Report

刘喆骐 2020013163 探微化01

1. 实验环境

windows10, vscode, c++11.

2. 算法简介

2.1 暴力算法

此法将模板P[1...m]和对应文本$T[s+1...s+m], 0 \leq s \leq n-m$逐一比较，得到解。外循环n-m-1次，内循环m次，时间复杂度为$O(nm)$

2.2 KMP算法

此法需要进行预处理，得到数组π，使得：

$$\pi[q] = max\{k : k < q \ and \ P_k \sqsupset P_q\}$$

预处理伪代码如下：

```
compute_prefix_funcction(P)
    m=P.size()
    let π[1...m] be a new array
    π[1]=0
    k=0
    for q=2 to m:
        while k>0 and P[k+1]!=P[q]:
            k=π[k]
        if P[k+1]==P[q]:
            k++
        π[q]=k
    return π
```

整个伪代码如下：

```
kmp(T,P)
    n=T.length
    m=P.length
    π=compute_prefix_function(P)
    q=0
    for i=1 to n:
        while(q>0 and P[q+1]!=T[i])
            q=π[q]
        if P[q+1]==T[i]
            q++
        if q==m:
            print(pattern occurs with shift i-m)
            q=π[q]
```

预处理时间复杂度为$\Theta(m)$，查找时间复杂度为$\Theta(n)$，总时间复杂度为$\Theta(m+n)$

2.2 BM算法

此法需要预处理,需要构造坏字符和好后缀。

2.2.1 坏字符规则

若 Text 中的失配字符在模式串中没有出现过, 那么不需要将模式串每次移动一个元素地进行后面的匹配, 可以直接跳过整个 Pattern; 同样, 若失配字符在 Pattern 中出现过, 则可移动到将其与模式串中该字符最靠右出现的位置处。计算该位置数组 bmBc 的函数COMPUTE−BMBC 的伪代码为:

```
COMPUTE – BMBC
m=P.length
for a in Σ:
    bmBc[a]=m
for i=1 to m-1:
    bmBc[P[i]]=m-i
```

2.2.2 好后缀规则

注意到模式串的前缀 P_i 的 k 位后缀可能与整个模式串的 k 位后缀相同, 而在匹配时通过相同后缀进行模式串的移动会很高效。在考虑参照后缀进行移动时, 也需考虑失配字符与 P_i 的 k 位后缀的前一个字符是否相同。计算好后缀规则表 bmGs 的函数伪代码为:

```
compute-osuff(P,Osuff)
    m=P.length
    k=0
    osuff[m]=m;
    for i=m to 1:
        for k=0 to i:
            if P[i-k]!=P[m-k]
                break;
        osuff[i]=k;


COMPUTE-BMGS(P,bmGs)
    m=P.length
    compute-osuff(P,Osuff)
    for i=1 to m:
        bmGs[i]=m

    j=1
    for i=m-1 to i:
        if Osuff[i]==i:
            while j<=m-i:
                if bmGs[j]==m:
                    bmGs[j]=m-i
                j=j+1
    for i=1 to m-1:
        bfGs[m-Osuff[i]]=m-i
```

总伪代码如下：

```
BM(T,P)
    n=T.length
    m=P.length
    COMPUTE-BMBC(P,bmBc)
    COMPUTE-BMGS(P,bmGs)
    s=0
    while s<=n-m:
        i=m
        while P[i]==T[s+i]
            if i==1:
                print(“Pattern occurs with shift” s)
            else:
                i=i-1
        s=s+MAX(bmGs[i],bmBc[T[s+i]]-m+i)
```

计算BMBC耗时$O(m+|\Sigma|)$，计算Osuff耗时$O(m^2)$，进而计算BMGS耗时$O(m+m^2)=O(m^2)$，匹配耗时$\Omega(\frac{n}{m})$。故总时间复杂度为$\Omega(\frac{n}{m}+m^2+|\Sigma|)$

3. 输入输出

使用的字符为{'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', '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',',','.',':','?',':',':','0','1','2','3','4','5','6','7','8','9'}。

可以在控制台中输入指令选择由控制台输入 Text 和 Pattern 并输出，或是由 input.txt 输入。算法执行完毕后会输出匹配开始处（如果有匹配，没有则不输出）和耗时。

```
Choose the method of input and output. 'manual' for typing yourself and 'auto' for reading fro
m existing sample.
auto
brute force time 38.1376ms
kmp time 49.1261ms
bm time 12.177ms


Choose the method of input and output. 'manual' for typing yourself and 'auto' for reading fro
m existing sample.
manual
please type the pattern and the text
123
1233938jsjhgs123skgj
brute force match at text position 1
brute force match at text position 14
kmp match at text position 1
kmp match at text position 14
bm match at text position 1
bm match at text position 14
brute force time 1.6652ms
kmp time 1.8188ms
bm time 1.1706ms
```

