Project 4 - parallel graph exploration

• You have a graph with nodes and directed edges

• Want to find the set of nodes reachable from a start node, and perform a computation at each node

  • both the determination of the set of edges from a node, and the computation at that node, may take time

• Want to use a certain amount of parallelism to do that

  • not much more, not much less
I apologize for the goof ups with the project 4 setup

Should all be resolved.
Project 4

- Projects are coming fast and furious
- Project's aren't easy
- Projects aren't long
  - project 4 is about 60 lines of code
- You will get stuck
  - get started right away, allow time to think and/or get help
Implementation

- Don't use locks, synchronization, or thread creation
- Instead, use:
  - ExecutorService
  - FutureTask
  - ConcurrentHashMap
  - other classes from java.util.concurrent
Responsible task/thread

- Only one task/thread should be responsible for a node
  - if two arrive at the same time, only one gets responsibility

- Responsible node has to
  - initiate computation for the node
  - compute neighbors of the node
  - for each neighbor, initiate exploration of the neighbor
Deciding on responsibility

• Need to figure out unique task/thread that gets responsibility for node

• Use ConcurrentHashMap with Node as a key

• use putIfAbsent to decide if you are responsible
  • only one putIfAbsent can win

• Can just use map to a boolean value (e.g., true), which makes it really just a Set, or use a Map that you will also need for some other purpose
Futures and FutureTasks

• Want to initiate requests to compute values for nodes

• and provide an easy way to ask "wait until the value for this node is computed, and return the value that does get computed"

• For each node, create a FutureTask<V> to compute the value for that node

• have a map from nodes to FutureTasks
Concurrent Cache code

- Concurrent Cache code from Thursday is similar to what we need
- but not identical
public class Cache<K, V> {

    final ConcurrentHashMap<K, FutureTask<V>> map  
        = new ConcurrentHashMap<K, FutureTask<V>>() ;

    public V get(final K key) throws InterruptedException {
        FutureTask<V> f = map.get(key);
        if (f == null) {
            Callable<V> c = new Callable<V>() { public V call() {
                // return value associated with key
            }};
            f = new FutureTask<V>(c);
            FutureTask<V> old = map.putIfAbsent(key, f);
            if (old == null)
                f.run();
            else
                f = old;
        }
        try { return f.get(); }
        catch(ExecutionException ex) { throw ex.getCause(); }
    }
}
Initiating a future value computation

- If you create a FutureTask for a node and add it to a map, that doesn't start the computation for the FutureTask
- don't want to initiate it until we are sure that our future value is the one that is stored into the map
- Can give a FutureTask to an executor to have it be executed in parallel
Neighbors

• If we are the responsible task/thread for a node
• Compute the neighbors
  • no advantage to doing this in a separate thread
• For each neighbor, start an exploration of that neighbor (in parallel, using an executor)
One more thing

• Have to figure out when done
• When is it the case that all enqueued tasks have completed?
• Approach: Use a counter, initialized to zero
  • increment count before enqueuing task
  • decrement count when task finishes
    • if the decrement brings the count down to zero, we are done