

EXP4 算法基础实验报告

廖佳怡 PB19151776

1. 实验设备和环境

编译环境: Windows10

编程语言: C++

机器内存: 16.0GB

时钟主频: 2.30GHz

2. 实验内容及要求

- 实验3.1 KMP算法

- 内容

给定文本串T、模式串P, T的长度为n, P的长度为m, 采用KMP算法进行字符串匹配。

- 要求

- (n, m)共有5组取值, 分别为: (28, 23), (29, 24), (210,25), (211,26), (212,27), 见/ex1/input/4_1_input.txt。
 - 输出所有匹配的T的开始下标和 Π 的函数值, 记录找到所有匹配的时间, 并画出曲线分析。
 - 关于画图分析, KMP算法计算前缀函数的复杂度是 $O(m)$, 实际匹配的复杂度是 $O(n)$, 画图时这两块可以分开画, 横坐标分别取m和n。或者, 考虑到算法总复杂度是 $O(m+n)$, n比m大得多, 不妨以n为横坐标, 直接画总运行时间的曲线。

- 实验3.2 RK算法

- 内容

给定文本串T、模式串P, T的长度为n, P的长度为m, 采用Rabin-Karp算法进行字符串匹配。

- 要求

- (n, m)共有5组取值, 分别为: (28, 23), (29, 24), (210,25), (211,26), (212,27), 见/ex2/input/4_2_input.txt。
 - 基数d和素数q共4组取值, (d,q)分别为: (2,13),(2,1009),(10,13),(10,1009)
 - 输出所有匹配的T的开始下标和伪命中次数, 记录找到所有匹配的时间, 并画出曲线分析。其中, 伪命中指Hash值相等但并不匹配的情况。
 - 关于画图分析, RK算法最坏复杂度是 $O(mn)$, 考虑到n比m大得多, 这里可以直接以n作横坐标, 或以mn为横坐标, 然后不同(d,q)不同曲线。

3. 方法和步骤

Exp1

- 方法

KMP算法由有限自动机发展而来，无需计算转移函数，只用根据模式预先计算

$\pi[q] = \max\{k : k < q \text{ 且 } P_k \sqsupset P_q\}$ 。具体地，通过 `Compute_Prefix` 将模式自身与自身匹配，计算 π 数组，然后通过 `KMP_Matcher` 函数利用 π 数组将模式与待匹配字符串进行匹配。

- 步骤（关键代码展示）

- `Compute_Prefix`:

```
void Compute_Prefix(const string& P)
{
    int m=P.length();
    pi[0]=-1;
    int k=-1;
    for(int q=1;q<m;q++)
    {
        while(k>-1 && P[k+1]!=P[q])
            k=pi[k];
        if(P[k+1]==P[q])
            k++;
        pi[q]=k;
    }
}
```

- `KMP_Matcher`

```
int KMP_Matcher(const string& P,const string& T)
{
    int n=T.length();
    int m=P.length();
    Compute_Prefix(P);
    int q=-1,cnt=0;
    for(int i=0;i<n;i++)
    {
        while(q>-1 && P[q+1]!=T[i])
            q=pi[q];
        if(P[q+1]==T[i])
            q++;
        if(q==m-1)
        {
            cnt++;
            s[cnt]=i-m+1;
            q=pi[q];
        }
    }
    return cnt;
}
```

Exp2

- 方法

RK算法基本思想是：计算出模式P在d进制下的数字，与文本的每个可能字串计算出的d进制下的数字进行比较，若相等则粗略判断其可能命中，再通过逐一比较精确确定是否命中（即排除伪命中）。由于计算的数字可能很大，故采用模q运算。

- 步骤（关键代码解析）

```
void RK_Matcher(const string& P,const string& T,int d,int q)
{
    int n=T.length(),m=P.length();
    int h=d%q;
    if(h<0)h+=q;
    for(int i=1;i<=m-2;i++)
    {
        h=(h*d)%q;
        if(h<0)h+=q;
    }
    int p=0,t=0;
    cnt=0,false_cnt=0;
    for(int i=0;i<m;i++)
    {
        p=(d*p+P[i])%q;
        if(p<0)p+=q;
        t=(d*t+T[i])%q;
        if(t<0)t+=q;
    }
    for(int s=0;s<=n-m;s++)
    {
        if(p==t)
        {
            false_cnt++;
            bool matched=true;
            for(int i=0;i<m;i++)
            {
                if(P[i]!=T[s+i])
                {
                    matched=false;
                    break;
                }
            }
            if(matched)
            {
                cnt++;
                shift[cnt]=s+1;
            }
        }
        if(s<n-m)
        {
            t=(d*(t-T[s]*h)+T[s+m])%q;
            if(t<0)t+=q;
        }
    }
}
```

- 计算P和T在d进制下的模q数p、t:

```

int p=0,t=0;
cnt=0,false_cnt=0;
for(int i=0;i<m;i++)
{
    p=(d*p+P[i])%q;
    if(p<0)p+=q;
    t=(d*t+T[i])%q;
    if(t<0)t+=q;
}

