Task C13. გვირაბები

⏳ 1.25 s 💾 1024 MB

კურდღელმა გათხარა გვირაბების სისტემა, რომელიც შედგება N დონისგან (გადანომრილია ზემოდან ქვემოთ: 0-დან $N-1$-მდე). თითოეულ დონეზე არის M ცალი ოთახი (გადანომრილია მარცხნიდან მარჯვნივ: 0-დან $M-1$-მდე). ოთახები შეგვიძლია წარმოვიდგინოთ როგორც უჯრები $N \times M$ ცხრილში. ყველა მეზობელ (რომელთაც აქვთ საერთო გვერდი) ოთახს შორის არსებობს გვირაბი. ყველაზე ქვემოთ მდებარეობს დიდი საგანძურის ოთახი (რომელიც ფარავს მთელ მე-N-ე დონეს). საგანძურის ოთახსა და ყველა იმ ოთახს შორის, რომლებიც $N-1$-ე დონეზეა, არსებობს გვირაბები. თუმცა, NM ოთახიდან ზოგიერთი ჩამონგრეულია და შესაბამისად მიუწვდომელია (მათთან დაკავშირებული ყველა გვირაბიც ასევე მიუწვდომელია).

კურდღელი თავისი საგანძურის დაცვას ცდილობს, ამიტომ თითქმის ყველა ვერტიკალურ გვირაბში ხაფანგი აქვს დამონტაჟებული. უფრო კონკრეტულად, თითოეულ დონეზე (მათ შორის საგანძურის დონეზეც) არსებობს ზუსტად ერთი უსაფრთხო (არახაფანგიანი) შემომავალი ვერტიკალური გვირაბი. გარანტირებულია, რომ არსებობს უსაფრთხო ბილიკი (ბილიკი, რომელიც არ გადის ხაფანგიან გვირაბებზე) ზედაპირიდან საგანძურის ოთახამდე.

ალისას სურს, კურდღლის საგანძურის ოთახში მოხვდეს, თუმცა ბრმად გვირაბების სისტემაში შესვლა უკიდურესად სახიფათოა. პირველ რიგში, მან ჩაატარა ოთახების კვლევა ხმოვანი ტალღებით და დაადგინა, რომელი ოთახები იყო ჩამონგრეული. თუმცა, მან ჯერ კიდევ არ იცის, რომელი გვირაბებია ხაფანგიანი და რომელი – უსაფრთხო. შემდეგ ალისამ კურდღლის დაკვირვება დაიწყო. მან იცის, რომ კურდღელი მუდამ ჩქარობს, ამიტომ ის ყოველთვის იყენებს ყველაზე მოკლე უსაფრთხო გზას (ბილიკს, რომელიც არ გადის ხაფანგიან გვირაბებზე) ზედაპირიდან საგანძურამდე (და პირიქით), ანუ იმ უნიკალურ ბილიკს, რომელშიც ოთახები არ მეორდება. ასევე, ალისამ დაინახა, რომ კურდღელი გვირაბების სისტემაში შედის ზედაპირიდან, სწორედ იმ ოთახის ზემოდან, რომელიც K ნომრით არის დანომრილი პირველ (ანუ 0) დონეზე ($0 \le K < M$). ეს აძლევს ალისას საშუალებას, უსაფრთხოდ შევიდეს გვირაბების სისტემაში. ბოლოს, ალისამ თან წამოიყვანა ძაღლი, რომელიც კურდღლის სუნს ცნობს. ძაღლის დახმარებით მას შეუძლია მივიდეს მიღწევად ოთახში (იმ დონეზე, რომელზეც ამჟამად იმყოფება) და შეამოწმოს – არის თუ არა კურდღლის კვალი იმ ოთახში. რამდენიმე ასეთი გამოკვლევის შემდეგ, როდესაც აბსოლუტურად დარწმუნებულია, რომ კონკრეტული ვერტიკალური გვირაბი უსაფრთხოა, ალისა შედის მისაწვდომ ოთახში და ქვემოთ ჩადის – გადადის შემდეგ დონეზე. ის არ აპირებს სიცოცხლის გარისკვას, ამიტომ უნდა იყოს სრულიად დარწმუნებული, რომ არჩეული გვირაბი უსაფრთხოა. ალისას უნდა, რომ საგანძურამდე უსაფრთხოდ მიაღწიოს და ამისთვის რაც შეიძლება ცოტა გამოკვლევა დასჭირდეს ყველაზე ცუდ შემთხვევაშიც კი (ისიც ჩქარობს – ჩაის წვეულებაზე უნდა დასწრება).

