Project 2: Tent Tree Game

DUE: Sunday, March 18th at 11:59pm

Basic Procedures

You must:

- Fill out a readme.txt file with your information (goes in your user folder, an example readme.txt file is provided)
- Have a style (indentation, good variable names, etc.)
- Comment your code well in JavaDoc style (no need to overdo it, just do it well)
- Have code that compiles with the command: javac *.java in your user directory
- Have code that runs with the command:
  `java PA2 puzzles/puzzleFILE.txt`

You may:

- Add additional methods, fields, and classes, however these must be private (or package default for fields/methods inside nested classes).

You may not:

- Make your program part of a package.
- Add additional public methods, variables, or classes.
- Use any built in Java Collections Framework classes anywhere in your program (e.g. no ArrayList, LinkedList, HashSet, HashMap, etc.).
- Alter any method signatures defined in this document or the template code.
- Add any additional libraries/packages which require downloading from the internet.

Setup

- Download the project1.zip and unzip it. This will create a folder `section-yourGMUUserName-p2`
- Replace "`section`" with `z001, z002, k003`, and `z004` for the 4 sections respectively.
- Rename the folder replacing `yourGMUUserName` with the first part of your GMU email address/netID. Example: `k003-jkrishn2-p2`
- Complete the `readme.txt` file (an example file is included)

Submission Instructions

- Make a backup copy of your user folder!
- Remove all test files, jar files, class files, etc.
- You should just submit your java files and your readme.txt
- Zip your user folder (not just the files) and name it as "section-`yourGMUUserName-p2.zip`" (no other type of archive) where "`yourGMUUserName`" is your GMU email address / netID.
- Submit to blackboard.

Grading Rubric

Due to the complexity of this assignment, an accompanying grading rubric pdf has been included with this assignment. Please refer to this document for a complete explanation of the grading.

Topics Covered

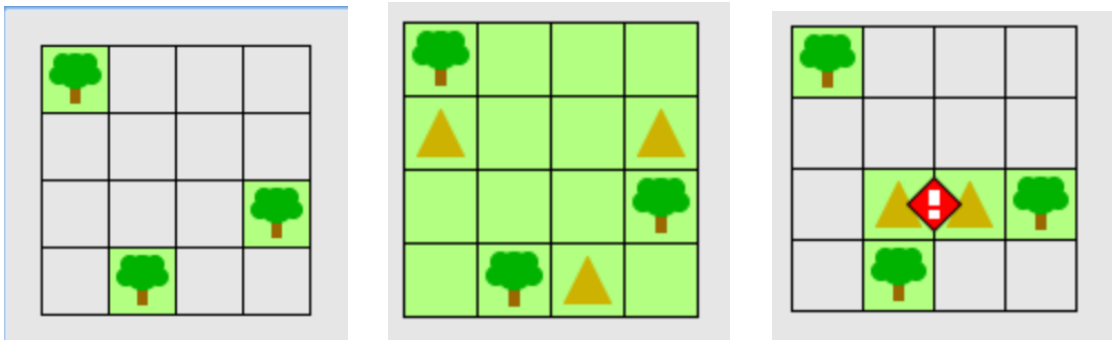
Linked Lists, Hashing

Overview

You are going to build a simplified version of **`Tree-Tent`** puzzle using Hashing. The main idea is to place tents on a 2D grid with initial placement of trees to meet all these conditions:

1. Every tree must be connected to a tent;
2. Every tent must be attached to a tree;
3. No two tents are adjacent to each other, neither horizontally, vertically, or diagonally.

For two things to be connected or attached, they must be adjacent either horizontally or vertically. For example, given (A) as an initial puzzle, (B) is a valid solution while (C) is not since one tree is missing a tent and also two tents are touching each other.



(A) (B) (C)

(Picture src: http://www.chiark.greenend.org.uk/~sgtatham/puzzles/js/tents.html)

There are many similar websites providing helpful description and online playing. For example: http://www.brainbashers.com/tentshelp.asp has a good explanation of rules. Notice we are implementing a simplified version for this project. You do NOT need to support the additional restriction of how many tents are allowed to be placed on each row/column.

Implementation

We are using a hash map to implement the 2D grid for our game. The map stores <cell, symbol> pairs for all non-empty cells of our puzzle. Suppose every cell on the board with N rows and M columns is represented with a (row_index, column_index) as shown to the right. For every cell at (r,c) that is occupied by a symbol S, we will have a pair <(r, c), S> in our hash map. For example, the board represented in Figure (A) above should include three entries as {(<0,0>,Tree), (<2,3>,Tree),(<3,1>, Tree)} (not necessarily in that order). All puzzle moves (adding a tent, removing a tent, checking a tent, etc) are therefore performed via hash map operations.

(0,0)	(0,1)	…	(0,M-1)
(1,0)	…	…	(1,M-1)
…	…	…	…
(N-1,0)	(N-1,1)	…	(N-1,M-1)

We are following the idea described in Ch6.8 of our textbook to implement the hash map using a set with a <key, value> pair. What is special is that equals/hashCode implementations of the <key, value> pair is based on the key only. The class implementing the hash map (HashMap.java) is implemented and provided to you. But you will need to implement the underlying set which keeps a unordered collection of no duplicates and supports fast retrival/searach/removal. This must be

implemented as a hash table with separate chaining to resolve collision in HashTable.java. You must also implement your own linked list to use in the hash table as SimpleList.java.

Classes Overview

There are a total of **6** classes in the project but two of them are fully implemented and provided to you. Make sure you check the project package for required methods and additional details.

