1. A. The number of uncompressed characters, the number of bits for the arithmetic encoding, and the encoding itself.

B. Asymptotically, the move to front method outputs 1 1 2 2 2 repeated over and over again. Depending on how probabilities are generated, the bits per symbol could vary…but we can likely store approximately 1 bit for each span of 1’s and 2’s, which is 2/5 bit per character.

C. Asymptotically, the move to front algorithm outputs all fours. Over time, then, the probability will approach 1, and consequently, the bits per symbol will approach 0.

D. Odds are that it would work about equal on all types, since each has a relatively small set of “common characters”.

E. Two symbols at a time would likely better match the phoneme layout of English, but would likely do much poorer on non transition words (and, as, it, or …), since they are not repeated often enough, and the probabilities are now distributed across a much larger symbol space, causing each “unlikely” term to be significantly more expensive.

2. A. Find the inverse of P, and transform the codeword to get \( wP^{-1} = xSG + eP^{-1} \) since \( eP^{-1} \) is still of weight t, we can ignore it and run the Goppa algorithm to get \( xS \)

Find x by multiplying \( xS \) by \( S^{-1} \).

B. By quickly, I assume that the goal is to decode \( w \) in time similar to the time it takes to legitimately decode a codeword with the private key. If we have unbounded memory and preprocessing time, we can create a hash of all codewords and their resulting messages by iterating over the message space and encoding without the error vector. Because the Goppa code generates codewords with a hamming distance is 2t+1, we know that the hashed codeword that matches the highest number of bits in the incoming codeword is the correct one.

Since there are \( 2^k \) codewords, we can put an upper bound on our memory usage at \( 2^kn \) bits (store each resulting codeword of n bits).

C. To provide 10 bit error correction, the encoder need merely use an error vector of weight t-10 (in this case 40 bits). If an attacker can somehow capture messages on
errorless channels, or at least with high probability, he can reduce his amount of computation to determine the error bits from being proportional to $10^1\binom{50}{40}$.

D. The public key size is over a thousand fold larger (quadratic in number of bits). The encoding nearly doubles the size of the data. As an advantage, it is much much faster to encode and decode.

3. A. No, since this requires a concave corner on our surface
   B. No, since that would require a cycle as defined by the problem, which is invalid by part A.
   C. It corresponds to shifting a portion of a given $x_{i,j}$ to connected edges in the graph.

5. A linear time algorithm for Hidden Markov Models like this is the Viterbi algorithm defined recursively as follows:

$$V(i,t) = P(y_t|Q_t = i) \max_j \{P(Q_t = i \mid Q_{t-1} = j)V(j,t-1)\}$$

With $V(i,1) = P(y_1 \mid Q_1 = i)P(Q_1 = i)$. If we keep pointers at each step to the preceding timestep’s state that optimally generated this state, we can determine the path by stepping backwards from the final state. This algorithm runs in time linear to the length of our string, in actuality $O(sn)$ where $s$ is the number of states in the HMM.