Model Solutions for Study Group 5 (Algorithms for Bioinformatics)

12.10.2011

1. Lex-BFS order: Like you saw in the study group, except without already given labels. During the breadth-first search, you *define the label* of a node v by its already visited neighborhood; details follow. Usually the algorithm is described backwards. Set all $label(v) = \varepsilon$ for all nodes v, where ε is an empty string. Then pick any node v numbering it $i = n$. For all egdes (v,w) do $label(w) = label(w)i$ (string concatenation in alphabet $\Sigma = \{1,2,\ldots,n\}$). Continue decreasing i from $n-1$ to 1 so that at each step you pick the node v with lexicographically largest label, number it i, and apply $label(w) = label(w)i$ for all edges (v,w) such that w is not assigned a number in earlier steps.

   For an example, consider graph with edges $(v,w),(v,e),(w,e),(w,f),(e,f)$. Pick v numbering it 4. Set $label(w) = label(e) = 4$. Pick w (solving tie arbitrarily) numbering it 3. Set $label(e) = label(e)3 = 43$ and $label(f) = 3$. Pick e numbering it 2. Set $label(f) = label(f)2 = 32$. Pick f numbering it 1.

   Notice that with a tree as the input, this algorithm is identical to BFS (breaking ties arbitrarily).

   To implement the algorithm to work in linear time, one can for example, use a trie data structure to record the growing labels. A node with path label α in the trie corresponds to a subset of nodes in the graph whose label is prefixed α. One needs to maintain a pointer from graph node v to the corresponding leaf v' of the trie. Then concatenation of the label is constant time operation: unless there already is an edge (v',v'') with label i in the trie, add a new leaf v'' and edge (v',v'') with label i, and assign v to point to v''. Notice that if there is such an edge, it was just added, so keeping a linked list of children is enough to do this step in constant time. Left-most leaf of the trie corresponds to the lexicographically largest label. Removing the left-most leaf from the trie (when the last graph node that was pointing to it is assigned a number) takes *amortized* constant time since each edge of the trie is visited at most three times: once when it is on the left-most path and you are looking for the new left-most leaf after removal of the previous, once when you insert the edge, and once when you delete the edge.

   In the trie above, let us call *active* those nodes that have pointers from graph nodes. Reading them in *preorder* forms a *partitioning* for the unnumbered nodes in the graph: each active node corresponds to a distinct subset of unnumbered graph nodes that point to that leaf. It follows that maintaining these distinct subsets as a (doubly-)linked list L is actually enough. Then at each step of the algorithm a node v is picked from the left-most subset (removing it from L when empty). For all (v,w) leading to an unnumbered node w, one can locate the corresponding subset S using a pointer, as in the trie. Let $W = \{w \mid w \in S, (v,w) \text{ is an edge}\}$. Subset S is partitioned into W and $S \setminus W$, placed in this order in the place of S in L. This is done for all S containing w such that there is an edge (v,w). Notice the connection to the trie; a preorder traversal after concatenation of labels will produce the same partitioning as this one.

   This latter algorithm is called *partitioning refinement* (for good reason).

   For our example, the partitioning refinement would look like this: $[\{v,w,e,f\}] \to [\{w,e\},\{f\}] \to [\{e\},\{f\}] \to [\{f\}]$.

2. Skipped.

3. Well-covered in the study group. To make this linear time, you need to maintain a doubly-linked list L of *active* nodes, i.e., nodes that have still unvisited edges left. Then you can pick the tail of the list to start the next cycle. When a node becomes in-active, i.e. all its edges are visited, you simply follow a pointer to its place in L and remove it.

4. Create a bipartite graph with genes of A and genes of B as nodes, and homolog-relationships as edges. Let $score(a,b)$ denote the alignment score between a and b. Let MAX be the maximum score. For each edge (a,b) of the graph, assign weight $w(a,b) = \text{MAX} - score(a,b)$. Let there be m genes in A and n genes in B, $m \leq n$. Add $n-m$ dummy nodes each pointing to all genes of B with weight 0, Now, solving *minimum weight perfect matching* (the same as was used in the shortest superstring approximation) on the graph results into a matching (if one exists) with $n-m$ dummy nodes matched with cost 0, and all m nodes of A matched with minimum total weight w, which equals maximum score $m * \max - w$. Notice that if there exists a one-to-one matching in the original graph (without

the dummy nodes) matching all nodes of A then there exists an equal cost matching with the dummy nodes added, and vice versa. Hence, this reduction solves the problem. With the matching given, you can number genes in A from left to right to the identity permutation of length m, and following the edges constituting the matching to create the corresponding permutation for B. This permutation is the input for the gene rearrangement problem.

