

自适应Huffman编码

实验内容

使用自适应Huffman编码对英语小说进行压缩。（在此基础上，我还实现了解压缩功能）

实验原理

变长编码

根据熵的定义，可以给频繁出现的符号分配能够快速传输的码字（编码短），给不经常出现的符号分配较长的码字，从而使得平均值降低。

Huffman编码

属于变长编码的一种，由David A.Huffman提出，通过构造Huffman树来实现。Huffman树为一棵满二叉树，采用自底向上的方式构建，所有可能的符号都对应一个叶节点，从根节点开始，沿左（0）或者右（1）一直到叶节点所走的路径对应的就是该符号的二进制编码。

Huffman编码又分为静态Huffman编码和自适应Huffman编码。静态Huffman编码需要首先统计被编码的所有符号出现的频率，并据此进行Huffman树的构造。这就需要对被编码符号扫描两遍，这通常不能接受，且频率信息一般也很难获得。这就有了本实验中的自适应Huffman编码。

自适应Huffman编码（Adaptive Huffman Coding）

在自适应Huffman编码中，统计数据是随着数据流的到达而动态地收集和更新的。概率不再是基于先验知识而是基于到目前为止实际收到的数据，随着接收到的符号的概率分布的改变，符号会被赋予新的（更长或更短的）码字

自适应Huffman编码伪代码：

```
1 Initial_code(); //为符号分配初始码字，本实验中使用八位ASCII码作为初始码
2 Initial_tree(); //初始化编码树
3 while not EOF{
4     get(c); //读入一个字符
5     encode(c); //获得该符号当前的编码：寻找从树根到叶节点的路径
6     update_tree(c); //更新编码树：将符号出现次数加1，调整节点位置（如果需要）
7 }
```

在自适应Huffman编码树中，每个节点有两个属性：节点编号和节点权重（即当前出现的频率），并满足以下**兄弟性质**（sibling property）：

- 节点按照从左到右，从下到上的递增顺序编号
- 权重值较大的节点，节点编号也较大；
- 父节点的节点编号总是大于子节点的节点编号

因此，伪代码中说到的调整节点位置就是为了始终保持兄弟性质。当新符号读入时，相应叶节点权重加1，就有可能破坏该性质。这时，具有权重N的最远节点（编号最大）将会与权重刚刚增加到N+1的节点交换，如果权重为N的节点不是叶子节点，就要整个子树交换。交换后，可能需要对父节点进行递归的同样操作，才能使最终满足兄弟性质。

此外，为了标识字符是第一次出现的，引入了一个NYT（Not Yet Transmitted），在教材中为NEW。我们会在新出现的字符编码前，先发送NEW的编码，标识接下来的内容暂时不是Huffman编码（实验中采用八位ASCII编码）。因此，Huffman树中，NEW也使用一个叶节点表示（初始化树的时候只有一个NEW节点），并且NEW的权重永远保持为0。

实验流程

程序共设计了两个类：

- Adpt_Huffman_Tree：实现自适应Huffman编码树
- Bits2Byte：实现将01编码组成字节写入文件

基于上述原理，以下为程序中的一些具体实现：

树节点编号

通过分析，不难发现，当需要交换时，我们总是按照从下到上，从左到右的蛇形顺序，去寻找与节点未更新前权重相同且编号最大的那个被交换节点。那么，我们自然也可以实现为从根节点开始，从上到下，从右到左的顺序查找第一个与未更新前权重相同的节点，即广度优先搜索（BFS）。因为考虑到编号后交换的过程中可能需要修改，本实验中我就采用了BFS的方式来查找交换的节点，间接但同样实现编号的作用。树节点Node结构体定义如下：

```
1 //同时，由于树节点并不都表示字符，并没有在Node结构体中增加char成员
2 struct Node
3 {
4     int weight;
5     Node* parent, *left, *right;
6
7     Node(int c, Node* p = NULL, Node* l = NULL, Node* r = NULL)
8         :weight(c), parent(p), left(l), right(r)
9     {}
10 };
```

