实验四：串匹配算法

PB19030888张舒恒

实验设备和环境

PC一台，Win11企业版操作系统，gcc 9.1.0编译器，Clion 2021.2.2代码编辑器，Excel绘图工具

实验内容

1.KMP算法

2.Rabin-Karp算法

实验要求

代码限制C/C++，建立根文件夹80-张舒恒-PB19030888-project4，在根文件夹下建立本实验报告，ex1和ex2实验文件夹，每个实验文件夹中建立3个子文件夹：input文件夹：存放输入数据，src文件夹：源程序，output文件夹：输出数据。

实验步骤及方法

KMP算法

1.文件输入输出

采用绝对路径的输入输出流

```
    string from =
"C:/Users/ASUS/Desktop/connecting/vscode/AL4/ex1/input/4_1_input.txt";
    string dest =
"C:/Users/ASUS/Desktop/connecting/vscode/AL4/ex1/output/result.txt";
    string time =
"C:/Users/ASUS/Desktop/connecting/vscode/AL4/ex1/output/time.txt";

    file_in.open(from);
    file_out.open(dest);
    time_out.open(time);
```

2.求解π数组

置$next[0]=-1$，如果$substr[j+1]$和$substr[i]$匹配则继续匹配下一个字符，如果不匹配则j返回到$next[j]$

```
void getNext(const string &substr, vector<int> &next){

    next.clear();
    next.resize(substr.size());
    int j = -1;
    next[0] = -1;
    for(int i = 1; i < substr.size(); ++i){
        while(j > -1 && substr[j + 1] != substr[i])
            j = next[j];
```

```
        if(substr[j + 1] == substr[i])
            ++j;
        next[i] = j;
    }
}
```

3.KMP算法

调用$getNext$计算出π数组，如果$substr[j+1]$和$substr[i]$匹配则继续匹配下一个字符，如果不匹配则j返回到$next[j]$，如果j已经到达模式串最后位置则匹配成功，并将源串起始位置记录下来

```
void kmp(const string &str, const string &substr, vector<int> &next){

    vector<int> find;
    int cnt = 0;
    getNext(substr, next);
    int j = -1;
    for(int i = 0; i < str.size(); ++i){
        while(j > -1 && substr[j + 1] != str[i])
            j = next[j];
        if(substr[j + 1] == str[i])
            ++j;
        if(j == substr.size() - 1){
            find.push_back(i-substr.size()+1);
            ++cnt;
            j = next[j];
        }
    }
    file_out << cnt << endl;
    for(auto i:next)
          file_out << i + 1 << " ";
    file_out << endl;
    for(auto i:find)
          file_out << i + 1 << " ";
    file_out << endl << endl;
}
```

4.打印运行时间，画出时间曲线并分析

Rabin-Karp算法

1.文件输入输出

采用绝对路径的输入输出流

```
    string from =
"C:/Users/ASUS/Desktop/connecting/vscode/AL4/ex2/input/4_2_input.txt";
    string dest =
"C:/Users/ASUS/Desktop/connecting/vscode/AL4/ex2/output/result.txt";
    string time =
"C:/Users/ASUS/Desktop/connecting/vscode/AL4/ex2/output/time.txt";

    file_in.open(from);
    file_out.open(dest);
    time_out.open(time);
```

2.Rabin-Karp算法

通过$h = h * d\%q$; 迭代$m-1$次计算出d^{m-1}，再计算出模式串的hash值p和源串起始位置开始的子串的hash值t。如果p和当前的t相等则截取源串的子串与模式串仔细匹配，若匹配成功则记录匹配位置，若匹配不成功则伪命中次数+1。通过迭代的方法计算从以下一个位置开始的子串的hash值。

