B-II-1 Krokodília cesta

Cez rieku sa *nedá* prejsť práve vtedy, keď sú niekde tri ponorené krokodíly za sebou. Takto dostávame priamočiare riešenie: Budeme mať pole boolovských premenných, v ktorom si o každom krokodílovi budeme pamätať, či je práve vynorený (true) alebo ponorený (false). Po každej udalosti prejdeme toto pole a skontrolujeme, či v ňom niekde nemáme tri ponorené krokodíly vedľa seba.
Ako sa toto riešenie dá zefektívniť? Trojicu vedľa seba stojacich krokodílov, ktorí sú všetci ponorení, nazveme zlá. Okrem vyššie popísaného poľa si už stačí pamätať len jednu číselnú premennú: aktuálny počet zlých trojíc. Každý konkrétny krokodíl je súčasťou nanajvýš troch trojíc. Keď sa teda nejaký krokodíl ponorí, najviac tri trojice sa môžu zmeniť z dobrých na zlé. A naopak, keď sa krokodíl vynorí, najviac tri trojice sa môžu zmeniť zo zlých na dobré. Toto vieme skontrolovať v konštantnom čase a podľa toho, čo zistíme, si upraviť počet zlých trojíc.
Cez rieku sa dá prejsť práve vtedy, keď je zlých trojíc presne nula.
V programe, ktorý nájdete nižšie, máme dve drobné technické zmeny: krokodíly číslujeme od nuly, a namiesto boolovských premenných používame nuly a jedničky. To druhé nám umožní pohodlnejšie zistiť, či je trojica zlá: je to vtedy, keď má súčet nula.

Listing programu (Python)

```
N = int( input() )
krokodily = [ 1 for n in range(N) ]
zlych_trojic = 0

def je_trojica_zla(x):
    # vrati 1 ak krokodily [x,x+1,x+2] existuju a su vsetky ponorene, inak vrati 0
    if x < 0 or x+2 >= N: return 0
    if sum(krokodily[x:x+3]) == 0: return 1
    return 0

while True:
    # nacitame dalsiu udalost
    zmena = int( input() )
    if zmena == 0: exit()

    # v programe krokodily cislujeme od 0, upravime si cislo zo vstupu
    k = abs(zmena)-1

    # prepocitame pocet zlych trojic: od celkoveho poctu odpocitame tie, ktore
    # obsahuju krokodila k, potom zmenime stav krokodila k, a na zaver znova
    # skontrolujeme trojice, ktore ho obsahuju, a pripocitame tie zle
    zlych_trojic -= je_trojica_zla(k-2)
    zlych_trojic -= je_trojica_zla(k-1)
    zlych_trojic -= je_trojica_zla(k)

    krokodily[k] = 1 - krokodily[k]

    zlych_trojic += je_trojica_zla(k-2)
    zlych_trojic += je_trojica_zla(k-1)
    zlych_trojic += je_trojica_zla(k)

    # vypiseme ci sa da prejst cez rieku
    print('ano' if zlych_trojic == 0 else 'nie')
```

B-II-2 Okrasné diplomy

Na okraj obdĺžnikového papiera máme napísať zadaný reťazec tak, aby žiaden z rohov nerozdeľoval žiadne zo slov. Našou úlohou je nájsť všetky možné rozmery papiera, ktoré niečo takéto spĺňajú. Vieme, že náš text musí začínať v ľavom hornom rohu papiera a musí presne zaplniť celý okraj. Vieme teda, že ak bude mať zvolený papier rozmery $a \times b$ (kde $a, b \geq 2$), tak musí platiť, že $2a + 2b - 4$ sa rovná dĺžke reťazca. To ale znamená, že náš reťazec musí mať párnu dĺžku. A navyše vieme povedať, že písmeno, ktorým začína druhá polovica zadaného reťazca, vždy padne presne do pravého dolného rohu diplomu. Na začiatku riešenia teda overíme dve podmienky: či je dĺžka zadaného reťazca párna a či písmeno, ktoré vychádza do pravého dolného rohu, nie je uprostred nejakého slova.
Podarilo sa nám určiť pozície dvoch protiľahlých rohov, ostávajú ešte dva. Ich pozícia je závislá od rozmerov papiera, teda od hodnôt a a b. Uvedomme si však, že ak určíme jednu z týchto hodnôt, druhú vieme dopočítať

