Bacterial Tactics 細菌の戦術

Problem 問題

Becca and Terry are microbiologists who have a friendly rivalry.

When they need a break from their research, they like to play a game together.

The game is played on a matrix of unit cells with **R** rows and **C** columns.

Initially, each cell is either empty, or contains radioactive material.

On each player's turn, if there are no empty cells in the matrix, that player loses the game.

Otherwise, they choose an empty cell and place a colony of bacteria there.

Bacteria colonies come in two types: H (for "horizontal") and V(for "vertical").

- When a type H colony is placed into an empty cell, it occupies that cell (making it non-empty), and also tries to spread into the cell immediately to the west (if there is one) and the cell immediately to the east (if there is one).
- When a type V coloney is placed into an empty cell, it occupies that cell (making it non-empty), and also tries to spread into the cell immediately to the south (if there is one) and the cell immediately to the north (if there is one).

ベッカとテリーは微生物学者であり、仲の良い競争相手だ。

研究の休憩のときには、一緒にゲームをするのが好きである。

ゲームは、セルを単位とする **R** 行 **C** 列の行列において行われる。

最初、それぞれのセルは空か、あるいは放射線物質を含んでいる。

各プレイヤーのターンにおいて、行列に空のセルがなければ、そのプレイヤーはゲームに負けである。

そうでないならば、プレイヤーは空のセルを選び、そこに細菌のコロニーを置く。

細菌のコロニーには次の 2 種類がある。(水平を意味する) H と、(垂直を意味する) V である。

- もしもタイプ H のコロニーが空のセルに置かれると、そのセルを占有する。(つまり、そのセルを空でなくする。)　そして、(もし西にセルがあれば) すぐ西のセルに、また (もし東にセルがあれば) すぐ東のセルに広がろうとする。
- もしもタイプ V のコロニーが空のセルに置かれると、そのセルを占有する。(つまり、そのセルを空でなくする。)　そして、(もし南にセルがあれば) すぐ南のセルに、また (もし北にセルがあれば) すぐ北のセルに広がろうとする。

Whenever a colony (of either type) tries to spread into a cell:

- If the cell contains radioactive material, the colony mutates and the player who placed the colony loses the game.
- If that cell is empty, the colony occupies that cell (making it non-empty), and then the rule above is triggered again (i.e. the colony will try to spread further).
- If the cell already contains bacteria (of any type), the colony does not spread into that cell.

Notice that it may be possible that all of a player's available moves would cause them to lose the game, and so they are doomed.

See the sample case explanations below for examples of how the game works.

Becca makes the first move, and then the two players alternate moves until one of them loses the game.

If both players play optimally, who will win?

And, if Becca will win, how many distinct winning opening moves does she have?

(Two opening moves are distinct if and only if either use different cells, or different kinds of colony, or both.)

Input 入力

The first line of the input gives the number of test cases, **T**.

T test cases follow.

Each case begins with one line containing two integers **R** and **C**: the number of rows and columns, respectively, in the matrix.

Then, there are **R** more rows of **C** characters

（タイプによらず）コロニーがあるセルに広がるときには次のことが起こる:

- もしセルに放射線物質があれば、コロニーは変異し、そのコロニーを置いたプレイヤーはゲームに負ける。
- もしセルが空ならば、コロニーはそのセルを支配する。（空でなくする。） そして、上の規則が再び起こる。（つまり、コロニーはさらに広がろうとする。）
- もしセルにすでに（どちらかのタイプの）細菌があれば、コロニーはそのセルには広がらない。

