Biological Sequence Analysis (Spring 2015)

Exercise 2 (Thu 22.1, 10-12, B222, Veli Mäkinen)

Do any 5 assignments from below. If you are not very familiar with alignments from your earlier studies, focus on the first 5; otherwise, last 5 should be challenging enough.

1. **Global alignment simulation.**

   Compute the global alignment scores s_{ij} and especially $s_{mn} = S(A, B)$ for $A =$`ACCGATG` and $B =$`ACGGCTA` using indel penalty $-d$, where $d = 1$, and the substitution matrix:

$s(a,b)$	'A'	'C'	'G'	'T'
'A'	1	−1	−0.5	−1
'C'	−1	1	−1	−0.5
'G'	−0.5	−1	1	−1
'T'	−1	−0.5	−1	1

   Trace an optimal alignment.

   Rather than simulating on paper, you can also opt to implement. You may use the edit distance code at `http://www.cs.helsinki.fi/group/gsa/book/implementations/dp2tikz.py` as a basis. The resulting tikz-code can be included in a `LaTeX` document to visualize the result:

```
\documentclass{article}
\usepackage{tikz}

\begin{document}

\begin{tikzpicture}[scale=6.0]
\input{dp.tikz}
\end{tikzpicture}

\end{document}
```

2. **Local alignment simulation.**

   Compute the local alignment scores l_{ij} for the same example as above. Trace an optimal local alignment.

3. **Space improvement I.**

   Give pseudocode (or e.g. python code) for global alignment algorithm using only space $O(m)$ to compute $S(A, B)$.

4. **Space improvement II.**

   Give pseudocode (or e.g. python code) for tracing an optimal path for maximum scoring local alignment, using space quadratic in the alignment length.

5. **Shortest detour.**

   The shortest detour algorithm assumes you can fill only the diagonal zone of the dynamic programming matrix. How can you do this without allocating memory for the whole matrix (d_{ij})? *Hint.* Coordinate change helps; allocate a matrix only containing the diagonal zone. Find a bijective map between cells in d_{ij} and cells in your new matrix.

6. **Sparse dynamic programming I.**

   Next week we will study sparse dynamic programming and we exploit a *range minimum query* data structure (outside the lecture script material):

   Lemma 3.1. The following two operations can be supported with a balanced binary search tree $\mathcal{T}$ in time $O(\log n)$, where n is the number of leaves in the tree.

   > $\texttt{update}(k, \texttt{val})$: For the leaf w with $\texttt{key}(w) = k$, update $\texttt{value}(w) = \texttt{val}$.
   >
   > $\mathsf{RMQ}(l, r)$: Return $\min_{w\,:\,l \le \texttt{key}(w) \le r} \texttt{val}(w)$ (*Range Minimum Query*).

   Moreover, the balanced binary search tree can be built in $O(n)$ time, given the n pairs $(\texttt{key}, \texttt{value})$ sorted by component $\texttt{key}$.

   Prove the lemma formally or give an example how the proposed structure works e.g. on 8 (value,key) pairs stored in its leaves and by visualizing the computation of some range minimum query for some non-empty interval.

7. **Sparse dynamic programming II.**

   A *van Emde Boas tree* (vEB tree) supports in $O(\log \log n)$ time insertions, deletions, and *predecessor/successor queries* for values in interval $[1, n]$. Predecessor query returns the largest element i' stored in the vEB tree smaller than query element i. Successor query returns the smallest element i' stored in the vEB tree greater than query element i. Show how the structure can be used instead of a balanced search tree of Lemma 3.1 to solve range minimum queries for semi-infinite intervals $(-\infty, i]$ (i.e. for the type of queries we use e.g. in the LCS algorithm to be studied next week).

8. **Space improvement III.**

   Develop an algorithm for tracing an optimal path for maximum scoring local alignment, using space linear in the alignment length. *Hint.* Let $[i', i] \times [j', j]$ define the rectangle containing a local alignment. Assume you know j_{mid} for row $(i - i')/2$ where the optimal alignment goes through. Then you can independently recursively consider rectangles defined by $[i', (i - i')/2] \times [j', j_{\text{mid}}]$ and $[(i - i')/2] \times [j_{\text{mid}}, j]$. How to compute j_{mid}?