哈尔滨工业大学计算学部

实验报告

课程名称：数据结构与算法

课程类型：专业核心基础课（必修）

实验项目：树形结构及应用

实验题目：哈夫曼编码与译码方法

实验日期：2023/10/11

班级：2203101

学号：2022113573

姓名：张宇杰

设计成绩	报告成绩	指导老师
		张岩

一、实验目的

哈夫曼编码是一种以哈夫曼树（最优二叉树，带权路径长度最小的二叉树）为基础变长编码方法。其基本思想是：将使用次数多的字符转换成长度较短的编码，而使用次数少的采用较长的编码，并且保持编码的唯一可解性。在计算机信息处理中，经常应用于数据压缩。是一种一致性编码法（又称"熵编码法"），用于数据的无损压缩。本实验要求实现一个完整的哈夫曼编码与译码系统。

二、实验要求及实验环境

实验要求：

1. 从文件中读入任意一篇英文文本文件，分别统计英文文本文件中各字符（包括标点符号和空格）的使用频率；
2. 根据已统计的字符使用频率构造哈夫曼编码树，并给出每个字符的哈夫曼编码（字符集的哈夫曼编码表）；
3. 将文本文件利用哈夫曼树进行编码，存储成压缩文件（哈夫曼编码文件）；
4. 将哈夫曼编码文件译码为文本文件，并与原文件进行比较。
5. 计算你的哈夫曼编码文件的平均编码长度和压缩率，并与实验结果比较验证；
6. 选做：上述 1-5 的编码和译码是基于字符的压缩，考虑基于单词的压缩，完成上述工作，讨论并比较压缩效果。
7. 选做：上述 1-5 的编码是二进制的编码，可以采用 K 叉的哈夫曼树完成上述

工作，实现“K 进制”的编码和译码，并与二进制的编码和译码进行比较。

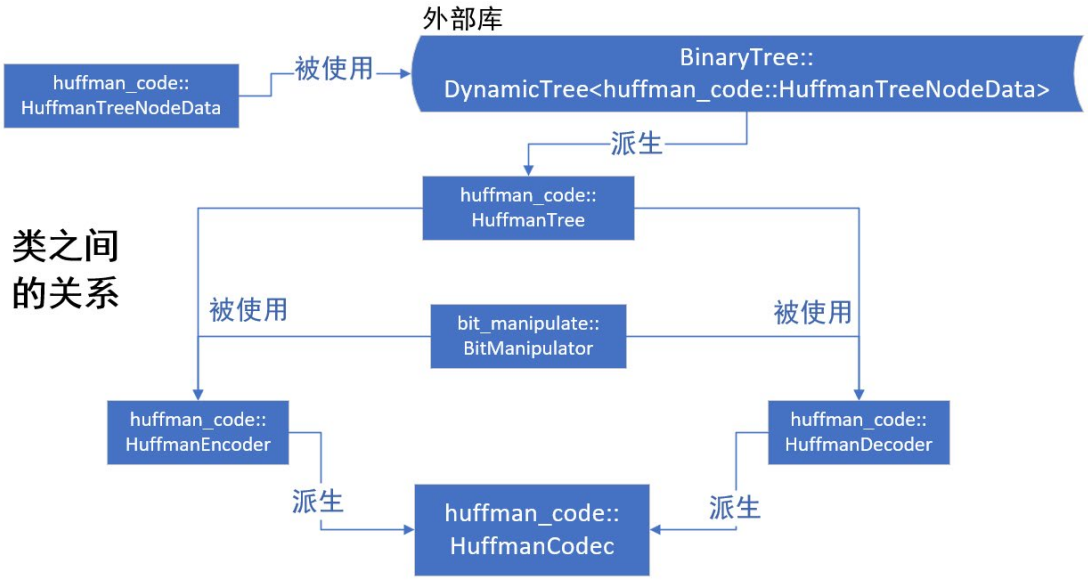
8. 选做：利用堆结构，优化哈夫曼编码算法。

实验环境：

Windows 11, g++, gdb

三、设计思想（本程序中的用到的所有数据类型的定义，主程序的流程图及各程序模块之间的调用关系、核心算法的主要步骤）

1. 逻辑设计



命名空间：bit_manipulate

```
namespace bit_manipulate
```

自定义类 bit_manipulate::BitManipulator

```
class BitManipulator
{
public:
    void write_bits(std::ostream& stream_dest, const std::string&
       bits_src);

    std::string read_bits(std::istream& stream_src);

    char byte_to_char(const std::string& bits_src, std::string::c
      onst_iterator& src_iter, int zero_supplement_len = 0);

    std::string char_to_byte(char char_src, int zero_supplement_l
      en = 0);
};
```

命名空间：huffman_code

```
namespace huffman_code
```

自定义类 huffman_code::HuffmanTreeNodeData

```
class HuffmanTreeNodeData
{
public:
    char ch;
    unsigned int weight;
```

```
6.
7.          HuffmanTreeNodeData(): ch('\0'), weight(0){};
8.
9.          HuffmanTreeNodeData(char _ch, unsigned int _weight): ch(_ch),
               weight(_weight){};
10.
11.         friend std::ostream& operator<<(std::ostream& out, const huff
              man_code::HuffmanTreeNodeData& data);
12.
13.         friend std::istream& operator>>(std::istream& in, huffman_cod
              e::HuffmanTreeNodeData& data);
14.     };
```

以下均使用别名

```
1.      // an alias
2.      using Node = BinaryTree::DynamicNode<HuffmanTreeNodeData>;
```

自定义类 huffman_code:: HuffmanTree

```
1.      class HuffmanTree: public BinaryTree::DynamicBinaryTree<HuffmanTr
              eeNodeData>
2.      {
3.      public:
4.          void make_huffman_tree(std::map<char, int>& freq_map);
5.
6.          class greater
7.          {
8.          public:
9.              inline bool operator()(Node* p1, Node* p2)
10.             {
11.                 return p1->data.weight > p2->data.weight;
12.             }
13.         };
14.     };
```

