算法基础实验四报告

> PB19051035周佳豪

实验设备与环境

> C++11

实验内容

KMP算法

- 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。
- (n, m)共有5组取值，分别为: (28，23), (29，24), (210,25), (211,26), (212,27).
- 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。

Rabin-Karp算法

- 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
- (n, m)共有5组取值，分别为: (28，23), (29，24), (210,25), (211,26), (212,27)。
- 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。

实验过程与结果

KMP

- 关键函数`compute_prefix()`与`kmp_matcher()`

```
vector<int> compute_prefix(string P)
{
    //   cout<<"P:"<<P<<endl;
    int m = P.length();
    vector<int> pi(m);
    pi[0] = 0;
    int k = 0;
    for (int q = 2; q <= m; q++)
    {
        while (k > 0 && P[k] != P[q - 1])
            k = pi[k - 1];
        if (P[k] == P[q - 1])
            k = k + 1;
        pi[q - 1] = k;
        // printf("pi[%d]=%d\n",q-1,k);
    }
    // cout<<"compute_prefix is over"<<endl;
    return pi;
}

void kmp_matcher(string T, string P, ofstream &outfile, ofstream &timefile)
{
    int n = T.length();
    int m = P.length();
```

```
    LARGE_INTEGER t1, t2, tc;
    double time;
    QueryPerformanceFrequency(&tc);
    QueryPerformanceCounter(&t1);
    vector<int> pi = compute_prefix(P);
    QueryPerformanceCounter(&t2);
    time = double(t2.QuadPart - t1.QuadPart) / (double)tc.QuadPart;
    timefile << "m= "<<m<<",m_time = " << time * 1000 << "ms";
    /*
    for(int i=0;i<pi.size();i++)
        cout<<pi[i]<<' ';*/
    vector<int> location;
    location.clear();
    //outfile << "index:";
    // cout<<endl;
    int q = 0;
    QueryPerformanceFrequency(&tc);
    QueryPerformanceCounter(&t1);
    for (int i = 1; i <= n; i++)
    {
        //cout<<"q="<<q<<endl;
        while (q > 0 && P[q] != T[i - 1])
            q = pi[q - 1];
        if (P[q] == T[i - 1])
            q = q + 1;
        if (q == m)
        {
            location.push_back(i-m+1);
            //outfile << to_string(i - m + 1) + ' ';
            //cout<<i-m+1<<endl;
            // cout<<"match is ok"<<endl;
            q = pi[q - 1];
        }
    }
    QueryPerformanceCounter(&t2);
    time = double(t2.QuadPart - t1.QuadPart) / (double)tc.QuadPart;
    timefile << ",n= "<<n<<",n_time = " << time * 1000 << "ms" << endl;
    outfile <<to_string(location.size())<<endl;
    for (int i = 0; i < pi.size(); i++)
        outfile << to_string(pi[i]) + ' ';
    outfile<<endl;
    for(int i = 0; i <location.size(); i++)
        outfile << to_string(location[i])+" ";


    outfile << endl<<endl;
}
```

需要注意字符串的起始位置是0，其余均和课本伪代码思路相同。location记录匹配的开始位置。分开记录计算前缀函数与匹配的时间，整个kmp算法的时间即为二者的时间之和。

