```
::::::::::::::
test1.in
::::::::::::::
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
tttttttttttttttttttttttttttttttttttttttt
aaaaaaaaaaaaaaaaaaaa
oooooooooooooooooooo
::::::::::::::
test1.out
::::::::::::::
x0A      5 01101
 a      20 0111
 e     100 1
 o      20 010
 t      40 00
EOF      1 01100
```

```
 1: ::::::::::::::
 2: test2.in
 3: ::::::::::::::
 4: eeeeeeeeeeeeeeeeeee eeeeeeeeeeeeeeeeeee eeeeeeeee
 5: eeeeeeeeeeeeeeeeeee eeeeeeeeeeeeeeeeeee eeeeeeeee
 6: ttttttttttttttttttt ttttttttttttttttttt
 7: aaaaaaaaaaaaaaaaaaa
 8: ooooooooooooooooooo
 9: iiiiiiiiii
10: nnnnn
11: sssss
12: h
13: r
14: ::::::::::::::
15: test2.out
16: ::::::::::::::
17: x0A    10 11001
18: x20     5 110001
19:  a     20 1101
20:  e    100 0
21:  h      1 11000010
22:  i     10 11100
23:  n      5 111010
24:  o     20 1111
25:  r      1 11000011
26:  s      5 111011
27:  t     40 10
28: EOF     1 1100000
```

```
 1: ::::::::::::::
 2: test3.in
 3: ::::::::::::::
 4: aaaaaaaaaa aaaaaaaaaa aaaaaaaaaa aaaaaaaaaa
 5: bbbbbbbbbb bbbbbbbbbb bbbbbbbbbb bbbbbbbbbb
 6: cccccccccc cccccccccc cccccccccc cccccccccc
 7: dddddddddd dddddddddd dddddddddd dddddddddd
 8: eeeeeeeeee eeeeeeeeee eeeeeeeeee eeeeeeeeee
 9: ffffffffff ffffffffff ffffffffff ffffffffff
10: gggggggggg gggggggggg gggggggggg gggggggggg
11: hhhhhhhhhh hhhhhhhhhh hhhhhhhhhh hhhhhhhhhh
12: iiiiiiiiii iiiiiiiiii iiiiiiiiii iiiiiiiiii
13: jjjjjjjjjj jjjjjjjjjj jjjjjjjjjj jjjjjjjjjj
14: ::::::::::::::
15: test3.out
16: ::::::::::::::
17: x0A    10 10001
18: x20    30 1001
19:  a     40 1010
20:  b     40 1011
21:  c     40 1100
22:  d     40 1101
23:  e     40 1110
24:  f     40 1111
25:  g     40 000
26:  h     40 001
27:  i     40 010
28:  j     40 011
29: EOF     1 10000
```

```
#!/bin/sh
# $Id: mk,v 1.13 2016-11-08 12:25:22-08 - - $

cat >test1.in <<__END__
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
tttttttttttttttttttttttttttttttttttttttt
aaaaaaaaaaaaaaaaaaaa
oooooooooooooooooooo
__END__

cat >test2.in <<__END__
eeeeeeeeeeeeeeeeeeee eeeeeeeeeeeeeeeeeeee eeeeeeeeee
eeeeeeeeeeeeeeeeeeee eeeeeeeeeeeeeeeeeeee eeeeeeeeee
tttttttttttttttttttt tttttttttttttttttttt
aaaaaaaaaaaaaaaaaaaa
oooooooooooooooooooo
iiiiiiiiii
nnnnn
sssss
h
r
__END__

cat >test3.in <<__END__
aaaaaaaaaa aaaaaaaaaa aaaaaaaaaa aaaaaaaaaa
bbbbbbbbbb bbbbbbbbbb bbbbbbbbbb bbbbbbbbbb
cccccccccc cccccccccc cccccccccc cccccccccc
dddddddddd dddddddddd dddddddddd dddddddddd
eeeeeeeeee eeeeeeeeee eeeeeeeeee eeeeeeeeee
ffffffffff ffffffffff ffffffffff ffffffffff
gggggggggg gggggggggg gggggggggg gggggggggg
hhhhhhhhhh hhhhhhhhhh hhhhhhhhhh hhhhhhhhhh
iiiiiiiiii iiiiiiiiii iiiiiiiiii iiiiiiiiii
jjjjjjjjjj jjjjjjjjjj jjjjjjjjjj jjjjjjjjjj
__END__

cid $0 phuffman.perl
for infile in test?.in
do
   outfile=$(echo $infile | sed s/in/out/)
   lisfile=$(echo $infile | sed s/in/lis/)
   phuffman.perl <$infile >$outfile 2>&1
   more $infile $outfile >$lisfile </dev/null
done
mkpspdf Listing.ps test?.lis $0 phuffman.perl
```

```
#!/usr/bin/perl
# $Id: phuffman.perl,v 1.6 2018-02-22 13:31:37-08 - - $

use strict;
use warnings;

$0 =~ s|/*$||;
$0 =~ s|^.*/||;
my $exit_status = 0;
sub note(@) {print STDERR "$0: @_"}
$SIG{__WARN__} = sub {note @_; $exit_status = 1};
$SIG{__DIE__} = sub {warn @_; exit};
END {exit $exit_status}

################################################################################

sub newtree($$;$$) {
   my ($char, $count, $leftp, $rightp) = @_;
   my $tree = {CHAR=> $char, COUNT=> $count};
   $tree->{CHILDREN} = [$leftp, $rightp] if $leftp || $rightp;
   return $tree;
