Huffman algorithm

Letter	Probability
‘a’	12
‘b’	40
‘c’	15
‘d’	8
‘e’	25

Sort

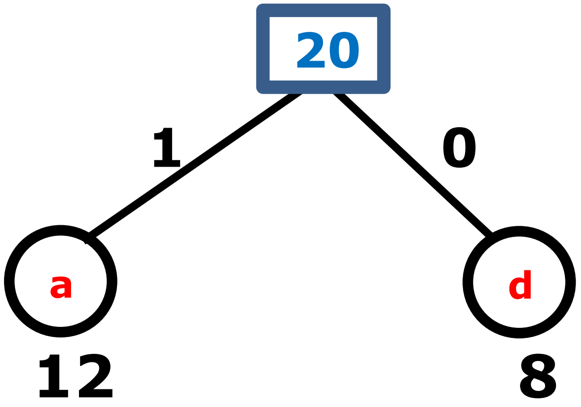
Letter	Probability
‘d’	8
‘a’	12
‘c’	15
‘e’	25
‘b’	40

Input

```
int freq1[]      = { 8, 12, 15, 25, 40};
char letter1[]  = {'d', 'a', 'c', 'e', 'b'};
```

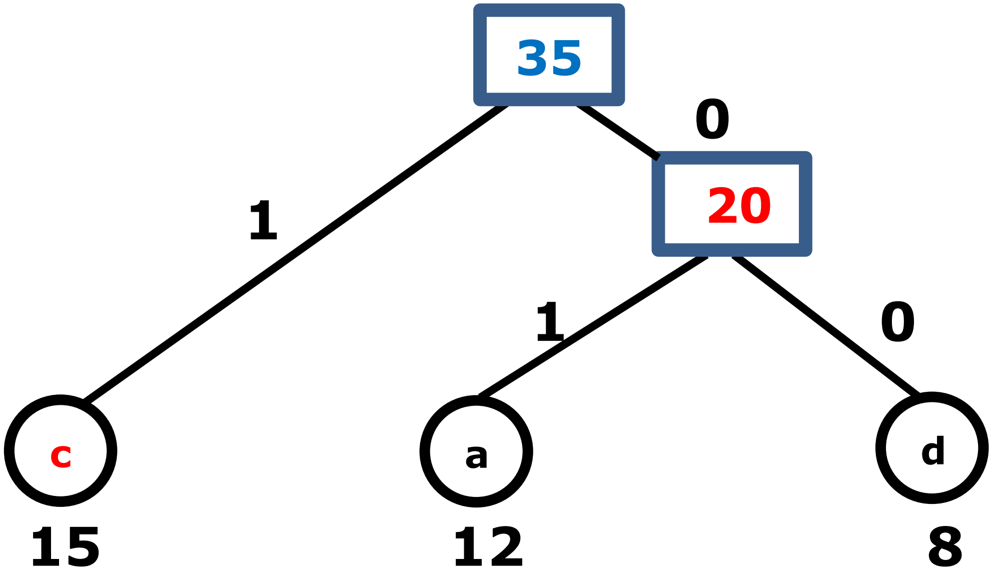
Build Huffman tree

1.