Huffman树相关操作

在Huffman树中，由于上述提到的，Node节点不包含字符信息，因此，为了快速找到字符对应的叶节点，我使用了map<char, Node*>容器，实现节点快速查找，避免每次都要遍历。获取对应字符当前编码的函数如下：

```

1 // 首先获得对应的叶节点，然后从该叶节点向上走到根节点，利用stack倒序输出路径即可
2 // 参数node为指针的引用，可以获取对应的叶节点，便于后续对其权重的更新
3 string Adpt_Huffman_Tree::getCode(char c, Node*& node) {
4     string huffmanCode = "";
5     map<char, Node*>::iterator it;
6
7     if ((it = charSet.find(c)) != charSet.end()) {
8         node = it->second;
9         Node* temp = it->second;
10        stack<char> s;
11        nodeSide side = getNodeSide(temp);
12        while (side != Root) {
13            if (side == LeftChild)
14                s.push('0');
15            else
16                s.push('1');
17            temp = temp->parent;
18            side = getNodeSide(temp);
19        }
20
21        while (!s.empty()) {
22            huffmanCode += s.top();
23            s.pop();
24        }
25    }
26    return huffmanCode;
27 }

```

更新哈夫曼编码树：

```

1 //每读取一个字符就调用一次
2 void Adpt_Huffman_Tree::update(char c, Node* node) {
3     if (node == NULL) {
4         //node为NULL，表示树中尚未有该字母对应的节点（新增字母）
5         Node* cc = new Node(1, newSignal);
6         Node* nn = new Node(0, newSignal);
7         newSignal->left = nn;
8         newSignal->right = cc;
9         newSignal = newSignal->left;
10
11        //放入map
12        charSet[c] = cc;
13        nodeSet[cc] = c;
14
15        addNodeWeight(cc->parent);
16    }
17    else {
18        addNodeWeight(node);
19    }
20 }

```

增加权重的函数：

```

1 //类似递归调用，父节点的权重也需要增加，并检查是否需要交换
2 void Adpt_Huffman_Tree::addNodeWeight(Node* node) {
3     while (node != nullptr) {
4         Node* toSwap = findNode(node->weight);
5         if (node != toSwap && node->parent != toSwap) {
6             swapNode(node, toSwap);
7         }
8         node->weight++;
9         node = node->parent;
10    }
11 }

```

还有交换节点的函数swapNode，以及寻找要被交换的函数findNode。swapNode函数就是简单的指针操作，findNode函数就是之前讲的BFS。这里就不一一贴出代码了。

编码文件的写入

由于计算机都是按照字节工作的，c++中没有可以直接向文件中写入二进制01串的方法（没有按位写入的函数），而我们生成的编码是二进制的位。因此，对于我们的编码，还需要按照八个一组转换成Byte的形式，才可以写入。（1Byte = 8 Bits）

程序中，类Bits2Byte实现了该功能，在类中使用vector<int>容器按位存储01串。我们把得到的编码存入vector中，利用如下转换函数：（该类其余函数见源代码文件）

```

1 //generate one byte
2 int Bits2Byte::generateByte()
3 {
4     int num = 0;
5     for (int i = 0; i < 8; i++) {
6         int p = 1;
7         for (int j = 1; j <= 7 - i; j++)
8             p *= 2;
9         num += bits[i] * p;
10    }
11
12    for (int i = 0; i < 8; i++)
13        bits.erase(bits.begin());
14    return num;
15 }

```

压缩函数

进行压缩的函数也在Adpt_Huffman_Tree中实现，在之前已给出伪代码，基本就是以上这些函数的调用了。由于函数较长，这里也不贴出了。可以见源代码。

解压缩函数

解压缩函数原理为：读取二进制文件，但由于也只能按byte读取，我们需要首先转化为01编码（类Bits2Byte中的addBits函数）。然后利用得到的编码从树根开始往下走，达到叶节点就得到对应的字符了。同样需要在这过程中不断地更新树，策略与压缩时相同。同样的，为了快速得到叶结点对应的字符，类中维护了map<Node*, char>这样一个数据结构。

