Lab 0: 哈夫曼压缩

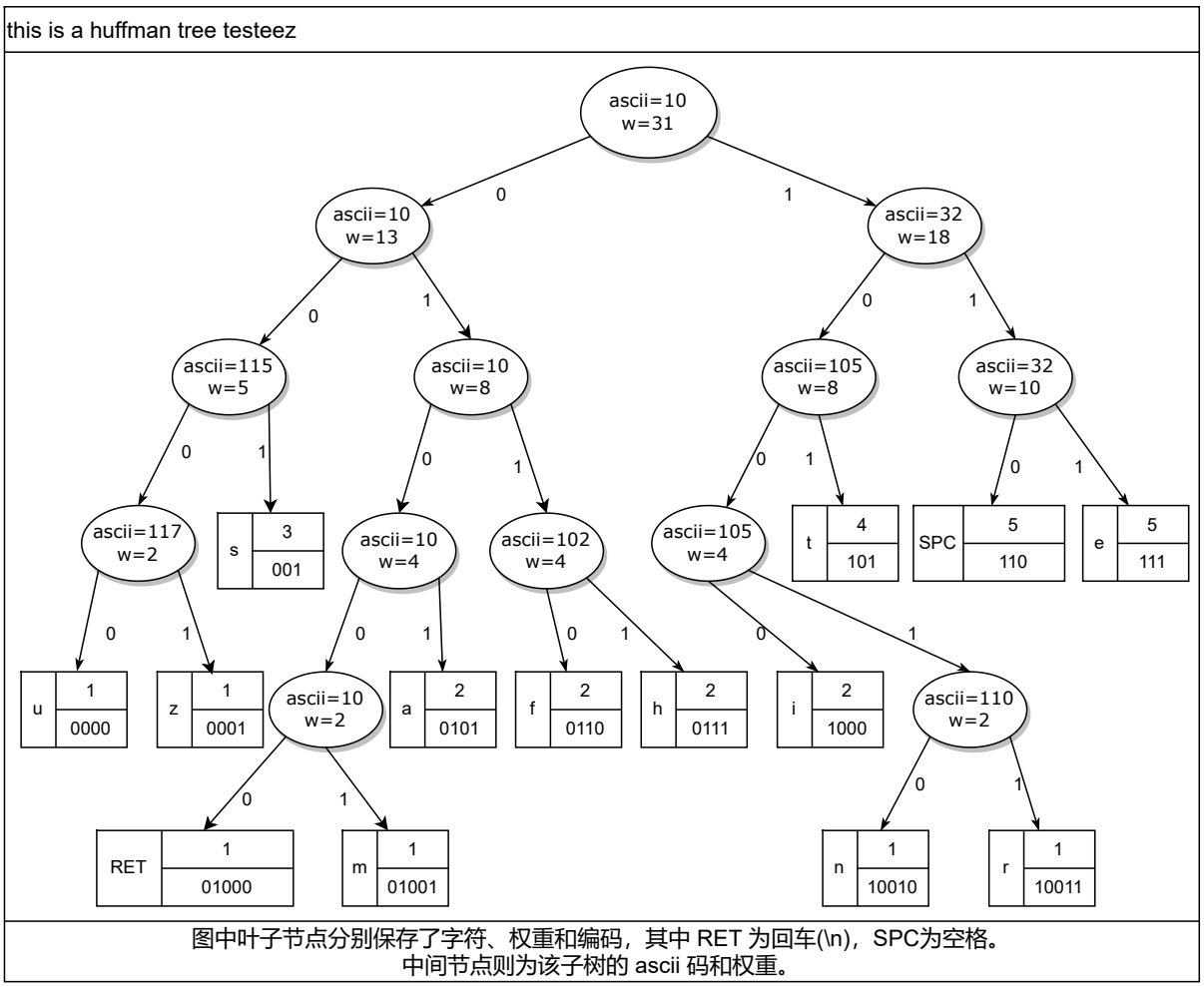
一、实验介绍

哈夫曼编码（Huffman Coding），是一种用于无损压缩编码的熵编码方法，由 美国计算机科学家 David Albert Huffman 于 1952 年发明。

哈夫曼编码使用变长编码表对原符号（如文件中的字符，或特定长度的字符组合）进行编码。变长编 码表通过评估符号出现的频率得到：出现频率高的符号使用较短的编码，而出现频率低的符号则使用较长的编码。这种编码方式使编码之后的字符串平均长度的期望值 降低，从而达到无损压缩数据的目的。

哈夫曼压缩中需要使用到哈夫曼树（Huffman Tree）。给定 N 个权值作为 N 个叶子结点，构造一棵 二叉树，若该树的带权路径长度达到最小，则称这样的二叉树为最优二叉树，也称为哈 夫曼树。哈夫曼树是带权路径长度最短的树，权值较大的结点离根 较近。

