

数据结构与算法实验报告-基于 huffman 编码的压缩算法实现

18324034 林天皓 linth5@mail2.sysu.edu.cn

摘要

本次数据结构与算法实验根据题目需求完成了一个基于 huffman 编码压缩算法的实现

引言

实验目的：设计一个基于 huffman 编码压缩算法的实现。通过这种方法，可以实现压缩任意文件，学习基本数据压缩的方法。

解决方法

在压缩过程中,使用一个 `class zip_huffman` 类来执行压缩过程。在用户使用的过程中,需要用户按照程序输出的说明,指定需要压缩的文件名和压缩后生成的压缩文件的文件名,同时本程序还会生成一个压缩文件名的 `key` 文件保存压缩信息。

第一部分：具体程序执行的压缩过程如下：

步骤 1：首先通过文件读取，将文件中的所有字符按照读入一个字符串中，在本程序中为 str_input。

```
//执行压缩过程
void zip_huffman::openfile(string inputfile_name, string &str_input)
{
    ifstream s_inputfile;
    s_inputfile.open(inputfile_name, ios::in | ios::binary);
    while (!s_inputfile.is_open())
    {
        return;
    }
    while (s_inputfile.peek() != EOF)
    {
        char t[1];
        s_inputfile.read(t, 1);
        str_input.push_back(t[0]);
    }
    //cout<<str_input;
    //10.21 19:53 str 无错误
}
```

步骤 2：统计字符串中每种字符的数量，存入 vector<int> huffman_count 中。

```
//统计字节出现的频数
void zip_huffman::count_huffman_node(vector<int> &huffman_count, const string &str_input)
{
    huffman_count.resize(256);
    for (auto &i : str_input)
    {
        unsigned char t=i;
        huffman_count[t]++;
    }
    for (int i = 0; i < 256; i++)
    {
        //cout << i << " : " << huffman_count[i] << endl;
    }
}
```

步骤 3: 将统计出来的数量结果通过自定义的优先队列储存, 按照 huffman 树的方式, 出现次数少的字符排在前面, 出现次数多的字符排在后面。

```
//建立 huffman 节点临时堆
void zip_huffman::createheap(const vector<int> &huffman_count, priority_queue<zip_huffman_node> &heap_huffman)
{
    for (int i = 0; i < 256; i++)
    {
        if (huffman_count[i] != 0)
        {
            zip_huffman_node t;
            t.c = i;
            t.count = huffman_count[i];
            t.left_child = t.right_child = t.parent = nullptr;
            heap_huffman.push(t);
        }
    }
}
```

步骤 4: 然后使用 huffman 树的构建方法, 得到一颗 huffman 二叉树。

```
//建立 huffman 树
zip_huffman_node *zip_huffman::createtree(priority_queue<zip_huffman_node> &heap_huffman)
{
    while (heap_huffman.size() > 1)
    {
        zip_huffman_node *tnode = new zip_huffman_node();
        zip_huffman_node *firstnode = new zip_huffman_node();
        zip_huffman_node *secnode = new zip_huffman_node();
        *firstnode = heap_huffman.top();
        heap_huffman.pop();
        *secnode = heap_huffman.top();
        heap_huffman.pop();
        tnode->left_child = firstnode;
        tnode->right_child = secnode;
        secnode->parent = tnode;
        firstnode->parent = tnode;
        tnode->count = firstnode->count + secnode->count;
    }
}
```

```

        heap_huffman.push(*tnode);
    }
    zip_huffman_node *ret = new zip_huffman_node();
    *ret = heap_huffman.top();
    return ret;
}

```

步骤 5: 根据 huffman 树, 构建各个字符的前缀码。

```

//根据 huffman 树, 构建各个字符的前缀码
void zip_huffman::travel(zip_huffman_node *root, string &st)
{
    if (root == nullptr)
    {
        return;
    }
    if (root->left_child == nullptr && root->right_child == nullptr)
    {
        //cout << (int)root->c << " " << (int)root->c << " " << st << endl;
        huffman_map[root->c] = st;

        return;
    }
    if (root->left_child != nullptr)
    {
        st.push_back('0');
        travel(root->left_child, st);
        st.erase(st.end() - 1);
    }
    if (root->right_child != nullptr)
    {
        st.push_back('1');
        travel(root->right_child, st);
        st.erase(st.end() - 1);
    }
}

```

步骤 6: 根据字符与前缀码的抓换表, 将源文件中的字符按照顺序一一转换, 将结果存入 `str_zip` 字符串中。

```
void zip_huffman::genzipstr(string huffman_map[], string &str_input, string &str_zip)
{
    //cout<<"input str size"<<str_input.size()<<endl;
    for (auto i : str_input)
    {
        unsigned char t=i;
        str_zip += huffman_map[t];
    }
    //cout << str_zip;
}
```

