Department of
Computer Science and Engineering

Title: Huffman Coding Algorithm

Algorithm Lab

CSE 206

Green University of Bangladesh

1 Objective(s)

- To define Huffman Coding.
- To understand how Huffman Coding works.
- To implement Huffman Coding algorithm.

2 Problem Analysis

The Huffman Coding Algorithm was proposed by David A. Huffman in 1950. It is a lossless data compression mechanism. It is also known as data compression encoding. It is widely used in image (JPEG or JPG) compression. In this section, we will discuss the Huffman encoding and decoding, and also implement its algorithm in a Java program.

We know that each character is a sequence of 0's and 1's and stores using 8-bits. The mechanism is called fixed-length encoding because each character uses the same number of fixed-bit storage.

Here, a question ascends, is it possible to reduce the amount of space required to store a character?

Yes, it is possible by using variable-length encoding. In this mechanism, we exploit some characters that occur more frequently in comparison to other characters. In this encoding technique, we can represent the same piece of text or string by reducing the number of bits.

2.1 Huffman Encoding

Huffman encoding implements the following steps:

- It assigns a variable-length code to all the given characters.
- The code length of a character depends on how frequently it occurs in the given text or string.
- A character gets the smallest code if it frequently occurs.
- A character gets the largest code if it least occurs.

Huffman coding follows a prefix rule that prevents ambiguity while decoding. The rule also ensures that the code assigned to the character is not treated as a prefix of the code assigned to any other character.

There are the following two major steps involved in Huffman coding:

- First, construct a Huffman tree from the given input string or characters or text.
- Assign, a Huffman code to each character by traversing over the tree.

Let's brief the above two steps.

2.2 Huffman Tree

- **Step 1:** For each character of the node, create a leaf node. The leaf node of a character contains the frequency of that character.
- **Step 2:** Set all the nodes in sorted order according to their frequency.
- **Step 3:** There may exist a condition in which two nodes may have the same frequency. In such a case, do the following:
    1. Create a new internal node.
    2. The frequency of the node will be the sum of the frequency of those two nodes that have the same frequency.
    3. Mark the first node as the left child and another node as the right child of the newly created internal node.
- **Step 4:** Repeat step 2 and 3 until all the node forms a single tree. Thus, we get a Huffman tree.

2.3 Huffman Encoding Example

Suppose, we have to encode string **abracadabra**. Determine the following:

- Huffman code for All the characters.
- Average code length for the given String.
- Length of the encoded string.

2.3.1 (i) Huffman Code for All the Characters

In order to determine the code for each character, first, we construct a **Huffman tree**.

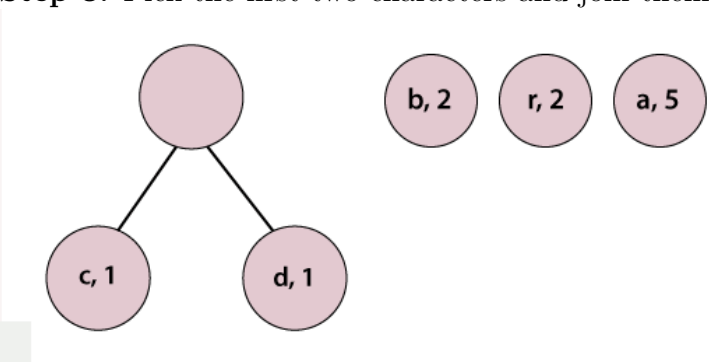
Step 1: Make pairs of characters and their frequencies.

$$(a, 5), (b, 2), (c, 1), (d, 1), (r, 2)$$

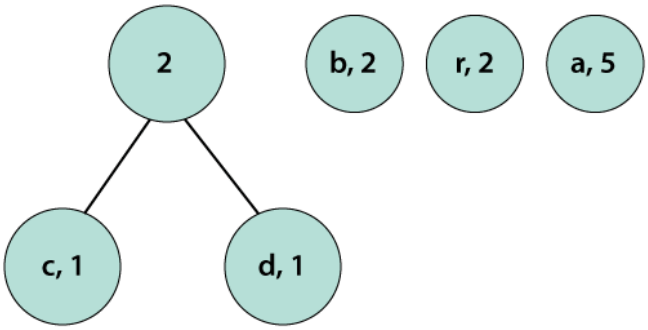
Step 2: Sort pairs with respect to frequency, we get:

