CSC A48 Assignment 1

Snakes And Ladders

Due: Sunday, February 12, 2017 (anytime on this day is OK).

⋆ introduction

The game of *Snakes And Ladders* is well known. You have probably played something like it before (google it if you are not familiar with it). We will use a variation of this game for our assignment, which is mostly about linked lists.

◯ our variation of snakes and ladders

⊙ snakes and ladders board

Our game is played on a board consisting of n squares, where n can be any positive integer. The squares are numbered $1, 2, 3, \cdots, n$. There are usually snakes and ladders on the board. A snake links a higher numbered square to a lower numbered square. A ladder links a lower numbered square to a higher numbered square.

Since we are computer scientist, who are inclined to make abstractions, we simply view both a snake and a ladder as a link from one square, called the *source*, to a different square, called the *destination*.

We use *snadder* to refer to either a snake or a ladder. So a snadder is link from its source to its destination.

Additional restrictions:

- No square can be the source of more than one snadder.
  The reason for this should become clear when you know how the game is played.
- No square can be the destination of more than one snadder.
  The reason for this should become clear when you see the concept of a dual board later.
- The last square cannot be the source or the destination of a snadder.

Aside: The board described here is essentially the same as that of the conventional game of snakes and ladders.

⊙ how it's played

Our game is played by two players, called player 1 and player 2. After creating the board on which to play, player 1 randomly generates a step size, which is a positive integer. Then player 2 does likewise. So player 1 has a step size, say s_1, and player 2 has a step size, say s_2. s_1 and s_2 may or may not be equal.

Player 1 starts on square s_1, unless that square is the source of a snadder, in which case player 1 starts at the destination of that snadder. Similarly, player 2 starts on square s_2, unless that square is the source of a snadder, in which case player 2 starts at the destination of that snadder.

Now the players take turns making moves, with player 1 going first. A move for player i, where $i = 1$ or $i = 2$, consists of moving s_i squares from the square where the player was. If the player lands on the source of a snadder, then the player *follows* the snadder and ends the move at the destination of that snadder.

Wraparound rule: If the player was less than s_i squares from square n (the last square), then the player moves the excess number of squares starting back at square 1.
For example, if $n = 100$, $s_1 = 5$, and player 1 was on square 97, then player 1 goes to square 2 ($97 + 5 - 100 = 2$). Furthermore, if square 2 is the source of a snadder with destination square 25, then player 1 would end the move at square 25.

⊙ **who wins**

The first player to end a move on the last square wins the game.

⊙ **when players loop**

It is possible that neither player ever moves onto the last square. They can both get into a *loop*, and they never end a move on the last square. In this case, player 2 is considered the winner.

Aside: The conventional game of snakes and ladders differs from our game in three ways.

1. Instead of using a fixed step size, the conventional game uses a randomly generated step size for every move. A die is commonly used.
2. The conventional game does not have the wraparound rule. Instead, the first player to go on or past the last square wins.
3. With random step size for every move, the concept of a player looping as described above cannot happen. The conventional game simply continues until a player go on or past the last square.

⋆ ways to represent a snakes and ladders board

Here are two ways to represent a snakes and ladders board.

◯ **dictionary representation**

We can represent a snakes and ladders board with a data structure that has two attributes.

`numSquares` An integer indicating the number of squares on the board.

`snadders` A dictionary where each key is the source of a snadder and its value is the corresponding destination. For example, if a snadder links square 23 to square 71, then `snadders[23] == 71`.

See the file `salboard.py` for details.

◯ **linked list representation**

We can also represent a snakes and ladders board as a circular singly linked list. Each node represents a square. The last node represents the last square. It is linked to the first node, which represents square 1. If there are n squares on the board, then there are n nodes in the linked list. As with any linked list, a pointer to the first node is all we need to have access to the whole list. The number of squares is not explicitly stored anywhere.

The only data stored in each node is the attribute `snadder`. If the node represents the source of a snadder, then `snadder` contains the address of the corresponding destination. Otherwise `snadder == None`.

See the file `salbnode.py` for details.

⋆ the dual board

Given a snakes and ladders board, we define its *dual* to be the snakes and ladders board with the same number of squares, and the same number of snadders, except the source and destination of each snadder are interchanged. For example, if the original board has 100 squares, then so does its dual. Furthermore, if square 3 links to square 8 in the original board, then square 8 links to square 3 in its dual.

⋆ what to do

1. Download the following files.

   - `salboard.py`
     It contains the declaration of the `SALboard` class, which is the dictionary representation of a snakes and ladders board. Read and understand this code.
   - `salbnode.py`
     It contains the declaration of the `SALBnode` class, which is the node used in a linked list representation of a snakes and ladders board. Read and understand this code.
   - `salbLLfunctions.py`
     To this file add your name to the license at the top to indicate that you have added intellectual value to it. You may only distribute these files, or modified versions of them, with the same license, and along with the file `GNUlicense`.

2. In `salbLLfunctions.py`, implement the following module-level functions, with good documentation.

   `salb2salbLL(salb)`: which takes a dictionary representation of a snakes and ladders board, and returns a linked list representation of the same board.

   `willfinish(first, stepsize)`: which takes a linked list representation of a snakes and ladders board and a step size, and returns whether a player playing on the board with the given step size will ever land on the last square.

   `whowins(first, step1, step2)`: which takes a linked list representation of a snakes and ladders board and two step sizes for players 1 and 2 respectively, and returns the number of the player who wins the game on the given board with players using their given step sizes.

   `dualboard(first)`: which takes a linked list representation of a snakes and ladders board, and returns the linked list that represents its dual.
   Your code must ensure that the input list represents the same board when the function is called and when it returns. However you may alter the input list while the function is running.

   Since this assignment is meant mainly as an opportunity for you to work with linked lists, therefore we impose the following restriction on your code.

   - *For all the above functions except* `salb2salbLL`, *you are not permitted to use the built-in Python data structures lists, tuples, sets, or dictionaries. You must work directly with* `SALBnode`*s.*

3. Submit your `salbLLfunctions.py` file to Markus.