5. Skipped, but here are the solutions.

   The first solution has time complexity $O(|\Sigma|^\ell + |s|)$:

   (a) Initialize an array of size $|\Sigma|^\ell$ to zero-values. This takes $O(|\Sigma|^\ell)$ time.

   (b) We use a sliding-window of length ℓ and show that each step can be computed in constant time if we know the value of the previous window: Let $\Sigma = \{0,1,2,\ldots,|\Sigma|-1\}$ denote our alphabet (i.e. map each symbol to an integer). We use the following function to map each window of symbols $x_1x_2\cdots x_\ell$ to unique position in our array:

   $$h(x_1x_2\cdots x_\ell) = x_1 + x_2|\Sigma| + x_3|\Sigma|^2 + \cdots + x_\ell|\Sigma|^{\ell-1} = \sum_{i=1}^{\ell} x_i|\Sigma|^{i-1}$$

   Now if we know the value of the i-th window, that is $h(s_is_{i+1}\cdots s_{i+\ell-1})$, the value of the next window $i+1$ can be computed in constant time:

   $$\begin{aligned} h(s_{i+1}s_{i+2}\cdots s_{i+\ell}) &= \left\lfloor \frac{h(s_is_{i+1}\cdots s_{i+\ell-1})}{|\Sigma|} \right\rfloor + x_{i+\ell}|\Sigma|^{\ell-1} \\ &= \left\lfloor \frac{x_i}{|\Sigma|} \right\rfloor + x_{i+1} + x_{i+2}|\Sigma|^1 + \cdots + x_{i+\ell-1}|\Sigma|^{\ell-2} + x_{i+\ell}|\Sigma|^{\ell-1} \end{aligned}$$

   The first term $\left\lfloor \frac{x_i}{|\Sigma|} \right\rfloor$ is zero because $x_i < |\Sigma|$ for all $x_i \in \Sigma$. The rest of the terms sum up to $h(s_{i+1}s_{i+2}\cdots s_{i+\ell})$. These arithmetic operations (one division and one addition) take constant time if we assume the RAM model of computation and large enough computer-word size. This is repeated for all windows over s and takes in total $O(|s|)$ time.

   (c) For each window, the array position given by the $h()$ function is incremented by one (requires constant time in random access model).

   The second solution has time complexity $O(|s|)$:

   (a) Build a suffix tree for s. This requires $O(|s|)$ time.

   (b) Traverse the tree in bottom-up manner to store frequencies for each internal node: all leaf nodes have frequency 1, and the frequency of an internal node is the sum of its children's frequencies. Since there are at most $O(|s|)$ nodes in the suffix tree, this traversal requires $O(|s|)$ time in total.

   (c) Now traverse the whole suffix tree top-down. For each internal node at string-depth $\geq \ell$ and whose parent is at depth $< \ell$, output the substring corresponding to the node and its frequency. Again, there is at most $O(|s|)$ nodes to traverse, thus, the total time is $O(|s|)$.

6. Skipped, but here is one possible solution.

   Let $M = m_1\cdots m_k$ be the measured spectrum and $T = t_1\cdots t_n$ the theoretical spectrum, with m_i and t_j being the masses. A good distance measure $d(M,T)$ could be the minimum cost of insertions, deletions and substitution of peaks (masses) to convert M to T, but the costs of the operations need to be adjusted. A natural substitution cost is $|t_j - m_i|$, or alternatively 0 if $m_i + \delta = t_j$ otherwise ∞, where $\delta \in \Delta$ is the mass of a lost molecular fragment from the measured fragment corresponding to m_i and Δ is the set of possible losses. Let us derive suitable insertion and deletion costs for the former case. A missing mass t_j after m_i may be due to being too close to m_i after the loss of mass, and thus being detected simultaneously. Insertion cost could then be $t_j - m_i$. An extra mass m_j in M has no counterpart in T, and its deletion should cost 0 to filter out chemical noise without any cost. The dynamic programming recurrence for the computation of $d(M,T) = d_{k,n}$ becomes $d_{i,j} = \min\{d_{i-1,j-1} + |t_j - m_i|, d_{i,j-1} + |t_j - m_i|, d_{i-1,j}\}$. However, there are many alternatives with similar arguments.

7. Skipped.

8. This can be seen by induction. Assume first, for contradiction, that two leaves i and j under the same parent with minimum d_{ij} over such leaf pairs in the correct ultrametric tree are *not* assigned this way by the UPGMA algorithm. That is, d_{ij} is not the minimum picked by the algorithm at any step. Then $d_{il} < d_{ij}$ or $d_{jl} < d_{ij}$ for some l must be picked and forces i and j to go under different parent. However, this a contradiction since d_{ij} should be the minimum in the beginning. Consider now the ultrametric tree with i and j removed making their parent k a leaf. The same thinking as above can be repeated for this tree, considering the new pair i and j with minimum d_{ij}, and so on.

9. Skipped.