Homework 3

Student Name: ZhangLe(张乐)

Student ID: 201628013229047

Question 1

Pseudo-code

```C++
bool Havel_Hakimi(arr,n){
    for(int i=0; i<n-1; ++i){
        sort(arr+i,arr+n,greater<int>());
        if(i+arr[i] >= n) return false;
        for(int j=i+1; j<=i+arr[i] ; ++j){
            --arr[j];
            if(arr[j] < 0) return false;
        }
    }
    if(arr[n-1]!=0) return false;
    return true;
}
```

Provement

算法正确性依赖于Havel-Hakimi定理。下面对Havel-Hakimi定理进行证明。

我对序列进行排序，使得$d_1 \geq d_2 \geq \ldots \geq d_n$ ，Havel-Hakimi定理提出如果$D = \{d_1, d_2, \ldots, d_n\}$可简单图，那么$D' = \{d_2 - 1, d_3 - 1, \ldots, d_{d_1+1} - 1, d_{d_1+2}, d_{d_1+3}, \ldots, d_n\}$可简单图。我们的算法用到这个定理的逆否命题。

若D课简单图，设得到的简单图为G。分两种情况考虑：

1.若G中存在边$(v_1, v_2), (v_1, v_3) \ldots (v_1, v_{d_1+1})$，则去掉这些变几个得到简单图$G'$，于是$D'$可简单图为$G'$
2.如果存在点$v_i, v_j (i < j)$使得(v_1, v_i)不在G中。此时因为$d_i >= d_j$，必存在k使得(v_i, v_k)在G中但(v_j, v_k)不在G中。此时我们可以令$G_t = G - \{(v_i, v_k), (v_1, v_j)\} + \{(v_k, v_j), (v_1, v_i)\}$。而$G_t$对应的$D$和原来相同。经过有限次的调整，我们将回到情况1。

Complexity

最坏情况下，即序列可简单图，对于d_i，对剩下的d_{i+}做减法的次数不会超过i。算法每次循环问题规模减一，即$T(n) = T(n-1) + n$。所以算法时间复杂度为$O(n^2)$

Question 4

Pseudo-code

```
max(A[1...n],B[1...n])
    res<=1
    Sort(A);Sort(B)
    for i=1...n
        res*=pow(A[i],B[i]);
    return res
```

Provement

设$\pi = \prod_{i=1}^{n} a_i^{b_j}$ 假设$a_1 \geq a_2 \geq \ldots \geq a_n$，$b_1 \geq b_2 \geq \ldots \geq b_n$。 如果存在项 $a_i^{b_j}$ 和$a_j^{b_i} (i \neq j)$，显然 $a_i^{b_i} a_j^{b_j} > a_i^{b_j} a_j^{b_i}$。如果不存在这项，则可以经过有限次两两交换变换为$\prod_{i=1}^{n} a_i^{b_i}$，在交换过程中，$\pi$不断增加。

Complexity

算法分为两步，第一步排序数组，第二步计算累乘。第一步时间复杂度为$O(n \log n)$，第二步为$O(n)$。所以算法综合时间复杂度为$O(n \log n)$

Question 5

Java

```java
package ucas.algorithm;

import java.io.*;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;
import java.util.Vector;

/**
 * Created by zl on 2016/10/27.
 */
public class Huffman {
    public String[] encodingTable = new String[256];

