**LZw 编码**

1. **任务说明**：

输入：由集合{a,b,c,d}内字符构成的输入串，输入序列长度 L<=100（至少 30）

处理：先编码，再对编码结果译码

输出：编码结果，译码结果

输入文件:in2.txt，含至少两组输入，每组包含满足要求的串

输出文件:out2.txt，对每组输入的编码和译码结果

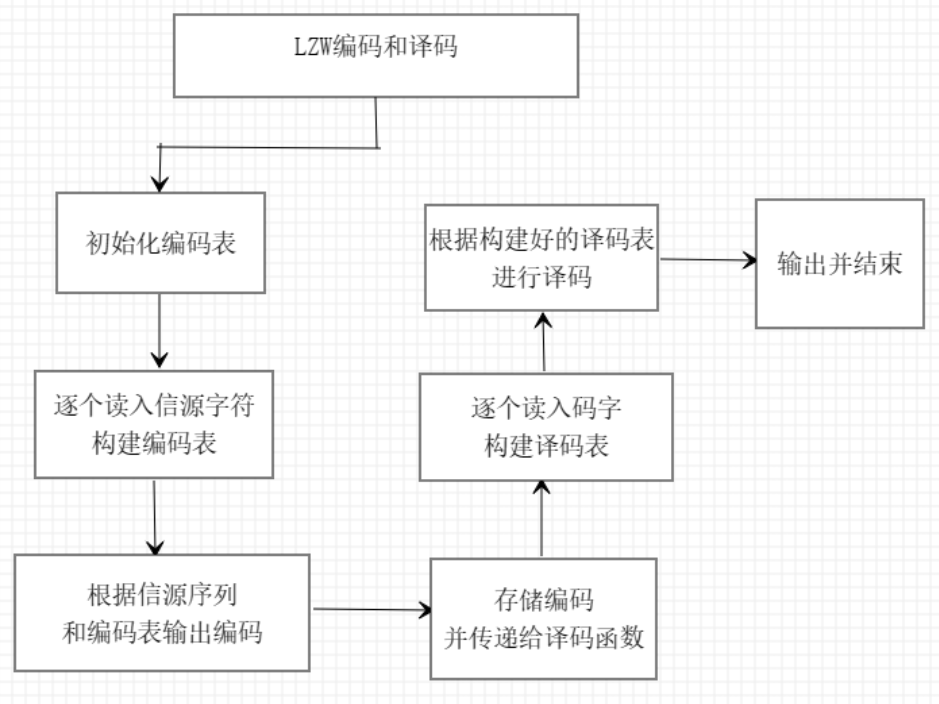
1. **问题分析、实现原理、流程图**
2. 问题分析

LWZ是一种字典码，难点在于编码字典的形成和译码字典的构建，如何保证字典的正确性和译码的正确性。

1. 实现原理

LZW 码的编码原理是：先建立初始化字典，然后将待编码的输入数据流分解成"短语词条"编码器要逐个输人字符，并累积串联成一个字符串，即"短语词条"L。若L是子典中已有的词条．就输人下一个字符 x ，形成新词条Lx。当L 在字典内，而Lx不在字典内时，编码器自先输出指向字典内词条L的指针（即L的相应码字）：再将Lx作为新词条存人学典，并为其确定顺序号；然后把 x 赋值给L，当做新词条的首字符。重复上述过程，直到输人流都处理完为止。

Lzw 的解码器也需首先建立初始化字典：初始化字典或由信源符号集构成（此时，编码器要传送初始字典），或由扩展的 ASCII码构成（此时不用传送初始字典）。解码的第1步，输入第1个码字（即第1个指针），并从字典中取回一个词条L，并将L输出，同时将Lx存入解码字典中。但此时， x 是未知的，x将是下一个从字典中读取的词条的首个字符。再输人下一个指针（即码字），又从字典中取回词条 J ，将其输出，并把J的首字符赋予上一步存人字典的词条 Lx 的 x ，则此时， Lx 已完全确定。重复上述过程，则自动重建了译码表，并将译码输出。

1. 流程图
2. **实现源码**

#pragma once

#include<iostream>

using namespace std;

struct LZW

{

string word;//词条

int NO;//码字

LZW\* lchild,\* rchild;

};

LZW\* InserBST(LZW\* bt, int k,string word) //插入节点数据and创建树

{

if (bt == NULL)

{

bt = new LZW;

bt->NO = k; bt->word = word;

bt->lchild = bt->rchild = NULL;