实验结果

在实验中，为了验证程序在大文本的情况下不出错，按顺序核对所有编码的方式肯定是不合实际的。所以，为了确保程序不出错，我的程序中还实现了解压缩功能，以验证压缩后的文件是否可以还原成原文件。

程序运行结果如下：（以下运行都是在Visual Studio 2017中进行的，确认无误。文本文件最好和源代码置于同一文件夹下）

压缩

C:\Users\97922\source\repos\adaptive_huffman_coding\Debug\adaptive_huffman_coding.exe

```
-----Adaptive Huffman Coding-----  
  
1、Compress  
2、Extract  
3、Quit  
  
-----  
Enter your choice: 1  
Please enter the filename that you want to compress: 英语短篇小说集(小王子英文版).txt  
Please enter the filename after compress: 小王子压缩版.txt  
Compressing...  
Compress completed! Compression rate: 1.78351  
  
Enter your choice:
```

📄 小王子压缩版.txt	2019/6/2 15:19	文本文档	50 KB
📄 英语短篇小说集(小王子英文版).txt	2019/5/23 15:17	文本文档	93 KB

可以看到，经过压缩后，文件大小减少了将近一半，效果还算挺好的。

压缩文件内的内容（为二进制乱码）：



文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

□eA?"d2脛\W?□?□y√櫟z莎F洸QB鼎X ?A\$爨□?]'螳4氦#G\?鉤□抖
□z翮嘩庖i□馨M鱗m G-□-棉訛诿犖糞5 嘆^□焗? 簪扞□
H??舂澗|;E鏢sD{E?u鎔襖鴉櫨轰□^6黔□z5量)Y裸_Z%l筭畎Uq 鳶詳
羸l:Q□%裨!煙?&PCH铈脩??业W脈躑p乏□:宿弭]檻□□c鐙\$<□圩{A/酶i
??辺h}m>□掬鵠增 膨d}B?O腴鳩? g■臨騷^v{□!?i)JR櫓m轆6]??N
V 嶸n?_□>?孚|湟 鑄漲T彝遣□^z;銓1j 祺葦?肱榛蠱o?彘憎析□>拔!
訖?、5?唏寐?b| 功墅C鮎□>邛k摻□;C腴?媛?+噉>]悝_僊嚴誼誕□?□? (+ 盪 濫 □□kg頡頔積] 撿□脞u{#[?瘡筐□?賅?纖鼈茫o勘囑稜~□裴硫銑j>
竝c??V怵務桂□.廐□B蝟殍?□g刁鬚紡}倒U 嫩Y&麀A柑%閨 蜨c縹□稻
孀F♀綱?殼悠?垓崽□+鮒□孀F驕←-M 篇跂梦3?!调5?B?~xみ杓卒殆謁如
髯慘^麒□l 玳貳□鮒鮮t]邲鉢g鞋老l?f莫慶□B□ 或□□?砵埴;哈>搬鉗?□
鮐Ⅷ8!□%悖穉茵遯藿3?o靱Xx?穀7饒r臻孝o杖郇S?壠填?勘?7 7丈謾?
s芥L蔭g鯉楂l?□症揀/嬖嶠□K>□眉齎駘鬚茱疹5i7荊□許埴?吽y ?go銑厶
0m?銖?駙駙(n:cf{6=课統铄x}&?P?W7辟彬}/o邳V斂+剝具?3f?cF?±5
U?□少?+?韃z□璫?枱??齡錶z仇Ai (??я駘廊>L趸磨□映濊C|7駙.幹? 僂?
兹Ⅲc 銓pn濤?笙w|bXIo?m淞>楔^?諺K鈕?(簪遲齟{A0□諛铎檣K駘6
?弑Yw9}?*~舉□鷹蜂Q崙 w□蛎~|稽滯"芊挽□y?GkKv卜陽v稻漾櫓?#±
瞄K?□閏:R/収6强淡□l??紘~)?趁騶7漱/郡鸚|換 q_癢o楠牙徹/汎揉 ?;
□■▣▣□u7缺,越?螞■\E□??闍u\=□碳預□標乏絕瞰噫9总_捷}徹麻梯了?j^
笞?SpR?援咎z#淑瘞← 鵬Ur;Z[??柏?m鰲?鰈^b?K藟N嫠o鸛△IO□藕4稼
O 岨Gm霏 退珙謐咏舩0 Cd統?g}娵寢?6稻吸欽oe y□驤?b徹蹲g|
?澱▲r孳ぞ郷恂#w/瀏??m韞€嫵?荅L5↘ 杜Z缙?苍&翌5+N???铍k]濕?嘮
□擢圓od藪g▣w鋤\c=Y矯m□棋??薇c□卖招?鉄 忠祫?"雕'暇賣F碎斗.0
1炊/鞋(吒?浹杓v1子???^歲??雌螯p詞蔗飴Yv5鰾鮎緄^g)?鷹9?E戾縲□岬
?KSq?而困)稜鑣p饑K??\p諧/?苟Z垠□耒cz英瘡M鋤|杓輟 陪?□銅H譚
矯m□□□?鞋?g壘?淙儋□螭/郊?鯉礪覆^ U芊肫?薇:鵠_a_紼%U↓□?精}

