CS 112 : Data Structures

Sesh Venugopal

Huffman Tree Building Algorithm: Running Time Analysis

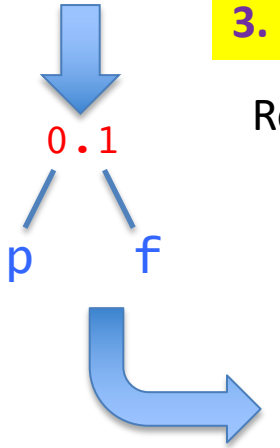
Basic Operations to Count

1. Enqueue in Symbols queue: O(1) per enqueue

Symbols queue

2. Dequeue from Symbols queue: O(1) per dequeue

3. Build a subtree out of two other subtrees: O(1) per build



Root node probability = 0.05 (p) + 0.05 (f)

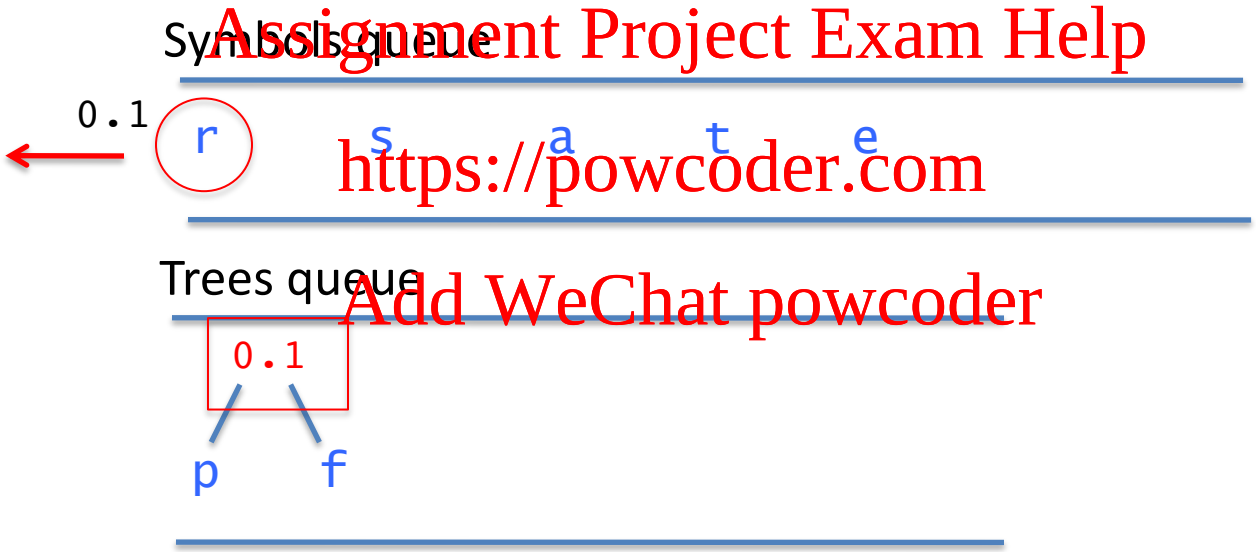
4. Enqueue in Trees queue: O(1) per enqueue

Trees queue

Basic Operations to Count

(a) Compare the probability of front of symbols queue, with that of the front of trees queue

5. Compare probabilities: O(1) per comparison

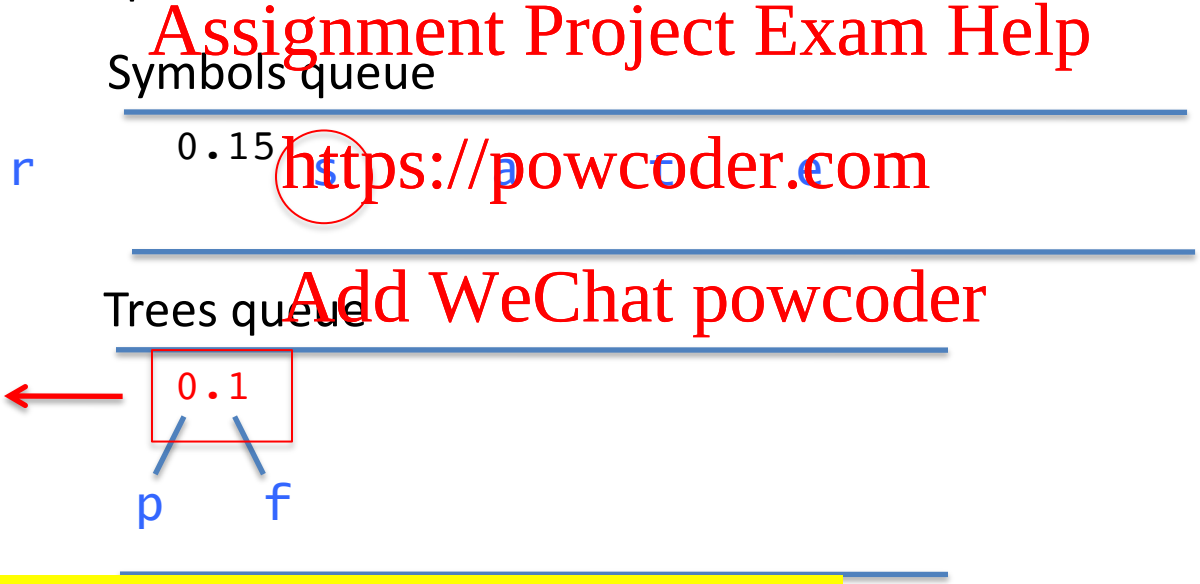


Dequeue the lesser of the two. Here, since they are both 0.1, either can be dequeued (pick arbitrarily). Say we pick `r` from the symbols queue, and dequeue it