$$(c, 1), (d, 1), (b, 2)(r, 2), (a, 5)$$

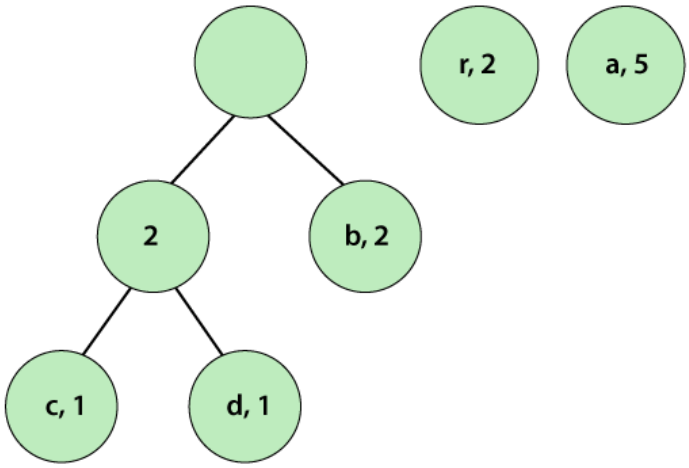
Step 3: Pick the first two characters and join them under a parent node.



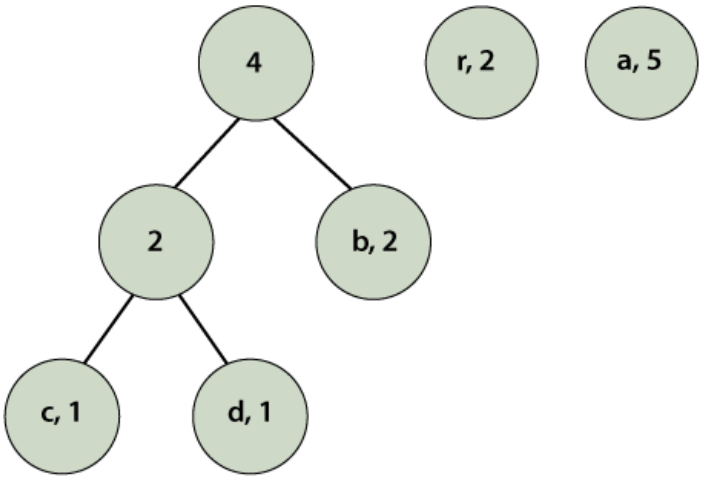
We observe that a parent node does not have a frequency so, we must assign a frequency to it. The parent node frequency will be the sum of its child nodes (left and right) i.e. 1+1=2.



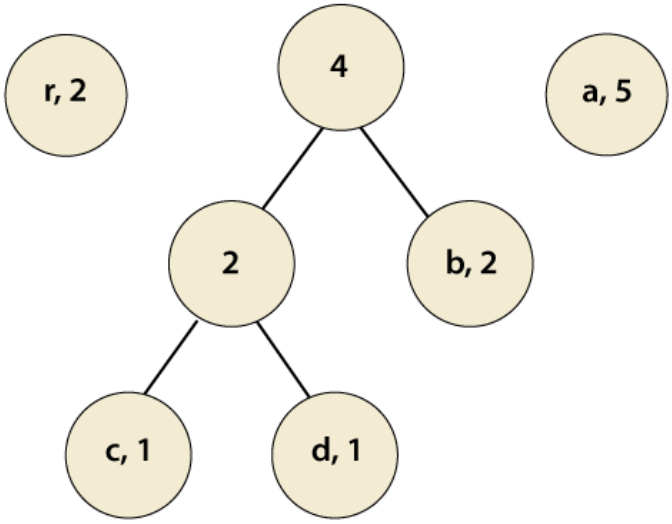
Step 4: Repeat Steps 2 and 3 until, we get a single tree.
We observe that the pairs are already in a sorted (by step 2) manner. Again, pick the first two pairs and join them.



