Идея: lock-free на немодифицирующих топ. порядок операциях

В рамках данной идеи разработаем алгоритм, который будет lock-free, если добавление ребра не нарушает текущий топологический порядок, и берется блокировка при операции модификации. Основная идея заключается в том, чтобы не производить "хорошие" (не нарушающие топ. порядок) добавления физически, а добавлять их в некую очередь.

ВНИМАНИЕ! ТУТ ЕСТЬ БАГИ! МНОГО!

```
class Node {
    OrderAndEpoch: Long or Descriptor // should have volatile semantics

    fun Ord(): Int // returns order
    fun Epoch(): Int // returns epoch when the order has been modified
}
data class Descriptor(newValue: Long)

shared Epoch: Int
shared AddQueueTail: MultipleProducerSingleConsumerQueue<Edge>

shared OrdInt: Map<Int, Node> // Maps orders to the nodes, for modifier thread only

fun add((u, v): Edge): List<Cycle> {
retry: while (true) {
    // Read current epoch and orders of the edge endings
    val curEpoch = Epoch
    val ordU = u.OrderAndEpoch
    val ordV = v.OrderAndEpoch
    // Check that ordU and ordV are not Descriptors,
    // otherwise help to complete the operation
    if (ordU is Descriptor) {
        u.OrderAndEpoch.CAS(ordU, ordU.newValue)
        continue retry
    } else if (ordV is Descriptor) {
        v.OrderAndEpoch.CAS(ordV, ordV.newValue)
        continue retry
    }
    // Add the edge
    if (ordU < ordV) {
        // Edge could be added without topological order modification.
        // So add it to the "add" operations queue if the epoch has not been changed
        val success = atomically {
            AddQueueTail.add( (u,v) )
            assert(curEpoch == Epoch)
        }
    } else {
        // Otherwise we should rearrange nodes between v and u
        exclusively {
                // Try to add the edge without modification after the writer lock is acquired
                if (tryToAddWithoutModification((u, v))
                        return noCycles() // just an empty list
            performAllOperationsFromAddQueue() // Optimisation only
            (newTopOrder, success) = countNewTopOrder() // Count new topological order
            if (success) {
                // If (u, v) edge could be added successfully,
                // update the topological order values for other threads.
                // Add descriptors firstly (for lock-free unmodifiable operations) ...
                for (i = v.Ord() .. u.Ord())
                    ordInv[i].OrderAndEpoch = Descriptor(newTopOrder[i])
                // ... Increment the epoch
                Epoch++
                // Edge (u, v) has been added, but it is not guaranteed
                // that new edges in the queue do not break new topological order,
                // so check it and revert topological order change if needed
                val successAdd = addEdgesFromQueueInBatch()
                                if (successAdd) {
                            // Edge (u, v) has been added successfully, replace Descriptors with new values if
```

```
needed
                        performDescriptors(v.Ord(), u.Ord())
                else {
                        // "Good" edges from the queue create a cycle,
                        // revert modification operation
                        revertTopOrder()
                        // Complete previous descriptors replacement
                        performDescriptors(v.Ord(), u.Ord())
                        // Revert topological order
                        for (i = v.Ord() .. u.Ord())
                            ordInv[i].OrderAndEpoch = Descriptor(prevTopOrder[i])
                    // Start new epoch
                    Epoch++
                    // Replace descriptors with their values
                    performDescriptors(v.Ord(), u.Ord())
                    // Find cycles and return them
                    return getCycles(u, v)
                }
            } else {
                    // Find cycles and return them
                    return getCycles(u, v)
            }
        }
    }
    return noCycles() // just an empty list
}

fun performDescriptors(from: Int, to: Int) {
        for (i = from .. to) {
            val ordI = ordInv[i].OrderAndEpoch
        if (ordI is Descriptor) // Otherwise another thread replaces the descriptor with its value before
                ordInv[i].OrderAndEpoch.CAS(ordI, ordI.newValue)
    }
}
```