Policists un zaglis

Baitimorā noziedzības līmenis ir sasniedzis visu laiku augstāko līmeni. Zādzības notiek katru dienu. Kad noziegums ir izdarīts, patrulējošajam policistam vienam pašam ir jāķer zaglis pa šaurajām ielām un krustojumiem, kas savieno ielas. Diemžēl zagļiem biežāk izdodas aizbēgt no policistiem, nekā tie tiek noķerti, jo zagļi pārzina pilsētu daudz labāk nekā policija.

Baitimoras pilsētas policijas departaments (BPPD) organizē sanāksmi, lai samazinātu noziedzību. Viena no iniciatīvām ir izmantot datoru zagļu ķeršanai. Šim nolūkam BPPD ir izveidojis precīzu pilsētas karti un tagad viņiem tikai nepieciešama programmatūra, kas atradīs ķeršanas stratēģijas.

Pakaļdzīšanās, kad viens policists ķer vienu zagli tiek modelēta šādā veidā:

1. Policists izvēlas krustojumu kurā patrulēt
2. Zaglis izvēlas krustojumu, kurā veikt zādzību (viņš zina policista atrašanās vietu). No šī brīža tiek pieņemts, ka gan policists gan zaglis zina otra atrašanās vietu.
3. Policista gājiena laikā viņš pārvietojas uz kādu no kaimiņu krustojumiem (t.i. krustojumiem kuros var nokļūt no pašreizejā krustojuma ejot pa ielu) vai ari gaida (t.i. nekur neiet).
4. Zagļa gājiena laikā viņš pārvietojas uz kādu no kaimiņu krustojumiem. Ievērjoiet, ka atškirībā no policista, zaglis nevar gaidīt, jo viņu instinkti liek tiem turpināt skriet.
5. Policists un zaglis pārmaiņus izdara gājienus (policists sāk pirmais), kamēr iestājas viens no šiem gadījumiem:
   (a) situācija atkārtojas (situācija tiek definēta kā policista un zagļa pozīcija un tas kuram nākamajam jāizdara gājiens). Ja situācija atkārtojas, tad zaglis var izvairities no policista bezgalīgi ilgi un tātad viņš aizbēg.
   (b) pēc policista vai zagļa gājiena abi nonāk vienā un tajā pašā krustojumā. Šajā gadījumā policists noķer zagli.

Uzdevums

Jums jāuzraksta programma, kas dotai pilsētas kartei noteiktu vai iespējams noķert zagli un ja tas ir iespējams, tad noķertu zagli, izdarot gajienus policista vietā.

Uzskatiet, ka zaglis izdara gājienus optimāli.

Implementācija

Jums jārealizē divas funkcijas:

- funkcija `start(N, A)`, ar parametriem:

- N — krustojumu skaits (krustojumi sanumurēti no 0 līdz $N-1$);
- A — divdimensionāls masīvs, kas apraksta ielas $0 \leq i, j \leq N-1$

$$A[i,j] \text{ ir } \begin{cases} \texttt{false} & \text{ja } i \text{ un } j \text{ nav savienoti ar ielu} \\ \texttt{true} & \text{ja } i \text{ un } j \text{ ir savienoti ar ielu} \end{cases}$$

Visas ielas ir divvirzienu (t.i. $A[i,j] = A[j,i]$ visiem i un j) un nav tādu ielu, kas sākas vienā un tajā pašā krustojumā (t.i. $A[i,i]$ būs `false` visiem i). Jūs varat pieņemt, ka no jebkura krustojuma, pārvietojoties pa ielām, būs iespējams nokļūt jebkurā citā krustojumā.

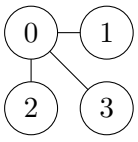
Ja parametros aprakstītajā kartē ir iespējams noķert zagli, tad funkcijai `start` jāatgriež tā krustojuma numurs uz kura policists izvēlas patrulēt. Citādi jāatgriež -1.

- funkcija `nextMove(R)`, kas parametrā saņem krustojuma numuru R, kurā pašlaik atrodas zaglis, un funkcijai jāatgriež tā krustojuma numuru, kurā atradīsies policists pēc sava gājiena izdarīšanas.

