

KMP

MengChunlei

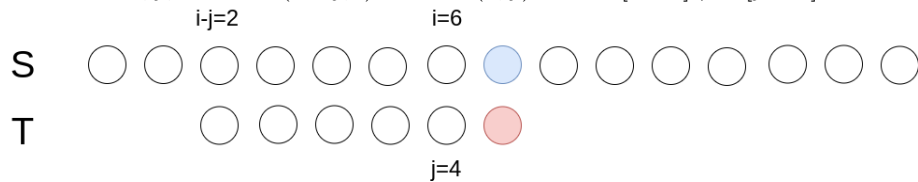
November 1, 2020

1 算法目标

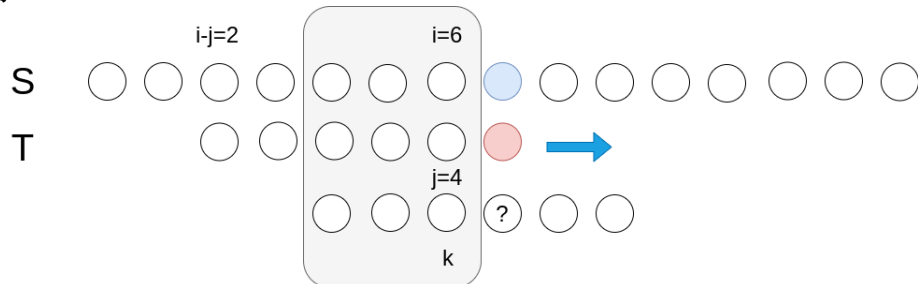
给定字符串 S 和 T (一般 T 会比 S 短), 判断 T 是不是 S 的子串. 如 $S = \text{"aabcaad"} , T = \text{"caa"} ,$ 则 T 是 S 的子串, $T = S_{3,5}, T = \text{SubS}(3, 5)$. 下面用 $\text{SubS}(i, j), \text{SubT}(i, j)$ 表示 S, T 的子串.

2 算法描述

假设对于位置 i, j , 有 $\text{SubS}(i - j, i) = \text{SubT}(0, j)$ 但是 $S[i + 1] \neq T[j + 1]$



这个时候, 算法不是从 $S[i - j + 1] = S[3]$ 的位置和 $T[0]$ 重新开始比较, 而是设法让 T 向右滑动一段距离, 也就是让 $S[i + 1]$ 和 $T[k + 1]$ 来比较. 像下面这个样子:



为了能够直接判断 $S[i + 1]$ 是否等于 $T[k + 1]$, 需要满足上面方框中的部分相等. 那么只需要满足 $\text{SubT}(j - k, j) = \text{SubT}(0, k)$. 可以发现, 这是在串 T 上的一个关系, 跟 S 没有关系. 假设现在有了这个关系, 记作数组 f , 其中 $k + 1 = f[j + 1]$.

比如对于串 "abaabcac" , 它的 f 数组是下面这样:

<i>index</i>	0	1	2	3	4	5	6	7
<i>string</i>	<i>a</i>	<i>b</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>a</i>	<i>c</i>
<i>f</i>	-1	0	0	1	1	2	0	1

有了这个 f 数组，寻找 T 在 S 中第一次出现的算法如下：

Listing 1: FindFirstPos in S

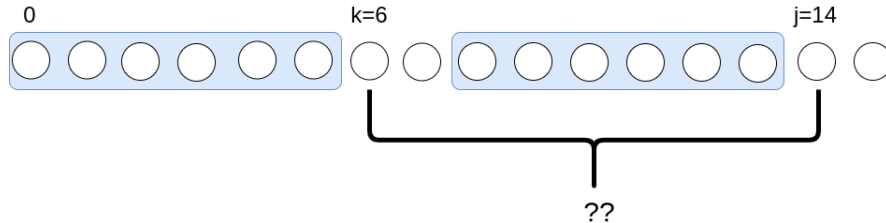
```

1
2 int FindFirstPos(const std::string &S, const std::string &T,
3                 const std::vector<int> &f) {
4     const int lens = static_cast<int>(S.size());
5     const int lent = static_cast<int>(T.size());
6     int i = 0, j = 0;
7     while (i < lens) {
8         if (j == -1 || S[i] == T[j]) {
9             i++;
10            j++;
11        } else {
12            j = f[j];
13        }
14        if (j >= lent) {
15            return i - lent;
16        }
17    }
18    return -1;
19 }