自定义类 huffman_code:: HuffmanEncoder

```
1.      class HuffmanEncoder
2.      {
3.      private:
4.          HuffmanTree tree;
5.
6.          // a map from char to the char's frequency
7.          std::map<char, int> freq_map;
8.
9.          // a map from char to the char's huffman-code
10.         std::map<char, std::string> code_map;
```

```
11.
12.         // the vector relative frequency
13.         std::vector<double> r_freq_vec;
14.
15.         // create code map by recursion
16.         void _rec_make_code_map(Node* p_node, std::string code);
17.
18.         // calculate the relative frequency
19.         void _calc_freq();
20.
21.     public:
22.         // calculate the frequency
23.         void get_freq(std::istream& stream_src);
24.
25.         // make code map by huffman tree
26.         void make_code_map();
27.
28.         // encode the source stream to binary string
29.         void encode_to_str(std::istream& stream_src, std::string& bit
              s_dest);
30.
31.         // encode the source stream to binary string, and output to a
              stream destination
32.         void encode_to_stream(std::istream& stream_src, std::ostream&
              code_dest);
33.
34.         void show_freq_map(std::ostream& out);
35.
36.         void show_code_map(std::ostream& out);
37.
38.         double avg_code_len();
39.     };
```

自定义类 huffman_code:: HuffmanDecoder

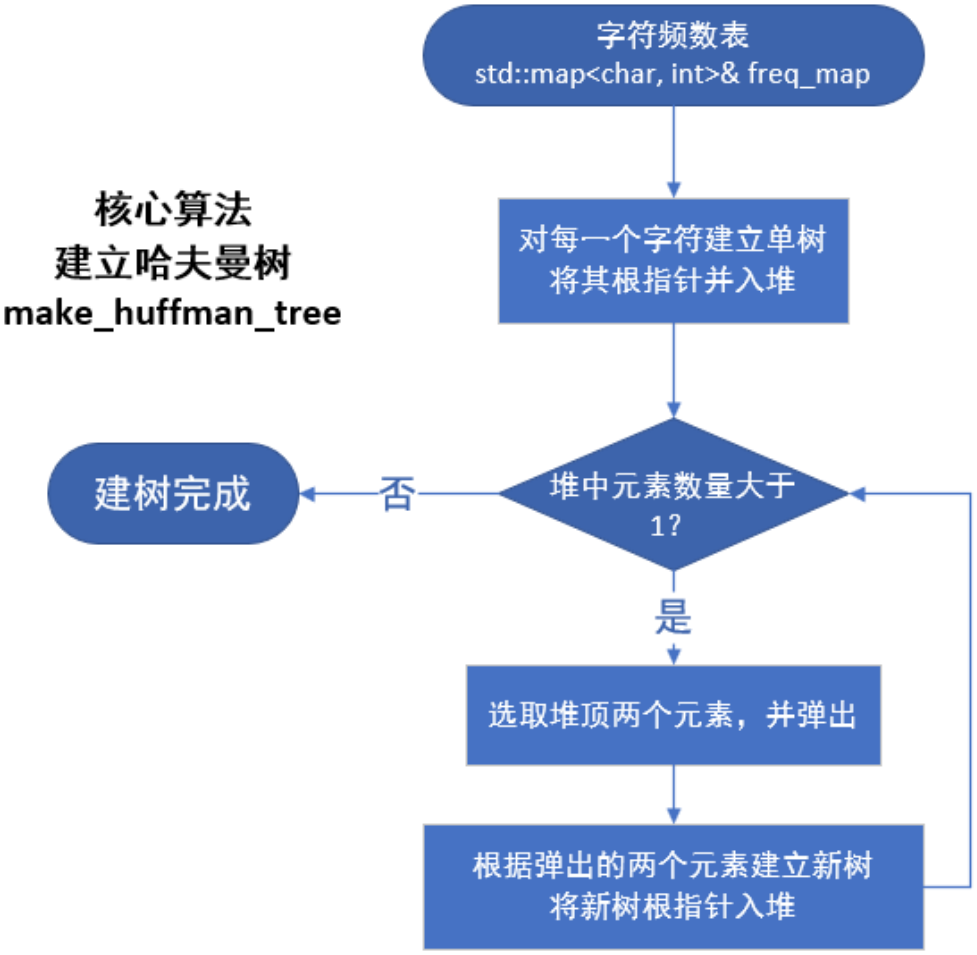
```
1.     class HuffmanDecoder
2.     {
3.     private:
4.         HuffmanTree tree;
5.
6.         void create_tree(std::istream& in, char placeholder);
7.
8.         void _create(std::istream& in, Node*& p, char placeholder);
9.
10.    public:
```

```
        void decode_to_stream(std::istream& huffman_stream_src, std::
          ostream& char_stream_dest);

        inline HuffmanTree& get_tree()
        {
            return tree;
        }
    };
```