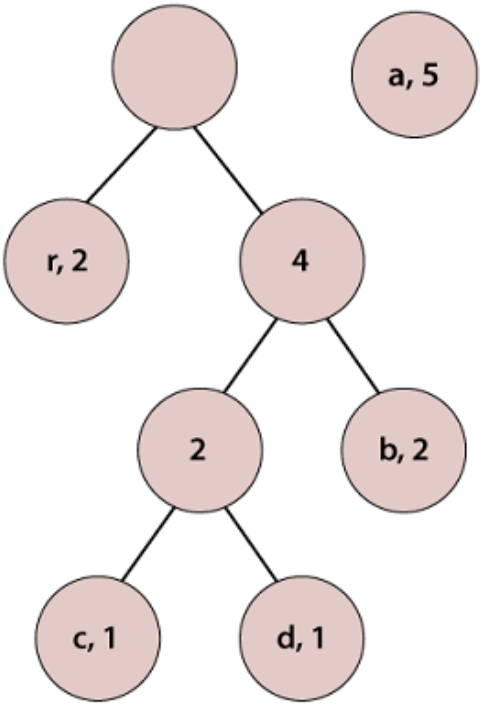
We observe that a parent node does not has a frequency so, we must assign a frequency to it. The parent node frequency will be the sum of its child nodes (left and right) i.e. 2+2=4.



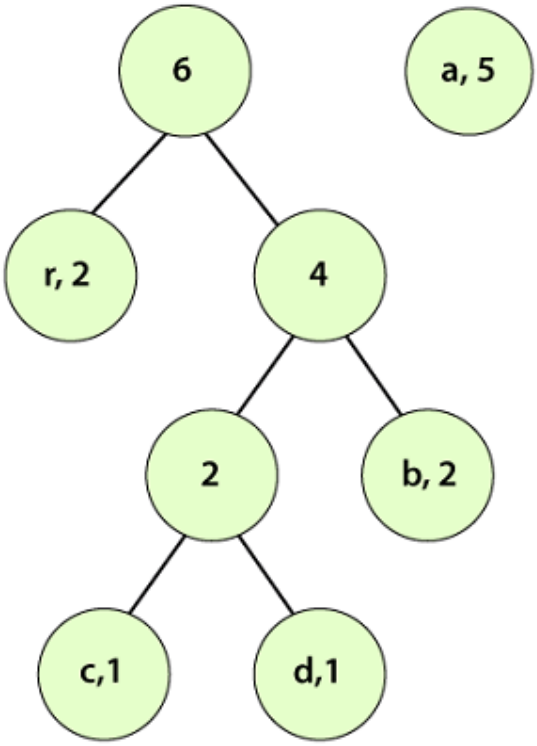
Again, we check if the pairs are in a sorted manner or not. At this step, we need to sort the pairs.



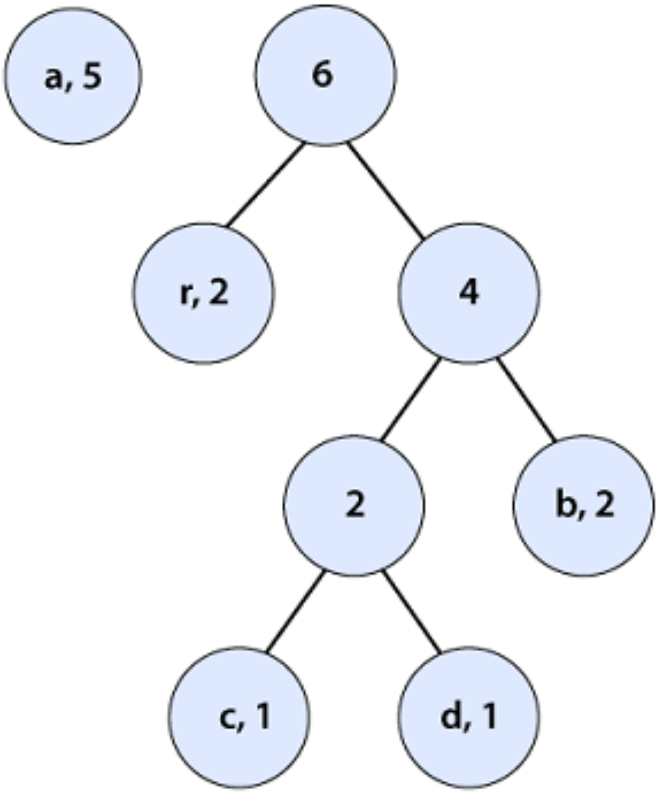
According to step 3, pick the first two pairs and join them, we get:



