BLG221E - DATA STRUCTURES
PROJECT 1
STARTING DATE: 17.04.2013 DURATION: 3 WEEKS

ASCII Encoding

- The ASCII character encoding is a standard method used to store data in computer memory and hard disk.
- ASCII encoding is a fixed-length encoding method, 8 bits (1 byte) are used for each character.
- This method can encode up to 256 (2^8) characters including English alphabet letters (a-z, A-Z), number digits (0-9), and all other keyboard characters.

Letter	ASCII Code (Decimal)	ASCII Code (Binary)
A	65	01000001
B	66	01000010
C	67	01000011
D	68	01000100
E	69	01000101
F	70	01000110
. . .		
. . .		
Z	90	01011010

Huffman Encoding

- Huffman Encoding is a variable-length encoding method.
- In Huffman encoding, some characters may require only a few bits, and some other characters may require more than 8 bits.
- Huffman encoding uses statistical frequencies (occurance probabilities) of each character that are used in texts.
- The followings are commonly accepted statistical frequencies of English letters.

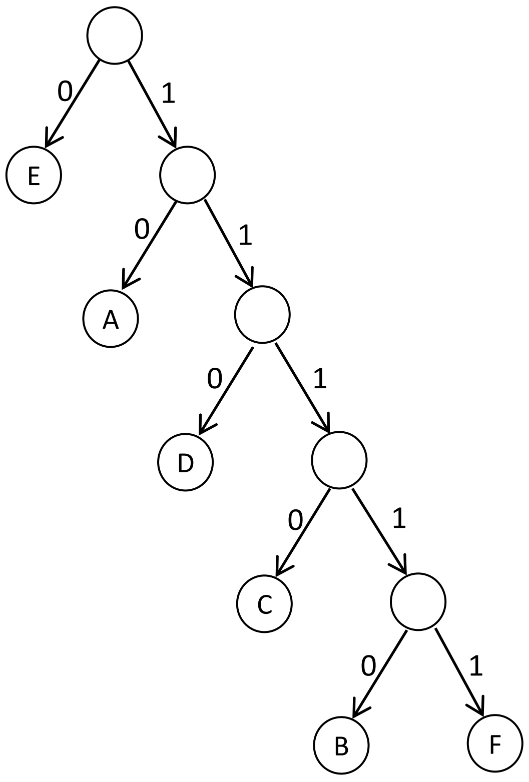
Letter	A	B	C	D	E	F	G	H	I	J	K	L	M
Frequency	8.1	1.5	2.8	4.3	12.8	2.3	2.0	6.1	7.1	0.2	0.1	4.0	2.4

Letter	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Frequency	6.9	7.6	2.0	0.1	6.1	6.4	9.1	2.8	1.0	2.4	0.1	2.0	0.1

- Huffman encoding method assigns shorter-length bits for more frequently used letters, and longer-length bits for less frequently used letters.
- Therefore, it can be used for compressing and decompressing data such as text, audio/visual images etc.

PROJECT

- In this project, you will write a program to find and display Huffman bit codes for English capital letters (A-Z).
- You will **not** do any text compression / decompression.
- You should use the letter frequencies given above and build a Huffman Tree as described below.
- For simplicity, let us show the **finished Huffman Tree** for the first five letters (A-F).



- This is a binary tree which each path from root to a leaf node gives the Huffman bit codes for a letter.
- Leaf nodes are always letters, other nodes are combination nodes.
- Each combination node has exactly two children.

Huffman Bits Codes Table

- To find the Huffman bit codes for each letter, we start from tree root and follow the path to a letter in a leaf node.
- Left branch is always considered as 0, right branch is always considered as 1 (or vice versa).

Letter	Huffman Bits Code (Binary)
A	10
B	11110
C	1110
D	110
E	0
F	11111

(It is guaranteed that no lettter's bit codes is a prefix of another.)