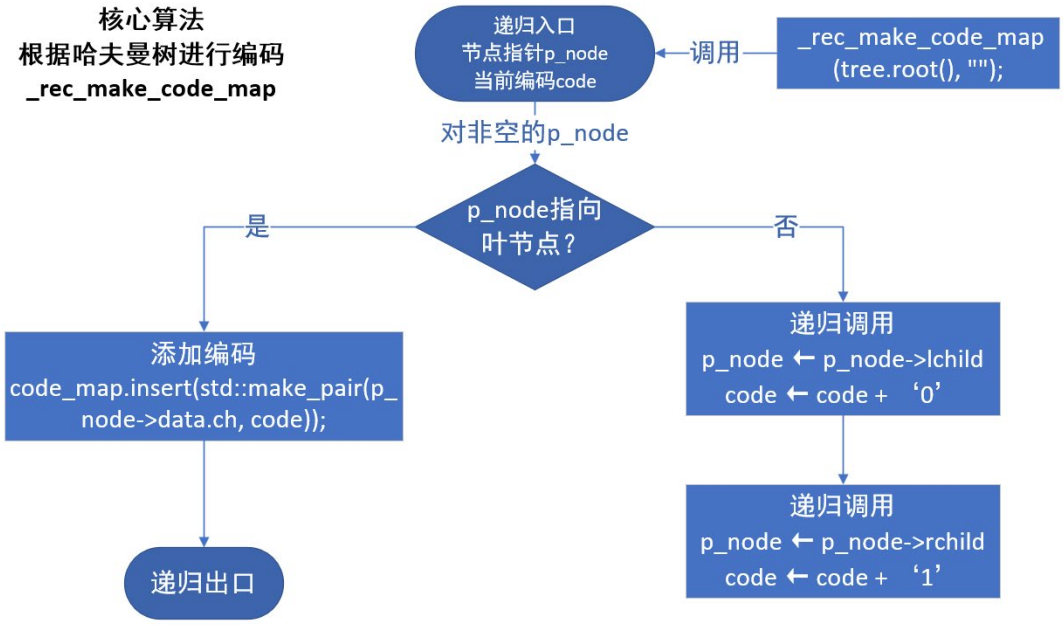
自定义类 huffman_code:: HuffmanCodec

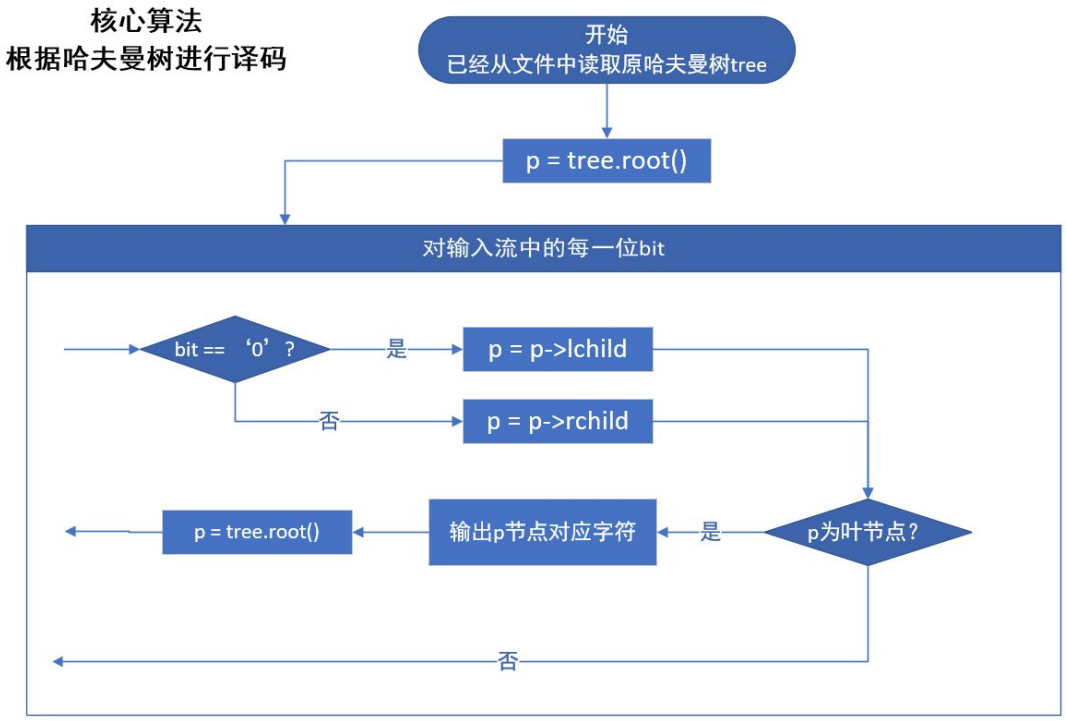
```
    class HuffmanCodec: private HuffmanEncoder, private HuffmanDecode
          r
    {
    public:
        void encode(std::string input_filename, std::string output_hu
          ffman_filename = "");

        void decode(std::string input_huffman_filename, std::string o
          utput_filename = "");

        void show_freq_map(std::ostream& out)
        {
            HuffmanEncoder::show_freq_map(out);
        }

        void show_code_map(std::ostream& out)
        {
            HuffmanEncoder::show_code_map(out);
        }

        double avg_code_len()
        {
            return HuffmanEncoder::avg_code_len();
        }
    };
```

核心算法
建立哈夫曼树
make_huffman_tree



核心算法
根据哈夫曼树进行编码
_rec_make_code_map





2. 物理设计（即存储结构设计）

使用 STL 容器 vector, map, priority_queue

使用自定义类 BinaryTree::DynamicTree

四、测试结果（包括测试数据、结果数据及结果的简单分析和结论，可以用截图得形式贴入此报告）

一、对字符进行统计

运行程序：

```
#include <iostream>
#include "./header/HuffmanCodec.hpp"

using namespace std;
using namespace huffman_code;

int main(int argc, const char *argv[])
{
    HuffmanCodec codec;

    codec.encode("./texts/Youth.txt");

    codec.show_freq_map(cout);

```

```
15.         return 0;
16.     }
```

输出结果：

```
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src> ./test
char | frequency  | relative freq
-128 | 4          | 0.3147%
-103 | 4          | 0.3147%
-30  | 4          | 0.3147%
\n   | 10         | 0.7868%
\r   | 10         | 0.7868%
sp   | 237        | 18.6467%
,    | 18         | 1.4162%
-    | 1          | 0.0787%
.    | 10         | 0.7868%
0    | 5          | 0.3934%
1    | 1          | 0.0787%
2    | 2          | 0.1574%
6    | 3          | 0.2360%
8    | 1          | 0.0787%
;    | 6          | 0.4721%
I    | 1          | 0.0787%
N    | 1          | 0.0787%
T    | 1          | 0.0787%
W    | 4          | 0.3147%
Y    | 4          | 0.3147%
a    | 72         | 5.6648%
b    | 11         | 0.8655%
c    | 13         | 1.0228%
d    | 26         | 2.0456%
e    | 121        | 9.5201%
f    | 34         | 2.6751%
g    | 20         | 1.5736%
h    | 43         | 3.3832%
i    | 70         | 5.5075%
j    | 1          | 0.0787%
k    | 6          | 0.4721%
l    | 27         | 2.1243%
m    | 30         | 2.3603%
n    | 59         | 4.6420%
o    | 85         | 6.6876%
p    | 20         | 1.5736%
q    | 1          | 0.0787%
r    | 63         | 4.9567%
s    | 70         | 5.5075%
t    | 84         | 6.6090%
u    | 32         | 2.5177%
v    | 13         | 1.0228%
w    | 15         | 1.1802%
x    | 2          | 0.1574%
y    | 26         | 2.0456%
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src>
```