解压

```

-----Adaptive Huffman Coding-----




    1、Compress
    2、Extract
    3、Quit

-----

Enter your choice: 1
Please enter the filename that you want to compress: 英语短篇小说集(小王子英文版).txt
Please enter the filename after compress: 小王子压缩版.txt
Compressing...
Compress completed! Compression rate: 1.78351

Enter your choice: 2
Please enter the filename that you want to extract: 小王子压缩版.txt
Please enter the filename after extract: 小王子解压版.txt
Extract...
Extract completed!

Enter your choice:
    
```

<input type="checkbox"/>  小王子解压版.txt	2019/6/2 15:23	文本文档	93 KB
 小王子压缩版.txt	2019/6/2 15:19	文本文档	50 KB
 英语短篇小说集(小王子英文版).txt	2019/5/23 15:17	文本文档	93 KB

解压后产生的新文件与老师给定文件大小一致，且内容、格式相同：

To Leon Werth 小王子英语版

I ask the indulgence of the children who may read this book for d
even books about children. I have a third reason: he lives in Franc
I will dedicate the book to the child from whom this grown-up gre

To Leon Werth
when he was a little boy

[Chapter 1]

- we are introduced to the narrator
a pilot
and his ideas about grown-ups

程序退出

C:\Users\97922\source\repos\adaptive_huffman_coding\Debug\adaptive_huffman_coding.exe

```
-----Adaptive Huffman Coding-----  
  
1、Compress  
2、Extract  
3、Quit  
  
-----  
Enter your choice: 1  
Please enter the filename that you want to compress: 英语短篇小说集(小王子英文版).txt  
Please enter the filename after compress: 小王子压缩版.txt  
Compressing...  
Compress completed! Compression rate: 1.78351  
  
Enter your choice: 2  
Please enter the filename that you want to extract: 小王子压缩版.txt  
Please enter the filename after extract: 小王子解压版.txt  
Extract...  
Extract completed!  
  
Enter your choice: 3  
Bye!
```

基于上述结果，可初步判断程序已经简单实现了自适应Huffman编码的功能。**压缩率**达到1.7，还算比较可观的。（压缩率计算的方式为：原文件大小/压缩后大小）

遇到的难点

按位的读写

之前已经提到过了，通过八个01位转化为一个Byte实现文件读写。

文件结束标记

在编码函数中，由于要转化为Byte，如果实际的01编码最后不足8位的话，我使用了0来填补。这就造成了在解压缩的时候，如何判断原编码哪里终止，而不把后面的0也算进去的问题。一种可行的方法是在编码的开头，对文件的大小进行额外编码。在本程序中，我通过在压缩完原文本后，再发送一个新字符（0x00，空字符）来标志文件结束（即先编码NEW，再编码0x00）。这样，当解压时，遇到NEW并且后面的字符为空字符，就可以得知解压结束了。