z dĺžky reťazca. Ak teda vyskúšame všetky možné hodnoty čísla a, pozrieme sa postupne na všetky možné rozmery papiera. A týchto možností je málo: je ich najviac polovica dĺžky zadaného reťazca.
Pre každú z možných hodnôt ešte musíme overiť, či je papier týchto rozmerov prípustný. Na to sa treba pozrieť na to, či zvyšné dva rohy nerozseknú nejaké slovo. Ich pozície sú však určené, pravý horný roh je a-te písmeno od začiatku reťazca, ľavý dolný a-te písmeno od stredu reťazca. Kontrolu v takomto prípade vieme spraviť v konštantnom čase.
Výsledné riešenie teda iba vyskúša všetky možné hodnoty a a zistí, či sa všetky rohy nachádzajú na začiatku alebo konci slova, poprípade na medzere. Správne možnosti vypíše. Ak si hodnotou n označíme dĺžku reťazca, časová zložitosť takéhoto riešenia je $O(n)$. Takéto riešenie je zjavne optimálne, keďže rádovo rovnako dlho trvá už len samotné načítanie reťazca zo vstupu. Nemá preto žiaden zmysel snažiť sa riešenie akokoľvek „optimalizovať“ – napríklad tým, že by sme skúšali iba hodnoty a zodpovedajúce začiatkom a koncom slov.

Listing programu (Python)

```
def dobra_pozicia(s, poz):
    if poz == 0 or poz == len(s) - 1:
        return True
    if s[poz] == '␣' or s[poz-1] == '␣' or s[poz+1] == '␣':
        return True
    return False

s = input()
n = len(s)
stred = n // 2
riesenie = False

if n%2 == 0 and dobra_pozicia(s, stred):
    for a in range(1, stred):
        if dobra_pozicia(s, a) and dobra_pozicia(s, stred + a):
            print(stred - a + 1, a+1)
            riesenie = True

if not riesenie:
    print('Nevhodny␣retazec')
```

B-II-3 Mravčia farma II

V tejto úlohe ste pracovali s pomerne bežnou štruktúrou, ktorá sa vyskytuje v informatike – grafom. Pre ňu existujú aj zaužívané pojmy (rozhodne zaužívanejšie ako mravenisko, komôrka či chodbička), predstavme si ich teda na začiatku tohto vzorového riešenia, aby sme sa potom mohli vyjadrovať formálnejšie. Komôrky v našom príklade predstavujú *vrcholy*, chodbičky medzi nimi nazývame *hranami*. *Graf* je tvorený niekoľkými vrcholmi, ktoré sú navzájom pospájané hranami. Naviac, v prípade že graf má taký špeciálny tvar ako v zadaní, voláme ho *zakorenený strom*. Horný vrchol zakoreneného stromu voláme *koreň* a vrcholy, z ktorých sa už nedá ísť ďalej dodola, voláme *listy*.
Keď sme si už ujasnili pojmy, môžeme sa pustiť do riešenia. Našou úlohou je pre odpovedať na niekoľko otázok. Každá otázka je tvaru: Do akého počtu vrcholov na úrovni x sa vieme dostať z určeného začiatočného vrchola pomocou cesty, ktorá ide dohora najviac y krát?
Pri prvom pohľade to vyzerá, že takéto cesty môžu vyzerať tak, že mravec hocijako strieda pohyby hore a dole. Uvedomme si však, že jeho cesta sa dá výrazne zjednodušiť. Vždy keď mravec spraví krok hore potom ako spravil krok dodola, v skutočnosti sa vôbec neposunul. Do rovnakých vrcholov sa teda dostaneme, ak takéto pohyby vylúčime. To ale znamená, že stačí uvažovať také cesty, pri ktorých mravec pôjde najprv chvíľu dohora a potom dodola až na úroveň x. A navyše, môžeme predpokladať aj to, že mravec pôjde dohora práve y-krát. Všetky pohyby dohora, ktoré nechcel spraviť, totiž môžeme odčiniť rovnakým počtom pohybov dodola. Mravcova cesta teda začína tým, že ide y-krát dohora.
(Technický detail: občas sa môže stať, že mravec v skutočnosti nevie spraviť y krokov dohora, keďže už skôr „narazí“ na koreň celého stromu. Toto doriešime tak, že sa dohodneme, že pohyb dohora z koreňa je dovolený, ale nič nerobí – mravec naďalej ostane v koreni stromu.)
Pohyb dohora je v zakorenenom strome jednoznačný. Pre každú začiatočnú pozíciu mravca je teda jednoznačne určené, do ktorého vrcholu ho dostane y krokov dohora. Našu úlohu si rozdelíme na dve časti: najskôr budeme