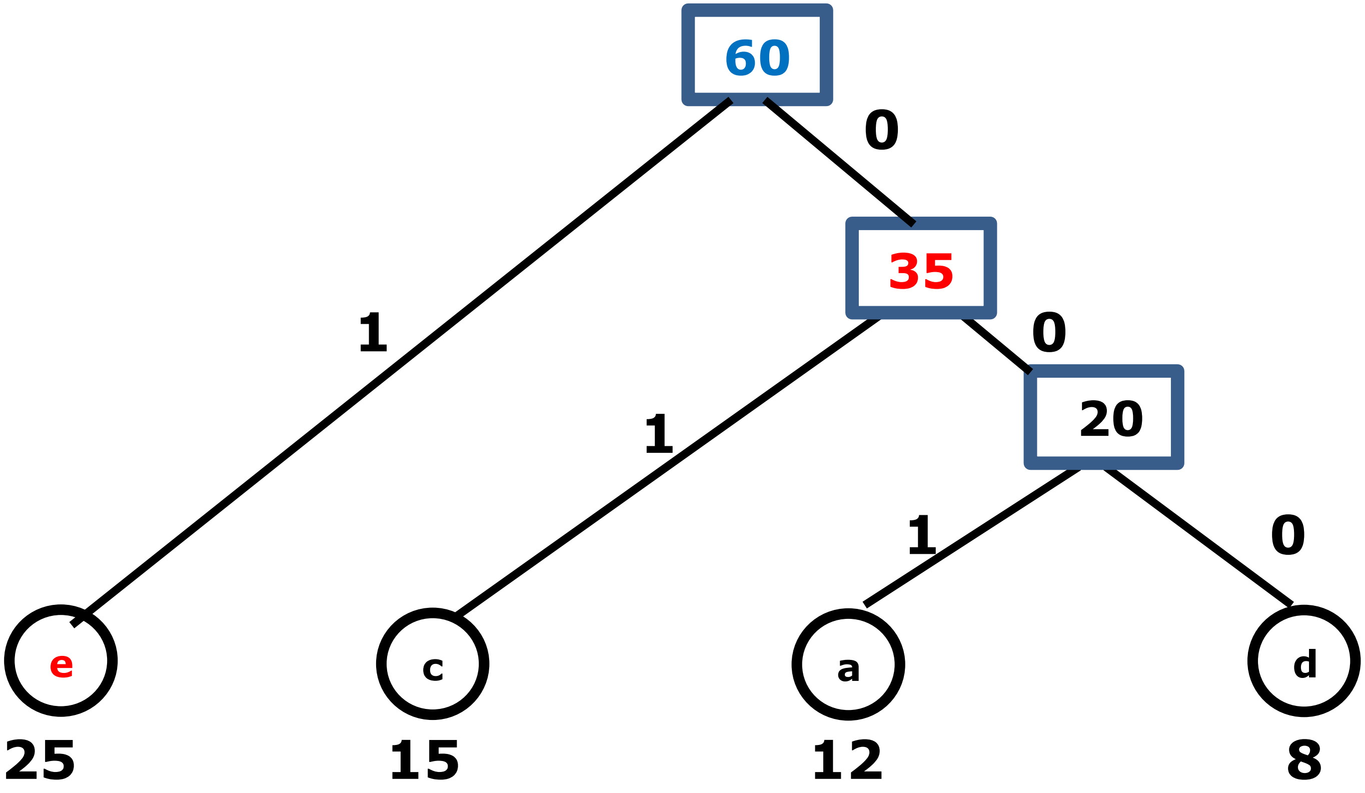
##	son	son	father	probability	letter	code
1			20	8	d	
2			20	12	a	
3				15	c	
4				25	e	
5				40	b	
6	8	12	20	20	20	
7						
8						
9						

2.



