Question 1. The answer is "yes". If the undirected version is connected then it has a spanning tree T. Let (v, u) be any directed edge that appears (in undirected form) on T: Starting with that directed edge (v, u), keep walking along never-used directed edges of G. Because there are finitely many edges, the process eventually stops at a vertex that has run out of fresh outgoing edges (all have already been used up). But that vertex has to be v, because every other vertex w has had its edges used up in pairs: 1 incoming when we enter it, followed by 1 outgoing when we exit it, and because there are as many incoming as outgoing we cannot "get stuck" at such a w (i.e., we can only get stuck at the v where we started). This cyclic walk that started at v, took edge (v, u), and eventually ended back at v, has gone through vertices that are clearly all in the same strongly connected component. Therefore v and u are in the same strongy connected component. As the above logic applies to every edge that appears in the spanning tree T, all the vertices of T (hence of G) are in the same strongly connected component.

Question 2.

1. Sort the points by increasing x coordinate, and let L_y be the resulting sequence of y coordinates. Compute the LIS of L_y: The length of this LIS is the α we seek.

2. Sort the points by decreasing x coordinate, and let L_y be the resulting sequence of y coordinates. Compute the LIS of L_y: The length of this LIS is the β we seek.

   Alternative solution. Sort the points by increasing x coordinate, and let L_y be the resulting sequence of y coordinates. Replace, in L_y, every y coordinate y_i by $-y_i$. Use the LIS algorithm on the resulting sequence: The length of the LIS is the β we seek.

3. For each point $p_i = (x_i, y_i)$, let S_i be the subset of S whose x coordinates are $\leq x_i$ (hence S_i includes p_i). Let λ_i be the number of points in a largest subset of S_i that contains p_i and is such that all the points in it are comparable (that is, λ_i is the size of a solution in S_i that is constrained to contain p_i as its rightmost point). Note that $\alpha = \max_{1 \leq i \leq n} \lambda_i$. If $\alpha \geq \sqrt{n}$ then $\max\{\alpha, \beta\} \geq \sqrt{n}$ and we are done. So suppose that $\alpha < \sqrt{n}$. For each $t \in \{1, 2, \ldots, \alpha\}$ let Θ_t denote the set of p_is in S whose $\lambda_i = t$. Observe that every pair of distinct points p_i, p_j for which $\lambda_i = \lambda_j$ must be incomparable (because if p_i dominated p_j then we would have $\lambda_i > \lambda_j$). This means that all the points in a set Θ_t are incomparable, and therefore $\beta \geq |\Theta_t|$ for all $1 \leq t \leq \alpha$. Therefore it suffices to show that at least one of the Θ_t's contains more than $\sqrt{n}$ points. If you view the n points as pigeons, and view each Θ_t as a pigeonhole, then the fact that there are n pigeons and fewer than $\sqrt{n}$ pigeonholes implies that at least one pigenhole contains more than $\sqrt{n}$ pigeons, completing the proof.

Question 3. To perform an $Increment(i, j, x)$ we locate the ith leaf (call it l_i) and the jth leaf l_j (in constant time each). Then in $O(h)$ time we find their lowest common ancestor (call it f) and the $O(h)$ internal nodes that are either right siblings of nodes on the f-to-l_i

path or left siblings of nodes on the f-to-l_j path. For each such node v we add x to its $\delta(v)$. A $Decrement(i, j, x)$ is performed similarly except that we subtract x instead of adding it.

Correctness of the above follows from the fact that the path between the root and a leaf whose associated item is to be modified, goes through exactly one of the nodes whose δ value was modified.

Question 4. The first idea that comes to mind is to use the convolution method (explained in class) modified in the following way: When setting up the convolution problem for calculating the contribution, to the score vector C, of a particular alphabetical symbol γ, you replace every occurrence of symbol γ and of symbol # by 1. You do this 4 times, once for every $\gamma \in \{a, b, c, d\}$.
However, adding the above 4 contributions together gives an overestimate of C because you count 4 times the contribution of a # matching another #. There are two ways to deal with this:

- *Alternative 1:* You do as in the above anyway, and then later you correct for the overestimate by calculating the contribution (call it $C^{(\#)}$) of all #-to-# matches (by doing a separate convolution for it in the usual way: You replace every # by 1 in both T and P and everything else by 0, etc). You then correct by subtracting 3 times $C^{(\#)}$ (because the #-to-# matches were counted 4 times). The second alternative (below) is probably better.

- *Alternative 2:* You first use the convolution method without any modification, effectively ignoring the # symbols by replacing them with zeroes (so you do one convolution for each $\gamma \in \{a, b, c, d\}$). This gives you the effects of matches that do not involve any # symbols. Then you do 2 seperate convolutions to calculate the effects of matches that involve # symbols: For the first convolution you replace only the #'s in T by 1, and everything in P by 1 (you can actually do this "convolution" in linear time rather than in $O(n \log n)$ time, because all of P was replaced by 1's). For the second convolution you replace everything except the #'s in T by 1, and only the #'s in P by 1. The first convolution captures the effect of matches of #'s in T with anything in P (including #'s), the second one captures the effect of matches of the 4 alphabetical letters in T with #'s in P. Note that there is no double-counting.

Of course, once we have the C vector, the entries of C equal to n correspond to occurrences of P in T.