**2018 Fall Data Compression Homework #1**

EE 248583
Advisor: 電子所 高立人 副教授

**106368002 張昌祺 Justin, Chang-Qi Zhang**
justin840727@gmail.com
Due Date: November 12 2018

## Problem 1 Entropy

Let X be a random variable with an alphabet $H = \{1, 2, 3, 4, 5\}$. Please determine $H(X)$ for the following three cases of probability mass function $p(i) = prob[X = i]$. (15%)

(a) $P(1) = P(2) = 1/2$:

**Ans**

$$
\begin{aligned}
H(X) &= -(P(1)\log_2 P(1) + P(2)\log_2 P(2)) \\
&= -(0.5\log_2(0.5) + 0.5\log_2(0.5)) \\
&= -(-0.5 - 0.5) \\
&= 1\ bits/symbol
\end{aligned}
$$

(b) $P(i) = 1/4, for\ i = 1, 2, 3, and\ p(4) = p(5) = 1/8$:

**Ans**

$$
\begin{aligned}
H(X) &= -(3 \times P(1)\log_2 P(1) + P(4)\log_2 P(4) + P(5)\log_2 P(5)) \\
&= -(3 \times 0.25\log_2(0.25) + 2 \times 0.125\log_2(0.125)) \\
&= -(-1.5 - 0.75) \\
&= 2.25\ bits/symbol
\end{aligned}
$$

(c) $P(i) = 2^{-i}, for\ i = 1, 2, 3, 4, and\ p(5) = 1/16$:

**Ans**

$$
\begin{aligned}
H(X) &= -(\sum_{i=1}^{4} 2^{-i}\log_2 2^{-i} + \frac{1}{16}\log_2\frac{1}{16}) \\
&= -(0.5 \times (-1) + 0.25 \times (-2) + 0.125 \times (-3) + 0.0625 \times (-4) + 0.0625 \times (-4)) \\
&= 1.875\ bits/symbol
\end{aligned}
$$

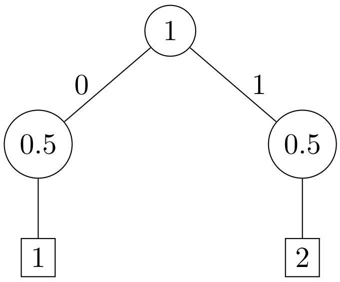
## Problem 2 Huffman Code

Design a Huffman code C for the source in Problem 1. (15%)

(a) Specify your codewords for individual pmf model in Problem 1.