图1,2 两种输入方法的结果截图（记字符串的第一个元素位置为1）

4. 实验结果

各方法耗时及作图如下，其中时间单位为ms：

n	brute	kmp	bm
500	0.0023	0.0028	0.0131
1000	0.0044	0.0053	0.0152
5000	0.033	0.0426	0.0271
10000	0.0313	0.0504	0.0379
50000	0.28	0.3152	0.1624
100000	0.5868	0.6976	0.3345
500000	1.9155	2.466	1.4178
1000000	3.8173	4.7626	2.3452

n	brute	kmp	bm
5000000	17.5135	24.2173	11.4565
10000000	33.9867	52.1365	26.14

表1 m=5 时各方法数据

n	brute	kmp	bm
500	0.0026	0.0031	0.0406
1000	0.0069	0.008	0.0195
5000	0.036	0.0373	0.0381
10000	0.0408	0.0484	0.061
50000	0.2471	0.2456	0.1357
100000	0.5628	0.5466	0.2274
500000	1.7169	2.2615	0.7239
1000000	3.784	4.7492	1.4532
5000000	16.9465	23.7302	7.602
10000000	47.6013	47.9529	14.6522

表2 m=10时各方法数据

n	brute	kmp	bm
500	0.0026	0.0034	0.0143
1000	0.0069	0.0079	0.0167
5000	0.017	0.023	0.0234
10000	0.0349	0.046	0.0269
50000	0.191	0.2563	0.0909
100000	0.3572	0.4495	0.1102
500000	1.7569	2.2641	0.0502

n	brute	kmp	bm
1000000	3.8409	4.6515	1.1755
5000000	16.6253	24.4479	5.2139
10000000	34.494	47.7694	16.4037

表3 m=20时各方法数据

n	brute	kmp	bm
500	0.0028	0.0037	0.0264
1000	0.0041	0.0055	0.0128
5000	0.0166	0.0233	0.0207
10000	0.0316	0.0459	0.025
50000	0.2676	0.3406	0.0861
100000	0.4454	0.4545	0.09
500000	2.2739	2.332	0.4278
1000000	3.4498	5.1286	0.08802
5000000	17.2677	23.7216	4.2995
10000000	34.8483	48.7031	9.6125

表4 m=50时各方法数据

m	brute	kmp	bm
5	33.9867	52.1365	26.14
10	47.6013	47.9529	14.6522
20	34.494	47.7694	16.4037
50	34.8483	48.7031	9.6125
100	35.5559	48.1713	7.9538
500	34.3663	47.6152	7.4636

