火星 (Mars)

人類で最初に宇宙空間に到達したのが古代エジプトのファラオたちであることは有名である. 彼らは人類史上初の宇宙船を惑星トトメス1世（最近では俗に火星と呼ばれている）に送り届けた. この惑星の表面は正方形のマスからなる $(2n+1) \times (2n+1)$ のグリッドとして表すことができ, 各マスは**陸**か**海**かのどちらかである. i 行目 j 列目 ($0 \leqq i, j \leqq 2 \cdot n$) のマスの**状態**は, $s[i][j] = 1$ であれば陸であり, $s[i][j] = 0$ であれば海である.

2 つの陸マスは, その 2 マスの間に陸マスのみからなる経路であって経路上のどの連続する 2 マスも辺を共有するようなものが存在する場合に**連結である**という. **島**とは, 含まれるどの 2 マスも連結であるようなマスの集合のうち極大であるもののことをいう.

宇宙船の任務は惑星にある島の数を数えることである. しかし, 宇宙船が古代のコンピュータを搭載しているためにこの課題は簡単ではない. 宇宙船のコンピュータは, $(2n+1) \times (2n+1)$ の 2 次元配列の形でデータを格納するメモリ h を持っており, この 2 次元配列の各要素は長さ 100 の二進文字列 (`'0'`(ASCII 48) および `'1'` (ASCII 49) からなる) を格納できる. 初期状態ではこのメモリの各要素の最初のビットには対応するマスの状態が格納されている. すなわち, 全ての $0 \leqq i, j \leqq 2 \cdot n$ について $h[i][j][0] = s[i][j]$ である. その他のビットは初期状態で `'0'` (ASCII 48) である.

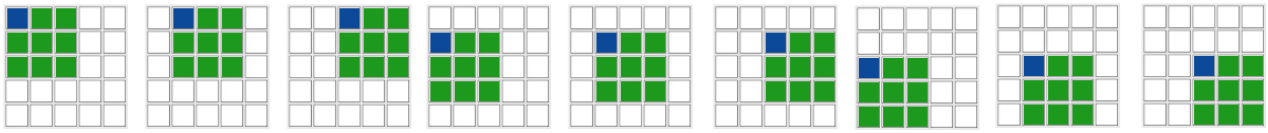
メモリに格納されたデータを処理する際, このコンピュータはメモリの 3×3 の部分にアクセスし, その一番左上のマスの値を書き換えることしかできない. より形式的には, コンピュータは $h[i..i+2][j..j+2]$ ($0 \leqq i, j \leqq 2 \cdot (n-1)$) のデータにアクセスし $h[i][j]$ の値を書き換えることしかできない. この操作を**マス** (i, j) **の処理**と呼ぶことにする.

このようなコンピュータの制約に対処するため, ファラオたちは以下のような方式を考案した:

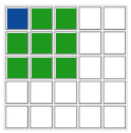
- コンピュータは n 段階に渡ってメモリを操作する.
- k $(0 \leqq k \leqq n-1)$ 段階目の操作では, $m = 2 \cdot (n-k-1)$ とすると, コンピュータは全ての $0 \leqq i, j \leqq m$ について i の昇順, 各 i について j の昇順にマス (i, j) の処理を行う. 言い換えると, コンピュータはマス $(0,0), (0,1), \cdots, (0,m), (1,0), (1,1), \cdots, (1,m), \cdots, (m,0), (m,1), \cdots, (m,m)$ の処理をこの順に行う.
- 最後の段階 ($k = n-1$) ではコンピュータはマス $(0,0)$ の処理のみを行う. その後 $h[0][0]$ に書かれている文字列は, 惑星上の島の数を最初の文字を最下位ビットとして二進表記したものになっていなければならない.

下のダイアグラムはこのコンピュータが 5×5 ($n = 2$) の大きさのメモリを処理する場合を示している. 青いマスはそのマスの値がちょうど書き換えられていることを表し, 緑のマスは今処理されている部分配列を表している.

0 段階目では, コンピュータは以下の部分配列を左から順に処理する:



1 段階目では, コンピュータは部分配列を 1 つだけ処理する:



あなたの課題は, 惑星トトメス1世にある島の数をこのコンピュータに計算させる方法を実装することである.

