**系统概要设计**

**一．程序总体架构**

该压缩程序包含以下文件：

treenode.h 定义了你将在编码树中使用的节点类型 EncodedTreeNode 结构体。

bits.h 定义了表示 0 或 1 的单一值的 Bit 类型。Bit 可以与整数 0 和 1 互换使用

bits.h 还定义了存储在 Huffman 压缩文件中的数据的 EncodedData 结构体。在 bits.cpp 中，我们提供了读写位流的代码,代码使用了 C++的位操作功能。

main.cpp 包含一个交互式控制台主程序，通过调用你的函数来进行端到端的文件压缩和解压缩。

huffman.cpp 是将实现 Huffman 压缩和解压缩所需功能并添加测试用例的地方

**二 .系统概要设计**

1. 解码/解压缩模块

通过以下函数实现功能：

string decodeText(EncodingTreeNode\* tree, Queue<Bit>& messageBits)

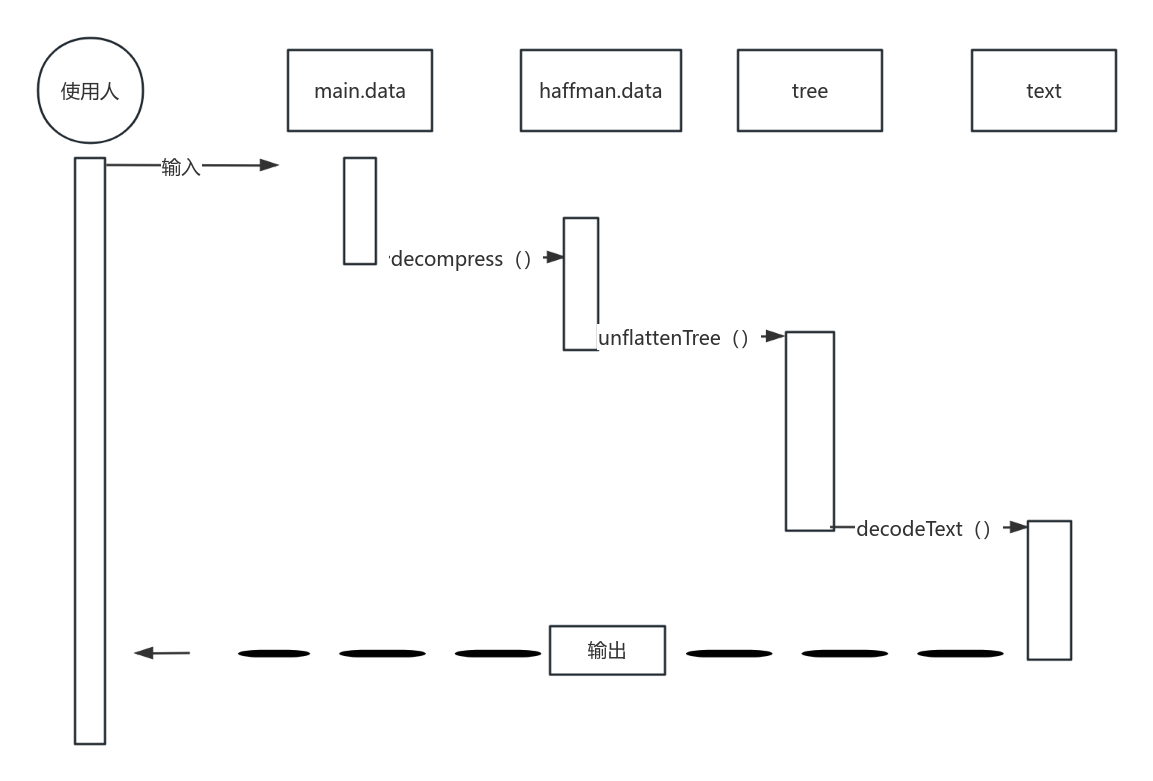
给定一个包含压缩位信息的 Queue与一个用于编码这些位的编码树，将位解码为原来的信息。

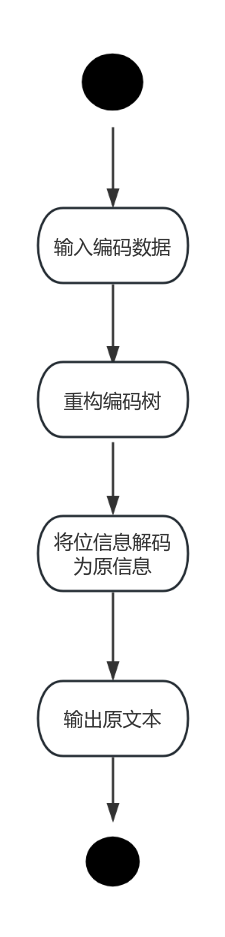
EncodingTreeNode\* unflattenTree(Queue<Bit>& treeShape, Queue<char>& treeLeaves)

根据平铺形式的 Queue与 Queue重构编码树

string decompress(EncodedData& data)

将给定的编码数据解压缩并返回原有的文本。





1. 编码/压缩模块

通过以下函数实现功能：

Queue encodeText(EncodingTreeNode\* tree, string text)

