**CS61B, Fall 2011** **HW #10** **P. N. Hilfinger**

**Due:** Mon., 21 November 2011

**1.** The skeleton in `hw10` in the staff repository contains a copy of the project 3 `graph` directory. Fill in enough of it to do the following problems. [Warning: if you want to use any of your hw10 solution in project 3, please copy it to a proj3 directory. Do *not,* as several did last time, keep working on project 3 in your `hw10` directory! That just asks for trouble, and will cause your instructor to become Peeved if you submit bug project 3 bug reports from a hw10 solution.]

**2.** Scheme (like Common Lisp) uses pairs to represent tree and graph structures. The S expression `(`$S_1$ `.` $S_2$`)` represents a pair whose first ('car') element if $S_1$ and whose second ('cdr') is $S_2$. The shorthand `(`$S_1$ $S_2$ $\cdots$ $S_n$`)` means

`(`$S_1$ `.` `(`$S_2$ `.` `(`$\cdots$ `.` `(`$S_n$ `()))`$\cdots$`))`

where '()' denotes the null (nil) value. Scheme has a notation for denoting general graph structures, including circular ones. To represent the circular list `(1 2 1 2 1 2 ...)`, for example, one can write

`#1=(1 . (2 . #1#))` *or* `#1=(1 2 . #1#)`

The `#`$n$`=`$S$ notation (where $n$ is a non-negative integer numeral and $S$ is an S-expression) defines `#`$n$`#` to be a pointer to object that the reader returns for $S$. The notation

`(1. (#1=(2) . (#1# . ())))` *or* `(1 #1=(2) #1#)`

denotes a DAG structure like this ('/' is null):

```
[ 1 . *]--->[ * . *]---->[ * . /]
             |            |
             +------------+
             |
             V
             [ 2 . /]
```

Fill in the skeleton in `hw10/CircPrint.java` to print out structures using this notation (the unabbreviated dot notation, that is). It uses labeled edges to denote "car" and "cdr" links and vertex labels to indicate the contents of atoms.

**3.** Fill in the following program to obey its comment.

```
/** Returns the result of non-destructively riffle-shuffling the
 *  two lists of T L1 and L2 together, using the
 *  Gilbert-Shannon-Reeds model, with R supplying pseudo-random
 *  numbers (so that shuffling the same lists twice with a
 *  generator starting in the same state both times results in the
 *  same list).
 */
static <T> List<T> riffle(List<T> L1, List<T> L2, Random R) {
    // FILL IN
}
```

The Gilbert-Shannon-Reeds model proceeds as follows (conceptually): at each step, we remove the head of `L1` and add it to the result with probability $A/(A+B)$ and the head of `L2` to the result with probability $B/(A+B)$, where $A$ and $B$ are the remaining lengths of `L1` and `L2`, respectively. The actual program you write is to produce the result without modifying the lists referenced by `L1` and `L2`.

**4.** [M. Dynin, from a contest in St. Petersburg] Given a rectangle of letters (such as

```
AB
BC
```

), a starting position within that rectangle (such as the character in the upper-left corner), and a string (such as `"ABBC"`), consider the question of finding paths through the letters that match the given string, begin at the starting position, and at each step move one square in one of the eight compass directions (north, south, east, west, northeast, northwest, southeast, southwest). For the given example, there are two such paths. They are (`E-SW-E`) and (`S-NE-S`)—that is, "one step east, one step southwest, one step east" and "one step south, one step northeast, and one step south." For the rectangle

```
CBB
BBA
```

and the string `"BBCBBA"`, starting at square in the middle of the top row, there are 12 paths. Paths are allowed to visit the same position twice.

Using the template in `hw10/CountPaths.java`, you are to write a program that, given such a rectangle, string, and starting position, reports the *number* of distinct paths that match the string, according to the definitions above. The input (on the standard input, System.in) will consist of four positive integers (call them $M$, $N$, $r$, and $c$), a string ($S$), and $M$ strings consisting of upper-case letters `A-Z`, each of which is $N$ characters long. These inputs are all separated from each other by whitespace. The $M$ strings are the rows of the rectangle, from top to bottom. The pair $(r, c)$ are the coordinates of the starting position, with $0 \le r < M$, $0 \le c < N$. Row 0 is the top row; column 0 is the left column. The output will be a line reporting the number of paths in the format shown in the examples.

You may assume that the number of paths is less than $2^{31}$, $M \le 80$, $N \le 80$. Nevertheless, be aware that the time limit is about 15 seconds.

**Example 1:**

| Input | Output |
|---|---|
| <pre>2 2 0 0<br>ABBC AB<br>     BC</pre> | `There are 2 paths.` |

**Example 2:**

| Input | Output |
|---|---|
| `2 3 0 1 BBCBBA CBB BBA` | `There are 12 paths.` |