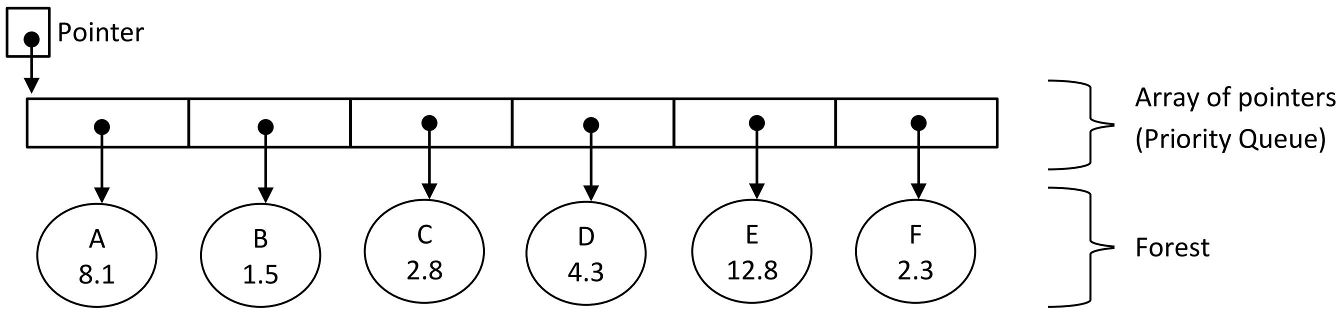
ALGORITHM FOR BUILDING THE HUFFMAN TREE

1. Create the array of pointers to trees, where each root node contains one letter.
2. Repeat the following steps until there is only one tree remains in the array.
- **2.1.** Sort the array by character frequencies in ascending order.
- **2.2.** From the array, select the two trees with the smallest frequencies and remove them from array.
- **2.3.** Shift all items in array leftward by two times.
- **2.4.** Combine the selected two trees into a new tree whose root contains the following information:
  - Frequency = Sum of the frequencies of the two selected trees.
  - Letter = '*'
  - Left and right children = The two trees as its left and right subtrees.
- **2.5.** Add the new combined tree at the end of the array.

3. The remaining one node is the root of the entire Huffman encoding tree.

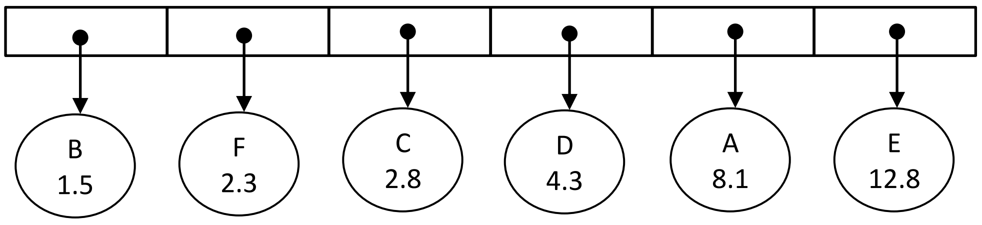
EXAMPLE

Step 1 : Create an array of pointers to the root nodes for the five letters (A-F).

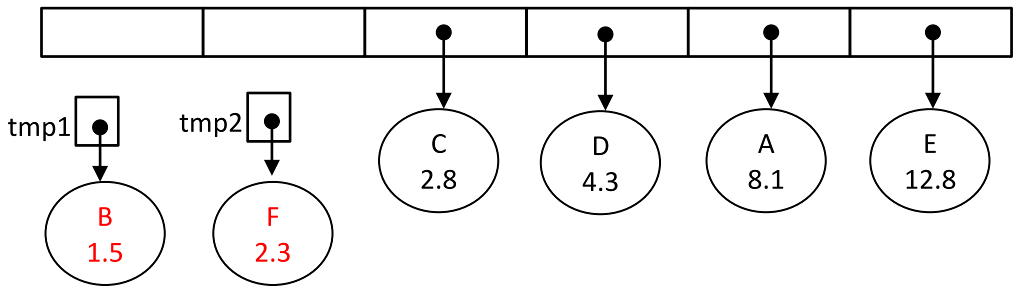


FIRST ITERATION :

