Contributor: Shivanshi Nagar Degree: 3

Problem I. Run Huffman algorithm Scheme SICP code (an adaptation into Racket is also allowed) on the example from week 3 lectures (figures 1-2). Provide examples of encoding and decoding.

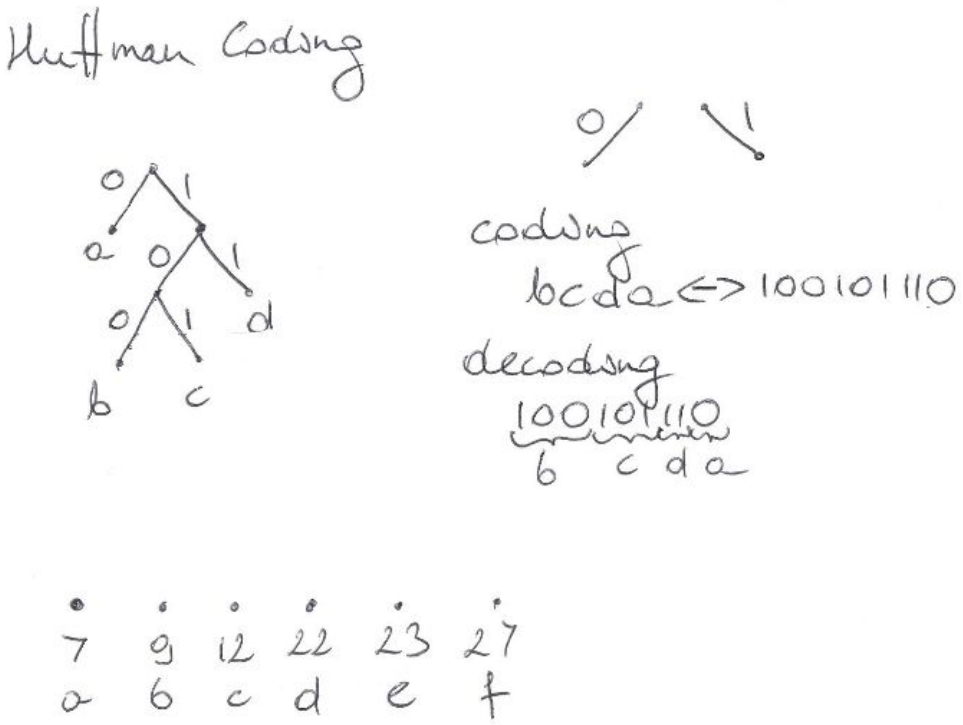
All programming targets were achieved to the 3rd degree. Below is proof of program correctness.

The following code was implemented from the week 3 lectures (figures 1-2). We chose to implement this in Racket.

```
#lang racket

;Leaves of the tree are represented by a list consisting of the symbol leaf
;the symbol at the leaf, and the weight
(define (make-leaf symbol weight)
  (list 'leaf symbol weight))
(define (leaf? object)
  (eq? (car object) 'leaf))
(define (symbol-leaf x) (cadr x))
(define (weight-leaf x) (caddr x))

;Creating left branch and right branch
(define (make-code-tree left right)
  (list left
        right
        (append (symbols left) (symbols right))
        (+ (weight left) (weight right))))

(define (left-branch tree) (car tree))

(define (right-branch tree) (cadr tree))
(define (symbols tree)
  (if (leaf? tree)
      (list (symbol-leaf tree))
      (caddr tree)))

;Defining the weight of the tree
(define (weight tree)
  (if (leaf? tree)
      (weight-leaf tree)
      (cadddr tree)))
```

```
;Decoding Huffman Tree
(define (decode bits tree)
  (define (decode-1 bits current-branch)
    (if (null? bits)
        '()
        (let ((next-branch
               (choose-branch (car bits) current-branch)))
          (if (leaf? next-branch)
              (cons (symbol-leaf next-branch)
                    (decode-1 (cdr bits) tree))
              (decode-1 (cdr bits) next-branch)))))
  (decode-1 bits tree))
(define (choose-branch bit branch)
  (cond ((= bit 0) (left-branch branch))
        ((= bit 1) (right-branch branch))
        (else (error "bad bit -- CHOOSE-BRANCH" bit))))

;Encoding the message
(define (encode message tree)
  (if (null? message)
      '()
      (append (encode-symbol (car message) tree)
              (encode (cdr message) tree))))

(define (encode-symbol symbol tree)
  (cond ((not (memq symbol (symbols tree)))
         (error "bad symbol -- ENCODE-SYMBOL" symbol))
        ((leaf? tree) '())
        ((memq symbol (symbols (left-branch tree)))
         (cons 0 (encode-symbol symbol (left-branch tree))))
        ((memq symbol (symbols (right-branch tree)))
         (cons 1 (encode-symbol symbol (right-branch tree))))))

;Merging new leaves onto a branch
(define (adjoin-set x set)
  (cond ((null? set) (list x))
        ((< (weight x) (weight (car set))) (cons x set))
        (else (cons (car set)
                    (adjoin-set x (cdr set))))))

;Creating pairs to merge
(define (make-leaf-set pairs)
  (if (null? pairs)
      '()
      (let ((pair (car pairs)))
        (adjoin-set (make-leaf (car pair)     ; symbol
                               (cadr pair))  ; frequency
                    (make-leaf-set (cdr pairs))))))
```

