

ECS 222A: Assignment #3

Due on Tuesday, January 20, 2015

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

Wenhao Wu

Contents

Problem 1	3
Problem 1(a)	3
Problem 1(b)	4
Problem 2	4
Problem 2(a)	5
Problem 2(b)	5
Problem 2(c)	5
Problem 3	5
Problem 4	5
Problem 4(a)	5
Problem 4(b)	6
Problem 4(c)	6
Problem 4(d)	6

Problem 1

The secondary structure problem discussed in section 6.5 in the book seeks to find the secondary structure that maximizes the number of base pairs it contains, i.e. the number of pairs that are in the matching (or pairing). Review the definitions of “secondary structure” on page 274 in the book. In class, we used the term “non-crossing pairing” for what the book calls a “secondary structure”.

Now suppose that instead of wanting to find the the secondary structure maximizing the number of pairs it contains, we want to *count* the exact *number* of distinct secondary structures possible in a given RNA sequence. Some of these secondary structures will not contain the largest number of pairs possible.

For example, in the RNA molecule ACGGGUGU there are five secondary structures. One contains no pairs (hey, its a legal secondary structure according to the definition); one pairs the A to the farthest U; one that pairs the A to closest U; one pairs the C to the farthest G; and one pairs the A to the farthest U, and the C to the farthest G.

This counting problem can be solved by DP.

Problem 1(a)

Write recurrence relations that give the solution to the counting problem.

Hint: You may be tempted to just do a simple conversion of the recurrences we used in class to find the maximum number of pairs in a non-crossing pairing (secondary structure), but you need to be careful. The reason, is that the cases in the recurrences we used in class were not disjoint. That is, the same secondary structure might arise by more than one case in the recurrences. Since we were taking the Max over all the cases, it did not matter if the same secondary structure arose different ways. But now that we want to count the number of distinct secondary structures, we have to be more careful. Define $N(i, j)$ as the number of secondary structures involving the positions from i to j inclusive. It includes the empty matching as one of the matchings. For technical reasons, you may want to define $N(j, j) = N$ and $N(j + 1, j) = 1$. The problem asks you to write recurrences for $N(i, j)$.

Be sure to explain why your recurrences give a correct recursive solution for the problem of counting the number of secondary structures.

Answer: Consider all the secondary structures on the inclusive interval (i, j) :

- The number of secondary structures in which element j is not paired with any element in $(i, j - 1)$ is the same as the number of secondary structures on $(i, j - 1)$, i.e.

$$N_1(i, j) = N(i, j - 1)$$

- If element j can be paired with some element $k \in (i, j - 1)$, indicated by $\beta(k, j) = 1$, then any secondary structure on $(i, k - 1)$ and any secondary structure on $(k + 1, j - 1)$, together with the pair formed by element k and j , forms a distinct secondary structure on (i, j) that is not included in the previous case. Consequently, the number of these secondary structures is

$$N_{2,k}(i, j) = \begin{cases} N(i, k - 1)N(k + 1, j - 1), & \text{if } \beta(k, j) = 1 \\ 0, & \text{else} \end{cases}$$

These 2 cases exhaust all distinct secondary structures, therefore the recurrence of counting is

$$\begin{aligned} N(i, j) &= N_1(i, j) + \sum_{k=i}^{j-1} N_{2,k}(i, j) \\ &= N(i, j-1) + \sum_{\substack{k=i \\ \beta(k,j)=1}}^{j-1} N(i, k-1)N(k+1, j-1) \end{aligned}$$

And the base cases are (according to the definition on the textbook and the example given here, no sharp turns)

$$N(j, j+m) = 1, \quad m = -1, 0, 1, 2, 3, 4.$$

In these cases the only secondary structure is no pairing at all.

Problem 1(b)

As before, instead of using the recurrences in a top-down recursive algorithm, we want to use them in a DP solution to the problem.

Write out the pseudo-code for a DP solution to the counting problem, and analyze the worst-case running time of the DP solution.