chcieť efektívne nájsť tento vrchol a potom budeme chcieť zistiť, do koľkých vrcholov na úrovni x sa z neho dá dostať pohybmi dodola.

Stredne dobré riešenie: predpočítanie všetkých odpovedí

Na riešenie druhej podúlohy môžeme využiť riešenie z domáceho kola, prejsť po úrovniach celý strom a zakaždým si pamätať iba tie vrcholy danej úrovne, ktoré ležia pod vrcholom v. Ak si však označíme hodnotou m počet všetkých vrcholov v strome, dostaneme pomerne pomalé riešenie so zložitosťou $O(m)$ na jednu otázku, teda dokopy $O(qm)$. Problémom je, že máme odpovedať na viacero otázok, a pri každej novej otázke prechádzame celým stromom, počítajúc informáciu častokrát duplicitnú vzhľadom na predchádzajúce otázky.
Takéto riešenie sa častokrát dá zrýchliť tým, že si dopredu vypočítame a zapamätáme nejaké dôležité informácie. Koľko rôznych otázok môžeme dostať? Vyzerá, že pomerne veľa, pretože začiatočný vrchol, x aj y sa môžu líšiť. V predchádzajúcej časti sme však videli, že v skutočnosti záleží iba na tom, do ktorého vrcholu sa dostaneme po y krokoch dohora a na ktorej úrovni x chceme skončiť. Všetky otázky sú teda definované hodnotami v a x, a preto je ich iba $O(m \cdot n)$. To síce vyzerá ako veľa možností, ale ak je hodnota $m \leq 1\,000$ tak si ich vieme bez problémov predpočítať dopredu.
Rovnakým spôsobom si vieme pomôcť aj v prvej podúlohe a vypočítať pre každú dvojicu u a y vrchol, do ktorého sa dostaneme zo začiatočného vrcholu u ak sa dohora posunieme y-krát. A v okamihu, keď toto všetko spravíme, na každú otázku vieme odpovedať v konštantnom čase – pozrieme sa do dvoch predpočítaných tabuliek a povieme odpoveď. Zrazu takýchto otázok môžeme bez problémom zodpovedať aj milión.
Ostáva len zistiť, či vieme tieto odpovede vypočítať rozumne efektívne. Ak by sme totiž pre každú mali investovať $O(m)$ času, trvalo by to dlho. Začnime druhou otázkou – do ktorého vrcholu sa dostaneme ak sa z vrcholu u posunieme y-krát dohora. Namiesto toho aby sme celý tento posun simulovali si uvedomíme, že odpoveď je rovnaká ako pre vrchol priamo nad u ak sa z neho posunieme dohora $y - 1$ ráz. A ak túto odpoveď poznáme, môžeme ju rovno použiť. Efektívne riešenie bude teda zisťovať všetky odpovede pre postupne sa zvyšujúce y. Ako prvé sa pozrieme na $y = 0$, vtedy je odpoveďou pre každý vrchol ten istý vrchol. Následne $y = 1$, vtedy je pre každé u odpoveďou vrchol priamo nad ním, čo už máme zadané na vstupe. Keď počítame $y = 2$, odpoveď pôsobí komplikovanejšie, vidíme však, že sme pred chvíľou pre každý vrchol zistili, kam z neho vedie cesta s $y = 1$. Pre vrchol u sa teda pozrieme na vrchol priamo nad ním a jeho odpoveď pre $y = 1$ a to bude našou odpoveďou. Pre zvyšné hodnoty postupujeme rovnako.
V druhom prípade si môžeme všimnúť podobný princíp. Koľko vrcholov na úrovni x je pod vrcholom v? Jedným pohybom dodola sa dostaneme do niektorého z vrcholov o úroveň nižšie. Pre každý z týchto vrcholov by sme mohli vedieť, koľko vrcholov na úrovni x je pod nimi. Tieto vrcholy sú naviac z definície stromu rôzne. Ak ich teda všetky spočítame dokopy, dostaneme hľadanú hodnotu pre vrchol v. Opäť sme teda vôbec nemuseli ísť až na úroveň x, stačilo spraviť iba jeden krok a využiť hodnoty, ktoré sme spočítali predtým. V tomto prípade treba pri počítaní postupovať od spodu stromu, zakaždým totiž potrebujeme už poznať hodnoty z o jedno nižšej úrovne.
Toto riešenie má zložitosť $O(nm + q)$, čo je výrazné zlepšenie oproti prvému prístupu. Navyše sme sa naučili dôležitý trik, ktorý ďalej využijeme aj pri vzorovom riešení. K tomuto riešeniu prikladáme aj implementáciu, ktorá vám ho pomôže lepšie pochopiť.