}

else if (k < bt->NO)

bt->lchild = InserBST(bt->lchild, k,word);

else if (k > bt->NO)

bt->rchild = InserBST(bt->rchild, k,word);

return bt;

}

void InOrder(LZW\* b) //中序遍历递归

{

if (b != NULL)

{

InOrder(b->lchild);

cout << b->NO << " " << b->word << endl;

InOrder(b->rchild);

}

}

LZW\* find\_for\_word(const string& word, LZW\* bt)

{

LZW\* p = bt;

LZW\* f = NULL;//初始化

while (p != NULL && p->word != word) {//往下搜索

f = p;

p = p->rchild;

}

return p;//若数值不存在，则指向NULL结点被返回

}

LZW\* find\_for\_NO(const int& NO, LZW\* bt)

{

LZW\* p = bt;

LZW\* f = NULL;//初始化

while (p != NULL && p->NO != NO) {//往下搜索

f = p;

p = p->rchild;

}

return p;//若数值不存在，则指向NULL结点被返回

}

void Delete(const int& k, LZW\*& p) {

//在p为根的二叉排序树上删除关键字为k的结点

LZW\* s, \* temp;

if (p != NULL)

if (k < p->NO)//还没找到p

Delete(k, p->lchild);//递归地在p的左子树上删除关键字为k的结点

else if (k > p->NO)

Delete(k, p->rchild);//递归地在p的左子树上删除关键字为k的结点

else if (p->lchild != NULL && p->rchild != NULL) {//找到p,但是p的左右子树都不空

//s = Min(p->rchild);

temp = p->rchild;

while (temp->lchild != NULL) {

temp = temp->lchild;

}//找到p的右子树上最小的数s，替换掉p，然后删掉s

s = temp;

p->NO = s->NO;//将p替换成s

Delete(s->NO, p->rchild);//递归地删掉s

}

else {//相等找到,但是左或右为空

temp = p;

if (p->lchild == NULL) p = p->rchild;//左子树空，则将p替换为右子树上第一个结点

else if (p->rchild == NULL) p = p->lchild;//右子树空，则将p替换为左子树上第一个结点

delete temp;

}

}

#include "LZW.h"

#include<iostream>

#include<string>

#include<fstream>

using namespace std;

string Sequence[100], Sequence1[100];//用来存储两个输入序列

LZW\* en\_root,\*de\_root, \* en\_root1, \* de\_root1;//两个序列的编码表和译码表

fstream in("in2.txt");

fstream out("out2.txt");//打开文件

int EN\_word[100],EN\_word1[100]; int en\_num = 0;//数组存储编出来的码字，en\_num为计数器

int enword\_num = 4,deword\_num=4;//记录码字编到多少

void encode(LZW\* bt, string s[], int num)//s为输入的序列数组，num为输入的序列的长度

{

LZW\* p = NULL,\*q=NULL;

int a = 0;//a是信源序列数组的下标指针

string i =s[0];//默认信源序列的第一个词条赋值给i

cout << "编码结果如下：" << endl;

out << "编码结果如下：" << endl;

while (1)

{

p = find\_for\_word(i, bt);//在编码表中找词条为i的元素，如果找不到则返回p=NULL

if (p != NULL)//找到了词条为i的 码字

{

q = p;//将p的地址保存，以便下面输出

a++;

if (a == num)//当编码进行到序列的最后一个时，直接输出

{

out << p->NO<<" ";

cout << p->NO << " ";

EN\_word[en\_num++]=p->NO;

break;

}

i = i + s[a];//i和紧随其后的一个词组合起来，继续搜索

}

else//在编码表中没有搜索到词条为i的码字，现在要创建新的码字

{

out << q->NO<<" ";//先把旧的，已经在编码表中的码字输出

cout << q->NO << " ";

EN\_word[en\_num++] = q->NO;//将码字保存在编码数组中，以便后面译码

enword\_num++;

InserBST(bt, enword\_num, i);//添加到编码表中

i = s[a];//i变为词条的最后一个字母

}

}

out << endl;

cout << endl;

}

void decode(LZW\* bt, int en\_num)//译码

{

int a = 0;//数组下标指针

LZW\* p = NULL,\*q=NULL;

string i;

cout << "译码结果如下：" << endl;

out << "译码结果如下：" << endl;

goon:while (1)

