

# 6.046/18.410 Problem Set 3

Yijun Jiang

Collaborator: Hengyun Zhou, Eric Lau

September 30, 2015

## 1 Key Word Search

Convention: Here for a string, its leftmost digit is labeled as #0. For example for  $M = 111000$ , we have  $M[0] = 1$  and  $M[5] = 0$ . A different indexing convention may lead to small differences in the following parts.

### 1.1 Part (a)

Description: Scan from the 0-th to the  $(r - s)$ -th digit of  $M$ , digit by digit. For the  $i$ -th digit, check the matching between substring  $M[i : i + s - 1]$  and  $S_1$ , as well as between  $M[i : i + s - 1]$  and  $S_2$ . Use two counters to keep track of the number of successful matches of  $S_1$  and  $S_2$ . When all the digits are checked, compare the two counters and output the more frequent substring.

Correctness: This algorithm checks all the possible matches for  $S_1$  and  $S_2$  by directly comparing against the substring of  $M$  at each location. Since the loop goes through all the  $s$ -digit substrings of  $M$ , it is guaranteed that the count gives the correct number of successful matches.

Runtime: There are  $r - s + 1$  possible  $s$ -digit substrings. The comparison against each of them takes  $O(s)$  time, since it involves  $s$  single-bit comparisons. The overall runtime is thus  $T(r, s) = O(s(r - s + 1)) = O(rs)$ . The simplification to  $O(rs)$  is based on the fact that  $s < r$ .

### 1.2 Part (b)

Description: Define a function  $f : \{0, 1\} \mapsto \{-1, 1\}$  that maps the binary digit 0 to coefficient  $-1$  and binary digit 1 to coefficient 1. Now construct the following polynomials

$$\begin{aligned} P(x) &= \sum_{i=0}^{r-1} f(M[i])x^{r-i-1} \\ Q_1(x) &= \sum_{i=0}^{s-1} f(S_1[i])x^i \\ Q_2(x) &= \sum_{i=0}^{s-1} f(S_2[i])x^i \end{aligned}$$

Notice that the polynomial for  $M$  is decending in the power of  $x$ , i.e. the 0-th digit of  $M$  corresponds to  $x^{r-1}$ , while the polynomials for  $S_1$  and  $S_2$  are ascending, i.e. the 0-th digit of  $S_1$  or  $S_2$  corresponds to  $x^0$ .

We claim that, in the polynomial  $C_1(x) = P(x)Q_1(x)$ , the coefficient for  $x^i$ , denoted by  $c_{1,i}$ , reflects the number of matched bits of  $S_1$  in the substring  $M[r - i - 1 : r + s - i - 2]$ , where  $i = s - 1, s, \dots, r - 1$ . Specifically, this relation is given by

$$\# \text{ of matched bits of } S_1 \text{ at } M[r - i - 1] = \frac{1}{2}(s + c_{1,i})$$

Replacing the index in  $M$  by  $i$ , where  $i = 0, 1, \dots, r - s$ , and focusing on unmatched bits,

$$\# \text{ of unmatched bits of } S_1 \text{ at } M[i] = \frac{1}{2}(s - c_{1,r-i-1})$$

Since we tolerate at most  $e$  unmatcheds, if

$$c_{1,r-i-1} \geq s - 2e$$

we identify a successful match of  $S_1$  at  $M[i]$ . Due to the range of  $i$ , the subscript  $r - i - 1$  runs from  $s - 1$  to  $r - 1$ .

Obviously, the same is true for  $S_2$ . The proof is stated below. But before that, I will show an example. This helps clarify the indexing used here.

Example: Let  $M = 110101$  and  $S_1 = 100$ . Then  $r = 6$  and  $s = 3$ . According to the indexing convention,  $M[0] = 1, M[1] = 1, M[2] = 0, \dots, M[5] = 1$ . This corresponds to a polynomial

$$P(x) = x^5 + x^4 - x^3 + x^2 - x + 1$$

The polynomial  $Q_1(x)$ , however, is ascending in the power of  $x$ .

$$S_1(x) = 1 - x - x^2$$

Their product is

$$C_1(x) = P(x)Q_1(x) = -x^7 - 2x^6 + x^5 + x^4 - x^3 + x^2 - 2x + 1$$

Let  $i = 0$  for example. Number of unmatched bits at  $M[0]$  can be calculated from  $c_{1,r-i-1} = c_{1,5}$ , i.e. the coefficient of  $x^5$ , which is 1.

$$\# \text{ of unmatched bits of } S_1 \text{ at } M[0] = \frac{1}{2}(s - c_{1,5}) = \frac{1}{2}(3 - 1) = 1$$

which is verified by the fact that  $M[0 : 2] = 110$  differs from  $S_1$  by one digit.

