《算法与数据结构》课程设计
指导手册

2024 年 6 月

目录

前言.....2

实验一、英文课本词汇表的生成.....3

实验二、使用动态规划算法完成文献查重.....12

实验三、使用哈夫曼算法完成文本压缩.....25

前言

本实验手册的目的是加强学生的算法设计能力，提高学生运用算法的能力，增强学生编程实践能力。

本实验教程的前序课程为《高级语言程序设计》、《数据结构》、《计算机算法》。本教程默认学生已具备基本的编程能力，完成了算法课程的学习。教程所安排的均为综合实验，需要综合运用算法设计能力、数据结构设计能力、程序编写能力。教程编撰者默认学生已经在《高级语言程序设计》、《数据结构》、《计算机算法》等课程中完成了基础实验。

实验一　英文课本词汇表的生成

【问题背景】

英文课本最后通常附有一张词汇表，它记录了课本中出现的所有单词。当今网络资源非常丰富，网络上有大量的英文短文，生成词汇表的需求也在增加。你的客户向你提出一个需求，他需要为大量的英文短文生成相应的词汇表。由于英文短文的数量较多，他希望这个任务可以由程序自动完成。

【问题描述】

给定一段短文：

May I have your attention please? I have a dream. You may have my book.

本任务需要你通过编程读入这段文本，然后输出两张词汇表。其中一张是按**字母排序**的词汇统计表，另外一张是按**单词频率降序排序**的词汇表，两张表格均需附上每个单词的频率。上述短文对应的词汇表如下所示：

表 1(按字母排序)

a	**1**
attention	**1**
book	**1**
dream	**1**
have	**3**
i	**2**
may	**2**
my	**1**
please	**1**
you	**1**
your	**1**

表 2（按词频排序）

have	**3**
i	**2**
may	**2**
a	**1**
attention	**1**
book	**1**
dream	**1**
my	**1**
please	**1**
you	**1**
your	**1**

在完成该任务时，我们做如下几个约定（使得任务相对简化）：

1、**输入文件为 Input.txt**，里面的内容为一段英文课文。

2、为了便于统计，**名词的单复数算两个不同的词，动词的时态不同也算两个不同的词**。譬如 book 与 books，do 和 does 都算不同的词。

3、**连词算两个词**，譬如 let's 算 let 和 s，分开统计。

4、**单词与单词的分割为非字母的字符**(不属于 a-z,也不属于 A-Z)。

5、排序的部分要求使用**快速排序或合并排序算法**，因为这两个算法是本任务的实验内容之一。

【实验目的】

1、如何将一个复杂任务分解为多个较为简单的子模块；

2、如何对各个子模块设计相应的算法；

3、如何综合全局，设计便于完成算法的数据结构。

4、熟悉快速排序与合并排序算法。

【任务 1】

请将本任务分解为多个便于实现的子模块。（模块可以理解工作量较小相对独立的小任务，模块之间先做什么，再做什么，模块之间是顺序结构，还是循环还是什么）

注意：为了便于同学们编程和参考，各任务的答案在后页都有给出。但请同学们务必先思考和完成任务再看答案，否则对你的编程思维训练毫无益处，你的编程能力也无法提高。

【任务 1 参考答案】

第一步，把 txt 文件的内容**读入**到内存中；

第二步，把读入的内容**分割**为**单词**，并将单词**存入**相应的**数据结构**中（内存）；

第三步，使用排序算法对单词或者词频进行**排序**，产生输出表格的内容；

第四步，按要求**输出**两张表格；

因此，本实验有 5 个关键环节：

读写文件、单词分割、单词存储、单词排序、表格输出。

【任务 2】

请为各步骤设计大致的算法框架（不需要特别具体，有算法的解决思路即可）。这个步骤类似于数学中解题思路的酝酿过程。

【任务 2】

各步骤的算法框架：

读写文件：逻辑简单，无需算法；

单词分割：单词**开始**，当前字符为 a-z, 或者 A-Z，而前一字符不是 a-z 或 A-Z；

单词**延续**，当前字符为 a-z, 或者 A-Z，而前一字符也是 a-z 或 A-Z；

单词**结束**，当前字符为 a-z, 或者 A-Z，而后一字符不是 a-z 或 A-Z；

找到单词的**开始和结束**就可以完成**单词分割**；

单词存储：分割出的单词要判断是不是已经存储到数据结构中。如果已经存储过，只需要在该词的频率上+1；如果未存入数据结构，需要分配相应的存储空间，该词的频率记为 1；

