlohe e) akurát hocikde môžeme pridať ľubovoľne veľa núl. Pridanie ľubovoľného množstva núl nám zabezpečí reťazec „0*" a jediné čo potrebujeme spraviť, je vložiť ho na všetky možné miesta reťazca z podúlohy e).

Je tu však ešte jeden háčik. Až po podúlohu d) sme státie na schode 0 (teda prázdny reťazec) netolerovali, teraz by sme však chceli akceptovať aj reťazce zložené iba zo samých núl. Preto na koniec nášho výrazu pridáme ešte „|0*".

Výsledok vyzerá trochu neprehľadne, ale uvedomte si, že je to naozaj len riešenie z podúlohy e) s množstvom núl, ktoré nič nerobia (a pridaným špeciálnym prípadom na konci). Možno by sme vedeli nejaké nuly ušetriť, keby sme náš regulárny výraz poriadne rozanalyzovali, jednoduchšie a bezpečnejšie však bude vložiť „0*" medzi každé dva znaky.

„(0*10*(0*10*20*)*(0*10*10*|0*20*)|0*20*(0*20*10*)*(0*10*|0*20*20*))*|0*"

Podúloha g)

Táto podúloha vyzerá na prvý pohľad najodstrašujúcejšie. Bolo ťažké poradiť si s ciframi 0, 1 a 2 a zrazu nám pribudne ďalších sedem. Budeme musieť urobiť tú istú úvahu celú od začiatku, akurát s toľkými rôznymi možnosťami? Našťastie nie.

Pozrime sa napríklad na cifru 3. Ak stojíme na zalepenom schode a posunieme sa o tri schody vyššie, opäť sa ocitneme na schode pokrytom lepidlom. Čo je v podstate to isté, ako keby sme zostali stáť na mieste. Áno, posunuli sme sa o tri schody vyššie, ale nás väčšinou nezaujíma ako vysoko sme, len to, kde pred nami sú ďalšie zalepené schody, poprípade to, aký zvyšok po delení 3 dáva schod na ktorom stojíme. A toto sa pri cifre 3 nemení.

Skúsme na to hľadieť nie cez schody, ale cez čísla. Hľadali sme čísla, ktorých ciferný súčet je deliteľný troma. Pridanie ľubovoľného množstva cifier 3 nám síce zmení ciferný súčet, ale nie jeho deliteľnosť číslom 3. Cifra 3 sa preto správa úplne rovnako ako cifra 0. Môžeme ju dať kam len chceme a koľkokrát len chceme. A toto isté platí aj pre cifry 6 a 9, ktoré sú tiež deliteľné troma.

Teraz snáď už začínate tušiť, ako to bude pokračovať. Ak si zoberieme cifru 4, použitie tejto cifry má na zvyšok po delení 3 rovnaký efekt, ako použitie cifry 1. A takisto cifra 7 má rovnaký efekt. To znamená, že tieto

cifry sú navzájom zameniteľné. Všade, kde sme v pôvodnom regulárnom výraze použili cifru 1, môžeme teraz použiť cifry 4 alebo 7 bez toho, aby sa čokoľvek pokazilo. No a samozrejme, cifry 5 a 8 sú rovnaké ako cifra 2.

Dostať regulárny výraz, pre podúlohu g) je potom vskutku jednododuché. Zoberieme si výraz z podúlohy f) a každý výskyt „0" nahradíme „[0369]", každý výskyt „1" nahradíme „[147]" a každý výskt „2" nahradíme „[258]". To boli totiž skupiny rovnako sa správajúcich cifier, pri ktorých je jedno, ktorú z nich použijeme.

Výsledný reťazec je už taký dlhý, že sme ho museli rozdeliť do viacerých riadkov. Opäť to však nie je nič strašné, ak si uvedomíme, akým spôsobom vznikol.

```
"([0369]*[147][0369]*([0369]*[147][0369]*[258][0369]*)*
([0369]*[147][0369]*[147][0369]*|[0369]*[258][0369]*)|
[0369]*[258][0369]*([0369]*[258][0369]*[147][0369]*)*
([0369]*[147][0369]*|[0369]*[258][0369]*[258][0369]*))*|
[0369]*"
```

Podúloha h)

Zostáva už len odstrániť prebytočné nuly na začiatku. Našťastie, toto sa dá urobiť pomerne jednoducho. Najprv úplne zakážeme, aby na začiatku čísla bola ktorákoľvek z cifier 0, 3, 6, 9. To urobíme tak, že zmažeme „|[0369]*" z konca výrazu a v oboch možnostiach veľkého or-u zmažeme zo začiatku „[0369]*".

Tu by mohol vzniknúť problém, keďže veľký or sa v reťazci môže opakovať a my chceme zmazať „[0369]*" iba zo začiatku prvého opakovania. Našťastie pre nás, aj na konci každej možnosti nášho veľkého or-u máme „[0369]*". O tom, čo je na začiatku druhého opakovania, teda môžeme prehlásiť, že je to v skutočnosti ešte na konci prvého opakovania. To znamená, že náš upravený výraz naozaj robí to, čo pôvodné riešenie podúlohy g), akurát nedovoľuje, aby číslo začínalo niektorou z cifier 0, 3, 6, 9.