Figure 1:



This is the Racket implementation for this tree:

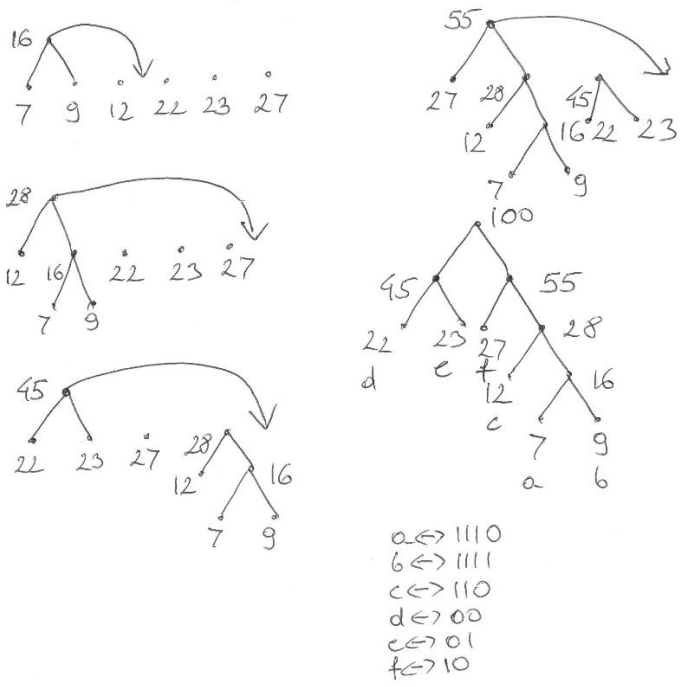
```
;Create sample message - figure1
(define sample-tree-figure1
  (make-code-tree
   (make-leaf 'A 7) (make-code-tree
                     (make-code-tree
                      (make-leaf 'B 9) (make-leaf 'C 12))
                     (make-leaf 'D 22))))

(display "Encoding figure 1 - 'BCDA'\n")
(encode '(B C D A) sample-tree-figure1)
(define encoded-message-figure1 (encode '(B C D A) sample-tree-figure1))
(display "Decoding figure 1 - 'BCDA'\n")
(decode encoded-message-figure1 sample-tree-figure1)
```

This is the output of the above code:

```
Encoding figure 1 - 'BCDA'
'(1 0 0 1 0 1 1 1 0)
Decoding figure 1 - 'BCDA'
'(B C D A)
```

It is clear that the output from the program matches what is shown in Figure 1.

Figure 2:



This is the Racket implementation for this tree:

```
;Create sample message - figure2
(define sample-tree-figure2
  (make-code-tree (make-code-tree
                   (make-leaf 'D 22) (make-leaf 'E 23))
                  (make-code-tree
                   (make-leaf 'F 27) (make-code-tree
                                      (make-leaf 'C 12) (make-code-tree
                                                         (make-leaf 'A 7) (make-leaf 'B 9))))))


(display "Encoding figure 2 - 'BCDA'\n")
(encode '(B C D A) sample-tree-figure2)
(define encoded-message-figure2 (encode '(B C D A) sample-tree-figure2))
(display "Decoding figure 2 - 'BCDA'\n")
(decode encoded-message-figure2 sample-tree-figure2)
```

This is the output of the above code:

```
Encoding figure 2 - 'BCDA'
'(1 1 1 1 1 1 0 0 0 1 1 1 0)
Decoding figure 2 - 'BCDA'
'(B C D A)
```

It is clear that the output from the program matches what is shown in Figure 2.