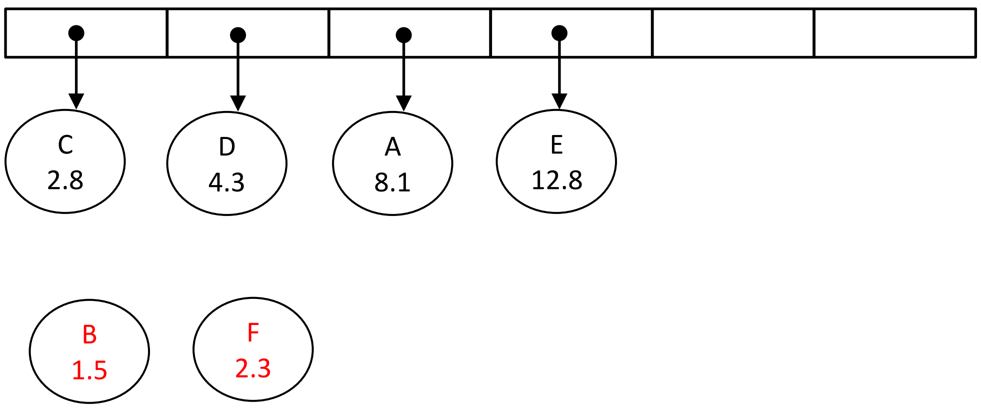
Step 2.1 : Sort the array by character frequencies in ascending order.



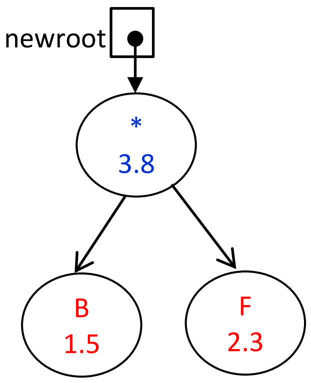
Step 2.2 : Select the two trees with the smallest frequencies and remove them from array.



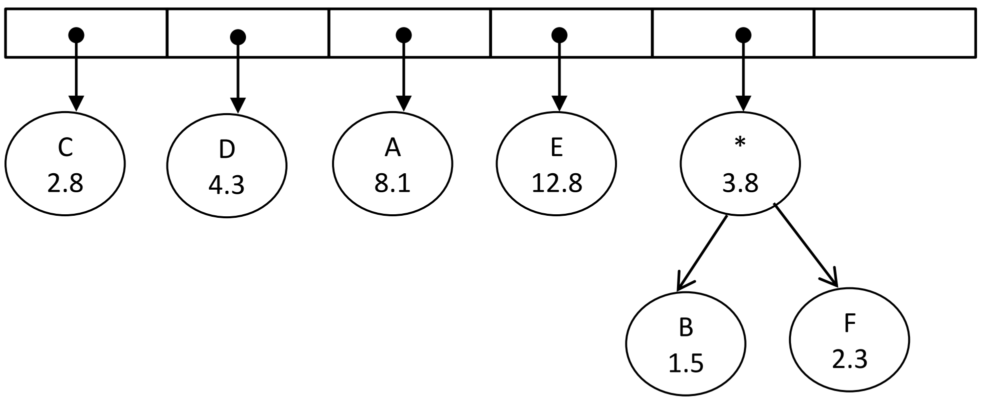
Step 2.3 : Shift all items in array leftward by two times.



Step 2.4 : Combine the selected two trees into a new tree.

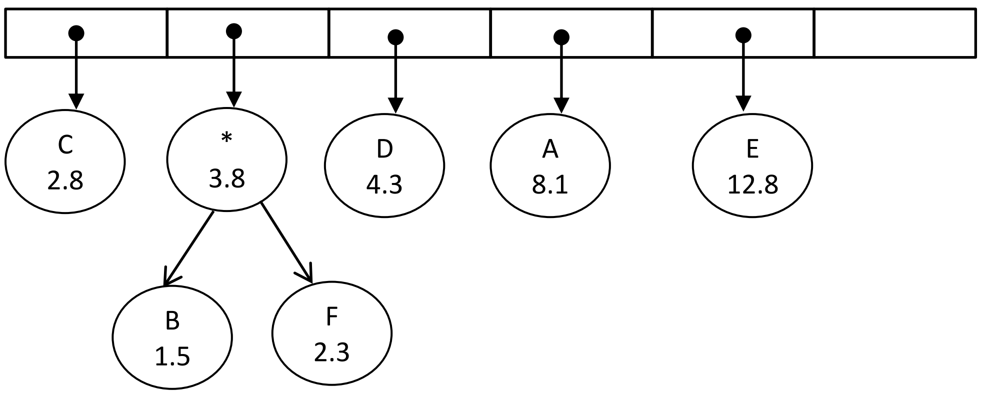


Step 2.5 : Add the new combined tree at the end of the array.

