KMP算法

next数字的求解

C++

```
vector<int> getNext(string s) {

    vector<int> next(s.size());

    int j = 0, i = 1;//j为前缀末尾，i为后缀末尾
    next[0] = 0;
    for (;i < s.size();i++) {
        while (j > 0 && s[i] != s[j]) {//当不匹配的时候
            j = next[j - 1];//j向前回退
        }
        if (s[i] == s[j]) {//如果j匹配
            j++;
        }
        next[i] = j;//j也代表i位置最小相等前后缀的值
    }

    return next;
}
```

Find函数的实现

```
int Find(string haystack, string needle) {
    //找到第一个出现的位置
    //这里返回的是下标
    if (needle.empty()) {
        return 0;
    }

    vector<int> next = getNext(needle);
    int j = 0;//j为前缀末尾，i为后缀末尾
    for (int i = 0;i < haystack.size();i++) {
        while (j > 0 && haystack[i] != needle[j]) {
            j = next[j - 1];
        }

        if (haystack[i] == needle[i]) {
            j++;
        }
```

```
        if (j == needle.size()) {
            //与上面getnum不同的地方
            //needle被遍历完全，说明找到了相应的位置
            return (i - needle.size() + 1);
        }
    }

    return -1;
}
```

所有的出现的位置，则需要进行遍历，并且

如果需要找到所有出现的位置，则需要进行遍历,并且对于find函数进行修改，更改其开始位置。

修改后的find函数

```
int Find(string haystack, string needle, int pos) {

    if (needle.empty()) {
        return 0;
    }

    vector<int> next = getNext(needle);
    int j = 0, i;//j为前缀末尾，i为后缀末尾
    for (int i = pos;i < haystack.size();i++) {
        while (j > 0 && haystack[i] != needle[j]) {
            j = next[j - 1];
        }

        if (haystack[i] == needle[j]) {
            j++;
        }

        if (j == needle.size()) {
            //与上面getnum不同的地方
            //needle被遍历完全，说明找到了相应的位置
            return (i - needle.size() + 1);
        }
    }

    return -1;
}
```

找到所有位置的函数

如果调用系统的find函数

```
std::vector<int> findAll(const std::string& a, const std::string& b) {
    std::vector<int> ret;
    int pos = a.find(b);

    while (pos != std::string::npos) {
        ret.push_back(pos);
        pos = a.find(b, pos + 1);  // 注意这里在开始新搜索前递增pos
    }

    return ret;
}
```

如果自定义调用自定义的Find函数

```
vector<int> findAll(string a, string b) {
    vector<int> ret;
    int pos = Find(a, b);

    while (pos != -1) {
        ret.push_back(pos);
        pos = Find(a, b, pos + 1);
    }

    return ret;
}
```

找出所有位置的优化版

```
vector<int> findAll(string haystack, string needle) {
    if (needle.empty()) {
        return vector<int>(haystack.size(), 0);
    }

    vector<int> ret;
    vector<int> next = getNext(needle);
    int j = 0;  // j为needle的索引

    for (int i = 0; i < haystack.size(); i++) {
        while (j > 0 && haystack[i] != needle[j]) {
```

```
            j = next[j - 1];
        }

        if (haystack[i] == needle[j]) {
            j++;
        }

        if (j == needle.size()) {
            ret.push_back(i - needle.size() + 1);
            j = next[j - 1];  // 为了避免从头开始，从next[j-1]开始
        }
    }

    return ret;
}
```