{

p = find\_for\_NO(EN\_word[a], bt);//从编码数组中拿出编好的码字，在译码表中查找

if (p != NULL&&a==0)//第一次译码，由于编码方法的性质，第一次一定能找到

{

out << p->word;

cout << p->word << " ";

i = p->word;

de\_root = InserBST(bt, ++deword\_num, i);//创建新的码字，先把不确定的词条i放入，后面做补充

a++;

goto goon;

}

if(p!=NULL&&a!=0)//第2+次在这里

{

string k;

out << p->word;

cout << p->word << " ";//先将查到的词条输出

i = p->word;

k = i.substr(0, 1);//k取词条的第一个字母，用来补全新码字中不确定的词条

q = find\_for\_NO(deword\_num, de\_root);//找到新码字的位置

q->word = q->word + k;//把k接到最后，完成构建

de\_root = InserBST(bt, ++deword\_num, i);//再次创建新的词条，如此循环

a++;

if (a == en\_num)break;

}

if(p==NULL)

{

cout << "译码错误！"; break;

}

}

out << endl;

cout << endl;

}

int main()

{

cout << "输入需要编码的符号序列，同时用@结尾" << endl;

int h = 0;//信源序列长度

while (1)//将要编码的字符存入到数组S中

{

char cc;

in >> cc;

if (cc == '@')

break;//在以“@”为结尾的时候结束存储

string x(1, cc);

Sequence[h++] = x;

cout << cc;

}

cout << endl;

en\_root = InserBST(en\_root, 1, "a");//初始化译码表和编码表

en\_root = InserBST(en\_root, 2, "b");

en\_root = InserBST(en\_root, 3, "c");

en\_root = InserBST(en\_root, 4, "d");

de\_root = InserBST(de\_root, 1, "a");

de\_root = InserBST(de\_root, 2, "b");

de\_root = InserBST(de\_root, 3, "c");

de\_root = InserBST(de\_root, 4, "d");

encode(en\_root, Sequence, h);

decode(de\_root, en\_num);

cout << endl;

cout << "第二组序列：" << endl;

//第二组序列，先将第一组序列用过的变量初始化

en\_num = 0;

enword\_num = 4; deword\_num = 4;

cout << "输入需要编码的符号序列，同时用@结尾" << endl;

h = 0;//信源序列长度

while (1)//将要编码的字符存入到数组S中

{

char cc;

in >> cc;

if (cc == '@')

break;//在以“@”为结尾的时候结束存储

string x(1, cc);

Sequence1[h++] = x;

cout << cc;

}

cout << endl;

en\_root1 = InserBST(en\_root1, 1, "a");

en\_root1 = InserBST(en\_root1, 2, "b");

en\_root1 = InserBST(en\_root1, 3, "c");

en\_root1 = InserBST(en\_root1, 4, "d");

de\_root1 = InserBST(de\_root1, 1, "a");

de\_root1 = InserBST(de\_root1, 2, "b");

de\_root1 = InserBST(de\_root1, 3, "c");

de\_root1 = InserBST(de\_root1, 4, "d");

encode(en\_root1, Sequence1, h);

decode(de\_root1, en\_num);

//cout << "输出编码表：" << endl;