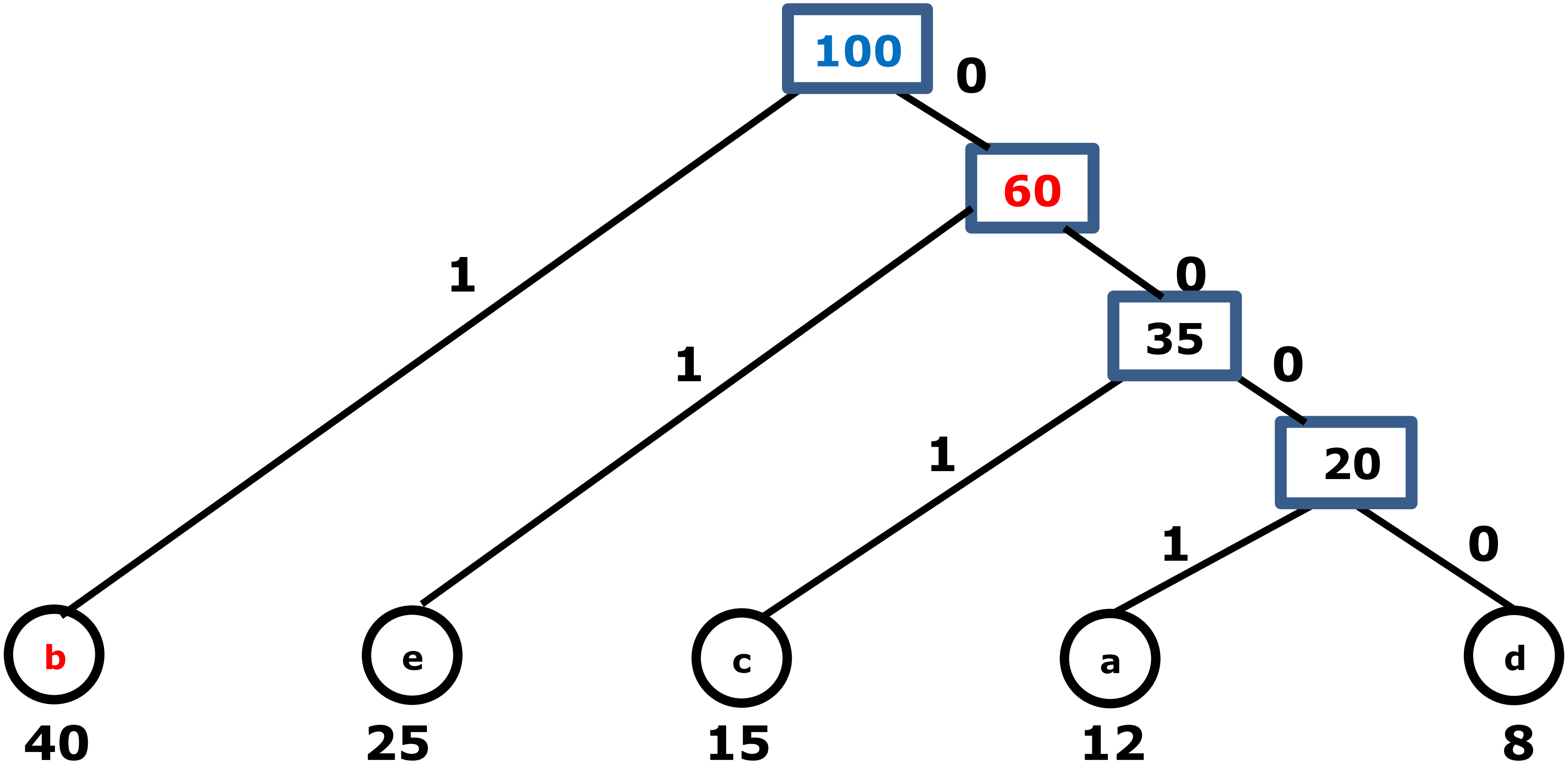
##	son	son	father	probability	letter	code
1			20	8	d	
2			20	12	a	
3			35	15	c	
4				25	e	
5				40	b	
6	8	12	35	20	20	
7	c	20		35	35	
8						
9						

3.