编码的快速查找

每次遍历寻找的话太慢，所以使用了一个map数据结构记录已编码的字符对应的叶节点，然后从该叶节点向上利用parent指针寻找到根的路径即可。解压时同理，也使用map记录叶节点对应的字符。

实验体会

心得

本次的实验过程并不是十分顺利，但却让我收获了很多。自适应Huffman编码的原理很简单也很易懂，但有些实现方面还是让我遇到了一些障碍，比如上述说到的一些难点。不过好在都成功解决了，实验的结果也让我比较满意，文件的压缩和还原的效果也很神奇。

经过本次的实验，让我对于Huffman编码的原理有了更加深入的认识，尤其是自适应Huffman编码的具体实现：它相比于静态Huffman编码，具有仅需单遍扫描、无需传送编码树、能够对输入符号流的局部统计规律变化作出反应等一系列优点，使得压缩率和压缩速度都得到了提高。

改进思路

由于之前所说的，我没有在节点结构体中使用编号，而是代替的采取了从上到下、从右到左的BFS方式寻找被交换的节点，这就可能造成了效率的降低，应该是本程序中最大可以改进的地方了。

基本的思路应该是使用一个链表或者数组的形式来顺序组织树节点，并把权重相同的树节点看成块，每个块中的节点按照编号顺序排列，这样查找起来可能会比较快。希望我以后可以加以改进。

此外，我在读文件的时候是每次只读取一个字符，也是导致速度慢的原因之一。可以使用一个Buffer数组，一次读取很多字符存储到Buffer中，以减少读取的次数。

代码附录

main.cpp

```
1 //encode using Adaptive Huffman Coding
2 #include "Adpt_Huffman_Tree.h"
3
4 void menu() {
5     cout << "-----Adaptive Huffman Coding-----\n" << endl;
6     cout << "1、Compress" << endl;
7     cout << "2、Extract" << endl;
8     cout << "3、Quit\n" << endl;
9     cout << "-----\n" << endl;
10    cout << "Enter your choice: ";
11 }
12
13
14 int main() {
15     Adpt_Huffman_Tree huffman;
16
17     menu();
18     int choice;
19     string filePath_in, filePath_out;
20     cin >> choice;
21     while (1) {
22         switch (choice) {
23             case 1:
24                 cout << "Please enter the filename that you want to compress: ";
25                 cin >> filePath_in;
26                 cout << "Please enter the filename after compress: ";
27                 cin >> filePath_out;
28                 huffman.compress(filePath_in, filePath_out);
29                 huffman.reset();
30                 cout << "\nEnter your choice: ";
31                 break;
32             case 2:
33                 cout << "Please enter the filename that you want to extract: ";
34                 cin >> filePath_in;
```

```

35         cout << "Please enter the filename after extract: ";
36         cin >> filePath_out;
37         huffman.extract(filePath_in, filePath_out);
38         huffman.reset();
39         cout << "\nEnter your choice: ";
40         break;
41     case 3:
42         cout << "Bye!" << endl;
43         break;
44     default:
45         cout << "wrong choice! Try again: ";
46     }
47     if (choice == 3)
48         break;
49     cin >> choice;
50 }
51 return 0;
52 }
53

```

Bits2Byte.h

```

1  #ifndef BITS2BYTE
2  #define BITS2BYTE
3
4  #include <vector>
5  using namespace std;
6
7  class Bits2Byte {
8  public:
9      Bits2Byte();
10     ~Bits2Byte();
11     void addBits(string str);
12     void addBits(char c);
13     int size();
14     int generateByte();
15     int readBit();
16     int addZero();
17
18 private:
19     vector<int> bits;
20 };
21 #endif
22

```

Bits2Byte.cpp

```

1  #include "Bits2Byte.h"
2  #include <string>
3  #include <iostream>
4  using namespace std;
5
6  Bits2Byte::Bits2Byte() {}

