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

The training set covers several topics and lessons including Huffman trees (it uses a greedy approach to optimise), priority queues and heaps. This will require a number of data-structures. We have used a single declaration of a C structure (`struct Huff`) to implement nodes for different purposes. It is therefore appropriate to sound a caution here about the code. The code is a bit fiddly and the students need to be careful lest they make programming mistakes. Mistakes are difficult to locate and correct in the programs.

Three data structures of primary interest in the program are: (a) an array of alphabets, (b) a Huffman tree, and (c) array heap to prioritise node selection for constructing Huffman tree. An array of `struct Huff` is used for storing data about each letter in English alphabet. These structs are not part of the Huffman tree constructed in the program. However, every leaf node in the Huffman tree will refer to a `struct Huff` element in array `alphabet` Huffman tree nodes will await in array `heap[]` for their turn to be processed. This array is initialised with `struct Huff` node that points to exactly one `struct Huff` entry in array alphabet. This information should be enough to understand the code provided to the students.

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

We will construct a Huffman tree to assign codes to these letters. The construction algorithm is based on a greedy approach. We will use the derived coding scheme to generate a bit string for a phrase in our address VILL: RUPA KI NANGAL. We will also compare the coding against codes that are based on other frequency profiles.

As in many of the previous training sets, the students have been provided a lot of ready to use code. Please read the code carefully to assist your DSA education. In the tasks set for this training session, students will be asked to write and practice code that adds to their DSA 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 leaf nodes of a Huffman tree to be constructed. Each leaf node in this tree refers to exactly one alphabet node. Alphabet node is always pointed through the left link of the leaf node in this program. The right link is set as NULL for each leaf node of the Huffman tree. Internal nodes in the tree may have leaf and internal nodes as their child nodes.

Each `struct Huff` node (including those in array `alphabet`) also keeps a reference to its parent node. This link will be used to print the code for each letter in Morse alphabet.

Each Huffman tree node carries the sum of frequencies of letters in its descendant nodes. These occurrences frequencies are used to prioritise the node selection when building the Huffman tree to assign Morse codes to the letters. "Morse code" we construct, assigns bit 0 to the left branch/node and 1 to the right branch/node of the tree.

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 codes.

PREPARE FOR THE LAB WORK BEFORE YOU ARRIVE IN THE LAB. Unlike a lecture class, a lab session requires significant preparation BEFORE the lab session.

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 in the program to ensure that these links satisfy the obvious relationships among these referenced nodes.

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 in the entry to get an `alphabet[]` entry. You must use the appropriate pointers from `struct Huff` to select the matching entry.

The array `heap[]` contains the root nodes of a forest. The trees in the forest need to be organised in a single Huffman tree before printing a Morse codes for the letters. The array `heap[]` is maintained as a priority queue using functions `heapifyUp()` and `heapifyDown()`.

Huffman tree is constructed by removing two minimum frequency `struct Huff` structures from priority queue `heap[]`. These `struct Huff` are combined under a single parent node. The new parent node is inserted in array `heap[]` to wait for its turn to join Huffman tree being constructed.

```
#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 };
/* Other test cases
int freq[] = {1,2,3,4,5,6,7,8,9,10,11,12,13,
                  14,15,16,17,18,19,20,21,22,23,24,25,26};

int freq[] = {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                  1,1,1,1,1,1,1};

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]; // Fixed array to store letters

// References to nodes awaiting to be in Huffman tree
// Roots of trees in a forest
struct Huff *heap[30];
// and count of entries in heap
int heapSz = 0; // index 0 is not used in this heap

// To re-establish heap property after an insertion
void heapifyUp(int index);
// To re-establish heap property after a deletion
void heapifyDown(int index);

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

            Leaf nodes in the heap will have
            reference to these entities in their

      */
      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);
            // Will use this to keep coding information
            newNode->alpha = ' ';
            newNode->freq = freq;
```

```
            newNode->right = right;
            newNode->left = left;
            if (left != NULL) {
                  left->parent = newNode;
                  // If not pointing to an entry in array alphabet
                  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); // Restore heap property
}

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

void initHeapLeaves(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
See Figure 27.7 in Reema Thareja textbook */
}

void heapifyDown(int parent) {
/* Removed 10 to 15 lines of code
See Figure 12.10 in Reema Thareja */
      return;
}
```

```
int heapIsRootOnly(void) { // Huffman tree is ready?
      return heapSz == 1;
}

void printLetterCode(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();
      initHeapLeaves();

      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
                  printLetterCode(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
```

The output for case with frequencies 1,.. 26 was:

```
Words in VILL: RUPA KI NANGAL are coded as follows
0001 01100 10101 10101
:
0100 0110 0000 01100101
11110 01100
10111 01100101 10111 001111 01100101 10101
```

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 additional 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 further test and demonstrate the coding schemes.

Vishv Malhotra
25 April 2020
The LNMIIT, Rupa Ki Nangal,
Jaipur 302031