Listing programu (Python)

```
n = int(input())

strom = []
dohora = []
pocet = []
for i in range(n):
    strom.append([int(_)-1 for _ in input().split()][1:])
    dohora.append([[j] for j in range(len(strom[i]))])
    pocet.append([[1 if k == i else 0 for k in range(n)] for j in range(len(strom[i]))])

# spocitaj cesty dohora
for y in range(1, n):
    for i in range(len(strom)):
        for j in range(len(strom[i])):
            if i == 0:
                dohora[i][j].append(0)
            else:
```

```
                dohora[i][j].append(dohora[i-1][strom[i][j]][y-1])

# spocitaj pocet vrcholov na urovni
for i in range(n-1, 0, -1):
    for j in range(len(strom[i])):
        for k in range(n):
            pocet[i-1][strom[i][j]][k] += pocet[i][j][k]

q = int(input())
for _ in range(q):
    u, v, x, y = map(int, input().split())
    v = v - 1
    if y >= u:
        y = u
    print(pocet[u - y][dohora[u][v][y]][x])
```

Vzorové riešenie

Pre obmedzenia zodpovedajúce plnému počtu bodov si už odpovede na všetky otázky dopredu vypočítať nemôžeme, pretože týchto otázok je pre veľký strom príliš veľa. Musíme teda vedieť rýchlejšie zistiť, ktoré vrcholy môžu byť správnou odpoveďou. Čo by nám umožnilo spočítať všetky vhodné vrcholy bez toho, aby sme sa na každý z nich samostatne pozreli? Uvedomme si, že hľadané vrcholy budú na úrovni x tvoriť súvislú postupnoť, čo vyplýva z toho, že hrany sa v našom strome nekrižujú.[1] Namiesto hľadania všetkých vrcholov by sme sa teda vedeli obmedziť na hľadanie najľavejšieho a najpravejšieho vrchola a výsledný počet zistiť z ich rozdielu.

Skôr, ako si vysvetlíme ako hľadať tieto dva vrcholy, potrebujeme vedieť rýchlo zistiť, či sa vrchol w na úrovni x nachádza pod vrcholom v. Samozrejme, mohli by sme spočítať všetky možné dvojice, tomu sa však chceme vyhnúť. Vypočítanej informácie teda musí byť menej a tým pádom musí byť všeobecnejšia. Vrchol w totiž neleží len pod vrcholom v ale aj pod inými vrcholmi (napríklad tým priamo nad v). Z informácie čo si zapamätáme teda musíme vedieť rýchlo vypočítať odpoveď pre všetky tieto vrcholy.

Čo teda majú spoločné všetky vrcholy, pod ktorými je vrchol w? Majú menšie číslo úrovne ako w a navyše platí, že do každého listu, do ktorého sa vieme dostať z w cestou dodola, sa vieme dostať aj z týchto vrcholov (tiež cestou len dodola). A je zjavné, že táto vlastnosť platí aj naopak: ak nejaký vrchol nemá menšie číslo úrovne ako w alebo sa z neho cestou dodola nevieme dostať do nejakého listu ležiaceho pod w, tak tento vrchol neleží nad vrcholom w.