m	brute	kmp	bm
1000	37.2239	54.6514	8.7328
10000	33.7103	47.801	8.3115

表5 n=10000000时各方法数据

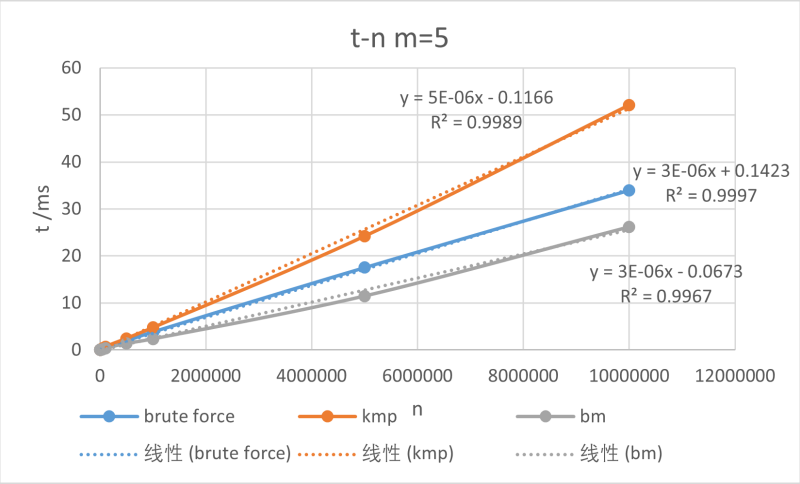

图3 m=5 时的 t-n 图

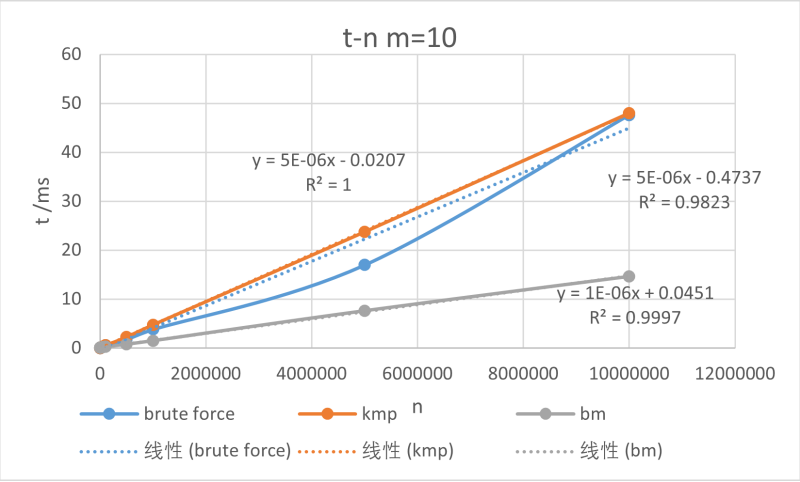

图4 m=10 时的 t-n 图

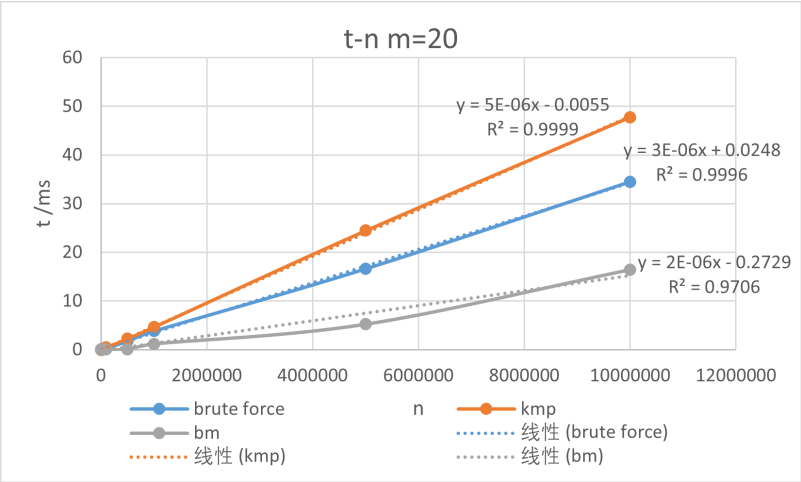

图5 m=20 时的 t-n 图

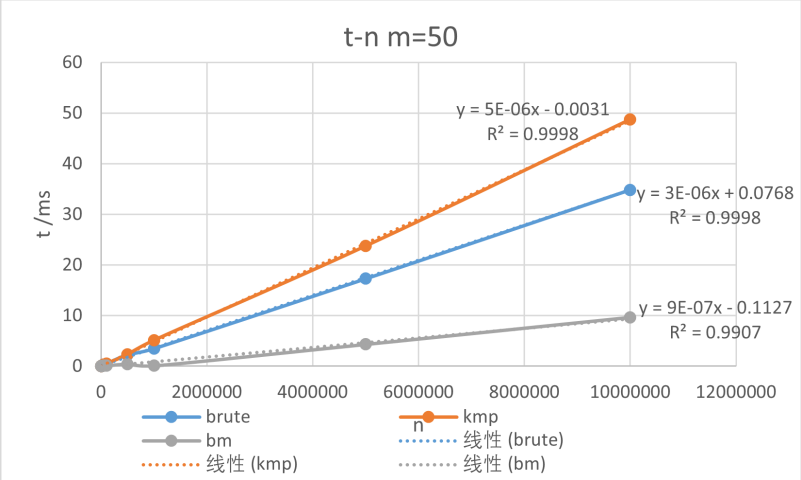

图6 m=50 时的 t-n 图

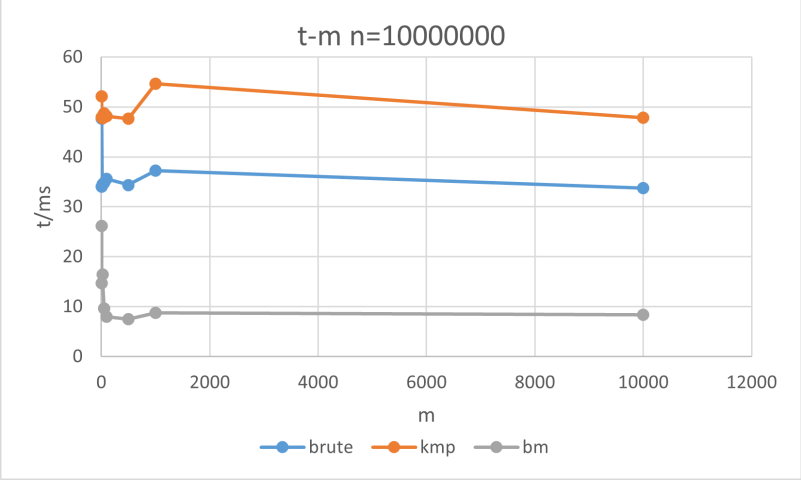

图7 n=10000000 时的 t-m 图

从各图中分析可以发现，各方法和n具有较好的线性关系，R^2值都较高，较为符合各方法时间复杂度正比与n的结论。而各方法对于m的表达式较为复杂，m较小时呈现先下降而后略上升的趋势，当m较大时呈下降的趋势，这和理论并不完全一致。对于bm算法，由于时间复杂度为$\Omega(\frac{n}{m}+m^2+|\Sigma|)$，

故确实存在先上升后下降的趋势，和曲线较为符合。而kmp和暴力算法的耗时应该随着m的增加而变大，而实验结论和此不符合，可能与随机数生成方式和常系数有关。

在本次实验中还发现使用str_rand.exe随机产生的字符串时，kmp算法总是比暴力算法慢。而对于手动输入的场景，有时kmp快，有时暴力算法快。这和理论不符合，这可能和本次使用的字符以及随机数的生成方式有关。