

### Task A11. Tunnels

 1.25 s  1024 MB

Заекът е изкопал система от тунели, която има  $N$  нива (номерирани отгоре надолу: от 0 до  $N - 1$ ). Всяко ниво има  $M$  камери (номерирани отляво надясно: от 0 до  $M - 1$ ). Можем да си представим камерите като клетки на мрежа  $N \times M$ . Има тунели между всички двойки съседни камери (споделящи една страна). Под всичко това има голяма съкровищница (която ефективно обхваща цялото ниво  $N$ ). Има тунели между нея и всички камери на ниво  $N - 1$ . Някои от  $NM$  камерите обаче са се срутили и по този начин са блокирани (както и всички техни тунели).

Заекът иска да защити съкровището си, затова е поставил капани в почти всички вертикални тунели. По-точно, всяко ниво (включително нивото на съкровищницата) има точно един безопасен (незатворен) входящ вертикален тунел. Освен това е гарантирано, че има безопасен път (път, минаващ през тунели без капани) от повърхността до съкровищницата.

Алиса иска да стигне до стаята със съкровищата на Заека, но влизането на сляпо в тунелната система би било изключително опасно. Първо, тя проучила камерите с помощта на сонар и разбрала кои са блокирани. Въпреки това, тя все още няма представа кои тунели са затворени и кои са безопасни. След това тя започнала да наблюдава Заека. Тя знае, че той винаги бърза, така че винаги избира най-краткия безопасен (без затворени тунели) път от повърхността до съкровището (и обратно), т.е. уникалния безопасен път, който не повтаря камерите. Тя също го е виждала да влиза в тунелната система (а именно ниво 0) от повърхността над камера  $K$  ( $0 \leq K < M$ ). Това ѝ позволява безопасно да влезе в тунелната система. Накрая Алиса довела хрътка, която познава миризмата на Заека. С помощта на кучето тя може да отиде до достижима камера (на текущото си ниво) и да я изследва – да провери дали Заекът е бил там или не. След серия от тези разследвания, когато е сигурна, тя може да отиде до достижима камера и да се спусне по-дълбоко (до следващото ниво) оттам. Тя не иска да рискува живота си, затова трябва да е абсолютно сигурна, че избраният вертикален тунел е безопасен. Иска да стигне до съкровището безопасно, използвайки възможно най-малко разследвания в най-лошия случай (също така бърза, тъй като трябва да присъства на чаено парти).

Формално, ние дефинираме най-лошия брой разследвания на дадена стратегия за изследване за даден тестов случай ( $N$ ,  $M$ ,  $K$  и набор от блокирани камери) както следва: Изброяват се всички възможни набори от безопасни тунели, изчислява се пътя на Заека за всеки от тях и се изпълнява стратегията за всеки от тези случаи (независимо). Най-лошият брой разследвания на стратегията е максималният брой разследвания, които тя извършва за всеки един от тези случаи (ако приемем, че е валидна и успява безопасно да стигне до съкровището във всички тях; в противен случай казваме, че най-лошият брой разследвания е безкрайност). След това изброяваме всички възможни стратегии и намираме минималния възможен най-лош брой разследвания. Това е максималният разрешен брой разследвания за тестовия случай.

Алиса е умно момиче, но тунелната система на Заека е огромна, така че дори тя не може да измисли оптимална стратегия за изследване. Помогнете ѝ, като напишете програма, която изследва тунелната система и достига до

съкровищницата, използвайки не повече от разрешения брой разследвания за тестовия случай, без да преминава през блокирани камери или през затворени тунели. (Вашата програма трябва да отговаря на всички тези изисквания, за да премине правилно даден тестов случай.)

### Подробности по имплементацията

Трябва да имплементирате функцията `solve`:

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

Ще бъде извикана веднъж за всеки тестов случай с  $N$ ,  $M$  и  $K$ , както и описание на камерите (индексирани като `blocked[level][position]`). Трябва да извърши изследването на тунелите (започвайки от ниво 0, камера  $K$ ), за да достигне нивото на съкровищницата (ниво  $N$ ), използвайки не повече от минималния възможен най-лош брой разследвания за тестовия случай. От тази функция (и други функции, които пишете), можете да извикате функциите `investigate` и `goDeeper`:

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

Извикването на `investigate` представлява отиване до камера  $S$  на текущото ниво (тя трябва да е достижима, т.е. не трябва да има блокирани камери по пътя) и проверка дали Заекът преминава през тази камера по пътя си към съкровището. Функцията връща `true`, ако го направи, и `false`, в противен случай.

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

Извикването на `goDeeper` представлява отиване до камера  $S$  на текущото ниво (тя трябва да е достижима) и след това слизване до следващото ниво, използвайки вертикалния му тунел. Тунелът трябва да е безопасен (т.е. да не е затворен) и камерата отдолу не трябва да е блокирана. След това продължавате изследването си от камера  $S$  на следващото ниво (освен ако не сте достигнали ниво  $N$ , в който случай вашата функция `solve` трябва да върне резултат).

Вашият код ще бъде компилиран заедно с грейдър, така че не трябва да имплементира `main` функция, нито трябва да четете от `stdin` или да пишете в `stdout`. Също така трябва да включите заглавния файл `tunnels.h`.

Грейдърът понякога може да бъде адаптивен (но все пак детерминистичен), т.е. може да реши какво връща `investigate` или дали `goDeeper` е успешен, въз основа на миналите действия на вашата програма, без да има фиксиран набор от безопасни тунели. Гарантирано е обаче, че всичко, което грейдерът прави, ще бъде валидно по отношение на някакъв възможен набор от безопасни тунели. Грейдерът може да прекрати изпълнението, ако извикате `goDeeper` и се опитате да преминете през затворен тунел или ако реши, че вече не е възможно да решите тестовия случай, използвайки необходимия брой разследвания. Освен това, времето за изпълнение на грейдера не се брои към времевия лимит.

### Локално тестване

За да тествате програмата си локално, са предоставени локален грейдър и заглавен файл. Локалният грейдър не е адаптивен и не извършва изчисления или проверки. Той чете  $N$ ,  $M$ ,  $K$  и матрица от 0 и 1 (без интервали), извиква вашия `solve` и след това извежда резултат всеки път, когато програмата ви извика `investigate` или `goDeeper` (очаква вход за всеки `investigate` - 0 или 1, които ще върне). Вие сте свободни да промените локалния грейдър.

### Ограничения

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

### Подзадачи

Подзадача	Точки	Ограничения
1	5	$M = 2$
2	16	Няма блокирани камери.
3	19	$N, M \leq 100$
4	19	$N, M \leq 400$
5	20	$N, M \leq 1000$
6	21	Няма.

Получавате точки за дадена подзадача, само ако успешно изпълните всички тестове в нея и всички останали подзадачи, които са включени в нея.

### Примерен тест

Вход	Взаимодействие
2 3 1 001 100	solve(2, 3, 1, {{0, 0, 1}, {1, 0, 0}}) goDeeper(1) investigate(2): return true goDeeper(2)

Минималният възможен брой разследвания в най-лошия случай за този тест е 1, така че решението би преминало този тест.