数据结构实验报告

软件学院 软件工程 4 班 2113850 李鹏

——实验题目：KMP 和 BM 算法性能分析

一、整体思路及算法：

为实现字符串的搜索匹配，设计 KMP 算法和 BM 算法，在输入相同文本条件（相同主串和模式串）下比较两种算法的比较次数和时间性能，对比分析两种算法的性能。

① 根据不同区间长度随机生成字符串；
② 比较次数通过每比较一次，计数变量 KMP_count，BM_count 加 1，分析主串与模式串的比对次数；
③ 运行时间使用 clock（）函数，记录程序时间，分析程序运行所用时间。

1. BF 算法：

在暴力（BF 算法）查找模式字符串时，只要有一个不匹配，主串的下标回溯到 i-j+1（主串的下一个元素），模式字符串的下标就会回溯到第一个字符。该算法不利用前面匹配成功的经验，会使时间复杂度非常高，接近 O(m*n)。

2. KMP 算法：

KMP 算法中，在主串中查找模式字符串时，基于已经匹配的经验，只需要回溯模式字符串的下标，避免主串下标的移动，以此来提高查找模式字符串的效率。

在匹配到模式字符串的第 j 个字符不相同时，就会让模式字符串中与后缀字符串相同的前缀字符串与主串的后缀字符串对齐，因此需要在匹配之前对模式字符串进行预处理。以 next 数组来记录匹配到模式字符串的第 j 个失配的字符时与后缀字符相同的前缀字符串的最大长度。一旦发生失配，j 就回溯到 next[j]的位置。

3. BM 算法：

从模式字符串的最后一个开始，从左往右开始比较。

坏字符规则：当发生不匹配时，主串和模式字符串不匹配的字符称为坏字符。此时，将左边离模式字符串坏字符最近的与主串相同的字符与主串坏字符对齐。即将模式字符串向右移动$L=j-X_j$（模式字符串中与主串坏字符最近的字符下标）的距离或者主串向左移 L 的距离。