Teraz naspäť povolíme, aby reťazec začínal niekoľkými (možno žiadnymi) ciframi 0, 3, 6 alebo 9, pričom ale prvá nemôže byť nula. Tomu zodpovedá regulárny výraz „[369][0369]*|", ktorý (v zátvorke) stačí napísať pred zvyšok nášho výrazu. Nakoniec ešte musíme pridať jeden špeciálny prípad – samotné číslo 0.

```
"([369][0369]*|)
([147][0369]*([0369]*[147][0369]*[258][0369]*)*
([0369]*[147][0369]*[147][0369]*|[0369]*[258][0369]*)|
[258][0369]*([0369]*[258][0369]*[147][0369]*)*
([0369]*[147][0369]*|[0369]*[258][0369]*[258][0369]*))*|0"
```

vzorák napísal Andrej a Jano
(max. 15 b za riešenie)

3. Príhody krokodíla Karola s tabuľkami

Vypracované Vzorové riešenie v Google Sheets môžete nájsť na adrese
docs.google.com/spreadsheets/d/1W9_NBYzKE8w6IZ-3EZLGSYTpUNPWz0m0ZGOKI7zRtkA Klikaním na bunky sa dozviete použité vzorce. V tomto texte si riešenia slovne vysvetlíme, prípadne uvedieme iné spôsoby riešenia.

a) Veľká násobilka

Môžeme vidieť, že hodnota bunky v našej tabuľke je daná násobkom čísla riadku a stĺpca, v ktorom sa nachádza. Pre prvú bunku, teda B4 musíme vynásobiť hodnotu buniek B3 a A4. Keď toto urobíme pomocou vzorca B4=`B3*A4` a z bunky B4 ho potiahneme nadol, zistíme, že nerobí to, čo by sme chceli. Problém je, že potiahnutím vzorca z bunky B4 do bunky B5 sa zmení vzorec na B5=`B4*A5` namiesto toho, aby sa zmenil na B5=`B3*A5`.

Riešením tohto problému je "uzamknutie" čísla riadku tejto bunky. Tým docielime, že sa nebude po potiahnutí vzorca smerom nadol meniť označenie bunky B3. Uzamykáme pridaním znaku dolára `$` pred to, čo chceme zamknúť. V našom prípade zamykáme číslo riadku bunky B3, teda náš vzorec v bunke B4 bude `=B$3*A4` a jeho potiahnutím už dostaneme v stĺpci správne výsledky. Podobne pri naťahovaní doprava potrebujeme uzamknúť písmeno stĺpca, takže konečný vzorec pre bunku B4 bude `=B$3*$A4` a úlohu vyriešime natiahnutím tohto vzorca do ostatných buniek.

b) Trojuholník

Jeden spôsob je vyriešiť úlohu pomocou príkazu `IF`. Chceme zapísať, že ak súčet čísel v bunkách nad mojou bunkou a naľavo od mojej bunky je jedna, tak v mojej bunke bude jedna, inak tam bude nula. Napríklad pre bunku C5 bude vzorec vyzerať `=IF(C4+B5=1, 1, 0)`.

Vieme to však spraviť aj trocha elegantnejšie pomocou funkcie `MOD(X, Y)`. Táto funkcia nám vráti zvyšok, ktorý dáva číslo X po delení Y, teda napríklad `MOD(17, 10)` vráti zvyšok, ktorý dá 17 po delení 10, čo je 7.

V našom krokodílom príklade budeme počítať zvyšok po delení dvoma. Využijeme, že `MOD(0, 2)` je 0 aj `MOD(2,2)` je 0 a `MOD(1,2)` je 1. Pre bunku C5 teda dostaneme vzorec `=MOD(C4+B5, 2)`.

Natiahneme vzorec do celého bojového pola a dostaneme rozostavenie vojakov. Otázkou ešte ostáva, ako zvýrazniť na bojovom poli miesta, kde sa nachádzajú krokodíly. Použijeme na to podmienené formátovanie. To nájdeme v Google Sheets v kolónke `Format` pod názvom `Conditional Formatting`. Označíme oblasť, ktorú chceme formátovať a klikneme na `Add another rule`, teda pridať Pravidlo formátovania. Tu nastavíme podmienku, ktorá bude vyzerať takto: `Format cells -> „if Is equal to 1"`. Nakoniec zvolíme, typ formátovania, teda zmenu farby bunky. V Exceli označíme formátovanú oblasť a použijeme nástroj podmienené formátovanie, kde nastavíme `Formátovať bunky` obsahujúce hodnotu rovnú 1 a nastavíme typ formátovania, teda zelenú farbu bunky. V OpenOffice použijeme `Podmienené formátovanie` z položky `Formát` a postupujeme rovnako ako pri Exceli.

Obrazec, ktorý sme dostali je nazývaný aj Sierpinského trojuholník. Pekné obrázky tohto tvaru nájdete aj na Wikipédii.

