Project 2 and 3 grading

- Posted
  - mistakes were made in grading
  - submit regrade request to tell us where
  - not email
Project 4 - Concurrency

• Concurrent graph explorer
• Due Wednesday, March 3rd
  • only one release token
• Posted to web, cvs and submit server
Project 5 - Concurrency

- Another concurrency project, TBA
- Due Tuesday, March 9th
Midterm

- Thursday, March 11th, in class
Web security exercise

• Craft a URL that, when opened, will make a signed in user add your linuxlab account to my guestbook on appengine

• I'll remove my CSRF forgery protection for GET requests

• Feel free to add more than just your name: e.g., cs433050 rocks

• Add URL to form for web security exercise

• Due Monday, March 1st
Open source contribution project

- Due March 12th
- Post a link to a web page describing your idea
  - form on web page
- Can be a google doc, a web page hosted somewhere, whatever
- If anyone wants to do a group effort, needs to be submitted by March 12th
- For solo projects, you need to submit an idea but you can change your mind
Cookies with the professor

- Wednesday, March 3rd, 2 - 2:30pm
- Friday, March 5th, 11 - 11:30am
- Girl scout cookies and godiva chocolates
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 ConcurrentMap<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
Design exercise

- Pair up
- In class design exercise
- Can be done on paper or in an IDE
- Feel free to ask questions about intended functionality or about Java APIs
CountDownLatch

• A CountDownLatch starts at a particular value, and can only be decremented

• Want to implement a CountDownLatch
  • has an initial value
  • can be incremented or decremented
  • if decremented to zero, is released
    • one released, released forever
  • can wait for it to be released
CountUpDownLatch

public class CountUpDownLatch {

    AtomicInteger count = new AtomicInteger();
    CountDownLatch done = new CountDownLatch(1);

    public void increment() {
        count.getAndIncrement();
    }

    public void decrement() {
        if (count.decrementAndGet() == 0)
            done.countDown();
    }

    public void awaitDecrementToZero() throws InterruptedException {
        done.await();
    }
}