Funkcija `start` tiks izsaukta tieši vienu reizi pirms funkcijas `nextMove` izsaukšanas. Ja funkcija `start` atgriež -1, tad funkcija `nextMove` netiks izsaukta, bet citos gadījumos funkciju `nextMove` izsauks tikmēr kamēr pakaļdzīšanās beigsies. Precīzāk, programma pārtrauks darbu tiklīdz izpildīsies viens no sekojošiem nosacījumiem:

- funkcija `nextMove` atgriež nekorektu gājienu;
- situācija atkārtojas;
- zaglis tiek noķerts.

Piemērs



Aplūkosim labajā pusē ilustrēto piemēru. Šajā gadījumā policistam jebkurš no krustojumiem ir laba sākuma pozīcija. Ja policists sāk patrulēt krustojumā 0, viņš var gaidīt un zaglis viņam uzskries virsū. Taču, ja policists sāk patrulēt jebkurā citā krustojumā, tad viņš var gaidīt kamēr zaglis nonāk krustojumā 0 un tad doties uz turieni.

Šādi varētu izskatīties piemēra izpilde.

Funkcijas izsaukums	**Rezultāts**
`start(4, [[0, 1, 1, 1], [1, 0, 0, 0], [1, 0, 0, 0], [1, 0, 0, 0]])`	3
`nextMove(1)`	3
`nextMove(0)`	0

Piezīme: īsākam pierakstam iepriekš aprakstītajā funkcijas `start` izsaukumā `0` apzīmē `false` un `1` apzīmē `true`.

Vērtēšana

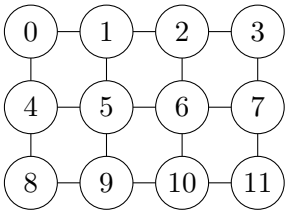
Lai iegūtu punktus Jūsu risinājumam jāspēj:

1. korekti noteikt vai policists var noķert zagli;
2. izpildot gājienus policista vietā noķert zagli.

Apakšuzdevumiem 3 un 4 risinājumi, kas izpilda tikai pirmo nosacījumu iegūs 30% no apakšuzdevuma punktiem.

Apakšuzdevums 1 (16 punkti): $2 \leq N \leq 500$. Starp katriem diviem krustojumiem ir tieši viens ceļš.

Apakšuzdevums 2 (14 punkti): $2 \leq N \leq 500$. Krustojumu un ielu tīkls veidos režģveida strutūru. Režģim būs vismaz divas rindas un kolonnas un krustojumu numerācija atbildīs zemāk norādītajam šablonam.



Apakšuzdevums 3 (30 punkti): $2 \leq N \leq 100$.

Apakšuzdevums 4 (40 punkti): $2 \leq N \leq 500$.

Ierobežojumi

Laika ierobežojums: 1.5 s.

Atmiņas ierobežojums: 256 MB.

Eksperimentēšana

Pārbaudes programma uz Jūsu datora lasīs ievaddatus no standarta ievada. Pirmajā ievaddatu rindā jābut naturālam skaitlim N — krustojumu skaitam. Nākamajās N rindām jāsatur kaimiņu matricu A, katrā no rindām jābūt N skaitļiem, kur katrs ir vai nu 0 vai 1. Matricai jābūt simetriskai un uz galvenās diagonāles jābūt nullēm.

Nākamajai rindai jāsatur skaitlis 1, ja policists var noķert zagli, bet 0 citos gadijumos.

Ja policists var noķert zagli, tad jābūt vēl N rindām, kas apraksta zagļa stratēģiju. Katrai no šīm N rindām jāsatur $N + 1$ vesels skaitlis ar vērtību no 0 līdz $N - 1$. r-tās rindas c-tās kolonnas, kur $c < N$, vērtība apraksta uz kurieni dosies zaglis, savā gājienā ja policists atrodas krustojumā r, bet zaglis — krustojumā c. Vērtības uz galvenās diagonāles tiks ignorētas, jo

tās atbilst situācijām, kad zaglis un policists ir vienā un tajā pašā krustojumā. Labējā kolonna norāda kurā vietā sāks zaglis atbilstoši katrai policista izvēlētajai sākuma pozīcijai.

Šeit redzams, kā pārbaudes programmai aprakstītu piemēru ar trim krustojumiem, kas savienoti savā starpā.

```
3
0 1 1
1 0 1
1 1 0
1
0 2 1 2
2 0 0 2
1 0 0 1
```

Šeit doti ievaddati, kas atbilst uzdevuma formulējumā dotajam piemēram.

```
4
0 1 1 1
1 0 0 0
1 0 0 0
1 0 0 0
1
0 0 0 0 1
2 0 0 0 2
3 0 0 0 3
1 0 0 0 1
```