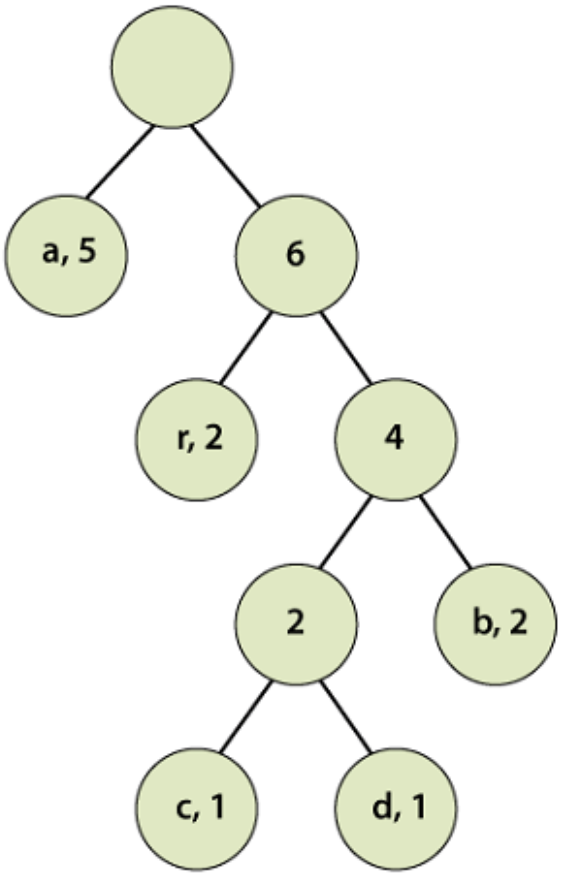
We observe that a parent node does not have a frequency so, we must assign a frequency to it. The parent node frequency will be the sum of its child nodes (left and right) i.e. 2+4=6.



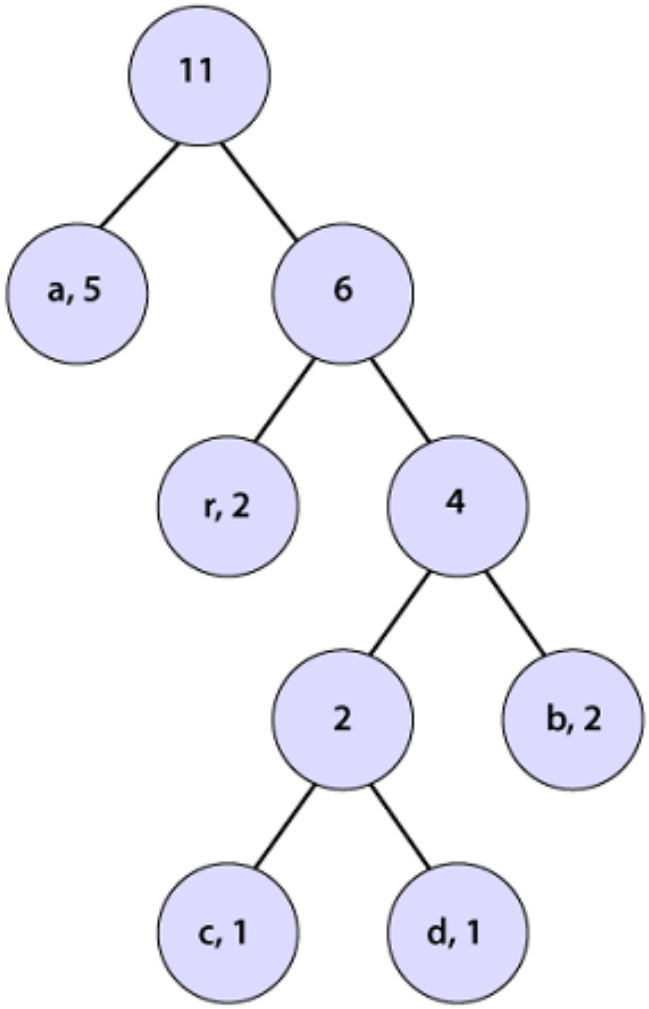
Again, we check if the pairs are in a sorted manner or not. At this step, we need to sort the pairs. After sorting the tree looks like the following:



According to step 3, pick the first two pairs and join them, we get:

