数据结构与算法

疫情面前，让我们一起努力！

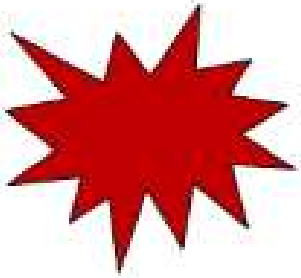
1

2

让人担忧

敏感信息、数据

序号	简要	描述
1	个人基本资料	个人姓名、生日、性别、民族、国籍、家庭关系、住址、个人电话号码、电子邮箱等
2	个人身份信息	身份证、军官证、护照、驾驶证、工作证、出入证、社保卡、居住证等
3	个人生物识别信息	个人基因、指纹、声纹、掌纹、耳廓、虹膜、面部特征等
4	网络身份标识信息	系统账号、IP地址、邮箱地址及与前述有关的密码、口令、口令保护答案、用户个人数字证书等
5	个人健康生理信息	个人因生病医治等产生的相关记录，如病症、住院史、医嘱单、检验报告、手术及麻醉记录、护理记录、用药记录、药物食物过敏信息、生育信息、以往病史、诊治情况、家族病史、现病史、传染病史等，以及与个人身体健康状况产生的相关信息，及体重、身高、肺活量等
6	个人教育工作信息	个人职业、职位、工作单位、学历、学位、教育经历、工作经历、培训记录、成绩单等
7	个人财产信息	银行账号、验证信息(口令)、存款信息（包括资金数量、支付收款记录等）、房产信息、信贷记录、征信信息、交易和消费记录、流水记录，以及虚拟货币、虚拟交易、游戏类兑换码等虚拟财产信息



在对文本内容的敏感信息进行保护时，如何实现在读取文本内容的同时，对敏感信息进行查找。

以往学习的数据结构，能够解决么？

如何**查找**敏感信息？

请大家一起找答案

Word 文本解析和关键字快速匹配方法*

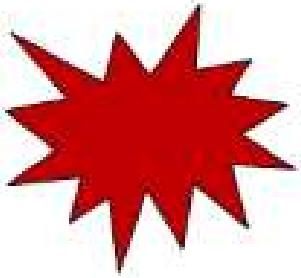
廖怨婷，兰小龙，陈庆春

（西南交通大学，四川 成都 611756）

摘　要：如今，Microsoft Word 文档是网络传输的主要文件。因此，研究有效监测 Word 文件等有关传输文本的网络通信安全至关重要。在详细分析 Word 文档的二进制格式后，给出了 Word 文本内容解析流程，针对满足服务器下载大量文本的实时性要求，在研究分析经典模式匹配算法的基础上，提出了一种改进的 BMHS 模式匹配算法。实验结果显示，相比 BMHS 算法，给出的改进 BMHS 算法可有效减少匹配次数，提高匹配效率，有效满足 Word 文本解析条件下的关键字快速匹配应用要求。

关键词：Word 文本解析；Word 二进制文件格式；模式匹配；BMHS 算法

中图分类号：TP309.2；TP301.6　　**文献标志码**：A　　**文章编号**：1002-0802(2018)-03-0647-06



从文献中可知，模式匹配是一种查找敏感信息的有效方法。

模式匹配问题，是串的主要应用。

串是什么？是字符串吗

串的基本概念

串(String)：是零个或多个字符组成的有限序列。是数据元素为单个字符的*特殊线性表*。

一般记作 $S="a_1a_2a_3\ldots a_n"$（$n\geq 0$）

其中：S 是串名；

双引号括起来的字符序列是串值；

$a_i(1\leq i\leq n)$可以是字母、数字或其它字符；

例如：S1="abcdefg"　　S2="123456789"

S3="changchun_7089hao"

S4="123456....."