```

```

7
8 void Bits2Byte::addBits(string str)
9 {
10     for (auto i : str) {
11         if (i == '1')
12             bits.push_back(1);
13         else
14             bits.push_back(0);
15     }
16 }
17
18 void Bits2Byte::addBits(char c) {
19     for (int i = 7; i >= 0; i--) {
20         bits.push_back(((c >> i) & 1));
21     }
22 }
23
24 Bits2Byte::~Bits2Byte()
25 {
26     bits.clear();
27 }
28
29 int Bits2Byte::size()
30 {
31     return bits.size();
32 }
33
34 int Bits2Byte::generateByte()
35 {
36     int num = 0;
37     for (int i = 0; i < 8; i++) {
38         int p = 1;
39         for (int j = 1; j <= 7 - i; j++)
40             p *= 2;
41         num += bits[i] * p;
42     }
43
44     for (int i = 0; i < 8; i++)
45         bits.erase(bits.begin());
46     return num;
47 }
48
49 int Bits2Byte::readBit() {
50     if (bits.size() == 0)
51         return -1;
52     int bit = bits[0];
53     bits.erase(bits.begin());
54     return bit;
55 }
56
57 int Bits2Byte::addZero()
58 {
59     while (bits.size() % 8 != 0) {
60         bits.push_back(0);
61     }
62     return 0;
63 }

```

Adpt_Huffman_Tree.h

```

1  #ifndef ADPT_HUFFMAN_TREE
2  #define ADPT_HUFFMAN_TREE
3
4  #include <fstream>
5  #include <iostream>
6  #include <queue>
7  #include <map>
8  #include <stack>
9  #include <string>
10 #include "Bits2Byte.h"
11 using namespace std;
12
13 #define NEW 0x00
14 enum nodeSide { Root, LeftChild, RightChild };
15
16 struct Node
17 {
18     int weight;
19     Node* parent, *left, *right;
20
21     Node(int c, Node* p = NULL, Node* l = NULL, Node* r = NULL)
22         :weight(c), parent(p), left(l), right(r)
23     {}
24 };
25
26
27 class Adpt_Huffman_Tree {
28 public:
29     Adpt_Huffman_Tree();
30     ~Adpt_Huffman_Tree();
31     bool compress(string filePath_in, string filePath_out);
32     bool extract(string filePath, string filePath_out);
33     void destory();
34     void reset();
35 private:
36     Node* root;
37     Node* newSignal;
38
39     map<char, Node*> charSet; //记录每个字符对应的结点
40     map<Node*, char> nodeSet; //记录节点对应的字符
41     void update(char c, Node* node);
42     void addNodeWeight(Node* node); //递归增加, update函数通过调用这个函数实现
43     Node* findNode(int weight); //寻找同weight中index最大的
44     nodeSide getNodeSide(Node* node);
45     void swapNode(Node* a, Node* b);
46     string getCode(char c, Node*& node);
47     string getCode(Node* node);
48
49     void destory(Node* root);
50 };
51
52 #endif