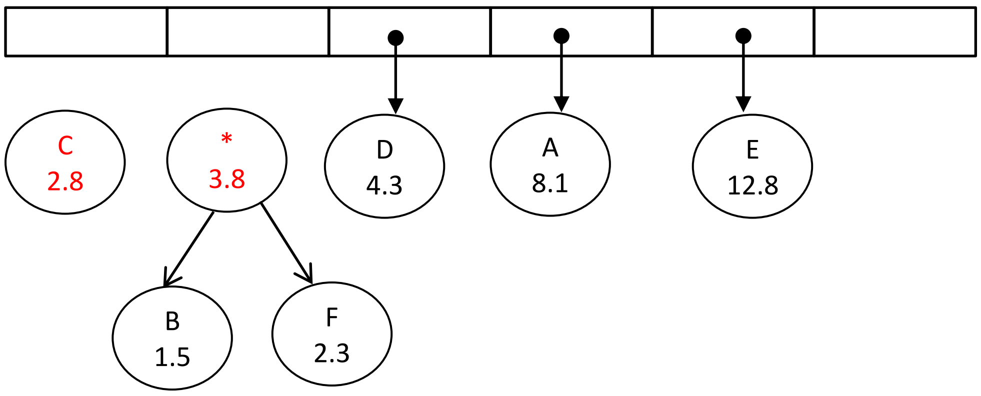


SECOND ITERATION :

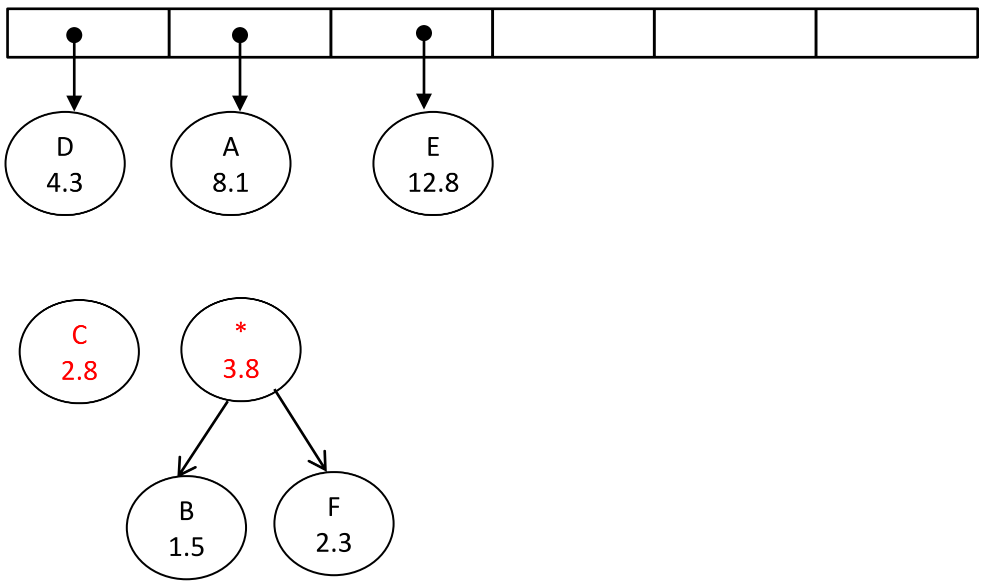
Step 2.1 : Sort the array by character frequencies in ascending order.



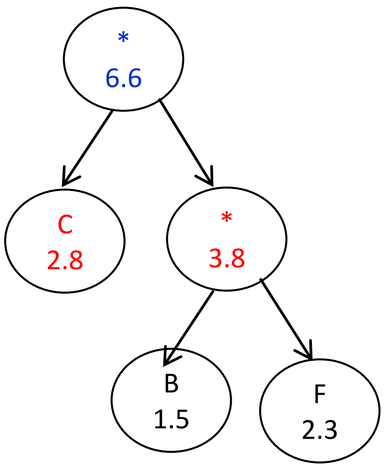
Step 2.2 : Select the two trees with the smallest frequencies and remove them from array.