Answer: The pseudo-code for a DP solution is given in Algorithm 1. There are $O(n^2)$ $N(i, j)$ terms in the

Algorithm 1 Count the number of secondary structures in a RNA string.

```

1: Initialize  $N(j, j+m) = 1$  for  $m = -1, 0, 1, 2, 3, 4, j = 1, \dots, n$ .
2: for  $m = 1$  to  $n-1$  do
3:   for  $i = 1$  to  $n$  do
4:     Compute
```

$$N(i, i+m) = N(i, i+m-1) + \sum_{\substack{k=i \\ \beta(k,i+m)=1}}^{i+m-1} N(i, k-1)N(k+1, i+m-1)$$

```

5:   end for
6: end for
7: return  $N(1, m)$ .
```

DP table to fill in. To fill in each term using the recurrence equation requires $O(n)$ multiplication, addition and table lookup (RAM model). Consequently, the worst-case running time of the DP solution is $O(n^3)$.

Problem 2

Suppose we are given a rooted tree T with n leaves and m non-leaf nodes. Each leaf is colored with one of $k < n$ given colors, so several leaves can have the same color. We need to color each interior node of T with one of the k given colors to *maximize* the number of edges whose (two) endpoints are colored the same color.

We can solve this with a DP algorithm that runs in $O(mk)$ time. Let $V(v, i)$ denote the optimal solution value when the problem is applied to the subtree rooted at node v , and v is required to be given color i . Let $V(v)$ denote the optimal solution value when the problem is applied to the subtree rooted at node v , and there is no restriction on which of the k colors v can be.

Problem 2(a)

Using that notation, develop recurrences for this problem, and explain the correctness of your recurrences.

Answer:

Problem 2(b)

Explain how the recurrences are evaluated (solved) in an efficient DP way.

Answer:

Problem 2(c)

Show that the time bound for your DP is $O(mk)$.

Answer:

Problem 3

In the sequence alignment problem, suppose now we want to compute the number of optimal alignments that align character i with character j , or align character i with a space, for each pair (i, j) . Show how to compute this via DP in $O(nm)$ time.

Hint: You might be able to extend your answer to Problem 5 in HW 2.

Answer:

Problem 4

Here is another Four-Russians approach to doing bit-matrix multiplication of $A \times B = C$. Assume that A and B are both of dimension $n \times n$. Instead of doing the preprocessing of B (which is how the Four Russians for bit-matrix mult. was done in class), we will preprocess A . Suppose A is dimension $n \times n$, and n is a multiple of q (to be set later). We partition A into squares of dimension $q \times q$. So there are n^2/q^2 of these squares, and they are indexed by (s, t) where s and t both range from 1 to n/q . For each pair (s, t) , build a table of size 2^q , one cell for each possible binary number of length q . The cells in that table are indexed by the binary numbers 0 through $2^q - 1$. For the cell in the (s, t) table, indexed by binary number b , compute the bit-matrix multiplication of square (s, t) times b . The result is a vector of length q . That is the preprocessing for A .

Problem 4(a)

In the RAM model, we can follow a pointer, or use an index, or lookup a value, or take the OR of two vectors in constant time, provided that each pointer, index, value or vector only uses $\log m$ bits, where m is the number of bits used to represent the input. In our case, m is n^2 . However, the bit-matrix multiplication of two vectors of size q , takes time q . How much time does the preprocessing of A take in the RAM model?

Answer:

Problem 4(b)

After preprocessing of A , we want to compute $A \times B$. We will view that as n multiplication $A \times B_j$, where B_j is the j column of B , and where j ranges from 1 to n . That is, the j column of C is A times the j column of B .

To do multiplication $A \times B_j = C_j$, we divide B_j and C_j into n/q groups of size q each.

Explain how to use the preprocessed tables to compute the first group of size q of C_j .

Answer:

Problem 4(c)

Estimate the total time needed to compute C_j , using the preprocessed tables, under the RAM model.

Answer:

Problem 4(d)

Show that this approach can compute the bit-matrix multiplication in $O(n^3 / \log n)$, by picking q to be $\epsilon \log n$, for any $\epsilon > 1$.

Answer: