Pravidlá

Riešenia tejto písomnej skúšky je potrebné odovzdať do FIXME, a to buď vo formáte PDF e-mailom na `misof@ksp.sk` alebo v čase FIXME osobne na M-263.

Počas riešenia je povolené využívať ľubovoľné neživé zdroje informácií ktoré existovali v okamihu, keď ste do rúk dostali jej zadanie (pochopiteľne, vrátane všetkého zverejneného na stránke predmetu). Použité externé zdroje **adekvátne citujte**. Až do deadline (a to bez ohľadu na to, kedy odovzdáte svoje riešenia) je zakázané akýmkoľvek spôsobom diskutovať o úlohách s kýmkoľvek živým. Do hodnotenia sa vám započíta **5 najlepšie vyriešených** úloh. Zapísaniu známky môže predchádzať krátky rozhovor o niektorých úlohách ktoré ste riešili.

---

1 Ukážkové zadanie 1

Na šachovnici rozmerov $n \times n$ je v veží. S nimi hrajú dvaja hráči nasledovnú hru: Hráč, ktorý je na ťahu, si vyberie ľubovoľnú jednu vežu a tou potiahne buď dodola alebo doľava. Potiahnuť vo zvolenom smere môže o ľubovoľne veľa políčok, avšak aspoň o jedno (t.j. nemôže nechať vežu tam kde bola). Veža počas svojho ťahu môže prechádzať cez políčka kde stoja iné veže, dokonca sa medzi ťahmi na každom políčku môže nachádzať ľubovoľne veľa veží. Hráč, ktorý už nevie spraviť platný ťah (lebo všetky veže sú už v ľavom dolnom rohu) prehráva.

Navrhnite čo najefektívnejší algoritmus, ktorý z počtu veží a zo súradníc jednotlivých veží vypočíta, či je daná pozícia hry vyhrávajúca alebo prehrávajúca – teda či existuje vyhrávajúca stratégia pre hráča, ktorý je práve na ťahu.

2 Ukážkové zadanie 2

Dané sú prirodzené čísla $a_1, a_2, \ldots, a_n$ a rozumne malé prirodzené číslo s.
Navrhnite algoritmus, ktorý spočíta, koľko existuje sedmíc indexov $(b_1, \ldots, b_7)$ takých, že $a_{b_1} + \cdots + a_{b_7} = s$.
Hodnoty b_i nemusia byť navzájom rôzne. Na ich poradí záleží.

Na plný počet bodov by váš algoritmus by mal byť prakticky použiteľný pre $n \leq 10^5$ a $a_i, s \leq 10^6$.
Nejaké body budú aj za algoritmus použiteľný pre $n \leq 100$ a $a_i, s \leq 10^6$. Nie, 100^7 nie je dostatočne malé číslo.

(Technický detail: Pre jednoduchosť predpokladajte, že čísla rádovo také veľké ako maximálna možná správna odpoveď sa vám ešte pohodlne zmestia do bežných premenných.)

3 Ukážkové zadanie 3

Máme obdĺžnik rozdelený na $3 \times c$ políčok. Na začiatku je prázdny. Na jeho políčka budeme klásť kamienky, na každé najviac jeden. Pritom musíme dodržať nasledovné pravidlo: *Nikde nesmú byť tri kamienky v rade vedľa seba* (v riadku, stĺpci ani na uhlopriečke).
Rozloženia kamienkov, ktoré spĺňajú vyššie uvedenú podmienku, budeme volať pozície.
Navrhnite algoritmus, ktorý načíta konkrétnu pozíciu a čo najefektívnejšie vypočíta počet rôznych pozícií, ktoré sú z nej dosiahnuteľné pridaním ďalších kamienkov.

4 Ukážkové zadanie 4 (ťažšie)

V hre zo zadania 3: Pre dané c, koľko rôznych pozícií (asymptoticky) existuje?
Inými slovami, koľkými spôsobmi vieme rozložiť ľubovoľne veľa kamienkov tak aby bola splnená podmienka?
Ak neviete nájsť presné riešenie, nájdite aspoň čo najtesnejší dolný a horný odhad.

5 Ukážkové zadanie 5, 6, 7

. . . . . .

