Letter Eating

Bidik enjoys letters. He enjoys them so much, he feels compelled to eat a letter every now and then. Bidik is given a matrix (of size $n \times m$) of uppercase Latin letters. Each letter is either old ('O') or new ('N'). Old letters are from 2023 and are worth 2023 calories, while new letters are from our current year (2024) and are worth 2024 calories (each letter is worth 2023 or 2024 calories). He also has some jumping power, a positive integer k. Bidik can't jump too far between letters, so he will eat the letters according to these rules:

- Bidik has a preference for the first and last letters. Thus, he must always start his eating rampage at the letter in position $(0, 0)$ and end it at $(n - 1, m - 1)$.
- After eating a letter at position (i, j), he can only eat a letter at any position (a, b) (not equal to (i, j)) such that:
  - $i \leq a$
  - $j \leq b$
  - $|i - a| + |j - b| \leq k$

i.e., he can't eat a letter that's too "far".

- He can only eat letters in order, i.e., after A he can only eat B, after B he can only eat C, etc. After Z he can eat A.
- Eating an old letter gives him 2023 calories, while eating a new letter gives him 2024 calories.

As such, each possible sequence of letter-eating has some value associated with it, computed as the sum of the calorie content of all the letters that were eaten. If there are no valid sequences of letter eating, the maximum value Bidik can earn is 0.

Mile doesn't want Bidik to be too stuffed, so he tries to stop him from eating letters. He has q possible actions. In action p, he will choose the letter at (x_p, y_p) and change it to t_p. For each action, he wants to know if after performing the action, the maximum value Bidik can earn increases (Losho!), stays the same (Arno!), or decreases (Dobro!). Note that the actions are independent of each other, i.e., they do not affect the matrix Bidik was given.

Implementation Details

Implement the letter-eating function in the provided `letter-eating.cpp` file. Include the header file `letter-eating.h`.

Implement the function

```
vector< char > letter_eating(int n, int m, int k, int q,
                             vector < vector < char > > c,
                             vector < vector < char > > v, vector < int > x,
                             vector < int >  y, vector < char > t)
```

provided in the letter-eating.cpp file with the following parameters:

- The aforementioned integers n, m, k, q $(1 \leq n, m, k, q \leq 10^5, 1 \leq nm \leq 10^5)$.
- Two matrices with n rows and m columns c_{ij} and v_{ij} ($0 \leq i \leq n - 1$, $0 \leq j \leq m - 1$) represent the letter and the age of the letter, respectively, at that position. c_{ij} is an uppercase Latin letter, and v_{ij} is either 'O' or 'N', as described above.
- Three arrays of size q, x_p, y_p, t_p ($0 \leq p \leq q - 1$), where each tuple (x_p, y_p, t_p) represents a query specified above.

Return an array of characters of size q, each entry containing a character equal to either 'L' for an increased max value, 'A' for the same max value, or 'D' for a decreased max value.

Sample Grader

The sample grader has the following input format:

- Line 1: n m k q
- Line 2 + i (for $0 \leq i \leq n - 1$): m characters, c_{i1} c_{i2} ... c_{im}
- Line 2 + $n + i$ (for $0 \leq i \leq n - 1$): m characters, v_{i1} v_{i2} ... v_{im}
- Line 2 + $2n + p$ (for $0 \leq p \leq q$): x_p y_p t_p

Example

Call	Return Value
`letter_eating`(3, 3, 2, 2, {{ A, A, B }, { D, B, C }, { D, B, D }}, {{ N, O, N }, { O, O, N }, { N, N, O }}, { 0, 2 }, { 0, 2 }, { Z, J })	{ L, D }
`letter_eating`(6, 1, 3, 5, {{ C }, { D }, { E }, { D }, { E }, { F }}, {{ N }, { N }, { N }, { O }, { O }, { O }}, { 3, 1, 3, 3, 0 }, { 0, 0, 0, 0, 0 }, { Z, F, X, Y, D })	{ A, D, A, A, D }

Subtasks

- **(14 points)** Answers will only be 'L' or 'A', $n = 1$, $q = 1$.
- **(16 points)** Answers will only be 'L' or 'A', $n = 1$, $k \leq 100$.
- **(10 points)** Answers will only be 'L' or 'A', $n = 1$.
- **(11 points)** $k \leq 100$.
- **(19 points)** $q = 1$.
- **(30 points)** No further constraints.