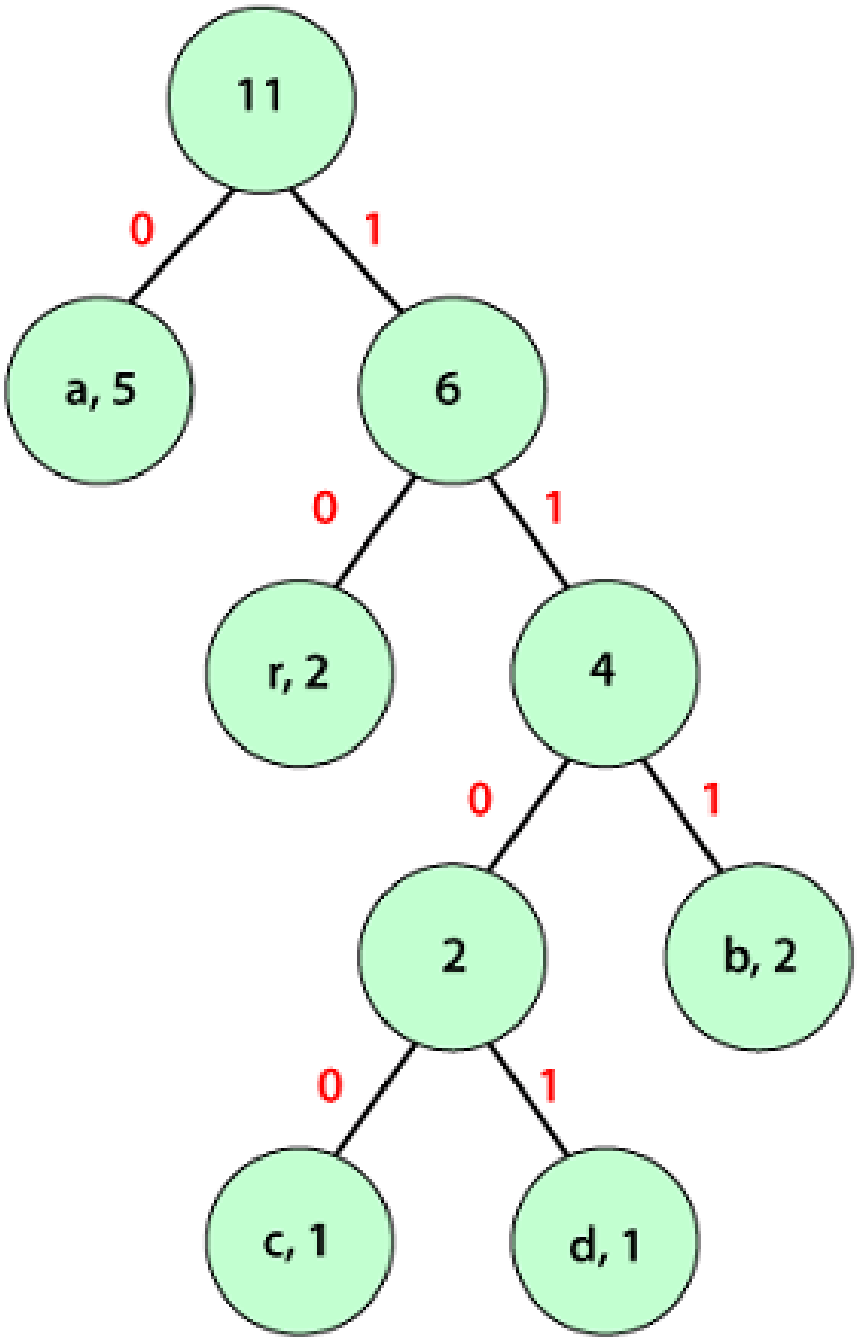


We observe that a parent node does not have a frequency so, we must assign a frequency to it. The parent node frequency will be the sum of its child nodes (left and right) i.e. 5+6=11.



Therefore, we get a single tree.

At last, we will find the code for each character with the help of the above tree. Assign a weight to each edge. Note that each left edge-weighted is 0 and the right edge-weighted is 1.



We observe that input characters are only presented in the leave nodes and the internal nodes have null values. In order to find the Huffman code for each character, traverse over the Huffman tree from the root node to the leaf node of that particular character for which we want to find code. The table describes the code and code length for each character.