串名

串值

不是串，S4是无限序列

串的基本概念

子串： 串中连续的任意个字符组成的子序列称为该串的子串。

主串： 包含子串的串。

子串在主串中的位置： 是子串在主串中首次出现时，该子串的首字符在主串中对应的序号。

例如：s="I Love China，China is beautiful！！"
p="China"
那么p是s的子串，s为主串。p在s中的位置是 8 。

特殊：空串是任意串的子串，任意串是其自身的子串。

8

串的基本概念

串的长度： 是指串中所包含的字符个数。

例如：S3="changchun_7089hao"

长度=17

串相等： 串长度相等，且对应位置上字符也相等。

空串(Empty String)： 长度为零的串，它不包含任何字符。（n=0）

空格串： 是由n个空格组成的串。（n≥1）

注意： 空串和空格串不同，如“ ”和“”分别表示长度为1的空格串和长度为0的空串（〇）。

9

串的抽象数据类型定义

ADT String {

数据对象　$D=\{ a_i | a_i \in CharacterSet,\ i=1,2,...,n \quad n\geq 0 \}$

数据关系　$R=\{ <a_{i-1}, a_i> | a_{i-1}, a_i \in D,\ i=2,...,n \}$

基本操作

1. StrAssign (&T, chars)　//字符串赋值

参数要求:chars 是字符串常量。

操作结果:生成一个值为 chars的串 T 。

串的抽象数据类型定义

2. StrCopy (&T, S)　//字符串复制

参数要求:串 S 存在。

操作结果:由串 S 复制得串 T。

3. DestroyString (&S)　//字符串销毁

参数要求:串 S 存在。

操作结果:串 S 被销毁。

4. StrEmpty(S)　//字符串判空

参数要求:串S存在。

操作结果:若 S 为空串，则返回TRUE，

否则返回 FALSE。

串的抽象数据类型定义

5. StrCompare (S, T)　//字符串比较

参数要求:串 S 和 T 存在。

操作结果:若S > T，则返回值 > 0；

若S = T，则返回值 = 0；

若S < T，则返回值 < 0。

例如：　**StrCompare("chang", "chun") < 0**

StrCompare("wu","wang") > 0

StrCompare("china", "china") = 0

串的抽象数据类型定义

6. StrLength(S)　//求字符串长度

参数要求：串 S 存在。

操作结果：返回 S 的元素个数，称为串的长度。

7. Concat(&T,S1,S2)　//串连接

参数要求：串 S1 和 S2 存在。

操作结果：用 T返回由S1和S2连接而成的新串。

13

串的抽象数据类型定义

8. SubString (&Sub, S, pos, len)　//截取子串

参数要求：串 S 存在，

1≤pos≤StrLength(S)

且0≤len≤StrLength(S)-pos+1。

操作结果：用 Sub 返回串 S 的第 pos 个字符起长len的子串。

例如：　**SubString(sub, “data_structure”, 6, 6)**

求得 sub = “struct”;

例如：　**SubString(sub, "data_structure", 6, 10)**

求得 sub =? ? ? ? ;

14

串的抽象数据类型定义

9. Index (S, T, pos)　//求子串T在主串S中的位置（序号）

参数要求：串S和T存在，T是非空串，

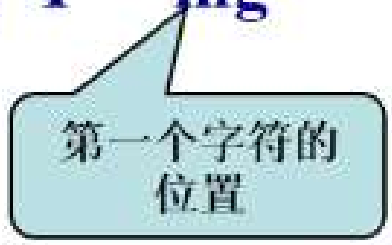
1≤pos≤StrLength(S)。

操作结果：若主串 S 中存在和串 T 值相同的子串，则返回T在主串 S 中第pos个字符之后第一次出现的位置；否则函数值为0。

例如：假设 S = "stringstringstring"，T ="ing"

Index (S, T, 2)=4

Index (S, T, 5)=10



串的抽象数据类型定义

10. Replace (&S, T, V)

参数要求：串S, T和 V 均已存在，且 T 是非空串。

操作结果：用V替换主串S中出现的所有与串T相等的不重叠的子串。

例如： **S = "bcbcbcbcbcb", T = "bcb"**

若 **V = "string"**, 则经置换后得到

S = "stringcstringcstring"

