CSCI-GA.1170-001/002 Fundamental Algorithms	October 29, 2015
Solutions to Problem 3 of Homework 5 (8 points)	
Name: GOWTHAM GOLI (N17656180)	*Due: Tuesday, October 13*

We want to build a data structure for maintaining a (potentially infinite) matrix M and support the following operations.

- INITIALIZE(M): Create an empty matrix M with all zero entries.
- FIND(M, i, j): Return the value at index i, j.
- UPDATE(M, i, j, e): Change the value at index i, j to e.
- TRANSPOSE(M): Transpose the matrix M.
- ADD(M): Return the sum of all entries of M.

Assume that the matrix is of arbitrary dimensions. Use 2-3 trees appropriately to obtain a data structure such that INITIALIZE, TRANSPOSE, and ADD run in $O(1)$ time, and FIND and UPDATE run in $O(\log k)$ time, where k is the number of non-zero entries in the matrix.

Solution:

Building M using 2-3 Tree

Let H be a hash function such that $H[(i, j)]$ produces a unique key for each pair (i, j). Assume that applying H on any index (i, j) takes only $O(1)$ time.

Let M be an empty 2-3 Tree. Every node $v \in M$ has the following fields

- $v.left, v.mid, v.right$
- If v is a leaf node then $v.key$ stores the hashed value of (i, j). If v is a non-leaf node then $v.key$ stores the maximum key of it's children
- $v.sum$ stores the sum of all the sums of it's children. If v is a leaf node then $v.sum = v.key$
- If v is a root node then v has one additional field $init$ such that $v.init = k$ initializes all the matrix entries to k
- If v is a leaf node then v has one additional field $value$ where $v.value$ stores the value at $M(i, j)$

Now compute $H[(i, j)]$ for every (i, j) and insert them into M while maintaining the above fields at every node.

Initialize(M)

Let x be the root of M. Update $x.init$ to 0. Therefore, Initialize takes $O(1)$ time

Find(M, i, j)

Compute $H(i, j)$. Now perform the usual Search operation and return $v.value$ where v is the leaf node returned by the Search operation. Therefore, Find takes $O(\log k)$ time

Update(M, i, j, e)

Compute $H(i, j)$. Now perform the usual Search operation and update $v.value$ to e where v is the leaf node returned by the Search operation. Therefore, Update takes $O(\log k)$ time

Transpose(M)

When a matrix is transposed, the indices are all flipped i.e $(i, j) = (j, i)$. Therefore to return any element at (i, j) position return the element at (j, i) position. Therefore, Transpose takes $O(1)$ time

Add(M)

If x is the root of M then return $x.sum$. Therefore, Add takes $O(1)$ time $\square$