单词排序：使用合并排序或快速排序，这两个算法在《计算机算法》课程中学习过，具体算法请参考课本。

表格输出：逻辑简单，无需算法，只需注意输出格式。

注：比较你给出的算法框架和参考答案，分析各自的优缺点。

【任务 3】

请为上述各步骤设计所需的数据结构（需要非常具体，因为已经接近程序编写）

【任务 3 参考答案】

读写文件：**读入的英文课文存为数组？string？或其它数据结构？（选择自己认为合理的，自己更熟练运用的）**

单词分割：单词开始，当前字符为 a-z, 或者 A-Z，而前一字符不是 a-z 或 A-Z；

单词延续，当前字符为 a-z, 或者 A-Z，而前一字符也是 a-z 或 A-Z；

单词结束，当前字符为 a-z, 或者 A-Z，而后一字符不是 a-z 或 A-Z；

找到单词的开始和结束就可以完成单词分割；

分割好的单词用什么数据结构来存储呢？我觉得数组比较麻烦，用 string 更方便一些（数据结构的选择标准是如何方便编程，减少代码量）。

单词存储：分割出的单词要判断是不是已经存储过，如果已经存储过，只需要在该词的频率上+1；如果未存入数据结构，需要分配相应的存储空间，该词的词率记为 1；

此步骤涉及到本任务核心的数据结构，即用什么数据结构来存储词汇表？

建议采用结构体(对象)，该结构体包含一个 string 表示单词拼写的字符串；一个整形变量，表示该单词的频率。

单词排序：使用合并排序或者快速排序，这两个算法在《计算机算法》课程中学习过，请参考课本。

此步骤主要涉及算法，不涉及到数据结构和存储；

表格输出：此步骤不涉及到数据结构。

【任务 4】

核心算法的具体化：请把任务 2 中的算法框架具体化，并写出各算法的伪代码

【任务 4.1 参考答案----单词分割】

注：以下代码均为伪代码，侧重代码的逻辑结构，未必符合程序语言的语法。请参照伪代码的逻辑，编写相应的程序代码。

单词分割：单词**开始**，当前字符为 a-z, 或者 A-Z，而前一字符不是 a-z 或 A-Z；

单词**延续**，当前字符为 a-z, 或者 A-Z，而前一字符也是 a-z 或 A-Z；

单词**结束**，当前字符为 a-z, 或者 A-Z，而后一字符不是 a-z 或 A-Z；

找到单词的**开始和结束**就可以完成**单词分割**；

```
String   text;   //读入的文本存入 string text 中
String   word="";   //被分割出的单词用 word 表示

If((text[k]>='a')&&( text[k]<='z'))||((text[k]>='A')&&(text[k]<='Z'))
//用中文表示上面语句的含义_____
{
      Word+=text[k];   //用中文表示本语句的含义_____
}

If((text[k]<'A')||((text[k]>'Z')&&(text[k]<'a'))||(text[k]>'z'))
//用中文表示上面语句的含义_____
{
      If(Word!="")      //用中文表示本语句的含义_____
      {
            将 word 存入数据结构；
            Word="";   //用中文表示本语句的含义_____
       }
  }
```