串的抽象数据类型定义

11. StrInsert (&S, pos, T)

参数要求：串S和T存在，1≤pos≤StrLength(S)＋1。

操作结果：在串S的第pos个字符上插入串T。

例如：**S = "changligong"，T = "chun"，**

则执行 **StrInsert(S, 6, T)** 之后得到

S = "changchunligong"

例如：**S = "changchun"，T = "ligong"，**

则执行 **StrInsert(S, 10, T)** 之后得到

S = "changchunligong"

17

串的抽象数据类型定义

12. StrDelete (&S, pos, len)

参数要求：串S存在1≤pos≤StrLength(S)-len+1。

操作结果：从串S中删除第pos个字符起长度为len的子串。

13. ClearString (&S)

参数要求：串S存在。

操作结果：将S清为空串。

} end String

10

串的抽象数据类型定义

在上述抽象数据类型定义的13种操作中

串赋值StrAssign　　**串复制Strcopy**

串比较StrCompare　　**求串长StrLength**

串联接Concat　　**求子串SubString**

等六种操作构成串类型的最小操作子集，这些操作不可能利用其它串操作来实现，其它串操作可在这个最小操作子集上实现。

单选题 4分

若串S1="ABCDEFG",
S2="9898",S3="###",S4="012345",执行
concat(replace(S1,substr(S1,length(S2),length(S3)),S3),su
bstr(S4,index(S2,"8",1),length(S2)))

A "ABC###G1234"

B "AB###G1234"

C "ABC##G1234"

D "ABC###G12"

串的表示和实现

串的表示和实现

串的存储方式

顺序存储

- 定长顺序存储表示

——用一组地址连续的存储单元（定长字符数组）存储串值的字符序列，存储空间采用静态分配方式。

- 堆分配存储表示

——用一组地址连续的存储单元存储串值的字符序列，但存储空间是在程序执行过程中动态分配而得。

链式存储

- 串的块链存储表示

——链式方式存储

串的两种不同表示方式。

串的定长顺序存储表示

用一组地址连续的存储单元（定长字符数组）存储串值的字符序列，存储空间采用静态分配方式。

例如：

```
#define  MAXSTRLEN  255
      // 用户定义的最大串长
typedef   unsigned char  SString[MAXSTRLEN + 1];
      // 0号单元存放串的长度
```

串的实际长度在此范围内可任意变化，若超出此长度，则被舍去，称为“截断”

sstring[0]存放串长，串值存放在从sstring[1]~sstring[sstring[0]]单元中

23

串的定长顺序存储表示

采用定长顺序存储方式，串的操作算法举例：

例如：求子串 SubString(&Sub, S, pos, len)

```
Status SubString(SString &Sub, SString S, int pos, int len) {
    // 用Sub返回串S的第pos个字符起长度为len的字串
  （1）if （pos<1 || pos>S[0] || len<0 || len>s[0]-pos+1）
  （2）      return ERROR;
  （3） Sub[1..len]=S[pos..pos+len-1];
  （4） Sub[0]=len;
  （5） return OK;
}//SubString
```

pos的合法取值范围是
0<pos<=S[0]

len的合法取值范围是
0<=len<=S[0]-pos+1

串的定长顺序存储表示

例如：串的联接操作Concat(&T, S1, S2)（见算法4.2）

由于采用定长静态字符数组存储，应当考虑联接后是否有可能被截断，故联接操作需分三种情况考虑：

第一种情况：

S1和**S2**联接后，长度小于**255**，**S1**，**S2**均未被截断。

第二种情况：

S1和**S2**联接后，长度大于**255**，**S1**没被截断，**S2**被截断了。

第三种情况：

S1长度为**255**，只将**S1**串复制到**T**串。

25

串的堆分配存储表示

如果想存储长的字符串怎么办？？
如果随着不断的对串执行连接，插入等操作，串长在不断变化怎么办？？

静态存储有弊端，改用动态分配的一维数组——堆 或 链式存储方式——块链

26

串的堆分配存储表示

用一组连续的存储单元来存放串，但存储空间是在程序执行过程中动态分配而得。

以C语言为例：系统利用函数malloc( ) 和 free( ) 进行串值空间的动态管理。

类型定义：

```
Typedef   struct {
    char  *ch; // 空串 ch = NULL；非空串ch存储分配的首地址
    int   length; //串长度
}HString
```