步骤 7: 根据转换后的结果, 将 `str_zip` 中的字符按照 8 个做为一个字节的方式, 写入压缩文件中。同时要记录字符串不是 8 的倍数的情况下最后补充的 0 的数量, 此时完成压缩文件的写入。

```
void zip_huffman::writezipfile(string &str_zip, string inputfile_name)
{
    string zipfile_name = outputfile_name;
    ofstream s_zipfile;
    s_zipfile.open(zipfile_name, ios::out|ios::binary);
    if (!s_zipfile.is_open())
        return;
    int count = 0;
    char temp = 0;
    //cout << endl;
    for (auto &i : str_zip)
    {
        temp *= 2;
        temp += (i - '0');
        count++;
        //cout << i;
        if (count == 8)
        {
            //cout << " " << hex << (short)temp;
            //cout << endl;
        }
    }
}
```

```

        s_zipfile << temp;
        temp = 0;
        count = 0;
    }
}
}

```

步骤 8: 将最后补充 0 的数量, 前缀码转换表存入解压信息 key 文件中, 完成解压缩信息文件的写入。

```

//写入压缩信息的文件
void zip_huffman::writekeyfile(string huffman_map[], int chunk, string
inputfile_name)
{
    string keyfile_name = outputfile_name + ".key";
    //cout << keyfile_name << endl;
    ofstream s_keyfile;
    s_keyfile.open(keyfile_name, ios::out|ios::binary);
    if (!s_keyfile.is_open())
        return;
    s_keyfile << chunk << endl;
    int vaild_key_count = 0;
    for (int i = 0; i < 256; i++)
    {
        if (huffman_map[i] != "")
            vaild_key_count++;
    }
    s_keyfile << vaild_key_count << endl;
    for (int i = 0; i < 256; i++)
    {
        if (huffman_map[i] != "")
        {
            s_keyfile << i << " " << huffman_map[i] << endl;
        }
    }
    //cout << "writekeyfile" << endl;
}

```

步骤 9: 释放创建 huffman 树所使用的内存空间。

```
//删除 huffman 树, 防止内存泄漏
void zip_huffman::release_huffman_tree(zip_huffman_node *root)
{
    if (root == nullptr)
        return;
    zip_huffman_node *now = root;
    release_huffman_tree(root->right_child);
    release_huffman_tree(root->left_child);
    delete now;
}
```

第二部分: 在解压缩过程中, 通过一个 unzip_huffman 的类来实现。

步骤 1: 读取压缩文件, 将压缩文件的所有字符读取到 zip_str 中, 同时将 key 文件中的字节对前缀码转换表写入到 string huffman_map[256] 中, 同时根据 key 文件中记录的向最后补充的 0 的数量, 去除相应的字符。

```
//读取文件到字符串
void unzip_huffman::readZipfile(string &zipfile_name, string &keyfile_name, string &zip_str)
{
    constructHuffmanMap(keyfile_name, huffman_map, chunk);
    readZipstr(zipfile_name, zip_str, chunk);
}
//将字符串的字节转换为 01 字符串
void unzip_huffman::readZipstr(string &zipfile_name, string &zip_str, int chunk)
{
    ifstream s_zipfile;
    s_zipfile.open(zipfile_name, ios::in | ios::binary);
    if (!s_zipfile.is_open()){
        fail();
        return;
    }
}
```

```

    }
    while (s_zipfile.peek() != EOF)
    {
        char t_temp[1];
        //s_zipfile >> temp;
        s_zipfile.read(t_temp, 1);
        unsigned char temp=t_temp[0];
        //cout << hex << (int)temp << " ";
        unsigned char mask = 128;
        string t_str;
        for (int i = 0; i < 8; i++)
        {
            t_str += ((temp & mask) != 0 ? '1' : '0');
            mask = mask / 2;
        }
        //cout << t_str << endl;
        zip_str += t_str;
    }
    for (int i = 0; i < chunk; i++)
    {
        zip_str.erase(zip_str.end() - 1);
    }
    //cout << zip_str << endl;
}

```

步骤 2：根据得到的前缀码转换表，重建 huffman 树。

```

//构建 huffman 树
void unzip_huffman::recreateHuffmanTree(string huffman_map[], unzip_huffman_node *root)
{
    for (int i = 0; i < 256; i++)
    {
        if (huffman_map[i] != "")
        {
            unzip_huffman_node *now = root;
            //cout << dec << i << " : ";
            for (int j = 0; j < huffman_map[i].size() - 1; j++)
            {
                if (huffman_map[i][j] == '0')
                {
                    if (now->right_child == NULL)
                    {