结论：输出无误

二、根据已统计的字符使用频率构造哈夫曼编码树，并给出每个字符的哈夫曼编码

运行程序：

```
#include <iostream>
#include "./header/HuffmanCodec.hpp"

using namespace std;
using namespace huffman_code;

int main(int argc, const char *argv[])
{
    HuffmanCodec codec;

    codec.encode("./texts/Youth.txt");

    codec.show_code_map(cout);

    return 0;
}
```

输出结果：

```
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src> ./test
char | code
-128 | 110111010
-103 | 10010000
-30  | 110111001
\n   | 1001001
\r   | 1001100
space| 111
,    | 010111
-    | 0101101000
.    | 1001101
0    | 10010001
1    | 0101101011
2    | 010110000
6    | 110111000
8    | 0101101001
;    | 11001011
I    | 0101101100
N    | 0101101111
T    | 0101101101
W    | 01011001
Y    | 110111011
a    | 1000
b    | 1100100
c    | 1101000
d    | 110011
e    | 000
f    | 01010
g    | 100101
h    | 11000
i    | 0111
j    | 0101101010
k    | 11001010
l    | 110110
m    | 00110
n    | 0010
o    | 1011
p    | 100111
q    | 0101101110
r    | 0100
s    | 0110
t    | 1010
u    | 00111
v    | 1101001
w    | 1101111
x    | 010110001
y    | 110101
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src>
```

结论：输出无误

三、将文本文件利用哈夫曼树进行编码，存储成压缩文件（哈夫曼编码文件）

运行程序：

```cpp
#include <iostream>
#include "./header/HuffmanCodec.hpp"

using namespace std;
using namespace huffman_code;

int main(int argc, const char *argv[])
{
    HuffmanCodec codec;

    codec.encode("./texts/Youth.txt");

    return 0;
}
```

输出结果：

```
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src> .\compile.bat

D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src>g++ main.cpp ./header/HuffmanTree.cpp ./header/HuffmanEncoder.cpp ./header/HuffmanDecoder.cpp ./header/BitManipulator.cpp ./header/HuffmanCodec.cpp -o test.exe
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src> cd ./texts
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src\texts> ls


    目录: D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src\texts


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----    2023/9/26 周二     22:24        6458580 Harry Potter.txt
-a----    2023/9/26 周二     22:05          97036 Little Prince.txt
-a----    2023/9/26 周二     18:10           1271 Youth.txt


(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src\texts> cd ..
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src> ./test
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src> cd ./texts
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src\texts> ls


    目录: D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src\texts


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----    2023/9/26 周二     22:24        6458580 Harry Potter.txt
-a----    2023/9/26 周二     22:05          97036 Little Prince.txt
-a----    2023/9/26 周二     18:10           1271 Youth.txt
-a----    2023/10/2 周一     21:43            884 Youth.txt.huffman


(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src\texts>
```

成功输出 Youth.txt.huffman 文件，大小为 884 字节

原文件 Youth.txt 大小为 1271 字节

四、将哈夫曼编码文件译码为文本文件，并与原文件进行比较

运行程序：

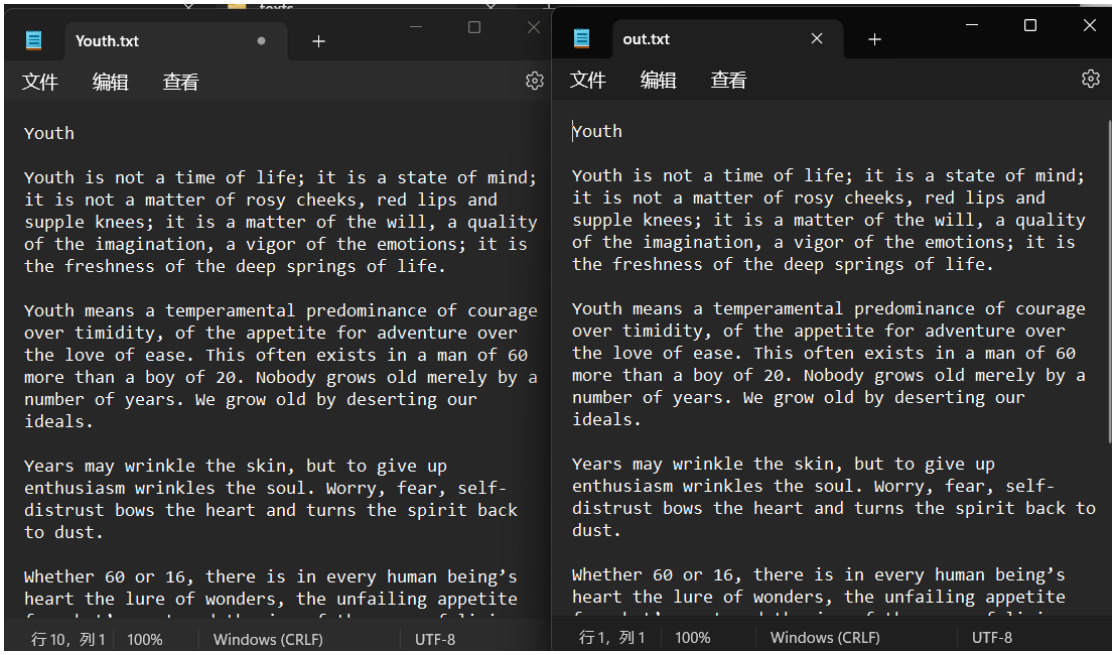
```cpp
#include <iostream>
#include "./header/HuffmanCodec.hpp"
```

```
3.
4.      using namespace std;
5.      using namespace huffman_code;
6.
7.      int main(int argc, const char *argv[])
8.      {
9.          HuffmanCodec codec;
10.
11.         codec.decode("./texts/Youth.txt.huffman", "./texts/out.txt");
12.
13.         return 0;
14.     }
```