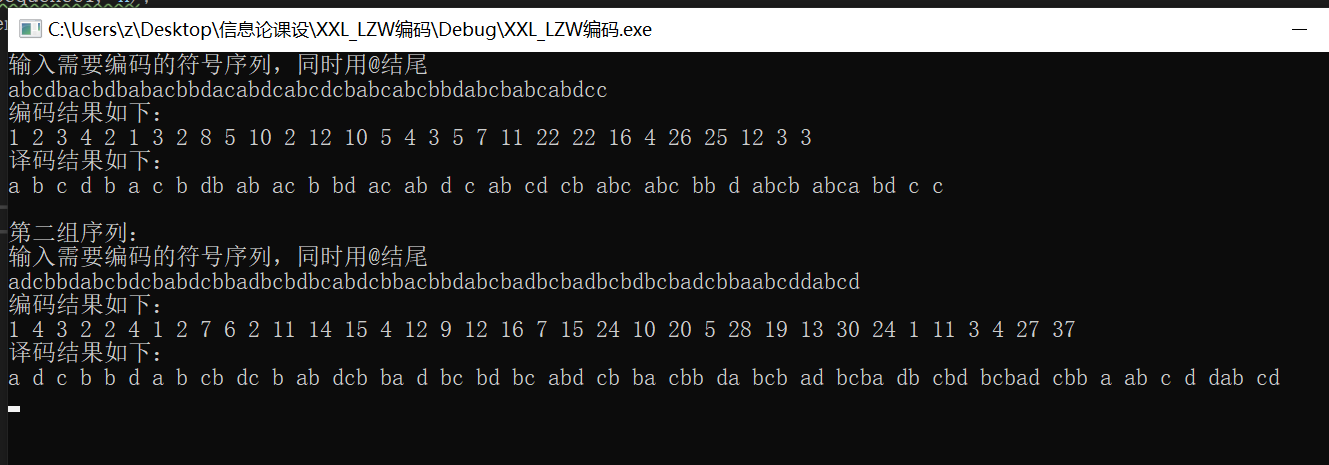
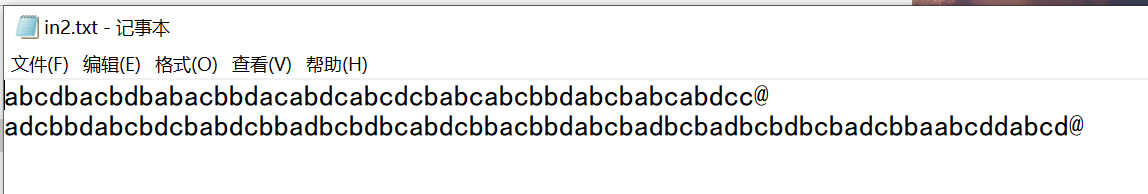
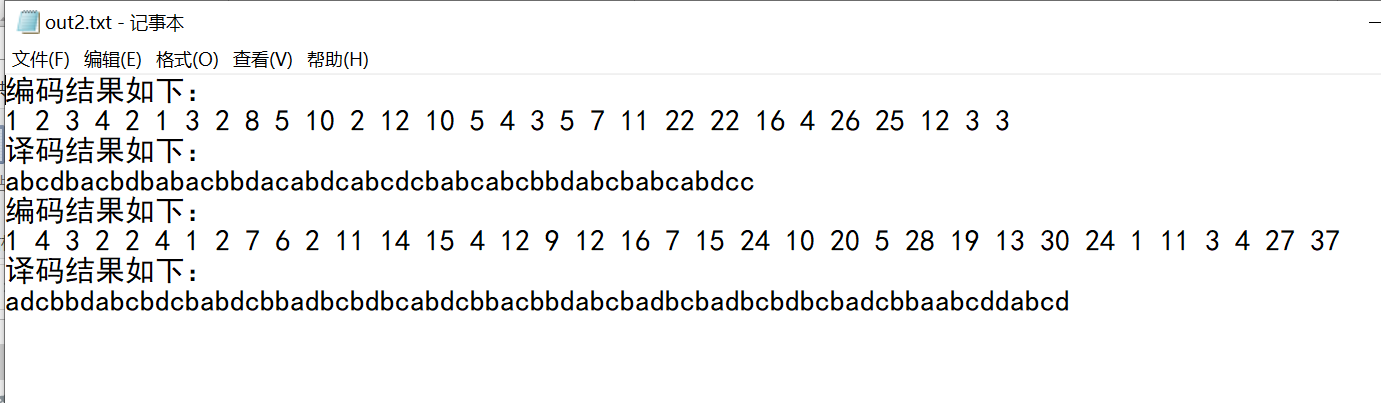
//InOrder(en\_root);

//cout << "输出译码表：" << endl;

//InOrder(de\_root);

return 0;

}

1. **运行结果：输入、输出及结果分析**

输入的两个序列都是随机手打的乱序序列，长度分别为50，71。

经过多次测试，译码结果和编码结果一致，译码时，译码表的最后一个码字和词条是错误的，同时也是无用的。所以并不影响译码的正确性，同时也符合LZW译码的原理，最后一个码字和词条的出现是为了保证倒数第二个码字和词条的正确性，倒数第二个词条是有效的，最后一个词条是无效，无用的，所以，程序仍然是正确的。

1. 设计体会

途中也遇到不少问题，在译码表的构建时，语句执行的顺序和计数器自加的时机尤为重要，谁在先，谁在后，决定了整个程序的正确与否。

在将编好的码字存入到数组中时，最后一个码字还未存入，数组下标指示器已经自加，导致了编码和译码的错误，属于自己编程时不注意的粗心问题，以后多加注意，慢慢改进。

通过这次设计，更加深入的了解了LZW的编码原理，通过编程也了解了编码和译码的具体过程，提升了编程能力。