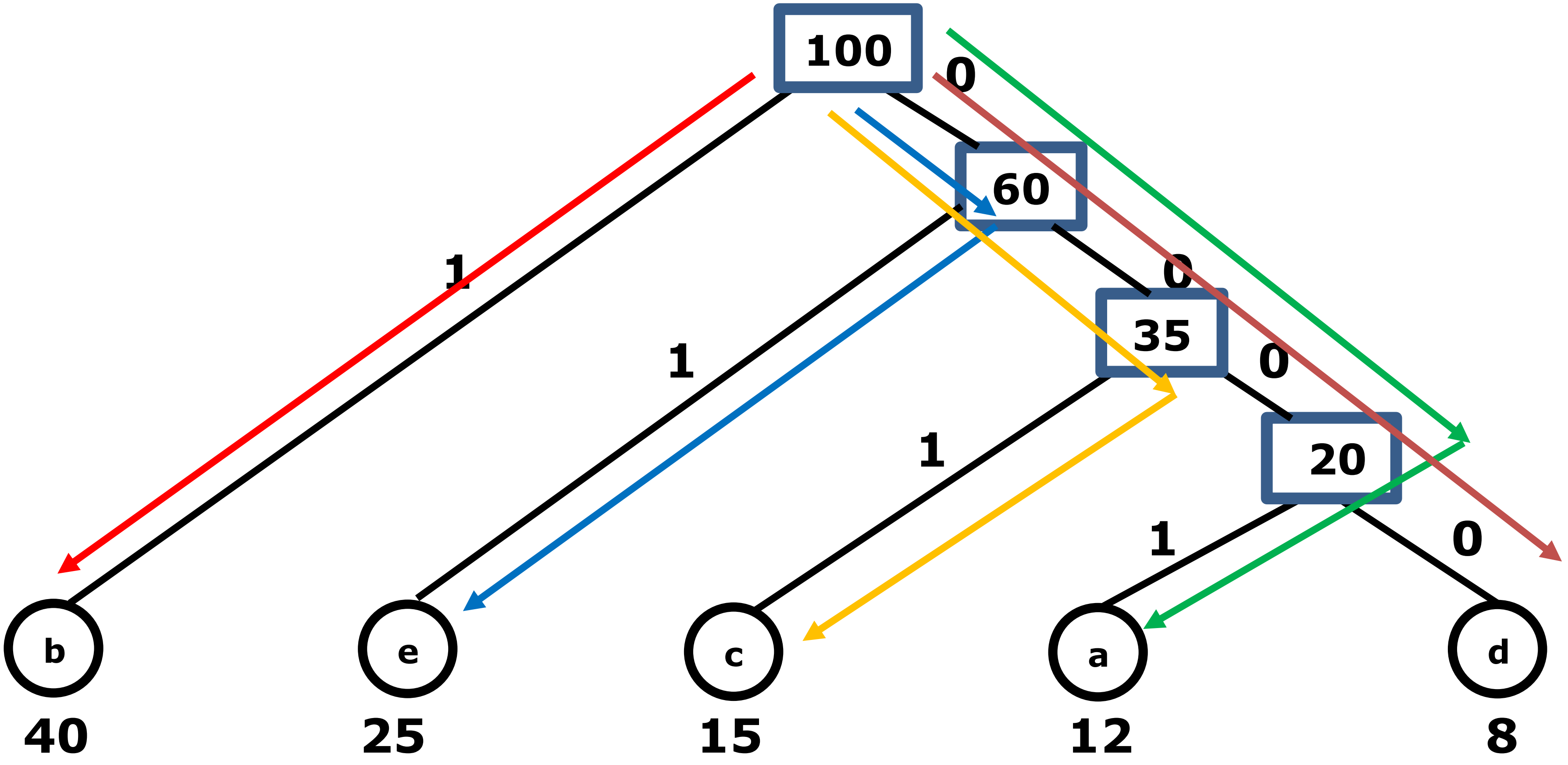
##	son	son	father	probability	letter	code
1			20	8	d	
2			20	12	a	
3			35	15	c	
4				25	e	
5				40	b	
6	8	12	35	20	20	
7	c	20	60	35	35	
8	e	35		60	60	
9						

4.



##	son	son	father	probability	letter	code
1			20	8	d	
2			20	12	a	
3			35	15	c	
4				25	e	
5				40	b	
6	8	12	35	20	20	
7	c	20	60	35	35	
8	e	35	100	60	60	
9	b	60		100	100	

5.



code

##	son	son	father	probability	letter	code
1			20	8	d	0000
2			20	12	a	0001
3			35	15	c	001
4				25	e	01
5				40	b	1
6	8	12	35	20	20	
7	c	20	60	35	35	
8	e	35	100	60	60	
9	b	60		100	100	

PREFIX (FREE) CODE

Example 1

A	1100
B	110
C	0

A B A

11001101100

11001101100

B C B A

Example 2

A	1100
B	100
C	0

A B A

11001101100

A **prefix code** is a type of code system (typically a variable-length code) distinguished by its possession of the "prefix property", which requires that there is no code word in the system that is a prefix (initial segment) of any other code word in the system.

