## 多媒体技术实验报告

姓名： 林瀚洋 学号 220210706 班级： 通信7班

实验日期： 2024.9.23 实验台号：

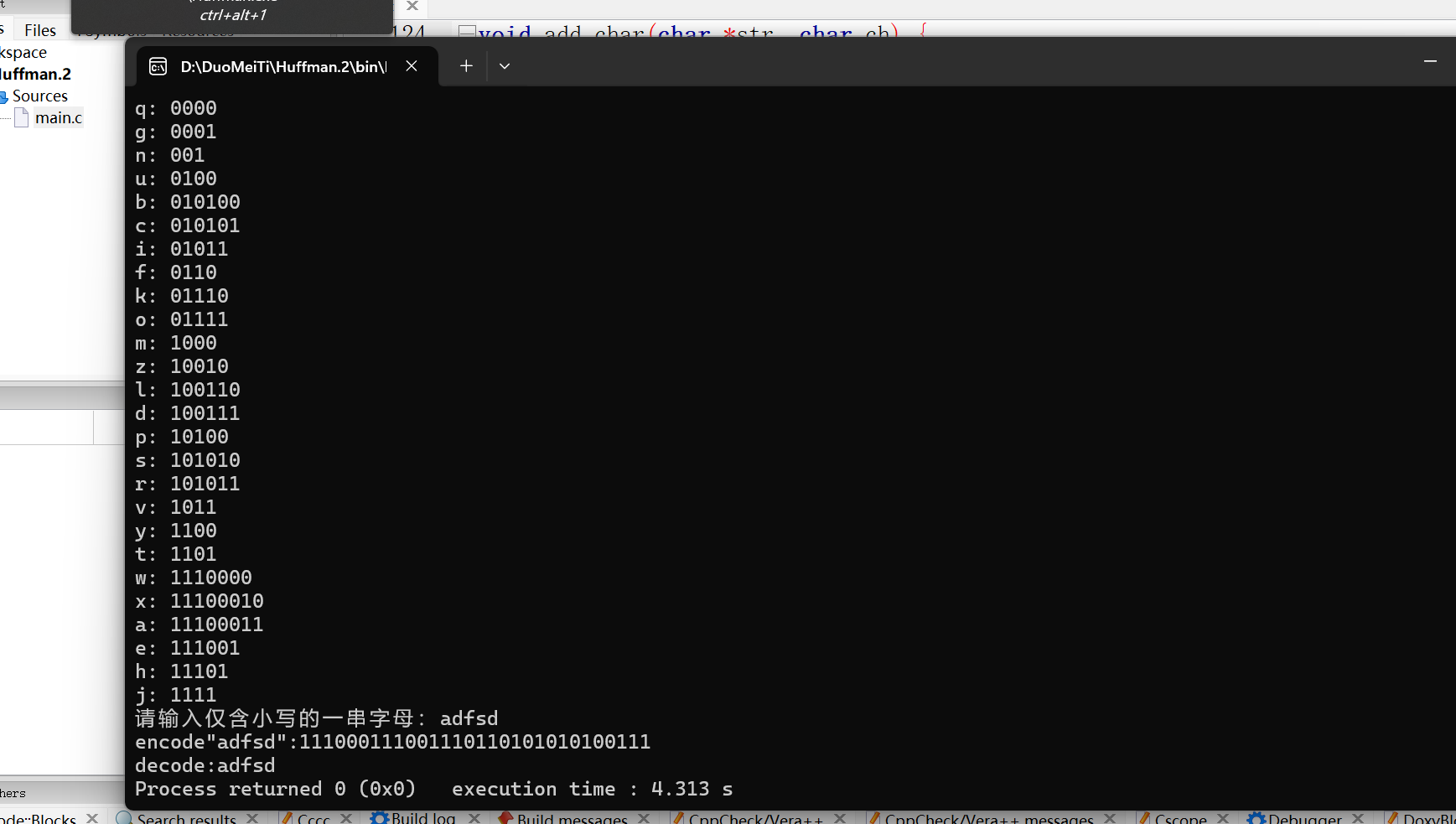
### 霍夫曼编码

原理：利用二叉树，将带有权重（不同出现频率）的字符编变长码，且各字符的编码的前缀不会与其他字符的编码相同，具有唯一性。

C语言实现思路：

1. 为各字符创建节点，节点包含字符和权重。
2. 创建最小二叉树。利用二叉树交换函数，将各节点建成如下的二叉树：最高权重的节点在堆顶，子节点（左右）的权重不大于父节点，叶子节点为各字符节点。
3. 为各字符节点编码。利用递归算法遍历二叉树，从堆顶到叶子节点，每往左就编码0（反之编码1），使得每个叶子节点都有前缀不同的唯一编码。