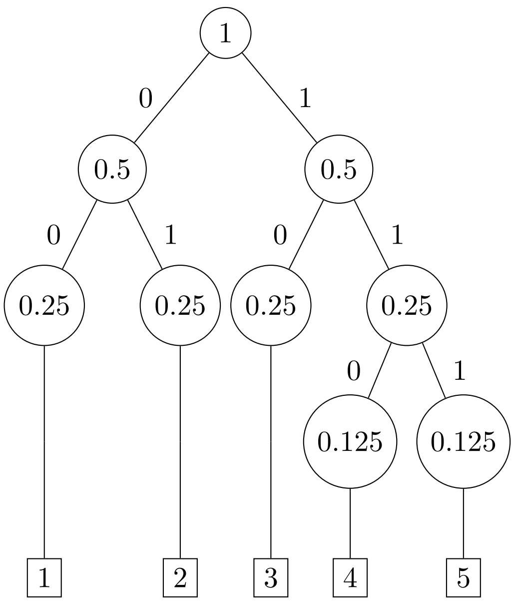
**Ans**

**1.(a)**



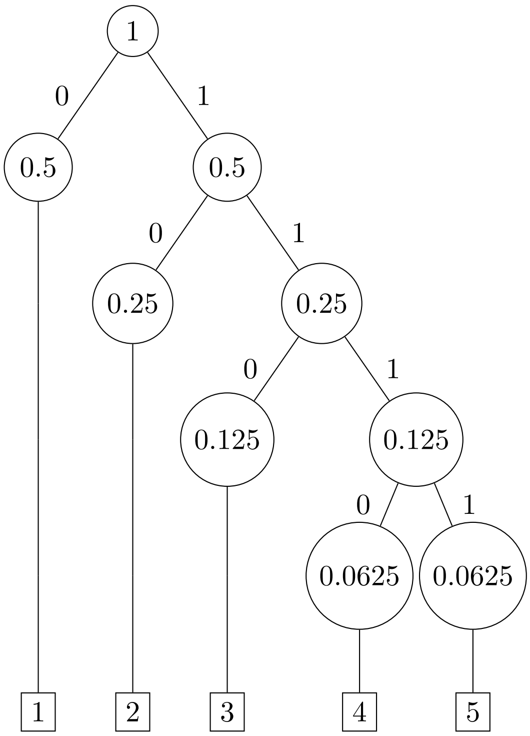

| Alphabet | P | Codeword |
|---|---|---|
| 1 | 0.5 | 0 |
| 2 | 0.5 | 1 |

**1.(b)**




| Alphabet | P | Codeword |
|---|---|---|
| 1 | 0.25 | 00 |
| 2 | 0.25 | 01 |
| 3 | 0.25 | 10 |
| 4 | 0.125 | 110 |
| 5 | 0.125 | 111 |

**1.(c)**




| Alphabet | P | Codeword |
|---|---|---|
| 1 | 0.5 | 0 |
| 2 | 0.25 | 10 |
| 3 | 0.125 | 110 |
| 4 | 0.0625 | 1110 |
| 5 | 0.0625 | 1111 |

(b) Compute the expected codeword length and compare with the entropy for your codes in (a).

**Ans**

**1.(b)**

$$\begin{aligned} Expected\ codeword\ length &= 0.5 \times 1 + 0.5 \times 1 \\ &= 1\ bits/symbol\ \textbf{(Equal Entropy)} \end{aligned}$$

**1.(b)**

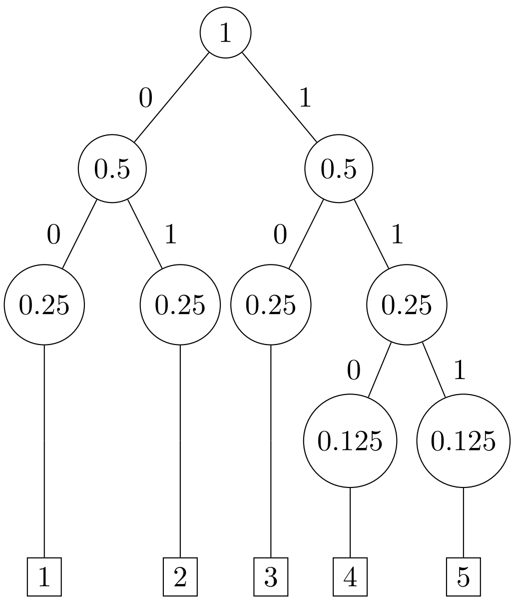
$$\begin{aligned} Expected\ codeword\ length &= 0.25 \times 2 + 0.25 \times 2 + 0.25 \times 2 + 0.125 \times 3 + 0.125 \times 3 \\ &= 2.25\ bits/symbol\ \textbf{(Equal Entropy)} \end{aligned}$$

**1.(c)**

$$\begin{aligned} Expected\ codeword\ length &= 0.5 \times 1 + 0.25 \times 2 + 0.125 \times 3 + 0.0626 \times 4 + 0.0625 \times 4 \\ &= 4.125\ bits/symbol\ \textbf{(NOT Equal Entropy)} \end{aligned}$$

(c) Design a code with minimum codeword length variance for the pmf model in Problem 1.(b)

**Ans**




| Alphabet | P | Codeword |
|---|---|---|
| 1 | 0.25 | 00 |
| 2 | 0.25 | 01 |
| 3 | 0.25 | 10 |
| 4 | 0.125 | 110 |
| 5 | 0.125 | 111 |

## Problem 3 Empirical Distribution C++

Empirical distribution. In the case a probability model is not known, it can be estimated from empirical data. Let's say the alphabet is $H = \{1, 2, 3, \dots , m\}$ . Given a set of observations of length $N$ , the empirical distribution is given by $p = total\ number\ of\ symbol\ 1/N,\ for\ i = 1, 2, 3, ..., m$. Please determine the empirical distribution for **santaclaus.txt**, which is an ASCII file with only lower-cased English letters (i.e., $a \sim z$), space and CR (carriage return), totally 28 symbols. The file can be found on the class web site. Compute the entropy. (14%)