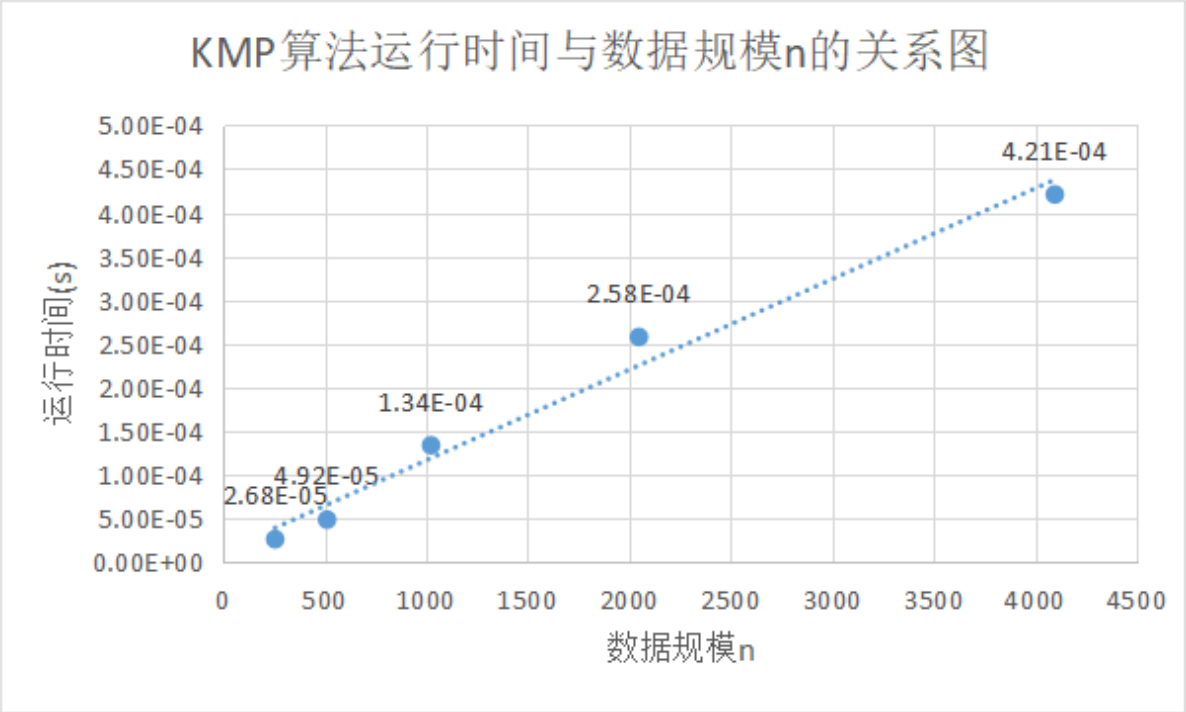
```
int RABIN_KRAP_MATCHER(string T, string P, int d, int q, int &false_count,
vector<int> &find){

    find.clear();
    false_count = 0;
    int n = T.length(), m = P.length();
    int p = 0, t = 0;
    int h = 1;
    for(auto i=0;i<m-1;i++){
        h = h * d % q;
    }
    for(auto i=0;i<m;i++){
        p = (d * p + P[i]) % q;
        t = (d * t + T[i]) % q;
    }
    for(auto s = 0; s <= n - m; s++){
        if(p == t){
            if(T.substr(s, m) == P)
                find.push_back(s);
            else
                false_count++;
        }
        t = (d * (t - T[s] * h % q) + T[s + m]) % q;
        if(t<0)
            t += q;
    }
}
```