Ak by sme pre každý vrchol stromu vedeli, do ktorých listov sa z neho vieme dostať iba cestou vedúcou dodola, ľahko by sme overili, či je vrchol w pod vrcholom v. Aby sme túto kontrolu vedeli robiť pohodlne, priradíme listom nové čísla. Tieto čísla budú rásť zľava doprava (bez ohľadu na úroveň, na ktorej príslušné listy ležia). Pri tomto novom číslovaní listov platí, že pre každý vrchol tvoria čísla listov, ktoré pod ním ležia, nejaký súvislý interval. Tieto intervaly si ľahko predpočítame (popísané nižšie) a pomocou nich potom vieme v konštantnom čase overiť, či je nejaký vrchol v nad vrcholom w: stačí, aby bol na vyššej úrovni a aby interval listov pre w bol obsiahnutý v intervale listov pre v.

Na výpočet týchto intervalov vieme použiť rovnaký postup ako sme si ukázali vyššie. Interval pre vrchol v totiž nie je ničím iným, ako zjednotením intervalov pre vrcholy priamo pod ním. Táto informácia teda vie byť vypočítaná v čase $O(m)$. Skutočným problémom je len očíslovanie listov zľava doprava, tieto listy sa totiž môžu nachádzať na rôznych úrovniach.

Aj to však vieme ľahko vyriešiť cez rekurzívnu funkciu, ktorej úlohou bude očíslovať všetky listy v časti stromu pod určeným vrcholom p, na začiatku koreňom celého stromu. V prípade, že daný vrchol p je list, priradí mu ďalšie číslo. A ak to list nie je, tak prejde všetkými vrcholmi priamo pod vrcholom p a zavolá na ne, v poradí zľava doprava, túto istú funkciu. Ako prvé teda očísluje všetky listy pod najľavejším vrcholom, potom pod tým napravo od neho atď. Táto funkcia môže dokonca zároveň vypočítať aj hľadané intervaly (vždy tesne pred návratom z rekurzie) a má časovú zložitosť $O(m)$.

Naše doterajšie riešenie teda v čase $O(m)$ priradí každému vrcholu nejaký interval listov, ktoré ležia pod ním, a vďaka ktorým vieme ľahko zistiť, či sa vrchol w nachádza pod vrcholom v. Poslednou vecou je teda nájsť najľavejší a najpravejší vrchol na úrovni x, ktorý sa nachádza pod vrcholom v.

Posledné pozorovanie, ktoré budeme potrebovať, je nasledovné: Všimnime si ľubovoľnú konkrétnu úroveň nášho

[1] V prípade, že by ste chceli túto vlastnosť formálne dokázať, najjednoduchšie je postupovať indukciou. Začíname so súvislou postupnosťou tvorenou samotným vrcholom v a pohybom dodola dostaneme zo súvislej postupnosti znova súvislú postupnosť.

stromu. Keď budeme prechádzať jej vrcholy zľava doprava a pre každý vrchol sa pozrieme na jeho interval listov, tak tieto intervaly tiež „pôjdu zľava doprava“ – budú disjunktné a každý ďalší bude začínať za koncom predchádzajúceho.

Pomocou vyššie popísaných predpočítaných údajov a pozorovaní teraz už vieme rozumne efektívne vyriešiť celú úlohu.
Zo vstupu dostaneme začiatočný vrchol u a hodnoty x a y. Ako prvé potrebujeme nájsť vrchol v, ktorý je o y úrovní nad vrcholom u. To spravíme tak, že si vypočítame číslo úrovne, na ktorej v leží, a potom ho na nej nájdeme binárnym vyhľadávaním. (Na tejto úrovni hľadáme ten jediný vrchol, ktorý je nad vrcholom u – čiže ten vrchol, ktorého interval čísel listov obsahuje interval pre vrchol u.)
Keď už poznáme vrchol v, pozrieme sa teraz na úroveň x. Na tej použijeme binárne vyhľadávanie dvakrát: raz na to, aby sme našli najľavejší a druhýkrát na to, aby sme našli najpravejší vrchol, ktorý ešte leží pod v.
Keďže na žiadnej úrovni nemôže byť viac ako m vrcholov, má každé binárne vyhľadávanie časovú zložitosť $O(\log m)$. Dokopy má teda toto riešenie časovú zložitosť $O(m + q \log m)$.