Correctness: Let  $x, y \in \{0, 1\}$ . Notice that  $f(x)f(y) = 1$  if  $x = y$ , and  $f(x)f(y) = -1$  if  $x \neq y$ . Therefore,

$$\begin{aligned} C_1(x) &= P(x)Q_1(x) \\ &= \sum_{i=0}^{r-1} \sum_{j=0}^{s-1} f(M[i])f(S_1[j])x^{r-i+j-1} \\ &= \sum_{k=0}^{r-s} x^{r-k-1} \sum_{j=0}^{s-1} f(M[k+j])f(S_1[j]) \\ &= \sum_{k=0}^{r-s} (\# \text{ of matches at } M[k] - \# \text{ of unmatcheds at } M[k])x^{r-k-1} \end{aligned}$$

where a substitution  $k = i - j$  has been made.

Let  $n_{1,k}^m = \#$  of matched bits of  $S_1$  at  $M[k]$ , and  $n_{1,k}^u = \#$  of unmatched bits of  $S_1$  at  $M[k]$ . From the equation above, it is clear that  $c_{1,r-k-1} = n_{1,k}^m - n_{1,k}^u$ . Since  $n_{1,k}^m + n_{1,k}^u = s$ , we have

$$n_{1,k}^u = \frac{1}{2}(s - c_{1,r-k-1})$$

A successful match of  $S_1$  has  $n_{1,k}^u \leq e$ . The criterion is thus

$$c_{1,r-k-1} \geq s - 2e$$

The same is true for  $S_2$ .

---

**Algorithm 1** Solving Eve's string matching problem in  $O(r \log r)$ 

---

```
1: procedure STRINGMATCHING( $M, S_1, S_2, e$ )
2:   Calculate coefficients of  $P(x), Q_1(x)$  and  $Q_2(x)$ 
3:   Call FFT to evaluate  $P(x), Q_1(x)$  and  $Q_2(x)$  on a collapsing set  $A$  of  $r + s - 1$  points:  $x_1, \dots, x_{r+s-1}$ 
4:   Calculate  $C_1(x_1), \dots, C_1(x_{r+s-1})$  and  $C_2(x_1), \dots, C_2(x_{r+s-1})$ , where  $C_1 = PQ_1$  and  $C_2 = PQ_2$ 
5:   Call IFFT to calculate  $c_{1,i}$  and  $c_{2,i}$ , i.e. coefficients of  $C_1(x)$  and  $C_2(x)$ , where  $i = 0, \dots, r + s - 2$ 
6:    $count1, count2 \leftarrow 0$ 
7:   for  $i = s - 1 : r - 1$  do                                 $\triangleright$  Counting, only coefficients of  $x^{s-1}, \dots, x^{r-1}$  matters
8:     if  $c_{1,i} \geq s - 2e$  then
9:        $count1++$ 
10:    if  $c_{2,i} \geq s - 2e$  then
11:       $count2++$ 
12:    if  $count1 > count2$  then return  $S_1$ 
13:    else if  $count1 < count2$  then return  $S_2$ 
14:    else return  $S_1, S_2$ 
```

---

### 1.3 Part (c)

Description: See the pseudocode.

Correctness: From the previous part, we know that  $n_{\alpha,k}^u = \frac{1}{2}(s - c_{\alpha,r-k-1})$ , where  $\alpha = 1, 2$ . Therefore, by looking at the coefficients of the product polynomials, we can determine the number of “good matches”, i.e. the number of  $k$  such that  $n_{\alpha,k}^u \leq e$ . The pattern that has more good matches is a more frequent pattern. Notice that since  $k = 0, \dots, r - s$ , only the coefficients of  $x^{r-k-1}$ , in other words  $x^{s-1}, \dots, x^{r-1}$ , matters.