So Sierpinského trojuholníkom súvisí aj Pascalov trojuholník, objekt známy pre svoje zaujímavé matematické vlastnosti. sk.wikipedia.org/wiki/Pascalov_trojuholník

c) Peniaze

Táto úloha má celkom priamočiare riešenie. Pokúsime sa zaradom odpovedať na otázky zo zadania.

Hodnotu vreckového za rok zistíme ako súčet vreckového za jednotlivé dni roka, B3=`SUM(B24:B388)`. Podobne zistíme aj hodnotu celkového nákupu za rok, B4=`SUM(C24:C388)`. Celkový zisk vypočítame ako *celkové vreckové mínus celkový nákup*, obidve hodnoty už poznáme, B5=`B3-B4`.

Na zistenie, ako dlho bol cez rok Karol v mínuse budeme potrebovať pomocný výpočet. Ten bude vyzerať tak, že pre každý deň si vypočítame aktuálny stav účtu. Pre prvý deň je to *vreckové mínus nákup* teda `=B24-C24`, túto hodnotu si uložíme do bunky E24. Pre druhý a všetky ostatné dni to bude, *stav účtu deň predtým plus vreckové mínus nákup*, napríklad pre druhý deň je tento vzorec E25=`E24+B25-C25`, tento vzorec potiahneme smerom nadol a v políčkach E24 až E388 budeme mať stav účtu pre jednotlivé dni. Teraz, keď už vieme rýchlo pre každý deň povedať, či bol v tento deň Karol v mínuse, spočítame počet týchto dní. Použijeme na to ďalší pomocný výpočet a v bunkách F24 až F388 si označíme 1 každý deň, kedy bol Karol v mínuse – vzorec, ktorý prenesieme z bunky F24 bude `=IF(E24<0,1,0)`. Počet jednotiek v tomto stĺpci je rovný súčtu tohto stĺpca. Riešením je teda vzorec B6=`SUM(F24:F388)`.

Iným spôsobom ako spočítať počet záporných čísel v stĺpci E je použiť priamo na takéto účely určenú funkciu: B6=`COUNTIF(E24:E388, „<0")`.

Pre vypočítanie priemerného stavu účtu potrebujeme vedieť stavy účtu pre jednotlivé dni. Tie už máme vypočítané, preto použijeme funkciu `AVERAGE()`, ktorá vypočíta aritmetický priemer zadaných buniek. K výsledku nám už stačí len použiť vzorec B8=`AVERAGE(E24:E388)`.

Ak chceme zistiť priemerné výdavky v pondelky, potrebujeme zistiť, ktoré dni sú pondelky. V stĺpci G chceme mať čísla 1 až 7 určujúce dni v týždni. Jednotka bude pondelok, dvojka utorok a tak ďalej. Pozrieme do kalendára a vidíme, že prvého januára 2015 bol štvrtok. Takže v G24 bude číslo 4. V ostatných bunkách bude vzorec tvaru `=MOD(X, 7)+1`. Tento vzorec nám vráti $X + 1$, ak X je menej ako 7 a jednotku, ak X je 7. Do bunky G25 teda dáme `=MOD(G24, 7)+1` a do zvyšných buniek skopírujeme natiahnutím.

Následne si do stĺpca H dáme jednotky do tých riadkov, ktoré zodpovedajú pondelkom a nuly všade inde: H24=`IF(G24=1, 1, 0)`. Do stĺpca I si potom dáme ku každému pondelku výdavky v tento deň a nulu ku všetkým ostatným dňom (I24=`IF(H24=1, C24, 0)`, alebo I24=`H24 * C24`). Priemerné výdaje v pondelky potom vypočítame ako súčet pondelkových výdajov vydelený počtom pondelkov, teda B7=`SUM(I24:I388)/SUM(H24:H388)`.

Skúsme odpovedať na poslednú otázku, koľko peňazí Karol v jednotlivých mesiacoch ušetril. Na riešenie znova použijeme pomocný výpočet, tentokrát si budeme pre každý deň pamätať stav účtu v tento deň s tým rozdielom, že každý mesiac bude účet štartovať odznova. Tým docielime, že dostaneme pre každý deň stav účtu vzhľadom na daný mesiac. Na toto riešenie použijeme ešte jeden pomocný stĺpec J, v ktorom si budeme ku každému dňu pamätať mesiac, v ktorom sa nachádza. Na to použijeme funkciu `MONTH()`. Napríklad, `MONTH(A100)` teda `MONTH(„18/03/2015")` bude 3. Pre prvý deň – políčko K24 – bude vzorec vyzerať takto: `=IF(J24=J23, K23, 0)+B24-C24`, teda pripočítame *vreckové mínus nákup* ku predošlej hodnote. Túto predošlú hodnotu budeme v prvý deň každého mesiaca považovať za nulovú.