なお、あるプレイヤーに可能なすべての動きが、そのプレイヤーを敗北させる場合はある。そのときは破滅の運命だ。

下で説明されているサンプルケースには、ゲームがどのようなものか、例が示されている。

ベッカが最初に動く。以降、どちらかのプレイヤーがゲームに負けるまで、2 人のプレイヤーが順番に動く。

両者の動きがどちらも最善なら、どちらが勝つ？

そしてもしベッカが勝つなら、彼女の可能な異なる最初の動きはいくつあったか？

（2 つの最初の動きは、異なるセルを使っているか、異なる種類のコロニーか、あるいはその両方のときにのみ、区別される。）

入力の最初の行は、テストケースの個数 **T** をもたらす。

T 個のテストケースらがつづく。

それぞれのケースは、2 つの整数 **R** と **C** を持つ 1 行で始まる。それらはそれぞれ、行列の行数と列数である。

そして、それぞれの行が **C** 文字ある、さらに **R** 行

each.

The j-th character on the i-th of these lines represents the j-th column of the i-th row of the matrix.

Each character is either . (an emtpy cell) or # (a cell with radioactive material).

が与えられる。

それらの行のなかで、i 番目の行にある j 個目の文字は、行列における、i 番目の行の j 番目の列を表している。

それぞれの文字は、（空のセルを表す）. か、（放射線物質のセルを表す）#である。

Output 出力

For each test cases, output one line containing `Case #x: y`, where x is the test case number (starting from 1), and y is an integer: either 0 if Becca will not win, or, if Becca will win, the number of distinct winning opening moves she can make, as described above.

各テストケースについて、`Case #x: y` という 1 行を出力せよ。ここで x は、（1 から始まる）テストケースの番号であり、y は次のような整数だ。Becca が勝たないときは y は 0 であり、Becca が勝つときは、上に述べたように、勝利できる異なる最初の動きの数が y である。

Limits 制約

Time limit: 30 seconds per test set.
Memory limit: 1GB
$1 \leq \mathbf{T} \leq 100$.

時間制限: テストケース 1 つにつき 30 秒。
メモリ制限: 1 GB
$1 \leq \mathbf{T} \leq 100$。

Test set 1(Visible) テストセット 1（可視）

$1 \leq \mathbf{R} \leq 4$.
$1 \leq \mathbf{C} \leq 4$.

$1 \leq \mathbf{R} \leq 4$。
$1 \leq \mathbf{C} \leq 4$。

Test set 2 (Hidden) テストセット 2（不可視）

$1 \leq \mathbf{R} \leq 15$.
$1 \leq \mathbf{C} \leq 15$.

$1 \leq \mathbf{R} \leq 15$。
$1 \leq \mathbf{C} \leq 15$。

Sample サンプル

Input

```
5
2 2
..
.#
4 4
.#..
```

Output

```
..#.
#...
...#
3 4
#.##
....
#.##
1 1
.
1 2
##
```

```
Case #1: 0
Case #2: 0
Case #3: 7
Case #4: 2
Case #5: 0
```

In Sample Case #1, Becca cannot place an H colony in the southwest empty cell or a V colony in the northeast empty cell, because those would lose.

She has only two possible strategies that do not cause her to lose immediately:

1. Place an H colony in the northwest or northeast empty cells. The colony will also spread to the other of those two cells.
2. Place a V colony in the northwest or southwest empty cell. The colony will also spread to the other of those two cells.

In Sample Case #2, any of Becca's opening moves would cause a mutation.

In Sample Case #3, five of Becca's possible opening moves would cause a mutation, but the other seven are winning.

She can place an H colony in any of the cells of the second row, or she can place a V colony in any of the cells of the second column.

In either case, she leaves two disconnected sets of 1 or 2 cells each.