**Ans**

The source code for this problem are available at `https://github.com/justin-changqi/2018_fall_data_compression.git`. Please check README.md to know how to execute the code. After I executed the program the entropy is 4.12 bits/symbol. Empirical distribution shows in Figure 2

```
> ./huffman_code
      Alphaba       SP        o        t        e        n        a        s        i        u       CR        c        y
 Total Number       83       45       41       36       34       29       24       23       20       18       16       15
          PMF    0.171   0.0926   0.0844   0.0741     0.07   0.0597   0.0494   0.0473   0.0412    0.037   0.0329   0.0309
          CDF    0.171    0.263    0.348    0.422    0.492    0.551    0.601    0.648    0.689    0.726    0.759     0.79

      Alphaba        h        w        g        r        l        b        m        d        k        p        f        v
 Total Number       15       15       14       12       10        9        7        7        6        3        3        1
          PMF   0.0309   0.0309   0.0288   0.0247   0.0206   0.0185   0.0144   0.0144   0.0123  0.00617  0.00617  0.00206
          CDF    0.821    0.852    0.881    0.905    0.926    0.944    0.959    0.973    0.986    0.992    0.998        1

Entropy: 4.12 bits/symbol
```

Figure 1: Statistics result for **santaclaus.txt**

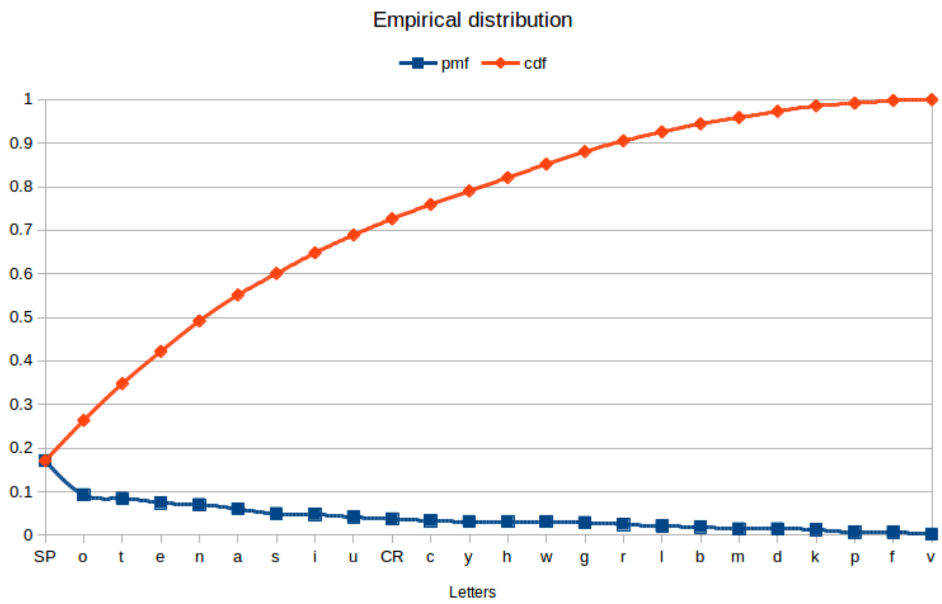



Figure 2: Empirical distribution for **santaclaus.txt**

## Problem 4 Huffman Code Encode C++

Write a program that designs a Huffman code for the given distribution in Problem 3. (14

**Ans**

The program for this problem was wrote together with Problem 3. The execute print the Huffman encode result as Figure 3

```
=================== Codeword Table =================

        SP:        111          t:        1101
         l:     110011          b:      110010
        CR:      11000          e:        1011
         n:       1010          c:       10011
         y:      10010          h:       10001
         w:      10000          a:        0111
         g:      01101          m:      011001
         d:     011000          f:    01011111
         v:   01011110          p:     0101110
         k:     010110          r:       01010
         s:       0100          o:         001
         i:       0001          u:        0000
```

Figure 3: Huffman encode result for **santaclaus.txt**

## Problem 5 Adaptive Huffman Tree

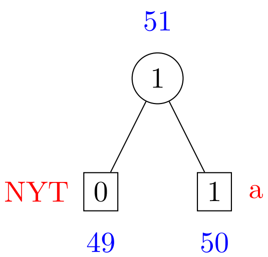
Let X be a random variable with an alphabet $H$ , i.e., the 26 lower-case letters. Use adaptive Huffman tree to find the binary code for the sequence **a a b b a**. (24%)
You are asked to use the following 5 bits fixed-length binary code as the initial codewords for the 26 letters. That is
a: 00000
b: 00001
⋮
z: 11001
**Note**: Show the Huffman tree during your coding process.