Character	Frequency	Code	Code Length
A	5	0	1
B	2	111	3
C	1	1100	4
D	1	1101	4
R	2	10	2

We observe that the most frequent character gets the shortest code length and the less frequent character gets the largest code length.

Now we can encode the string **(abracadabra)** that we have taken above.

```
0 111 10 0 1100 0 1101 0 111 10 0
```

2.3.2 (ii) Average Code Length for the String

The average code length of the Huffman tree can be determined by using the formula given below:

```
Average Code Length = ∑ ( frequency × code length ) / ∑ ( frequency )
```

$= (5 \times 1) + (2 \times 3) + (1 \times 4) + (1 \times 4) + (2 \times 2) \ / (5+2+1+1+2)$
= 2.09090909

2.3.3 (iii) Length of the Encoded String

The length of the encoded message can be determined by using the following formula:

```
length= Total number of characters in the text x Average code length per character
```

= 11 x 2.09090909

= 23 bits

3 Algorithm

Algorithm 1: Huffman Coding

Algorithm *Huffman(c)*:
  $n = |c|$
  $Q = c$
  for $i < -1$ *to* $n - 1$ **do**
    $temp = get\_node()$
    $left\,[temp]\ get\_min(Q)$
    $right\,[temp]\ get\_min(Q)$
    $a = left\,[temp]$
    $b = right\,[temp]$
    $F\,[temp] = f[a] + [b]$
    $insert(Q, temp)$
  end
return get_min(0)

4 Implementation

```
// Huffman Coding in Java

import java.util.PriorityQueue;
import java.util.Comparator;

class HuffmanNode {
  int item;
  char c;
  HuffmanNode left;
  HuffmanNode right;
}

// For comparing the nodes
class ImplementComparator implements Comparator<HuffmanNode> {
  public int compare(HuffmanNode x, HuffmanNode y) {
    return x.item - y.item;
  }
}

// IMplementing the huffman algorithm
public class Main {
  public static void printCode(HuffmanNode root, String s) {
```

```
23      if (root.left == null && root.right == null && Character.
           isLetter(root.c)) {
24
25        System.out.println(root.c + "    |  " + s);
26
27        return;
28      }
29      printCode(root.left, s + "0");
30      printCode(root.right, s + "1");
31    }
32
33    public static void main(String[] args) {
34
35      int n = 5;
36      char[] charArray = { 'A', 'B', 'C', 'D', 'R' };
37      int[] charfreq = { 5, 2, 1, 1, 2 };
38
39      PriorityQueue<HuffmanNode> q = new PriorityQueue<HuffmanNode>(n
           , new ImplementComparator());
40
41      for (int i = 0; i < n; i++) {
42        HuffmanNode hn = new HuffmanNode();
43
44        hn.c = charArray[i];
45        hn.item = charfreq[i];
46
47        hn.left = null;
48        hn.right = null;
49
50        q.add(hn);
51      }
52
53      HuffmanNode root = null;
54
55      while (q.size() > 1) {
56
57        HuffmanNode x = q.peek();
58        q.poll();
59
60        HuffmanNode y = q.peek();
61        q.poll();
62
63        HuffmanNode f = new HuffmanNode();
64
65        f.item = x.item + y.item;
66        f.c = '-';
67        f.left = x;
68        f.right = y;
```

```
69          root = f;
70
71          q.add(f);
72        }
73        System.out.println(" Char | Code ");
74        System.out.println("-------------");
75        printCode(root, "");
76      }
77  }
```

5 Sample Input/Output (Compilation, Debugging & Testing)

Output:

Char | Code

A | 0
B | 10
R | 110
C | 1110
D | 1111

6 Discussion & Conclusion

Huffman coding is a lossless data compression algorithm that assigns variable-length binary codes to symbols based on their frequency of occurrence. The algorithm constructs a Huffman tree to determine the optimal prefix code.

7 Lab Task (Please implement yourself and show the output to the instructor)

Construct a Huffman tree by using these nodes.

Value	A	B	C	D	E	F
Frequency	5	25	7	15	4	12

8 Lab Exercise (Submit as a report)

A file contains the following characters with the frequencies as shown. If Huffman Coding is used for data compression, determine-

1. Huffman Code for each character

2. Average code length

3. Length of Huffman encoded message (in bits)

Characters	A	E	I	O	U	S	T
Frequencies	10	15	12	3	4	13	1

9 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.