In each of those sets, only one type of colony

サンプルケース#1 では、ベッカは、南西の空のセルには H コロニーを置くことはできず、北東のセルに V コロニーを置くことはできない。そうすれば負けてしまうからだ。

彼女がただちに負けないためには、次の 2 つの可能な戦略だけが存在する:

1. H コロニーを北西か北東の空のセルに置く。そのコロニーには、それら 2 つのセルのもう片方にも広がることになる。
2. V コロニーを北西か南西の空のセルに置く。そのコロニーには、それら 2 つのセルのもう片方にも広がることになる。

サンプルケース#2 では、Becca のどの最初の動きも、変異を起こす。

サンプルケース#3 では、ベッカの可能な最初の動きのうち 5 個は変異を引き起こす。だが他の 7 個は勝利をもたらすものだ。

彼女は 2 行目の任意のセルに H コロニーを置けるし、そうではなく、2 列目の任意のセルに V コロニーを置くこともできる。

どちらの場合にも、1 つまたは 2 つのセルからなる互いに接続しない 2 つの集まりができる。

そのどのセットについても、1 種類のコロニーし

can be played, and playing it consumes all of the empty cells in that set.

So whichever of those sets Terry chooses to consume, Becca can consume the other, leaving Terry with no moves.

In Sample Case #4, both of Becca's two distinct possible opening moves are winning.

In Sample Case #5, Becca has no possible opening moves.

か置くことができない。そして置けるコロニーを置くと、そのセットの空のセルは使い果たされる。

よって、テリーがそのどのセットを選ぶとしても、ベッカはもう片方を使い果たし、テリーの置き場をなくすことができる。

サンプルケース#4 では、存在するベッカの可能な最初の動きはどちらも勝利をもたらす。

サンプルケース#5 では、ベッカには可能な最初の動きがない。

Analysis 解析

Test set 1 テストセット 1

On a player's turn, the grid will be in some state, according to whether any bacteria have already been placed, and how they have spread.

We can determine whether a state is *losing* via the following recursive definition:

- the player has no moves because there are no empty cells
- any of the player's moves would either cause a mutation, or give the opponent a *winning* state.

Observe that if a state is not losing, it must be winning, since it has at least one move that does not cause a mutation and does not give the opponent a winning state.

To find the number of winning opening moves (if any) for Becca, we can check each move to see whether it is a winning move.

Of course, to do this, we have to investigate the resulting state recursively per the above definition.

However, since there are up to two moves per empty cell per state, the naive implementation that recursively counts the number of winning

プレイヤーのターンのときに、細菌がどのように配置され、そしてどのように広がったかによって、グリッドはある状態にある。

私達は、次の再帰的な定義を用いて、ある状態が**負けている**かを決定できる。

- 空のセルがないため、プレイヤーが可能な動きがない
- どのプレイヤーの動きも、変異を起こすか、対戦相手に**勝っている**状態を与えてしまうかである。

こう考えると、ある状態が負けているのでなければ勝っているのだと言える。なぜなら、その状態において、変異も起こさず、対戦相手に勝っている状態を渡しもしない動きが 1 つ以上あるからである。

ベッカの勝利する最初の動きの個数を（もしあれば）見つけるために、私達はそれぞれの動きについて、勝利する動きかチェックできる。

もちろん、それを行うためには、上記の定義ごとに結果の状態を再帰的に調べなければならない。

だが、それぞれの状態について、空のセルごとに、可能な 2 つの動きがあるから、それぞれの状態の勝っている動きの数を再帰的に数えるための素朴な

moves for each state may not be fast enough to handle even the 4×4 grids in test set 1, so we should optimize it.

実装は、テストセット 1 の 4×4 のグリッドを扱うためにすら十分に速くはないかもしれない。よって私達はこれを最適化せねばならない。

訳注。例えばごく単純化して考えて、4×4 の 16 個のセルを 2 者が互いに選ぶとする。最初の選び方が 16 通り、次の相手の選び方が 15 通り、と続くが、$16 = 20{,}922{,}789{,}888{,}000$ であり、桁数で考えると 10^{13} だ。解空間が広いので全探索が使えない。

Notice that whether a state is winning or losing does not depend on who the player is or on any previous moves.

Since the same state may come up multiple times, we should consider memoizing our findings about each state to use in the future.

It may be daunting that the number of possible states is intractably large.