Listing programu (Python)

```
n = int(input())

strom = []
dodola = []
for i in range(n):
    strom.append([int(_)-1 for _ in input().split()][1:])
    dodola.append([[] for _ in strom[i]])
    if i == 0:
        continue
    for j, v in enumerate(strom[i]):
        dodola[i-1][v].append(j)

listy = [[] for _ in range(n)]

def spocitaj_listy(uroven, vrchol, pocet_listov):
    if len(dodola[uroven][vrchol]) == 0:
        listy[uroven].append((pocet_listov, pocet_listov))
        return pocet_listov+1
    listy[uroven].append((pocet_listov, pocet_listov))
    for x in dodola[uroven][vrchol]:
        pocet_listov = spocitaj_listy(uroven + 1, x, pocet_listov)
        listy[uroven][vrchol] = (listy[uroven][vrchol][0], pocet_listov - 1)
    return pocet_listov

spocitaj_listy(0, 0, 0)

def porovnaj_intervaly(a, b):
    if a[1] < b[0]:
        return -1
    if a[0] > b[1]:
        return 1
    return 0

q = int(input())

for _ in range(q):
    u, v, x, y = map(int, input().split())
    v = v - 1
    # najdi vrchol o y urovni vyssie
    uroven = max(0, u - y)
    beg, end = 0, len(strom[uroven])
    while end - beg > 1:
        mid = (beg + end) // 2
        if porovnaj_intervaly(listy[uroven][mid], listy[u][v]) <= 0:
            beg = mid
        else:
            end = mid
    u, v = uroven, beg
    # najdi najlavejsi vrchol na urovni x
    beg, end = -1, len(strom[x]) - 1
    while end - beg > 1:
        mid = (beg + end) // 2
        if porovnaj_intervaly(listy[x][mid], listy[u][v]) < 0:
            beg = mid
        else:
            end = mid
    najlavejsi = end
    # najdi najpravejsi vrchol na urovni x
    beg, end = 0, len(strom[x])
    while end - beg > 1:
        mid = (beg + end) // 2
```

```
        if porovnaj_intervaly(listy[x][mid], listy[u][v]) <= 0:
            beg = mid
        else:
            end = mid
    najpravejsi = beg
    print(najpravejsi - najlavejsi + 1)
```

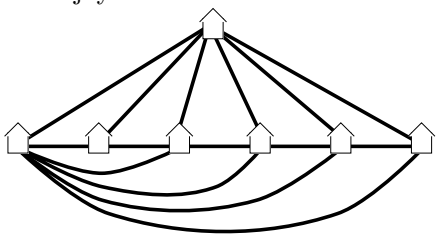
B-II-4 Domy nielen na lúke

Podúloha F (2 body)

Pripomeňme si, že z domáceho kola vieme, že na lúke, na ktorej stojí $n \geq 2$ domov, vieme postaviť nanajvýš $3n - 6$ ciest. To preto, že v optimálnom riešení platí:

- Všetky domy sú zjavne nejak prepojené, preto počet častí lúky (f) vieme pomocou počtu ciest (m) a počtu domov (n) vyjadriť vzťahom $f = m + 2 - n$.
- Každá oblasť lúky je „trojuholník“ a každá cesta patrí dvom častiam lúky, preto celkový počet ciest je $m = 3f/2$.
- Dosadením $f = 2m/3$ do vzťahu z prvého bodu dostávame $3n - 6 = m$.

Pre $n = 7$ domov teda vieme postaviť nanajvýš $3n - 6 = 15$ ciest. Jedna možnosť, ako to spraviť, je na obrázku.

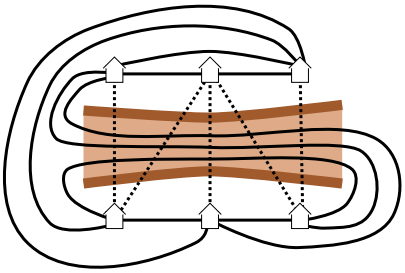


Podúloha G (2 body)