实验结果：



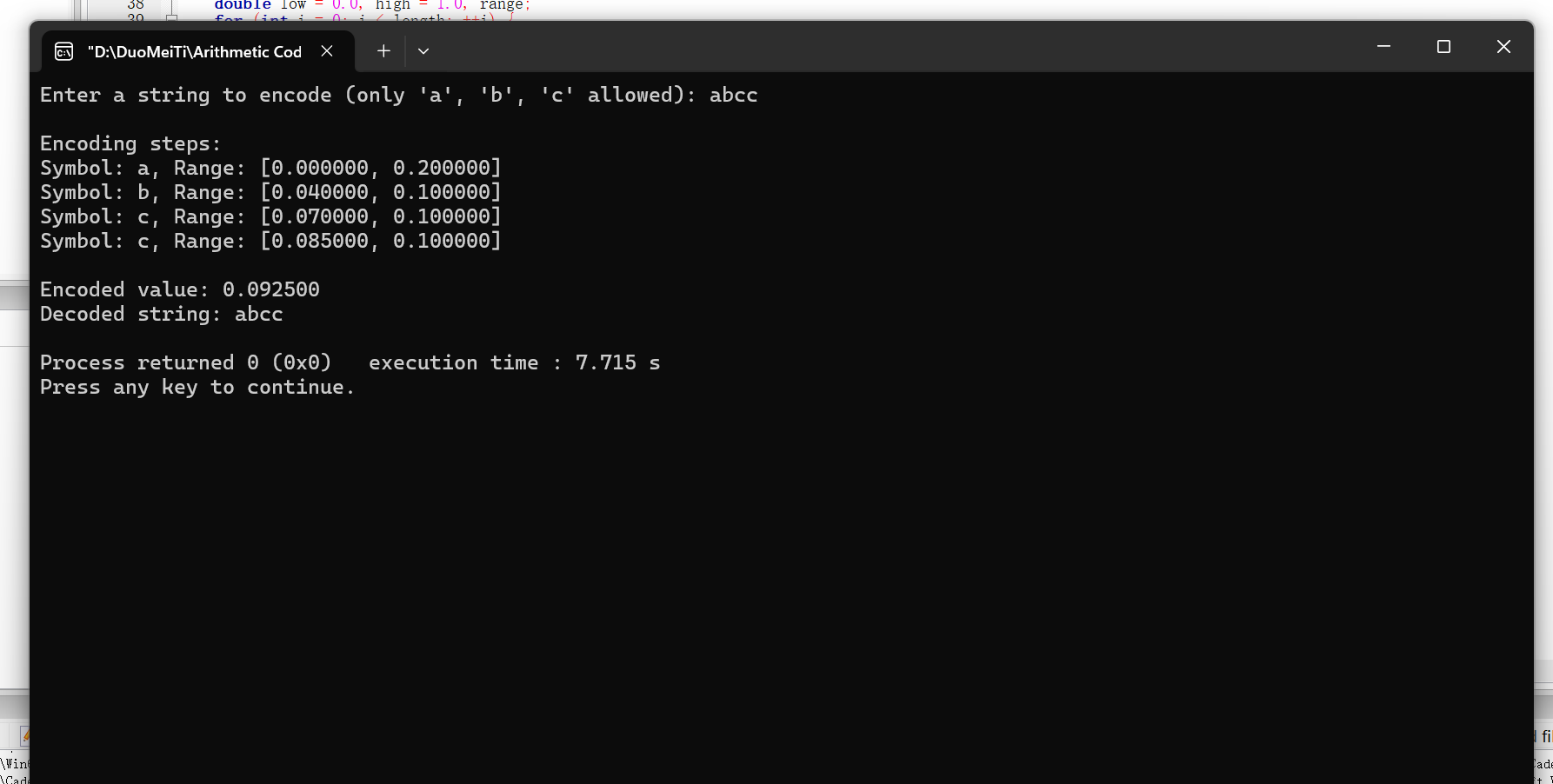
代码见文件尾

### 算术编码

实验原理：将不同字符划分在 [0,1] 不同的区间，每编码一个字符都在前一串字符编码区间基础上，划分该字符对应子区间，最终取编码完成的区间的区间中值作为该字符串的编码。