ფორმალურად, მოცემული ტესტქეისისთვის (N, M, K და დაბლოკილი ოთაები) ყველაზე ცუდ შემთხვევაში საჭირო გამოძიებების რაოდენობა განვსაზღვროთ შემდეგნაირად: განვიხილოთ უსაფრთხო ვერტიკალური გვირაბების ყველა შესაძლო განლაგება, თითოეულ მათგანზე გამოვთვალოთ კურდღლის ბილიკი და თითოეულ შემთხვევაში ცალ-ცალკე გავუშვათ ალისას სტრატეგია. სტრატეგიისთვის ყველაზე ცუდ შემთხვევაში გამოძიებების რაოდენობა არის მაქსიმალური გამოძიებების რაოდენობა, რომელსაც ის აკეთებს ამ შემთხვევებიდან რომელიმეზე (მაქსიმუმი იმ შემთხვევებს შორის, სადაც სტრატეგია ვალიდურია და ალისის შეუძლია უსაფრთხოდ

მიაღწიოს საგანძურამდე; წინააღმდეგ შემთხვევაში, ვამბობთ, რომ მისი უარეს შემთხვევაზე გამოძიებების რაოდენობა უსასრულოა). შემდეგ განვიხილავთ ყველა შესაძლო სტრატეგიას და ვპოულობთ იმ მინიმალურ მაქსიმუმს, რაც შეიძლება მიიღოს რომელიმე მათგანმა. ეს იქნება ტესტქეისისთვის დაშვებული გამოძიებების მაქსიმალური რაოდენობა.

ალისა ჭკვიანი გოგოა, მაგრამ კურდღლის გვირაბების სისტემა ძალიან დიდია, ამიტომ ოპტიმალური სტრატეგიის მოფიქრება ვერ შეძლო. დაეხმარე მას ასეთი პროგრამის დაწერით, რომელიც გამოიკვლევს გვირაბებს და მიაღწევს საგანძურს ისე, რომ არ დაარღვიოს გამოძიებების დაშვებული ზღვარი, არ შევიდეს დაბლოკილ ოთახებში და არ გადალახოს ხაფანგიანი გვირაბები. (თქვენმა პროგრამამ ყველა ეს პირობა უნდა დააკმაყოფილოს, რომ ტესტქეისი ჩაითვალოს სწორად შესრულებულად.)

იმპლემენტაციის დეტალები

თქვენ უნდა დაწეროთ ფუნქცია `solve`:

```
void solve(int n, int m, int k, const std::vector<std::vector<bool>>& blocked)
```

ფუნქციას თითოეული ტესტქეისი გამოიძახებს ერთხელ N, M და K-თი და ოთახების აღწერით (გადანომრილია როგორც `blocked[level][position]`). მან უნდა გამოიკვლიოს გვირაბების სისტემა (იწყებს მე-0 დონეზე, მე-K ოთახში) და მიაღწიოს საგანძურის დონემდე (მე-N დონემდე) ისე, რომ გამოძიებების რაოდენობამ არ გადააჭარბოს ტესტქეისისთვის მინიმალურ შესაძლო ყველაზე ცუდი შემთხვევის გამოძიებების რაოდენობას. ამ ფუნქციიდან (და სხვა ფუნქციებიდან, რომლებსაც თავად დაწერთ) შეგიძლიათ გამოძახოთ შემდეგი ფუნქციები: `investigate` და `goDeeper`.

```
bool investigate(int s)
```

`investigate`-ის გამოძახება ნიშნავს გადასვლას ამჟამინდელ დონეზე მდებარე მე-S ოთახში (ის უნდა იყოს მიღწევადი, ანუ გზაში არ უნდა იყოს ჩანგრეული ოთახები) და შემოწმებას, გაივლის თუ არა კურდღელი ამ ოთახში საგანძურამდე მისვლისას. ფუნქცია აბრუნებს `true`-ს, თუ ის მართლაც გაივლის, და `false`-ს – თუ არა.

```
bool goDeeper(int s)
```