```

Adpt_Huffman_Tree.cpp

```
1  #include "Adpt_Huffman_Tree.h"
2
3  Adpt_Huffman_Tree::Adpt_Huffman_Tree() {
4      root = new Node(0); //初始结点, 就是一个NEW, 其权重为0
5      newSignal = root;
6  }
7  Adpt_Huffman_Tree::~Adpt_Huffman_Tree() {
8      destory(root);
9  }
10 bool Adpt_Huffman_Tree::compress(string filePath_in, string filePath_out) {
11     ifstream fin(filePath_in, ios::in);
12     ofstream fout(filePath_out, ios::out | ios::binary);
13     if (!fin.is_open() || !fout.is_open()) {
14         cout << "error: file is not exist!" << endl;
15         return false;
16     }
17
18     char c;
19     Node* node;
20     string huffmanCode;
21     int byte;
22     Bits2Byte bits;
23
24     long long originBit = 0;
25     long long compressBit = 0;
26     cout << "Compressing..." << endl;
27
28     while (fin.peek() != EOF) {
29         c = fin.get(); //读一个字符
30         originBit += 8;
31         if ((huffmanCode = getCode(c, node)) != "") {
32             //cout << c << "已存在, 当前编码为: " << huffmanCode << endl;
33             bits.addBits(huffmanCode);
34             update(c, node); //更新树
35         }
36         else {
37             //cout << c << "首次出现, 设定其编码为原ascii码, 并在之前先编码NEW" << endl;
38             bits.addBits(getCode(newSignal));
39             bits.addBits(c);
40             update(c, NULL);
41         }
42         if (bits.size() >= 8) {
43             byte = bits.generateByte();
44             const char* code = new char(byte);
45             fout.write(code, sizeof(char));
46             compressBit += 8;
47             delete code;
48         }
49     }
50
51     bits.addBits(getCode(newSignal));
52     bits.addBits(NEW);
53 }
```

```

54     bits.addZero();
55     while (bits.size() != 0) {
56         byte = bits.generateByte();
57         const char* code = new char(byte);
58         fout.write(code, sizeof(char));
59         compressBit += 8;
60         delete code;
61     }
62
63     fin.close();
64     fout.close();
65
66     cout << "Compress completed! Compression rate: " << (double)originBit / compressBit <<
endl;
67     return true;
68 }
69
70 bool Adpt_Huffman_Tree::extract(string filePath_in, string filePath_out) {
71     ifstream fin(filePath_in, ios::in | ios::binary);
72     ofstream fout(filePath_out, ios::out);
73     if (!fin.is_open() || !fout.is_open()) {
74         cout << "error: file is not exist!" << endl;
75         return false;
76     }
77
78     int bit = 0;
79     char c;
80     bool firstBit = true, end = false;
81     Node* node = root;
82     Bits2Byte bits;
83     char buffer[1];
84
85     cout << "Extract..." << endl;
86
87     while (fin.peek() != EOF && end != true) {
88         fin.read(buffer, sizeof(buffer));
89         bits.addBits(buffer[0]);
90
91         //对于第一个0特殊处理
92         if (firstBit) {
93             bits.readBit();
94             firstBit = false;
95         }
96
97         while (node->left != NULL && node->left != NULL && end != true) {
98             bit = bits.readBit();
99             if (bit == -1)
100                 break;
101             if (bit == 0)
102                 node = node->left;
103             else
104                 node = node->right;
105         }
106         if (bit == -1)
107             continue;
108
109         if (node == newSignal) {

```



```

110         if (bits.size() < 8) {
111             fin.read(buffer, sizeof(buffer));
112             bits.addBits(buffer[0]);
113         }
114         c = (char)bits.generateByte();
115         if (c == NEW) {
116             end = true;
117             continue;
118         }
119         update(c, NULL);
120         fout << c;
121     }
122     else {
123         update(nodeSet[node], node);
124         fout << nodeSet[node];
125     }
126     node = root;
127 }
128
129 while (bits.size() != 0 && end != true) {
130     while (node->left != NULL && node->left != NULL && end != true) {
131         bit = bits.readBit();
132         if (bit == -1)
133             break;
134         if (bit == 0)
135             node = node->left;
136         else
137             node = node->right;
138     }
139
140     if (node == newSignal) {
141         if (bits.size() < 8) {
142             fin.read(buffer, sizeof(buffer));
143             bits.addBits(buffer[0]);
144         }
145         c = (char)bits.generateByte();
146         if (c == NEW) {
147             end = true;
148             continue;
149         }
150         update(c, NULL);
151         fout << c;
152     }
153     else {
154         update(nodeSet[node], node);
155         fout << nodeSet[node];
156     }
157     node = root;
158 }
159
160 cout << "Extract completed!" << endl;
161
162 fin.close();
163 fout.close();
164 return true;
165 }
166