```

```
    public Node huffmanTree = null;

    private void buildHuffmanTree(int[] freq) {
        List<Node> list = new LinkedList<>();

        for (int i = 0; i < freq.length; i++) {
            if (freq[i] != 0)
                list.add(new Node((char) i, freq[i]));
        }
        while (list.size() != 1) {
            Collections.sort(list);
            Node n1 = list.get(0);
            Node n2 = list.get(1);
            Node r = n1.freq < n2.freq ? new Node((char) 0, n1.freq + n2.freq,
            list.add(r);
            list.remove(0);
            list.remove(0);
        }
        this.huffmanTree = list.get(0);

        travel(this.huffmanTree, "");

    }

    private void travel(Node r, String code) {
        if (r.isLeaf()) {
            encodingTable[r.val] = code;
            return;
        }
        travel(r.left, code + 0);
        travel(r.right, code + 1);
    }

    char[] encode(char[] bytes) {

        int[] freq = new int[256];
        for (char b : bytes) {
            freq[b] += 1;
        }
        buildHuffmanTree(freq);

        StringBuffer sb = new StringBuffer();
        for (char b : bytes) {
            sb.append(this.encodingTable[b]);
        }
        return str2bytes(sb.toString());
    }

```

```
63
64      char[] decode(char[] bytes) {
65          String code = bytes2str(bytes);
66          List<Character> result = new LinkedList<Character>();
67          Node curr = this.huffmanTree;
68          for (byte c : code.getBytes()) {
69              if (c == '0') {
70                  curr = curr.left;
71              } else {
72                  curr = curr.right;
73              }
74              if (curr.isLeaf()) {
75                  result.add(curr.val);
76                  curr = this.huffmanTree;
77              }
78          }
79          char[] re = new char[result.size()];
80          int i = 0;
81          for (Character c : result) {
82              re[i++] = c;
83          }
84          return re;
85      }
86
87      char[] str2bytes(String encode) {
88          char[] result = new char[(int) Math.ceil(encode.length() / 8.0)];
89          for (int i = 0; i < result.length; i++) {
90              for (int j = 0; j < 8; j++) {
91                  result[i] <<= 1;
92                  if (i * 8 + j < encode.length())
93                      result[i] |= encode.charAt(i * 8 + j) == '1' ? 1 : 0;
94              }
95          }
96          return result;
97      }
98
99      String bytes2str(char[] bytes) {
100         StringBuffer encode = new StringBuffer();
101         char[] result = new char[(int) Math.ceil(encode.length() / 8.0)];
102         for (int i = 0; i < bytes.length; i++) {
103             for (int j = 7; j >= 0; j--) {
104                 encode.append(((bytes[i] >> j) & 1) > 0 ? '1' : '0');
105             }
106         }
107         return encode.toString();
108     }
109
110     public static void main(String args[]) throws Exception {
```

```
111
112         Huffman huffman = new Huffman();
113
114         Vector<Character> vector = new Vector<Character>();
115         DataInputStream is =
116                 new DataInputStream(
117                         new BufferedInputStream(
118                                 new FileInputStream("d:\\\\Aesop_Fables.txt")))
119         try {
120             while (true) {
121                 vector.add((char) is.readUnsignedByte());
122             }
123         } catch (Exception e) {
124
125         }
126
127         char[] origin = new char[vector.size()];
128         for (int i = 0; i < origin.length; i++) {
129             origin[i] = vector.get(i);
130         }
131
132         char code[] = huffman.encode(origin);
133
134         DataOutputStream out = null;
135         out = new DataOutputStream(new FileOutputStream("d:\\\\out"));
136         for (char c : code) {
137             out.write(c);
138         }
139         out.close();
140
141         char ex[] = huffman.decode(code);
142         out = new DataOutputStream(new FileOutputStream("d:\\\\ex.txt"));
143         for (char c : ex) {
144             out.write(c);
145         }
146         out.close();
147
148         System.out.println("压缩率: " + code.length / (origin.length + 0.0));
149     }
150 }
151
152 class Node implements Comparable<Node> {
153     public char val;
154     public int freq;
155     public Node left = null;
156     public Node right = null;
157
158     public Node(char val, int freq) {
```

```
        this.val = val;
        this.freq = freq;
    }

    public Node(char val, int freq, Node left, Node right) {
        this.val = val;
        this.freq = freq;
        this.left = left;
        this.right = right;
    }

    public boolean isLeaf() {
        return left == null && right == null;
    }

    @Override
    public int compareTo(Node o) {
        return this.freq - o.freq;
    }
}
```