```

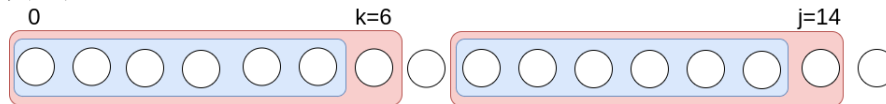
3 f 数组计算

首先 $f[0] = -1$. 假设现在已经计算了 $f[1], f[2], \dots, f[j]$, 令 $k = f[j]$. 现在来看 $f[j+1]$. 由于 $k = f[j]$, 那么有下面的条件满足:



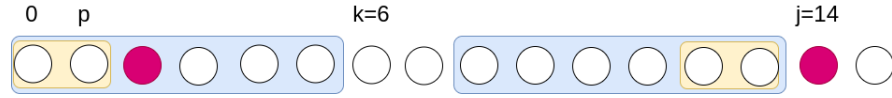
$Sub(0, k-1) = Sub(j-k, j-1)$, 即两个蓝色框里面的串是相等的. 下面来检查 $T[k]$ 和 $T[j]$ 是否相等:

第一种情况, $T[j] == T[k]$, 此时 $f[j+1] = k+1$. 即下面两个粉色框里面的串相等.



第二种情况, $T[j] \neq T[k]$, 那么如下图所示, 需要找到一个位置 p 满足满足两个条件:

- i $Sub(0, p) = Sub(j - (p + 1), j - 1)$, 即两个黄色方框相等
- ii $T[p + 1] = T[j]$



如果要使得第一个条件满足, 那么如下图所示这三个黄色方框里面的串应该相等, 所以 p 应该满足 $p + 1 = f[k]$.



按照这样依次迭代下去, 直到找到一个位置满足条件 2, 那么就找到了 $f[j + 1]$.

下面是计算 f 的代码:

Listing 2: Compute f

```

1 std::vector<int> GetF(const std::string &T) {
2     const int lent = static_cast<int>(T.size());
3     std::vector<int> f(lent);
4     f[0] = -1;
5     int i = 0, j = -1;
6     while (i + 1 < lent) {
7         if (j == -1 || T[i] == T[j]) {
8             f[++i] = ++j;
9         } else {
10            j = f[j];
11        }
12    }
13    return f;
14 }
```

4 进一步思考

上面求出的 f 数组有一些缺陷。我们设 $T = \text{"aaaab"}$, 那么我们求出的 f 如下:

<i>index</i>	0	1	2	3	4
<i>string</i>	a	a	a	a	b
<i>f</i>	-1	0	1	2	3

那么在匹配的时候, 假设匹配到 $T[3]$ 的时候失配了, 那么按照 f 数组, 接下来, 我们将比较 $T[2]$ 和 S 的那个字母, 很明显又失败了 (因为 $T[2], T[3]$ 都是 a), 接着比较 $T[1], T[0]$, 依次都失败了。这就是出现的问题。针对这个问题, 对 F 数组进行以下改进:

Listing 3: Better algorithm to compute f

```

1  std::vector<int> FastF(const std::string &T) {
2      const int lent = static_cast<int>(T.size());
3      std::vector<int> f(lent);
4      f[0] = -1;
5      int i = 0, j = -1;
6      while (i + 1 < lent) {
7          if (j == -1 || T[i] == T[j]) {
8              i++;
9              j++;
10             if (T[i] != T[j]) {
11                 f[i] = j;
12             } else {
13                 f[i] = f[j];
14             }
15         } else {
16             j = f[j];
17         }
18     }
19     return f;
20 }

```

现在对于 T="aaaab" 求出的 f 数组为:

<i>index</i>	0	1	2	3	4
<i>string</i>	a	a	a	a	b
<i>f</i>	-1	-1	-1	-1	3