**Ans**

1. Initial step:
   $Total\ nodes = 2m - 1 = 26 \times 2 - 1 = 51$

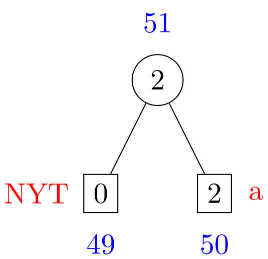
   
   

2. **a** encoded:

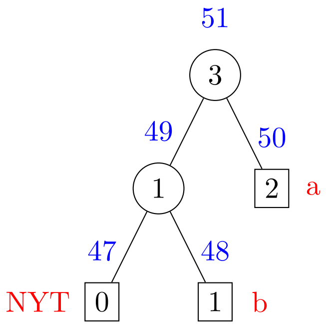
   00000
   a

3. **a a** encoded:

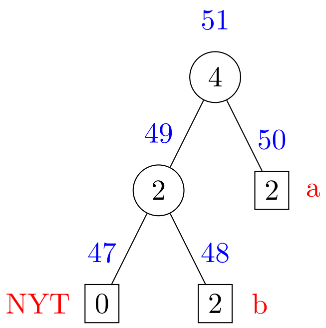
   00000 1
   a a

4. **a a b** encoded:



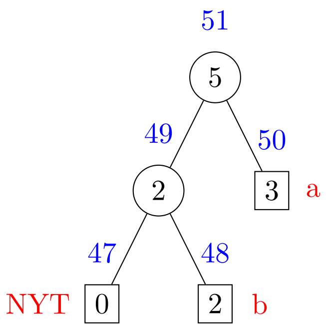

| | | | |
|---|---|---|---|
| 00000 | 1 | 0 | 00001 |
| a | a | NYT | b |

5. **a a b b** encoded:




| | | | | |
|---|---|---|---|---|
| 00000 | 1 | 0 | 00001 | 01 |
| a | a | NYT | b | b |

6. **a a b b a** encoded:




| | | | | | |
|---|---|---|---|---|---|
| 00000 | 1 | 0 | 00001 | 01 | 1 |
| a | a | NYT | b | b | a |

## Problem 6 Golomb Encoding and Decoding.

(a) Find the Golomb code of n=21 when m=4.

**Ans**

$$
\begin{aligned}
&2^{\lceil \log_2^m \rceil} - m = 2^2 - 4 = 0 \\
&\textit{encoded } 21 = 21/4 = 5 \ldots 1 = 111110\ 01
\end{aligned}
$$

(b) Find the Golomb code of n=14 when m=4.

**Ans**

$$
\begin{aligned}
&2^{\lceil \log_2^m \rceil} - m = 2^2 - 4 = 0 \\
&\textit{encoded } 14 = 14/4 = 3 \ldots 2 = 1110\ 10
\end{aligned}
$$

(c) Find the Golomb code of n=21 when m=5.

**Ans**

$$
\begin{aligned}
&2^{\lceil \log_2^m \rceil} - m = 2^3 - 5 = 3 \\
&\textit{encoded } 21 = 21/5 = 2 \ldots 1 = 110\ 01
\end{aligned}
$$

(d) Find the Golomb code of n=14 when m=5.

**Ans**

$$
\begin{aligned}
&2^{\lceil \log_2^m \rceil} - m = 2^3 - 5 = 3 \\
&\textit{encoded } 14 = 14/5 = 2 \ldots 4 = 110\ 111
\end{aligned}
$$

(e) A two-integer sequence is encoded by Golomb code with m=4 to get the bitstream 11101111000. What's the decoded two-integer sequence?

