

学号：20184484

班级：计算机 1803

姓名：胥卜凡

#### 目录

1 在输入的长度为 $n$ 的正文串中，查找某一字符是否出现，若出现输出 1，否则输出 0，设计时间复杂度为 $O(n)$ 的算法求解这个问题。 .....	2
1.1 伪代码 .....	2
1.2 分析 .....	2
1.3 样例程序 .....	3
2: 设计亚线性算法检查两个字符串是否相同 .....	5
2.1 伪代码 .....	5
2.2 分析 .....	5
2.3 样例程序 .....	6
附录 I: .....	9

1 在输入的长度为  $n$  的正文串中，查找某一字符是否出现，若出现输出 1，否则输出 0，设计时间复杂度为  $O(n)$  的算法求解这个问题。

## 1.1 伪代码

---

**算法 1** 判断字符串中是否有目标字符  $w$

---

**输入:** 字符串  $S$ , 目标字符  $w$ , 近似参数  $\epsilon$

**输出:**  $S$  中有  $w$  输出 1,  $S$  中无  $w$  输出 0

```
1:  $result = 0$ ;  
2: for  $i=1$  to  $d=2/\epsilon$  do  
3:   随机选取  $S$  中字符  $k$ ;  
4:   if  $k==w$  then;  
5:      $result = 1$ ;  
6:   return  $result$ ;  
7:   end if;  
8: end for;  
9: return  $result$ ;
```

---

## 1.2 分析

(1) 算法设计:

考虑定义  $\epsilon$ -远离如下, ”对于字符串  $S$  和目标字符  $w$ , 如果  $S$  中  $w$  的数量大于  $\epsilon n$ , 则是  $\epsilon$ -远离的”。

如果  $S$  中没有目标字符, 抽样必定没有目标字符, 算法不会出错。

如果  $S$  中有目标字符, 由于  $S$  是  $\epsilon$ -远离的 ( $S$  中  $w$  的数量大于  $\epsilon n$ ), 也就是说随机抽出一个数, 是  $w$  的概率大于  $\epsilon$ , 不是  $w$  的概率小于等于  $(1-\epsilon)$ 。那么抽样全部都是非  $w$  的概率, 即出错的概率

$$\Pr[\text{error}] = \Pr[\text{抽样中没有 } w] \leq (1-\epsilon)^d \approx e^{-\epsilon d} = e^{-2} < \frac{1}{3}$$

(2) 复杂度分析:

显然时间复杂度为 for 循环, 即  $T(n) = O(2/\epsilon) < O(n)$ , 即  $T(n) = o(n)$ , 满足题意。

## 1.3 样例程序

代码:

```
1. #include <iostream>
2. #include <string>
3. #include <ctime>
4. using namespace std;
5. int main()
6. {
7.     unsigned seed=time(0);
8.     string A;
9.     cin>>A;
10.    int result=0;
11.    float e;
12.    cin>>e;
13.    char w;
14.    cin>>w;
15.    int len=A.size();
16.    int you=0,wu=0;
17.
18.    for(int j=0; j<10000; j++)
19.    {
20.        int flag=1;
21.        for(int i =0; i<2/e; i++)
22.        {
23.            int k=rand()%len;
24.            if(A[k]==w)
25.            {
26.                cout<<"有目标字符"<<endl;
27.                flag=0;
28.                you++;
29.                break;
30.            }
31.        }
32.        if(flag)
33.            {wu++;
34.            cout<<"无目标字符"<<endl;}
35.    }
36.    cout<<"字符串长度"<<len<<endl;
37.    int num=0;
38.    for(auto &i:A)
39.    {
40.        if(i==w)
```

```
41.             num++;
42.         }
43.         cout<<w<<"共"<<num<<endl;;
44.         cout<<"成功:"<<you<<"失败:"<<wu<<endl;
45.
46. }
47. //aaaaaaaaabaaaaaaaaacbbbbbbbbbbcasdwpgepgccpdap
```

结果:

```
C:\Users\29459\Documents\cb\p121\bin\Debug\p121.exe  
有目标字符  
有目标字符  
有目标字符  
有目标字符  
无目标字符  
有目标字符  
有目标字符  
有目标字符  
有目标字符  
有目标字符  
有目标字符  
有目标字符  
有目标字符  
有目标字符  
有目标字符  
有目标字符  
有目标字符  
有目标字符  
有目标字符  
有目标字符  
有目标字符  
有目标字符  
字符串长度49  
c共4  
成功:8151失败:1849  
  
Process returned 0 (0x0)    execution time : 70.515 s  
Press any key to continue.
```