【任务 4.2 参考答案----单词存储】

单词存储：分割出的单词要判断是不是已经存储到数据结构中。如果已经存储过，只需要在该词的频率上+1；如果未存入数据结构，需要分配相应的存储空间，该词的频率记为 1；

```
Struct Unit{
    String   sequence;
    int      freq;
};      //用于存储词汇表, sequence 表示单词, freq 表示该单词的频率

Unit verb[10000];    //10000 表示很大的一个值，单词的个数不会超过这个值
int   numVerb=0;    //该变量表示不同单词的数量
int   newWord=1;    //该单词用于标记一个单词是否收录进词汇表中

For(int i=0;i< numVerb;i++)
{
    If(word==verb[i].sequence)    //用中文表示本语句的含义_____
    {
        Verb[i].freq++;
        newWord=0;          //用中文表示本语句的含义_____
    }
}
If(newWord==1)                //用中文表示本语句的含义_____
{
    verb[numVerb].sequence=word;   verb[numVerb].freq=1;   numVerb++;
}     //将 word 存入词汇表中，单词的数量+1
```

【任务 4.3 参考答案----单词排序】

单词排序：合并排序或者快速排序算法**二选一**，完成词汇表的排序。使用 verb.sequence 排序生成按字母排序的词汇表；使用 verb.freq 排序生成按词频排序的词汇表。请参照课本中的算法，将算法的伪代码写在下面。

【任务 5】

将任务 4 中的算法转化为具体的程序代码，请你用 C/C++语言实现本任务。

【总结】

在完成本项任务时，我们的解答过程分为以下步骤：

第一，将任务分解为便于实现的多个子模块；

第二，为各步骤（子模块）设计大致的算法框架；

第三，为各步骤（子模块）设计所需的数据结构（需要非常具体）；

第四，将上述的算法框架具体化，并写出各算法的伪代码；

第五，将上述的具体算法转化为程序代码。

实验二　使用动态规划算法完成文献查重

【问题背景】

当今网络资源极其丰富，有一部分人在写论文时，暴力的粘贴网络文献中的句子。一方面不利于知识产权的保护，另一方面导致论文质量下降甚至完全无意义。因此文献查重是当今学术界的一个重要课题，对净化学术环境有重要益处。本实验试图用动态规划算法完成文献的查重。一方面可增强学生对动态规划算法的理解能力，另一方面也向学生演示了动态规划算法的用处。

【问题描述】

在某网页上有一段关于机器学习的描述：

Machine learning is a method of data analysis that automates analytical model building. Using algorithms that iteratively learn from data, machine learning allows computers to find hidden insights without being explicitly programmed where to look.

假设某人在写论文时写了这样一句话 “Machine learning allows computers to identify hidden insights and patterns without being explicitly programmed where to look。”使用动态规划算法对这句话与网页的文本进行比对，结果如下：

论文： machine learning allows computers to identify hidden insights

网页： machine learning allows computers to find hidden insights

论文： and patterns without being explicitly programmed where to look

网页： - - without being explicitly programmed where to look

图 1

黄色的部分为比对上的文字。我们发现论文中的该句话与网页内容中句子的相似度高达 14/17=82.4%（论文句子有 17 个单词，有 14 个被比对中）。因此，我们可以使用动态规划算法来完成查重的任务。

在完成该任务时，我们做如下几个约定：

1、输入文件为（**被用于查重的文本**）text.txt，里面的内容为一段英文文本。该文本将面向一个库文件查重，库文件为 lib.txt, 内容也是英文文本。

2、为了便于完成查重任务，我们约定查重算法的执行过程约定如下：顺序取出 text.txt 中的一句话，与 lib.txt 中的每句话分别比对，并保存最高的相似度。譬如 text.txt 中的某句话与 lib.txt 中的第一句话的相似度为 20%，与 lib.txt 中的第二句话的相似度为 35%，则将 35%保存下来作为该句话与 lib.txt 的相似度（若 lib.txt 有 *n* 句话，则保留 *n* 个相似度中最高的一句）。如果最高的相似度超过 **40%**，则认为 text.txt 中的这句话与 lib.txt 中的文本**重复**了。