```

```

167 void Adpt_Huffman_Tree::destory()
168 {
169     destory(root);
170 }
171
172 void Adpt_Huffman_Tree::reset()
173 {
174     destory(root);
175     root = new Node(0);
176     newSignal = root;
177 }
178
179 void Adpt_Huffman_Tree::destory(Node* root) {
180     if (root != NULL)
181         return;
182
183     destory(root->left);
184     destory(root->right);
185     delete root;
186     root = NULL;
187 }
188
189 void Adpt_Huffman_Tree::update(char c, Node* node) {
190     if (node == NULL) {
191         //新增字母
192         Node* cc = new Node(1, newSignal);
193         Node* nn = new Node(0, newSignal);
194         newSignal->left = nn;
195         newSignal->right = cc;
196         newSignal = newSignal->left;
197
198         //放入map
199         charSet[c] = cc;
200         nodeSet[cc] = c;
201
202         addNodeWeight(cc->parent);
203     }
204     else {
205         addNodeWeight(node);
206     }
207 }
208
209 void Adpt_Huffman_Tree::addNodeWeight(Node* node) {
210     while (node != nullptr) {
211         Node* toSwap = findNode(node->weight);
212         if (node != toSwap && node->parent != toSwap) {
213             swapNode(node, toSwap);
214         }
215         node->weight++;
216         node = node->parent;
217     }
218 }
219
220 Node* Adpt_Huffman_Tree::findNode(int weight) {
221     Node* temp = NULL;
222     queue<Node*> q;
223     q.push(root);

```

```

224     while (!q.empty()) {
225         temp = q.front();
226         if (temp->weight == weight) {
227             return temp;
228         }
229         q.pop();
230         if (temp->right) q.push(temp->right);
231         if (temp->left) q.push(temp->left);
232     }
233     cout << endl;
234     return temp;
235 }
236
237 nodeSide Adpt_Huffman_Tree::getNodeSide(Node* node) {
238     if (node->parent == NULL)
239         return Root;
240     else if (node->parent->left == node)
241         return LeftChild;
242     else
243         return RightChild;
244 }
245
246
247 void Adpt_Huffman_Tree::swapNode(Node* a, Node* b) {
248     nodeSide a_side = getNodeSide(a);
249     nodeSide b_side = getNodeSide(b);
250     if (a_side == LeftChild) {
251         if (b_side == LeftChild) {
252             a->parent->left = b;
253             b->parent->left = a;
254         }
255         else {
256             a->parent->left = b;
257             b->parent->right = a;
258         }
259     }
260     else {
261         if (b_side == LeftChild) {
262             a->parent->right = b;
263             b->parent->left = a;
264         }
265         else {
266             a->parent->right = b;
267             b->parent->right = a;
268         }
269     }
270     Node* temp = a->parent;
271     a->parent = b->parent;
272     b->parent = temp;
273 }
274
275 string Adpt_Huffman_Tree::getCode(char c, Node*& node) {
276     string huffmanCode = "";
277     map<char, Node*>::iterator it;
278
279     if ((it = charSet.find(c)) != charSet.end()) {
280         node = it->second;
281         Node* temp = it->second;

```

```

281     stack<char> s;
282     nodeSide side = getNodeSide(temp);
283     while (side != Root) {
284         if (side == LeftChild)
285             s.push('0');
286         else
287             s.push('1');
288         temp = temp->parent;
289         side = getNodeSide(temp);
290     }
291
292     while (!s.empty()) {
293         huffmanCode += s.top();
294         s.pop();
295     }
296 }
297 return huffmanCode;
298 }
299
300 string Adpt_Huffman_Tree::getCode(Node* node)
301 {
302     //获取NEW的编码
303     string huffmanCode = "";
304     Node* temp = node;
305     stack<char> s;
306     nodeSide side = getNodeSide(temp);
307
308     // 第一次编码NEW时, 位于根节点, 所以要另外考虑
309     if (side == Root) {
310         return "0";
311     }
312
313     while (side != Root) {
314         if (side == LeftChild)
315             s.push('0');
316         else
317             s.push('1');
318         temp = temp->parent;
319         side = getNodeSide(temp);
320     }
321
322     while (!s.empty()) {
323         huffmanCode += s.top();
324         s.pop();
325     }
326     return huffmanCode;
327 }
328

```