}

sub cmptree($$) {
   my ($tree1p, $tree2p) = @_;
   return $tree1p->{COUNT} <=> $tree2p->{COUNT}
       || $tree1p->{CHAR}  <=> $tree2p->{CHAR}
}

sub hencode($$$);
sub hencode($$$) {
   my ($encodings, $tree, $encoding) = @_;
   if ($tree->{CHILDREN}) {
      hencode $encodings, $tree->{CHILDREN}->[$_], $encoding . $_
              for 0 .. $#{$tree->{CHILDREN}}
   }else {
      $encodings->[$tree->{CHAR}] = $encoding;
   }
}

```

```
41:
42: ##############################################################################
43:
44: use constant ROOT=> 1;
45:
46: sub parent($) {my ($index) = @_; $index >> 1}
47: sub lchild($) {my ($index) = @_; $index << 1}
48: sub rchild($) {my ($index) = @_; $index << 1 | 1}
49: sub empty($)  {my ($pqueue) = @_; $#$pqueue < ROOT}
50: sub newpqueue() {[0]}
51:
52: sub swap($$$) {
53:    my ($pqueue, $index1, $index2) = @_;
54:    @$pqueue[$index1, $index2] = @$pqueue[$index2, $index1];
55: }
56:
57: sub rootward($$$) {
58:    my ($pqueue, $index1, $index2) = @_;
59:    return (cmptree $pqueue->[$index1], $pqueue->[$index2]) < 0
60: }
61:
62: sub insert($$) {
63:    my ($pqueue, $tree) = @_;
64:    push @$pqueue, $tree;
65:    for (my $child = $#$pqueue; $child > ROOT; ) {
66:       my $parent = parent $child;
67:       last if rootward $pqueue, $parent, $child;
68:       swap $pqueue, $child, $parent;
69:       $child = $parent;
70:    }
71: }
72:
73: sub deletemin($) {
74:    my ($pqueue) = @_;
75:    die "deletemin: pqueue is empty" if empty $pqueue;
76:    swap $pqueue, ROOT, $#$pqueue;
77:    my $result = pop @$pqueue;
78:    my $parent = ROOT;
79:    for (;;) {
80:       my $child = lchild $parent;
81:       last if $child > $#$pqueue;
82:       my $rchild = rchild $parent;
83:       $child = $rchild if $rchild <= $#$pqueue
84:                        && rootward $pqueue, $rchild, $child;
85:       last if rootward $pqueue, $parent, $child;
86:       swap $pqueue, $parent, $child;
87:       $parent = $child;
88:    }
89:    return $result;
90: }
91:
```

```
 92:
 93: ##########################################################################
 94:
 95: # 1. Load frequency table.
 96:
 97: my @frequencies;
 98: for my $filename (@ARGV ? @ARGV : "-") {
 99:    open my $file, "<$filename" or do {warn "$filename: $!\n"; next};
100:    map {++$frequencies[ord $_]} split "" while <$file>;
101:    close $file;
102:    $frequencies[256] = 1;
103: }
104:
105: # 2. Load priority queue from frequency table.
106:
107: my $pqueue = newpqueue;
108: for my $char (0..$#frequencies) {
109:    insert $pqueue, newtree $char, $frequencies[$char]
110:           if $frequencies[$char];
111: }
112:
113: # 3. Unload priority queue into Huffman tree.
114:
115: my $tree;
116: for (;;) {
117:    last if empty $pqueue;
118:    $tree = deletemin $pqueue;
119:    last if empty $pqueue;
120:    my $rtree = deletemin $pqueue;
121:    insert $pqueue, newtree $tree->{CHAR},
122:                    $tree->{COUNT} + $rtree->{COUNT}, $tree, $rtree;
123: }
124:
125: # 4. Traverse Huffman tree into encoding array.
126:
127: my @encodings;
128: hencode \@encodings, $tree, "" if $tree;
129:
130: # 5. Print out frequency and encoding table.
131:
132: for my $char (0 .. $#frequencies) {
133:    next unless $frequencies[$char];
134:    my $fmt = (chr $char) =~ m/[[:graph:]]/ ? " %c " : "x%02X";
135:    printf $char == 256 ? "EOF"
136:         : (chr $char) =~ m/[[:graph:]]/ ? " %c "
137:         : "x%02X", $char;
138:    printf " %5d %s\n", $frequencies[$char], $encodings[$char];
139: }
140:
```