SPOILER ALERT
na druhej strane sú riešenia

Riešenie: Ukážkové zadanie 1

Polynomiálne riešenie dostaneme napríklad tak, že si uvedomíme, že všetky veže sú na sebe nezávislé, ide teda o súčet v hier. Stačí nám teda vypočítať Sprague-Grundyho hodnotu pre každé políčko šachovnice pri hre s jednou vežou a potom pozrieť, kde stojí našich v veží, a prexorovať hodnoty pre príslušné políčka.
Samotný výpočet Sprague-Grundyho čísel vieme spraviť napr. polohrubou silou v čase $O(n^3)$: postupne prechádzame šachovnicu, pre každé políčko prejdeme všetky na ktoré z neho vieme ťahať a nájdeme najmenšie číslo, ktoré sa medzi ich číslami nenachádza.
Ešte jednoduchšie a efektívnejšie riešenie však dostávame keď si uvedomíme, že každá veža sú vlastne dve nezávislé kôpky NIMu: každá súradnica zvlášť. Ide teda o dobre zamaskovaný obyčajný $2v$-kopový NIM. Prehrávajúce pozície sú zjavne práve tie, kde je xor všetkých $2v$ súradníc veží rovný nule.

Riešenie: Ukážkové zadanie 2

Zostrojíme si polynóm $p(x) = \sum_i x^{a_i}$, pričom stačí uvažovať len tie a_i ktoré sú $\leq s$. V čase $O(s \log s)$ pomocou DFT/NTT vypočítame $p^7(x)$. Koeficient pri x^s v tomto polynóme je zjavne riešením našej úlohy.

Pre $n = 100$ existujú aj pomalšie riešenia využívajúce buď vhodné dynamické programovanie alebo meet in the middle. Prvý prístup vedie k času $O(ns)$, druhý k času $O(n^4 \log n)$.
Pri dynamickom programovaní si kladieme otázky tvaru $Q(x, y, z) =$„Koľkými spôsobmi viem vyrobiť hodnotu x ako súčet presne y sčítancov, pričom každý z nich je jedno z čísel $a_1, \ldots, a_z$?“
Pri meet in the middle vygenerujeme zvlášť všetky možné súčty pre (b_1, b_2, b_3) a zvlášť všetky pre (b_4, b_5, b_6, b_7).

Riešenie: Ukážkové zadanie 3

Dynamické programovanie s približne $c \cdot 7^2$ stavmi. Riešime otázky nasledovného tvaru: „koľkými spôsobmi viem pridať kamienky na prvých i stĺpcov, ak sú v stĺpcoch $i+1$ a $i+2$ kamienky na týchto konkrétnych miestach?“ Pri riešení konkrétnej otázky vyskúšame všetky prípustné možnosti ako budú rozložené kamienky v i-om stĺpci a zakaždým sa rekurzívne zavoláme s o jedno menšou hodnotou i.
Nižšie uvádzam zjednodušenú implementáciu fungujúcu pre prázdny plán $3 \times c$. Na jej úpravu na všeobecné riešenie by bolo potrebné načítať popis už rozložených kamienkov a potom vo funkcii `solve` preskočiť tie možnosti pre premennú `s0` ktoré nie sú nadmnožinou zadaných kamienkov v dotyčnom stĺpci.

```
# kazdy stlpec ma jeden z tvarov 000, 001, 010, 011, 100, 101, 110
# toto interpretujeme ako cisla 0-6 v dvojkovej sustave

def compatible(s0,s1,s2):
    # mozu po sebe nasledovat stlpce s kodmi s0, s1, s2?
    # vyrobime si maticu 3x3 v ktorej su zaznacene nase stlpce
    matrix = [ [ col & 1<<row != 0 for col in [s0,s1,s2] ] for row in range(3) ]

    # skontrolujeme vsetkych 5 zvysnych trojic
    if all( matrix[0][i]   for i in range(3) ): return False
    if all( matrix[1][i]   for i in range(3) ): return False
    if all( matrix[2][i]   for i in range(3) ): return False
    if all( matrix[i][i]   for i in range(3) ): return False
    if all( matrix[i][2-i] for i in range(3) ): return False
    return True

memo = {}

def solve(c,s1,s2):
    # kolkymi sposobmi sa da vyplnit prvych c stlpcov, ak stlpce c+1 a c+2 obsahuju patterny s1 a s2?
    if c == 0: return 1
    key = (c,s1,s2)
    if key in memo: return memo[key]
    memo[key] = 0
    for s0 in range(7):
        if compatible(s0,s1,s2):
            memo[key] += solve(c-1,s0,s1)
    return memo[key]

def count(c):
    # kolkymi sposobmi sa da vyplnit prazdna plocha 3xc?
    return sum( solve(c-2,i,j) for i in range(7) for j in range(7) )

print( count( int(input()) ) )
```

Riešenie: Ukážkové zadanie 4

S týmto je trochu viac práce a je vhodné pomôcť si programom.
Z povahy problému je jasné, že počet pozícii bude od c závisieť exponenciálne. V praxi potom stačí spraviť program ako ten, ktorý sme videli v riešení predchádzajúcej úlohy. Keď ho spustíme s $c = 98, 99, 100$, uvidíme, že každá ďalšia výstupná hodnota je približne rovnakým konštantným násobkom predchádzajúcej.
My však teraz chceme korektné teoretické riešenie: nielen nájsť správnu konštantu, ale aj zdôvodniť, že je to naozaj ona.

