CSE312-L: DSA
LNMIIT, Rupa-ki-Nangal, Jaipur 302031
Training Set 07

The training set covers several topics and lessons including Huffman trees, a greedy algorithm, priority queue and heap. We use a single declaration of a C structure (`struct Huff`) to support all these topics. It maybe appropriate to sound an early caution here about the code. The code is a bit fiddly and the students need to be very vigilant lest they make a mistake. Mistakes are very difficult to locate and correct in these programs.

Telegrams were important communication media till a few decades ago before the modern digital technology revolution changed the communication environment. Sam Morse used a text corpus to estimate a frequency of use of letters in English alphabet. The frequently used letters were assigned shorter code. The less used letters were relegated to longer Morse codes. I have downloaded the frequency of letter usage that (perhaps) was observed by Samuel Morse.

We will use Huffman tree to assign codes to these letters. The algorithm is based on greedy algorithmic approach. We will use the coding to generate bit string for a phrase in our address VILL: RUPA KI NANGAL. We will also compare the string against a code that is based on equal frequency for all letters. Finally, we will try a code if the corpus was derived from the phrase we are seeking to encode.

Like in many of the previous training sets, the students have been provided a lot of code already prepared for them to use. Please ready the code carefully to support your learning. In the tasks students will be asked to write and practice code that adds to their classroom lessons.

We will have three sets of `struct Huff` nodes in this program. First set holds 26 uppercase letters of English alphabet together with their frequencies in a text corpus. This data is declared as an array in the program. Next set again comprises 26 external nodes of the Huffman tree. Each external node is parent to exactly one alphabet node. Alphabet node is always pointed through the left link of the external node in the provided program. The right link is set as NULL for each external node. Internal nodes in the tree may have external and internal nodes as their children.

Each Huffman tree node carries the sum of occurrences of letter in its descendant nodes in the (referred) corpus. These occurrences are used to prioritise (the basic algorithmic approach in the algorithm is greedy) the node selection while constructing the Huffman tree. “Morse code” we construct, assigns bit 0 to the left branch/node and 1 to the right branch/node in the tree to emit “Morse” codes for the letters.

Students have been provided with most of the code to quickly begin working on this training set tasks. Once again students will note my preference for recursive functions over the iterative code. PREPARE FOR THE LAB WORK BEFORE YOU ARRIVE IN THE LAB.

Training Set 07: Task 01

Each Huffman tree node has been declared with three pointers. Two pointers connect the parent node to its two children nodes. And, the third connects each child to its parent. Write and add `assert()` declarations to ensure that these links satisfy the obvious sanity requirement.

After reading the code below test your understanding of the code by printing data in elements of array `alphabets[]` by iterating over the data inserted in array `heap[]`. That is, use a for loop to select each element in `heap[]` and then use pointer there to get the `alphabet[]` element. You must use the appropriate pointers from `struct Huff` for this purpose.

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <ctype.h>

char town[] = "VILL: RUPA KI NANGAL";

struct Huff {
      int freq;
      char alpha;
      struct Huff *left;
      struct Huff *right;
      struct Huff *parent;
};

char letters[] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
            'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q',
            'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'};


int freq[] = {21912, 16587, 14810, 14003, 13318, 12666,
                  11450, 10977, 10795, 7874, 7253, 5246, 4943,
                  4761, 4200, 3853, 3819, 3693, 3316, 2715,
                  2019, 1257, 315, 205, 188, 128 };
/*
int freq[] = {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                                    1,1,1,1,1,1,1};

int freq[] = {3,0,0,0,0,0,1,0,
      2,0,2,3,0,2,0,1,0,
      1,0,0,1,1,0,0,0,0};
*/

struct Huff alphabet[26]; // External nodes of Huffman tree.
struct Huff *heap[30];
int heapSz = 0; // index 0 is not used in this heap

void heapifyUp(int index);
void heapifyDown(int index);

void initAlphaNodes(void) {
      /* Fill freq data in array alphabet
            Alphabets are External nodes in
            our Huffman tree
```