However, for any given case, there can be at most 16 initially empty cells, each of which can be either filled in by bacteria or not.

(After a colony has been placed and has spread, it no longer matters what type it was.)

So, we can put an upper bound of 2^{16} on the number of states per case.

In practice, there will be even fewer because not all states are reachable.

ある状態が勝っているか負けているかは、プレイヤーが誰か、および、それ以前の動きが何かに依存しないことに注意しよう。

同じ状態が複数回現れるかもしれないから、判定できたそれぞれの状態を将来に使うためにメモ化することを考えるべきだろう。

ありうる状態の数が扱いがたいほど大きいことが気力を削ぐかもしれない。

しかし、入力される任意のケースについて、最初の空のセルの個数は最大で 16 であり、そのそれぞれは、バクテリアで覆われているかいないかかだ。

（コロニーが置かれて広がってしまえば、コロニーの型は関係なくなる。）

よって、入力ケースあたりの状態の個数の上界として 2^{16} を置ける。

実際には、すべての状態が到達可能なのではないから、状態の個数はさらに少ない。

訳注。なお、$2^{16} = 65536$ である。

Moreover, we can save some time by not computing the exact number of winning moves for every state we examine.

We only care about this value for an initial state; for every other state, it suffices to determine whether it is winning or losing.

If we are investigating a non-initial state's moves and we find a winning move, we can declare the state to be winning, and stop.

This optimization alone may be enough to solve test set 1.

さらに私達は、調査するすべての状態についての勝っている動きの正確な個数を計算しないことで、いくらかの時間を節約できる。

初期状態についてしか私達はこの値に興味がない。他のすべての状態については、勝っているか負けているかさえ分かれば十分である。

もし私達が初期状態ではない状態の動きらを調査しており、勝っている動きを見つけたならば、その状態は勝っていると宣言して、終えることができる。

テストセット 1 を解くには、この最適化だけで十分かもしれない。

Test set 2 テストセット 2

When a player makes a legal move, the bacteria spread across the entire width or length of the row or column, up until the line of bacteria reaches the edge of the grid or a cell that is already infected.

Therefore, each move creates up to two subproblems that are independent in the sense that a move in one subproblem does not affect the state of the other.

Each subproblem can be expressed as a rectangle contained within the full grid.

There are therefore at most O($\mathbf{R}^2\mathbf{C}^2$) subproblems.

How can we use the results of these subproblems to determine the overall winner of the game?

あるプレイヤーが合法な動きを行ったとき、グリッドの端か、あるいはすでに感染しているセルに達しない限り、バクテリアは行か列の幅か高さ全体にまで広がる。

よって、それぞれの動きは、最大で 2 つのサブ問題を作成する。それらサブ問題は、片方のサブ問題での動きがもう片方の状態に影響しないという意味で、独立である。

各サブ問題は、全体のグリッドに含まれるある長方形として表現することができる。

よって、最大で O($\mathbf{R}^2\mathbf{C}^2$) 個のサブ問題が存在することになる。

それらのサブ問題の結果をどのようにして、ゲーム全体の勝者を決めるのに使うことができるだろう？

訳注。R 行 C 列の行列のなかにある長方形を考えるとき、（雑にオーダーを考えるとして、）上辺の選び方が R 通り、下辺の選び方が R 通り、また、左辺の選び方が C 通り、右辺の選び方が C 通りあるから、O($RRCC$)=O(R^2C^2) である。

The goal of the game is to force the opponent into a situation in which there is no move they can make that leads them down a path to victory.

The game is *impartial*: both players have access to the same set of moves.

It is therefore apt to draw a comparison between Bacterial Tactics and the ancient game Nim, an impartial game with similar types of decisions.

The mathematics of Nim are well-studied.

A discovery particularly useful to us is the Sprague-grundy Theorem, which says that any impartial game can be mapped to a state of Nim.

Every state in Nim corresponds to a non-negative *Grundy number*, or *nimber*, where any nonzero nimber indicates a winnable game.

ゲームの目的は、対戦相手をある状況、つまり、対戦相手の勝利につながる動きが選べない状況に追い込むことだ。

このゲームは**不偏**である。すなわち、選択しうる動きはどちらのプレイヤーも変わらない。

よって、Bacterial Tactics と、似たタイプの決定をする不偏ゲームである古代のゲーム Nim とを比較したくなる。

Nim についての数理はよく研究されている。

私達にとって特に有用な発見は、スプレイグ・グランディの定理だ。任意の不偏ゲームは、Nim の状態に写像できるとされる。

Nim のすべての状態は、非負のグランディ数に対応する。任意のゼロでないグランディ数は、勝てるゲームを意味する。

According to nimber addition, the nimber of a game state after we place a colony is equal to the *XOR* of the two subproblems.

The nimber of a game state before we place a colony is the *minimum excludant*, or *MEX*, of the set of possible nimbers after placing colonies.

We can therefore solve Bacterial Tactics recursively using the following pseudocode:

グランディ数の加算によると、私達がコロニーを置いた後のあるゲームの状態のグランディ数は、2 つのサブ問題の XOR に等しい。

私達がコロニーを置く前のゲームの状態のグランディ数は、コロニーらを置いたあとに可能なグランディ数らの補集合の最小値である。

よって私達は、次の疑似コードを使って Bacterial Tactics を再帰的に解ける。

```
let solve(state) be a function:
  let s = Ø
  for each legal colony placement:
    add [solve(first subproblem) XOR solve(second subproblem)] to s
  return MEX(s)
```

Given this general framework, we can now optimize our implementation.

この一般的なフレームワークがあるものとして、私達の実装を最適化することができる。

First, as in test set 1, we can memoize the game states, which are now defined using rectangles of various sizes within the original grid.

Note that it is not possible to have bacteria from previous moves in a subproblem, because we always cut the rectangle along the row or column of cells infected by a colony placement.

We may also want to pre-compute the nimbers of all sizes of an empty rectangle (no radioactive cells), which is information that can be shared across all test cases.

第 1 に、テストセット 1 と同じく、ゲームの状態らをメモ化できる。各状態は今回は、オリジナルのグリッドのなかの様々な大きさの長方形で定義される。

なお、あるサブ問題では、以前の動きらから細菌を得ることは不可能だ。なぜなら、私達は常に、コロニーを置いて感染した行ないし列によって長方形を切断するからである。

私達はまた、（放射性物質のセルを含まない）すべての大きさの空の長方形のグランディ数を事前に算出しておきたいかもしれない。その情報は、テストケース全体で共有できる。

Second, observe that if it is legal to place a V colony in a cell, then it is also legal to place a V colony in any cell in that column, and similarly for H colonies in a row, within the boundary of the current subproblems's rectangle.

We therfore need to check only the rows and columns for legal colony placements, not each individual cell.

第 2 に、現在のサブ問題の長方形の枠のなかで、あるセルに V コロニーを置くことが合法であるなら、同じ列のどのセルに V を置くこともまた合法であり、行については H コロニーについても同様に言えることに気づく。

よって私達は、コロニーの合法な設置のために行らや列らを調べるだけでよく、それぞれのセル自体を調べずにすむ。

Third, we can construct a data structure that allows us to determine whether a colony placement

第 3 に、あるデータ構造を作成することで、与えられた長方形の任意の行や列について、コロニーの

is legal for any row or column in a given rectangle in O(1) time, allowing us to evaluate any game state in O($\mathbf{R}$+$\mathbf{C}$) operations.

For each row and column in the full grid, create an array.

Check the cells in the row or column in ascending order, appending the 1-indexed position of the most recently seen radioactive cell, or 0 if a radioactive cell has not been encountered yet.

For example, for the row .#..#, the array would be [0, 2, 2, 2, 5].

