ECS 222A: Assignment #1

Due on Tusday, January 13, 2015

Daniel Gusfield TR 4:40pm-6:00pm

Wenhao Wu

Contents

Problem 0

In the bit-model every bit-level operation must be counted. For example, to take the OR of two binary strings of length q each, takes q operations, and to use or set an index consisting of q bits, takes q operations. Show in detail that in the bit-model, then the 4-Russians method for bitmatrix multiplication only takes $\mathcal{O}(n^3/(\log n))$ operations.

Answer: The steps of 4-Russian's method for bitmatrix multiplication (BMM) are summarized in Algorithm 1.

Algorithm 1 4-Russian's algorithm for BMM of $n \times n$ bit matrix A and B: C = AB

1: Define $k = \log_4 n$. Build a $2^k \times 2^k$ look up tables T. $T(u, v)$ in which $u = 0, \ldots, 2^k - 1$ and $v = 0, \ldots, 2^k - 1$ is the multiplication between the row bit vector representing i and the column vit vector representing j.
2: **for** i = 1 **to** n **do**
3: **for** j = 1 **to** n **do**
4: Divide A's i-th row $A_{i,:}$ into n/k row vectors of length k, denoted as $A^{(1)}_{i,:}, \ldots, A^{(n/k)}_{i,:}$. Similarly, divide B's j-th row $B_{:,j}$ into n/k column vectors of length k, denoted as $B^{(1)}_{:,j}, \ldots, B^{(n/k)}_{:,j}$.
5: Lookup the multiplication value $A^{(w)}_{i,:} B^{(w)}_{:,j}, w = 1, \ldots, n/k$, from table T.
6: Take the OR operation of all $A^{(w)}_{i,:} B^{(w)}_{:,j}, w = 1, \ldots, n/k$ as the result C_{ij}.
7: **end for**
8: **end for**

In line 1, to build the $2^k \times 2^k = \sqrt{n} \times \sqrt{n}$ lookup table, according to the matrix multiplication definition, it takes $\mathcal{O}(n^{3/2})$ operations
In line 5, given (i, j), for each w it takes $\mathcal{O}(k)$ to index the table T, so the total number of cost of index is $\mathcal{O}(\log_4 n)$.
In line 6, given (i, j), the sum-OR takes $\mathcal{O}(n/k) = \mathcal{O}(n/\log_4 n)$ operations.
Consequently, the total number of operations is

$$\mathcal{O}(n^{3/2}) + n^2(\mathcal{O}(\log_4 n) + \mathcal{O}(n/\log_4 n)) = \mathcal{O}(n^3/(\log(n)))$$

Problem 1

Prove that the edit distance is the same no matter which definition is used.

Answer: In order to change both S_1 and S_2 into a same string (no matter what exactly this string is), given a position p_1 in S_1 and position p_2 in S_2, we need to carry out some operation either at p_1 or p_2 (but not both) so that after the operation the charactors on this two positions are matched. In this sense the following pairs of operations are equivalent:

- Insertion in S_1 and deletion in S_2.
- Deletion in S_1 and insertion in S_2.
- Replacement in S_1 and replacement in S_2.

We can see that for each of the 3 allowed operation on 1 string, there is an equivalent operation on the other string that has the same effect (get a match in the current position). Therefore, the edit distance is the same no matter which definition is used.

Problem 2

From the mathematical standpoint, an alignment and an edit transcript are equivalent ways to describe a relationship between two strings. An alignment can be easily converted to the equivalent edit transcript and vice-versa. Completely explain and justify the above statement.

Answer: The algorithm to convert an alignment to an edit transcript is shown in Algorithm 2.

Algorithm 2 Convert an alignment (S_1', S_2') to an edit transcript E.

for i = 1 **to** length(S_1') **do**
  if $S_1'[i] == S_2'[i]$ **then**
    $E[i] = \text{M}$
  else if $S_1'[i] = \_$ **then**
    $E[i] = \text{I}$
  else if $S_2'[i] = \_$ **then**
    $E[i] = \text{D}$
  else
    $E[i] = \text{R}$
  end if
end for

The algorithm to convert an edit transcript to an alignment is shown in Algorithm 3.

Algorithm 3 Convert an edit transcript E and the original strings (S_1, S_2) to an alignment (S_1', S_2').

Initialize pointer p_1 to the beginning of S_1, p_2 to the beginning of S_2, e to the beginning of E, and a to the beginning of the alignment S_1', S_2'.
repeat
  if $E[e] = \text{M}$ **or** $E[e] = \text{R}$ **then**
    $S_1'[a] = S_1[p_1]$, $S_2'[a] = S_2[p_2]$.
    $e+ = 1$, $p_1+ = 1$, $p_2+ = 1$, $a+ = 1$.
  else if $E[e] = \text{I}$ **then**
    $S_1'[a] = \_$, $S_2'[a] = S_2[p_2]$.
    $e+ = 1$, $p_2+ = 1$, $a+ = 1$.
  else
    $S_1'[a] = S_1[p_1]$, $S_2'[a] = \_$.
    $e+ = 1$, $p_1+ = 1$, $a+ = 1$.
  end if
until p_1 at the end of S_1 **and** p_2 at the end of S_2

Problem 3

Prove that any traceback path specifies an optimal edit transcript, and an optimal alignment. In the latter case, explain how the path specifies where the spaces should go in the two strings.

Answer: Firstly we prove that any traceback path specifies an optimal edit transcript. To do so we can use mathematical induction. Denote $E(i, j)$ as the edit transcript defined by the traceback path from cell (i, j) back to cell $(0, 0)$ in the table. We would like to prove the more general result that $E(i, j)$ is an optimal edit transcript, which implies:

1. $E(i, j)$ is uses exactly $D(i, j)$ conversions, i.e. the sum of number of (I)(D)(R) in $E(i, j)$, defined as $N(i, j)$ always equals to $D(i, j)$.