输出结果：

```
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src> .\compile.bat

D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src>g++ main.cpp ./header/HuffmanTree.cpp ./header/HuffmanEncoder.cpp .
/header/HuffmanDecoder.cpp ./header/BitManipulator.cpp ./header/HuffmanCodec.cpp -o test.exe
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src> cd ./texts
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src\texts> ls


    目录: D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src\texts


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----     2023/9/26 周二     22:24        6458580 Harry Potter.txt
-a----     2023/9/26 周二     22:05          97036 Little Prince.txt
-a----     2023/9/26 周二     18:10           1271 Youth.txt
-a----     2023/10/2 周一     21:43            884 Youth.txt.huffman


(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src\texts> cd ..
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src> ./test
704
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src> cd ./texts
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src\texts> ls


    目录: D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src\texts


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----     2023/9/26 周二     22:24        6458580 Harry Potter.txt
-a----     2023/9/26 周二     22:05          97036 Little Prince.txt
-a----     2023/10/2 周一     21:47           1271 out.txt
-a----     2023/9/26 周二     18:10           1271 Youth.txt
-a----     2023/10/2 周一     21:43            884 Youth.txt.huffman


(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src\texts>
```

成功生成输出文件 out.txt



其内容与原文件一致

五、计算你的哈夫曼编码文件的平均编码长度和压缩率，并与实验结果比较验证

运行程序：

```
#include <iostream>
#include "./header/HuffmanCodec.hpp"

using namespace std;
using namespace huffman_code;

int main(int argc, const char *argv[])
{
    HuffmanCodec codec;

    codec.encode(argv[1]);

    cout << codec.avg_code_len() << endl;

    return 0;
}
```

输出结果：

```
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src> .\compile.bat

D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src>g++ main.cpp ./header/HuffmanTree.cpp ./header/HuffmanEncoder.cpp .
/header/HuffmanDecoder.cpp ./header/BitManipulator.cpp ./header/HuffmanCodec.cpp -o test.exe
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src> ./test .\texts\Youth.txt
4.42801
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src> ./test '.\texts\Little Prince.txt'
4.57977
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src> ./test '.\texts\Harry Potter.txt'
4.66008
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src> cd ./texts
(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src\texts> ls


    目录: D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src\texts


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----     2023/9/26 周二     22:24        6458580 Harry Potter.txt
-a----     2023/10/2 周一     21:56        3762606 Harry Potter.txt.huffman
-a----     2023/9/26 周二     22:05          97036 Little Prince.txt
-a----     2023/10/2 周一     21:56          55871 Little Prince.txt.huffman
-a----     2023/9/26 周二     18:10           1271 Youth.txt
-a----     2023/10/2 周一     21:56            884 Youth.txt.huffman


(base) PS D:\File\大二秋\DSA\实验2 哈夫曼编码\huffman-code\src\texts>
```

结果汇总：

文本	平均编码长度	平均编码长度/8 (%)	原文件大小(byte)	压缩文件大小(byte)	压缩比(%)
Youth.txt	4.42801	55.35%	1271	884	69.55%
Little Prince.txt	4.57977	57.25%	97036	55871	57.58%
Harry Potter.txt	4.66008	58.25%	6458580	3762606	58.26%

随着文件大小的上升，“压缩比”逐渐趋同于“平均编码长度比 8 值”

这是因为，即使文件大小有了显著的增加，存储哈夫曼树所需的空间也几乎不会有太大变化

五、经验体会与不足

经验体会：

深刻地理解了哈夫曼编码的原理，成功实现了压缩文件的功能

不足：

代码结构有待改善

六、附录：源代码（带注释）

BitManipulator.hpp

```cpp
#ifndef _BIT_MANIPULATOR_HPP_
#define _BIT_MANIPULATOR_HPP_

#include <iostream>

namespace bit_manipulate
{
    class BitManipulator
    {
    public:
```

```
11.           /**
12.            * @brief write a binary string to the stream
13.            * supplement some zeros in front of the binary string
14.            * if the string len is not divisible by 8
15.            * write the length of the supplement zero before writing
           the binary string
16.            * @param stream_dest the stream destination
17.            * @param bits_src the binary string source
18.            */
19.           void write_bits(std::ostream& stream_dest, const std::str
          ing& bits_src);
20.
21.           /**
22.            * @brief read binary string from the stream
23.            * the stream source must be written by "write_bits"
24.            * @param stream_src the stream source
25.            * @return the binary string
26.            */
27.           std::string read_bits(std::istream& stream_src);
28.
29.           /**
30.            * @brief convert a binary string byte to a char
31.            * i.e. a binary string with the length of 8
32.            */
33.           char byte_to_char(const std::string& bits_src, std::strin
          g::const_iterator& src_iter, int zero_supplement_len = 0);
34.
35.           /**
36.            * @brief convert a char to a binary string byte
37.            * i.e. a binary string with the length of 8
38.            */
39.           std::string char_to_byte(char char_src, int zero_suppleme
          nt_len = 0);
40.       };
41.   }
42.
43.   #endif
```

BitManipulator.cpp

```
1.    #include "./BitManipulator.hpp"
2.
3.    /**
4.     * @brief write a binary string to the stream
5.     * supplement some zeros in front of the binary string
```

```
     * if the string len is not divisible by 8
     * write the length of the supplement zero before writing the bin
         ary string
     * @param stream_dest the stream destination
     * @param bits_src the binary string source
    */
    void bit_manipulate::BitManipulator::write_bits(std::ostream& str
         eam_dest, const std::string& bits_src)
    {
        // calc and write the length of zeros to be supplemented
        int zero_supplement_len = 8 - bits_src.size() % 8;
        stream_dest << (char)zero_supplement_len;

        // write the binary string
        auto iter = bits_src.cbegin();
        while (iter != bits_src.cend())
        {
            stream_dest << byte_to_char(bits_src, iter, zero_suppleme
          nt_len);

            // if the zeros have been supplemented
            if (zero_supplement_len > 0)
                // no need to supplement zeros
                zero_supplement_len = 0;
        }
    }

    /**
     * @brief read binary string from the stream
     * the stream source must be written by "write_bits"
     * @param stream_src the stream source
     * @return the binary string
    */
    std::string bit_manipulate::BitManipulator::read_bits(std::istrea
         m& stream_src)
    {
        // buffer for return value
        std::string ret;

        // get the zeros supplemented
        char zero_supplement_len = stream_src.get();

        char ch;
        /// int cnt = 0;
```

```
46.         while (ch = stream_src.get(), !stream_src.eof())
47.         {
48.             // append binary strings to the return value
49.             // byte by byte
50.             ret += char_to_byte(ch, (int)zero_supplement_len);
51.
52.             if (zero_supplement_len > '\0')
53.                 zero_supplement_len = '\0';
54.
55.             /// cnt++;
56.         }
57.         /// std::cout << cnt << std::endl;
58.
59.         return ret;
60.         // return std::move(ret);
61.     }
62.
63.     /**
64.      * @brief convert a binary string byte to a char
65.      * i.e. a binary string with the length of 8
66.     */
67.     char bit_manipulate::BitManipulator::byte_to_char(const std::stri
           ng& bits_src, std::string::const_iterator& src_iter, int ze
           ro_supplement_len)
68.     {
69.         char ret = 0;
70.         for (int i = 0; i < 8 - zero_supplement_len; ++i, ++src_iter)
71.         {
72.             ret <<= 1;
73.             if (*src_iter == '1')
74.                 ret |= 1;
75.         }
76.         return ret;
77.     }
78.
79.     /**
80.      * @brief convert a char to a binary string byte
81.      * i.e. a binary string with the length of 8
82.     */
83.     std::string bit_manipulate::BitManipulator::char_to_byte(char cha
           r_src, int zero_supplement_len)
84.     {
85.         std::string ret;
```

```
        for (int i = (1 << (7 - zero_supplement_len)); i > 0; i >>= 1
          )
        {
            if ((i & char_src) != 0)
                ret += '1';
            else
                ret += '0';
        }
        return std::move(ret);
    }
```

