

### Задача A11. Тоннели

 1.25 с  1024 МБ

Кролик выкопал систему тоннелей, которая состоит из  $N$  уровней (пронумерованных сверху вниз от 0 до  $N - 1$ ). Каждый уровень содержит  $M$  комнат (пронумерованных слева направо от 0 до  $M - 1$ ). Мы можем представить комнаты как ячейки сетки размером  $N$  на  $M$ . Любые две соседние (имеющие общую сторону) комнаты соединены тоннелем. Под всей этой системой расположена большая комната с сокровищами (фактически занимающая еще один следующий уровень с номером  $N$ ). Существуют тоннели между ней и всеми комнатами на уровне  $N - 1$ . Однако некоторые из  $NM$  комнат обрушились и теперь заблокированы (а также заблокированы все их тоннели).

Кролик хочет защитить свои сокровища, поэтому он заминировал почти все вертикальные тоннели. Точнее, на каждом уровне (включая уровень с сокровищами) есть ровно один безопасный (незаминированный) вертикальный тоннель с уровня над ним. Также гарантируется, что существует безопасный путь (путь, не проходящий через заминированные тоннели) с поверхности до комнаты с сокровищами.

Алиса хочет добраться до комнаты с сокровищами Кролика, но заходить в систему тоннелей вслепую слишком опасно. Сначала она изучила комнаты с помощью сонара и выяснила, какие из них заблокированы. Однако она всё ещё не знает, какие тоннели заминированы, а какие безопасны. Затем она начала наблюдать за Кроликом. Она знает, что он всегда торопится, поэтому всегда выбирает кратчайший безопасный путь (без заминированных тоннелей) от поверхности к сокровищам (и обратно), то есть единственный безопасный путь, не проходящий через одну комнату дважды. Она также видела, как он входит в систему тоннелей (а именно, спускается на уровень 0) с поверхности над комнатой  $K$  ( $0 \leq K < M$ ). Это позволяет и ей безопасно войти в систему тоннелей. Наконец, Алиса привела с собой ищейку, которая знает запах Кролика. Она может дойти до достижимой комнаты (на текущем уровне) и с помощью собаки исследовать её — проверить, бывал ли там Кролик. После серии таких исследований, когда она будет уверена, что нашла незаминированный тоннель, она сможет спуститься в комнату на следующем уровне. Она не хочет рисковать жизнью, поэтому должна быть абсолютно уверена, что выбранный вертикальный тоннель безопасен. Она хочет добраться до сокровищ безопасно, используя как можно меньше исследований в наихудшем случае (она тоже торопится, ведь ей нужно успеть на чаепитие).

Формально, мы определяем худший случай числа исследований для стратегии исследования на заданном тесте ( $N$ ,  $M$ ,  $K$  и множество заблокированных комнат) так: перечисляем все возможные множества безопасных тоннелей, вычисляем путь Кролика в каждом из них и выполняем стратегию для каждого из этих случаев (независимо). Худший случай для стратегии — это максимум исследований, произведённых в каком-либо из этих случаев (при условии, что стратегия допустима и достигает комнаты с сокровищами во всех случаях; иначе — считается, что в худшем случае число исследований бесконечно). Затем мы перебираем все возможные стратегии и находим минимальное возможное худшее число исследований. Это и есть максимальное допустимое количество исследований для данного теста.

## XVI МЕЖДУНАРОДНЫЙ ПРОДВИНУТЫЙ ТУРНИР ПО ИНФОРМАТИКЕ ШУМЕН 2025

Алиса умная девочка, но система тоннелей Кролика огромна, и даже она не может придумать оптимальную стратегию. Помогите ей, написав программу, которая исследует систему тоннелей и добирается до комнаты с сокровищами, не превышая допустимое количество исследований для теста, не проходя через заблокированные комнаты или заминированные тоннели. (Ваша программа должна выполнять все эти требования, чтобы пройти тест.)

### Детали реализации

Вам нужно реализовать функцию `solve`:

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

Она будет вызываться один раз для каждого теста с  $N$ ,  $M$ ,  $K$  и описанием комнат (индексация: `blocked[level][position]`). Она должна выполнять исследование системы тоннелей (начиная с уровня 0, комнаты  $K$ ) и добраться до уровня с сокровищами (уровень  $N$ ), не превышая минимально возможное худшее число исследований. Из этой функции (и других написанных вами функций) можно вызывать функции `investigate` и `goDeeper`:

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

Вызов `investigate` означает, что вы идёте в комнату  $S$  на текущем уровне (она должна быть достижима, т.е. не должно быть заблокированных комнат на пути) и проверяете, проходил ли через неё Кролик. Функция возвращает `true`, если проходил, иначе — `false`.

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

Вызов `goDeeper` означает, что вы идёте в комнату  $S$  на текущем уровне (она должна быть достижима), а затем спускаетесь вниз на следующий уровень через вертикальный тоннель. Этот тоннель должен быть безопасным (не заминированным), и комната ниже не должна быть заблокированной. После этого вы продолжаете исследование с комнаты  $S$  следующего уровня (если вы не достигли уровня  $N$ , в этом случае функция `solve` должна завершиться).

Ваш код будет компилироваться с грейдером, поэтому он не должен реализовывать функцию `main`, не должен читать из `stdin` и писать в `stdout`. Также необходимо подключить заголовочный файл `tunnels.h`.

Грейдер может быть адаптивным (но детерминированным), то есть может решать, что возвращает `investigate` и успешен ли `goDeeper`, основываясь на действиях вашей программы, без фиксированного множества безопасных тоннелей. Однако гарантируется, что всё поведение грейдера будет допустимым для некоторого возможного множества безопасных тоннелей. Грейдер может завершить выполнение, если вы попытаетесь вызвать `goDeeper` через заминированный тоннель или если он определит, что выполнить тест с необходимым числом исследований более невозможно. Время работы грейдера не учитывается в ограничение по времени.

## XVI МЕЖДУНАРОДНЫЙ ПРОДВИНУТЫЙ ТУРНИР ПО ИНФОРМАТИКЕ ШУМЕН 2025

### Локальное тестирование

Для локального тестирования предоставлены локальный грейдер и заголовочный файл. Локальный грейдер не является адаптивным и не производит вычислений или проверок. Он считывает  $N$ ,  $M$ ,  $K$  и матрицу из 0 и 1 (без пробелов), вызывает вашу функцию `solve` и выводит действия вашей программы при вызовах `investigate` и `goDeeper` (ожидая ввод для каждого `investigate` — 0 или 1, который он вернёт). Вы можете свободно модифицировать локальный грейдер.

### Ограничения

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

### Подзадачи

Подзадача	Баллы	Ограничение
1	5	$M = 2$
2	21	Нет заблокированных комнат.
3	18	$N, M \leq 100$
4	18	$N, M \leq 400$
5	19	$N, M \leq 1000$
6	19	Без дополнительных ограничений.

Баллы за подзадачу начисляются, только если успешно пройдены все тесты в ней и во всех включенных в неё подзадачах.

### Пример теста

Ввод	Взаимодействие
2 3 1 001 100	<code>solve(2, 3, 1, {{0, 0, 1}, {1, 0, 0}})</code> <code>goDeeper(1)</code> <code>investigate(2): return true</code> <code>goDeeper(2)</code>

Минимально возможное худшее число исследований в этом тесте — 1, поэтому решение его проходит.