C语言实现思路：

1. 将各字符根据权重划分到[0,1]的不同区间。
2. 对输入字符串进行循环编码：一步一步取子区间
3. 输出最终子区间中间值作为编码

实验结果：

代码见文件尾

### 3.代码

#### 霍夫曼编码及译码

*#include <stdio.h>*  
*#include <stdlib.h>*  
*#include <string.h>*  
  
*#define MAX\_TREE\_NODES 256*  
  
typedef struct Code{  
    char character;  
    char huffcode[MAX\_TREE\_NODES];  
}Code;  
  
typedef struct Node {  
    char character;  
    int frequency;  
    struct Node \*left, \*right;  
} Node;  
  
Node\* createNode(char character, int frequency) {  
    Node\* newNode = (Node\*)malloc(sizeof(Node));  
    newNode->character = character;  
    newNode->frequency = frequency;  
    newNode->left = newNode->right = NULL;  
    return newNode;  
}  
  
typedef struct {  
    char chara[MAX\_TREE\_NODES];  
  
  
}Dictionary;  
//最小堆，用于存储二叉树成员，size是最小堆当前的成员数量  
typedef struct {  
    Node\* nodes[MAX\_TREE\_NODES];  
    int size;  
} MinHeap;  
  
//功能：插入节点，插入地址是最小堆的末尾  
//输入参数：最小堆的指针，待插入节点的指针  
void insertHeap(MinHeap\* heap, Node\* node) {  
    heap->nodes[heap->size++] = node;  
}  
  
//功能：找出最小堆中“权”最小（即频率最小）的节点，并将堆尾元素替换掉最小权节点，并删除堆尾节点  
//输入参数：最小堆指针  
//输出参数：返回最小节点  
Node\* extractMin(MinHeap\* heap) {  
    int minIndex = 0;  
    for (int i = 1; i < heap->size; i++) {  
        if (heap->nodes[i]->frequency < heap->nodes[minIndex]->frequency) {  
            minIndex = i;  
        }  
    }  
    Node\* minNode = heap->nodes[minIndex];  
    heap->nodes[minIndex] = heap->nodes[--heap->size]; // Replace with last node  
    return minNode;  
}  
  
//如果是叶子节点，没有子节点，则返回0  
int isLeaf(Node\* node) {  
    return !(node->left) && !(node->right);  
}  
  
void buildHuffmanTree(char characters[], int frequencies[], int size, Node\*\* root) {  
    MinHeap heap = { .size = 0 };  
  
    //将所有字符生成一个堆，每个节点没有子节点  
    for (int i = 0; i < size; i++) {  
        insertHeap(&heap, createNode(characters[i], frequencies[i]));  
    }  
  
    //将最小的两个节点抽出（size-2)，作为新节点的子节点，然后将这个新节点放到最小堆的堆尾(size+1)  
    //重复这个操作，直到堆中只剩一个节点，此时这个节点即为最小堆的堆顶（root）  
    while (heap.size > 1) {  
        Node\* left = extractMin(&heap);  
        Node\* right = extractMin(&heap);  
        Node\* newNode = createNode('\0', left->frequency + right->frequency);  
        newNode->left = left;  
        newNode->right = right;  
        insertHeap(&heap, newNode);  
    }  
  
    \*root = extractMin(&heap);  
}  
  
//打印霍夫曼码  
//通过buildHuffmanTree生成的堆顶节点，利用递归的方法，将最小堆的叶子节点的值及其对应字符打印出来  
void printCodes(Node\* root, char\* code, int top,Code\* Huffman) {  
    static int index = 0;  
    if (root->left) {  
        code[top] = '0';  
        printCodes(root->left, code, top + 1,Huffman);  
    }  
  
    if (root->right) {  
        code[top] = '1';  
        printCodes(root->right, code, top + 1,Huffman);  
    }  
  
    if (isLeaf(root)) {  
        code[top] = '\0';  
        printf("%c: %s\n", root->character, code);  
        Huffman[index].character = root->character;  
        strncpy(Huffman[index++].huffcode, code, sizeof(Huffman[index++].huffcode) - 1);  
        }  
}  
  
void encode(Code\* Huffman, char\* sentence, int size,char\* coding\_result)  
{  
     strcpy(coding\_result, ""); // 将 coding\_result 重置为空字符串  
    int i=0;  
//printf("%d",size);  
    for(i=0;i<size;i++)  
    {  
        int j=0;  
        for(j=0;j<26;j++)  
        {  
            if(sentence[i] == Huffman[j].character)  
                strcat(coding\_result, Huffman[j].huffcode);  
        }  
    }  
    coding\_result += '\0';  
}  
  
void add\_char(char \*str, char ch) {  
    size\_t len = strlen(str);  
    if (len < sizeof(str) - 1) {  // 确保不会超出数组界限  
        str[len] = ch;  
        str[len + 1] = '\0';  // 确保字符串以 null 字符结尾  
    }  
}  
  
void decode(char\* decode\_sentence,Node\* root,char\* encode\_sentense)  
{  
       Node\* current = root;  
    decode\_sentence[0] = '\0'; // 将 decode\_sentence 重置为空字符串  
    for (int i = 0; encode\_sentense[i] != '\0'; i++) {  
        if (encode\_sentense[i] == '0') {  
            current = current->left;  
        } else {  
            current = current->right;  
        }  
  