**Ans**

$$2^{\lceil \log_2^m \rceil} - m = 2^2 - 4 = 0$$

$$
\begin{array}{cccc}
\underline{1110} & \underline{11} & \underline{110} & \underline{00} \\
3 & 3 & 2 & 0 \\
\multicolumn{2}{c}{15} & \multicolumn{2}{c}{8}
\end{array}
$$

sequence: 15, 8

(f) A two-integer sequence is encoded by Golomb code with m=5 to get the bitstream 11101111000 (the same bitstream as that in (e)). What's the decoded two-integer sequence?
**Hint**: The unary code for a positive integer q is simply q 1s followed by a 0.

**Ans**

$$2^{\lceil \log_2^m \rceil} - m = 2^3 - 5 = 3$$

$$\begin{array}{cccc} \underline{1110} & \underline{111} & \underline{10} & \underline{00} \\ 3 & \text{7-3} = 4 & 2 & 0 \\ \multicolumn{2}{c}{18} & \multicolumn{2}{c}{10} \end{array}$$

sequence: 19, 10

## Source code for Problem 3 & 4

*huffman_code.hpp*

```cpp
#include <iostream>
#include <fstream>
#include <algorithm>     // std::find
#include <vector>        // std::vector
#include <iomanip>
#include <sstream>
#include <string>
#include <numeric>
#include <cstring>

class Node
{
  public:
  std::string letter;
  char symbol;
  int cnt;
  Node *child_r;
  Node *child_l;
  Node(std::string letter, char symbol, int cnt);
  Node(int cnt, Node *child_l, Node *child_r);
  bool isLeaf();
  bool operator<(const  Node & other)
  {
      return cnt < other.cnt;
  }
  bool operator>=(const  Node & other)
  {
      return cnt >= other.cnt;
  }
  bool operator==(const  Node & other)
  {
      return cnt == other.cnt;
  }
  int operator+(const  Node & other)
  {
      return cnt + other.cnt;
  }
};

class HuffmanCode {
```

```cpp
  public:
  std::vector<Node> nodes;
  Node *root;
  std::vector< std::pair <char, std::string> > code_list;
  double total_letters;
  HuffmanCode(std::string file, std::string csv_file);
  void getPmfCdf(std::vector<Node> node_list,
                 std::vector<double> &pmf,
                 std::vector<double> &cdf);
  double getEntropy(std::vector<double> &pmf);
  void printTable(std::vector<Node> node_list,
                  std::vector<double> &pmf,
                  std::vector<double> &cdf);
  void writeToCsv(std::string file_path,
                  std::vector<Node> node_list,
                  std::vector<double> &pmf,
                  std::vector<double> &cdf);
  std::string getSymbol(char c);
  void initNodes(std::vector<char> alphabas, std::vector<int> counts);
  void buildTree();
  void mergeNodesToList(std::vector<Node> &list, int start_indx, int end_indx);
  void printTree(Node *root, int spaces);
  void encodeData(Node *root, std::string code);
  bool isLeaf(Node *root);
  void printCodeWord();
};
```

*huffman_code.cpp*

```cpp
#include "huffman_code.hpp"

Node::Node(std::string letter, char symbol, int cnt)
{
  this->symbol = symbol;
  this->letter = letter;
  this->cnt = cnt;
  this->child_l = NULL;
  this->child_r = NULL;
}

Node::Node(int cnt, Node *child_l, Node *child_r)
{
  this->cnt = cnt;
  // this->parent = NULL;
  this->child_l = child_l;
  this->child_r = child_r;
}

bool Node::isLeaf()
{
  if (this->child_l == NULL && this->child_r == NULL)
  {
    return true;
  }
  else
  {
    return false;
  }
}

HuffmanCode::HuffmanCode( std::string file, std::string csv_file)
```

```cpp
{
  std::vector<char> alphabas;
  std::vector<int> counts;
  std::ifstream infile;
  char letter[0];
  infile.open (file, std::ios::app);
  while ( infile.peek() != EOF ) {
    infile.read (letter, 1);
    // counting letter
    std::vector<char>::iterator it;
    it = std::find (alphabas.begin(), alphabas.end(), *letter);
    if (it != alphabas.end())
    {
      size_t index = it - alphabas.begin();
      counts[index] += 1;
    }
    else
    {
      if(letter[0] != 0x0a)
      {
        alphabas.push_back(letter[0]);
        counts.push_back(1);
      }
    }
  }
  infile.close();
  this->initNodes(alphabas, counts);
  // Get reverse list
  std::vector<Node> nodes_r;
  for (int i = this->nodes.size()-1; i >= 0; i--)
  {
    nodes_r.push_back(this->nodes[i]);
  }
  std::vector <double> pmf, cdf;
  this->getPmfCdf(nodes_r, pmf, cdf);
  this->printTable(nodes_r, pmf, cdf);
  std::cout << "\nEntropy: " << this->getEntropy(pmf) << " bits/symbol" << std::endl;
  this->writeToCsv(csv_file, nodes_r, pmf, cdf);
  this->buildTree();
}

void HuffmanCode::getPmfCdf(std::vector<Node> node_list,
                            std::vector<double> &pmf,
                            std::vector<double> &cdf)
{
  for (int i = 0; i < node_list.size(); i++)
  {
    pmf.push_back(node_list[i].cnt / this->total_letters);
    if (i == 0)
    {
      cdf.push_back(pmf[0]);
    }
    else
    {
      cdf.push_back(cdf[i-1]+pmf[i]);
    }
  }
}