Runtime: The first step, calculating coefficients for  $P(x), Q_1(x)$  and  $Q_2(x)$ , costs  $O(r) + O(s) + O(s) = O(r)$  time. The second step, FFT, costs  $O((r + s) \log(r + s)) = O(r \log r)$  time. The third step, evaluating  $C_1(x)$  and  $C_2(x)$  at  $r + s - 1$  points, costs  $O(r + s) = O(r)$  time. The fourth step, IFFT, costs  $O(r \log r)$  time. Finally, counting and comparing costs  $O(r - s) = O(r)$  time. Therefore, the total runtime is  $O(r \log r)$ .

## 2 Optical Fiber Network

### 2.1 Part (a)

Proof:

By contradiction. Suppose  $S$  differs from  $T$  by more than one edges. Consider the set  $T - S$  of all edges that are in  $T$  but not in  $S$ . Let  $e_T$  be the edge of minimum weight in  $T - S$ . Adding  $e_T$  into  $S$  creates a cycle  $C_S$ .  $T$  cannot contain cycles, so  $C_S \not\subseteq T$ , which means that there must be some  $e_S \in C_S$  such that  $e_S \in S - T$ .

Adding  $e_S$  into  $T$  creates a cycle  $C_T$ . By the same reasoning, there must be some  $e'_T \in C_T$  such that  $e'_T \in T - S$ . Consider the spanning tree  $T' = T - \{e'_T\} + \{e_S\}$ . We have  $w(T) < w(T')$  due to the uniqueness of  $T$  as an MST. Therefore,  $w(e'_T) < w(e_S)$ . Moreover, since  $e_T$  is chosen to be the lightest edge in  $T - S$ ,  $w(e_T) \leq w(e'_T)$ . Therefore,  $w(e_T) < w(e_S)$ .

Consider the spanning tree  $S' = S - \{e_S\} + \{e_T\}$ . Then  $w(S') < w(S)$ . Since  $S'$  only differs from  $S$  by one edge,  $S' \neq T$ . Therefore,  $w(T) < w(S') < w(S)$ , contradicting the fact that  $S$  is the second minimum spanning tree.

Consequently, when MST is unique, a second minimum spanning tree differs from MST by exactly one edge.

### 2.2 Part (b)

Description: For convenience let  $D[u, u] = \text{NULL}$  and let its weight be  $w(\text{NULL}) = -\infty$ . The algorithm performs a depth-first traversal of  $T$ . Maintain  $A$  as a subset of  $V$  containing all the vertices that are visited.

$A$  is initialized to contain only the root  $r$ . Maintain the property that  $D[u_1, u_2]$  is determined for every  $u_1, u_2 \in A$ . This is done by the following: whenever a new vertex  $v$  is visited, calculate  $D[u_i, v] = D[v, u_i]$  for all  $u_i \in A$ , before adding  $v$  into  $A$ . This calculation is done by the equation below, where  $w(u, v)$  denotes the weight of  $(u, v)$ .

$$D[u_i, v] = \begin{cases} D[u_i, v.father] & \text{if } w(D[u_i, v.father]) \geq w(v, v.father) \\ (v, v.father) & \text{otherwise} \end{cases}$$

which holds for all newly added  $v$ .  $v \neq r$  since  $r \in A$  at the beginning, so  $v.father$  exists. Moreover,  $v.father \in A$  when  $v$  is visited since the traversal is depth-first, so  $D[u_i, v.father]$  can be accessed.

The procedure ends when every vertex is visited. In the meantime,  $D$  is complete filled.

The pseudocode contains more details.

---

**Algorithm 2** Finding the longest edge on a unique path

---

```

1: procedure LONGESTEDGE( $T$ )
2:    $A \leftarrow [T.root]$ 
3:    $D(T.root, T.root) \leftarrow \text{NULL}$ 
4:   DEPTH-FIRST( $T.root, A, D$ )
5:   return  $D$ 
6: procedure DEPTH-FIRST( $root, A, D$ )
7:   if  $root.childNum = 0$  then return
8:   else
9:     for  $i = 1 : root.childNum$  do
10:      for  $u$  in  $A$  do
11:        if  $w(D[u, root]) \geq w(root, child[i])$  then  $\triangleright w(\text{NULL}) = -\infty$ 
12:           $D[u, root.child[i]] \leftarrow D[u, root]$ 
13:           $D[root.child[i], u] \leftarrow D[u, root]$ 
14:        else
15:           $D[u, root.child[i]] \leftarrow (root, child[i])$ 
16:           $D[root.child[i], u] \leftarrow (root, child[i])$ 
17:         $D[root.child[i], root.child[i]] \leftarrow \text{NULL}$ 
18:         $A.append(root.child[i])$ 
19:        DEPTH-FIRST( $root.child[i], A, D$ )
20:   return

