



算法基础 Lab 6

Huffman 编码问题

张芷苒

PB21081601

Part 1: 实验要求

1. 编程实现 Huffman 编码问题，并理解其核心思想。
2. 对文件 original.txt 中所有的大小写字母、数字 (0-9) 以及标点符号（即：除空格换行符之外的所有字符）按照 Huffman 编码方式编码为 01 序列，输出如下格式的 table.txt 文件，并在控制台打印压缩率。（编码方式可能不唯一，但压缩率是确定的）

Part 2: 算法设计思路

首先，将所有的字符以及它们出现的频率存在结点中，并将这些结点放入优先队列，形成森林。

每一次将优先队列的前两个结点（频率最小的两个结点）出队，并合并成为一个新的结点，之后再将此结点加入优先队列，并维护优先队列的性质。

当优先队列中只剩下一个结点的时候，树便建立成功。

之后从根节点开始向下遍历，依次添加编码。

Part 3: 算法实现过程

结点结构：

```
1 // Huffman树的节点类
2 typedef struct node
3 {
4     char value;           // 节点的字符值
5     double freq;          // 节点字符出现的频度
6     vector<int> code;      // 节点字符对应的编码
7     node *lchild, *rchild; // 节点的左右孩子
8 } HFMNode, *pHFMNode;
```

优先队列比较方法：

```

1 struct CMP
2 {
3     bool operator()(const pHFMNode &p, const pHFMNode &q)
4     {
5         return p->freq > q->freq;
6     }
7 };

```

建立哈夫曼树:

```

1 void Huffman(priority_queue<pHFMNode, vector<pHFMNode>, CMP> &vctNode)
2 {
3     while (vctNode.size() > 1)
4     {
5         pHFMNode first = vctNode.top(); //取vctNode森林中频度最小的树根
6         vctNode.pop();
7         pHFMNode second = vctNode.top(); //取vctNode森林中频度第二小的树根
8         vctNode.pop();
9         pHFMNode merge = new HFNode; //合并上面两个树
10        merge->freq = first->freq + second->freq;
11        merge->lchild = first; //小的放左子树
12        merge->rchild = second; //大的放右子树
13        vctNode.push(merge); //向vctNode森林中添加合并后的merge树
14    }
15    return;
16 }

```

输出编码:

```

1 void PrintHuffman(pHFMNode node, vector<int> &vctCode, ofstream &outfile)
2 {
3     if ((node->lchild == NULL) && (node->rchild == NULL))
4     {
5         node->code.assign(vctCode.begin(), vctCode.end());
6         //复制vctCode到code
7         outfile << node->value << "    " << left << setw(8) << node->freq;
8         for (vector<int>::iterator iter = node->code.begin();
9             iter != node->code.end(); iter++)
10        {
11            outfile << *iter;
12        }
13        outfile << endl;
14        code_length += vctCode.size() * node->freq;
15    }

```

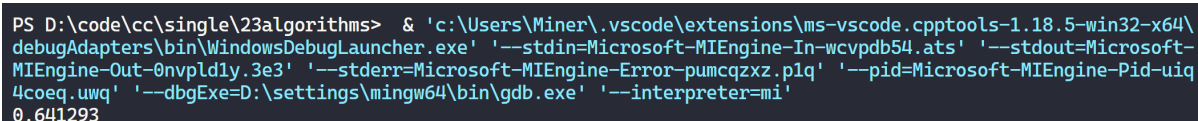
```

16     else
17     {
18         vctCode.push_back(0); //遇到左子树时给vctCode中加一个0
19         PrintHuffman(node->lchild, vctCode, outfile);
20         vctCode.pop_back(); //回溯，删除刚刚加进去的1
21         vctCode.push_back(1); //遇到右子树时给vctCode中加一个1
22         PrintHuffman(node->rchild, vctCode, outfile);
23         vctCode.pop_back(); //回溯，删除刚刚加进去的0
24     }
25     return;
26 }

```

Part 4: 实验结果分析

运行程序，可以得到压缩率：



```

PS D:\code\cc\single\23algorithms> & 'c:\Users\Miner\.vscode\extensions\ms-vscode.cpptools-1.18.5-win32-x64\
debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-wcvpdb54.ats' '--stdout=Microsoft-
MIEngine-Out-0nvpldly.3e3' '--stderr=Microsoft-MIEngine-Error-pumcqzxx.plq' '--pid=Microsoft-MIEngine-Pid-uiq
4coeq.uwq' '--dbgExe=D:\settings\mingw64\bin\gdb.exe' '--interpreter=mi'
0.641293

```

图 1: 输出结果

同时，可以得到编码：

lab6 > v1 > ≡ table.txt

1	字符	出现频率	编码
2	d	6281	0000
3	f	2878	00010
4	u	3428	00011
5	m	3494	00100
6	w	3518	00101
7	F	188	001100000
8	E	95	0011000010
9	R	49	00110000110
10	P	50	00110000111
11	"	391	00110001
12	M	197	001100100
13	1	5	00110010100000
14	2	1	0011001010000100
15	K	1	0011001010000101
16	3	3	001100101000011
17	X	3	001100101000100
18	7	1	0011001010001010
19	5	2	0011001010001011
20	4	4	001100101000110
21	Q	4	001100101000111
22	U	15	0011001010010
23	9	4	001100101001100
24	6	2	0011001010011010
25	0	2	0011001010011011
26	V	9	00110010100111
27	;	56	00110010101
28	!	112	0011001011
29	T	430	00110011
30	,	1938	001101
31	'	233	001110000
32	W	120	0011100010
33	j	120	0011100011
34	N	245	001110010

图 2: 编码结果 (局部)

以上结果符合预期，由此可以认为算法正确。