Iným spôsobom riešenia bolo ručne si pre každý mesiac zistiť jeho prvý a posledný riadok v tabuľke a následne pomocou funkcie `SUM` spočítať odpovede.

d) Výsledkovka

Našou prvou úlohou je zistiť počet bodov pre každého súťažiaceho, pričom si musíme dať pozor, aby sa úloha za najviac a úloha za najmenej bodov rátali iba za polovicu. Toto pre prvú bunku (teda I3) jednoducho zapíšeme vzorcom `=SUM(D3:H3)-MIN(D3:H3)/2-MAX(D3:H3)/2`. Keď máme pre každého súťažiaceho vypočítaný jeho bodový zisk, je potrebné, aby sme výsledkovú listinu zoradili podľa tohto počtu bodov.

Označíme teda celú tabuľku, od políčka B3 až po I213. Potom použijeme funkciu zoradenia podľa hodnoty nejakej bunky, v našom prípade to bude počet bodov. Táto funkcia pracuje tak, že zoradí tabuľku podľa nami zvoleného stĺpca. V Exceli nájdeme túto funkciu pod záložkou `Dáta` s názvom `Zoradiť`. Tu nastavíme "zoradiť podľa" počet bodov "od najväčšieho po najmenší".

Keď už máme súťažiacich zoradených podľa bodového zisku, určíme každému poradové číslo. Vytvoríme si v pomocnom stĺpci od K3 po K213 čísla 1 až 211. Potom poradie súťažiaceho bude buď príslušná hodnota v stĺpci K, ak mal menej bodov ako súťažiaci pred ním, alebo to bude rovnaká hodnota ako mal súťažiaci pred ním. Teda napríklad do A3 dáme vzorec `=IF(I3=I2, A2, K3)`, ktorý môžeme natiahnuť do ostatných riadkov.

Nakoniec maximálne počty za jednotlivé úlohy zvýrazníme rovnakým spôsobom ako jednotky v podúlohe (b).

e) Zvieratá

Na vyriešenie tejto úlohy bolo základom úspechu dobre odsimulovať celý proces a nikde sa nepomýliť.

Najjednoduchším riešením je napísať vzorec pre počty zvierat po konci druhého roku a tento vzorec potom použiť aj pre iné dni. V našom prípade je počet krokodílov $k = C6$, počet mamutov $m = B6$, počet holubov $h = D6$ a aktuálny rok je $i = A6$.

Keď si prečítame zadanie, vidíme, že počet mamutov na druhý rok m_2 je daný vzťahom $m_2 = m + 2 - h/200000 - k/2000 - i/500$. Dosadením hodnôt buniek za premenné k, m, h dostaneme vzorec pre bunku B7, teda počet mamutov na druhý rok `=B6+2-(D6/200000)-(C6/2000)-(A6/500)`. Podobne počet holubov zapíšeme vzorcom D7`=D6+5*SQRT(D6)-3*C6`. Napokon, počet krokodílov v druhom roku je daný vzorcom C7`=C6*0.99-B6/20+D6/10000`.

Tieto 3 vzorce môžeme jednoducho ťahať smerom nadol, pokým neklesne počet mamutov pod nulu. Pre vykreslenie grafu sme si tabuľku skopírovali aj so správnymi konštantami, celú tabuľku sme označili a vybrali sme vhodný typ grafu.

f) Jedlo

V úlohe sa pýtame, koľko najviac holubov vie Karol zjesť, keď skončí úplne na juhovýchode. My však zistíme pre každé políčko námestia, koľko najviac holubov by Karol mohol zjesť, keby sa mal dostať iba po toto políčko. Zistíme teda oveľa viac ako úloha žiada, no napriek tomu to bude jednoduchšie. Dôvodom je to, že tieto čiastkové odpovede spolu úzko súvisia.

Označíme si maximálny počet holubov, ktoré môže Karol zjesť, ak skončí na políčku (i, j) (t. j. na políčku v i-tom riadku a j-tom stĺpci) ako $holuby(i, j)$. Ako sa mohol Karol dostať na políčko (i, j)? Buď tam prišiel zo severu alebo zo západu, čiže z políčka $(i - 1, j)$ alebo z políčka $(i, j - 1)$. Takže maximálny počet holubov, ktorý mohol zjesť pred tým, ako sa dostal na políčko (i, j) bude väčšie z čísel $holuby(i - 1, j)$ a $holuby(i, j - 1)$. Navyše mohol nejakého holuba zjesť na políčku (i, j).

$$holuby(i, j) = \max(holuby(i - 1, j), holuby(i, j - 1)) + \text{počet holubov na } (i, j)$$

Pre riešenie úlohy si teda vytvoríme novú tabuľku od bunky B41 napravo a dolu. Do buniek dáme vzorec, ktorý spočíta $holub(i, j)$ pre dané políčko. Presnejšie, do bunky B41 napíšeme `=MAX(B40, A41)+IF(B7="H", 1, 0)` a natiahneme do zvyšných buniek. V bunke W72 sa nám potom zjaví odpoveď na úlohu. Úžasné, však?

Posledný problém, s ktorým sme sa museli popasovať je vykreslenie najlepšej Karolovej trasy. Tú budeme vyrábať odzadu. Budeme sa snažiť, aby na políčkach trasy bola jednotka a na ostatných nula. Vieme, že sa Karol niekedy musel dostať do cieľa, takže na úplne juhovýchodné políčko dáme jednotku.