3、约定一句话的终止符号为‘。’，‘？’，‘！’。查重的时候只比对文字，不比对标点符号。为了方便程序处理，text.txt 与 lib.txt 只包含‘。’,‘？’,‘!’ 和 ‘ ,’。

4、文本比对时的计分方式如下：如果单词相同，得分+1；如果单词不同，得分 0；如果单词比对上插入的空格，得分 0。以上页所示的例子，有 14 个单词比对相同（黄色部分），有两个单词比对上空格（and、patterns），还有一组单词不同（identify、find），**比对得分**为 14。用得分除以来自 text.txt 中的这句话中的单词数（14/17=82.4%），这个值用于表示这句话与 lib.txt 中文本的相似度。

5、为了方便编程，约定：同一个词的不同时态算两个不同的词，同一个词的单复数算两个不同的词，但同一个词的大小写形式算一个词。譬如 go 和 goes 算两个不同的词，book 和 books 算两个不同的词，book 和 BOOK 算相同的词。

6、为 text.txt 的每个句子寻找到 lib.txt 文件中与之最相似的句子，如果它们之间的相似度≥40%，**按图 1 中句子对齐的方式，输出这两个句子（相同的单词的字符上下对齐，比对不上的部分用空格或短线代替，请确保相同单词上下字符对齐）。**

【实验目的】

1、如何将一个复杂任务分解为多个较为简单的子模块；

2、如何对各个子模块设计相应的算法；

3、如何综合全局，设计便于完成算法的数据结构。

4、熟悉并使用动态规划算法编程。

【任务 1】

请将本任务分解为多个便于实现的子模块。（模块可以理解工作量较小相对独立的小任务，模块之间先做什么，再做什么，模块之间是顺序结构，还是循环还是什么）

注意：为了便于同学们编程和参考，各任务的答案在后页都有给出。但请同学们务必先思考和完成任务再看答案，否则对你的编程思维训练毫无益处，你的编程能力也无法提高。

【任务 1 参考答案】

第一步，**读取** lib.txt 文件，并将 lib.txt 中的文本按“**逐句**”的方式存储；

第二步，把 text.txt 文件的内容**读入**到内存中，按顺序方式取出 text.txt 中的一句话，并执行第三步；

第三步，完成取出的这句话与 lib.txt 中每句话的文字比对，并记录最高的相似度；然后回到第二步（即取出下一句话，继续与 lib.txt 比对）。

第四步，汇总比对结果，计算 text.txt 与 lib.txt 的相似度。

任务的逻辑流程

****

读取 lib.txt，“逐句”存储 lib.txt 中的文本；

While(text.txt 没有读完)

{

If(当前字符是英文字母) 将单词存入临时句子 TempSentence;

Else if (当前字符是‘。’，‘？’，‘！’)

{

比对 TempSentence 与 lib.txt 中的每一句，并记录最高的相似度；

//与每一句比对通过 For 循环实现

}

}

汇总比对结果；

****

【任务 2】

请为各步骤设计大致的算法框架（不需要特别具体，有算法的解决思路即可）。这个步骤类似于数学中解题思路的酝酿过程。

【任务 2】

各步骤的算法框架：

步骤一：读取 lib.txt，“逐句”存储 lib.txt 中的文本；

算法框架：如果当前字符是‘。’，‘？’，‘！’，当前句结束，新开为空的一句；

如果当前字符是英文字母，把该字符存入当前句；

如果不是以上两种情况，只需要在单词之间存储一个空格即可；

步骤二：把 text.txt 文件的内容**读入**到内存中，按顺序方式取出 text.txt 中的一句话。

算法框架：如果当前字符是‘。’，‘？’，‘！’，此句结束；

如果当前字符是英文字母，把该字符存入此句；

如果不是以上两种情况，只需要在单词之间存储一个空格即可；

步骤三：比对 TempSentence 与 lib.txt 中的每一句，采用动态规划算法；

算法框架：分别计算 TempSentence 与来自 lib.txt 文件的一句话 libSentence 各有几个单词(假设分别为 m，n 个单词)。建立一个 m*n 的矩阵 S，表示单词是否相同；若 TempSentence 的第 i 个单词与 libSentence 的第 j 个单词相同，则 S(i,j)=1;否则 S(i,j)=-1；采用标准的动态规划算法完成 libSentence 与 TempSentence 的比对，其中要使用 S(i,j)矩阵。