```

```
 double HuffmanCode::getEntropy(std::vector<double> &pmf)
 {
  double entropy = 0;
  for (int i = 0; i < pmf.size(); i++)
  {
    entropy += pmf[i] * log2(pmf[i]);
  }
  return -entropy;
 }

void HuffmanCode::printTable(std::vector<Node> node_list,
                             std::vector<double> &pmf,
                             std::vector<double> &cdf)
{
  std::cout << std::setw(15) << "Alphaba";
  for (int i = 0; i < node_list.size() / 2; i++)
  {
    std::cout << std::setw(10) << node_list[i].letter;
  }
  std::cout << std::endl;
  std::cout << std::setw(15) << "Total Number";
  for (int i = 0; i < node_list.size() / 2; i++)
  {
    std::cout << std::setw(10) << node_list[i].cnt;
  }
  std::cout << std::endl;
  std::cout << std::setw(15) << "PMF";
  for (int i = 0; i < node_list.size() / 2; i++)
  {
    std::cout << std::setw(10) << std::setprecision (3)<< pmf[i];
  }
  std::cout << std::endl;
  std::cout << std::setw(15) << "CDF";
  for (int i = 0; i < node_list.size() / 2; i++)
  {
    std::cout << std::setw(10) << std::setprecision (3)<< cdf[i];
  }
  std::cout << std::endl << std::endl;;
  std::cout << std::setw(15) << "Alphaba";
  for (int i = node_list.size() / 2; i < node_list.size(); i++)
  {
    std::cout << std::setw(10) << node_list[i].letter;
  }
  std::cout << std::endl;
  std::cout << std::setw(15) << "Total Number";
  for (int i = node_list.size() / 2; i < node_list.size(); i++)
  {
    std::cout << std::setw(10) << node_list[i].cnt;
  }
  std::cout << std::endl;
  std::cout << std::setw(15) << "PMF";
  for (int i = node_list.size() / 2; i < node_list.size(); i++)
  {
    std::cout << std::setw(10) << std::setprecision (3)<< pmf[i];
  }
  std::cout << std::endl;
  std::cout << std::setw(15) << "CDF";
  for (int i = node_list.size() / 2; i < node_list.size(); i++)
  {
    std::cout << std::setw(10) << std::setprecision (3)<< cdf[i];
```

```cpp
  }
  std::cout << std::endl;
}

void HuffmanCode::writeToCsv( std::string file_path,
                              std::vector<Node> node_list,
                              std::vector<double> &pmf,
                              std::vector<double> &cdf)
{
  std::ofstream myfile(file_path);
  myfile << "letter,pmf,cdf" << std::endl;
  for (int i = 0; i < node_list.size(); i++)
  {
    myfile << node_list[i].letter << "," << pmf[i] << "," <<  cdf[i] << std::endl;
  }
  myfile.close();
}

std::string HuffmanCode::getSymbol(char c)
{
  switch(c)
  {
    case 0x0d:
      return "CR";
      break;
    case 0x20:
      return "SP";
      break;
    default:
      std::string s;
      s += c;
      return s;
      break;
  }
}

void HuffmanCode::initNodes(std::vector<char> alphabas, std::vector<int> counts)
{
  for (int i = 0; i < alphabas.size(); i++)
  {
    Node n(this->getSymbol(alphabas[i]), alphabas[i],counts[i]);
    this->nodes.push_back(n);
  }
  std::sort(this->nodes.begin(), this->nodes.end());
  this->total_letters = 0;
  for (auto& n : this->nodes)
    this->total_letters += n.cnt;
}

void HuffmanCode::buildTree()
{
  int start_index = 0;
  while(this->nodes.size()-1 > start_index)
  {
    int node_merge_cnt = 1;
    Node &node = this->nodes[start_index];