```



```

        unzip_huffman_node *temp = new unzip_huffman_node
de();
        now->right_child = temp;
    }
    now = now->right_child;
    //cout << 0;
}
if (huffman_map[i][j] == '1')
{
    if (now->left_child == NULL)
    {
        unzip_huffman_node *temp = new unzip_huffman_node
de();
        now->left_child = temp;
    }
    now = now->left_child;
    //cout << 1;
}
}
if (huffman_map[i][huffman_map[i].size() - 1] == '0')
{
    unzip_huffman_node *leave = new unzip_huffman_node(NULL
, NULL, true, i);
    now->right_child = leave;
    //cout << 0;
}
if (huffman_map[i][huffman_map[i].size() - 1] == '1')
{
    unzip_huffman_node *leave = new unzip_huffman_node(NULL
, NULL, true, i);
    now->left_child = leave;
    //cout << 1;
}
//cout << " end " << endl;
}
}
}
}

```

步骤 3: 根据 huffman 树和压缩字符串, 将解压缩后的字符串写入 unzip_str 中。具体的转换方式如下, 先判断当前所在 huffman 树的位置是不是叶子节点, 如果是, 则完成一次转换, 输出解压缩的字

字符串，同时将目前的节点位置重置为树根。否则，如果当前的压缩字符是‘0’则将目前指向的节点向左移动，如果当前的压缩字符是‘1’则将目前指向的节点向右移动，如此循环往复，直到所有压缩的字符串转换完毕。

```
//根据 huffman 树和压缩字符串，将解压缩后的字符串写入 unzip_str 中
void unzip_huffman::unzipFile(string &zip_str, string huffman_map[], string &unzip_str, unzip_huffman_node *root)
{
    unzip_huffman_node *now = root;
    for (auto &c : zip_str)
    {
        if (c == '0')
        {
            now = now->right_child;
            if (now->leave)
            {
                unzip_str += now->c;
                now = root;
            }
        }
        if (c == '1')
        {
            now = now->left_child;
            if (now->leave)
            {
                unzip_str += now->c;
                now = root;
            }
        }
    }
    //cout << unzip_str;
}
```

步骤 4: 将 unzip_str 按照字节的单位, 写入最终需要输出的文件中。

```
void unzip_huffman::writeUnzipFile(string &unzip_str, string &unzipfile_name)
{
    ofstream s_unzipfile;
    s_unzipfile.open(unzipfile_name, ios::out | ios::binary);
    if (!s_unzipfile.is_open())
        return;
    s_unzipfile << unzip_str;
}
```

步骤 5: 释放创建 huffman 树所使用的内存空间。

```
//删除 huffman 树
void unzip_huffman::releaseHuffmanTree(unzip_huffman_node *root)
{
    if (root == NULL)
    {
        return;
    }
    unzip_huffman_node *now = root;
    releaseHuffmanTree(root->right_child);
    releaseHuffmanTree(root->left_child);
    delete now;
}
```

程序使用和测试说明

由于在本次实验中, 老师的要求是解压缩任意文件, 所以, 在此使用了一张 bmp 格式的图片作为代表进行测试, 选择该种格式的图片进行压缩测试的优点如下:

1. 该格式展示的是一张图片，使用肉眼较容易直观地能观察到该图片的压缩与解压缩得到的结果是不是正确的。

2. bmp 格式是直接将原始的颜色信息存入文件中，比较好直接观察它的二进制文件来判断文件的信息。

3. 正是由于 bmp 格式的文件直接储存颜色信息，对于颜色内容不复杂的图像，使用 huffman 算法就能达到较高的压缩率，更加方便的体现该程序的压缩与解压缩功能。