- 实验结果与时间分析
  - 实验结果

```
2
0 0 0 1 2 1 1 2
72 189
```

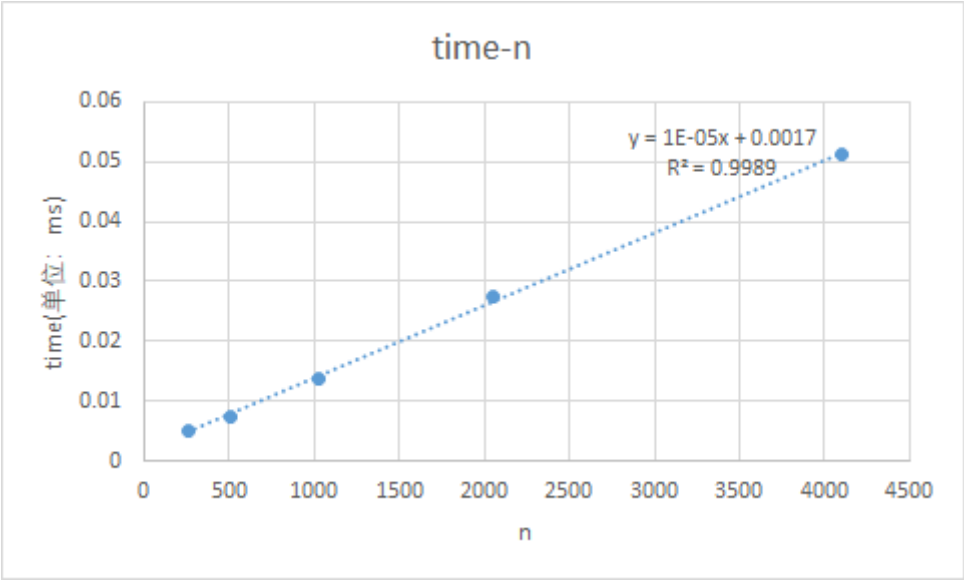
```
2
0 1 0 0 1 0 0 0 1 2 3 1 0 1 2 3
234 295

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
441 698

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 1 2 1 1 2 3 0
928 1908

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 0 1 0 0 1 0 1 0 1 0 1 2 3 4 4
5 6 0 0 0 0 0 1 2 0 0 0 0 1 0 1 2 3 4 5 6 7 0 1 0 0 1 2 3 0 0 1 0 1 0 0
1 2 0 1 0 1 2 3 0 0 0 0 1 2 0 0 0 0 1 0
1124 2727
```

- 运行时间分析

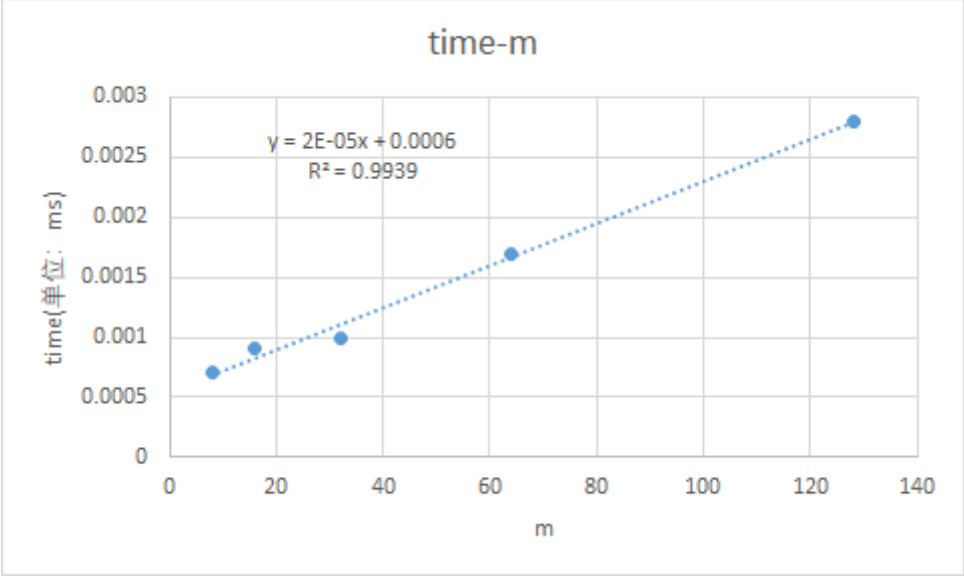
n	time(单位：ms)
256	0.0049
512	0.0075
1024	0.0137
2048	0.0276
4096	0.0511

m	time(单位：ms)
8	0.0007
16	0.0009
32	0.001
64	0.0017
128	0.0028

对匹配进行拟合：



对前缀函数进行拟合：



根据拟合结果可知，匹配所需要的时间与n成正比，计算前缀函数所花费的时间与m成正比，故验证了前缀函数的复杂度说O(m)，实际匹配的复杂度为O(n),实际复杂度和理论复杂度相同.

Rabin-Karp

- 关键函数 `rabin_karp_matcher()`

```
map<pair<int, int>, vector<int>> result;
map<pair<int, int>, vector<int>> matchs;
void rabin_karp_matcher(string T, string P, int d, int q)
{
    int n = T.length();
    int m = P.length();
    int h=d % q;
    for(int i=1;i<=m-2;i++)
    {
        h = (h*(d%q))%q;
    }
    int p = 0;
    vector<int> t(n);
    t[0] = 0;
    for (int i = 1; i <= m; i++)
    {
```

```
        p = (d * p %q+ P[i - 1]%q) % q;
        t[0] = (d * t[0]%q + T[i - 1]%q) % q;
    }

    int sum = 0;
    vector<int> match;
    for (int s = 0; s <= n - m; s++)
    {
        if ((p - t[s]) % q == 0)
        {
            if (P == T.substr(s, m))
            {
                match.push_back(s + 1);
            }
            else
                sum++;
        }
        if (s < n - m)
        {
            t[s + 1] = (d * (t[s] - T[s] * h)%q + T[s + m]%q) % q;
        }
    }

    result[make_pair(n, m)].push_back(sum);
    matchs[make_pair(n, m)]=match;
}
```

