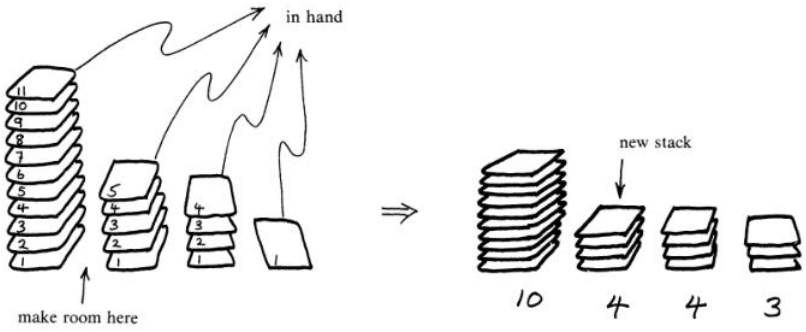
CS 7A - Spring 2015 - Bulgarian Solitaire. Due 2/26/15

Background Theory

Bulgarian Solitaire is played with a deck of N cards. The first thing you do is to divide the N cards into one or more stacks. For example, 21 cards could be divided into four stacks consisting of 11, 5, 4, and 1 cards. Now you can play the game. Take one card from each stack. Since it doesn't matter which card you take, you might as well take the top card. In the example you would get four cards. Put these cards down on the table to form a new stack. This gives you four stacks consisting of 10, 4, 4, and 3 cards. If you do this 24 more times you get six stacks consisting of 6, 5, 4, 3, 2, 1 cards. This is called the winning position because it repeats with any subsequent move.



In general, the game of Bulgarian solitaire is considered to be won if you get k stacks consisting of $k, k-1, k-2, \ldots, 3, 2, 1$ cards. This means you cannot win the game unless you start with $N = 1+2+3+\cdots+k = k(k+1)/2$ for some k. These numbers are called triangular numbers. In his paper on Bulgarian Solitaire, Kiyoshi Igusa uses the notation $T_k = \dfrac{k(k+1)}{2}$.

Use pencil and paper to consider various scenarios with $k = 6$ and $T_6 = 21$. The Bulgarian solitaire conjecture says that any game with 21 cards can be won in $k(k-1) = 30$ moves or less. We win in 25 moves; however the initial division of 21 cards into seven stacks with 5,5,4,3,2,1,1 cards requires 30 moves.

1. Write code to meet these specifications:
   Precondition: A user supplied sequence of numbers in decreasing order (such as 5,5,4,3,2,1,1) which sum to some triangular number, T_n.
   Postcondition: A sequence of Bulgarian Solitaire moves printed to the console such as

```
Enter a sequence of positive numbers. Enter 0 when done:
1 1 2 3 4 5 5 0
1 2 3 4 4 7
1 2 3 3 6 6
1 2 2 5 5 6
1 1 4 4 5 6
3 3 4 5 6
2 2 3 4 5 5
1 1 2 3 4 4 6
1 2 3 3 5 7
1 2 2 4 6 6
1 1 3 5 5 6
2 4 4 5 6
1 3 3 4 5 5
2 2 3 4 4 6
1 1 2 3 3 5 6
```

```
1 2 2 4 5 7
1 1 3 4 6 6
2 3 5 5 6
1 2 4 4 5 5
1 3 3 4 4 6
2 2 3 3 5 6
1 1 2 2 4 5 6
1 1 3 4 5 7
2 3 4 6 6
1 2 3 5 5 5
1 2 4 4 4 6
1 3 3 3 5 6
2 2 2 4 5 6
1 1 1 3 4 5 6
2 3 4 5 7
1 2 3 4 5 6
```

Here's some starter code that may (or may not) be helpful:

```
//<your name here>
//Working the babylonian algorithm

///prototypes
bool endState(vector<unsigned> v);
///Return true only if the input is a vector of the form {1,2,3,...,k}

void show_stacks(vector<unsigned> v);
///Print the vector v

void show_history(vector< vector<unsigned> > history);
///Print the sequence of vectors in history

bool is_new(vector<unsigned> v, vector< vector<unsigned> > history)
///Compare the vector v (the latest addition to history)
///with all the previous vectors in history.  If there's no match
///return true.  If there is a match, return false

void bulgarianMove(vector<unsigned>& v);
///Perform the Bulgarian Solitaire move on v
///note that, since v is passed by reference, it will change v.

int main() {
    vector<unsigned> vStacks;
    vector< vector<unsigned> > playHistory;
///Get the user's input for the initial vector of stack sizes
///and start building playHistory with the first vector.

///loop the Bulgarian Solitaire move, building the playHistory
///and checking at each loop
///for endState(vStacks)) and is_new(vStacks,playHistory)

///When the a root loop is found (or an end state), display
///the number of moves it took to get there and the size of the root loop.
}
```

Then go further to report the number of moves to a loop and the length of the loop. Here are a few typical runs of the code I wrote for this:

```
Enter a sequence of positive numbers. Enter 0 when done:
1 1 2 3 4 5 0

The root loop size is = 6
It takes 6 moves to get to the steady state:
1 1 2 3 4 5
1 2 3 4 6
1 2 3 5 5
1 2 4 4 5
1 3 3 4 5
2 2 3 4 5
1 1 2 3 4 5


================================================================

Enter a sequence of positive numbers. Enter 0 when done:
7 6 5 4 0

The root loop size is = 7
It takes 26 moves to get to the steady state:
7 6 5 4
3 4 4 5 6
2 3 3 4 5 5
1 2 2 3 4 4 6
1 1 2 3 3 5 7
1 2 2 4 6 7
1 1 3 5 6 6
2 4 5 5 6
1 3 4 4 5 5
2 3 3 4 4 6
1 2 2 3 3 5 6
1 1 2 2 4 5 7
1 1 3 4 6 7
2 3 5 6 6
1 2 4 5 5 5
1 3 4 4 4 6
2 3 3 3 5 6
1 2 2 2 4 5 6
1 1 1 3 4 5 7
2 3 4 6 7
1 2 3 5 5 6
1 2 4 4 5 6
1 3 3 4 5 6
2 2 3 4 5 6
1 1 2 3 4 5 6
1 2 3 4 5 7
1 2 3 4 6 6
```

Submit the code for #1 using your initials in the usual format: say GH_bulgarian.cpp