4.5 Lecture Summary

4.5 Minimum Spanning Tree Example

Lecture Summary: In this lecture, we discussed how to apply concepts learned in this course to design a concurrent algorithm that solves the problem of finding a *minimum-cost spanning tree* (MST) for an undirected graph. It is well known that undirected graphs can be used to represent all kinds of networks, including roadways, train routes, and air routes. A spanning tree is a data structure that contains a subset of edges from the graph which connect all nodes in the graph without including a cycle. The cost of a spanning tree is computed as the sum of the weights of all edges in the tree.

The concurrent algorithm studied in this lecture builds on a well-known sequential algorithm that iteratively performs *edge contraction* operations, such that given a node *N1* in the graph, *GetMinEdge(N1)* returns an edge adjacent to *N1* with minimum cost for inclusion in the MST. If the minimum-cost edge is (*N1,N2*), the algorithm will attempt to combine nodes *N1* and *N2* in the graph and replace the pair by a single node, *N3*. To perform edge contractions in parallel, we have to look out for the case when two threads may collide on the same vertex. For example, even if two threads started with vertices *A* and *D*, they may both end up with *C* as the neighbor with the minimum cost edge. We must avoid a situation in which the algorithm tries to combine both *A* and *C* and *D* and *C*. One possible approach is to use unstructured locks with calls to *tryLock()* to perform the combining safely, but without creating the possibility of deadlock or livelock situations. A key challenge with calling *tryLock()* is that some fix-up is required if the call returns false. Finally, it also helps to use a concurrent queue data structure to keep track of nodes that are available for processing.

Optional Reading:

1. Wikipedia article on Borvka's algorithm for finding a minimum cost spanning tree of an undirected graph

Mark as completed