Appendix: Input Format

To allow for comparison and testing, we will use an incredibly simple input format. Its simplicity is from the point of view of parsing into your program, it may not be the clearest for reading as a human. The format is as follows, with an example below:

- We will represent cards as integers from 1 to 52 (for a standard deck) or 1 to number of suits times number of ranks in each suit for a variant deck.
- Each suit is a consecutive sequence of cards. E.g. in a standard deck we will represent the Spades by 1 to 13, the Hearts by 14 to 26, Diamonds by 27 to 39, and Clubs by 40 to 52. Aces are considered to be 1, Jacks 11, Queens 12 and Kings 13. This may be a bit confusing but fortunately it is not important. For this practical you are not required to convert between regular cards and integers. Instances are supplied as sets of integers.
- The first three numbers represent, in order: the number of ranks in each suit; the number of suits in the deck of cards, and the number of cards in the layout, i.e. the value of n.
- The remaining numbers give the list of cards. This list should contain at least as many numbers as the third number n, and each card should be between 1 and ranks*suits. No card should be repeated.
- After the nth card is specified all subsequent text is ignored, so does not invalidate a correct solution.
- A simple reader is provided in Java for reading this input format. Note that this reader ignores line breaks. This means that the input would be valid even if it was all in one line or alternatively if every number was on a separate line. If you are creating your own instances you can format them how you wish.

For example, the unsolvable layout from earlier, $10\clubsuit, A\heartsuit, 2\clubsuit, 2\spadesuit, A\spadesuit$, could be given by this input:

```
13 4 5
49 14 41
2 1
Some text which is ignored can follow.
```

As another example, the simple solvable problem from earlier, $2\heartsuit, 3\heartsuit, 4\heartsuit, 4\diamondsuit$ could be entered:

```
13 4 4
15 16 17 30
```

Note that any problem with 1 card is guaranteed to be solvable, e.g.

```
13 4 1
51
```

As a final example, note that we could have many more suits and cards correctly, e.g.

```
100 10 7
991 992 993 994 995 996 997
This is definitely solvable since all cards are the same suit
```

Appendix: Output Format

To simplify checking we are requiring the following output format, but it only applies to the FIRST line of your output. Following that, you may choose to add any additional output that you wish. This could be useful e.g. if you want to record debugging information or evaluation information like time taken and number of nodes searched. These can be freely added on any line after the first.

The FIRST line of your output should be a sequence of integers as follows:

- It should start with -2, -1, or a non-negative integer (0 or higher).

  -1 indicates that your program proved that the game was impossible to win. The line should then end.

  -2 indicates that your program could not determine whether the game was winnable or not. The line should then end.

  A non-negative integer indicates a winning solution, where the first integer is the number of moves in the solution and the following numbers give the solution itself as follows:

  – The solution is given as two numbers for each move.
  – The first number in each pair is the card that was moved, represented by its integer.
  – The second number represents the pile number that the card was moved to. The piles are counted starting from 0.
  – Piles are numbered as they exist at the time the move is made.

As an example, one solution for the simple solvable puzzle from earlier, $2\heartsuit, 3\heartsuit, 4\heartsuit, 4\diamondsuit$ could be:

```
3 17 1 17 0 30 0
This is a correct solution for our example puzzle
```

In this three move solution we move the $4\heartsuit$ onto $3\heartsuit$ and then $2\heartsuit$, and finally we can move the $4\diamondsuit$ onto the $4\heartsuit$ leaving just one pile. Note that even though the card 30 was moved onto card 17 as the final move, we refer to the 17 by its position at that time (0) rather than its original pile number (2).

An example of an *incorrect* solution for the same instance of Late-Binding Solitaire would be:

```
3 30 2 16 0 30 0
This is not a solution of LBS but does work for LBS with Saving Grace
```

This solution is incorrect for LBS because the last move attempts to move the $4\diamondsuit$ onto the $3\heartsuit$ as the last move, which is illegal. However, it is a correct solution of LBS with Saving Grace. In that case the final move is legal as it is the first and only time we have moved to the first pile disobeying the standard rules.

Finally, here is a proposed solution which is *incorrect* even for LBS with Saving Grace.

```
3 17 0 16 0 30 0
```

The first move is of the $4\heartsuit$ onto pile 0, which is not normally legal because it is two piles to the left (not one or three), but is allowed by the Saving Grace. The move of $3\heartsuit$ onto pile 0 is allowed as normal. The final move of $4\diamondsuit$ onto pile 0 is now illegal because the top card of that pile is $3\heartsuit$ and we have already used our Saving Grace.