其中，用result存储伪命中次数，matchs存储匹配的位置，考虑到h若用书上的代码会溢出，故使用循环处理。其余均与书上伪代码思路相同。

- 实验结果与时间分析
  - 实验结果

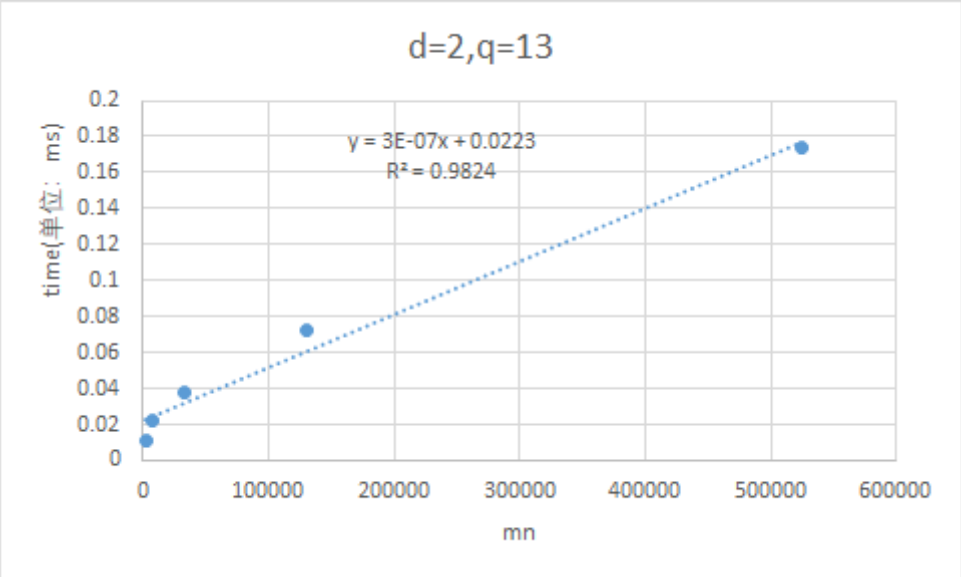
```
2
15 0 15 0
79 197

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
```

  - 运行时间分析

    d=2,q=13

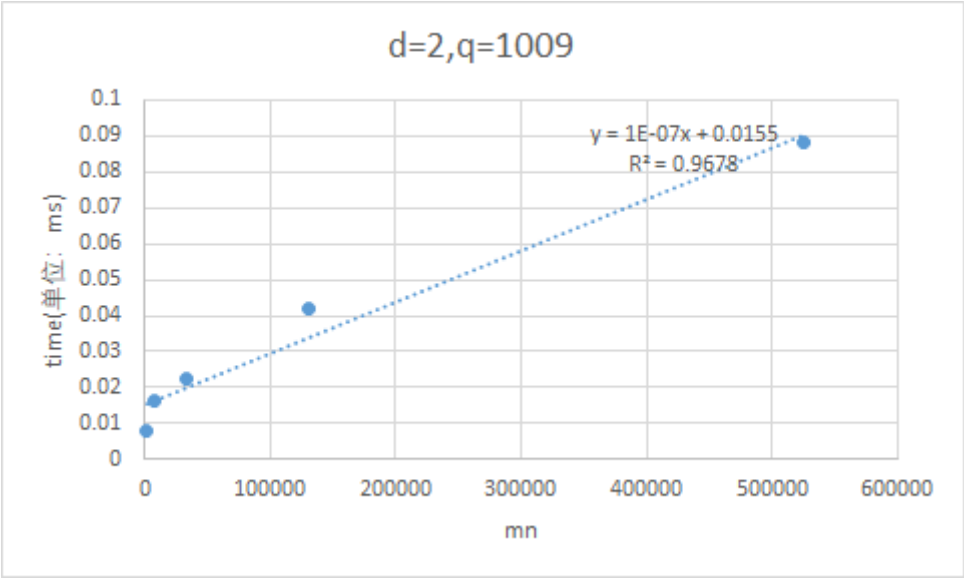
mn	time(单位：ms)
2048	0.0115
8192	0.0224
32768	0.0373
131072	0.0725
524288	0.174

