

## Problem 4

**General Idea:** Create a DAG for which (1) each path from the root to a vertex  $V$  (that is  $k$  steps from the root) represents the portion of the message up to the  $k^{\text{th}}$  bit; (2) each vertex  $V$  (that is  $k$  steps from the root) contains the states  $(s_{k-1}, m_k)$  associated with the  $\tilde{c}_k$ ; and (3) each edge from  $(s_{k-1}, m_k)$  to  $(s_k, m_{k+1})$  represents the different ways of going from one realization of the  $k^{\text{th}}$  vertex to the  $k+1^{\text{th}}$  vertex and has weight  $-\log(P(\tilde{c}_k|m))$ , where  $P(\tilde{c}_i|m) = \begin{cases} 0.97 & f(s_{i-1}, m_i) = \tilde{c}_i \\ .01 & \text{otherwise} \end{cases}$ .

Hence, to find the path of maximum probability is to find the path with  $m_1, m_2, \dots, m_n$  such that  $P(\tilde{c}_1|m)P(\tilde{c}_2|m)\dots P(\tilde{c}_n|m)$  is maximized. To maximize the product is to minimize the negative log, hence each edge has the weight attached as previously described. Therefore, we just need an algorithm that builds the DAG and we can use the standard algorithm for finding the shortest path between two nodes in a DAG to find the shortest path between the source and the sink, keeping track of the  $m_i$ s along the way (not including the source and the sink which don't have any). The sequence of  $m_i$ s along the optimal path is the most likely original code.

**Algorithm:** Building the DAG:

```
Build(State[],  $\tilde{c}$ ):
```

```
  Let G = DAG containing just a root element
  for i = 1 ... n-1:
```

```
    foreach ith state  $v_i$ :
```

```
      Let  $(s_{i-1}, m_i) = \text{State}[v_i]$ 
```

```
      Add edge  $E_1 = (v, \text{Helper}(i, (s_{i-1} + m_i \text{ modulo } 4), 1))$ 
```

```
      Set  $l(E_1) = -\log(P(\tilde{c}_i|(m_1, m_2, \dots, m_{i-1}, 1)))$ 
```

```
      Add edge  $E_2 = (v, \text{Helper}(i, (s_{i-1} + m_i \text{ modulo } 4), 0))$ 
```

```
      Set  $l(E_2) = -\log(P(\tilde{c}_i|(m_1, m_2, \dots, m_{i-1}, 0)))$ 
```

```
  Let Sink = new Node
```

```
  foreach nth state v:
```

```
    Add edge  $(v, \text{Sink})$  whose weight is 1
```

```
  Return G
```

```
Helper(i, (s,m)):
```

```
  if there already exists a vertex with the same state (s,m) that is i edges away fi
```

**Proof of Correctness:**

First, that we correctly build the DAG: At the  $i$ th iteration of the loop, we look at all the possible states (the two numbers that determine the real character) and create a vertex for each state we can reach from the  $i$ th states if they don't exist (two states in the  $i$ th iteration may have edges to the same state in the  $i+1$ th state), and create an edge between each state  $S$  in the  $i$ th iteration and each state that is one symbol from  $S$ . Therefore,  $((u, v) \in E) \Leftrightarrow (v \text{ is one symbol from } u)$ , as desired.

**Running Time:**  $O(n)$ .

Since there can be two edges max coming out of each vertex/state (based on whether the bit is 1 or 0),  $|E| = \theta(|V|)$ . The distance from the source (root) to the sinks is  $n$  edges because each edge represents one potential bit that  $m$  can have, and at each level at most every possible state is represented. There are 8 possible states, so  $|V| \leq 8n$ , which means  $|V| \in O(n)$ . Therefore, finding the shortest path between root and each of the 8 possible states at the very end takes  $8 \times O(|E| + |V|) = O(|V|) = O(n)$  time. To prove the linearity of the whole operation, which is one call to Build and one call to find the shortest path, I simply need to show that Build() is linear.

We start by noting that the instructions in the first foreach are constant with respect to  $n$ ; we do 2 additions, 2 modulo, 2 logarithms and 2 calls to  $f()$ . Then we note that for each iteration of the for loop, we do the instructions in the foreach at most 8 times since there are at most 8 states at any given level. Then the for loop runs  $O(n)$  times and the rest runs in constant time, so the whole algorithm Build is linear.