1. [20 pts.] Any correct answer should have the following properties:

   completeness: The regular expression should describe all valid strings.

   correctness: The regular expression should not describe any invalid strings.

   Some correct regular expressions are:

$$\Big((baaa^*)^* \mid (aaa^*b)^* \mid b(aaa^*b)^* \mid (aaa^*b)^*aaa^*\Big)$$

$$\Big((b \mid \epsilon)\,(aaa^* \mid aabaa)^*\,(b \mid \epsilon) \mid b\Big)$$

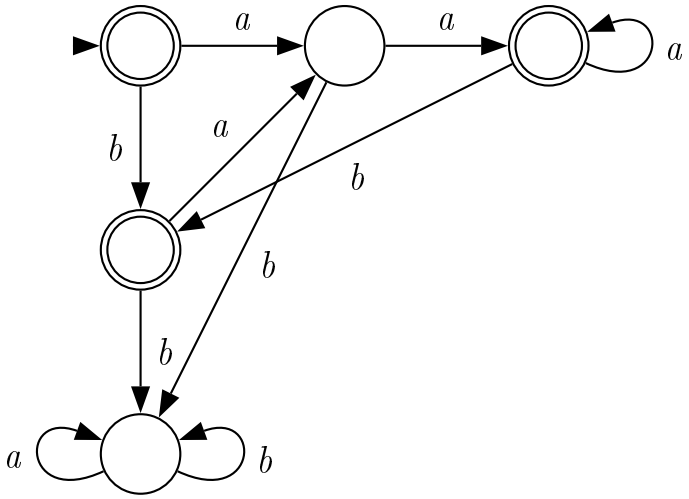
$$(b \mid \epsilon)(aaa^*b)^*(\epsilon \mid aaa^*)$$

$$(aaa^* \mid \epsilon)(baaa^*)^*(b \mid \epsilon)$$

$$((b \mid \epsilon)\,aaa^*)^*\,(b \mid \epsilon)$$

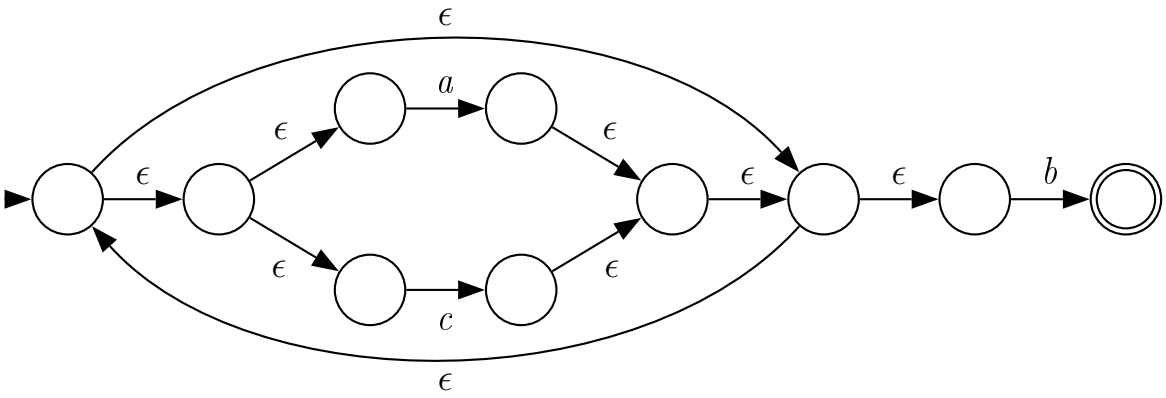
   Of course other answers could be correct also.

2. [26 pts.] As in the previous problem, a correct answer must accept all valid strings and only valid strings. One correct DFA is:



   As the problem emphasizes, no notational shortcuts should be used. A DFA always has exactly one outgoing transition for every symbol of its alphabet (otherwise it is not a valid DFA). Omitting a dead state and transitions to it, or transitions from the dead state to itself, are notational shortcuts that the problem said were not allowed.

3. [12 pts.] Notice that the problem says that not any NFA is desired (even if it is correct), but the specific NFA that would result from applying the construction given in class. Although there are many different NFAs that would recognize or accept this langauge (an infinite number in fact), there is only one NFA that would result from following the construction given in class.



4. [12 pts.]

   a. 1) `[]`

      2) `[7; 6; 5]`

      3) `2`

   b. 1) Three examples are `[[1]]` and `[[1; 2]]` and `[[1; 2]; [3; 4]]`

      2) Two examples are `[(1, 2)]` and `[(1, 2); (3, 4)]`

      3) Two examples are `(1, [2])` and `(1, [2; 3; 4])`

5. [30 pts.] Different solutions are obviously possible. The first solution below builds a hash (`element_counts`) mapping values to their number of occurrences in the input list. Then we want to get the largest value in the `element_counts` hash, and determine the key corresponding to it. The easiest way is to invert the hash by creating a new hash with the keys and values of `element_counts` reversed. There's actually a method in the `Hash` class that does that (`invert()`), which those who read about `Hash` may know about, but it's not required. (The solution below just does the hash inversion explicitly, without using the library method.)

   The second solution is a variation that just iterates over the hash to find the key that has the largest corresponding value:

   Another approach that could work would be to use nested loops, but that would be $\mathcal{O}(n^2)$, and the problem says that efficiency matters.

   The easiest way to extract the body of the list is probably a backreference, as used in the solution below.

   Notice that the problem says that the program should only read **one** input line. Also, integers are sequences of one or more digit characters, so a regular expression like `\d*(;\d*)*` would recognize sequences of adjacent semicolons with zero–length integers between them.

```
#!/usr/local/bin/ruby

list= gets()

if (list !~ /\[(\d+(;\d+)*)\]/) then
  puts("Invalid list.")
else

  # build hash mapping values to their number of occurrences in the list
  elements= $1
  element_counts= Hash.new()
  # or use elements.split(';').each() { |elt|
  elements.scan(/\d+/).each() { |elt|
    element_counts[elt]= 0 if (element_counts[elt] == nil)
    element_counts[elt] += 1
  }

  # invert hash
  reverse= Hash.new()
  element_counts.keys().each() { |key|
    reverse[element_counts[key]]= key
  }

  # print the value associated with the largest key, using Array.max()
  puts("#{reverse[reverse.keys().max()]}")

end


#!/usr/local/bin/ruby

list= gets()

if (list !~ /\[(\d+(;\d+)*)\]/) then
  puts("Invalid list.")
else

  # build hash mapping values to their number of occurrences in the list
  elements= $1
  element_counts= Hash.new()
  # or use elements.split(';').each() { |elt|
  elements.scan(/\d+/).each() { |elt|
    element_counts[elt]= 0 if (element_counts[elt] == nil)
    element_counts[elt] += 1
  }

  # find key with maximum value
  max_key= nil
  element_counts.keys().each() { |key|
    if (max_key == nil ||
        element_counts[key] > element_counts[max_key]) then
      max_key= key
    end
  }

  # print the value associated with the largest key
  puts(max_key)

end
```