Huffman algorithm

```
private class Node    {

    int      element;                                    // frequency
    int      index;

    public Node(int element, int index){
         this.element        = element;
         this.index          = index;               //0-left child, 1-right child
    }

    public String toString(){
         return "(w="+_element+",i="+_index+")";
    }
}
```

```
public class Huffman2Queue {
    int                              mat[][];
    char                             letters[];
    int                              n, nMax;
    ArrayBlockingQueue<Node>         q1;
    ArrayBlockingQueue<Node>         q2;
    String                           code[];
```

```
Huffman2Queue(){                              // constructor

    n = freq.length;
    letters = new char[n];
    code = new String[n];
    nMax = 2*_n - 1;
    mat = new int[2*_n-1][4];                 //table: _mat[][0]- parent index
    q1 = new ArrayBlockingQueue<Node>(nMax);
    q2 = new ArrayBlockingQueue<Node>(nMax);

    for (int i = 0; i < n; i++) {
        mat[i][0] = freq[i];
        letters[i] = letters[i];
        code[i] = new String();
        q1.add(new Node(freq[i], i));
    }
    System.out.println(q1.toString());
}
```

```
public void buildTable(){
     Node x1 = q1.remove();
     Node x2 = q1.remove();
     int parent = n;
     int weight = x1._element + x2._element;
     q2.add(new Node(weight, parent));
     mat[parent][0] = weight;
     mat[parent][1] = x1.index;                     //left child (0)
     mat[parent][2] = x2.index;                     //right child (1)
     mat[x1._index][3] = parent;                    // parent
     mat[x2._index][3] = parent;                    // parent
     parent++;

     while (q1.size() + q2.size()>1){
        x1 = nextMin();
        x2 = nextMin();
        weight = x1.element + x2.element;
        q2.add(new Node(weight, parent));
        mat[parent][0] = weight;
        mat[parent][1] = x1._index;                 //left child (0)
        mat[parent][2] = x2._index;                 //right child (1)
        mat[x1._index][3] = parent;                 // parent
        mat[x2._index][3] = parent;                 // parent
        parent++;
     }
   }
```

```
private Node nextMin(){
        Node x, y;
        if (q1.isEmpty())
            x = q2.remove();
        else
            if (q2.isEmpty())
                x = q1.remove();
            else{
                x = q1.peek();
                y = q2.peek();
                if (x.element > y.element)
                    x = q2.remove();
                else
                    x = q1.remove();
            }
        return x;
    }
```

```
// build the Huffman's Code for all letters
public void huffmanCode(){
    for (int i=0; i<_n; i++){
        int child = i;
        int parent = mat[child][3];
        while(parent!=0){
         if (mat[parent][1]==child)
             code[i]=_code[i]+"0";
         else
             code[i]=_code[i]+"1";
         child = parent;
         parent = mat[child][3];
        }
    }
  }
```

```
// print the table
public void printMat(){
    for (int i=0; i<_n*2-1; i++){
        for (int j=0; j<4; j++){
            System.out.print(_mat[i][j]+" ");
        }
        System.out.println();
    }
}

// print the Huffman's Codes
public void printCode(){
    for (int i=0; i<n; i++){
        System.out.println(letters[i]+": "+code[i]);
    }
}
```

```
public static void main(String[] args) {
    int freq1[]        = {8,12,15,25,40};
    char letter1[]     = {'d','a','c','e','b'};
    Huffman2Queue hq = new Huffman2Queue(freq1, letter1);
    hq.buildTable();
    hq.printMat();
    hq.huffmanCode();
    hq.printCode();
}
```

Result :

```
run:

[(w=8,i=0), (w=12,i=1), (w=15,i=2), (w=25,i=3), (w=40,i=4)]

  8 0 0 5
 12 0 0 5
 15 0 0 6
 25 0 0 7
 40 0 0 8
 20 0 1 6
 35 2 5 7
 60 3 6 8
100 4 7 0

d: 0111
a: 1111
c: 011
e: 01
b: 0
```

```
BUILD SUCCESSFUL (total time: 1 second)
```