Začneme tým, že si uvedomíme, že pre každý stĺpec zvlášť máme 7 možností ako ho vyplniť – nie 8, keďže nemôžeme dať kamienky naraz na všetky tri jeho políčka. Toto nám dáva horný odhad, že pozícií je nanajvýš 7^c.
Určite môžeme dokola ukladať kamienky tak, že vyrobíme dva ľubovoľné stĺpce a potom necháme jeden prázdny. Pozícií takéhoto typu je pre $c = 3k + 2$ presne 7^{2k}. Asymptoticky sme teda našli rádovo $7^{2c/3} \approx 3.6593^c$ platných pozícií. Ich celkový počet teda musí rásť aspoň takto rýchlo.

Počet pozícií teda zjavne rastie exponenciálne. Ako zistiť presnú hodnotu základu tejto exponenciálnej funkcie? Predstavme si, že pre konkrétne c spočítame všetky platné pozície. Tieto si môžeme rozdeliť do 7×7 vedierok podľa toho, ako v nich vyzerajú posledné dva stĺpce. Takže namiesto jedného počtu p_c všetkých pozícií máme teraz 49 počtov: $p_{c,0}, \ \ldots, \ p_{c,48}$.
Z počtov $p_{c,0}, \ \ldots, \ p_{c,48}$ vieme vypočítať počty $p_{c+1,0}, \ \ldots, \ p_{c+1,48}$. (Toto je to isté dynamické programovanie ktoré sme použili v predchádzajúcej úlohe, navyše zjednodušené o to, že nemáme vopred rozložené žiadne kamienky.)
Presnejšie, každé $p_{c+1,i}$ je súčtom niektorých $p_{c,j}$: tých, ktoré sú kompatibilné. Index j kóduje obsah stĺpcov $c-1$ a c, index i kóduje obsah stĺpcov c a $c+1$. Oba tieto indexy sa musia zhodovať na stĺpci c a navyše musí platiť, že nikde v týchto troch stĺpcoch nevznikne vodorovná ani šikmá trojica kamienkov.

Vyššie uvedené podmienky sa najľahšie skontrolujú programom. Výsledky skúšania si môžeme zapísať ako maticu A rozmerov 49×49, pre ktorú platí

$$\forall c \geq 2: \ (p_{c,0}, \ \ldots, \ p_{c,48}) \cdot A = (p_{c+1,0}, \ \ldots, \ p_{c+1,48})$$

Najväčšie vlastné číslo takto zostrojenej matice je približne 5.27093639, takže vieme, že počet pozícií určite nerastie rýchlejšie ako 5.27093639^c.
Navyše zistíme, že vlastný vektor $\vec{v}$ prislúchajúci k najväčšiemu vlastnému číslu má všetky zložky kladné reálne. Začiatočné podmienky pre našu rekurenciu sú $\forall i : p_{2,i} = 1$, takže počty platných pozícií pre c stĺpcov vypočítame ako $(1, \ldots, 1) \cdot A^{c-2}$. No a vektor $(1, \ldots, 1)$ má s vektorom $\vec{v}$ nenulový skalárny súčin, čiže nie sú na seba kolmé. Inými slovami, keď vektor začiatočných podmienok zapíšeme v báze tvorenej vlastnými vektormi, bude zjavne mať nenulovú zložku v smere $\vec{v}$. Preto veľkosť vektora $(1, \ldots, 1) \cdot A^{c-2}$ naozaj rastie s rastúcim c vyššie uvedenou rýchlosťou.

Počet platných pozícií na pláne rozmerov $3 \times c$ teda rastie rádovo rýchlosťou 5.27093639^c.

```
# pouzivame take iste kodovanie stlpcov a funkciu compatible(s0,s1,s2) ako minule

# zostrojime maticu A v ktorej na policku [i+7*j][j+7*k] bude 1 ak po stlpcoch i,j moze nasledovat stlpec k

A = [ [ 0 for j in range(49) ] for i in range(49) ]
for i in range(7):
    for j in range(7):
        for k in range(7):
            if compatible(i,j,k):
                A[i+7*j][j+7*k] = 1

import numpy
eigenvalues, eigenvectors = numpy.linalg.eig(A)
print( eigenvalues[0] )
print( list(eigenvectors[:,0]) )
```