Exercise 2.69.

The following procedure takes as its argument a list of symbol-frequency pairs (where no symbol appears in more than one pair) and generates a Huffman encoding tree according to the Huffman algorithm.

```
(define (generate-huffman-tree pairs)
  (successive-merge (make-leaf-set pairs)))
```

`Make-leaf-set` is the procedure given above that transforms the list of pairs into an ordered set of leaves. `Successive-merge` is the procedure you must write, using `make-code-tree` to successively merge the smallest-weight elements of the set until there is only one element left, which is the desired Huffman tree. (This procedure is slightly tricky, but not really complicated. If you find yourself designing a complex procedure, then you are almost certainly doing something wrong. You can take significant advantage of the fact that we are using an ordered set representation.)

Answer.

To design the `successive-merge` procedure, we first have to address two intractable problems:

(1) How to identify that there is only one element left in the set?

(2) How to keep all the elements in order after adding the merged element into the set?

We have just seen in the text that a set of leaves and trees is represented as a list of elements. Hence, the set contains only one element whenever the `cdr` of the set is empty. This solve the first obstacle above. On the other hand, we know that `adjoin-set` is a procedure which adds an element to a set while keeping the order of elements invariant in this process. Thus, we can solve the difficulty in adding newly merged element back into the set by using the `adjoin-set` procedure. Therefore, to successively merge the smallest-weight elements of the set, do the following:

- If the `cdr` of the set is empty, return the `car` of the set which is the desired only element.

- Otherwise, `successive-merge` on the new set which is obtained by `adjoin-set`ing the tree of the first two elements to the set of the remaining elements.

```
(define (successive-merge set)
  (cond ((null? set) '())
        ((null? (cdr set)) (car set))
        (else
          (let ((new-set (adjoin-set (make-code-tree (car set)
                                                      (cdr set))
                                      (cddr set))))
            (successive-merge new-set)))))
```

*. Creative Commons BY NC SA 2013, Lawrence R. Amlord(颜世敏).