输入:

aaaaaaaaabaaaaaaaaaacbbbbbbbbbbcasdwpgegpccpdap

0.1

C

得到结果为成功率 0.81，其中  $\epsilon$  取 0.1，显然满足算法中要求的概率

## 2: 设计亚线性算法检查两个字符串是否相同

### 2.1 伪代码

---

**算法 2** 检查两个字符串是否相同

---

**输入:** 字符串  $s_1$ , 字符串  $s_2$

**输出:**  $s_1 == s_2$ , 输出 1;  $s_1 != s_2$ , 输出 0

```
1:  $result = 0$ 
2:  $n = \max(\text{length}(s_1), \text{length}(s_2))$ 
3:  $p = 2^{N^2 \ln(n)}$ 
4:  $F_p(s_1) = \sum_{i=1}^n s_{1i} 2^{i-1}$ 
5:  $F_p(s_2) = \sum_{i=1}^n s_{2i} 2^{i-1}$ 
6: if  $F_p(s_1) == F_p(s_2)$  then
7:    $result = 1$ 
8: end if
9: return  $result$ 
```

---

### 2.2 分析

一种简单的方法是像之前的算法一样，随机选择一些位置来进行比较判断。但这种方法产生的错误在总的串中占的比例很小，会失效。

所以可以为两个字符串制作指纹来体现字符串的全局特征，即我们将数据看成  $n$  位的整数  $a$  和  $b$ （哈希）

$$a = \sum_{i=1}^n a_i 2^{i-1}$$

$$b = \sum_{i=1}^n b_i 2^{i-1}$$

定义指纹函数（ $p$  为素数）：

$$F_p(x) = x \bmod p$$

算法随机选择素数  $p$ ，检测上述指纹函数是否相等。

根据这个算法，在计算过程中，需要比较的数字最大不超过  $p$ ，因而需要传输  $\log p$  位，而不是  $n$  位。考虑到传输效率的问题，希望选择一个较小的  $p$ 。

在  $a$  和  $b$  不等且都和  $p$  同余的情况下，会发生假阳性的误判。设  $c=|a-b|$ ，也就是，当  $a \neq b$  且  $c$  整除  $p$  时发生误判。由于  $c \leq 2n$ ， $c$  的素因子至多有  $n$  个。给定一个  $p$ ，通过素数定理，大概有  $p/\ln(p)$  个小于  $p$  的素数，此时出错的概率

$$Pr[\text{error}] = n \times \ln(p) / p。$$

与此同时  $p$  要足够大以确保有足够小于  $p$  的素数，又要足够小以确保高效的通信。

取  $p=2n^2\ln(n)$ ， $p$  为大于  $2n^2\ln(n)$  但和  $2n^2\ln(n)$  最接近的素数，由出错概率公式可知发生错误的概率不超过  $2/n$ ，这种情况下需要传输的位数为