如下图所示，是一棵哈夫曼树的例子：



二、实验要求

本次 Lab 中，我们要求使用 C++ 实现哈夫曼树这一数据结构，并根据要求对指定文本进行压缩。

1、哈夫曼树

在 hftree.h 中，我们已经部分声明好了哈夫曼树类，其包括了一个枚举类 Option，用以区分不同的压缩编码要求（见下文 ***Option*** 部分），以及该哈夫曼树需要对外提供的功能。你需要自定义一些成员变量来帮助你实现哈夫曼树的功能，我们不对树的具体实现方式作任何要求。哈夫曼树对外提供的功能实现在 hftree.cpp 中，你需要完成的部分如下：

①**HfTree::hfTree(const std::string &text, const Option op)：** 哈夫曼树类的构造函数。text 表示输入的文本（通过 utils 模块中的函数从输入文件中取得，包括空格换行等字符），op 表示构建这棵哈夫曼树所遵循的编码规则。为了保证哈夫曼树的唯一性，在选择两个树合并时，**权值较小的树应作为左子树，权值较大的子树作为右子树。**若有多个树的权值相同，则优先选择字典序最小的树（一个树的字典序，为这个树中所有节点的字典序的最小值），同时，**字典序较小的子树应作为左子树，字典序较大的子树作为右子树。**

②**std::map<std::string, std::string> HfTree::getCodingTable()：** 编码。你需要根据哈夫曼树的结构对各个字符进行编码，其中向左走（左子树）应赋值为 0，向右走（右子树）应赋值为 1，将编码保存在一个 <string, string> 的 map 中并返回。

2、文件读写

与文件读写相关的函数定义在 utils.h 中，你需要完成的部分如下：

①**std::string parseText(const std::string &input)：** 读取待压缩编码的文件内容。input 表示待压缩文件的文件名，你需要将该文件内的全部内容（包括空格换行等字符）读入并储存在一个 string 变量中返回。

②**void output(const std::string &output, const std::string &data)：** 将 data 中的内容输出到 output 指定的文件。

③**std::string codingTable2String(const std::map<std::string, std::string> &codingTable)：** 将编码表转为字符串。该函数将前序步骤得到的编码表转为字符串便于后续输出，你需要将所有的原符号（**按字典序排列**）及其对应的编码按**原符号+空格+对应编码+换行符**（换行符应为\n）的格式拼接为一个字符串。

3、Option

如前文所述，哈夫曼编码使用变长编码表对原符号（如文件中的字符，或特定长度的字符组合）进行编码。本次 Lab 要求实现两种编码，即分别针对文件中的单个字符（Option::SingleChar）和单字符及任意连续的两个字符（Option::MultiChar）进行编码。在此我们不赘述针对单个字符的编码。当采用针对单字符及任意连续的两个字符进行编码的方式时，你需要首先得到原文本中所有单个字符及任意连续两个字符出现的频率。

例如，假定原文本为 `This(SPC)is(SPC)me\n`（(SPC)为空格），那么你应该得到如下的频率表：

原符号	频率
T	1
h	1

原符号	频率
i	2
s	2
(SPC)	2
m	1
e	1
\n	1
Th	1
hi	1
is	2
s(SPC)	2
(SPC)i	1
(SPC)m	1
me	1
e\n	1

接下来你应该在所有连续的两个字符的组合中，选择出现频率最高的**前三个**字符组合（若有多个组合频率相同，则优先选择字典序较小的组合；若字符组合不足三个，则全部选择），与所有出现的单个字符共同参与哈夫曼树的构建。此外，由于**编码时需要优先匹配字符组合**，故当你选定了频率最高的三个字符组合之后，你需要对应地**更新频率表。**仍以上例进行说明，出现频率最高的三个字符组合应该是`is, s(SPC),(SPC)i`。于是你应该得到如下更新后的频率表：

原符号	频率	说明
T	1	
h	1	
i	**0**	i 均被 is 或 (SPC)i 吸收
s	**0**	s 均被 is 或 s(SPC) 吸收
(SPC)	**0**	(SPC) 均被 (SPC)i 或 s(SPC)吸收
m	1	
e	1	
\n	1	
is	**1**	第二个 is 被 (SPC)i 和 s(SPC)吸收
s(SPC)	**1**	第一个 s(SPC) 被 is 吸收
(SPC)i	1	

最后，你需要**剔除更新后频率表中出现频率为0的符号**，并进行哈夫曼树的构建。

4、压缩文件

与文件压缩相关的函数定义在 utils.h 中，你需要完成的部分如下：

①void loadCodingTable(const std::string &input, std::map<std::string, std::string> &codingTable)： 读取编码表。input 为编码表所在文件的文件名，codingTable 为存储编码表的变量。