HuffmanTree.hpp

```
    #ifndef _HUFFMAN_TREE_HPP_
    #define _HUFFMAN_TREE_HPP_

    #include "./binary-tree/DynamicBinaryTree.hpp"
    #include <iostream>
    #include <map>

    namespace huffman_code
    {
        class HuffmanTreeNodeData
        {
        public:
            char ch;
            unsigned int weight;

            HuffmanTreeNodeData(): ch('\0'), weight(0){};

            HuffmanTreeNodeData(char _ch, unsigned int _weight): ch(_
          ch), weight(_weight){};

            friend std::ostream& operator<<(std::ostream& out, const
          huffman_code::HuffmanTreeNodeData& data);

            friend std::istream& operator>>(std::istream& in, huffman
          _code::HuffmanTreeNodeData& data);
        };

        // an alias
        using Node = BinaryTree::DynamicNode<HuffmanTreeNodeData>;

        class HuffmanTree: public BinaryTree::DynamicBinaryTree<Huffm
          anTreeNodeData>
```

```
29.         {
30.         public:
31.             void make_huffman_tree(std::map<char, int>& freq_map);
32.
33.             // a compare functor, which compares the weight in Huffma
              nTreeNodeData
34.             class greater
35.             {
36.             public:
37.                 inline bool operator()(Node* p1, Node* p2)
38.                 {
39.                     return p1->data.weight > p2->data.weight;
40.                 }
41.             };
42.         };
43.     }
44.
45.     #endif
```

HuffmanTree.cpp

```
1.      #include "./HuffmanTree.hpp"
2.      #include <queue>
3.      #include "HuffmanTree.hpp"
4.
5.      std::ostream& operator<<(std::ostream& out, const huffman_code::H
            uffmanTreeNodeData& data)
6.      {
7.          // out.write(&(data.ch), 1);
8.          out << data.ch <<  ' ' << data.weight;
9.          return out;
10.     }
11.
12.     std::istream& operator>>(std::istream& in, huffman_code::HuffmanT
            reeNodeData& data)
13.     {
14.         data.ch = in.get();
15.         in >> data.weight;
16.         return in;
17.     }
18.
19.     void huffman_code::HuffmanTree::make_huffman_tree(std::map<char,
            int>& freq_map)
20.     {
21.         // a small root heap
```

```
22.         std::priority_queue<Node*, std::vector<Node*>, greater> heap;
23.
24.         // create tree for every single term in freq_map
25.         for (auto key_val: freq_map)
26.         {
27.             Node* p = new_node();
28.             p->data.ch = key_val.first;
29.             p->data.weight = key_val.second;
30.             heap.push(p);
31.         }
32.
33.         if (!empty())
34.             delete_tree(root());
35.
36.         // create huffman tree
37.         while (heap.size() > 1)
38.         {
39.             // get two smallest tree
40.             // "smallest" means the data.weight are smallest
41.             Node* node_1 = heap.top();
42.             heap.pop();
43.             Node* node_2 = heap.top();
44.             heap.pop();
45.
46.             // using the two to make a new tree
47.             Node* root = new_node();
48.             root->lchild = node_1;
49.             root->rchild = node_2;
50.             root->data = HuffmanTreeNodeData('\0', node_1->data.weigh
              t + node_2->data.weight);
51.
52.             // push the new tree into the heap
53.             heap.push(root);
54.         }
55.
56.         this->root() = heap.top();
57.     }
```

HuffmanEncoder.hpp

```
1.  #ifndef _HUFFMAN_ENCODER_HPP_
2.  #define _HUFFMAN_ENCODER_HPP_
3.
4.  #include "./HuffmanTree.hpp"
5.  #include <iostream>
```

```
6.    #include <map>
7.    #include <vector>
8.
9.    namespace huffman_code
10.   {
11.       class HuffmanEncoder
12.       {
13.       private:
14.           HuffmanTree tree;
15.
16.           // a map from char to the char's frequency
17.           std::map<char, int> freq_map;
18.
19.           // a map from char to the char's huffman-code
20.           std::map<char, std::string> code_map;
21.
22.           // the vector relative frequency
23.           std::vector<double> r_freq_vec;
24.
25.           // create code map by recursion
26.           void _rec_make_code_map(Node* p_node, std::string code);
27.
28.           // calculate the relative frequency
29.           void _calc_freq();
30.
31.       public:
32.           // calculate the frequency
33.           void get_freq(std::istream& stream_src);
34.
35.           // make code map by huffman tree
36.           void make_code_map();
37.
38.           // encode the source stream to binary string
39.           void encode_to_str(std::istream& stream_src, std::string&
            bits_dest);
40.
41.           // encode the source stream to binary string, and output
           to a stream destination
42.           void encode_to_stream(std::istream& stream_src, std::ostr
           eam& code_dest);
43.
44.           void show_freq_map(std::ostream& out);
45.
46.           void show_code_map(std::ostream& out);
```

```
47.
48.             double avg_code_len();
49.         };
50.     }
51.
52.     #endif
```