3.打印运行时间画出时间曲线并分析

实验结果分析

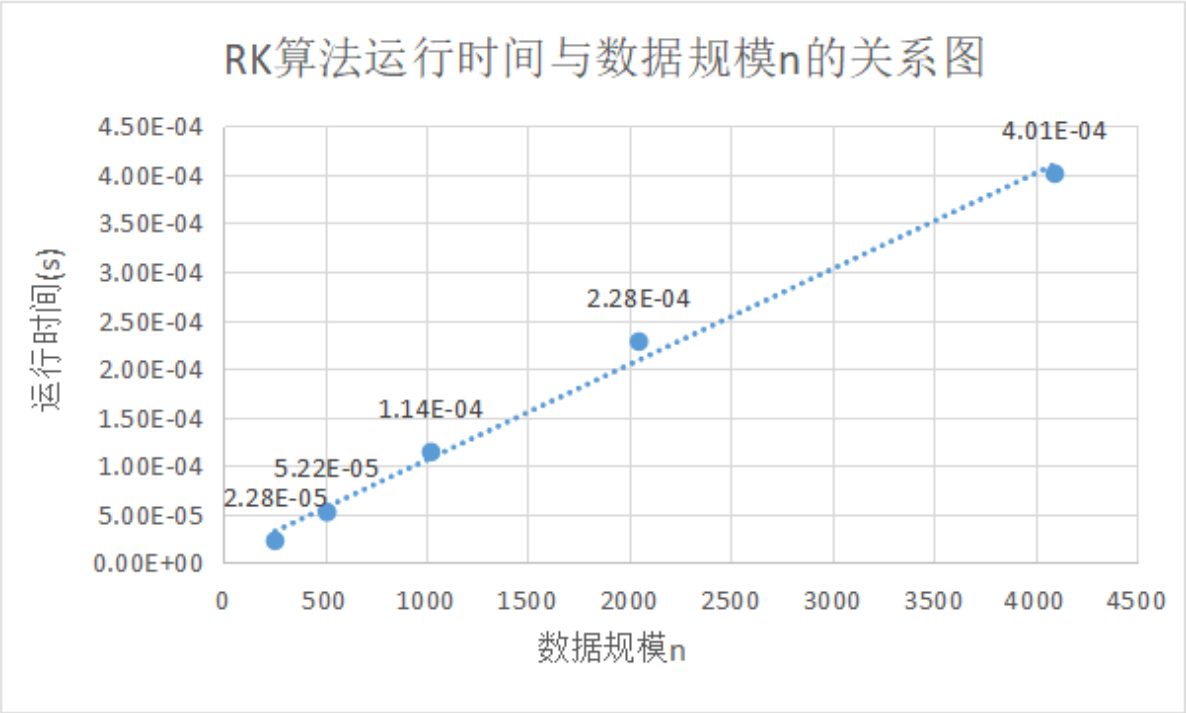
KMP算法

画出时间曲线，用线性函数进行拟合，结果基本符合一次函数增长模型，n=2048时有误差，但误差在可接受范围内，主要原因可能是数据分布的影响，实际时间复杂度和理论时间复杂度近似相同，均为$O(n)$

