CSE 421 Lecture 9

1 Prefix code

Theorem 1. *Given k symbols. There is a prefix code with length l_i for symbol i if and only if*

$$\sum_{i=1}^{k} 2^{-l_i} = 1.$$

We split the proof into two parts: showing the condition $\sum_{i=1}^{k} 2^{-l_i}$ is sufficient and necessary separately.

Lemma 2. $\sum_{i=1}^{k} 2^{-l_i}$ *condition is sufficient.*

Proof:
Algorithm:

- Insert each symbol as a leaf into priority queue Q by length.
- While $Q.size() > 1$
  - Remove 2 trees with largest lengths, call them x, y
  - Create a new tree z with x, y as its children with

$$l_z = l_x - 1.$$

  - Insert z into Q
- **Return** the only tree in Q

.

Runtime:
$O(k \log k)$. (k iterations, each iteration priority queue takes $O(\log k)$ time)

.

Proof: (by induction)
Let $P(k)$ be the statement "For k symbols, the algorithm outputs a prefix tree with required length if $\sum_{i=1}^{k} 2^{-l_i} = 1$."

Base $k = 1$. By the condition, we have $2^{-l_1} = 1$ and hence $l_1 = 0$. On the other hand, the algorithm indeed outputs a tree with only 1 vertex with length 0.

Induction: Suppose $l_1 \geq l_2 \geq \cdots \geq l_k \geq l_{k+1}$. Since $\sum_{i=1}^{k+1} 2^{-l_i} = 1$ and l_i are integers, we have that $l_k = l_{k+1}$.

Due to the priority queue, in the first step, the largest elements we pick are l_k and l_{k+1}. After merging element k and $k+1$ into one new element, z with $l_z = l_k - 1$, we have that

$$1 = \sum_{i=1}^{k-1} 2^{-l_i} + 2^{-l_k} + 2^{-l_{k+1}} = \sum_{i=1}^{k-1} 2^{-l_i} + 2^{-l_z}.$$

Since $(l_1, l_2, \cdots, l_{k-1}, l_z)$ satisfies the condition, induction hypothesis shows that the rest of the algorithm will output a prefix tree with these length. Since z is a tree with children k and $k+1$, the length for k and $k+1$ are also correct.

Lemma 3. $\sum_{i=1}^{k} 2^{-l_i}$ *condition is necessary.*

Proof:
Again, it can be proved by induction. In the induction step, we simply remove two leafs with largest length. The detailed proof is omitted here.

2 Correctness of Huffman's Algorithm

Lemma 4. *There is a prefix code T with minimum $cost(T)$ such that the 2 least frequent letters are siblings*

Proof:
Consider any prefix code T with minimum $cost(T)$
Let x and y be that two letters. Let $d(x)$ and $f(x)$ be the depth and frequency of x.
Without loss of generality, $d(x) \geq d(y)$. (the proof for $d(x) < d(y)$ is the same).
Let x' be the sibling of x. Let $f(x')$ be the frequency of x' (or its leaves).
Since x, y be two letters with least frequency. We have $f(y) \leq f(x')$.
When we swap y and x',
$>$ the length of letter y increased by $d(x) - d(y)$.
$>$ the length of letter x' (or its leaves) decreased by $d(x) - d(y)$.
Hence, the total cost is changed by

$$f(y)(d(x) - d(y)) - f(x')(d(x) - d(y)) \leq 0.$$

Hence, the new tree is still has minimum cost and x, y are siblings.

Theorem 5. *Huffman's algorithm produces a prefix code T with minimum $cost(T)$.*

Proof: (by induction)
Let $P(k)$ be the statement "For k symbols, Huffman's algorithm produces a optimal prefix code."
Base $k = 2$: There only one prefix code. So, the output is optimal
Induction:
Let T be the output of Huffman.
Previous Lemma shows that there is a prefix code T^* such that s_k, s_{k+1} are sibling and

$$\text{cost}(T^*) = \text{OPT}.$$

Let $s_1, s_2, \cdots, s_{k+1}$ be the symbols with its frequency $f_1 \geq f_2 \geq \cdots \geq f_{k+1}$.
Note that

$$T_- := T - \{s_k, s_{k+1}\}$$

is the output of Huffman with the symbols $s_1, s_2, \cdots, s_{k-1}, z$ where z is a new symbol with frequency $f_k + f_{k+1}$.
By viewing $T_-^* = T^* - \{s_k, s_{k+1}\}$ as a prefix code for symbols $s_1, s_2, \cdots, s_{k-1}, z$, we have

$$\text{cost}(T_-) \leq \text{cost}(T_-^*).$$

Since the symbol s_k, s_{k+1} has length 1 unit longer than z, we have

$$\text{cost}(T) = \text{cost}(T_-) + f_k + f_{k+1},$$
$$\text{cost}(T^*) = \text{cost}(T_-^*) + f_k + f_{k+1}.$$

Thus, we have

$$\text{cost}(T) \leq \text{cost}(T^*).$$

This proves T is optimal.

3 Party

.

Algorithm:

- Let S be the set of candidates for the party.
- Initialize $S = \{\text{everyone}\}$.
- Do
  - (a) $S = \{i \in S : i \text{ knows at least 4 people in } S\}$

- (b) $S = \{i \in S : i$ does not know at least 4 people in $S\}$

- while (if S changed)
- Return S

.

Runtime:
$O(m+n)$. We maintain

- the degree of each people in S.
- the list of people we are going to eliminate.

.

The cost of removing one people in S while maintaining the above is exactly degree of that people.

.

Correctness:
Let S^* be the set of people that can participate in any valid party.
By induction, one can show that each step of the algorithm, we have $S^* \subset S$.
To see this, look at step (a) and (b).

For step (a), if i knows less than 4 people in S, then i knows less than 4 people in S^* and hence i cannot be in any valid party. Hence, removing i in step (a) is fine.

For step (b), if i "does not know" less than 4 people in S, then i "does not know" less than 4 people in S^* and hence i cannot be in any valid party. Hence, removing i in step (b) is fine.

Next, we note that the output party is valid by the construction. Since $|S| \geq |S^*| \geq OPT$, S is the biggest party.