Problem 3 continued on next page...

2. $E(i,j)$ can indeed convert $S_1[1:i]$ to $S_2[1:j]$.

We prove these two points with mathematical induction:

- Since $E(i,0)$ is just i D's, we have $N(i,0) = i = D(i,0)$. Also, carrying out i deletion to $S_1[1:i]$ results in the empty string $S_2[1:0]$. On the other hand, $E(0,j)$ is just j I's, so $N(0,j) = j = D(0,j)$. Besides, carrying out j insertions to $S_1[1:0]$ apparently results in the string $S_2[1:j]$.
- If $E(i,j-1)$, $E(i-1,j)$ and $E(i-1,j-1)$ are an optimal edit transcript for $(S_1[1:i], S_2[1:j-1])$, $(S_1[1:i-1], S_2[1:j])$ and $(S_1[1:i-1], S_2[1:j-1])$, respectively, we would like to prove that $E(i,j)$ is also an optimal edit transcript for $(S_1[1:i], S_2[1:j])$:

  1. If the pointer to cell (i,j) is from cell $(i,j-1)$, then we know that $D(i,j) = D(i,j-1) + 1$. According to the description of how the edit transcript is derived, $E(i,j) = [E(i,j-1), \mathrm{I}]$. Consequently, $N(i,j) = N(i,j-1) + 1 = D(i,j-1) + 1 = D(i,j)$.

     Also, since with $E(i,j-1)$ we can convert $S_1[1:i]$ to $S_2[1:j-1]$, then by inserting $S_2[j]$ to the end of $S_1[1:i]$, we can exactly convert $S_1[1:i]$ to $S_2[1:j]$ with $E(i,j)$.
  2. If the pointer to cell (i,j) is from cell $(i-1,j)$, then we know that $D(i,j) = D(i-1,j) + 1$. According to the description of how the edit transcript is derived, $E(i,j) = [E(i-1,j), \mathrm{D}]$. Consequently, $N(i,j) = N(i-1,j) + 1 = D(i-1,j) + 1 = D(i,j)$.

     Also, since with $E(i-1,j)$ we can convert $S_1[1:i-1]$ to $S_2[1:j]$, then by deleting $S_1[i]$ at the end of $S_1[1:i]$, we can exactly convert $S_1[1:i]$ to $S_2[1:j]$ with $E(i,j)$.
  3. If the pointer to cell (i,j) is from cell $(i-1,j-1)$, then we know that $D(i,j) = D(i-1,j-1) + t(i,j)$.
     - If $t(i,j) = 0$, we know that $S_1[i] = S_2[j]$, thus $E(i,j) = [E(i-1,j-1), \mathrm{M}]$. Consequently, $N(i,j) = N(i-1,j-1) = D(i-1,j-1) = D(i,j)$. Also, since with $E(i-1,j-1)$ we can convert $S_1[1:i-1]$ to $S_2[1:j-1]$, by appending the same character to the end of $S_1[1:i-1]$ and $S_2[1:j-1]$ we can exactly convert $S_1[1:i]$ to $S_2[1:j]$ with $E(i,j)$.
     - If $t(i,j) = 1$, we know that $S_1[i] \neq S_2[j]$, thus $E(i,j) = [E(i-1,j-1), \mathrm{R}]$. Consequently, $N(i,j) = N(i-1,j-1) + 1 = D(i-1,j-1) + 1 = D(i,j)$. Also, since with $E(i-1,j-1)$ we can convert $S_1[1:i-1]$ to $S_2[1:j-1]$, by replacing $S_1[i]$ with $S_2[j]$ we can exactly convert $S_1[1:i]$ to $S_2[1:j]$ with $E(i,j)$.

According to the mathematical induction, we have proved that any traceback path specifies an optimal edit transcript. From Problem 2, such a traceback path also specifies an optimal alignment. The algorithm that computes the optimal alignment directly from a traceback is described in Algorithm 4.

Problem 4

Theorem Any path from (n,m) to $(0,0)$ following pointers established during the computation of $D(i,j)$ specifies an edit transcript with the minimum number of edit operations. Conversely, any optimal edit transcript is specified by such a path. Moreover, since a path describes only one transcript, the correspondence between paths and optimal transcripts is one-one.
Prove this theorem.
Answer:

Problem 5

Since the traceback paths in a dynamic programming table correspond one-to-one with the optimal alignments, the number of distinct co-optimal alignments can be obtained by computing the number of distinct

Problem 5 continued on next page...

Algorithm 4 Convert an optimal traceback pathc T edit and the original strings (S_1, S_2) to an optimal alignment (S'_1, S'_2).

Initialize pointer p_1 to the beginning of S_1, p_2 to the beginning of S_2, t to the beginning of T, and a to the beginning of the alignment S'_1, S'_2.
repeat
  if $T[t+1]$ takes a horizontal step from $T[t]$ **then**
    $S'_1[a] =$ _, $S'_2[a] = S_2[p_2]$.
    $t+ = 1$, $p_2+ = 1$, $a+ = 1$.
  else if $T[t+1]$ takes a vertical step from $T[t]$ **then**
    $S'_1[a] = S_1[p_1]$, $S'_2[a] =$ _.
    $t+ = 1$, $p_1+ = 1$, $a+ = 1$.
  else
    $S'_1[a] = S_1[p_1]$, $S'_2[a] = S_2[p_2]$.
    $t+ = 1$, $p_1+ = 1$, $p_2+ = 1$, $a+ = 1$.
  end if
until t at the end of T

traceback paths. Give an algorithm to compute this number in $O(nm)$ time. Hint: use dynamic programming.

Answer: