Marathon Match

Problem Statement

Contest: 2014 TCO Marathon Round 1
Problem: SquareRemover

Normal view

Problem Statement

IMPORTANT: This problem is used for two simultaneous matches: TCO'14 Marathon Round 1 and Marathon Match 83. You can compete in TCO'14 R1 only if you are eligible for TCO'14. You can compete in MM 83 if you are not eligible for TCO'14 or if you would like to skip TCO'14 R1 by some reason. Competing in both matches is not allowed. Doing so will lead to disqualification. Note that registration does not count as competing. In order to be considered a competitor, you need to make at least one submit (example or full).

"Square Remover" is a single player game with full information. It consists of:

- An NxN board where each 1x1 tile is colored in one of **colors** colors.
- A line of tiles where each 1x1 tile is also colored in one of **colors** colors. The line has a fixed left end and goes infinitely long to the right. This line will be referred to as *buffer*. At certain moments of the game, the leftmost tile can be removed from the buffer and get placed on the board.

During the game you will need to make exactly 10,000 move. Each move consists of choosing two tiles on the board that share a side and swapping them with each other. At any point in the game, if there exists a 2x2 square on the board where all 4 tiles have the same color, your score is increased by 1 and 4 tiles of the square get replaced by 4 leftmost tiles taken from the buffer. Your goal is to obtain the maximum possible score in the end of the game.

The following pseudocode illustrates the game process in more details:

```
Procedure SquareRemoverGameplay:
        GameAdjustsYourScore
        For move = 0, 1, ..., 9999:
                YouMakeNextMove
                GameAdjustsYourScore

Procedure GameAdjustsYourScore:
        While there is a 2x2 square on the board where all 4 tiles have the same color:
                Let S be the topmost of such squares.
                If there are several topmost squares, then let S be the leftmost among them.
                Increase score by 1.
                Replace the top left tile in S with the leftmost tile taken from the buffer.
                Replace the top right tile in S with the leftmost tile taken from the buffer.
                Replace the bottom left tile in S with the leftmost tile taken from the buffer.
                Replace the bottom right tile in S with the leftmost tile taken from the buffer.
```

Implementation

You will need to implement a method `playIt` that takes int **colors**, vector <string> **board** and int **startSeed** as input parameters.

colors is the number of different colors in this instance of the game. Colors are numbered from 0 to **colors**-1. **board** describes the initial contents of the board. It contains N elements and each element contains N characters. j-th character in element i is a digit that describes the color of the tile at row i, column j. Rows are numbered 0 to N-1 from top to bottom. Columns are numbered 0 to N-1 from left to right.

startSeed allows you to generate the contents of the buffer. Consider the following sequence of pseudorandom numbers A[i]:

```
A[0] = startSeed
A[i] = (A[i-1] * 48271) MOD 2147483647
```

The color of the i-th (0-based) tile from the left in the buffer is A[i] MOD **colors**. Here X MOD Y returns the remainder of integer division X/Y. For example, 123 MOD 10 = 3. The sequence A allows you to generate the colors of as many tiles from the buffer as you need. Please be advised to use 64-bit integers when evaluating A[i] from A[i-1] in order to avoid integer overflow.

The return value from `playIt` should be a vector <int> containing 30,000 elements. Your i-th move should be describe by 3 elements of the return value:

- Elements 3*i and 3*i+1 give the row and the column of one of the tiles used in your move.
- Element 3*i+2 gives the direction from already specified tile to another tile used in your move. Directions are encoded as follows: 0 -- up, 1 -- right, 2 -- down, 3 -- left.

Scoring

For each test case we will calculate your raw and normalized scores. If you were not able to produce a valid return value (due to time limit, memory limit, crash, numbers in your return value do not describe 10,000 valid moves, etc.), then your raw score is -1 and the normalized score is 0. Otherwise, the raw score will be your score once the game is finished. The normalized score for each test is 1,000,000.0 * YOUR / BEST, where BEST is the highest score currently obtained on this test case (considering only the last submission from each competitor). Finally, your total score is equal to the arithmetic average of normalized scores on all test cases.

You can see your raw scores on each example test case when you are making an example submit. You can also see total scores of all competitors on provisional test set in the match standings. No other information about scores is available during the match.

Test case generation

Each test case is generated as follows:

- **colors** is chosen uniformly, at random, between 4 and 6, inclusive.
- N is chosen uniformly, at random, between 8 and 16, inclusive.
- Color of each tile on the board is chosen uniformly and independently, at random, between 0 and **colors**-1, inclusive.
- **startSeed** is chosen uniformly, at random, between 1 and 2,147,483,646, inclusive.

Tools

An offline tester/visualizer is available. You can use it to test/debug your solution locally. You can also check its source code for exact implementation of test case generation and score calculation.

Definition

Class:	SquareRemover
Method:	playIt
Parameters:	int, vector <string>, int
Returns:	vector <int>
Method signature:	vector <int> playIt(int colors, vector <string> board, int startSeed)

(be sure your method is public)

Notes

- The time limit is 30 seconds per test case (this includes only the time spent in your code). The memory limit is 1024 megabytes.
- There is no explicit code size limit. The implicit source code size limit is around 1 MB (it is not advisable to submit codes of size close to that or larger). Once your code is compiled, the binary size should not exceed 1 MB.
- The compilation time limit is 60 seconds. You can find information about compilers that we use and compilation options here.
- There are 10 example test cases and 100 full submission (provisional) test cases.

Examples

0)

```
colors = 5
N = 14
startSeed = 857377961
```

1)

```
colors = 4
N = 11
startSeed = 55004692
```

2)

```
colors = 5
N = 8
startSeed = 257017653
```

3)

```
colors = 4
N = 8
startSeed = 373353050
```

4)

```
colors = 6
N = 11
startSeed = 104364742
```

5)

```
colors = 4
```

```
N = 11
startSeed = 1329682683
```

6)

```
colors = 4
N = 16
startSeed = 656773882
```

7)

```
colors = 5
N = 14
startSeed = 1086656358
```

8)

```
colors = 5
N = 14
startSeed = 1786703902
```

9)

```
colors = 5
N = 12
startSeed = 45966541
```