    for (int i = start_index+1; i < this->nodes.size(); i++)
    {
      Node node_next =  this->nodes[i];
```

```
      if (node == node_next)
      {
        node_merge_cnt++;
      }
      else
      {
        break;
      }
    }
    if (node_merge_cnt == 1)
    {
      node_merge_cnt = 2;
    }
    else
    {
      node_merge_cnt = (node_merge_cnt / 2) * 2;
    }
    // Merge Node then put into node list
    this->mergeNodesToList(this->nodes, start_index, start_index+node_merge_cnt);
    start_index += node_merge_cnt;
  }
  this->root = &this->nodes.back();
}

void HuffmanCode::mergeNodesToList(std::vector<Node> &list,
                                   int start_indx, int end_indx)
{
  for (int i = start_indx; i < end_indx; i = i + 2)
  {
    Node *nodes_ptr = this->nodes.data();
    Node *node_i = nodes_ptr+i;
    Node *node_i_1 = nodes_ptr+i+1;
    Node n(list[i]+list[i+1], node_i, node_i_1);
    if (list.size() > 1)
    {
      bool inserted = false;
      for (int j = 0; j < list.size(); j++)
      {
        if (list[j] >= n)
        {
          list.insert(list.begin()+j, n);
          inserted = true;
          break;
        }
      }
      if (!inserted)
      {
        this->nodes.push_back(n);
      }
    }
    else
    {
      this->nodes.push_back(n);
    }
  }
}

void HuffmanCode::printTree(Node *root, int spaces)
{
  if(root != NULL)
```

```cpp
  {
    this->printTree(root->child_r, spaces + 5);
    for(int i = 0; i < spaces; i++)
      std::cout << ' ';
    std::cout << "    " << root->cnt << std::endl;
    this->printTree(root->child_l, spaces + 5);
  }
  else
  {
    return;
  }
}

void HuffmanCode::encodeData(Node *root, std::string code)
{
  if(root != NULL)
  {
    this->encodeData(root->child_r, code+"1");
    this->encodeData(root->child_l, code+"0");
    if (root->isLeaf())
    {
      std::pair <char, std::string> codeword(root->symbol, code);
      this->code_list.push_back(codeword);
    }
  }
  else
  {
    return;
  }
}

void HuffmanCode::printCodeWord()
{
  int cols = 2;
  for (int i = 0; i < code_list.size(); i=i+cols)
  {
    for (int j = i; j < i + cols; j++)
    {
      if (j <  code_list.size())
      {
        std::cout << std::setw(10) << this->getSymbol(this->code_list[j].first) << ": "
                  << std::setw(10) << this->code_list[j].second;
      }
      else
      {
        break;
      }
    }
    std::cout << std::endl;
  }
}

int main(int argc, char const *argv[])
{
  HuffmanCode huffman_code("../santaclaus.txt", "../statistic_result.csv");
  // HuffmanCode huffman_code("../test.txt",  "../statistic_result.csv");
  // HuffmanCode huffman_code("./hw#1_entropy_huffman_golomb/test.txt",  "../hw#1_entropy_huffman_golomb/statistic_result.csv");
  std::cout << std::endl << "==================== Huffman Tree ===================="
```

```
    << std::endl;
  // HuffmanCode huffman_code("./hw#1_entropy_huffman_golomb/test.txt");
  huffman_code.printTree(huffman_code.root, 1);
  std::cout << std::endl << "==================== Codeword Table ===================="
    << std::endl << std::endl;
  huffman_code.encodeData(huffman_code.root, "");
  huffman_code.printCodeWord();
  std::cout << std::endl;
  return 0;
}
```