步骤四：汇总比对结果。

此步骤只涉及简单计算，无复杂的算法。

注：比较你给出的算法框架和参考答案，分析各自的优缺点。

你可以提出并使用自己的算法框架，只要逻辑上成立，算法合理。

【任务 3】

请为上述各步骤设计所需的数据结构（需要非常具体，因为已经接近程序编写）

【任务 3 参考答案】

步骤一：读取 lib.txt，“逐句”存储 lib.txt 中的文本；

可以用一个结构体存储每个句子，包含一个 string 和一个整形变量。String 用于存储句子的内容，单词之间用空格连接；用整形变量存储句子中单词的数量。

步骤二：把 text.txt 文件的内容**读入**到内存中，按顺序方式取出 text.txt 中的一句话。

可以用一个结构体存储读取的当前句子，包含一个 string 和一个整形变量。String 用于存储句子的内容，单词之间用空格连接；用整形变量存储句子中单词的数量。

步骤三：比对 TempSentence 与 lib.txt 中的每一句，采用动态规划算法；

需要两个 m*n 的矩阵，一个用于表示单词之间的异同（在算法框架部分已经有描述）；另一个用于动态规划算法，计算两个句子的相似度得分。

步骤四：汇总比对结果。

需要把 text.txt 中每句话与 lib.txt 的相似度以及 text.txt 中每句话的单词数存入一个数组中。然后才能汇总计算。

【任务 4】

核心算法的具体化：请把任务 2 中的算法框架具体化，并写出各算法的伪代码

【任务 4.1 参考答案----步骤一】

注：以下代码均为伪代码，侧重代码的逻辑结构，未必符合程序语言的语法。请参照伪代码的逻辑，编写相应的程序代码。

算法框架：如果当前字符是‘。’，‘？’，‘！’，此句结束；

如果当前字符是英文字母，把该字符存入此句；

如果不是以上两种情况，只需要在单词之间存储一个空格即可；

```
Char    currentChar;   //当前正在处理的字符
String   TempSentence="";   //用于存储当前处理的句子

If((currentChar >='a')&&( currentChar <='z'))||
(( currentChar >='A')&&( currentChar <='Z'))
//用中文表示上面语句的含义_____
{
        TempSentence += currentChar;   //用中文表示本语句的含义_____
}
If((currentChar =='.')|| (currentChar =='?')|| (currentChar =='!'))
//用中文表示上面语句的含义_____
{
        把 TempSentence 存入数据结构；
}
else
{
      如果 TempSentence 最后一个字符是空格，do nothing;
      否则在 TempSentence 最后加上一个空格；
         //解释一下，为什么写这两个语句_____
}
```

步骤二中的算法与步骤一类似，请同学们自行完成。

【任务 4.2----步骤三】

比对 TempSentence 与 lib.txt 中的每一句，采用动态规划算法；来自 lib.txt 中的句子用 LibSentence 表示。TempSentence 与 LibSentence 分别包含 m 与 n 个单词。

String　TempWord, LibWord;

//表示从 TempSentence 与 LibSentence 分别取的单词

int　　wordSimilarity[m][n];

//wordSimilarity[i][j]表示 TempSentence 的第 i 个单词与 LibSentence 的第 j 个单词是否相同

Float　score[m][n];

//该矩阵表示运行动态规划算法时 TempSentence 与 LibSentence 的比对得分

首先建立矩阵 wordSimilarity[m][n]