拟合结果为：



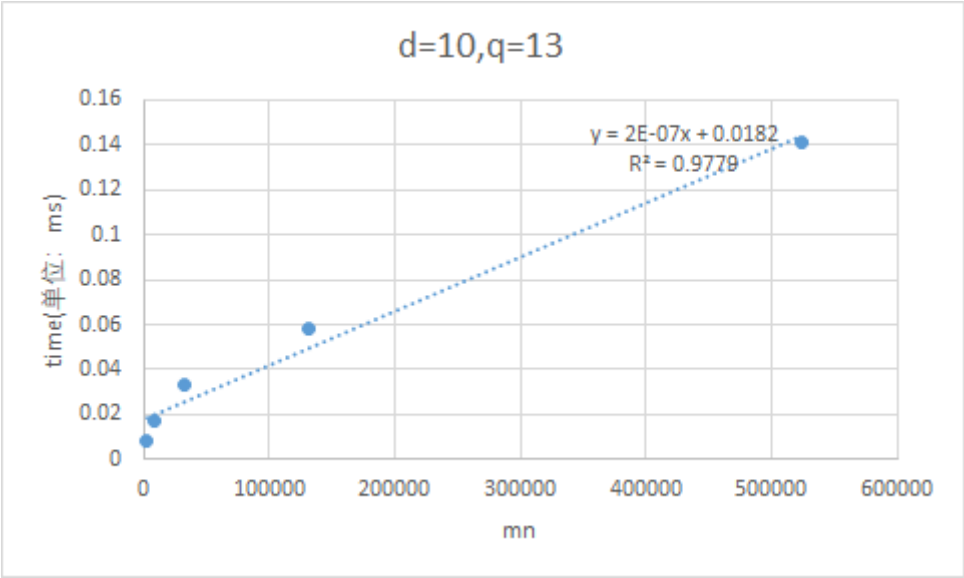
d=2,q=1009

mn	time(单位：ms)
2048	0.008
8192	0.0163
32768	0.0225
131072	0.042
524288	0.0881



d=10,q=13

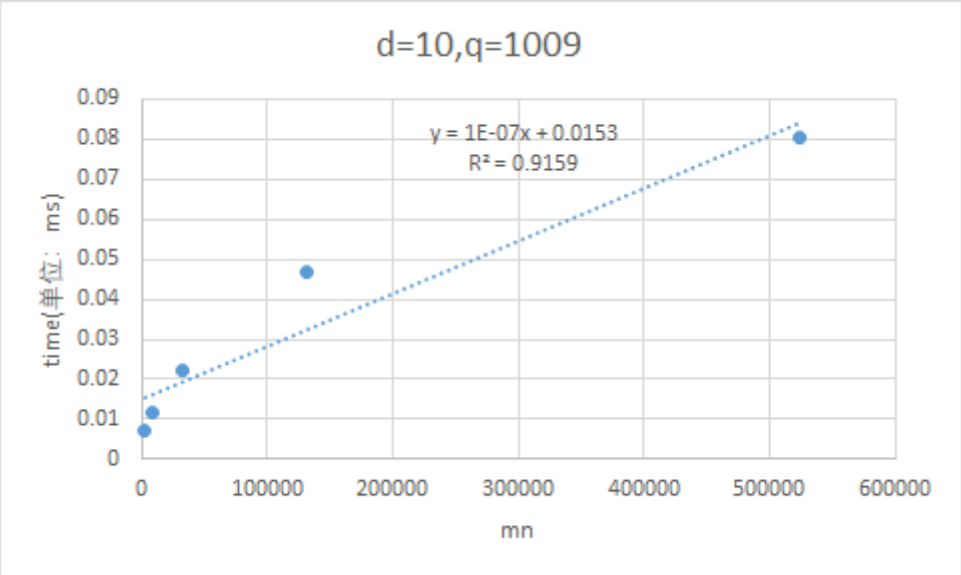
mn	time(单位：ms)
2048	0.0083
8192	0.017
32768	0.0335
131072	0.0583
524288	0.1414



d=10,q=1009

mn	time（单位：ms)
2048	0.0071
8192	0.0119
32768	0.022
131072	0.0466
524288	0.0805

拟合结果为：



根据拟合结果可知，运行时间与mn线性相关，可得实际RK算法复杂度为O（mn），与理论时间复杂度相同。同时根据时间数据可知，同等mn、d条件下，q越大，算法运行时间越短。