Lab Assignment 5

CS213

Marks 10.

Due: Part A: 6pm today. Part B: 6pm Friday, Oct 1.

Late Submission: 24 hour late submission with 10% penalty.

(**Warning**: Academic dishonesty will lead to straight FF grade in both CS213 and CS293. We will be running copy-detection software. Discussion with others is perfectly fine. You can discuss the logic at a high level. But you must write the code yourself. If your code is just not working, it is OK to take someone else's code for part of your solution, as long as you explicitly acknowledge them and are willing to lose marks for that part.It is also OK to seek help from someone in debugging your code, after you have spent quite some time. But they should be debugging your code not giving you their code.)

Problem Description: Implement the Heap based ADT PriorityQueue using a dynamic memory allocation based heapTree where each node contains left and right children.

A priorityQueue implements, following two methods:

Insert, GetMin

The signature of your priorityQueue class will take two types: that of a **key** and an **element**.

```
template <class key, class element> class priorityQueue{
private:
        heap<key,element>* t;
.......
}
```

(If you are having difficulty with templates, you can work with fix types without any penalty.)

'key' indicates the priority of the element. In our application, key happens to be a value contained inside the element itself, but in other applications, like simulation discussed in the class, key is completely independent of the element.

The heapTree itself will have to be a templatized class: The value stored in each tree node will be a struct containing both a key and an element:

```
template <class key, class element> class heap{
```

```
public:
  key priority;
  element value;
  heap *left;
  heap *right;
  ...
}
```

a) (7 marks) Verify your implementation by implementing PQsort:
Insert N elements in the heap and then call getMin N times. Verify that numbers are correctly sorted.

b) (3 marks) Implement the Huffman Encoding using you priorityQueue class. Note here that the **element**, as described below, will be a tree itself, though a different type than a heapTree. Hence your program will contain a tree of tree, just as in mergesort we had list of lists.

Following description of Huffman coding is based on the book CLR.

In the pseudocode that follows, we assume that C is a set of n characters and that each character c ∈ C is an object with a defined frequency f [c]. The algorithm builds the tree T corresponding to the optimal code in a bottom-up manner. It begins with a set of |C| leaves and performs a sequence of |C| − 1 "merging" operations to create the final tree. A min-priority queue Q, keyed on f , is used to identify the two least-frequent objects to merge together. The result of the merger of two objects is a new object whose frequency is the sum of the frequencies of the two objects that were merged.

```
HUFFMAN(C)
1  n ← |C|
2  Q ← C
3  for i ← 1 to n − 1
4       do allocate a new node z
5          left[z] ← x ← EXTRACT-MIN(Q)
6          right[z] ← y ← EXTRACT-MIN(Q)
7          f[z] ← f[x] + f[y]
8          INSERT(Q, z)
9  return EXTRACT-MIN(Q)          ▷ Return the root of the tree.
```

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

Figure 16.3 A character-coding problem. A data file of 100,000 characters contains only the characters a–f, with the frequencies indicated. If each character is assigned a 3-bit codeword, the file can be encoded in 300,000 bits. Using the variable-length code shown, the file can be encoded in 224,000 bits.