`goDeeper`-ის გამოძახება ნიშნავს გადასვლას ამჟამინდელ დონეზე მდებარე მე-S ოთახში (იგი უნდა იყოს მიღწევადი) და შემდეგ იქიდან ჩამოსვლას ქვედა დონეზე ვერტიკალური გვირაბის საშუალებით. გვირაბი აუცილებლად უნდა იყოს უსაფრთხო (ანუ არახაფანგიანი), ხოლო ქვედა დონეზე არსებული ოთახი არ უნდა იყოს ჩანგრეული. ამის შემდეგ, განაგრძობთ გამოძიებას ქვედა დონის მე-S ოთახიდან (გარდა იმ შემთხვევისა, როცა უკვე მიაღწიეთ მე-N დონეს, როდესაც თქვენი `solve` ფუნქცია უნდა დასრულდეს და დააბრუნოს შედეგი).

თქვენი კოდი კომპილირდება შემფასებელთან ერთად, ამიტომ არ უნდა მოიცავდეს `main` ფუნქციას, და ასევე არ უნდა წაიკითხოთ `stdin`-დან ან გამობეჭდოთ `stdout`-ით. ასევე საჭიროა `tunnels.h` ფაილის ჩართვა.

შეფასებელი ზოგჯერ შეიძლება იყოს ადაპტური, (რაც ნიშნავს, რომ ის შეიძლება განსაზღვრავდეს `investigate`-ის პასუხებს ან `goDeeper`-ის წარმატებას თქვენი კოდის წარსული ქმედებების მიხედვით, ფიქსირებული უსაფრთხო გვირაბების გარეშე. თუმცა გარანტირებულია, რომ შემფასებლის ყველა ქმედება თავსებადია უსაფრთხო გვირაბების რომელიმე შესაძლო ვერსიასთან. შემფასებელს შეუძლია შეაჩეროს პროგრამის მუშაობა, თუ თქვენ `goDeeper` გამოიძახეთ და ცდილობთ ხაფანგიან გვირაბში ჩასვლას, ან თუ ის გადაწყვეტს, რომ თქვენი ამოცანის გადაჭრა მოთხოვნილ გამოძიებათა რაოდენობით აღარ არის შესაძლებელი. შემფასებლის გასაშვები დრო არ ითვლება დროის ლიმიტში.

ლოკალური ტესტირება

ლოკალურად პროგრამის შესამოწმებლად გეძლევათ ლოკალური შემფასებელი და შესაბამისი ჰედერ ფაილი. ლოკალური შემფასებელი არ არის ადაპტური და არ ახორციელებს გამოთვლებს ან შემოწმებებს. ის კითხულობს N, M, K და 0-ებისა და 1-ების მატრიცას (ჰარების გარეშე), იძახებს თქვენს `solve` ფუნქციას და შემდეგ ბეჭდავს თქვენს მიერ გამოძახებულ `investigate` ან `goDeeper` მოქმედებებს (ყოველ

`investigate`-ის გამოძახებაზე უნდა შეიყვანოთ პასუხი – 0 ან 1, რასაც დააბრუნებს ფუნქცია. ლოკალური შემფასებლის პროგრამის შეცვლა შეგიძლიათ.

შეზღუდვები

- $1 \leq N, M \leq 5000$

ქვეამოცანები

ქვეამოცანა	**ქულები**	**შეზღუდვები**	
1	5	$M = 2$	
2	21	არცერთი ოთახი დაბლოკილი არ არის.	3
18	$N, M \leq 100$		
4	18	$N, M \leq 400$	
5	19	$N, M \leq 1000$	
6	19	სხვა არაფერი.	

ქვეამოცანისთვის ქულას იღებთ მხოლოდ იმ შემთხვევაში, თუ თქვენი პროგრამა სწორად გაივლის ამ ქვეამოცანის ყველა ტესტს და ყველა საჭირო წინა ქვეამოცანას.

მაგალითი

შეტანა	**ინტერაქცია**
2 3 1 001 100	solve(2, 3, 1, {{0, 0, 1}, {1, 0, 0}}) goDeeper(1) investigate(2): return true goDeeper(2)

ამ ტესტისთვის მინიმალური შესაძლო ყველაზე ცუდი შემთხვევის გამოძიებების რაოდენობა არის 1, შესაბამისად მოცემული ამოხსნა ამ ტესტს გაივლიდა.