Extend KMP

MengChunlei

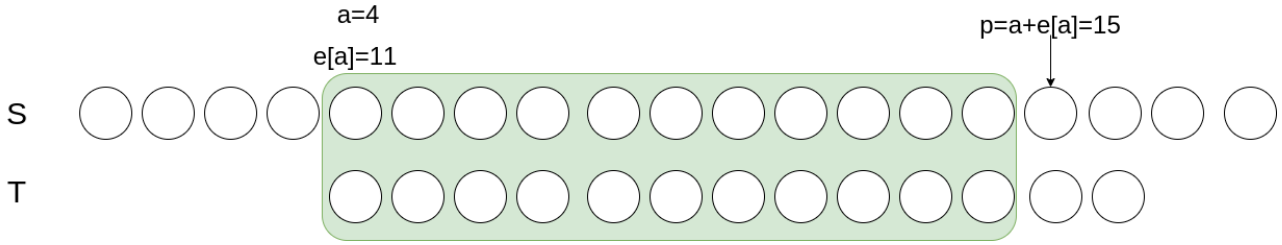
November 1, 2020

1 算法目标

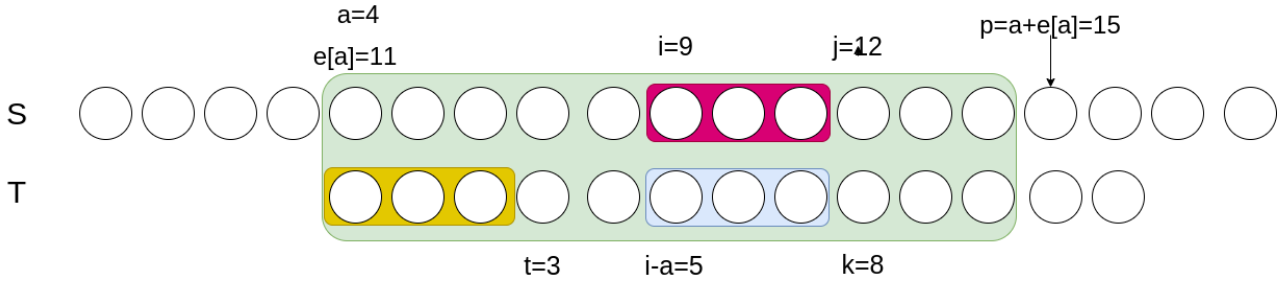
给定字符串 S 和 T，长度分别为 n, m. 用 $SubS(i,j), SubT(i,j)$ 表示 S,T 的子串. 计算一个长度为 n 的数组 e. 其中 $e[i]$ 表示 $SubS(i, n-1)$ 和 T 的最长公共前缀.
可以看出, 如果 $e[i] = m$, 那么说明 S 包含 T. 所以这个算法是 kmp 算法的扩展.

2 算法描述

这个算法依赖于在 T 上的一个辅助数组 f, 其中 $f[i]$ 表示 $SubT(i, m-1)$ 和 T 的最长公共前缀. 假设现在已经计算出了 f.
另外, 这个算法是递进的. 也就是说, $e[i]$ 的计算依赖于 $e[0], e[1], ..., e[i-1]$. 所以, 现在假设已经计算出了 $e[0], e[1], ..., e[i-1]$.
这里定义两个数字 a, p. 其中 $p = max\ \sum_{k=0}^{i-1}(k + e[k])$, a 为使得 p 取最大值时对应的 k.

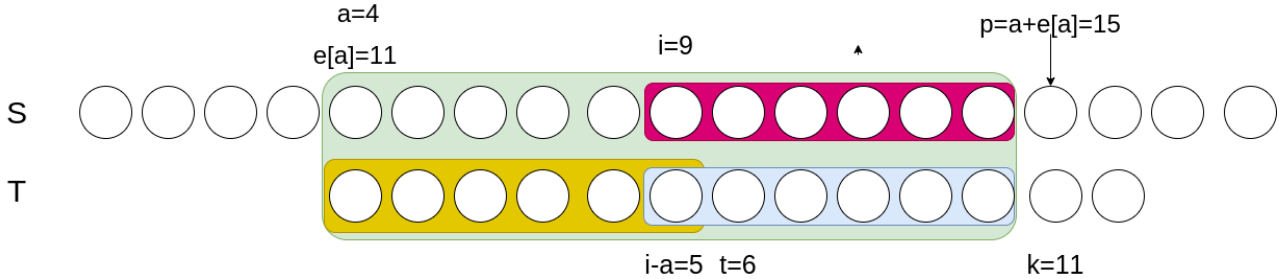


如上图所示, 绿色的上下两部分是相等的. 下面考虑 $e[i]$.
第一种情况, $i + f[i-a] < p$. 如下图, $f[i-a] = 3$. 其中 $t = f[i-a]$