HuffmanEncoder.cpp

```
1.     #include "./HuffmanEncoder.hpp"
2.     #include "./BitManipulator.hpp"
3.     #include <iomanip>
4.
5.     void huffman_code::HuffmanEncoder::get_freq(std::istream& stream_
           src)
6.     {
7.         freq_map.clear();
8.
9.         char ch;
10.        while (ch = stream_src.get(), !stream_src.eof())
11.        {
12.            auto it = freq_map.find(ch);
13.            if (it != freq_map.end())
14.            {
15.                ++(it->second);
16.            }
17.            else
18.            {
19.                freq_map.insert(std::make_pair(ch, 1));
20.            }
21.        }
22.
23.        _calc_freq();
24.    }
25.
26.    void huffman_code::HuffmanEncoder::make_code_map()
27.    {
28.        tree.make_huffman_tree(freq_map);
29.
30.        code_map.clear();
31.
32.        _rec_make_code_map(tree.root(), "");
33.    }
34.
```

```
35.  void huffman_code::HuffmanEncoder::encode_to_str(std::istream& st
         ream_src, std::string& bits_dest)
36.  {
37.      char ch;
38.      while (ch = stream_src.get(), !stream_src.eof())
39.      {
40.          bits_dest += code_map[ch];
41.      }
42.  }
43.
44.  void huffman_code::HuffmanEncoder::encode_to_stream(std::istream&
        stream_src, std::ostream& code_dest)
45.  {
46.      // get the encoded binary string
47.      std::string bits;
48.      encode_to_str(stream_src, bits);
49.
50.      // save the tree structure
51.      tree.for_each(
52.          BinaryTree::PRE_ORDER,
53.          [&](HuffmanTreeNodeData e){code_dest << e.ch;},
54.          [&](){code_dest << (char)0b00001111;},
55.          tree.root()
56.      );
57.
58.      // write binary string to stream
59.      bit_manipulate::BitManipulator bit_manip;
60.      bit_manip.write_bits(code_dest, bits);
61.  }
62.
63.  void huffman_code::HuffmanEncoder::_rec_make_code_map(Node* p_nod
         e, std::string code)
64.  {
65.      if (p_node != nullptr)
66.      {
67.          if (p_node->lchild == nullptr && p_node->rchild == nullpt
        r)
68.          // if is a leaf node
69.          {
70.              // a recursion exit
71.              code_map.insert(std::make_pair(p_node->data.ch, code)
        );
72.          }
73.          else
```

```
74.             // is not a leaf node
75.             {
76.                 _rec_make_code_map(p_node->lchild, code + '0');
77.                 _rec_make_code_map(p_node->rchild, code + '1');
78.             }
79.         }
80.     }
81.
82.     void huffman_code::HuffmanEncoder::_calc_freq()
83.     {
84.         // calc total frequency
85.         size_t total = 0;
86.         for (auto& key_val: freq_map)
87.         {
88.             total += key_val.second;
89.         }
90.
91.         r_freq_vec.clear();
92.
93.         for (auto& key_val: freq_map)
94.         {
95.             r_freq_vec.push_back((double)(key_val.second) / total);
96.         }
97.     }
98.
99.     double huffman_code::HuffmanEncoder::avg_code_len()
100.    {
101.        double ret = 0;
102.        int i = 0;
103.        for (auto& key_val: code_map)
104.        {
105.            ret += r_freq_vec.at(i++) * key_val.second.size();
106.        }
107.
108.        return ret;
109.    }
110.
111.    void huffman_code::HuffmanEncoder::show_freq_map(std::ostream& ou
           t)
112.    {
113.        out << "char | frequency  | relative freq" << std::endl;
114.        int i = 0;
115.        for (auto key_val: freq_map)
116.        {
```

```
117.           if (key_val.first == '\r')
118.               out << "\\r   | ";
119.           else if (key_val.first == '\n')
120.               out << "\\n   | ";
121.           else if (key_val.first == '\t')
122.               out << "\\t   | ";
123.           else if (key_val.first == ' ')
124.               out << "space| ";
125.           else if (!std::isprint(key_val.first))
126.               out << std::setw(5) << std::left << (int)(key_val.fir
          st) << "| ";
127.           else
128.               out << key_val.first << "    | ";
129.
130.           out << std::setw(10) << std::left << key_val.second
131.               << " | "
132.               << std::fixed << std::setprecision(4)
133.               << r_freq_vec.at(i++) * 100
134.               << '%'
135.               << std::endl;
136.       }
137.   }
138.
139.   void huffman_code::HuffmanEncoder::show_code_map(std::ostream& ou
           t)
140.   {
141.       out << "char | code" << std::endl;
142.       for (auto key_val: code_map)
143.       {
144.           if (key_val.first == '\r')
145.               out << "\\r   | ";
146.           else if (key_val.first == '\n')
147.               out << "\\n   | ";
148.           else if (key_val.first == '\t')
149.               out << "\\t   | ";
150.           else if (key_val.first == ' ')
151.               out << "space| ";
152.           else if (!std::isprint(key_val.first))
153.               out << std::setw(5) << std::left << (int)(key_val.fir
          st) << "| ";
154.           else
155.               out << key_val.first << "    | ";
156.
157.           out << key_val.second << std::endl;
```

```
158.        }
159.    }
```

HuffmanDecoder.hpp

```
1.      #ifndef _HUFFMAN_DECODER_HPP_
2.      #define _HUFFMAN_DECODER_HPP_
3.
4.      #include "./HuffmanTree.hpp"
5.      #include <iostream>
6.
7.      namespace huffman_code
8.      {
9.          class HuffmanDecoder
10.         {
11.         private:
12.             HuffmanTree tree;
13.
14.             void create_tree(std::istream& in, char placeholder);
15.
16.             void _create(std::istream& in, Node*& p, char placeholder
                );
17.
18.         public:
19.             void decode_to_stream(std::istream& huffman_stream_src, s
                td::ostream& char_stream_dest);
20.
21.             inline HuffmanTree& get_tree()
22.             {
23.                 return tree;
24.             }
25.         };
26.     }
27.
28.     #endif
```