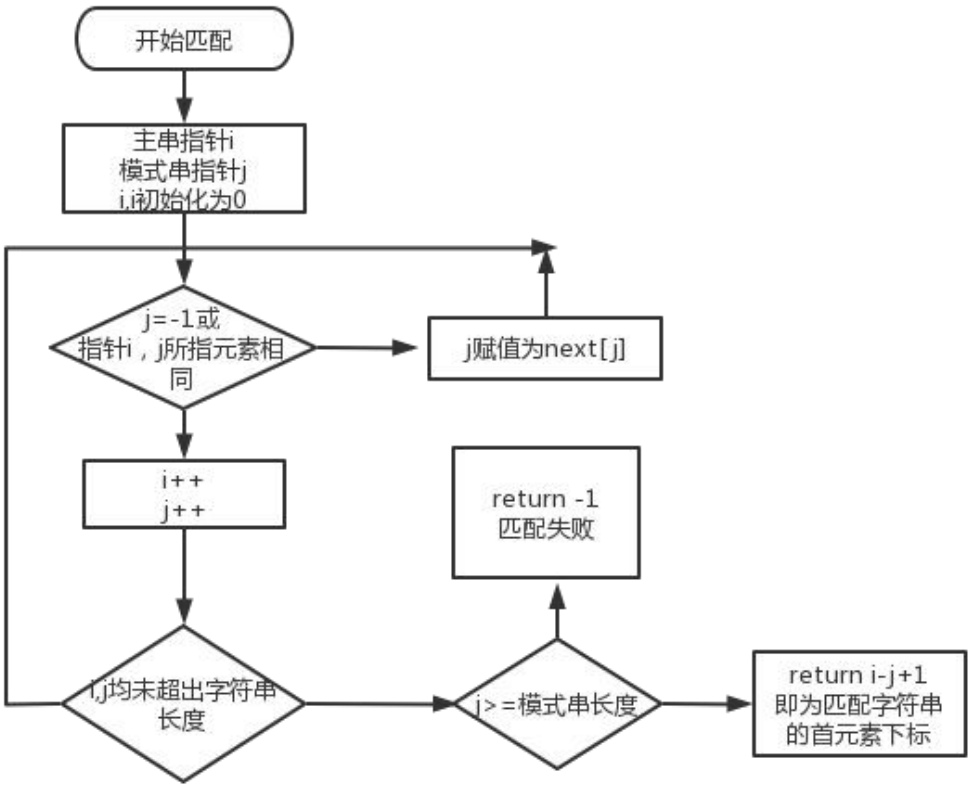
好后缀规则：当发生失配时，将左边离坏字符最近的与好后缀的子串相

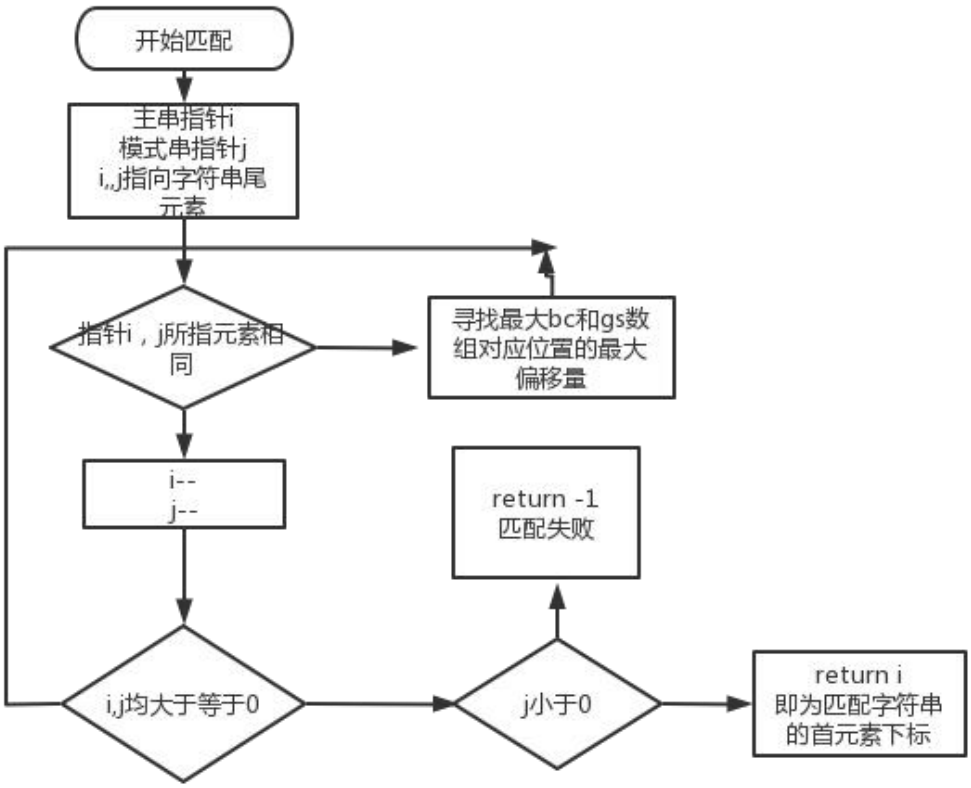
同的前缀与已成功匹配的主串对齐。模式字符串向右移动距离为 $L=j-X_j$（前缀的第一个字符的下标）。

坏字符规则和好后缀规则是独立计算的，具体使用哪个规则，是通过对比两个规则计算出来需要往后移动的位置数的大小，取其中最大的。从该原理可以看出，BM 每次匹配前也需要做预处理，需要针对模式串 P 分别生成一个坏字符辅助数组和好后缀辅助数组，它们分别存放着各自规则下模式串 P 的字符发生失配时，需要相应地向右移动的位置数，

模式字符串向右移动的距离应为坏字符规则和好后缀规则移动距离的最大值，这样既可以避免移动过程中遗失可能的匹配有避免模式字符串向右移动的距离为负数，即向左移动。

二、流程图：





三、部分关键代码分析：

1. KMP 算法中 next 数组

获取模式串中个元素位置前的最大公共字符子串长度，当此位置元素与主串失配时，模式串搜索洗标 j 回退到 next[j]。

```
void getnext(string t, int * nextval)
{
    int i = 0, j = -1;
    nextval[0] = -1;
    while (i < t.length())
    {
        if (j == -1 || t[i] == t[j])
        {
            j++, j++;
            if (t[j] == t[j])//当两个字符相同时，就跳过
                nextval[i] = nextval[j];
            else
                nextval[i] = j;
        }
        else
            j = nextval[j];
    }
}
```