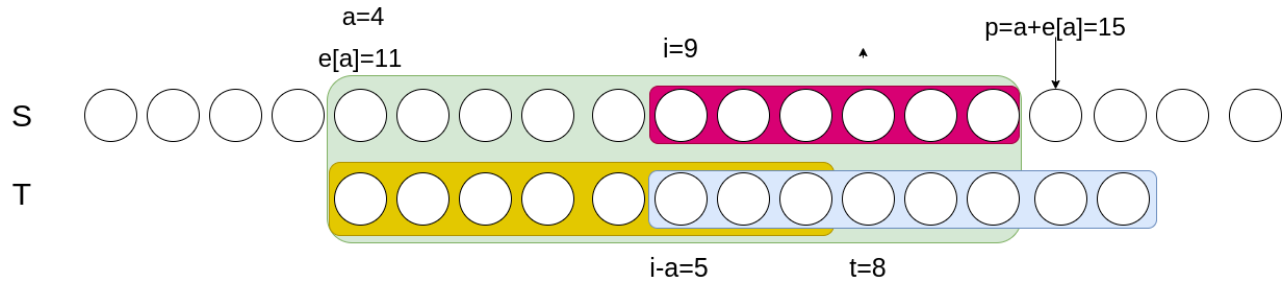
那么由上图可以知道, 蓝色的跟黄色的相等, 蓝色的跟红色的相等, 所以黄色的跟红色的相等. 并且, 由于 $f[i-a] = 3$, 所以 $T[t] \neq T[k]$, 但是 $T[k] = S[j]$, 所以 $T[t] \neq S[j]$, 所以 $e[i] = f[i-a] = 3$.

第二种情况,$i + f[i-a] = p$. 如下图,$f[i-a] = 6$



那么由上图可以知道, 蓝色的跟黄色的相等, 蓝色的跟红色的相等, 所以黄色的跟红色的相等. 并且, 由于 $f[i-a] = 6$, 所以 $T[t] \neq T[k]$, 同时 $T[k] \neq S[p]$, 所以 $T[t]$ 有可能等于 $S[p]$, 所以这个时候需要向后比较, 直到找到一个位置 x 满足 $S[x]$ 不等于对应的 T 的字符, 那么此时 $e[i] = x - i$. 并且这时候应该更新 p, a 为 $p = x, a = i$.

第三种情况,$i + f[i-a] > p$. 如下图,$f[i-a] = 8$



这种情况导致的结果跟第二种情况一样, 需要从 $S[p]$ 开始, 跟 T 对应的字符依次开始比较, 直到找到一个不等的位置, 然后同样要更新 p, a.
下面给出计算 e 的代码.

Listing 1: Compute array e

```
std::vector<size_t> GetE(const std::string &S, const std::string &T,
                         const std::vector<size_t> &f) {
  const size_t n = S.size();
  const size_t m = T.size();
  std::vector<size_t> e(n);
  size_t a = 0, p = 0;
  for (size_t i = 0; i < n; ++i) {
    if (i >= p || i + f[i - a] >= p) {
      if (i > p) {
        p = i;
      }
      while (p < n && p - i < m && S[p] == T[p - i]) {
        ++p;
      }
      e[i] = p - i;
      a = i;
    } else {
      e[i] = f[i - a];
    }
  }
  return e;
}
```

3 f 的计算

来看下 f 和 e 的区别
i $f[i]$ 表示 $SubT(i, m-1)$ 和 T 的最长公共前缀
ii $e[i]$ 表示 $SubS(i, n-1)$ 和 T 的最长公共前缀
所以 f 的计算方法跟 e 类似. 下面是计算 f 的代码.

Listing 2: Compute array f

```
std::vector<size_t> GetF(const std::string &T) {
  const size_t m = T.size();
  std::vector<size_t> f(m, m);
  size_t a = 0, p = 0;
  for (size_t i = 1; i < m; ++i) {
    if (i >= p || i + f[i - a] >= p) {
      if (i > p) {
        p = i;
      }
      while (p < m && T[p] == T[p - i]) {
        ++p;
      }
      f[i] = p - i;
      a = i;
    } else {
      f[i] = f[i - a];
    }
```

```
18      }
19      return f;
20  }
```