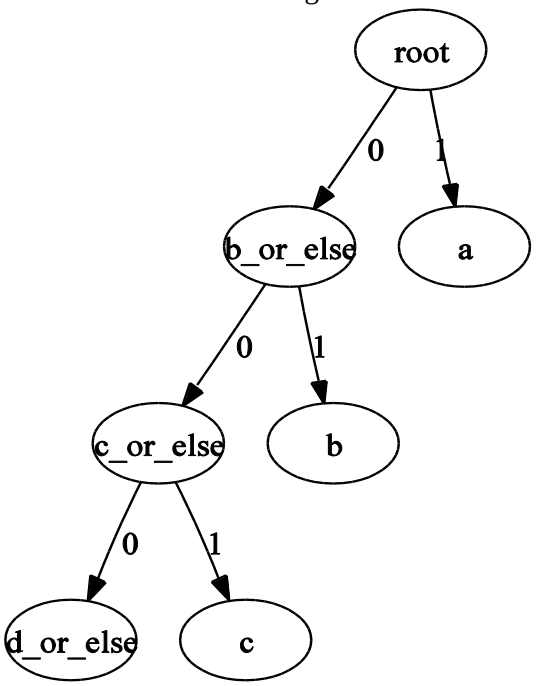
The Huffman code reduces to unary when the Huffman tree is left branching:



In general, β need not be ½. Without loss of generality, assume $\Pr(t) = (1-\beta)\beta^t$ where $\frac{1}{2} \le \beta < 1$ and $t \ge 0$. β depends on $E[t]$, the expected value of the interarrivals:

$$E[t] = \frac{P}{N} = \frac{\beta}{1-\beta} \Rightarrow \beta = \frac{E[t]}{1+E[t]}$$

Recall that the recipe above calls for expressing t as $m \cdot t_q + t_r$ where $t_q = \lfloor \frac{t}{m} \rfloor$ and $t_r = t \bmod m$. We encode t_q in unary and t_r in binary. (The binary piece consumes $\log_2 m$ bits, since t_r ranges from 0 to m.)

How do we pick m? For convenience, let m be a power of 2. The unary encoding makes sense as a Huffman code if $\beta^m \approx \frac{1}{2}$.

Thus, a reasonable choice[4] is $m \approx \left\lceil \frac{E[t]}{2} \right\rceil$. If $\beta = \frac{E[t]}{1+E[t]}$, then $\beta^m = \frac{E[t]^m}{(1+E[t])^m} \approx 1 - \frac{m}{E[t]}$. Setting $\beta^m \approx \frac{1}{2}$, means $m \approx \frac{E[t]}{2}$.

6 HashTBO Format

The HashTBO format is basically the same as McIlroy's format, except that McIlroy was storing words and we are storing *n*-grams. One could store all of the n-grams in a single table, though we actually store unigrams in a separate table. An *n*-gram is represented as a key of *n* integers (offsets into the vocabulary) and two values, a log likelihood and, if appropriate, an alpha for backing off. We'll address the keys first.

6.1 HashTBO Keys

Trigrams consist of three integers (offsets into the Vocabulary): $w_1 w_2 w_3$. These three integers are mapped into a single hash between 0 and $P-1$ in the obvious way:

$$hash = (w_3 V^0 + w_2 V^1 + w_1 V^2) \bmod P$$

where V is vocabulary size. Bigrams are hashed the same way, except that the vocabulary is padded with an extra symbol for NA (not applicable). In the bigram case, w_3 is NA.

We then follow a simple recipe for bigrams and trigrams:

1. Stolcke prune appropriately
2. Let *N* be the number of n-grams
3. Choose an appropriate *P* (hash range)
4. Hash the *N* n-grams
5. Sort the hash codes
6. Take the first differences (which are modeled as interarrivals of a Poisson process)
7. Golomb code the first differences

We did not use this method for unigrams, since we assumed (perhaps incorrectly) that we will have explicit likelihoods for most of them and therefore there is little opportunity to take advantage of sparseness.

Most of the recipe can be fully automated with a turnkey process, but two steps require appropriate hand intervention to meet the memory allocation for a particular application:

1. Stolcke prune appropriately, and
2. Choose an appropriate *P*

[4] This discussion follows slide 29 of http://www.stanford.edu/class/ee398a/handouts/lectures/01-EntropyLosslessCoding.pdf. See (Witten *et al*, 1999) and http://en.wikipedia.org/wiki/Golomb_coding, for similar discussion, though with slightly different notation. The primary reference is (Golomb, 1966).