测试压缩的图片文件 sunzip.bmp 显示的图像如下：

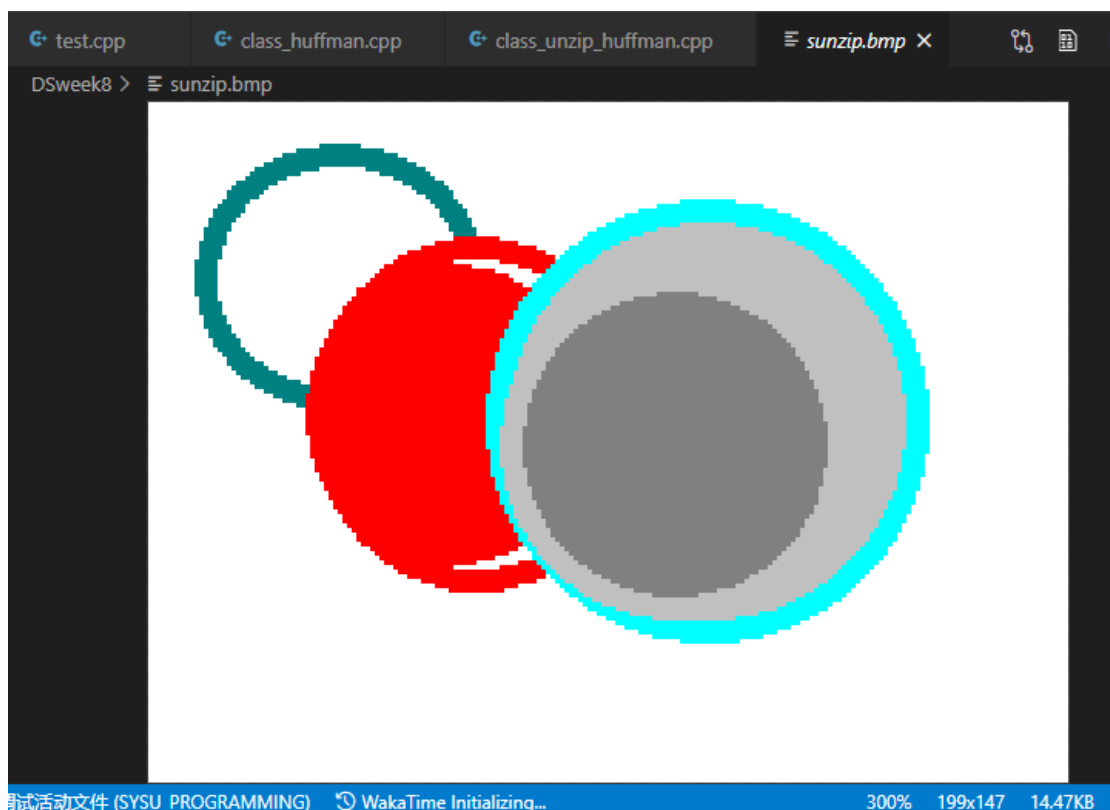


图 1-sunzip.bmp

其对应的二进制文件如下（节选）：

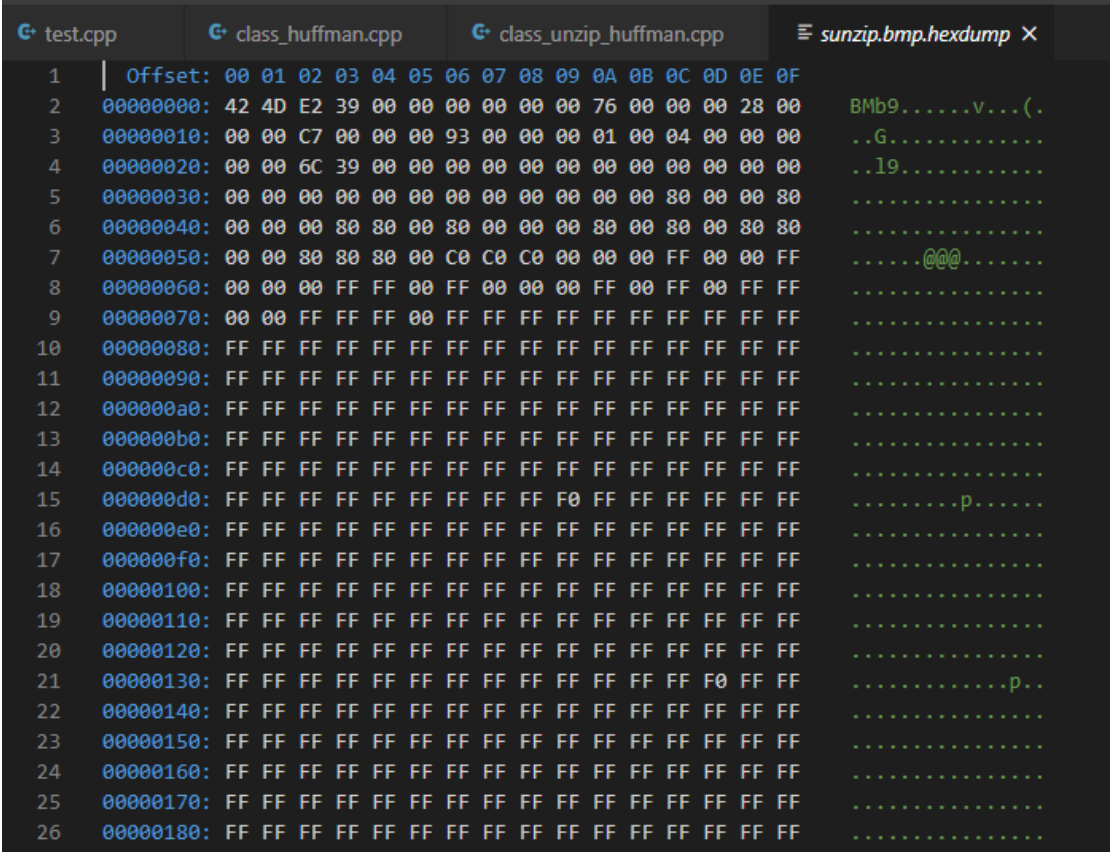


图 2-sunzip. bmp 对应的二进制文件（头部节选）

接下来进行压缩，程序执行与输出如下

用户输入为：

sunzip. bmp

zip

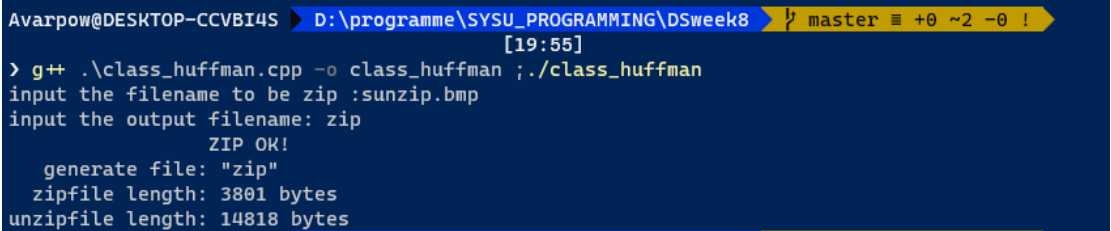


图 3-执行压缩程序

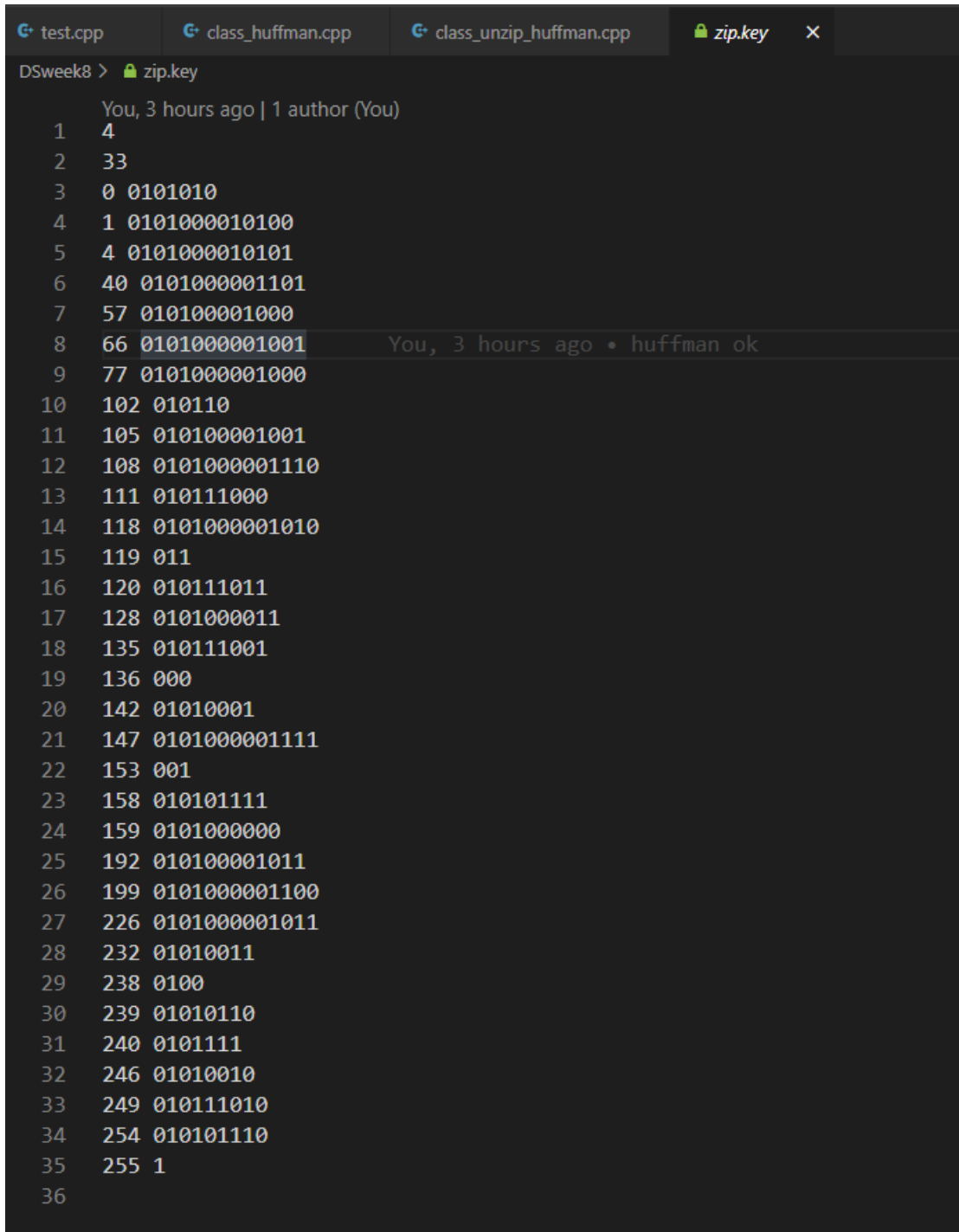
由上述输出，可见该压缩将文件压缩到大约四分之一的原始大小。

所生成的压缩文件 zip 的二进制如下（节选）：

test.cpp	class_huffman.cpp	class_unzip_huffman.cpp	zip.hexdump X
1	Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F		
2	00000000: 50 4A 82 14 16 A1 0A 95 2A 54 A9 52 82 95 2A 54		PJ...!...*T)R..*T
3	00000010: A0 D5 4A 95 28 31 52 A5 4A 0F 54 A9 52 85 15 28		.UJ.(1R%J.T)R..(
4	00000020: 55 52 A5 4A 95 28 39 42 15 2A 54 A9 52 A5 4A 95		UR%J.(9B.*T)R%J.
5	00000030: 2A 54 A9 52 A5 4A 95 2A 54 A9 52 A5 4A 94 35 4A		*T)R%J.*T)R%J.5J
6	00000040: 94 35 4A 95 28 6A 1A A5 0D 52 A5 4A 1A A5 0D 52		.5J.(j.%.R%J.%.R
7	00000050: 86 A1 AA 54 A1 A8 6A 1A A5 0B 50 B5 0B 54 A9 55		.!*T!(j.%.P5.T)U
8	00000060: 52 AA A5 4A B5 55 4A 95 55 55 6A 95 75 5F FF FF		R*%J5UJ.UUj.u_..
9	00000070: FF FF FF FF FF FF FF FF FD 7F FF FF FF FF FF	}......
10	00000080: FF FF FF FF FF FF FF FF 5F FF FF FF FF FF FF	_.....
11	00000090: FF FF FF FF D7 FF FF FF FF FF FF FF FF FF FF		...W.....
