Readings

- Lecture Notes Chapter 15: Huffman Coding
- Lecture Notes Chapter 16: Graph Traversals: BFS

Review: Huffman Coding

The motivation behind Huffman Coding is to encode and decode characters as bits, minimizing the average bits per letter (ABL). Furthermore, we seek a prefix-free code, where no encoding is a prefix of another — implying that a bit sequence can be parsed and decoded without any ambiguity.

The Huffman algorithm is a greedy algorithm that does this. Given a set of characters and their frequencies, the algorithm outputs an encoding by repeatedly merging the 2 nodes with the smallest frequency values until only one node remains. This one node is the root of the Huffman tree, whose leaves are characters and each root-to-leaf path is an encoding of that character. Furthermore, this tree is a full binary tree, where each internal node has exactly 2 children. Therefore, the Huffman algorithm produces an optimal and prefix-free encoding that minimizes the ABL.

The running time of the Huffman algorithm is $O(n \log n)$ if we utilize a min-heap to find the 2 nodes with minimum frequency in each step, as seen in the pseudocode. This is because at each step, we perform a constant number of EXTRACT-MIN and INSERT operations, which take $O(\log n)$ time, and we repeat this for $O(n)$ iterations.

Review: Graph Representations

Let $G = (V, E)$, where $|V| = n$ and $|E| = m$. In other words, $G$ contains $n$ vertices and $m$ edges.

Adjacency Matrix

We can represent $G$ with an $n \times n$ matrix $A$, where $A[i][j] = 1$ if there is an edge $(i,j)$ and 0 otherwise. The primary advantage of an adjacency matrix is that we can check whether or not there is an edge between two vertices in $O(1)$ time since all we have to do is index into the matrix. On the other hand, the matrix uses $\Theta(n^2)$ space, and it takes $\Theta(n)$ time to enumerate the neighbors of a vertex $v$ since we must iterate through an entire row in the matrix.

Adjacency List

We can also represent $G$ as an adjacency list, where we have an array $A$ of $|V|$ lists such that $A[u]$ contains a LinkedList of vertices $v$, such that for every vertex $v_i$ in $v$, $(u, v_i) \in E$. (Note that if we have a directed graph, we can store two LinkedLists at index $u$, one for $u$’s in-neighbors and one for $u$’s out-neighbors.) The primary advantage of an adjacency list is that we only use $\Theta(n + m)$ space, which is better than the $\Theta(n^2)$ space usage of an adjacency matrix especially if $m \ll n^2$. Furthermore, it only takes $O(deg(v))$ time to enumerate the neighbors of a vertex $v$, but on the other hand, checking whether $(u, v) \in E$ also takes $O(deg(v))$ time since we have to traverse the LinkedList. In CIS 121, we usually use adjacency lists.
Review: BFS (Breadth First Search)

At a high-level, in BFS, we explore the graph in “layers,” so we explore “wide” before exploring “deep.” We begin at a node $v$ (layer 0); then explore the children of $v$ (layer 1); then the children of the nodes in layer 1 (which make up layer 2); and so on. In other words, we explore all nodes at layer $L_i$ before exploring any nodes at layer $L_{i+1}$. Because we explore the graph in layers, as seen in the pseudocode, we use a queue to implement this algorithm since it is FIFO. Note that the output of BFS is a BFS tree, and that as written, BFS is not guaranteed to visit every node in the graph.

The running time of BFS is $O(n + m)$. This is because we add and remove each node from the queue at most once, and for each node, we only examine its neighbors.

Problems

Problem 1

Construct an optimal Huffman coding for the following alphabet and frequency table $S$:

<table>
<thead>
<tr>
<th>Character</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>0.4</td>
<td>0.3</td>
<td>0.15</td>
<td>0.1</td>
<td>0.05</td>
</tr>
</tbody>
</table>

What is the ABL, or average bits per letter, for this encoding?

Problem 2

What are the pros and cons of Huffman Coding?

Problem 3

You have an alphabet with $n > 2$ letters and frequencies. You perform Huffman encoding on this alphabet, and notice that the character with the largest frequency is encoded by just a 0. In this alphabet, symbol $i$ occurs with probability $p_i; p_1 \geq p_2 \geq p_3 \geq ... \geq p_n$.

Given this alphabet and encoding, does there exist an assignment of probabilities to $p_1$ through $p_n$ such that $p_1 < \frac{1}{2}$? Justify your answer.

Problem 4

Design an $O(n + m)$ algorithm to find the shortest path between nodes $u$ and $v$ in a connected, unweighted graph.