実装の詳細

以下の関数を実装しなさい:

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

- a: コンピュータが今処理している 3×3 の部分配列, 具体的には $a = h[i..i+2][j..j+2]$ である. a の各要素は, `'0'` (ASCII 48) および `'1'` (ASCII 49) のみからなる長さ 100 の文字列である.
- i, j: コンピュータが今処理しているマスの行番号と列番号
- k: 今何段階目かを表す数
- n: 合計の段階数. またこれは惑星の表面が $(2n+1) \times (2n+1)$ のグリッドであることを表す.
- この関数は長さ 100 の二進文字列を返さなければならない. 返された値は $h[i][j]$ に格納される.
- この関数の最後の呼び出しでは $k = n - 1$ である. この呼び出しにおいてこの関数は, 最下位ビットを 0 番目の文字, 次に下位のビットを 1 番目の文字, ...として惑星上の島の数を二進表記した文字列を返さなければならない.
- この関数は静的変数やグローバル変数に依存してはならず, また戻り値は引数の値以外に依存してはならない.

各テストケースは T 個の独立なシナリオ (違う惑星について考える場合など) からなる. あなたの実装の挙動はシナリオの順序に依存してはならない. 各シナリオに対応する `process` の呼び出しは連続して行われない可能性があるからである. なお, 各シナリオ内では問題文で指定された順序で `process` が呼び出されることが保証される.

また, 各テストケースについてあなたのプログラムの複数のインスタンスが同時に起動される. メモリ制限やCPU時間の制限はこれらのインスタンス全て合計したものに対して課せられる. 故意に正規でない方法でこれらのインスタンス間でデータを送受信しようとすることは不正とみなされ, 失格の事由となる.

特に, `process` 関数の呼び出し中に静的変数やグローバル変数に格納されたデータは次の呼び出し時まで利用可能であるとは限らない.

制約

- $1 \leqq T \leqq 10$
- $1 \leqq n \leqq 20$

- $0 \leqq s[i][j] \leqq 1$ ($0 \leqq i, j \leqq 2 \cdot n$)
- $h[i][j]$ の長さは 100 である ($0 \leqq i, j \leqq 2 \cdot n$)
- $h[i][j]$ の各文字は `'0'` (ASCII 48) または `'1'` (ASCII 49) である ($0 \leqq i, j \leqq 2 \cdot n$)

各 `process` 関数の呼び出しについて:

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

小課題

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

入出力例

入出力例 1

$n = 1$ で s が以下のようになっている場合を考える:

```
1 0 0
1 1 0
0 0 1
```

この例では, 惑星の表面は 3×3 のマスで構成され 2 つの島がある. `process` 関数が呼ばれる処理の段階は 1 つしかない.

段階 0 において, 採点プログラムは `process` 関数をちょうど 1 回次のように呼び出す

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

上では各 h の要素の最初の 3 ビットのみが書かれていることに注意せよ.

この関数呼び出しは `"0100..."` (省略されているビットは全て `0` である) を返さなければならない. ここで, 二進表記の ....0010 は十進表記の 2 に等しい. 96 個の `0` が省略されて `...` となっていることに注意せよ.

入出力例 2

$n = 2$ で s が以下のようになっている場合を考える:

```
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
```

この例では, 惑星の表面は 5×5 のマスで構成され 4 つの島がある. `process` 関数が呼ばれる処理の段階は 2 つある.

段階 0 において, 採点プログラムは `process` 関数を 9 回呼び出す:

```
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)
```

上の呼び出しがそれぞれ `"011", "000", "000", "111", "111", "011", "110", "010", "111"` (ただし省略されたビットは全て `0`) を返したとする . すると段階 0 が終了した時点で h は以下のようなデータを格納している:

```
"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"
```

段階 1 において, 採点プログラムは `process` 関数を 1 回呼び出す:

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

最後に, この関数呼び出しは `"0010000...."` (ただし省略されたビットは全て `0`) を返さなければならない. ここで, 二進表記の $....0000100$ は十進表記の 4 に等しい. 93 個の `0` が省略されて `...` となっていることに注意せよ.

採点プログラムのサンプル

採点プログラムのサンプルは入力を以下の書式に従って読み込む:

- 1 行目: T
- ブロック i ($0 \leqq i \leqq T-1$): シナリオ i を表すブロック.

    - 1 行目: n
    - $2 + j$ 行目 ($0 \leqq j \leqq 2 \cdot n$): $s[j][0]$ $s[j][1]$ $\ldots$ $s[j][2 \cdot n]$

採点プログラムのサンプルは結果を以下の書式に従って表示する:

- $1 + i$ 行目 ($0 \leqq i \leqq T - 1$): シナリオ i における `process` 関数の最後の呼び出しの戻り値 (十進表記)