12	000000a0: FF F5 FF FF FF FF FF FF FF FF FF FF FF FD 7F		.u.....}..
13	000000b0: FF FF FF FF FF FF FF FF FF FF FF FF 5F FF FF	_....
14	000000c0: FF FF FF FF FF FF FF FF D7 FF FF FF FF FF FF	W.....
15	000000d0: FF FF FF FF FF FF F5 FF FF FF FF FF FF FF FF	u.....
16	000000e0: FF FF FF FD 7F FF FF FF FF FF FF FF FF FF FF		...}......
17	000000f0: FF 5F FF FF FF FF FF FF FF FF FF FF FF D7 FF		._.....W.
18	00000100: FF FF FF FF FF FF FF FF FF FF FF F5 FF FF FF	u....
19	00000110: FF FF FF FF FF FF FF FD 7F FF FF FF FF FF FF	}......
20	00000120: FF FF FF FF FF FF 5F FF FF FF FF FF FF FF FF	_.....
21	00000130: FF FF FF D7 FF FF FF FF FF FF FF FF FF FF FF		...W.....
22	00000140: F5 FF FF FF FF FF FF FF FF FF FF FF FD 7F FF		u.....}..
23	00000150: FF FF FF FF FF FF FF FF FF FF FF 5F FF FF FF	_....
24	00000160: FF FF FF FF FF FF D7 FF FF FF FF FF FF FF FF	W.....
25	00000170: FF FF FF FF FF F5 FF FF FF FF FF FF FF FF FF	u.....
26	00000180: FF FF FD 7F FF FF FF FF FF FF FF FF FF FF FF		..}......
27	00000190: 5F FF FF FF FF FF FF FF FF FF FF FF D7 FF FF		_.....W..
28	000001a0: FF FF FF FF FF FF FF FF FF FF F5 FF FF FF FF	u.....

图 4-压缩后的文件 zip 二进制（头部节选）

所生成的用于储存解压缩信息的 key 文件如下：



```