        // 如果到达叶子节点，打印字符并返回到根节点  
        if (current->left == NULL && current->right == NULL) {  
            add\_char(decode\_sentence,current->character);  
            current = root; // 返回到根节点  
        }  
    }  
}  
//  
int main() {  
    char characters[] = { 'a', 'b', 'c', 'd', 'e', 'f' ,'g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'};  
    int frequencies[] = { 5, 9, 12, 13, 16, 45 ,43,33,23,68,23,12,46,82,23,25,34,14,13,64,43,59,8,3,63,25};  
    int size = sizeof(characters) / sizeof(characters[0]);  
    Code Huffman[size];  
     char sentense[MAX\_TREE\_NODES];  
    char encode\_sentense[MAX\_TREE\_NODES] = {};  
    char decode\_sentense[MAX\_TREE\_NODES] = {};  
    Node\* root = NULL;  
    buildHuffmanTree(characters, frequencies, size, &root);  
  
    char code[MAX\_TREE\_NODES];  
    printCodes(root, code, 0,Huffman);  
  
    printf("请输入仅含小写的一串字母：");  
    scanf("%s",sentense);  
  
    encode(Huffman, sentense, strlen(sentense), encode\_sentense);  
    printf("encode\"%s\":%s\n", sentense, encode\_sentense);  
  
    decode(decode\_sentense, root, encode\_sentense);  
    printf("decode:%s", decode\_sentense);  
    return 0;  
}

#### 算术编码及译码

*#include <stdio.h>*  
*#include <string.h>*  
  
*#define MAX\_SYMBOLS 256*  
  
typedef struct {  
    char symbol;  
    double probability;  
    double cumulativeProbability;  
} Symbol;  
  
void calculateCumulativeProbabilities(Symbol symbols[], int count) {  
    double cumulative = 0.0;  
    **for** (int i = 0; i < count; ++i) {  
        symbols[i].cumulativeProbability = cumulative;  
        cumulative += symbols[i].probability;  
    }  
}  
  
double encode(const char \*input, Symbol symbols[], int count) {  
    double low = 0.0, high = 1.0, range;  
    printf("\nEncoding steps:\n");  
    **for** (const char \*p = input; \*p; ++p) {  
        **for** (int i = 0; i < count; ++i) {  
            **if** (symbols[i].symbol == \*p) {  
                range = high - low;  
                high = low + range \* (symbols[i].cumulativeProbability + symbols[i].probability);  
                low = low + range \* symbols[i].cumulativeProbability;  
                printf("Symbol: %c, Range: [%lf, %lf]\n", \*p, low, high);  
                break;  
            }  
        }  
    }  
    return (high + low) / 2;  
}  
  
void decode(double code, int length, Symbol symbols[], int count, char \*output) {  
    double low = 0.0, high = 1.0, range;  
    **for** (int i = 0; i < length; ++i) {  
        range = high - low;  
        double value = (code - low) / range;  
        **for** (int j = 0; j < count; ++j) {  
            **if** (value >= symbols[j].cumulativeProbability && value < symbols[j].cumulativeProbability + symbols[j].probability) {  
                output[i] = symbols[j].symbol;  
                high = low + range \* (symbols[j].cumulativeProbability + symbols[j].probability);  
                low = low + range \* symbols[j].cumulativeProbability;  
                break;  
            }  
        }  
    }  
    output[length] = '\0';  
}  
  
int **main**() {  
    Symbol symbols[] = {  
        {'a', 0.2}, {'b', 0.3}, {'c', 0.5}  
    };  
    int count = sizeof(symbols) / sizeof(symbols[0]);  
  
    calculateCumulativeProbabilities(symbols, count);  
  
    char input[100];  
    printf("Enter a string to encode (only 'a', 'b', 'c' allowed): ");  
    scanf("%s", input);  
  
    double encoded = encode(input, symbols, count);  
    printf("\nEncoded value: %lf\n", encoded);  
  
    char decoded[100];  
    decode(encoded, strlen(input), symbols, count, decoded);  
    printf("Decoded string: %s\n", decoded);  
  
    return 0;  
}