Position (Position.java): This class represents the cell position in a 2-D grid. In addition to the normal accessors, you also need to implement .equals and .hashCode to compare two positions and to return an integer hash code. Remember to follow hash contract and pick a hash function that is easy to compute and distribute well.

SimpleList(SimpleList.java): This class is a generic linked list that you will use to implement separate chaining in hash table. You can choose how the linked list is organized as far as the required operations are supported with the specified runtime overhead. You also need to define a basic iterator for this class and implement efficient .hasNext() and .next() in it.

HashTable(HashTable.java): This class is a generic hash table using separate chaining to resolve collision. The table is initialized as an array of 11 spots but may need to be expanded. In addition to the load, we will also be monitoring the average chain length of all active buckets. You are required to expand and rehash to a bigger hash table if the average chain length is > 1.2. The new table size must be a prime number larger than twice the size before. A .nextPrime() is provided for you to use. *Definition*: **Average chain length** = number of elements / non-empty active chains.

HashMap(HashMap.java): This class is a generic hash map (dictionary) implemented using HahTable.java. This class is fully written and provided to you, including a Pair class that makes sure both .equal and .hashCode are determined only by the key of a <key, value> pair. Read the code to understand how HashTable class will be used.

TentTree: This class is the primary class representing the tent-tree puzzle. It consists of the tent-tree board stored as an instance of HashMap class. The class contains methods to add and remove tents, to check whether at least one of the 4-way or 8-way neighbors of the specified position has the given symbol, to check if the puzzle has solved or not. Make sure to utilize the fast retrieval/search of the hash map to implement the operations when possible. A .toString() is provided to you to help testing and debugging. The puzzle status checking is considered as **extra credit** (5%).

PA2: This class provides a basic user interface for you to play the game interactively once your implementation is in place. This class is fully written and provided to you. It accepts a .txt file to initialize the board. The user can then add/remove tents to find a solution of the puzzle. Even if you do not implement the extra credit part for status checking, you can still play (and maybe manually verify whether you are done). A set of puzzle files are provided to you for testing purpose.

Big-O

Template given to you in the starter package contains instructions on the REQUIRED Big-O runtime for each method. Your methods should not have a higher Big-O and you will be graded on this.

Testing

Test cases will not be provided for this project. However, feel free to create test cases by yourselves. In addition, the main methods provided along with the template classes contain useful code to test your code. You can use command like "`java Position`" to run the testing defined in main( ). You could also edit main( ) to perform additional testing. And yes, a part of your grade will be based on automatic grading using test cases that are not provided to you.

EXAMPLE RUN #1

```
java PA2 puzzles/puzzle1.txt
    -     -     -
    0     -     0
    -     -     0


Please select what you'd like to do:
1) Exit
2) Add a tent
3) Remove a tent
4) Check whether the puzzle has been solved
2

Please select where you'd like to add the tent (X):
which row (0-2):
0
which col (0-2):
0
    X     -     -
    0     -     0
    -     -     0


Please select what you'd like to do:
1) Exit
2) Add a tent
3) Remove a tent
4) Check whether the puzzle has been solved
4
```

```
unfinished puzzle: tree missing tent!
    X     -     -
    0     -     0
    -     -     0


Please select what you'd like to do:
1) Exit
2) Add a tent
3) Remove a tent
4) Check whether the puzzle has been solved
2

Please select where you'd like to add the tent (X):
which row (0-2):
0
which col (0-2):
2
    X     -     X
    0     -     0
    -     -     0


Please select what you'd like to do:
1) Exit
2) Add a tent
3) Remove a tent
4) Check whether the puzzle has been solved
2

Please select where you'd like to add the tent (X):
which row (0-2):
2
which col (0-2):
0
    X     -     X
    0     -     0
    X     -     0


Please select what you'd like to do:
1) Exit
2) Add a tent
3) Remove a tent
```

```
4) Check whether the puzzle has been solved
3

Please select where you'd like to remove a tent (X):
which row (0-2):
2
which col (0-2):
0
    X     -     X
    0     -     0
    -     -     0


Please select what you'd like to do:
1) Exit
2) Add a tent
3) Remove a tent
4) Check whether the puzzle has been solved
2

Please select where you'd like to add the tent (X):
which row (0-2):
2
which col (0-2):
1
    X     -     X
    0     -     0
    -     X     0


Please select what you'd like to do:
1) Exit
2) Add a tent
3) Remove a tent
4) Check whether the puzzle has been solved
4
Puzzle solved!
Congratulations!!!
```

EXAMPLE RUN #2

```
java PA2 puzzles/puzzle1.txt
    -     -     -
    0     -     0
    -     -     0


Please select what you'd like to do:
1) Exit
2) Add a tent
3) Remove a tent
4) Check whether the puzzle has been solved
3

Please select where you'd like to remove a tent (X):
which row (0-2):
0
which col (0-2):
0
Cannot remove tent(X) at row 0 col 0!
    -     -     -
    0     -     0
    -     -     0


Please select what you'd like to do:
1) Exit
2) Add a tent
3) Remove a tent
4) Check whether the puzzle has been solved
2

Please select where you'd like to add the tent (X):
which row (0-2):
1
which col (0-2):
0
Cannot add a tent (X) at row 1 col 0!
    -     -     -
    0     -     0
    -     -     0


Please select what you'd like to do:
1) Exit
```

```
2) Add a tent
3) Remove a tent
4) Check whether the puzzle has been solved
```