Teraz potrebujeme zistiť, či je na toto políčko lepšie prísť zo severu alebo zo západu. Ak je väčší počet holubov na západe, tak je lepšie prísť zo západu, ak je viac holubov na severe, je lepšie prísť zo severu. V prípade rovnakých počtov je jedno odkiaľ sme prišli, ale nemôžeme prísť naraz z oboch smerov. Preto si určíme, že v prípade remízy chceme prísť zo severu.

To bola celá myšlienka, zvyšok roboty je zapísať to do tabuľky. Sú dva možné dôvody, prečo by v políčku (i, j) mala byť jednotka. Prvý dôvod je, keď z daného políčka budeme v ceste pokračovať na juh, teda keď v $(i + 1, j)$ je jednotka a zároveň $holuby(i, j) \geq holuby(i + 1, j - 1)$. Druhý dôvod je, keď z daného políčka budeme pokračovať na východ. Jeho formálny zápis bude podobný, len máme $>$ namiesto $\geq$. Navyše, iba jeden

z týchto dôvodov sa môže uplatniť, pretože nemôže byť jednotka aj na políčku $(i+1, j)$ aj na $(i, j+1)$. Takže v tabuľke to môžeme zapísať ako súčet dvoch IFov. Takto vyzerá vzorec v políčku AW72: `=IF(W72>=V73, AW73, 0)+IF(W72>X71, AX72, 0)`, do ostatných políčok stačí vzorec natiahnuť z tohto.

Záverečné slovo

Videli sme, že tabuľkové kalkulátory sú silné nástroje. Ale to sme videli len malý zlomok ich schopností. Dá sa v nich spočítať oveľa viac, teoreticky sa v nich dá spočítať všetko, čo sa dá vôbec spočítať ;). Existuje dokonca aj počítačová hra naprogramovaná v takomto kalkulátore. Ale netreba robiť všetko v tabuľkách. Primárny cieľ tabuliek je spracovávanie dát.

Pokiaľ potrebujete naprogramovať aplikáciu, alebo nejaký zložitejší výpočet, je lepšie siahnuť po nejakom praktickejšom programovacom jazyku ako je C++, Python alebo Java. Ak sa raz naučíte dobre pracovať s týmito jazykmi, budete schopní robiť úžasné veci, o ktorých sa vám možno ani nesnívalo.

4. Perníková optimalizácia

vzorák napísal Matúš
(max. 15 b za riešenie)

Spôsobov ako riešiť túto úlohu je mnoho. V tomto vzorovom riešení sa postupne pozrieme na zopár z nich, začínajúc od najmenej efektívnych.

Ako prvé asi každému napadne nasledovné riešenie – budeme postupne skúšať čísla n a vyššie a pre každé overovať, či vyhovuje podmienkam zadania (teda či je deliteľné aspoň jedným z čísel a, b, c). Akonáhle nájdeme m vyhovujúcich čísel, môžeme skončiť.

Takéto riešenie úspešne zvládne prvé tri testovacie sady, na zvyšné je však príliš pomalé. Čísel, ktoré musíme vyskúšať, je v najhoršom prípade až $m \cdot min(a, b, c)$, čo je vzhľadom na veľkosti vstupov zhruba m^2. To znamená, že náš algoritmus bude musieť v najhoršom prípade urobiť zhruba m^2 krokov. Povedané odbornejšie, jeho časová zložitosť je $O(m^2)$. Takéto riešenie je pomalé pre m väčšie ako pár tisíc.

Listing programu (C++)

```
#include <iostream>

using namespace std;

int main() {
    long long n, m, a, b, c;
    cin >> n >> m;
    cin >> a >> b >> c;
    int pocet = 0;
    while (pocet < m) {
        if(n%a == 0 || n%b == 0 || n%c == 0) {
            cout << n;
            pocet++;
            if(pocet == m) cout << endl;
            else cout << " ";
        }
        n++;
    }
}
```

Ak chceme zlepšiť náš algoritmus, musíme vymyslieť spôsob ako neskúšať veľa zbytočných čísel. Uvedomme si, že čísla, ktoré máme vrátiť na výstup sú všetko násobky čísel a, b alebo c. Určite teda nebudeme potrebovať viac ako prvých m násobkov a väčších ako n, prvých m násobkov b väčších ako n a prvých m násobkov c väčších ako n. Všetky čísla, ktoré máme vypočítať sa musia nachádzať medzi týmito $3m$ číslami.

Poďme sa preto pozrieť, ako by sa dala vygenerovať postupnosť násobkov čísla a väčších ako n alebo rovných n.

V prvom rade treba nájsť najmenší prvok – teda najmenšie číslo deliteľné a, ktoré je väčšie alebo rovné n. Vo väčšine programovacích jazykoch máme celočíselné delenie. To znamená, že podiel n/a sa zaokrúhľuje nadol a ak je jeho výsledok číslo x, tak platí, že $x \cdot a \leq n$. Ak si teda napríklad v C++ zoberieme výraz `(n/a) * a`, nedostaneme hodnotu n, ale násobok čísla a, ktorý nie je väčší ako n.

