CS61B, Fall 2012 **HW #9** **P. N. Hilfinger**

Due: Wed., 21 November 2012

1. The file `TrieSet.java` in the `hw9` staff code directory contains a skeleton for a set-of-Strings class For this exercise, we will *not* worry about compressing the nodes. Fill in the `contains` methods in the skeleton to comply with its comment. You will also have to pick a representation for `InnerTrieNode` and complete its constructor. Then fill in the `insert` methods to comply with their comments.

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 is S_1 and whose second ('cdr') is S_2. The shorthand `(`S_1 S_2 $\cdots$ S_{n-1} `.` S_n`)` means

`(`S_1 `.` `(`S_2 `.` `(`$\cdots$ `.` `(`S_{n-1} `.` S_n`))`$\cdots$`))`

and `(`S_1 S_2 $\cdots$ S_n`)` is short for

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

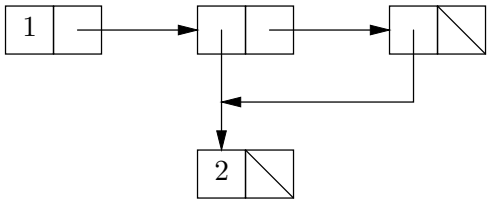
where '()' denotes the null (nil) value. Scheme has a notation for denoting general graph structures, including circular ones. To represent the circular list `(1 2 1 2 1 2 ...)`, for example, where the tail of the second item is the beginning of the list, one can write

`#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 an 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):



Fill in the skeleton in `hw9/CircPrint.java` to print out structures using this notation.

3. Fill in the `riffle` method in `Shuffle.java` to non-destructively riffle-shuffle two lists of L1 and L2 together, using the Gilbert-Shannon-Reeds model. 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 `hw9/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

`2 2 0 0`
`ABBC AB`
`BC`
	`There are 2 paths.`

Example 2:

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