2. BM 算法中坏字符 bc 数组

字符串失配时模式串指针偏移量

```
void getbc(string p, vector<int> bc)
{
    for (int i = 0; i < 256; i++)
        bc[i] = -1;
    for (int i = 0; i < p.length(); i++)
    {
        bc[p[i]] = i;
    }
}
```

3. BM 算法中的好后缀 gs 数组

字符串失配时模式串指针偏移量

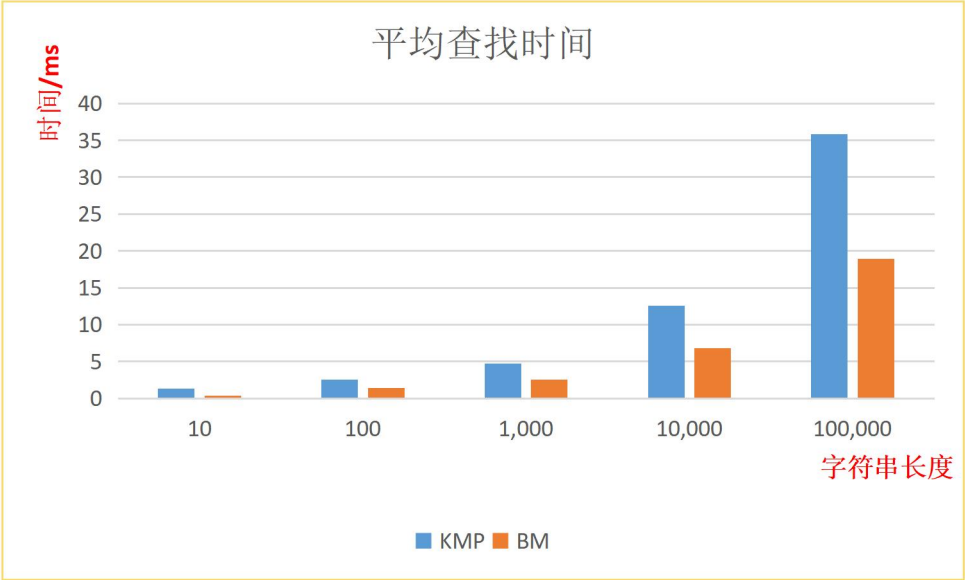
```
void getgs(string t, vector<int> suffix, vector<bool> prefix)
{
    int n = t.length();
    for (int i = 0; i < n - 1; i++)
    {
        suffix[i] = -1;
        prefix[i] = false;
    }

    for (int i = 0; i < n - 1; i++)
    {
        int j = i;//从第一个字符开始遍历，t[j]
        int k = 0;//最后一个字符的变化，对应下面的t[n - 1 - k]
        while (j >= 0 && t[j] == t[n - 1 - k])//和最后一个字符对比，相等则倒数第二个
        {
            j--;
            k++;
            suffix[k] = j + 1;//如果k=1，则是一个字符长度的后缀对应匹配位置的值
        }
        if (j == -1)//说明有前缀字符对应
            prefix[k] = true;
    }
}
```