V tomto momente musíme rozlíšiť dve možnosti. Ak je n deliteľné číslom a, tak n je najmenší násobok čísla a, ktorý je väčší alebo rovný n. V opačnom prípade, ak číslo n nie je deliteľné a, je týmto najmenším vhodným násobkom číslo `(n/a) * a + a` (ak sme mali najväčší násobok a menší ako n, tak pričítaním a dostaneme násobok väčší ako n).

Ak však nechceme písať zbytočné `if`-y na rozlišovanie týchto dvoch možností, dá sa to zhrnúť aj do jedného výrazu `((n+a-1)/a) * a` (v iných jazykoch analogicky). Uvedomte si, že ak n bolo deliteľné a, tak výsledok `(n+a-1)/a` bude rovnaký ako `n/a`, lebo podiel sa zaokrúhlil nadol, ak však n malo nejaký zvyšok po delení a,

tak hodnota `(n+a-1)/a` bude rovná `(n/a)+1`, čo je presne to, čo sme chceli. (Pozor si treba dať v Pythone, kde sa celočíselné delenie zapisuje dvoma lomkami `n//a`.)

No a ak máme najmenšie číslo, všetky nasledujúce násobky získame pripočítavaním čísla a, čo vieme spraviť v jednom `for` cykle. Samozrejme, pre čísla b a c funguje úplne rovnaký postup.

Listing programu (C++)

```
#include <iostream>

using namespace std;

long long pole[1000000]; // dostatocne velke pole

void postupnost(long long n, int m, long long a) {
    long long zac = ((n+a-1)/a) * a;
    for (int i = 0; i < m; i++) {
        pole[i] = zac;
        zac += a;
    }
}
```

Ostáva otázka, čo spraviť s týmito troma postupnosťami m čísel a ako z nich získať výslednú odpoveď. Predstavme si, že ich dáme všetky dokopy a usporiadame od najmenšieho po najväčšie, pričom vyhádžeme duplikáty (opakujúce sa čísla). Potom prvých m čísel z toho, čo nám zostane bude odpoveďou na našu úlohu.

V tomto mometne niektorí z vás vedia naprogramovať nasledovný algoritmus: pre každé z čísel a, b a c si vypočítame prvých m čísel väčších ako n a deliteľných týmto číslom. Tieto tri postupnosti spojíme do jedného poľa a toto pole usporiadame nejakým štandardným knižničným algoritmom. Následne prejdeme toto usporiadané pole a vždy keď narazíme na číslo, ktoré sme ešte nevypísali (je iné ako predchádzajúce číslo), tak ho vypíšeme. Keď vypíšeme m čísel algoritmus zastavíme.

Problém ale je, že sme museli použiť triedenie z nejakej knižnice (čo síce môžete robiť ak to poznáte, ale nie všetci to vedia) a časová zložitosť tohto riešenia nie je optimálna, lebo triedenie je o niečo pomalšie (o pomerne málo, takže takéto riešenie vám pravdepodobne aj tak pobehá na plný počet bodov, to však nie je dobrý pohľad).

Potrebujeme teda vymyslieť spôsob, ako získať m najmenších čísel z troch postupností. Ešte sme nevyužili fakt, že čísla v týchto postupnostiach sú už usporiadané od najmenšieho po najväčšie. Predstavme si, že máme tri postupnosti kartičiek, na ktorých sú napísané rôzne čísla. V každej postupnosti sú však tieto čísla usporiadané zľava doprava rastúco. Naším cieľom je vytvoriť z týchto troch postupností jednu spoločnú, usporiadanú postupnosť, ktorá neobsahuje rovnaké čísla viackrát.

Ktoré číslo môže byť na začiatku tejto výslednej postupnosti? Iba jedno z troch naľavejších kartičiek, na ktorých sú najmenšie čísla daných postupností. No a samozrejme, to najmenšie z nich. Pozrieme sa teda na začiatky postupností, zistíme, ktoré číslo má byť ako prvé a túto kartičku odoberieme a položíme ako prvú kartičku výslednej postupnosti. Naviac, aby sme sa zbavili duplikátov, tak ak najmenšie číslo bolo na viacerích najľavejších kartičkách, tie zvyšné zahodíme.

No a tento postup môžeme opakovať. Ak chceme vedieť, ktoré číslo bude na druhom mieste, opäť budeme hľadať na začiatku troch postupností, ktoré teraz vyzerajú trošku ináč, lebo sme niektorím odstránili najmenší prvok. Takto pokračujeme až kým sa nám neminú všetky postupnosti kartičiek.

Ak sme si teda nami vygenerované tri postupnosti deliteľov čísel a, b a c uložili do troch polí, môžeme m krát zopakovať vyššie popísaný postup a získať správnu odpoveď. Vyhadzovanie kartičiek zo začiatkov polí implementujeme tak, že si pre každé z našich troch polí budeme v jednej premennej pamätať, kde sa v ňom nachádza prvá nevyhodená kartička. Ak chceme z daného poľa prvú kartičku vyhodiť, túto premennú zvýšime o 1.

