1. a)

   Let s^* be a feasible solution (assuming one exists). For every $j \in [n]$ clearly the set $\{s_j^1, s_j^2, \ldots, s_j^k, s_j^*\}$ has size at most $k+1$. But it is easy to see that we can replace s_j^* by one of s_j^i for $i \in [k]$ without affecting whether s^* is a feasible solution or not. Thus we can "rename" the alphabet in each column to be characters $\{1, 2, \ldots, k\}$ and hence we can bound $\ell \leq k$.

   b)

   After the renaming above, each of the columns of the input will look of the input matrix will be a vector of characters from an alphabet of size at most k and a column has length k. Thus there are at most k^k different ways a column can look. If there are duplicate columns on input, we can consolidate them into one, since we can solve all of them using the same character. We write $K = k^k$.

   c)

   Nothing to do here.

   d)

   For $c \in [\ell]$, $t \in [K]$ and $j \in [k]$ let $\Delta(c, t, j)$ denote whether the j-th character in column type t **differs from** c or not. For a zero-one variable $x_{t,c}$ with t and c defined as before, $x_{t,c}$ will be 1 iff the output character for columns of type t should be character c. Thus the first set of constraints we write is that for every column type $t \in [K]$ there is exactly one output character:

   $$\sum_{i=1}^{\ell} x_{t,i} = 1. \tag{1}$$

   And then there is one gigantic constraint that says whether the output string has Hamming distance at most d from each of s^j's:

   $$\sum_{c \in [\ell]} \sum_{t \in [K]} \Delta_{c,t,j} x_{t,c} \leq d. \tag{2}$$

   You might now be used to the fact that there is no objective function ;) It takes time $k^{\mathcal{O}(k)}$ to write down this ILP. And we can bash it with Lenstra's algorithm to get something with running time $K^{\mathcal{O}}(K)$ which is something like $k^{\mathcal{O}(k^{2^k})}$ (double exponential in k in case it is not visible in print).

   e)

   Constraints (2) are the top row constraints of an n-fold integer program and constraints (1) are the local off-diagonal constraints. Thus we in fact get a single exponential algorithm in k.

   f)

   Instead of all constraints (2) having the same right hand side d, it is d_j.

   (This is a hard exercise in the textbook but using the power of integer programming we solve it basically for free. Hooray for integer programming.)

2. We solve this exercise using bidimensionality. Observe that a $k \times k$ grid has a vertex cover of size at least $\Omega(k^2)$: in every row of the minor we have to take every second vertex into the vertex cover. Thus we apply the excluded grid minor theorem and if we find a grid minor of size $\Omega(\sqrt{k}) \times \Omega(\sqrt{k})$, then we can immediately conclude that the answer is no (same multiplicative constant between the Ω's in this sentence). Otherwise, the graph is treewidth at most $\mathcal{O}(\sqrt{k})$, then we apply the treewidth dynamic programming procedure we all know and love.