Mars

Хорошо известно, что фараоны первыми вышли в открытый космос. Они запустили первый космический корабль, который приземлился на планете Тутмус 1 (сейчас эта планета известна как Марс). Поверхность планеты представляет собой клетчатое поле размером $(2n+1) \times (2n+1)$, каждая клетка которого представляет собой либо сушу, либо воду. Состояние клетки в i-м ряду и j-м столбце ($0 \le i, j \le 2 \cdot n$) задается значением $s[i][j] =$'1', если это клетка суши, либо $s[i][j] =$'0', если это клетка воды.

Две клетки суши считаются связанными, если есть путь между ними по клеткам суши, причем любые две соседние клетки на пути имеют общую сторону. Остров на планете — это максимальное по включению множество клеток суши, любые две из которых связаны.

Миссия космического корабля заключается в том, чтобы подсчитать количество островов на планете. К сожалению, задача оказалась труднее чем обычно из-за древнего компьютера, установленного на космическом корабле. Память компьютера представляет собой двумерный массив h размера $(2n+1) \times (2n+1)$, каждая ячейка которого может содержать строку из `'0'` и `'1'` длиной ровно 100. Исходно память заполнена строками, где символ с индексом 0 содержит тип клетки, то есть $h[i][j][0] = s[i][j]$ (для всех $0 \le i, j \le 2 \cdot n$), а остальные символы равны `'0'`.

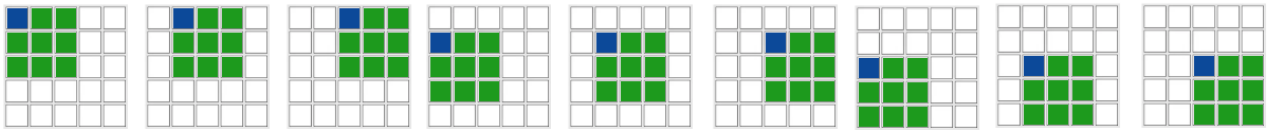
Для обработки данных в памяти компьютер может обращаться к участкам памяти размером 3×3, перезаписывая значение в левой верхней ячейке этого участка. Формально, компьютер может обратиться к участку памяти $h[i..i+2][j..j+2]$ ($0 \le i, j \le 2 \cdot (n-1)$) и перезаписать значение $h[i][j]$. Будем называть это действие **обработкой ячейки** (i, j).

Чтобы учесть ограничения компьютера, фараоны приняли решение действовать следующим образом:

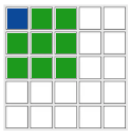
- Компьютер будет обрабатывать ячейки в n фаз.
- На фазе k ($0 \le k \le n-1$), обозначив $m = 2 \cdot (n-k-1)$, компьютер обработает ячейку (i, j) для всех $0 \le i, j \le m$, в возрастающем порядке значения i, а для каждого i, в возрастающем порядке значения j. Другими словами, компьютер обработает ячейки в следующем порядке: $(0,0), (0,1), \cdots, (0,m), (1,0), (1,1), \cdots, (1,m), \cdots, (m,0), (m,1), \cdots, (m,m)$.
- На последней фазе ($k = n-1$), компьютер обработает только ячейку $(0, 0)$. После этого действия значение в ячейке $h[0][0]$ должно быть равно двоичной записи числа островов на планете, при этом младший бит должен соответствовать символу строки с индексом 0.

Рисунок ниже показывает, как компьютер будет работать, если память имеет размер 5×5 ($n = 2$). Синим цветом показана ячейка, которую компьютер обрабатывает и заменяет в ней значение, цветные ячейки показывают участок памяти, к которому обращается компьютер.

На фазе 0 компьютер обработает ячейки в следующем порядке:



На фазе 1 компьютер обратится только к одному участку памяти:



Ваша задача состоит в том, чтобы реализовать метод, который позволит вычислить количество островов на планете Тутмус 1, учитывая то, как действует компьютер.

Implementation details

Вам следует реализовать следующую функцию

```
string process(string[][] a, int i, int j, int k, int n)
```

- a: массив размера 3×3, задающий фрагмент памяти $a = h[i..i+2][j..j+2]$. Каждый элемент a представляет собой строку длины 100, каждый символ которой равен или `'0'` (ASCII 48) или `'1'` (ASCII 49).
- i, j: ряд и столбец ячейки памяти, которую обрабатывает компьютер.
- k: текущий номер фазы.
- n: общее число фаз, а также значение, от которого зависит размер планеты, она состоит из $(2n+1) \times (2n+1)$ ячеек.
- Функция должна вернуть строку длины 100. Возвращенное значение будет сохранено в памяти компьютера в ячейке $h[i][j]$.
- Последний вызов этой функции будет для $k = n - 1$. Во результате этого вызова функция должна вернуть строку, которая содержит двоичное представление числа островов на планете. При этом младший бит должен располагаться в символе с индексом 0, второй младший бит в символе с индексом 1, и так далее.
- Функция не должна использовать статические или глобальные переменные и ее значение должно зависеть только от переданных ей параметров.

В каждом тесте ваше решение будет параллельно решать задачу для T независимых сценариев (то есть различных конфигураций планеты). Поведение вашего решения в каждом сценарии не должно зависеть от других сценариев и взаимного порядка вызова функции `process` для разных сценариев. Но для одного сценария гарантируется, что вызовы функции `process` будут выполняться в том порядке, в котором описано в условии задачи.