V zadaní sa však písalo, že táto úloha sa dá riešiť bez toho, aby sme použili nejaké polia.

Už sme si povedali, že vieme vypočítať najmenšie násobky čísel a, b a c, ktoré sú väčšie alebo rovné n. Označme si tieto najmenšie násobky x, y a z. Je jasné, že prvé číslo, ktoré chceme vypísať na výstup bude číslo $\min(x, y, z)$, lebo sú to najmenšie čísla našich postupností. Nech napríklad toto najmenšie číslo je x a navyše $x = y$. Potom tieto dve čísla už nemôžeme použiť a hľadáme ďalšie násobky čísel a a b. Ale to sú samozrejme čísla $x + a$ a $y + b$ (ak budeme potrebovať ďalší násobok čísla c, bude to $z + c$). Vieme teda upraviť hodnotu x aj y a pokračovať rovnakým spôsobom. Opäť sa pozrieme na minimum x, y a z, vypíšeme ho a príslušné premenné zväčšíme o príslušnú hodnotu (čo je v podstate to isté, ako keď sme v kartičkovom riešení vyhadzovali najľavejšiu kartu z radu). Toto zopakujeme m krát, čím získame výsledok.

Na toto riešenie nám stačí zopár premenných a nepotrebujeme použiť žiadne pole. O takýchto riešeniach hovoríme, že majú konštantnú pamäťovú zložitosť, čo zapíšeme ako $O(1)$. Čo sa týka rýchlosti tohto algoritmu, musíme urobiť rádovo m operácií, takže časová zložitosť je $O(m)$.

Listing programu (C++)

```cpp
#include <iostream>

using namespace std;

int main() {
    long long n, m, a, b, c;
    cin >> n >> m;
    cin >> a >> b >> c;
    long long x = ((n + a -1)/a) * a;
    long long y = ((n + b -1)/b) * b;
    long long z = ((n + c -1)/c) * c;

    for (int i = 0; i < m; i++) {
        long long mini = min(x, min(y,z)); //minimum vie zobrat iba dva argumenty
        if(mini == x) x += a;
        if(mini == y) y += b;
        if(mini == z) z += c;
        cout << mini;
        if(i == m-1) cout << endl;
        else cout << " ";
    }
}
```

Listing programu (Python)

```python
n, m = map(int, input().split())
a,b,c = map(int, input().split())
x, y, z = (n//a)*a, (n//b)*b, (n//c)*c

for i in range(m):
    if x < n: x += a
    if y < n: y += b
    if z < n: z += c

    # Za cislom vypiseme koniec riadku, ak je to posledne cislo.
    # V opacnom pripade vypiseme medzeru.
    print(min(x,y,z), end=' \n'[i==m-1])
    n = min(x,y,z) + 1
```

5. Poradie tanečníkov

vzorák napísal Roman
(max. 15 b za riešenie)

V tomto vzorovom riešení začneme tým, že si ukážeme ako zistiť, ktorý žiak je prvý v rade. S pomocou tejto informácie navrhneme algoritmus na riešenie úlohy a pokúsime sa ho čo najviac zefektívniť. Pripravení?

Kto je prvý v poradí?

Zadanie nám garantuje, že v polovici testovacích sád bude prvý vždy žiak s číslom 1. To nám však k vzorovému riešeniu nestačí. Na vstupe dostaneme o $n-1$ študentoch informáciu, kto sa nachádza pred nimi. Prečo o $n-1$ a nie n? Pretože niekto musí byť vždy prvý v rade. A pred ním sa už nenachádza nikto. Potrebujeme teda nájsť číslo žiaka, o ktorom nedosatneme na vstupe informáciu o žiakovi pred ním.

Použijeme na to n prvkové pole `videni[]`, ktoré bude najskôr obsahovať samé 0. Na pozíciu `videni[i]` zapíšeme 1, ak sme na vstupe dostali informáciu o žiakovi, ktorí stojí pred žiakom i. Na konci celé pole prejdeme a pozícia, na ktorej sa nachádza 0, je náš hľadaný prvý žiak v poradí. Ak používame C++, musíme si ešte dať pozor na indexovanie prvkov poľa – spraviť ho buď o 1 prvok väčšie, alebo všetky čísla žiakov zmenšovať o 1.

Listing programu (C++)

```cpp
#include <iostream>

using namespace std;

int main() {
    int prvy, n, x, y;
    cin >> n;
    int videni[n+1];
    for (int i = 1; i <= n; ++i) videni[i] = 0;

    for (int i = 0; i < n-1; ++i) {
        cin >> x >> y;
        videni[x] = 1;
    }
    for (int i = 1; i <= n; ++i) {
        if (videni[i] == 0) {
            prvy = i; break;
        }
    }
    cout << prvy << endl;
}
```

Základná myšlienka

Teraz už vieme, kto je na prvej pozícii – nech je to žiak s číslom k. Potrebujeme zistiť, kto sa nachádza na druhom mieste. Niekde na vstupe sme určite dostali dvojicu v tvare `z k`, ktorá hovorila, že pred tanečníkom z sa nachádza tanečník k. To ale znemená, že na pozícii 2 musí byť žiak s číslom z. Následne môžeme zobrať číslo z a zopakovať rovnaký postup pre zistenie čísla na tretej pozícii. Stačí ak nájdeme na vstupe dvojicu `y z`. A toto môžeme opakovať postupne pre všetky pozície, teda $n-1$ krát ($n-1$ preto, lebo prvý prvok sme zisťovali zvlášť).