27

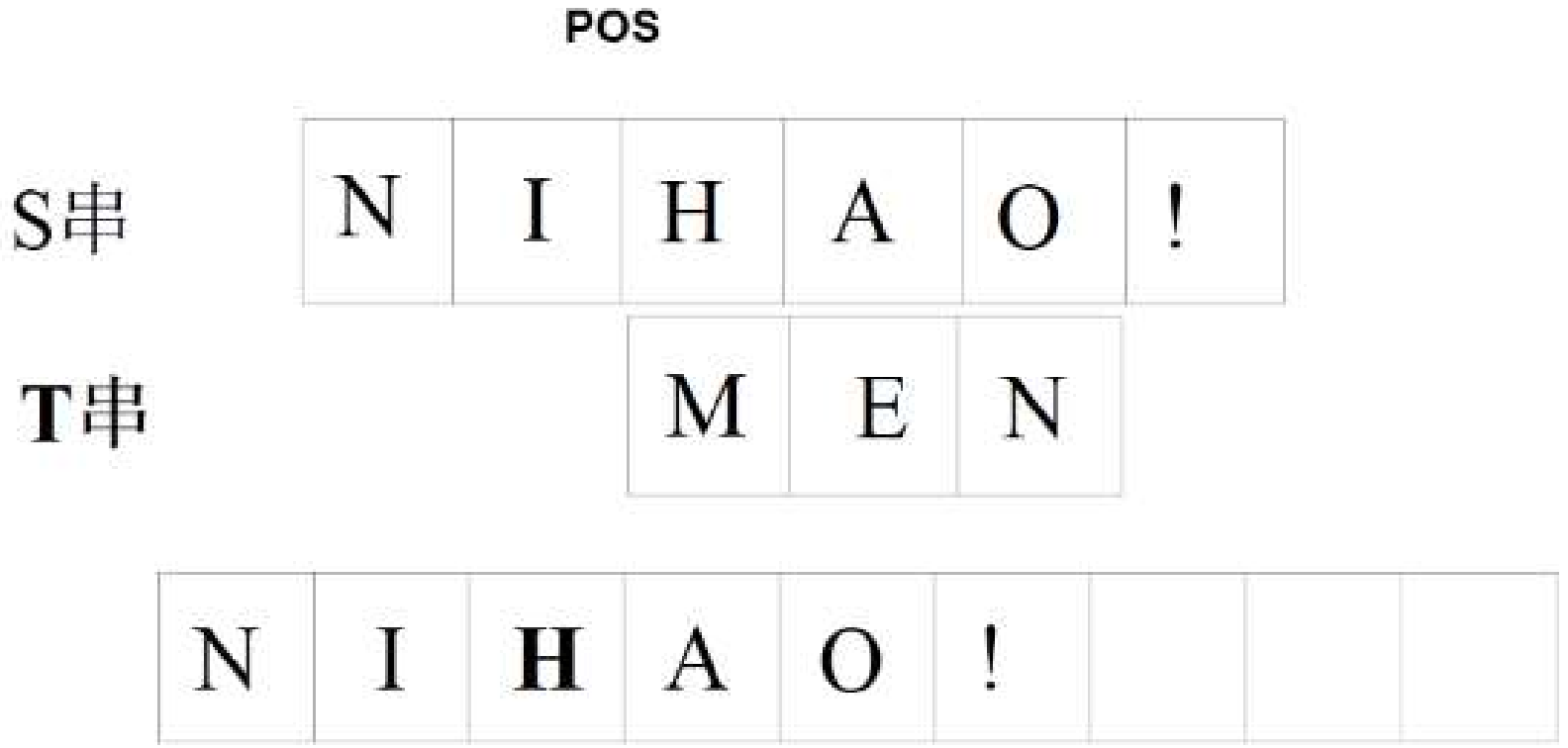
串的堆分配存储表示

堆分配存储的主要思想：根据要生成的串的实际长度动态分配内存空间，再进行串的复制。

下面以串插入操作**StrInsert( )**
截子串**Substring( )**为例：

20

串的堆分配存储表示



串插入操作：在S串pos位置插入T串示例

29

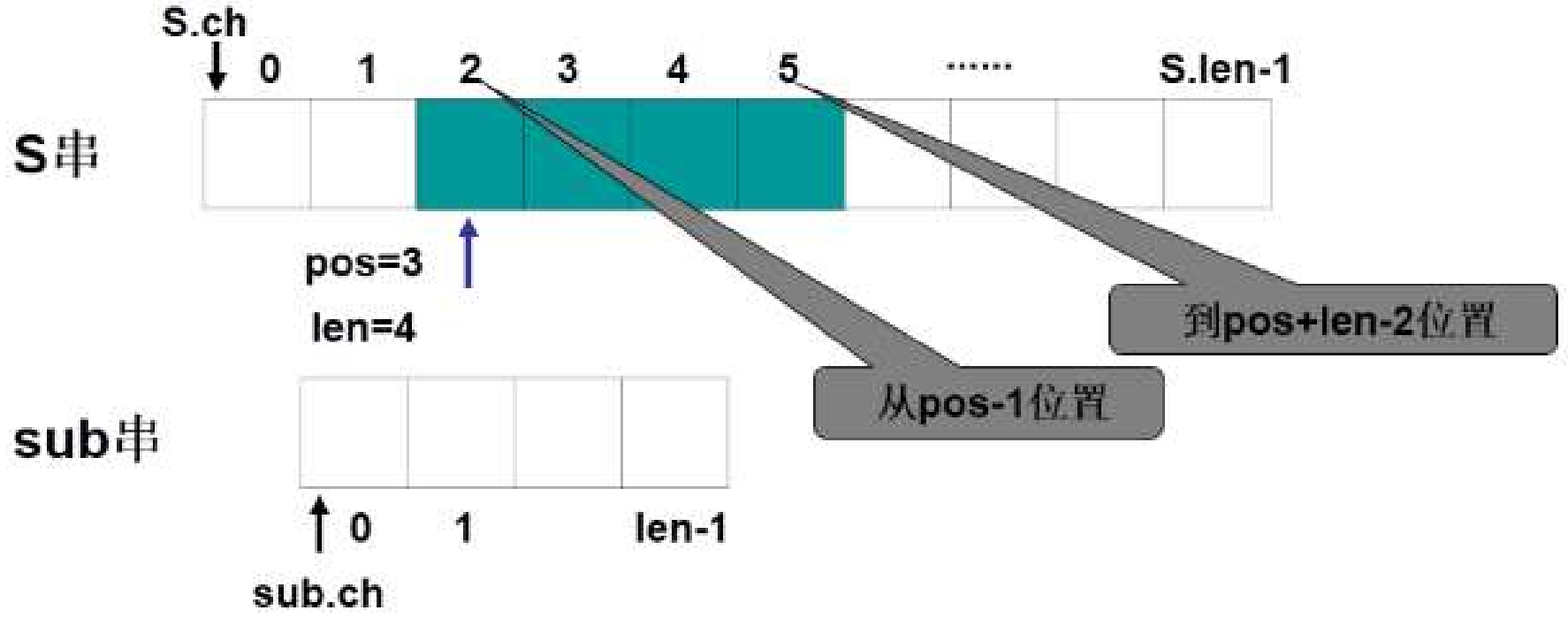
串的堆分配存储表示

```
Status StrInsert ( HString &S, int pos, HString T ){
        //在串S的第pos个字符之前（包括尾部）插入串T
 （1）if (pos<1||pos>S.length+1) return ERROR;   //pos不合法则告警
 （2）if(T.length){      //只要串T不空，就需要重新分配S空间，以便插入T
 （3）S.ch=(char*)realloc(S.ch, (S.length+T.length)*sizeof(char));
 （4）for ( i=S.length-1; i>=pos-1; --i )     //为插入T而腾出pos之后的位置
 （5）   S.ch[i+T.length] = S.ch[i];      //从S的pos位置起全部字符均后移
 （6）   S.ch[pos-1…pos+T.length-2] = T.ch[0…T.length-1]; //插入T
 （7）   S.length + = T.length;          //更新S串长度
      }  return OK;
}//StrInsert
```

