582670 Algorithms for Bioinformatics, 4 cr — Exam 19.10.2011 — Solutions/grading

1. **Greedy algorithms and genome rearrangements.**

   Simulate the improved breakpoint reversal sort 4-approximation algorithm on the permutation

   ```
   4 3 2 7 9 6 5 1 8.
   ```

   Based on the properties of this problem instance, estimate how many more reversals does the algorithm make compared to the optimal solution?

   Solution.

   Recall increasing and decreasing strips. One element strips are defined as decreasing, except for the special case of elements 1 and n: if they are located at their correct positions, then there is no breakpoint before / after, respectively, and they can be seen as increasing strips.

   The improved breakpoint reversal sorting finds a reversal distance of 4 reversals:

   ```
   4 3 2 (7 9 6 5 1) 8    7bp
   (4 3 2 1) 5 6 9 7 8    5bp
   1 2 3 4 5 6 (9 7 8)    3bp
   1 2 3 4 5 6 (8 7) 9    2bp
   1 2 3 4 5 6 7 8 9      0bp
   ```

   Originally there were 7 breakpoints and, since each reversal can remove at most two breakpoints, we have $OPT \geq 3.5$ (i.e. $OPT \geq 4$) reversals. In this problem instance, the improved breakpoint reversal sort used four reversals and therefore it returned an optimal solution.

   Grading:

   - +8 points for correct simulation of the improved breakpoint reversal sort.
   - +4 points from relating the answer to the lower bound.
   - Some points given for partially correct simulations.

2. **Algorithms for finding phylogenetic trees.** (12 points)

   Define the problem of finding a phylogenetic tree. UPGMA and neighbor joining algorithm both solve this problem. Describe briefly the common idea behind the algorithms. How do they differ? In which conditions are the algorithms able to construct the correct tree?

   Solution:

   See course material for answers.

   Grading:

- +3 points for problem definition
- +3 points for describing the common idea
- +3 points for describing the differences
- +3 points for describing conditions for correct tree construction

3. **Dynamic programming and sequence alignment.** (6+6 points)

   A tandem repeat P^k of a pattern $P = p_1 \dots p_m$ is a pattern of length $k \cdot m$ formed by concatenating k copies of P. The *tandem repeat* problem is to find the best global alignment between some interval of a text $T = t_1 \dots t_n$ and a tandem repeat P^k for some k. When computing global alignments the premium for matching is $+1$ and the mismatch and gap penalties are -1.

   For example given the pattern $P =$ `GGT` and the text $T =$ `ACCGGTGCTGTAA` the best tandem repeat is given by $k = 3$ and the following alignment:

```
ACCGGTGCTG-TAA
   GGTGGTGGT
```

   (a) Modify the global alignment dynamic programming solution to solve the tandem repeat problem by computing the alignment matrix between P^n (the maximum number of times P can possibly repeat in T) and T. What is the running time of the algorithm?

   (b) Devise an algorithm based on dynamic programming that solves the tandem repeat problem in $O(nm)$ time. (*Hint: Modify the alignment DAG to wrap around for the occurrences of P.*)

   Solution:

   (a) We want to align a prefix of P^n against any substring of T. Thus deletions in the beginning of T should be free but deletions in the beginning of P^n should not. Therefore we initialize D as

$$D[0, j] = 0, \text{ where } 0 \leq j \leq n$$
$$D[i, 0] = -i, \text{ where } 0 \leq i \leq nm$$

   The array is filled using the normal recursion for the alignment problems:

$$D[i, j] = \max \begin{cases} D[i-1, j-1] + \delta(P^n[i], T[j]) \\ D[i-1, j] + \delta(P^n[i], -) \\ D[i, j-1] + \delta(-, T[j]) \end{cases}$$

   The best global alignment score between any substring of T and a tandem repeat P^k is the maximum score in the matrix on the rows $k \cdot m$ for $0 \leq k \leq n$. The alignment can be obtained by tracing through the matrix as usual.

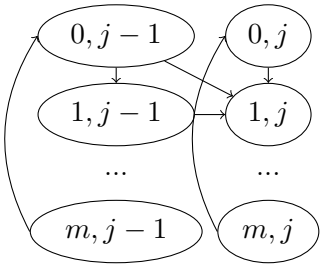
   The running time of this algorithm is $O(n^2 m)$.

   Grading:

- +2 points for the initialization
- +1 points for using the standard recurrence
- +2 points for explaining how to find the maximum score
- +1 points for the time complexity

(b) We will use a dynamic programming table D of size nm here. The interpretation is that $D[i, j]$ gives the maximal score for aligning the suffix of $T[1, j]$ against $P^k P[1...i]$ for some k.

We will now initilize the first row of the matrix differently allowing also a new repeat of P to start. Essentially we will allow the alignment DAG to wrap around from the last row to the first one:



The recurrence relation corresponding to computing the first row (for other rows the recurrence does not change) of the matrix:

$$D[0, j] = \max \left\{ \begin{array}{l} 0 \\ D[m, j] \end{array} \right.$$

We will fill the matrix in comlumnwise manner. The problem here is that the value $D[m, j]$ is not available when we compute $D[0, j]$. However, we can argue that the maximum value in column j cannot be obtained from the previous value in the same column (considering the column circular) because deletions always decrease the score. So ignoring the value $D[m, j]$ in initialization will give us at least one correct value in column j and the rest of the values cannot be greater than the correct value. Furthermore all values computed after the correct value are also correct and so in particular $D[m, j]$ will be computed correctly. Therefore doing a second round of computation using the correct value for $D[m, j]$ for each column, we will compute the scores correctly. (This was actually more complicated than I thought...)

The maximum score can now be obtained by finding the maximum value in the last row of the table. The actual alignment can be found by tracing back in the matrix and this will also reveal the value for k.

There are nm cells in the matrix and each will be computed twice. Therefore the runtime of the algorithm is $O(nm)$.

Grading:

- +2 points for using an $n \times m$ table
- +4 points for understanding the wrap around idea
- Some points were reduced for minor errors in the details of the algorithm.

- The evaluation of the recurrence was more involved than intended and thus no points were reduced for not getting that right.

4. **Your choice.**

   Choose one of the (non-trivial) problems studied during the course (in study groups, lectures, or/and exercises) not related to the three assignments above. Define the problem (input, output), explain how the problem is motivated by molecular biology, and describe an algorithm for the problem either simulating an example or by giving its pseudocode.

   Solution. The most popular problem was shortest common superstring. Other choices were partial digest problem, sequencing by hybridization, motif finding problem and median string problem.

   Grading:

   - +4 points for correct definition
   - +4 points for correct simulation or pseudocode.
   - +4 points for correct motivation.
   - Some points reduced when the description was not clear enough, or mistakes in simulation.