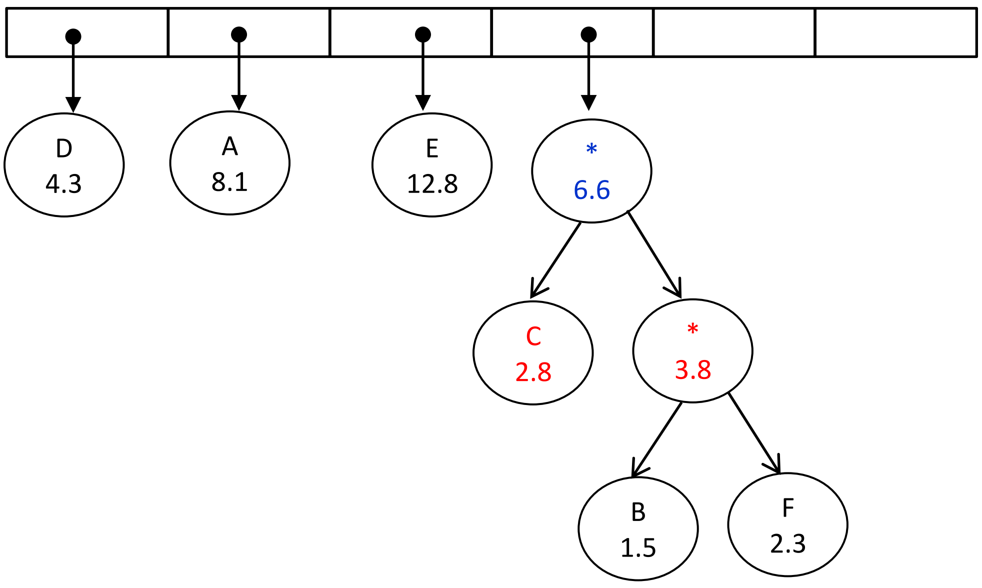
Step 2.3 : Shift all items in array leftward by two times.



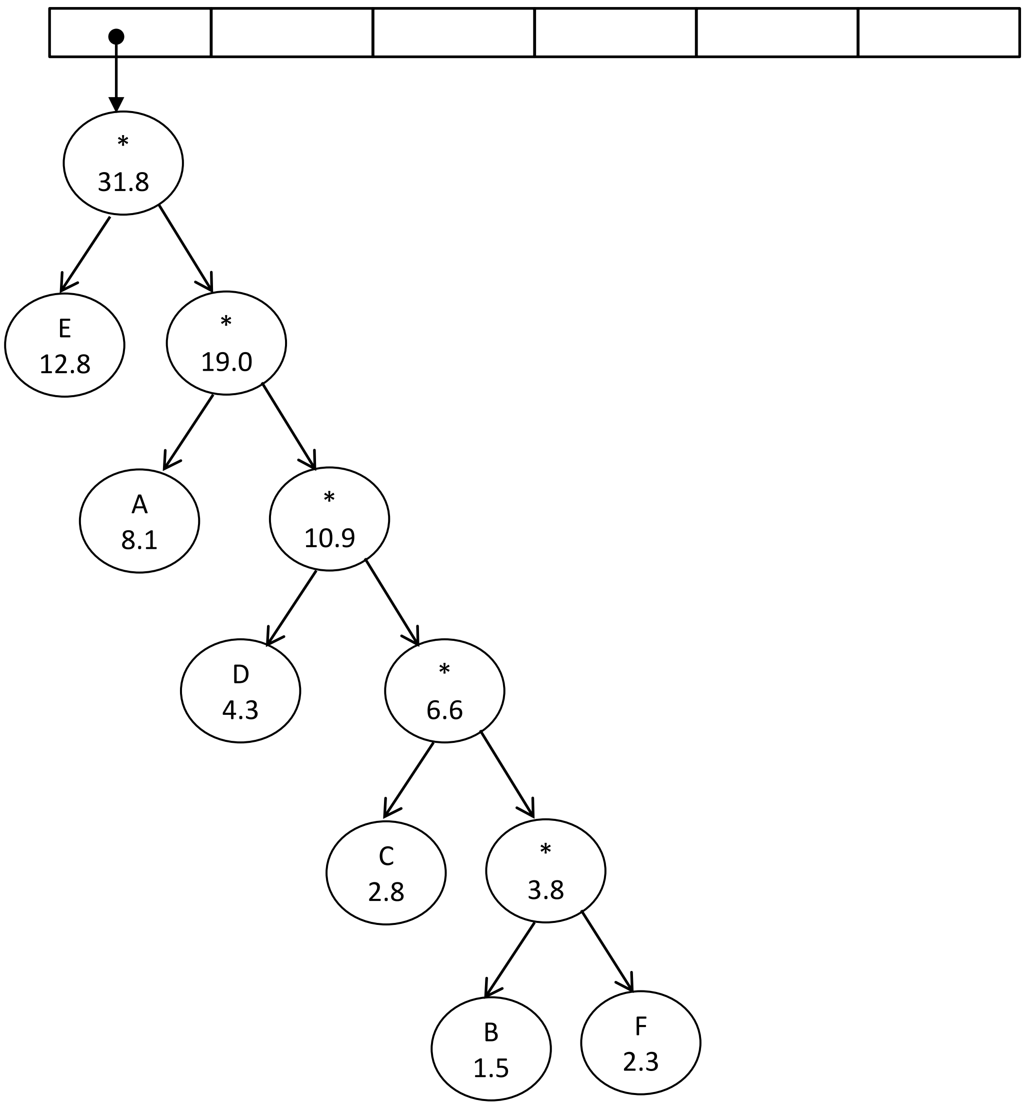
Step 2.4 : Combine the selected two trees into a new tree.