$$\log(2n^2\ln(n)) = \log(\sqrt{2}) + 2\log(n) + \log(\ln(n)) > \log(n)$$

因此传输位数的复杂度为  $O(\log(n))$ 。

## 2.3 样例程序

```
1. #include <string>
2. #include <iostream>
3. #include <math.h>
4. #include <ctime>
5. using namespace std;
6.
7. int get(int length, int x)
8. {
9.     int ok=1;
10.    //先判断是否是素数
11.    if(length==x)
```

```

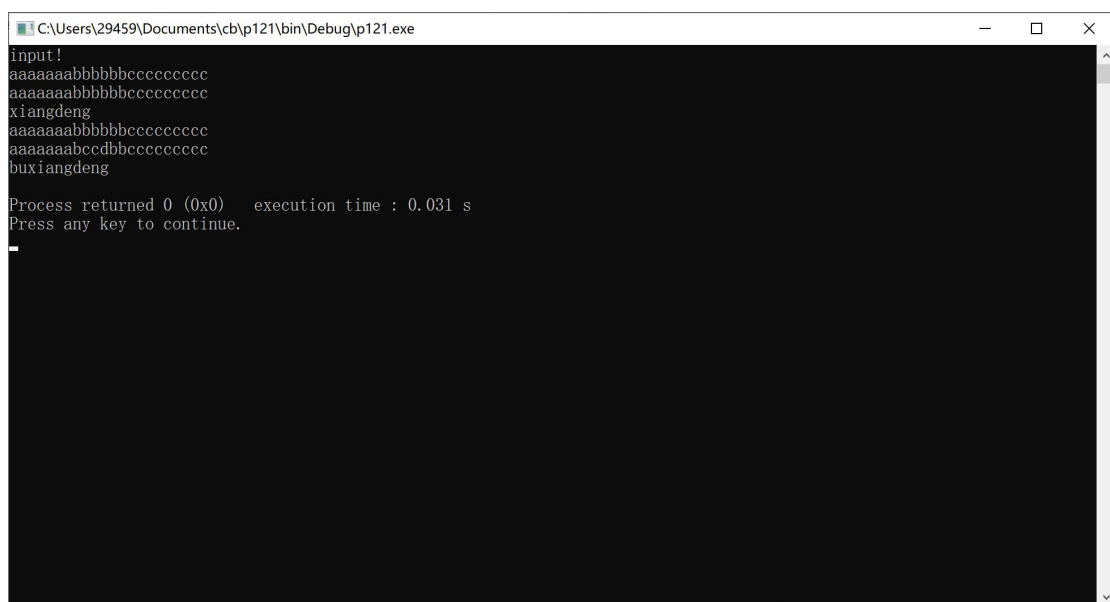
12.         return 0;
13.     for(int i=2; i<=length/2; i++)
14.     {
15.         if(length%i==0)
16.         {
17.             ok=0;
18.             break;
19.         }
20.     }
21.     if(ok==1)
22.         return length;
23.     else
24.     {
25.         length=get(length+1, x);
26.         return length;
27.     }
28. }
29. void panduan(string s1, string s2)
30. {
31.     int len1=s1.size();
32.     int len2=s2.size();
33.     srand(time(0));
34.     if(len1==len2)
35.     {
36.         long long asum=0;
37.         long long bsum=0;
38.         int p=(int)2*len1*len1*log(len1);
39.         int length=ceil(log(p));
40.         p=get(p, p*p); //找到离他最近, 比他大的素数
41.         for(int i=0; i<length; i++)
42.         {
43.             int j=rand()%len1;
44.             {
45.                 asum+=s1[j]*pow(2, i);
46.                 bsum+=s2[j]*pow(2, i);
47.             }
48.         }
49.         /*
50.         cout<<p<<endl;
51.         cout<<length<<endl;
52.         cout<<asum<<endl;
53.         cout<<bsum<<endl;
54.         */
55.         if(asum%p==bsum%p)

```

```

56.             cout<<"xiangdeng"<<endl;
57.         else
58.             cout<<"buxiangdeng"<<endl;
59.     }
60.     else
61.         cout<<"buxiangdeng"<<endl;
62. }
63. int main()
64. {
65.     string s1;
66.     string s2;
67.     string s3;
68.     string s4;
69.     cout<<"input!"<<endl;
70.     s1="aaaaaaabbbbbccccccccc";
71.     s2=s1;
72.     s3=s1;
73.     s4="aaaaaabccdbbccccccccc";
74.     cout<<s1<<endl;
75.     cout<<s2<<endl;
76.     panduan(s1,s2);
77.     cout<<s3<<endl;
78.     cout<<s4<<endl;
79.     panduan(s3,s4);
80.
81.     return 0;
82. }

```



```

C:\Users\29459\Documents\cb\p121\bin\Debug\p121.exe
input!
aaaaaaabbbbbccccccccc
aaaaaaabbbbbccccccccc
xiangdeng
aaaaaaabbbbbccccccccc
aaaaaabccdbbccccccccc
buxiangdeng

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.

```



# 附录 I:

## 1 算法截图

算法 1 判断字符串中是否有目标字符 $\omega$

输入: 字符串S,目标字符 $\omega$ ,近似参数 $\varepsilon$

输出: S中有 $\omega$ 输出1, S中无 $\omega$ 输出0

1:  $result = 0$ ;

2: **for**  $i=1$  **to**  $d=2/\varepsilon$  **do**

3:     随机选取S中字符 $k$ ;

4:     **if**  $k==\omega$  **then**;

5:          $result = 1$ ;

6:         **return**  $result$ ;

7:     **end if**;

8: **end for**;

9: **return**  $result$ ;

算法 2 检查两个字符串是否相同

输入: 字符串 $s_1$ ,字符串 $s_2$

输出:  $s_1==s_2$ , 输出1; $s_1!=s_2$ ,输出0

1:  $result = 0$

2:  $n=\max(length(s_1),length(s_2))$

3:  $p=2N^2\ln(n)$

4:  $F_p(s_1)=\sum_{i=1}^n s_{1i}2^{i-1}$

5:  $F_p(s_2)=\sum_{i=1}^n s_{2i}2^{i-1}$

6: **if**  $F_p(s_1) == F_p(s_2)$  **then**

7:      $result=1$

8: **end if**

9: **return**  $result$

## 2 参考文献

[1]王宏志. 大数据算法[M]. 机械工业出版社, 2015.