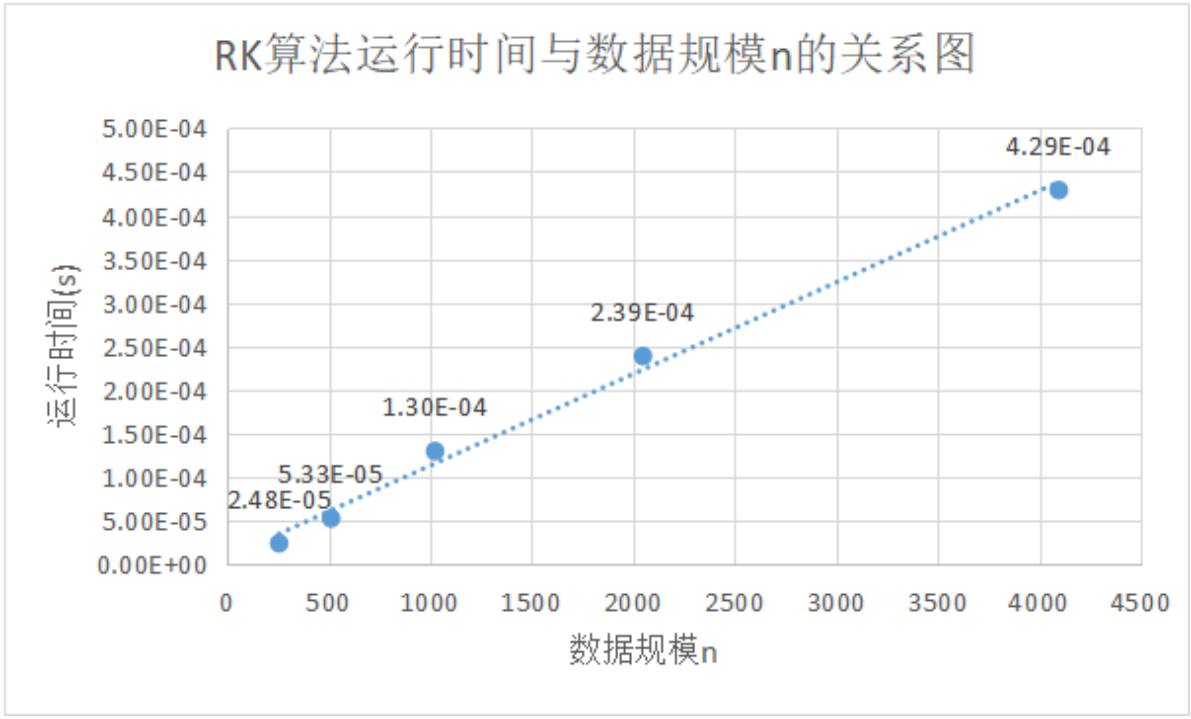


Rabin-Karp算法

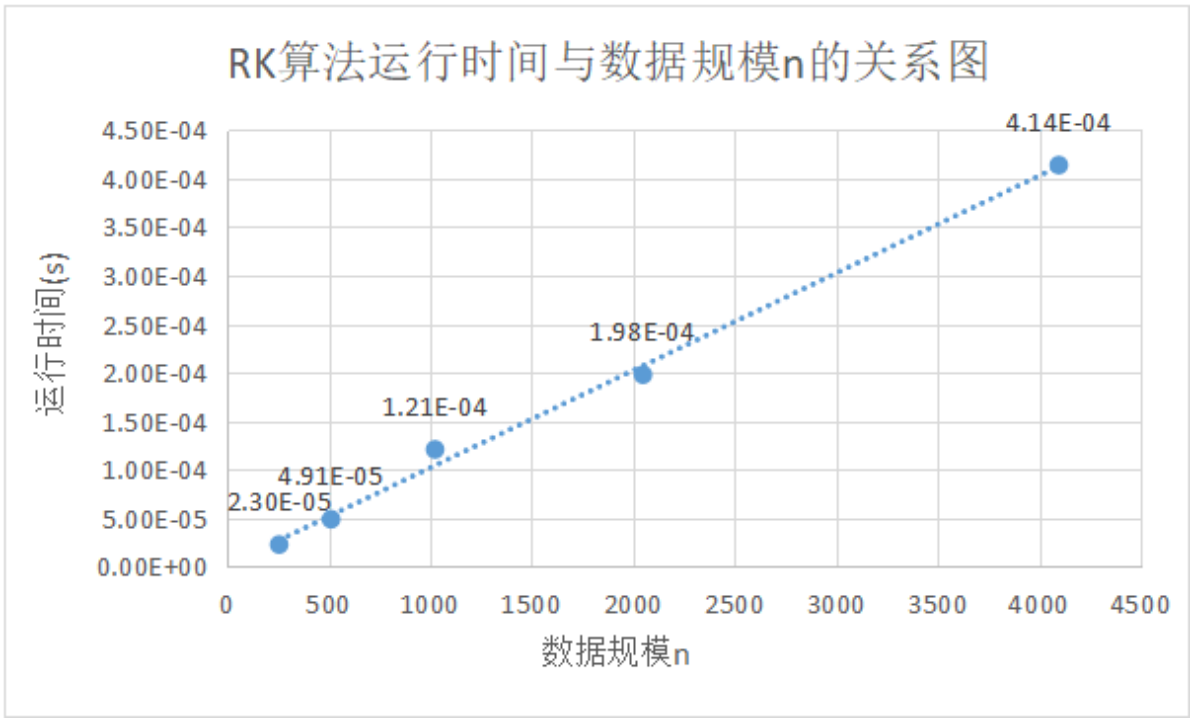
分别画出$(d,q)=(2,13),(2,1009),(10,13),(10,1009)$情况下的时间曲线，用线性函数进行拟合，结果基本符合一次函数增长模型，所以实际时间复杂度和理论时间复杂度近似相同，均为$O(n)$