```
            External nodes are declared in our code.
      */
      int i;
      for (i=0; i<26; i++) {
            alphabet[i].alpha = letters[i];
            alphabet[i].freq = freq[i];
            alphabet[i].left = NULL;
            alphabet[i].right = NULL;
      }
}

struct Huff *makeHuffNode(int freq, struct Huff *left,
                                                struct Huff *right) {
            struct Huff *newNode = malloc(sizeof(struct Huff));
            assert(newNode != NULL);
            newNode->alpha = ' '; // symbol marks internal node
            newNode->freq = freq;
            newNode->right = right;
            newNode->left = left;
            if (left != NULL) {
                  left->parent = newNode;
                  if (left->alpha == ' ')
                        left->alpha = '0';
            }
            if (right != NULL) {
                  right->parent = newNode;
                  right->alpha = '1';
            }

            return newNode;
}

void addToHeap(struct Huff *newNode) {
      assert(heapSz < 30);
      heap[++heapSz] = newNode;
      heapifyUp(heapSz); // Make the priority queue good
}

struct Huff *removeMin(void) {
      assert(heapSz>1);
      struct Huff *tmp = heap[1];
      heap[1] = heap[heapSz--];
      heapifyDown(1);
      return tmp;
}

void initExternalHeapNodes(void) {
      int i;
      for (i=0; i<26; i++) {
            addToHeap(makeHuffNode(alphabet[i].freq,
                                          &alphabet[i], NULL));
      }
}

void swapAtIdx(int x, int y) {
      /* Used by heapify functions */
      struct Huff *tmp;
```

```
      tmp = heap[x];
      heap[x] = heap[y];
      heap[y] = tmp;
}

void heapifyUp(int index) {
/* Removed about 10 lines of code */
}
void heapifyDown(int parent) {
/* Remove 10 to 15 lines of code */
      return;
}

int heapIsRootOnly(void) {
      return heapSz == 1;
}

void emitLetterCode(char c) {
      int i = c-'A';
      struct Huff *bit = alphabet[i].parent;
      while (bit != NULL) {
            printf("%c", bit->alpha);
            bit = bit->parent;
      }
}

int main(void) {
      struct Huff *left, *right, *newNode, *node;
      int i;

      initAlphaNodes();
      initExternalHeapNodes();

      while (!heapIsRootOnly()) {
            left = removeMin();
            if (heapIsRootOnly()) {
                  right = heap[1];
                  heap[1] =
                        makeHuffNode(left->freq+right->freq,
                                                      left, right);
                  break;
            }
            right = removeMin();
            addToHeap(makeHuffNode(left->freq+right->freq,
                                                      left, right));
      }

      /* Code town */
      printf("Words in %s are coded as follows\n", town);
      i =-1;
      while (town[++i] != '\0')
            if (isspace(town[i]))
                  printf("\n");
            else if (!isupper(town[i]))
                  printf("\n%c", town[i]);
            else
                  emitLetterCode(town[i]);
```

```
}
```

The program was run to generate the following output.

```
Words in VILL: RUPA KI NANGAL are coded as follows
1110100 1010 10010 10010
:
001111 010100 00100 110
11101 1010
11100 110 11100 1001 110 10010
```

Training Set 07: Task 02

Primary coding and learning task in this training set for the students is to write codes for two functions. These functions are:

```
void heapifyUp(int index) {
/* Removed about 10 lines of code */
}

void heapifyDown(int parent) {
/* Remove 10 to 15 lines of code */
     return;
```

Students should be able to determine the specifications for the functions from their classroom lessons, functions names and their reading of the given code.

Training Set 07: Task 03

Notice that we have given three sets of values for array `freq[]`. These are provided to compare Huffman codes under different frequency of occurrences of the letters in the corpus. For our beloved town at the entre of the world, the alternate codes are not significantly different in their space requirements. Try a bigger size data to test the coding schemes.