CMSC701 Final

Yi Qian

December 16, 2014

Problem 1.

1. For string S of length n, it can have at most $\lfloor n/2 \rfloor$ uncontained repeats.

2. An internal node v of the suffix tree of S represents an uncontained repeat if and only there is no suffix link maps to it.

   Proof. ($\Leftarrow$) First, since v is an internal node, it has at least 2 children. Hence, there are at least 2 different suffix starts with v. Second, we know there is no substring that contains v, because otherwise there will be a suffix link maps to v.

   ($\Rightarrow$) The other direction is similar. ■

3. (a) Construct the suffix tree T of S.

   (b) Traverse T in a DFS manner and return all unmapped internal nodes in T.

   The correctness of the algorithm follows part (2). The running time of the algorithm is obvious.

Problem 2.

The solution here answers both (1) and (2).

Define $V[i, j]$ to be the score of the best fold of substring $S[i \dots j]$.

Define $C[i, j]$ to be the number of consecutive matches .

Problem 3.

The algorithm follows a greedy fashion. Intuitively, in each step it find stwo bins from A and B with the minimum equal sum.

1. BinPacking(A, B)

2. For $i = 1 : n$:

3. Pick bin $A_i = A[1 \dots i]$. Compute its sum S_{A_i}.

4. For $j = 1 : m$:

5. Pick bin $B_j = B[1 \dots j]$. Compute its sum S_{B_j}

6. If $S_{A_i} = S_{B_j}$:

7. Return $\{(A_i, B_i)\} \cup$ BinPacking(A[i+1 ... n], B[j+1 ... m])

8. If $S_{A_i} < S_{B_j}$:

9. Break //continue the loop at line 2

The running time of algorithm in the worst case is $O(nm)$.

Correctness Proof. Assume by contradiction that there is a better solution O' for (A, B) than O given by the above algorithm. Assume without losing generality that the first pair of bins in O and O' differ. Since the first pair of bins in O, $(A_1, B_1$, is of minimum possible sum, the first pair of bins in O', $(A'_1, B'_1$, has a larger sum. But then (A'_1, B'_1) can be further divided into two bins (A_1, B_1) and $(A'_1 - A_1, B'_1 - B_1)$. Contradiction. ■

Problem 4

The following algorithm deals with the case when there is no sequencing error. Let R denote the read.

1. Construct the suffix tree T of the genome

2. Compare R with every suffix of T. If the read matches the prefix of one of the suffix, return the suffix and the read is break-point free.

3. For $i = 1 : m - 1$:

4. Compare both $R[1 \dots i]$ and $R[i + 1 \dots i]$ with every suffix of T.

5. If they both match the prefixes of some suffixes, return them and the break-point i.

In each exact matching, the algorithm does at most m comparisons. There are in total n suffix string and m possible break-points. Hence, in total it takes $O(nm^2)$ comparisons in the worst case.

Problem 5.