Example Homework Solutions

Dan DeBlasio

1 Given a string S and an integer k, find the length and position of the first k (lexicographically) positions of this longest subsequence in S that occurs at least k times in $O(nk)$ time, where $|S| = n$.

Algorithm:

1. Construct the suffix array, a, and the longest common prefix array, lcp, for S.
2. Define two new variables max_ℓ and max_c which will hold the length and starting position in a of the current longest sequence (initialize these variables to 0).
3. For increasing starting positions $1 \leq i \leq n-k$ in a, let $\ell_i = min\{lcp[i], lcp[i+1], ..., lcp[i+k-1]\}$. if $max_\ell < \ell_i$ then set $max_\ell = \ell_i$ and $max_c = i$.

The maximum sequence length will be in max_ℓ, and $a[max_c], a[max_c+1], ..., a[max_c+k]$ will be the locations in S of the lexicographically minimum substrings that contain this string.

The longest common prefix for a pair of suffixes in a is the minimum of the lcp values of those prefixes, since the suffixes are in lexicographic order that same prefix must be shared by the suffixes in between them. Therefore, by finding the set of $k-1$ locations in lcp that has the largest minimum value, we will have found the longest substring that repeated k or more times. Because the condition in Step 3 requires max_ℓ to be strictly less than ℓ_i, if there are strings of length max_ℓ that occur more than k times, or more than one string that occurs k times, max_c will be set to the minimum position in a where this limit is first encountered. a is in lexicographic order, therefore the returned location is the lexicographic minimum such set of prefixes.

Assume we have run our algorithm and we are at step 4. Let there exist some other substring of length $\ell' > max_\ell$ that appeared k times. The suffixes that begin with that string would be grouped together in a, call the starting position of these suffixes i'. Because all k of these suffixes share an i length prefix, the first pair satisfy the following: $S[a[i']...a[i'] + \ell'] = S[a[i'+1]...a[i'+1] + \ell']$ and $lcp[i'] \geq i$. Similarly $lcp[i'+1] \geq i, ...lcp[i'+k-1] \geq i$. Therefore the min operation in step 3, when $i = i'$ would result in the value $\ell' > max_\ell$ and would have triggered the resetting of the values max_ℓ and max_c. This contradicts the original assumption that the algorithm ran to completion, and therefore a longer substring does not exist.

The running time of step 1 for a string of length n is $O(n)$ as given by Manber and Myers. Steps 2 is an $O(1)$-time operation. Step 3 can be performed in $O(kn)$-time: at each position i in a we compare k values meaning the time for each position is $O(k)$, and we examine $n-k+1$ starting positions i which is $O(n)$. Therefore the total running time is $O(kn)$.

2 Given a binary graph connection matrix g such that $g_{ij} = g_{ji} = 1$ if nodes i and j are connected, and a weight w_i for each node. Formulate an ILP to find a subset of nodes (using a binary indicator variable x_i) that minimizes the total weight left out of the set (i.e. the sum of the non-included nodes) but does not allow any neighbors. Assume there are n nodes in the graph,

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{1 \le i \le n} w_i(1 - x_i) & \\ \text{subject to} & x_i + x_j \le 1, & 1 \le i, j \le n \;\&\; g_{ij} = 1 \\ & x_i \in \{0, 1\}, & 1 \le i \le n \end{array}$$

The first set of constraints ensures no neighboring nodes from the graph are included in the set. If two nodes are connected (i.e. $g_{ij} = 1$) then there will be an equation in the program stating that the sum of the indicators must be at most 1. Since the x_i and x_j variables can only be 0 or 1 (due to the final set of constraints) either only one or neither one can be included in the set. This means the constraint set limit the assignments to those that both only select non-neighbors and can only make binary inclusion choices.

If a node i is not included in the set the variable x_i will be 0, therefore the value of $(1 - x_i)$ will be 1. Therefore the sum defined in the objective function finds the sum of the weights for the nodes *not* included in the set (as specified in the problem). Thus by minimizing this sum under the conditions above, we are finding the set of is optimal for the problem description.