```

---

Correctness: Since  $A$  is connected, for  $u_1, u_2 \in A$ , all the edges of  $T$  in the unique path connecting  $u_1$  and  $u_2$  lie in  $A$ . So  $D[u_1, u_2]$  can be determined within  $A$ , even though  $A$  is only a subset of  $V$ . Moreover, all such  $D$  entries are determined for  $A$ . This is because (1)  $A$  is initialized with this property (2) whenever a new vertex  $v$  is about to be added into  $A$ ,  $D[u_i, v] = D[v, u_i]$  is calculated for every  $u_i \in A$ . Therefore, as  $A$  eventually expands to  $V$ , we get the full  $D$  matrix.

Furthermore, the correctness of this algorithm relies on the correctness of the following equation

$$D[u_i, v] = \begin{cases} D[u_i, v.father] & \text{if } w(D[u_i, v.father]) \geq w(v, v.father) \\ (v, v.father) & \text{otherwise} \end{cases}$$

Since the new vertex  $v$  connects to  $A$  at  $v.father \in A$ , the unique path from  $v$  to  $u_i \in A$  is the union of  $(v, v.father)$  and the unique path from  $v.father$  to  $u_i$ . Therefore, the heaviest edge must be either  $(v, v.father)$  or  $D[u_i, v.father]$ , depending on which has a higher weight. This proves the correctness of the equation above.

Runtime: The algorithm visits  $V$  vertices. At vertex  $v$ , it calculates  $D[u, v]$  for all  $u \in A$ . For one  $v$ , this means  $O(V)$  operations, each of which contains constant times of comparing and value assigning. Therefore, the total runtime is  $O(V) \times O(V) = O(V^2)$ .

## 2.3 Part (c)

Description: For each  $(u, v) \in E$ , consider the tree  $S_{u,v} = T \cup \{(u, v)\} - \{D[u, v]\}$ . The  $S_{u,v}$  with minimal weight is the second-best spanning tree. See the pseudocode.

---

**Algorithm 3** Finding the second-best spanning tree

---

```

1: procedure SECONDBEST( $T, E, V$ )
2:    $D = \text{LONGESTEDGE}(T)$ 
3:    $\text{maxWeightReduce} \leftarrow -\infty$ 
4:   for  $(u, v)$  in  $E$  do
5:     if  $w(D[u, v]) - w(u, v) > \text{maxWeightReduce}$  then
6:        $\text{maxWeightReduce} \leftarrow w(D[u, v]) - w(u, v)$ 
7:        $(u^*, v^*) \leftarrow (u, v)$ 
8:    $S \leftarrow T \cup \{(u^*, v^*)\} - \{D[u^*, v^*]\}$ 
9:   return  $S$ 

```

---

Correctness: According to part (a),  $S$  only differs from  $T$  by one edge. Suppose  $(u, v) \in S$  but  $(u, v) \notin T$ . Then in order to avoid a loop, one of the edges along the original path connecting  $u$  and  $v$  must be removed from  $S$ . In order to minimize  $w(S)$ , the optimal choice of this edge is the heaviest one,  $D[u, v]$ . By looping over all possible  $(u, v) \in E$ , we compare across different  $S_{u,v} = T \cup \{(u, v)\} - \{D[u, v]\}$ . The lightest of all is the lightest possible spanning tree that differs from  $T$  by exactly one edge. Part (a) tells us that it must be the second-best spanning tree  $S$ .

Runtime: `LONGESTEDGE` costs  $O(V^2)$ . Then the algorithm loops through all  $|E|$  vertex pairs. In each iteration only constant amount of work is done. Therefore, the loops runs at  $O(E) = O(V^2)$  time. Altogether, this algorithm has a runtime of  $O(V^2)$ .