Segment Tree - Composite Transformation using Matrix Multiplication with Lazy Propagation

Description of Problem

1. Functionality

   Given an array of transformations that can be applied to a point in 2D space, you are asked to support the following functionalities:

   (a) Query: Apply transformations to a given point.
   Given an (x, y) coordinate, a left index l, and a right index r, find the new location of this point after applying all of the transformations in the transformation array from index l to r to the point.

   (b) Range Update: Apply update to a range of transformations in the array
   Given a new transformation, a left index l, and a right index, replace all the transformations from index l to r in the array with the new transformation

2. Transformation Types

   There are three types of transformations which will be used:

   (a) Translation: Move the point (x, y) to $(x + dx,\ y + dy)$

   (b) Scale: change the point (x, y) to $(x \cdot Sx,\ y \cdot Sy)$. It actually moves the point away from the origin by some multiplicative factor (Sx, Sy)

   (c) Rotation: Rotate the point around the origin by degree θ. Note that positive rotations should move counterclockwise around the origin.

3. Implementation Requirement

   (a) Segment Tree will be used to store transformations.

   (b) To improve efficiency of query and update, it is required to support lazy propagation. Lazy propagation postpones updates to child nodes until necessary. When a non-leaf node (including the root node) of the segment tree receives a range update and the update index range matches the node index range, the node won't propagate the transformation update to its children until another query or update is received.

4. Input and output

   (a) Input is similar to the "Transforms" homework, except the update ("U")
   The original update format in homework is "$U\ i\ < New\,Transformation >$ ", which only replaces the index i in the transformation list with the new transformation

The new update format is "$U\ l\ r\ < New\,Transformation >$", which replaces the transformation from index l to index r in the transformation list with the new transformation.

(b) Output is same as the output of the "Transforms" homework

Solution of Problem

1. Transformation Matrix

For each transformation type, a transformation matrix will be generated before applying transformation to any point.

(a) Translation $(dx,\ dy)$: the corresponding transformation matrix is as follows

$$\begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}$$

(b) Scale (Sx, Sy): the corresponding transformation matrix is as follows

$$\begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(c) Rotation θ: the corresponding transformation matrix is as follows

$$\begin{bmatrix} cos\theta & -\,sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2. Composite Transformation

When applying a transformation to a range of m

(a) For translation $(dx,\ dy)$, the composition translation is $(m\ \cdot\ dx,\ m\ \cdot\ dy)$

(b) For scale (Sx, Sy), the composition scale is $(Sx^{m},\ Sy^{m})$

Note - due to the exponential growth of scale transformation, it is suggested to limit the update range (value of m) especially when the size n of the transformation array is big.

(c) For rotation θ m t, the composition rotation is $m\theta$

3. Apply Transformation

Since the transformation matrix T is 3×3, we have to convert the original (x, y) to a 3×1 vector v as follows:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The new point (x', y') is also mapped to a vector v' as follows:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

So $v' = Tv$

4. Segment Tree

   Each node of Segment Tree will contain the following data:

   - Transformation matrix 3×3
   - Lazy Propagation Flag
   - Lazy Propagation Transformation (type, related data)

5. Build Segment Tree using the given list of transformations

   Each transformation from input will be mapped to a transformation matrix and loaded into a leaf node of Segment Tree.

   Transformation Matrix (A) of each internal node is calculated using Transformation Matrix (L) of left child and Transformation Matrix (R) of right child: $A = LR$

6. Propagation

   If Lazy Propagation Flag of a node is true, Propagation is invoked during query or range update of this node

   - Propagate lazy transformation (flag and transformation data) to the left child and right child
   - Calculate Composite Transformation Matrix using using Lazy Propagation Transformation and the update range (n = right index - left index + 1) and use it to update Transformation matrix of the node

7. Range Update

    Starting from the root node of Segment Tree

    - If Lazy Propagation Flag is True, invoke Propagation
    - If the update index range matches the current index range of the node,
        - set Lazy Propagation Flag to True
        - update Lazy Propagation Transformation with the new transformation
    - Otherwise
        - Depending on update range, recursively call Range Update to left child and/or right child
        - Recalculate Transformation matrix (A) of the current node using updated Transformation Matrix (L) of left child and Transformation Matrix (R) of right child: $A = LR$

8. Query

    (1) Find Composite Transformation Matrix

    Starting from the root node of Segment Tree

    - If Lazy Propagation Flag is True, invoke Propagation
    - If the current index range matches the current index range of the node,
        - Return Transformation matrix of the current node
    - Otherwise
        - Depending on query range, recursively call for left child and/or right child
        - If recursive calls to both children, return multiplication of Transformation matrix from left chid and Transformation matrix from right chid.
        - If recursive call to only one child, return Transformation matrix from that child

    (2) Apply composite transformation to the given point using Composite Transformation Matrix from (1) to calculate the new location.

9. Complexity

    For Query and Update, time complexity is O(log N)

    For Build Tree, both time and space complexity is O(N)

Implementation of Solution

The solution is implemented in C++. The file is transform_range_update.cpp under the directory transforms_assignment.

Test Cases

All the test cases are stored under the directory transforms_assignment/range_io_20.
The test cases are generated using the test files of the original “Transforms” homework but change the single index for update to an index range (left index and right index) for range update. The index range for Update is generated randomly. For scale, we restrict the index range no more than 20. The file to generate all test files is generate_range.cpp.

The brute force comparison is implemented in brute_force_main.cpp. The result of brute force comparison is used to verify the test result of each test case.