30

串的堆分配存储表示

截子串操作：SubString(HString &Sub, HString S,int pos, int len)



31

串的堆分配存储表示

```
Status SubString(HString &Sub, HString S,int pos, int len) {
 // 用Sub返回串S的第pos个字符起长度为len的子串
  (1) if (pos < 1 || pos > S.length || len < 0 || len > S.length-pos+1)
        return ERROR;
  (2) if (Sub.ch)    free (Sub.ch);          // 释放旧空间
  (3)   if (!len)
  (4)    { Sub.ch = NULL;  Sub.length =       // 截取空串作为子串
  (5)   else {          截完整子串          }
  (6)   return OK;
} // SubString
```

首先判断pos和len的合法范围

如果截长度为0的子串

32

主观题 1分

尝试将截完整子串的操作补充完整。

串的堆分配存储表示

截完整子串

```
Sub.ch = (char *)malloc(len*sizeof(char));

Sub.ch[0..len-1] = S.ch[pos-1..pos+len-2];

Sub.length = len;
```

34

串的堆分配存储表示

```
int Concat(HString &T, HString S1, HString S2) {
  // 用T返回由S1和S2联接而成的新串
  (1)if (T.ch)  delete(T.ch);        // 释放旧空间
  (2)if (!(T.ch =(char *)malloc((S1.length+S2.length)*sizeof(char))))
       exit (OVERFLOW);
  (3)for(i=0;i<=S1.length-1;i++)T.ch[i] = S1.ch[i];
  (4)T.length = S1.length + S2.length;
  (5)for(i=S1.length;i<=T.length-1;i++)T.ch[i] = S2.ch[i];
  (6)return OK;
} // Concat
```

35

串的块链存储表示

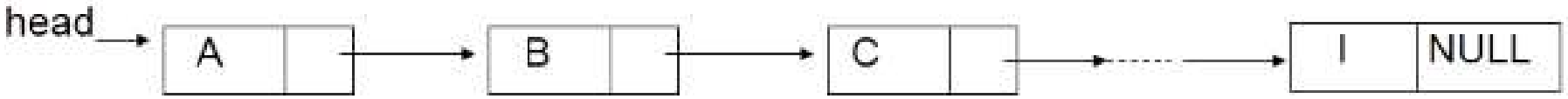
也可用链表来存储串值，由于串的每个数据元素是一个字符，因此用链表存储时，通常一个结点中可以存放一个字符，也可以存放多个字符。

$$\text{存储密度} = \frac{\text{数据元素所占存储空间}}{\text{实际分配的存储空间}}$$

36

串的块链存储表示

结点大小为1，即每个结点存放1个字符



存储密度为= $\frac{1}{5}$ （假设一个字符占一个字节，指针占4个字节）

结点大小为4，即每个结点存放4个字符

存储密度为= $\frac{4}{8}$ （假设一个字符占一个字节，指针占4个字节）

串的块链存储表示

块链类型定义：

```
#define  CHUNKSIZE  80                    //可由用户定义的块大小
typedef  struct  Chunk {
         char    ch [ CHUNKSIZE ];
         struct  Chunk * next ;
}Chunk;
                typedef  struct {
                      Chunk  *head;          //头指针
                      Chunk  *tail;          //尾指针
                      int  curLen;           //串的当前长度
                  } LString;
```

结点结构

块链结构

数据结构与算法

41