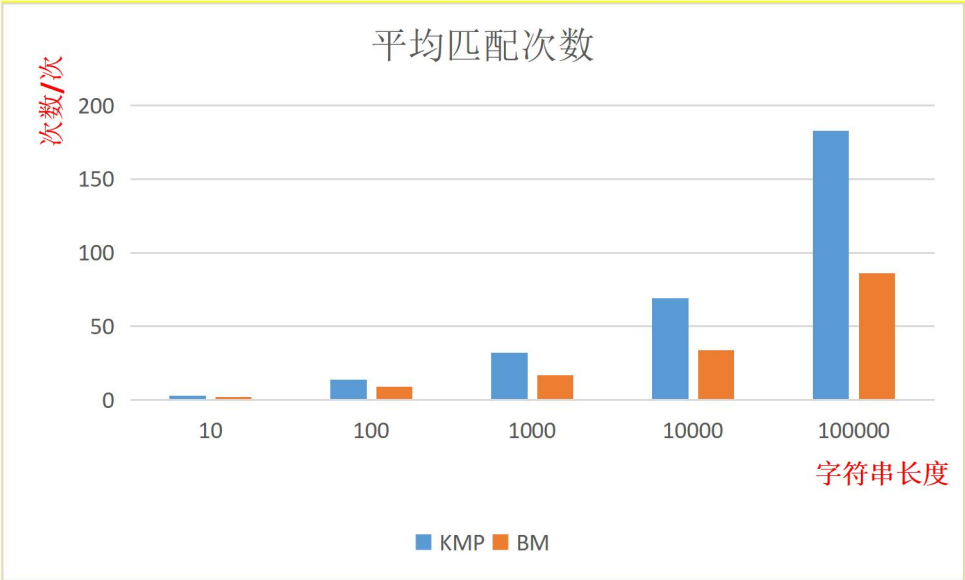
四、样例测试：

样例类型	样例输入	KMP 比较次数	KMP 耗时(ms)	BM 比较次数	BM 耗时(ms)
串中字符只包含 0 和 1	01001011001111000011 0011	20	0.038	12	0.785
	11001110001101110100 11101	27	0.042	16	0.835
	11011011000011110010 0001	24	0.035	13	0.220
	10001000111011101 011	22	0.061	13	0.355
	01010101010101010 1010	6	0.021	5	0.312
串中字符包括 10 种字符	/;<=?>@{|}~ {|}	17	0.374	6	0.512
	/;<=?>@rst~ @rs~	20	0.032	3	0.421
	+-*/,.%$###!123 ##!12	23	0.764	8	0.497
	(**+/-/-[]]]=>^]]=》	28	0.454	4	0.100
	[/]_$!~``:;??asdd $!``	33	0.586	5	0.141

串中包含ASCII 32~126(共95个)字符	qwertyuiopasdfghjklzxcvbnm0147852369!"#$%&•（ ）*+ ,-./:;<=>?@[]\^_'QWERTYUIOPLKJHGFDSAZXCVBNM{}\|~ 369!"#$%	74	0.358	13	0.52
	QWERTYUIOPLKJHGFDSAZXCVBNM{}|~!"#$%&•（）*+,-./:;<=>?@[]\^_'uiopasdfghjklzxcvbnm0147852369qwerty qwerty	190	0.853	4	0.320
	+ ,-./:;<=>?@[]\^_'uiopasdfghjklzxcvbnm0147852369qwertyQWERTYUIOPLKJHGFDSAZXCVBNM{}|~!"#$%&•（）* 369qw	97	0.401	15	0.357
	M{}|~!"#$%&• （ ）*HGFDSAZXCVBN+ ,-./:;<=>?@[]\^_'qwertyQWERTYUIOPLKJ'uiopasdfghjklzxcvbnm0147852369 >?@[]\^_'qwe	92	0.450	16	0.286
	@[]\^_'qwertyQWERTYUIOPLKJ'uiopasdfghjklzxcvbnm0147852369M{}|~!"#$%&• （ ）*HGFDSAZXCVBN+,-./:;<=>? VBN+, .,.,.-./:;<=	193	0.145	5	0.144

随机字符模式串位于主串中间位置时，平均性能分析：





测试结果分析：

根据实验结果来看：

1. 在比较次数上，BM 算法在搜索主串中的模式串的比较次数少于 KMP 算法的比较次数，特别是在主串中含有的字符种类更多，长度更长或连续重复字符较多时，二者的比较次数差别明显，BM 性能更好。

2. 在程序运行所需时间方面，两种算法在大多数测试样例中耗时接近。在部分测试样例中 BM 算法耗时要少于 KMP 算法，与理论相符。根据分析，若主串较短或模式串在主串较靠前的位置时，二者大概率用时接近，当主串较长且模式串在主串较靠后的位置时，BM 算法耗时更少，体现出 BM 算法的优越性。

3. 结论：

无论文本串的大小如何，ＢＭ 算法的平均查找时间最短。当主串为不同长度时，不管模式串在主串中的匹配是否成功，或者在主串中匹配成功的位置（头部、中部、尾部 ）如何，ＢＭ算法匹配次数绝大部分是最少的。 综上比较，ＢＭ算法的平均查找时间最短、匹配次数少，说明ＢＭ所花时间短，速度也最快。 由于ＢＭ算法是从右向左的把模式同文本做比较，当模式符号做比较的文本符号在模式中没有 出现时，模式可以在这个文本符号之后移位多个位置。由于它能避免不必要的回溯，所以它的速度较快。

五、实验总结：

通过本次字符串搜索匹配的实验，我初步体会到了程序设计中性能的重要性。在之前的编程练习中，我通常只追求能够通过测试样例，达到实现程序的目的，几乎从未考虑程序性能、复杂度的问题。在以后的编程练习中，我将会逐渐融入优化程序的思想，降低复杂度，提高程序性能。