Step 2.5 : Add the new combined tree at the end of the array.



THE FINAL HUFFMAN TREE, AFTER LAST ITERATION :



IMPLEMENTATION OF THE TREES

You should use the following **node** definition for Huffman Trees.

```
struct Node
{
   char      letter;
   float     frequency;
   Node *  left;
   Node *  right;
};
```

You should define and create the following **array**, which will be used as a **Priority Queue**.

```
#define  N  26          // Number of English letters
Node *  array[N];    // Set of pointers to trees (Forest)
```

ALGORITHM FOR DETERMINING THE BIT CODES FOR LETTERS

After building the Huffman Tree, you need to determine the Huffman bit codes (as char string) for each letter. To do this, you can **recursively** traverse the tree (Inorder, Postorder, or Preorder method) and build a **Stack** for storing the bits.

```
char   bits_stack[20];     // Stack for storing bits.
int   stack_top=0;         // Stack Top.
```

- **PUSH operation :**
  Every time you go to left branch in tree, add '0' to the Stack.
  Every time you go to right branch, add '1' to the Stack.
  Then increment stack_top by 1.

- **POP operation :**
  Every time you reach a leaf node which contains a letter, you finish finding the bits code for that letter.
  Copy the entire bits_stack string to the bits_code array as defined below.
  Then decrement stack_top by 1.

  ```
  char   bits_code[N][20];   // Array of Huffman bit codes for N letters. (Huffman Bits Codes Table)
  ```

EXAMPLE SCREEN OUTPUT

Your program should display the results of Huffman Bits codes for all letters (A-Z).
Notice that resulting bits would be different when you used only the first 5 letters.

```
RESULTS OF HUFFMAN TREE BUILDING

Huffman Coding Bits   Letter  Frequency
1110                  A       8.1
110000                B       1.5
01000                 C       2.8
11111                 D       4.3
100                   E       12.8
00101                 F       2.3
111100                G       2.0
0101                  H       6.1
1011                  I       7.1
110001010             J       0.2
1100010110            K       0.1
11001                 L       4.0
00110                 M       2.4
1010                  N       6.9
1101                  O       7.6
111101                P       2.0
1100010111            Q       0.1
0110                  R       6.1
0111                  S       6.4
000                   T       9.1
01001                 U       2.8
1100011               V       1.0
00111                 W       2.4
110001000             X       0.1
00100                 Y       2.0
110001001             Z       0.1
Devam etmek için bir tuşa basın . . .
```