test.cpp class_huffman.cpp class_unzip_huffman.cpp zip.key x
DSweek8 > zip.key
    You, 3 hours ago | 1 author (You)
1    4
2    33
3    0 0101010
4    1 0101000010100
5    4 0101000010101
6    40 0101000001101
7    57 010100001000
8    66 010100001001    You, 3 hours ago • huffman ok
9    77 010100001000
10   102 010110
11   105 010100001001
12   108 0101000001110
13   111 010111000
14   118 0101000001010
15   119 011
16   120 010111011
17   128 0101000011
18   135 010111001
19   136 000
20   142 01010001
21   147 0101000001111
22   153 001
23   158 010101111
24   159 0101000000
25   192 010100001011
26   199 0101000001100
27   226 0101000001011
28   232 01010011
29   238 0100
30   239 01010110
31   240 0101111
32   246 01010010
33   249 010111010
34   254 010101110
35   255 1
36
```

图 5-保存压缩信息的 zip.key 文件

其中第一行储存了在压缩文件的最后补充了几个‘0’来保证是 8 的倍数，第二行表示一个数字 n，接下来 n 行是不同的字节所对应的前缀码。

第二部分：接下来测试解压缩文件：

按照提示输入文件，执行解压缩过程输出语句如下，用户输入为：

zip

zip.key

```
> g++ .\class_unzip_huffman.cpp -o class_unzip_huffman ; ./class_unzip_huffman
input the zipfile name: zip
input the key file name: zip.key
input the output file name: myunzip.bmp
UNZIP OK!
generate file: "myunzip.bmp"
zipfile length: 3800 bytes
unzipfile length: 14818 bytes
```