Suppose we have a rectangle that includes the third and fourth cells of that row.

The fourth entry of the array is a 2.

Since cell 2 is not in our rectangle (we have cells 3 and 4 only), we can conclude that it is safe to place an H colony in this row of our rectangle.

This data structure can be pre-computed for each test case in O($\mathbf{RC}$) time.

設置が合法かどうかを O(1) で決められる。これにより、任意のゲームの状態を O($\mathbf{R}$+$\mathbf{C}$) 回の操作で評価できる。

グリッド全体のそれぞれの行とそれぞれの列について配列を作る。

行または列のセルらを昇順にチェックする。最も最近見た放射性物質のセルの 1-indexed 位置を追加し、放射性物質のセルをまだ見てなければ 0 を追加する。

例えば、行.#..#については、配列は [0, 2, 2, 2, 5] となる。

その行の 3 番目および 4 番目のセルを含む長方形があったとしよう。

配列の 4 番目の要素は 2 だ。

2 番目のセルは私達の長方形に入っていない（3 番目と 4 番目だけ入っている）から、長方形のこの行に H コロニーを置くことは安全だと結論できる。

このデータ構造は、それぞれのテストケースについて O($\mathbf{RC}$) 時間で事前に算出できる。

To summarize, there are O($\mathbf{R}^2\mathbf{C}^2$) subproblems, and each subproblem takes O($\mathbf{R}$+$\mathbf{C}$) operations.

If we let N be max($\mathbf{R}$, $\mathbf{C}$), this leads to O(N^5) total time complexity, sufficient for test set 2.

Less efficient solutions might still pass, depending on their implementations.

まとめると、O($\mathbf{R}^2\mathbf{C}^2$) 個のサブ問題が存在し、それぞれについて O($\mathbf{R}$+$\mathbf{C}$) 回の操作が必要だ。

N を max($\mathbf{R}$, $\mathbf{C}$) とすると、合計 O(N^5) 時間の計算量となり、テストセット 2 に十分だ。

もっと非効率な解法であっても、実装によっては通過できるかもしれない。

implementation

```
# test set 1 用の公式解説による解法。test set 1 のみ AC。
R, C = [-1, -1]

def get_empties(ss):
    """空のセルの座標らを返す。"""
    empties = []
    for r in range(R):
        for c in range(C):
            if ss[r][c] == ".":          # 空のセルを見つけた
                empties.append((r, c))  # 追加
    return empties
```

```
def place(ss, coord, is_H):
    """行列 ss の座標 coord にコロニーを置いたときの次の状態を返す。
    置けないなら False を返す。"""
    ss = ss.copy()
    if is_H:
        directions = [(0, -1), (0, +1)]  # 横に広がるつまり列が動く
    else:
        directions = [(-1, 0), (+1, 0)]  # 縦に広がるつまり行が動く
    if is_H:                             # デバッグ用
        B = "H"
    else:
        B = "V"
    B = "B"                              # メモ化のためには細菌は区別しない
    for dir in directions:               # 2 つの方向を扱う
        r, c = [coord[0], coord[1]]
        ss[r] = ss[r][:c] + B + ss[r][c+1:]  # コード簡略化のため 2 回やってる
        while True:
            r, c = r+dir[0], c+dir[1]        # 前進する
            if r<0 or r>=R or c<0 or c>=C:   # 範囲外に出たらこの方向は終わり
                break
            if ss[r][c] == "#":  # 放射性物質に遭遇した
                return False     # coord に置くことはできない
            if ss[r][c] != ".":  # すでに細菌がある
                break            # この方向は終わり
            ss[r] = ss[r][:c] + B + ss[r][c+1:]  # コロニーを広げる
    return ss  # 次の状態を返す

memoize = {}  # この辞書でメモ化する
def is_winning(ss, is_initial):
    """状態 ss が winning なら True を返す。
    ただし is_initial なら winning な move のリストを返す。"""
    args = (tuple(ss), is_initial)  # リストは hash にできないので tuple とする
    if args in memoize:
        return memoize[args]        # メモ化されている値を返す
    empties = get_empties(ss)       # 空のセルら
    winning_moves = []
    if len(empties) == 0:  # 置ける場所がない
        if is_initial:
            memoize[args] = winning_moves  # つまり空のリスト
```

```
        else:
            memoize[args] = False
        return is_winning(*args)
    for empty in empties:          # 空のセルそれぞれに置いてみる
        for hv in [True, False]:  # コロニー H と V をそれぞれ置いてみる
            next = place(ss, empty, hv)  # 置いた次の状態を得る
            if next == False:  # 座標 empty に細菌 hv は置けない
                continue
            result = is_winning(next, False)  # 次の状態は winning か？
            if result == True:  # 勝っている状態を渡してしまう
                continue
            if not is_initial:  # 置ける場所があった
                memoize[args] = True
                return is_winning(*args)
            # 座標 empty にコロニー hv を置くのは winning な move の 1 つだ。
            winning_moves.append([empty, hv])
    memoize[args] = winning_moves
    return is_winning(*args)

T = int(input())
for t in range(1, T+1):
    R, C = [int(x) for x in input().split()]
    ss = []
    for r in range(R):
        ss.append(input())
    ans = is_winning(ss, True)  # 状態 ss の winning な move らを得る
    print("Case #{}: {}".format(t, len(ans)))
```

