CSE 332: Data Abstractions
Assignment #7
December 3, 2010
due: Friday, December 10, 2:30 p.m.
last update: December 7, 4:00 p.m.

1. Given a (very long) string T called the "text" and a (short) string P called the "pattern", the *string-matching problem* is to find substrings of T that are equal to P. Let n be the length of T. You can assume that the length of P is a constant. All of your algorithms below should have work $O(n)$ and span $O(\log n)$. Be sure to explain for each algorithm why this is true.

   (a) Describe a fork-join parallel algorithm that outputs the index of the leftmost occurrence of the pattern P in T, using a sequential cutoff of 1.

   (b) Describe a fork-join parallel algorithm that outputs an array of the indices of all occurrences of the pattern P in T, using a sequential cutoff of 1. (Hint: use Pack.)

   (c) What changes do you need to make to your solution in part (a) to use a more sensible sequential cutoff?

2. Consider this Java implementation of a queue with two stacks. We do not show the entire stack implementation, but assume it is correct. Notice the stack has synchronized methods but the queue does not. The queue is incorrect in a concurrent setting.

```
class Stack<E> {
  ...
  synchronized boolean isEmpty() { ... }
  synchronized E pop() { ... }
  synchronized void push(E x) { ... }
}
```

```
class Queue<E> {
  Stack<E> in  = new Stack<E>();
  Stack<E> out = new Stack<E>();
  void enqueue(E x){in.push(x); }
  E dequeue() {
    if(out.isEmpty()) {
      while(!in.isEmpty()) {
        out.push(in.pop());
      }
    }
    return out.pop();
  }
}
```

   (a) Show the queue is incorrect by showing an interleaving that meets the following criteria:

      i. Only one thread ever performs `enqueue` operations and that thread enqueues numbers in increasing order (1, 2, 3, ...).

      ii. There is a thread that performs two `dequeue` operations such that its first `dequeue` returns a number larger than its second `dequeue`, which should never happen.

      iii. Every `dequeue` succeeds (the queue is never empty).

   Your solution can use 1 or more additional threads that perform `dequeue` operations.

   (b) A simple fix would make `enqueue` and `dequeue` synchronized methods, but this would never allow an `enqueue` and `dequeue` to happen at the same time. To allow an `enqueue` and a `dequeue` to operate on the queue at the same time (at least when `out` is not empty), we could try either of the approaches below for `dequeue`. For each, show an interleaving that demonstrates the approach is broken. Your interleaving should satisfy the three properties listed in part (a).

```
E dequeue() {
  synchronized(out) {
    if(out.isEmpty()) {
      while(!in.isEmpty()) {
        out.push(in.pop());
      }
    }
    return out.pop();
  }
}
```

```
E dequeue() {
  synchronized(in) {
    if(out.isEmpty()) {
      while(!in.isEmpty()) {
        out.push(in.pop());
      }
    }
   }
   return out.pop();
}
```

(c) Provide a solution, based on two stacks as above, that correctly allows an `enqueue` and a `dequeue` to operate on the queue at the same time, at least when `out` is not empty. Your solution should define `dequeue` and involve multiple locks.

3. (a) Simulate Kruskal's algorithm to find a minimum cost spanning tree T of the graph of Figure 9.82 in the textbook. Each time an edge e is added to T, redraw the current forest T and the full collection of up-trees resulting from adding e to T. Each time an edge e is considered but not added to T, list the `find` operations that explain why e was not included in T. Use weighted unions (called union-by-size in Section 8.4 of the textbook), but without path compression. Whenever you have a tie in the weighted union, that is, you are merging two up-trees that have exactly the same size, break such ties by making the root whose vertex comes earlier in the alphabet the parent of the root whose vertex comes later in the alphabet. For instance, if you are taking the union of two sets whose up-trees have roots B and E, and each of these up-trees has 2 nodes in it, then break the tie by making B the parent of E.

(b) What is the cost of the minimum spanning tree that you found?