图 6-执行解压缩程序

这里显示的压缩后的字符串长度与原始不相符是因为这里计算的是删除了因为不是 8 的倍数所补充的 ‘0’ 字符之后的长度，所以比压缩时候显示的长度少 1。

解压缩得到的文件 myunzip. bmp 文件如下

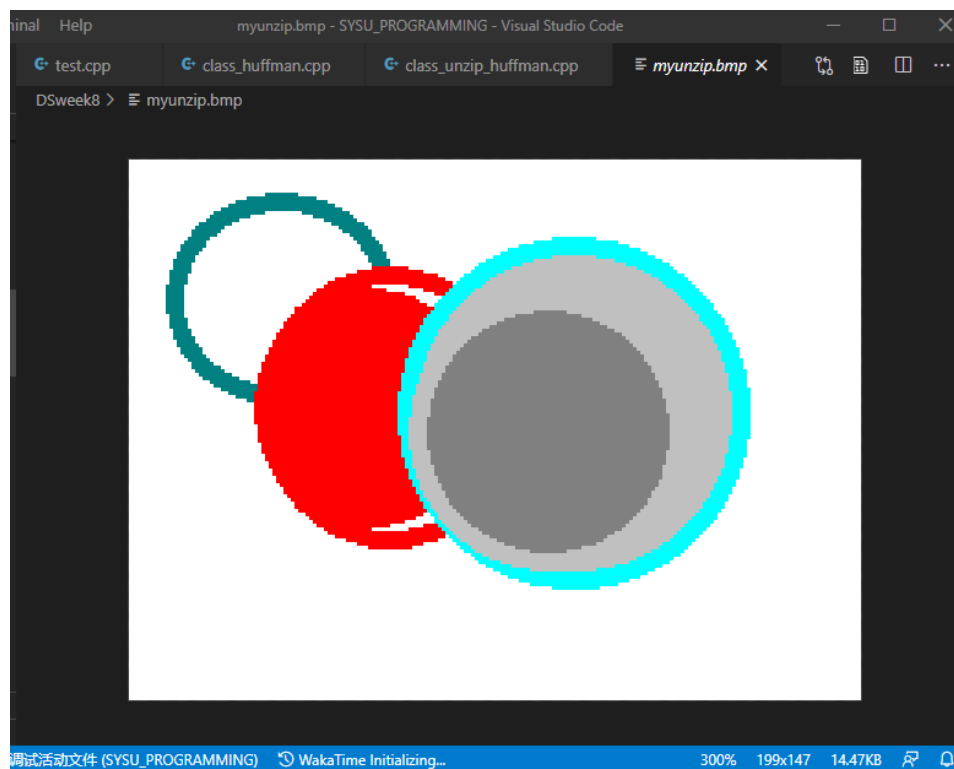
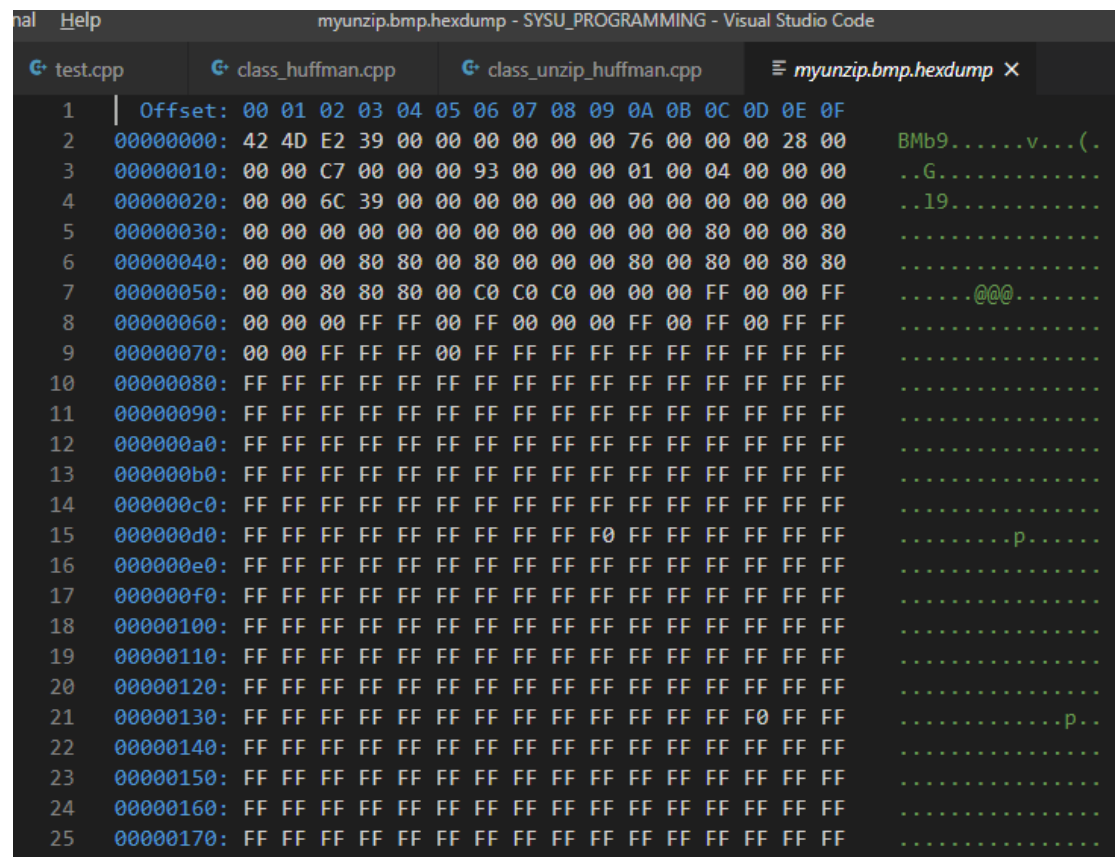