```

- 粗略判断与精细判断命中:

```

for(int s=0;s<=n-m;s++)
{
    if(p==t)
    {
        false_cnt++;
        bool matched=true;
        for(int i=0;i<m;i++)
        {
            if(P[i]!=T[s+i])
            {
                matched=false;
                break;
            }
        }
        if(matched)
        {
            cnt++;
            shift[cnt]=s+1;
        }
    }
    if(s<n-m)
    {
        t=(d*(t-T[s]*h)+T[s+m])%q;
        if(t<0)t+=q;
    }
}

```

4. 结果和分析

Exp1

- 结果

完整版结果见result.txt

```

2
0 0 0 1 2 1 1 2
71 188

2
0 1 0 0 1 0 0 0 1 2 3 1 0 1 2 3
233 294

2
0 1 0 0 1 0 0 1 2 3 4 5 2 3 1 0 1 2 2 3 4 0 1 0 1 2 3 1 2 3 4 5
440 697

2
0 0 0 1 1 2 3 4 5 1 1 2 1 2 1 2 3 0 0 1 1 2 3 0 1 2 3 4 2 3 4 5 6 7 8 9 10 2 3 0 1 2 1 1 1 1 1 2 3 0 0 1 2 1 1 1 2
927 1907

2
0 1 2 3 0 0 1 2 0 1 2 0 1 0 1 2 3 4 4 4 5 1 2 0 1 0 0 1 2 3 0 0 0 1 2 3 4 4 5 6 7 8 9 0 1 0 1 2 3 4 4 4 5 6 7 0 0
1123 2726

```

- 时间

- 理论时间复杂度分析:

由于在 `Compute_Prefix` 中每次操作都与 k 的值变化相关 (每次 k 的值变化, 操作时间为 $O(1)$)。而 k 有两种变化, 递增1和递减为 $\pi[k]$ 。由于 k 的值不超过 m , 由聚合分析, k 的变化次数为 $O(m)$ 的, 故运行时间为 $O(m)$ 。

在 `KMP_Matcher` 中可由类似聚合分析得知运行时间为 $O(n)$ 的。

故总时间为 $O(m+n)$ 。

- 实际运行时间:

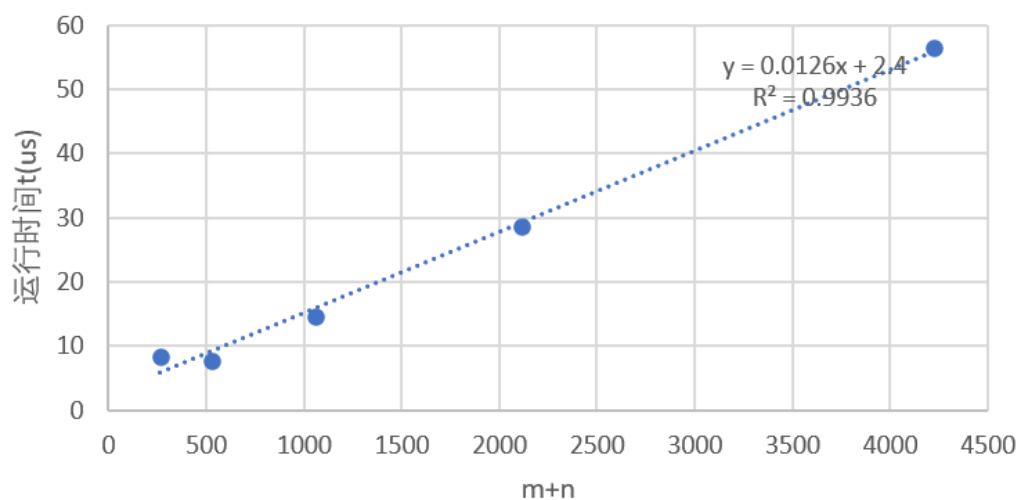
```

8.3us
7.7us
14.5us
28.6us
56.4us

```

将运行时间 t 与 $(m+n)$ 拟合如下图, $R^2 = 0.9936$, 与理论时间复杂度符合。

Exp1运行时间与数据规模的关系



Exp2

- 结果

```
2
15 0 15 0
79 197

2
40 2 34 0
162 261

2
74 1 77 0
400 687

2
145 1 134 0
624 1788

2
324 3 296 1
1476 2609
```

- 时间

- 理论时间复杂度分析：

RK算法的预处理时间(第一个循环有m次)为 $O(m)$ ，匹配时间（第二个循环外循环有 $n-m$ 次，精确匹配为m次循环）为 $O(mn)$ 的。

- 实际运行时间：

```
6.1us
8.4us
16.4us
32.4us
66.1us
```

将运行时间t与mn拟合如下图， $R^2 = 0.9608$ ，与理论时间复杂度符合。

Exp2运行时间与数据规模的关系