Кроме того, для каждого теста тестирующая система может запустить несколько экземпляров вашего решения и вызывать функцию `process` для одного или разных сценариев у разных запущенных экземпляров. Ограничения по памяти и времени для параллельно запущенных решений будут

суммироваться. Любая явная попытка обменяться информацией между запущенными экземплярами решения может привести к вашей дисквалификации!

В частности, любая информация, сохраненная в статических или глобальных переменных во время вызова `process`, может быть недоступна во время последующих вызовов этой функции.

Constraints

- $1 \leq T \leq 10$
- $1 \leq n \leq 17$
- $s[i][j]$ равен либо `'0'`, либо `'1'` (для всех $0 \leq i, j \leq 2 \cdot n$)
- Длина строки $h[i][j]$ равна 100 (для всех $0 \leq i, j \leq 2 \cdot n$)
- Каждый символ $h[i][j]$ равен либо `'0'`, либо `'1'` (для всех $0 \leq i, j \leq 2 \cdot n$)

Для каждого вызова функции `process`:

- $0 \leq k \leq n - 1$
- $0 \leq i, j \leq 2 \cdot (n - k - 1)$

Subtasks

1. (6 баллов) $n \leq 2$
2. (8 баллов) $n \leq 4$
3. (7 баллов) $n \leq 6$
4. (8 баллов) $n \leq 8$
5. (7 баллов) $n \leq 10$
6. (8 баллов) $n \leq 12$
7. (10 баллов) $n \leq 14$
8. (24 балла) $n \leq 16$
9. (11 баллов) $n \leq 18$
10. (11 баллов) $n \leq 20$

Examples

Example 1

Пусть $n = 1$ и s имеет следующий вид:

```
'1' '0' '0'
'1' '1' '0'
'0' '0' '1'
```

В этом примере размер планеты 3×3, на ней 2 островка. Будет только 1 фаза запуска функции `process`.

Во время фазы 0, грейдер вызовет функцию `process` ровно один раз:

```
process([["100","000","000"],["100","100","000"],["000","000","100"]],0,0,0,1)
```

Для примера мы привели только три первых символа каждого элемента h, при реальном вызове все строки будут иметь длину 100.

Функция должна вернуть `"0100..."` (опущены символы, равные `'0'`), где $...0010$ это двоичная записиь числа 2. Обратите внимание, что `...` в данном случае заменяет 96 нулей.

Example 2

Рассмотрим случай с $n = 2$ и массивом s следующего вида:

```
'1' '1' '0' '1' '1'
'1' '1' '0' '0' '0'
'1' '0' '1' '1' '1'
'0' '1' '0' '0' '0'
'0' '1' '1' '1' '1'
```

В этом примере планета имеет размер 5×5 и содержит 4 острова. Будет 2 фазы вызовов функции `process`.

Во время фазы 0 грейдер вызовет функцию `process` 9 раз:

```
process([["100","100","000"],["100","100","000"],["100","000","100"]],0,0,0,2)
process([["100","000","100"],["100","000","000"],["000","100","100"]],0,1,0,2)
process([["000","100","100"],["000","000","000"],["100","100","100"]],0,2,0,2)
process([["100","100","000"],["100","000","100"],["000","100","000"]],1,0,0,2)
process([["100","000","000"],["000","100","100"],["100","000","000"]],1,1,0,2)
process([["000","000","000"],["100","100","100"],["000","000","000"]],1,2,0,2)
process([["100","000","100"],["000","100","000"],["000","100","100"]],2,0,0,2)
process([["000","100","100"],["100","000","000"],["100","100","100"]],2,1,0,2)
process([["100","100","100"],["000","000","000"],["100","100","100"]],2,2,0,2)
```

Пусть, например, указанные вызовы вернули значения `"011", "000", "000", "111", "111", "011", "110", "010", "111"`, соответственно (везде мы опустили по 97 нулей в конце каждой строки). Тогда после окончания фазы 0 массив h содержит следующие значения:

```
"011", "000", "000", "100", "100"
"111", "111", "011", "000", "000"
"110", "010", "111", "100", "100"
"000", "100", "000", "000", "000"
"000", "100", "100", "100", "100"
```

Во время фазы 1, грейдер вызовет `process` один раз:

```
process([["011","000","000"],["111","111","011"],["110","010","111"]],0,0,1,2)
```

Этот вызов функции должен вернуть `"0010000...."` (все опущенные символы равны `'0'`), где ....0000100 представляет собой двоичную запись числа 4.

Sample grader

Пример грейдера читает данные в следующем формате:

- строка 1: T
- блок i ($0 \le i \le T-1$): блок, задающий сценарий i.
  - строка 1: n
  - строка $2+j$ ($0 \le j \le 2 \cdot n$): $s[j][0]$ $s[j][1]$ $\ldots$ $s[j][2 \cdot n]$

Пример грейдера выводит результат в следующем формате:

- строка $1+i$ ($0 \le i \le T-1$): последнее значение, которое вернула функция `process` в i-м сценарии, переведенное для удобства в десятичную систему счисления.