图 7-解压缩得到的 myunzip. bmp

myunzip.bmp 的二进制形式如下（节选）：



```
1 | Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
2 | 00000000: 42 4D E2 39 00 00 00 00 00 00 76 00 00 00 28 00  BMb9.....v...(.
3 | 00000010: 00 00 C7 00 00 00 93 00 00 00 01 00 04 00 00 00  ..G.....
4 | 00000020: 00 00 6C 39 00 00 00 00 00 00 00 00 00 00 00 00  ..19.....
5 | 00000030: 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80  .....
6 | 00000040: 00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80  .....
7 | 00000050: 00 00 80 80 80 00 C0 C0 C0 00 00 00 FF 00 00 FF  ....@@@.....
8 | 00000060: 00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF  .....
9 | 00000070: 00 00 FF FF FF 00 FF FF FF FF FF FF FF FF FF FF  .....
10 | 00000080: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  .....
11 | 00000090: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  .....
12 | 000000a0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  .....
13 | 000000b0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  .....
14 | 000000c0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  .....
15 | 000000d0: FF FF FF FF FF FF FF FF FF FF F0 FF FF FF FF FF  .....p.....
16 | 000000e0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  .....
17 | 000000f0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  .....
18 | 00000100: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  .....
19 | 00000110: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  .....
20 | 00000120: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  .....
21 | 00000130: FF FF FF FF FF FF FF FF FF FF FF FF FF FF F0 FF  .....p.....
22 | 00000140: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  .....
23 | 00000150: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  .....
24 | 00000160: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  .....
25 | 00000170: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  .....
26 | 00000180: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  .....
```

图 8-解压缩得到的 myunzip.bmp 二进制（头部节选）

经过 diff 对对比，解压缩后的文件与压缩前的文件一致。

综上，该实现程序能够实现对文件的压缩和解压缩过程。

总结与讨论：

在本次实验中，实现了一个 huffman 编码方式的压缩程序。在实验中也遇到了很多问题，一是对二进制文件的输入与输出不熟悉，在二进制文件中，所需要 `ios::binary` 来保证对回车字符的正常读写，在测试的时候，如果没有在打开文件的时候使用 `ios::binary`，在输出 `0x0a` 的时候会自动加上一个 `0x0d`，发生错误。

除此之外，还需要保证对 `string` 中的 `char` 和文件中用到的

unsigned char 的转换不发生错误。

在读取文件的时候也遇到了一个问题，定义了一个 char t, 接下来使用流输入直接读取文件的时候，会自动忽略的 0x0a 导致解压缩出来的图片发生了错位。后来通过

```
char t_temp[1];  
//s_zipfile >> temp;  
s_zipfile.read(t_temp, 1);  
unsigned char temp=t_temp[0];
```

这样的方式才避免了 0x0a 被吞掉，最终完成的将文件解压了出来。

参考文献

[1]. 《离散数学基础》周晓聪, 乔海燕 2020

附录：

由于本文件的代码文件较长，此处不再复制代码到这里。

代码文件 class_huffman.cpp:

https://github.com/avarpow/SYSU_PROGRAMMING/blob/master/DSweek8/class_huffman.cpp

代码文件 class_unzip_huffman.cpp:

https://github.com/avarpow/SYSU_PROGRAMMING/blob/master/DSweek8/class_unzip_huffman.cpp

用到的 bmp 图片 sunzip.bmp:

https://github.com/avarpow/SYSU_PROGRAMMING/blob/master/DSweek8/sunzip.bmp