HuffmanDecoder.cpp

```
1.      #include "./HuffmanDecoder.hpp"
2.      #include "./BitManipulator.hpp"
3.
4.      void huffman_code::HuffmanDecoder::decode_to_stream(std::istream&
               stream_src, std::ostream& char_stream_dest)
5.      {
6.          create_tree(stream_src, (char)0b00001111);
7.
8.          // std::cout << stream_src.peek() << std::endl;
```

```
9.          // std::cout << stream_src.eof() << std::endl;
10.
11.         // tree.for_each(
12.         //     BinaryTree::PRE_ORDER,
13.         //     [&](HuffmanTreeNode e){std::cout << e.ch;},
14.         //     [&](){std::cout << (char)0b00001111;},
15.         //     tree.root()
16.         // );
17.
18.         bit_manipulate::BitManipulator bit_manip;
19.
20.         // std::string bits(std::move(bit_manip.read_bits(stream_src)
              ));
21.         std::string bits(bit_manip.read_bits(stream_src));
22.
23.         // std::cout << bits.size() << std::endl;
24.
25.         Node* p_node = tree.root();
26.
27.         for (char bit: bits)
28.         {
29.             if (bit == '0')
30.             {
31.                 p_node = p_node->lchild;
32.             }
33.             else
34.             // bit == '1'
35.             {
36.                 p_node = p_node->rchild;
37.             }
38.
39.             if (p_node->lchild == nullptr && p_node->rchild == nullpt
              r)
40.             // reach a leaf node
41.             {
42.                 char_stream_dest << p_node->data.ch;
43.                 p_node = tree.root();
44.             }
45.         }
46.     }
47.
48.     void huffman_code::HuffmanDecoder::create_tree(std::istream& in,
            char placeholder)
49.     {
```

```
50.         if (!tree.empty())
51.             tree.delete_tree(tree.root());
52.
53.         _create(in, tree.root(), placeholder);
54.     }
55.
56.     void huffman_code::HuffmanDecoder::_create(std::istream& in, Node
          *& p, char placeholder)
57.     {
58.         // prevent dead loop
59.         if (in.eof())
60.             return;
61.
62.         if (in.peek() != placeholder)
63.         {
64.             char data = in.get();
65.             p = tree.new_node();
66.
67.             p->data = HuffmanTreeNodeData(data, 0);
68.             _create(in, p->lchild, placeholder);
69.             _create(in, p->rchild, placeholder);
70.         }
71.         else
72.         {
73.             in.get();
74.             p = nullptr;
75.         }
76.     }
```

HuffmanCodec.hpp

```
1.     #ifndef _HUFFMAN_CODEC_HPP_
2.     #define _HUFFMAN_CODEC_HPP_
3.
4.     #include "HuffmanDecoder.hpp"
5.     #include "HuffmanEncoder.hpp"
6.
7.     #include <iostream>
8.
9.     namespace huffman_code
10.    {
11.        class HuffmanCodec: private HuffmanEncoder, private HuffmanDe
             coder
12.        {
13.        public:
```

```
14.
15.             /**
16.              * @brief encode file to huffman-code file
17.              * @param input_filename the file path to be encoded
18.              * @param output_huffman_filename the output huffman-code
             file path, default: input_filename + ".huffman"
19.              */
20.             void encode(std::string input_filename, std::string outpu
            t_huffman_filename = "");
21.
22.             /**
23.              * @brief decode huffman-code file to the original file
24.              * @param input_huffman_filename the huffman-code file pa
            th, must with the postfix ".huffman"
25.              * @param output_filename the output file path, default:
            input_huffman_filename - ".huffman"
26.              */
27.             void decode(std::string input_huffman_filename, std::stri
            ng output_filename = "");
28.
29.             inline void show_freq_map(std::ostream& out)
30.             {
31.                 HuffmanEncoder::show_freq_map(out);
32.             }
33.
34.             inline void show_code_map(std::ostream& out)
35.             {
36.                 HuffmanEncoder::show_code_map(out);
37.             }
38.
39.             inline double avg_code_len()
40.             {
41.                 return HuffmanEncoder::avg_code_len();
42.             }
43.         };
44.     }
45.
46.     #endif
```

HuffmanCodec.cpp

```
1.     #include "HuffmanCodec.hpp"
2.     #include <fstream>
3.
4.     /**
```

```
5.      * @brief encode file to huffman-code file
6.      * @param input_filename the file path to be encoded
7.      * @param output_huffman_filename the output huffman-code file pa
            th
8.     */
9.     void huffman_code::HuffmanCodec::encode(std::string input_filenam
           e, std::string output_huffman_filename)
10.    {
11.        if (output_huffman_filename.empty())
12.            output_huffman_filename = input_filename + ".huffman";
13.
14.        std::ifstream in(input_filename, std::ios::binary);
15.        std::ofstream out(output_huffman_filename, std::ios::binary);
16.
17.        get_freq(in);
18.        make_code_map();
19.
20.        in.clear();
21.        in.seekg(0, std::ios::beg);
22.
23.        encode_to_stream(in, out);
24.
25.        out.close();
26.        in.close();
27.    }
28.
29.    /**
30.     * @brief decode huffman-code file to the original file
31.     * @param input_huffman_filename the huffman-code file path, must
            with the postfix ".huffman"
32.     * @param output_filename the output file path, default: input_hu
            ffman_filename - ".huffman"
33.    */
34.    void huffman_code::HuffmanCodec::decode(std::string input_huffman
           _filename, std::string output_filename)
35.    {
36.        auto pos = input_huffman_filename.find(".huffman");
37.        if (pos == std::string::npos)
38.            throw std::range_error("file postfix is not .huffman");
39.
40.        if (output_filename == "")
41.        {
42.            output_filename = input_huffman_filename;
```

```
43.             output_filename.erase(output_filename.begin() + pos, outp
                ut_filename.end());
44.         }
45.
46.         std::ifstream in(input_huffman_filename, std::ios::binary);
47.         std::ofstream out(output_filename, std::ios::binary);
48.
49.         decode_to_stream(in, out);
50.
51.         out.close();
52.         in.close();
53.     }
```