②std::string compress(const std::map<std::string, std::string> &codingTable, const std::string &text)： 压缩。codingTable 为前述过程得到的编码表，text 为待压缩的文件内容。该函数的返回值（string）由两部分构成：前八个字节以**小端模式**保存一个数字，为 text 压缩后的**有效比特数**；后续部分则为 text 压缩后的比特流。由于压缩编码后的压缩文件仍以字节流的形式呈现，如果 text 压缩后的比特流不足以转换为字节流（即比特数不是 8 的倍数），则在末尾以二进制 0 进行补足（注意此时补足所用的 0 不算作有效比特）。以下为一个示例（压缩编码方式采用单一字符编码）：

字符流：This(SPC)is(SPC)me\n （(SPC)为空格）

比特流：1011 1111 0001 1100 0011 1010 0111 0101 0

补足：1011 1111 0001 1100 0011 1010 0111 0101 0000 0000 （最后 7 个 0 为补足）

字节流：0xBF 0x1C 0x3A 0x75 0x00

4、输入输出

(1)输入部分

本次 Lab 的输入部分通过文件给出，该文件可以为任意的文本序列。Lab 中用于测试的部分文本文件位于 input 目录下，你也可以自定义一些输入文件用以检测你实现的正确性。

(2)输出部分

本次 Lab 的输出部分通过 main.cpp 中调用 utils.h 中的函数实现，其中编码表保存在 *.sin/mul.huffidx 文件中（sin 表示单一字符编码，mul 表示单一字符和字符组合编码）；压缩文件保存在 *.sin/mul.huffzip 中，为二进制文件，具体格式要求如前文所述。上述输出文件均会保存在 output 目录下。

三、实验评分

1、代码运行

我们提供了 Makefile 工具，在主目录下执行 **make** 指令生成可执行文件 main。你可以通过下述指令运行程序并得到对应的输出文件（位于 output 目录下）：

```
./main 输入文件的路径
```

例如：

```
./main ./input/sample.txt
```

程序运行后，你可以通过使用 **diff** 命令（Linux/Mac OS 在终端可直接运行该命令，Windows 请在 Git Bash 中运行该命令或使用 PowerShell 中的 **Compare-Object** 命令）比较你得到的输出文件和参考输出文件（位于 answer 目录下）的差异，你也可以自行实现一个文件分析程序来帮助你更好地 debug。

2、代码评分

你可以在主目录下执行 **make grade** 指令（Linux/Mac OS 可以直接在 Terminal 中执行，Windows 可以使用 PowerShell（推荐）或 cmd，但请**不要使用 Git Bash**）查看自己实现是否正确，如果不正确，会打印你的输出和答案的差异；如果正确实现，你应该看到如下输出：

```
<<<<<<<< grade test >>>>>>>>
test_a: OK
test_b: OK
test_c: OK
test_d: OK
grade: 100/100
<<<<<<<< grade test over >>>>>>>>
```

实验评分会有额外的数据集，正确通过将得到其余分数，**请勿针对测试数据编程。**

3、书面报告

在上述实验过程中，我们并没有限制哈夫曼树的具体实现方式，因此请你简要描述你的哈夫曼树是如何实现的，这包括但不限于你所定义的成员变量，哈夫曼树的建树过程以及功能函数的实现方式。

本次实验我们分别采用了两种方式进行文本的压缩，你需要至少额外**再自选三组**不同的文本（长度应具有较为显著的差异，但三个文件的大小之和不要超过 1 MB），比较两种压缩方式在这些文本上压缩效果的差异并给出相应的实验数据（包括但不限于原文本的字节数，压缩后文件的字节数，压缩率）。**特别提醒**，本次 Lab 使用的**换行符为\n**（与 Linux/Mac OS 中一致，Windows 中为 \r\n），因此在自定义文本文件时**请确保文本编辑器使用的换行符为 \n**（一般在编辑器的右下角你会看到 `LF` 或 `CRLF`，请选择 `LF`），同时注意**不要通过文本编辑器将测试的输入文件和参考文件的换行符更改为 \r\n。**

显然，本次实验中采用选择频率最高的三个字符组合的方式并不是最优的，请根据你的理解并查阅相关资料，简要描述一种你认为可能更优的压缩策略并给出理由（简述即可，不超过 200 字）。

请将上述内容生成一个 PDF 文件。

你可能需要修改代码添加输出来收集实验数据，由于我们会使用输出判断实现的正确性，请注意保存一份能够通过先前 grade test 的版本。

4、实验上交

如有需要（例如为了更好的模块化），你可以自定义额外的 .h 和 .cpp 文件（需要相应地修改 Makefile）。你**不能**修改 main.cpp 文件，你可以在 utils.h 文件中新增其他函数以帮助你更好地编写代码。对于实验中需要实现的函数，你**不能**修改其声明。**提交实验请将除 main.cpp 以外的源文件（如果你修改了 Makefile 文件也请一并上传）及你自选的三组文本文件打包上传至 Canvas，命名请使用“学号+姓名+lab0”，如“522123456789+张三+lab0.zip”。**