Teraz už potrebujeme iba pre každého žiaka vedieť rýchlo vyhľadať, kto stojí za ním (tohto človeka nazvime *prechodcom* nášho žiaka), aby sme sa mohli postupne posúvať ďalej. Ukážeme si 2 možné riešenia.

Pomalšie riešenie

Prvé riešenie spočíva v tom, že zakaždým budeme prechádzať všetky údaje, ktoré sme dostali na vstupe, až kým nenájdeme hľadanú informáciu. Vstup si budeme pamätať ako n prvkové dvojrozmerné pole `ziaci[n+1][2]`, kde sa na i-tej pozícii nachádza i-ta dvojica zo vstupu.

Keď budeme hľadať predchodcu čísla k, postupne prejdeme celé pole `ziaci[][]`, až kým sa prvok `ziaci[i][1]` nebude rovnať k. Potom vieme, že v rade stojí za žiakom k žiak s číslom `ziaci[i][0]`.

Nakoľko budeme $n-1$ krát prechádzať $n-1$ prvkov poľa, náš algoritmus spraví rádovo $(n-1)^2$ operácií, čo je zhruba n^2. Povedané vznešenejšie, náš algoritmus má časovú zložitosť $O(n^2)$. Keďže používame iba dve n prvkové polia, pamäťová zložitosť bude $O(n)$.

Listing programu (C++)

```
#include <iostream>

using namespace std;

int main() {
    int n, x, y, k, z;
    cin >> n;
    int ziaci[n+1][2], videni[n+1];

    for (int i = 1; i <= n; ++i) videni[i] = 0;

    for (int i = 0; i < n-1; ++i) {
        cin >> x >> y;
        ziaci[i][0] = x;
        ziaci[i][1] = y;
        videni[x] = 1;
    }

    for (int i = 1; i <= n; ++i) {
        if (videni[i] == 0) {
            k = i; break;
        }
    }

    cout << k << endl;

    for (int i = 0; i < n-1; ++i) {
        //vyhladavame nasledovnika k
        for (int j = 0; j < n-1; ++j) {
            if (ziaci[j][1] == k) {
                z = ziaci[j][0];
                break;
            }
        }
        //nasli sme ho, vypiseme ho a stava sa novym k
        cout << z << endl;
        k = z;
    }
}
```

Vzorové riešenie

Čo ak by sme si vstup pamätali trochu prefíkanejšie? Vytvorme n (alebo $n+1$) prvkové pole `ziaci[n+1]`, kde na i-tej pozícii bude číslo žiaka, ktorý stál v rade za žiakom i. Ak potom budeme chcieť v našom algoritme nájsť ďalšieho v poradí po k, stačí sa pozrieť na pozíciu `ziaci[k]`.

Informácia zo vstupu v tvare `x y` nám hovorí, že pred tanečníkom x sa nachádza tanečník y. To ale inak povedané znamená, že za tanečníkom y sa nachádza tanečník x. A to je práve tá informácia o predchodcovi, ktorú sme hľadali. Do nášho poľa si preto poznačíme `ziaci[y] = x`. Opäť si dajte pozor na indexovanie.

Nájdenie nasledovníka nám teraz zaberie iba konštantný čas $O(1)$, pretože sa stačí pozrieť na jeden prvok poľa. Tento proces budeme opakovať $n-1$ krát, preto časová zložitosť optimálneho riešenia je $O(n)$. Pamäťová zložitosť sa oproti pomalšiemu riešeniu nezmenila, nakoľko stále používame iba 2 polia dlhé n, a je stále $O(n)$.

Listing programu (C++)

```cpp
#include <iostream>

using namespace std;

int main() {
    int n, x, y, k, z;
    cin >> n;
    int ziaci[n+1], videni[n+1];

    for (int i = 1; i <= n; ++i) videni[i] = 0;

    for (int i = 0; i < n-1; ++i) {
        int x, y; cin >> x >> y;
        videni[x] = 1;
        ziaci[y] = x;
    }

    for (int i = 1; i <= n; ++i) {
        if (videni[i] == 0) {
            k = i;
            break;
        }
    }

    cout << k << endl;
    for (int i = 0; i < n-1; ++i) {
        z = ziaci[k];
        cout << z << endl;
        k = z;
    }
}
```

Listing programu (Python)

```python
n = int(input())

ziaci = [0] * (n+1)
videni = [0] * (n+1)

for i in range(n-1):
    x,y = [int(x) for x in input().split()]
    videni[x] = 1
    ziaci[y] = x

k = 0
for i in range(1,n+1):
    if videni[i] == 0:
        k = i
        break

print(k)
for i in range(n-1):
    k = ziaci[k]
    print(k)
```