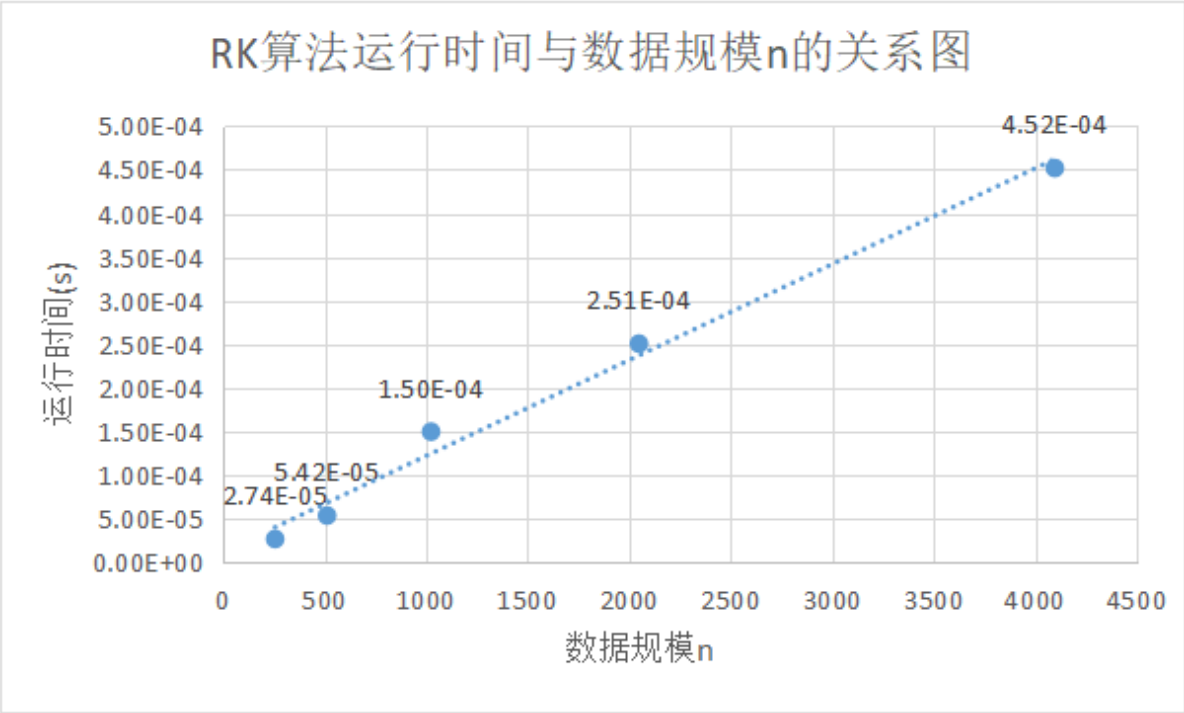
$(d,q)=(2,13)$时运行时间图



$(d, q) = (2, 1009)$时运行时间图



$(d, q) = (10, 13)$时运行时间图



$(d, q) = (10, 1009)$时运行时间图

实验总结

通过本次实验我基本掌握了串匹配相关算法