

```
1: ::::::::::::::
2: test1.in
3: ::::::::::::::
4: eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
5: eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
6: tttttttttttttttttttttttttttttttttttttttttt
7: aaaaaaaaaaaaaaaaaaaaaa
8: oooooooooooooooooooooo
9: ::::::::::::::
10: test1.out
11: ::::::::::::::
12: x0A      5  0110
13: a        20 0111
14: e       100  1
15: o        20 010
16: t        40  00
17: ::::::::::::::
18: test2.in
19: ::::::::::::::
20: eeeeeeeeeeeeeeeeeee eeeeeeeeeeeeeeeeeee eeeeeeeeeee
21: eeeeeeeeeeeeeeeeeee eeeeeeeeeeeeeeeeeee eeeeeeeeeee
22: tttttttttttttttttttt tttttttttttttttttttt
23: aaaaaaaaaaaaaaaaaaaaaa
24: oooooooooooooooooooooo
25: iiiiiiiiii
26: nnnnn
27: sssss
28: h
29: r
30: ::::::::::::::
31: test2.out
32: ::::::::::::::
33: x0A      10 11001
34: x20       5 110001
35: a        20 1101
36: e       100  0
37: h         1 1100000
38: i        10 11100
39: n         5 111010
40: o        20 1111
41: r         1 1100001
42: s         5 111011
43: t        40  10
```

```
1: #!/bin/sh
2: # $Id: mk,v 1.7 2015-10-23 18:17:18-07 - - $
3:
4: cat >test1.in <<__END__
5: eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
6: eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
7: tttttttttttttttttttttttttttttttttttttttttttttttttttttttt
8: aaaaaaaaaaaaaaaaaaaaaa
9: oooooooooooooooooooooo
10: __END__
11:
12: cat >test2.in <<__END__
13: eeeeeeeeeeeeeeeeeeee eeeeeeeeeeeeeeeeeeee eeeeeeeeeeee
14: eeeeeeeeeeeeeeeeeeee eeeeeeeeeeeeeeeeeeee eeeeeeeeeeee
15: tttttttttttttttttttttt tttttttttttttttttttttt
16: aaaaaaaaaaaaaaaaaaaaaa
17: oooooooooooooooooooooo
18: iiiiiiiiii
19: nnnnn
20: sssss
21: h
22: r
23: __END__
24:
25: cid $0 phuffman.perl
26: phuffman.perl <test1.in >test1.out 2>&1
27: phuffman.perl <test2.in >test2.out 2>&1
28: more test1.in test1.out test2.in test2.out >tests.lis </dev/null
29: mkpspdf Listing.ps tests.lis $0 phuffman.perl
```

```
1: #!/usr/bin/perl
2: # $Id: phuffman.perl,v 1.1 2014-10-24 17:30:27-07 - - $
3:
4: use strict;
5: use warnings;
6:
7: $0 =~ s|/*$||;
8: $0 =~ s|^\.*/||;
9: my $exit_status = 0;
10: sub note(@) {print STDERR "$0: @_"}
11: $SIG{__WARN__} = sub {note @_; $exit_status = 1};
12: $SIG{__DIE__} = sub {warn @_; exit};
13: END {exit $exit_status}
14:
15: #####
16:
17: sub newtree($$;$) {
18:     my ($char, $count, $leftp, $rightp) = @_;
19:     my $tree = {CHAR=> $char, COUNT=> $count};
20:     $tree->{CHILDREN} = [$leftp, $rightp] if $leftp || $rightp;
21:     return $tree;
22: }
23:
24: sub cmptree($$) {
25:     my ($treelp, $tree2p) = @_;
26:     return $treelp->{COUNT} <=> $tree2p->{COUNT}
27:         || $treelp->{CHAR} <=> $tree2p->{CHAR}
28: }
29:
30: sub hencode($$$);
31: sub hencode($$$) {
32:     my ($encodings, $tree, $encoding) = @_;
33:     if ($tree->{CHILDREN}) {
34:         hencode $encodings, $tree->{CHILDREN}->[$_], $encoding . $_
35:             for 0 .. $#{$tree->{CHILDREN}}
36:     }else {
37:         $encodings->[$tree->{CHAR}] = $encoding;
38:     }
39: }
40:
```

```
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($$$) {
58:     my ($pqueue, $index1, $index2) = @_;
59:     return (cmpmtree $pqueue->[$index1], $pqueue->[$index2]) < 0
60: }
61:
62: sub insert($$) {
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: }
103:
104: # 2. Load priority queue from frequency table.
105:
106: my $pqueue = newpqueue;
107: for my $char (0..$#frequencies) {
108:     insert $pqueue, newtree $char, $frequencies[$char]
109:         if $frequencies[$char];
110: }
111:
112: # 3. Unload priority queue into Huffman tree.
113:
114: my $tree;
115: for (;;) {
116:     last if empty $pqueue;
117:     $tree = deletemin $pqueue;
118:     last if empty $pqueue;
119:     my $rtree = deletemin $pqueue;
120:     insert $pqueue, newtree $tree->{CHAR},
121:         $tree->{COUNT} + $rtree->{COUNT}, $tree, $rtree;
122: }
123:
124: # 4. Traverse Huffman tree into encoding array.
125:
126: my @encodings;
127: hencode \@encodings, $tree, "" if $tree;
128:
129: # 5. Print out frequency and encoding table.
130:
131: for my $char (0 .. $#frequencies) {
132:     next unless $frequencies[$char];
133:     my $fmt = (chr $char) =~ m/[[:graph:]]/ ? " %c " : "x%02X";
134:     printf $fmt . "%8d %s\n", $char,
135:         $frequencies[$char], $encodings[$char];
136: }
137:
```