implementation

```
# AC.
def nimber(top, lft, btm, rgt):
    if btm <= top or rgt <= lft:         # 範囲が異常
        return 0, 0                      # 敗北状態だ
    if (top, lft, btm, rgt) in memoize:  # メモ化されている
        return memoize[(top, lft, btm, rgt)]
    radio_rows = set()                   # 置けない行ら
    radio_clms = set()                   # 置けない列ら
```

```
    for r in range(top, btm):            # 細菌で閉じられてる範囲内で改めて集める
        for c in range(lft, rgt):
            if radioss[r][c]:            # 放射性物質がある
                radio_rows.add(r)
                radio_clms.add(c)
    next_nims = set()                    # 動いたあとの nimber ら。
    n_winnings = 0                       # 勝てる動きの個数
    for r in range(top, btm):            # 各行に H コロニーを置く動きら
        if r in radio_rows:              # 置けない
            continue
        # H コロニーを第 r 行に置いて上下 2 つのサブ問題に分ける。
        next_nim = nimber(top, lft, r, rgt)[0] ^ nimber(r+1, lft, btm, rgt)[0]
        next_nims.add(next_nim)
        if next_nim == 0:                # 敵に敗北状態を渡す。つまり勝利状態。
            n_winnings += rgt-lft        # H が置けるセルの数を加算する
    for c in range(lft, rgt):            # 各列に V コロニーを置く動きら
        if c in radio_clms:
            continue
        # V コロニーを第 c 列に置いて左右 2 つのサブ問題に分ける。
        next_nim = nimber(top, lft, btm, c)[0] ^ nimber(top, c+1, btm, rgt)[0]
        next_nims.add(next_nim)
        if next_nim == 0:
            n_winnings += btm-top
    nim = 0      # minimum excludant (MEX) を得る
    while True:  # 動く前の nimber は動いた後の nimber らの MEX として得られる
        if nim not in next_nims:
            memoize[(top, lft, btm, rgt)] = nim, n_winnings
            return nim, n_winnings
        else:
            nim += 1

T = int(input())
for t in range(1, T+1):
    R, C = [int(x) for x in input().split()]
    memoize = {}
    radioss = []
    for i in range(R):
        s = input().strip()
        radios = [c != '.' for c in s]  # 放射性物質があるセルを True で表す
        radioss.append(radios)
```

```
_, n_winnings = nimber(0, 0, R, C)
print("Case #{}: {}".format(t, n_winnings))
```