给定一个字符串与一个编码树，用编码树将文本编码并返回一个包括编码位序列的 Queue

void flattenTree(EncodingTreeNode\* tree, Queue& treeShape, Queue& treeLeaves)

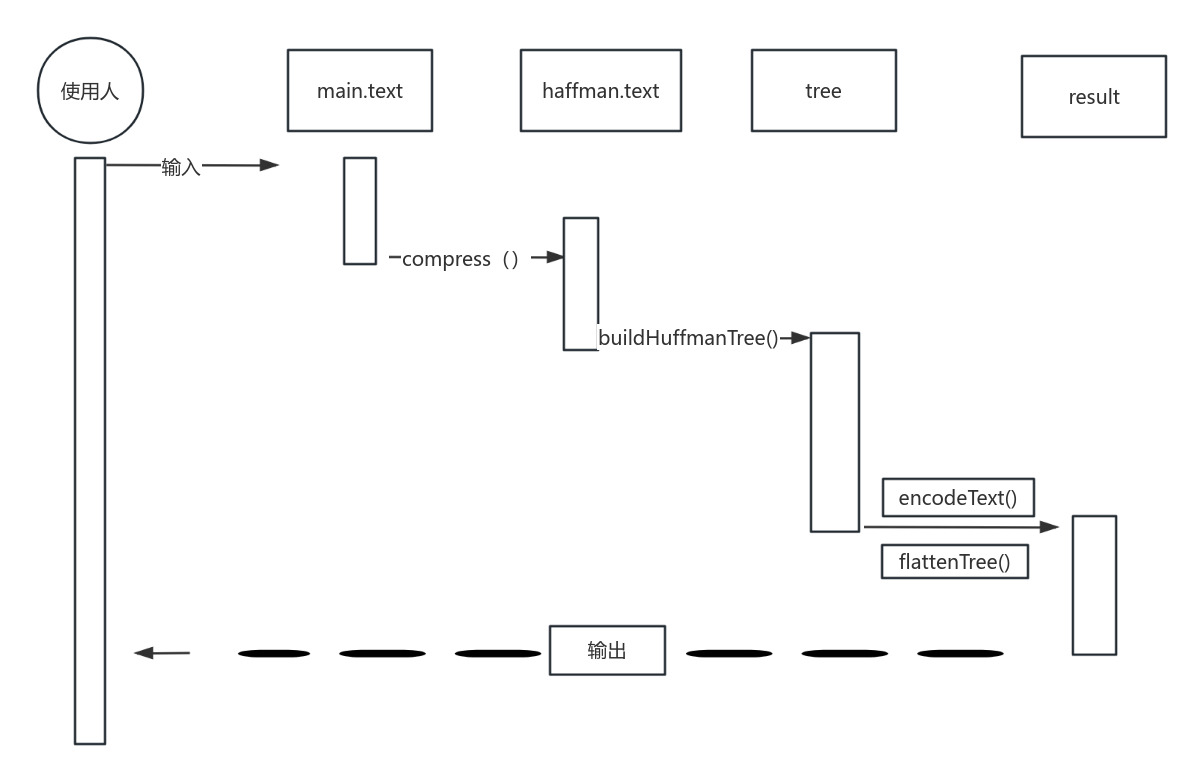
将给定的树平铺为一个 Queue（treeshape）和 Queue（treeleaves），

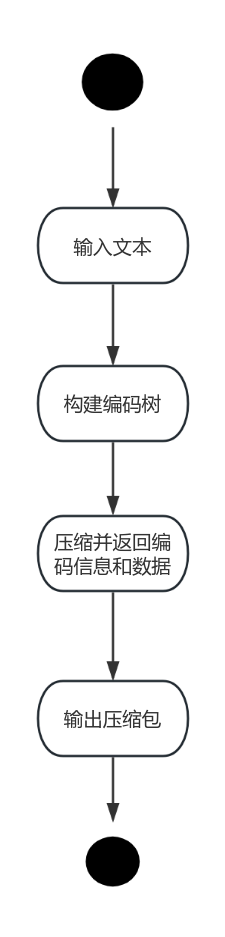
EncodingTreeNode\* buildHuffmanTree(string text) buildHuffmanTree

该函数是整个程序的核心。它接受输入文本并遵循 huffman 算法构建一个针对输入文本的最优 huffman 编码树。

EncodedData compress(string messageText)

将给定的文本用 huffman 编码压缩，产生一个包括编码信息与编码树的编码数据。





运行程序，当主菜单提示运行哪些测试时，选择 0（无）。现在主程序进入一个简单的控制台程序，允许你选择要压缩或解压缩的文件。起始项目包含一个你可以用来试验解压缩功能的压缩文件。

当提示你输入文件名时，指定相对于项目文件夹的名称，例如 res/mystery.jpg.huf（Windows 使用反斜杠 res\mystery.jpg.huf）。未压缩版本将被写入名为 res/unhuf.mystery.jpg 的文件。打开未压缩的文件以确认你成功地还原了原始文件，压缩亦然。