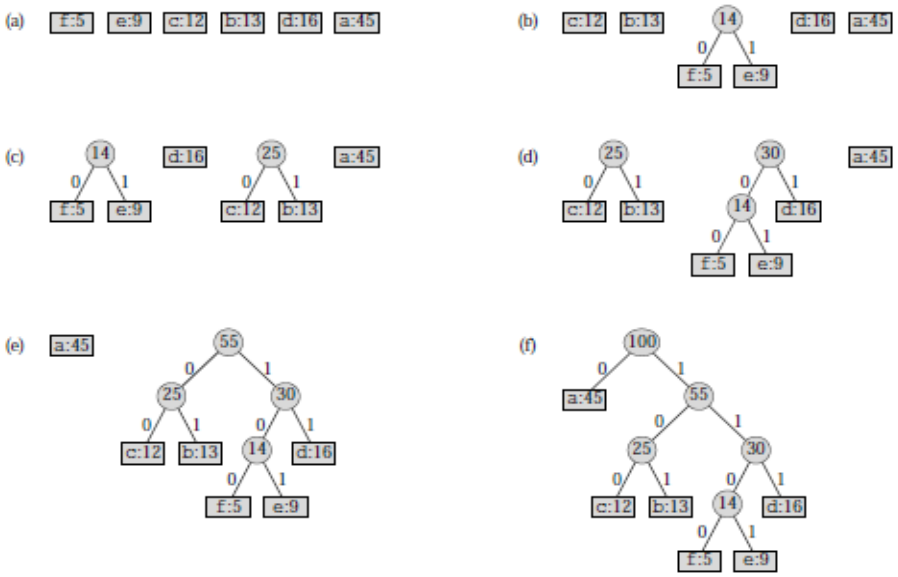
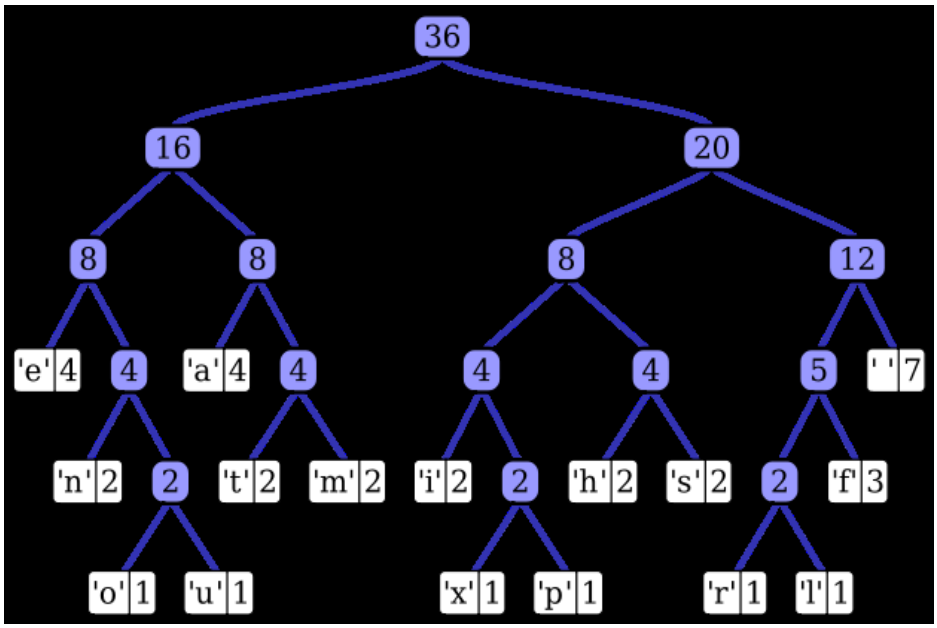


Figure 16.5 The steps of Huffman's algorithm for the frequencies given in Figure 16.3. Each part shows the contents of the queue sorted into increasing order by frequency. At each step, the two trees with lowest frequencies are merged. Leaves are shown as rectangles containing a character and its frequency. Internal nodes are shown as circles containing the sum of the frequencies of its children. An edge connecting an internal node with its children is labeled 0 if it is an edge to a left child and 1 if it is an edge to a right child. The codeword for a letter is the sequence of labels on the edges connecting the root to the leaf for that letter. **(a)** The initial set of $n = 6$ nodes, one for each letter. **(b)**–**(e)** Intermediate stages. **(f)** The final tree.

For another source of verification you can use the following data from wikipedia:

Char	Freq	Code
space	7	111
a	4	010
e	4	000
f	3	1101
h	2	1010
i	2	1000
m	2	0111
n	2	0010
s	2	1011
t	2	0110
l	1	11001
o	1	00110
p	1	10011
r	1	11000
u	1	00111
x	1	10010

The tree corresponding to the above data is:



Verification:

Take a input file where each line contains a character at the beginning, followed by a frequency count. Assume that only 26 lower case alphabets are given and space etc. is not given. The output will be in same form, with the character code following the frequency. Use a single space to separate the fields in a line. Do not use tab or double space. Note that your output may be different from one given below depending on how you break the ties.

Example Input:

```
b       7
a       4
e       4
f       3
h       2
i       2
m       2
n       2
s       2
t       2
l       1
o       1
p       1
r       1
```

u 1
x 1

Example Output:

b	7	111
a	4	010
e	4	000
f	3	1101
h	2	1010
i	2	1000
m	2	0111
n	2	0010
s	2	1011
t	2	0110
l	1	11001
o	1	00110
p	1	10011
r	1	11000
u	1	00111
x	1	10010