CSE 2320 Lab Assignment 2

Due October 19, 2010

Goals:

1. Understanding of heaps as priority queues along with linking a PQ with a dictionary.

2. Understanding of unweighted interval scheduling when generalized from one "room" to an arbitrary number (k), commonly known as the "maximum cardinality k-coloring problem for n intervals".

Requirements:

1. Design, code, and test a Java program to assign (reusable) colors 1 . . k to a set of intervals such that overlapping intervals are never assigned the same color. In some cases an interval will not be assigned a color (i.e. it is rejected). Your program should be based on the pseudocode in Getting Started #2. The first line of the input will be k and n, the number of available colors and the number of (integer) interval pairs in the remaining input lines. Each of the intervals $[x, y)$ will have $0 < x < y$. The input should be read from standard input using `java.util.Scanner`. The first line of the output is k, n, and the number of intervals that were successfully colored. Each of the remaining n output lines will give one input interval pair and its color. If an interval was not colored, then output 0 as its color value. The input and output orderings of the intervals may be different. ***Since your output will be checked by another program, this format must be followed***. Your code should take $\Theta(n \log n)$ time.

2. Email your source files (as attachments) to `huawang2007@mavs.uta.edu` by 9:15 a.m. on October 19. The `Subject` should be your name as recorded by the University and you should `cc:` yourself to verify that you sent the message correctly.

Getting Started:

1. Notes 6.B gives a simple greedy strategy that applies when $k = 1$. That strategy sorted the intervals $[x\ y)$ by their y values. For this assignment, you are to use the preprocessing strategy of separating the n intervals into $2n$ endpoints that are then sorted and processed left-to-right. (Note that if both $[x$ and $x)$ appear, $x)$ will be before $[x$ in the sorted order.) For the one-room problem, this may be formalized as:

```
Initially, there is no currentInterval [a, b)
for each endpoint in left-to-right order
  if endpoint is [x
    if there is no currentInterval
      [x and its matching right endpoint y) become the currentInterval
    else if the matching right end for [x is y) and y < b
      currentInterval [a, b) is rejected
      [x, y) becomes the currentInterval
    else
      [x, y) is rejected
  else // processing some right endpoint y)
    if [x, y) is the currentInterval
      Change status to having no currentInterval (but [x, y) remains permanently
        in schedule)
    else
      // y) is ignored (its interval was rejected earlier)
```

NOTE: If duplicate endpoints do not occur, this strategy gives the same result as the greedy strategy in Notes 6.B.

2. Your program is to generalize #1 to arbitrary *k* by using the following pseudocode:

```
Initially, there are no currentIntervals and all k colors are available
for each endpoint in left-to-right order
  if endpoint is [x
    if there are fewer than k currentIntervals
      [x and its matching right endpoint y) become a currentInterval using
        any available color
    else if the matching right end for [x is y), b) is the latest right endpoint
           over all of the k currentIntervals, and y < b
      currentInterval [a, b) is rejected and [x, y) becomes a currentInterval by
        stealing the color from [a, b)
    else
      [x, y) is rejected
  else // processing some right endpoint y)
    if [x, y) is a currentInterval
      Change status to having one less currentInterval (but [x, y) permanently
        retains its color)
    else
      // y) is ignored (it was rejected earlier)
```

3. To code an efficient program, the following additional data structures are useful:

   a. A stack (e.g. an `int` table and a stack pointer) for the available colors.
   b. A table (e.g. dictionary) indicating which current interval is assigned to each color.
   c. A maxHeap for finding the current interval with the latest right endpoint (e.g. the priority). The minHeap on the course webpage may be adapted (`insert`, `maximum`, `extractMax`, `deleteId` will be needed). Since handles are already included, you will find it convenient to use the colors assigned to intervals as the `id`s for items in the maxHeap.

4. The code for $k = 1$ (`oneRoom.java`) is available on the course webpage.

Example with *k*=3