Maľovanie na loptu je rovnakou úlohou, ako keby našu lúku, na ktorej staviame domy a cesty, tvoril povrch celej zemegule. Ukážeme si, že stavať na lúke je to isté ako stavať na celej Zemi, nič iné sa nám nepodarí dosiahnuť. Zjavne na guľu vieme nakresliť čokoľvek, čo vieme fyzicky postaviť v rovine: stačí si vybrať dostatočne malý kúsok povrchu gule a tváriť sa, že je to rovina.
Zložitejšia bude úvaha opačným smerom. Pre tú si predstavme, že je naša guľa vlastne bublina zo skla. Pokreslime ju domami a cestami ako len chceme. Teraz chceme ukázať, že rovnako prepojené domy a cesty vieme postaviť aj v rovine.
Natočme celú guľu tak, aby jej najvyšší bod nebol pokreslený. Do tohto bodu umiestnime lampu a rozsvieťme ju. Na rovinu padne tieň od fixkou pokreslených častí gule. Kam padne tieň domu, tam postavíme dom, kam padne tieň od cesty, tadiaľ povedie cesta v rovine. Takto „premietneme“ celú kresbu z gule do roviny. Rozmyslite si, že ani v rovine sa cesty nebudú križovať.
Záverom tejto úvahy teda dostávame, že aj pre guľu platí, že ak na ňu nakreslíme $n \geq 2$ domov, tak vieme nakresliť nanajvýš $3n - 6$ ciest medzi nimi. Gusto teda na svoju loptu vedel nakresliť nanajvýš 15 ciest.

Podúloha H (2 body)

Zatiaľ čo bez mostu sme nevedeli prepojiť ani len päť domov každý s každým, jediný most nám toho umožní spojiť o dosť viac. Na obrázku nižšie je jedna možnosť, ako prepojiť pomocou jedného mosta šesť domov každý s každým. (Bodkovane sú nakreslené tie cesty, ktoré vedú popod most.)



Podúloha I (1 bod)

Celkový počet strán všetkých stien je ps. Každá hrana mnohostena patrí dvom stenám, preto je hrán $m = ps/2$. Celkový počet vrcholov všetkých stien je tiež ps. Každý vrchol leží na q stenách, preto je vrcholov $n = ps/q$.

Zdôvodnenie Eulerovho vzťahu pre mnohosteny

Prečo platí vzťah, ktorý sme vám uviedli v zadaní podúlohy J? Ide vlastne o ten istý vzťah ako ten, ktorý už poznáme.
Predstavme si, že na náš mnohosten navlečieme vyfúknutý gumený balón tak, aby sa všade dotýkal povrchu mnohostenu. Na balón si fixkou nakreslíme, kde sú jednotlivé vrcholy a hrany mnohouholníka (domy a cesty). Keď tento balón dofúkame, ostanú na ňom nakreslené tie isté domy a medzi nimi nekrižujúce sa cesty, ale už budú všetky na povrchu gule. No a z podúlohy G už vieme, že kreslenie na guli je ekvivalentné so stavaním v rovine.

Podúloha J (3 body)

Keďže každá stena je aspoň trojuholník a v každom vrchole sa stretajú aspoň tri steny, platí $p, q \geq 3$.
V podúlohe I sme si vyjadrili počet vrcholov a počet hrán. Keď ich dosadíme do daného vzorca, dostávame $ps/q + s = ps/2 + 2$. Keďže q je kladné, ekvivalentný tvar tohto vzťahu je $2ps + 2qs = pqs + 4q$.
Z tohto vzťahu intuitívne vidíme, že p aj q musia byť malé, lebo keby niektoré z p a q bolo veľké, bude pqs väčšie ako celá ľavá strana. Podľa tejto intuície teraz budeme exaktne postupovať.
Keďže $4q$ je kladné, musí platiť $2ps + 2qs > pqs$. Keďže s je kladné, musí platiť $2p + 2q > pq$, resp. ekvivalentne $pq - 2p - 2q < 0$.
Ľavú stranu tejto nerovnice si vieme upraviť na $(p-2)(q-2)-4$, čiže naša nerovnica je ekvivalentná s nerovnicou $(p-2)(q-2) < 4$. No a táto nerovnica má v prípustných číslach zjavne len päť riešení: (p, q) môže byť $(3, 3)$, $(3, 4)$, $(3, 5)$, $(4, 3)$, alebo $(5, 3)$.
(Dodáme ešte, že jednotlivým možnostiam, v uvedenom poradí, zodpovedajú pravidelný štvorsten, pravidelný osemsten, pravidelný dvadsaťsten, kocka a pravidelný dvanásťsten.)

TRIDSIATY PIATY ROČNÍK OLYMPIÁDY V INFORMATIKE

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