此步骤的输入为两个空格分隔的句子譬如

I have a book

You have a car

则输出为 4*4 的一个矩阵，矩阵的第 i 行第 j 列表示第一句的第 i 个单词与第二句的第 j 个单词是否相同。此例中构造的矩阵为

$$\begin{vmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

k=0;　i=0;

TempWord="";　LibWord="";

TempCount=0;　LibCount=0;

```
While(k<TempSentence.length())
{
    If(TempSentence.length[k]是英文字母)
    {
        TempWord+= TempSentence.length[k];
    }   //用中文表示上面语句的含义_____
   Else if  ((TempSentence.length[k]=='.')|| (TempSentence.length[k]=='!')||
            (TempSentence.length[k]=='?'))
            {
                i=0;
                while(i<LibSentence.length())
                {
                    与外面的逻辑相同，获取 LibWord;
                    If(LibWord==TempWord)
                    {
                        wordSimilarity[TempCount][LibCount]=1;
                    }
                    Else
                    {
                        wordSimilarity[TempCount][LibCount]=0;
                    }
                    LibCount++;
                }
                TempCount++;
            }
        }
```

```
            Else
            {
                If(TempWord 的最后一个字符不是空格)
                {
                    TempWord+='  '; //用中文表示上面语句的含义
_____
                }
            }
        }
}
```

然后建立矩阵 score[m][n]，该矩阵的递归关系有三种情况:

TempSentence 的最后一个单词比对上空格，则 score[i][j]=score[i-1][j];

LibSentence 的最后一个单词比对上空格，则 score[i][j]=score[i][j-1];

TempSentence 的最后一个单词比对上 LibSentence 的最后一个单词，则 score[i][j]= score[i-1][j-1] + wordSimilarity[i][j];

最终 score[i][j]取三者的最大值，即

score[i][j]=max{ score[i-1][j]，score[i][j-1]，

score[i-1][j-1] + wordSimilarity[i][j]}；

```
for(k=1;k<=m;k++)
{
    For(j=1;j<=n;j++)
    {
        Score[k][j]= max{ score[k-1][j] , score[k][j-1] , score[k-1][j-1] + wordSimilarity[k][j];
    }
```

}

注：wordSimilarity[i][j]矩阵下标从 1 开始；Score[0][j]与 Score[k][0]的值初始化为 0；

此处可以参考算法课的最长公共子序列的初始化与递归关系。

【任务 5】

将任务 4 中的算法转化为具体的程序代码，请你用 C/C++语言实现本任务。

【总结】

在完成本项任务时，我们的解答过程分为以下步骤：

第一，将任务分解为便于实现的多个子模块；

第二，为各步骤（子模块）设计大致的算法框架；

第三，为各步骤（子模块）设计所需的数据结构（需要非常具体）；

第四，将上述的算法框架具体化，并写出各算法的伪代码；

第五，将上述的具体算法转化为程序代码。

实验三 使用哈夫曼算法完成文本压缩

一个文本 data.txt，仅包含中文字符和标点。本项目要求你采用哈夫曼编码算法，将其中的每个字符和标点（仅包含中文字符和标点，不包括数字和英文字母）分配一个二进制前缀码，使得编码的文件尽可能的小。

例如

输入文本：

一个苹果、一个芒果、一个百香果共三个水果。

输出文本包括三个：

文件 1：字符编码表。此文件包含每个字符的二进制编码和频率，以及编码的总码长。

输入格式（字符频率从高到低）

字符　频率　编码

个　4　11

果　4　10

一　3　011

、　2　010

苹　1　00111

。　1　00110

芒　1　00101

百　1　00100

香　1　00011

共　1　00010

三　1　00001

水　1　00000

编码总码长：　71 位

文件 2：编码文件（二进制存储）。请注意，以下的 0、1 不是字符，而是二进制

的一个 bit。

输出格式

01111011110010011110101100100111101000100000111000010000011100000100110

文件 3：解码文件。请写一个函数，读取文件 2（不能跳过），利用编码表解析出原文。

输入格式

一个苹果、一个芒果、一个百香果共三个水果。