Basic Operations to Count

p	f	r	s	a	t	e
0.05	0.05	0.1	0.15	0.2	0.2	0.25

(b) Compare the probability of front of symbols queue, with that of the front of trees queue



6. Dequeue from Trees queue: O(1) per dequeue

Dequeue the lesser of the two. Here, the trees queue front has a smaller probability, so it will be dequeued

Basic Operations to Count

1. Enqueue in Symbols queue: O(1) per enqueue
2. Dequeue from Symbols queue: O(1) per dequeue
3. Build a subtree out of two other subtrees: O(1) per build
4. Enqueue in Trees queue: O(1) per enqueue
5. Compare probabilities: O(1) per comparison
6. Dequeue from Trees queue: O(1) per dequeue

Assuming *n* symbols, we need big O worst case running time to build a Huffman tree for *n* symbols

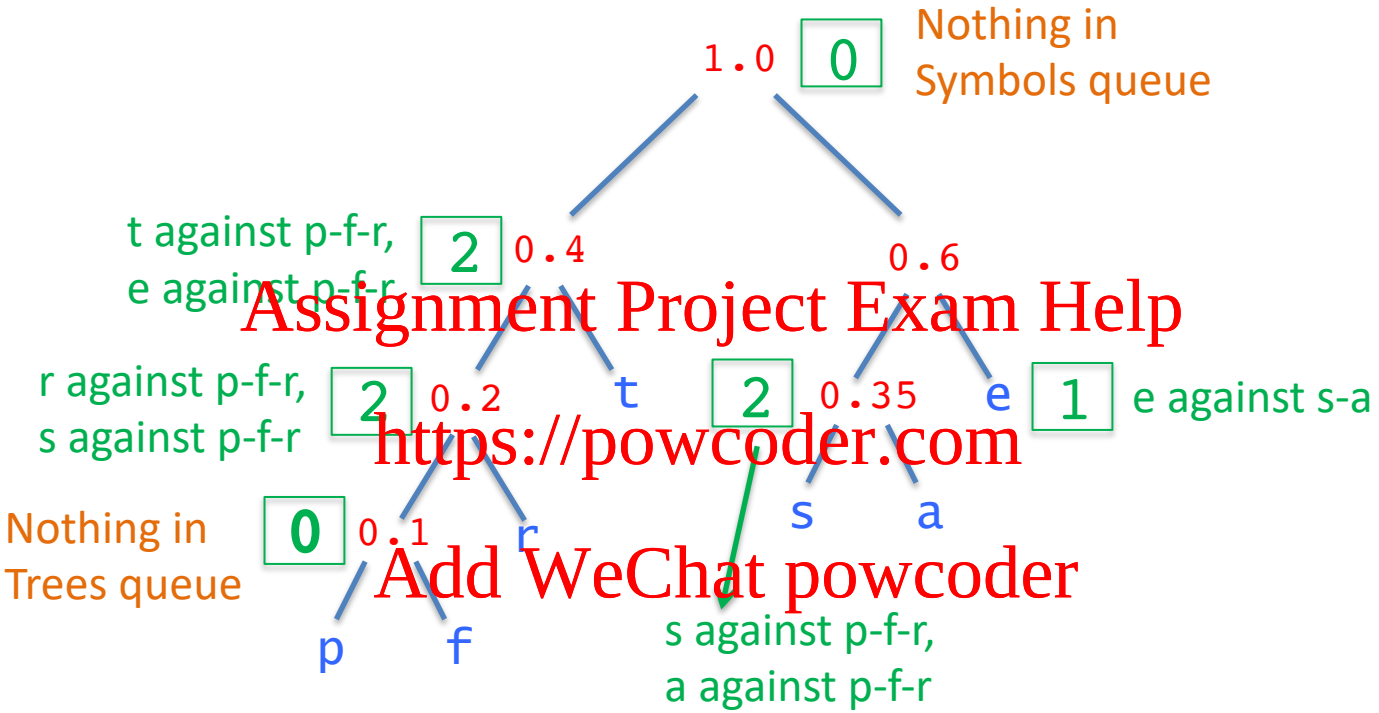
Basic Operations to Count

Suppose there are *k* subtrees

"Order Arithmetic":
O(n)*O(m) = O(m*n)

1. Enqueue in Symbols queue: O(1) per enqueue * n = O(n)
2. Dequeue from Symbols queue: O(1) per dequeue * n = O(n)
3. Build a subtree out of two other subtrees: O(1) per build * k = O(k)
4. Enqueue in Trees queue: O(1) per enqueue * k = O(k)
5. Compare probabilities: O(1) per comparison* ??
6. Dequeue from Trees queue: O(1) per dequeue * k = O(k)

How many probability comparisons?



- For any Huffman tree, the first subtree will require 0 comparisons.
- Any other subtree except final (top) will require AT MOST 2 comparisons.
- The final subtree here requires 0 comparisons, but it's possible that it requires 1 comparison in other cases (if last subtree is made out of last symbol in Symbols queue and single subtree in Trees queue)

Basic Operations to Count

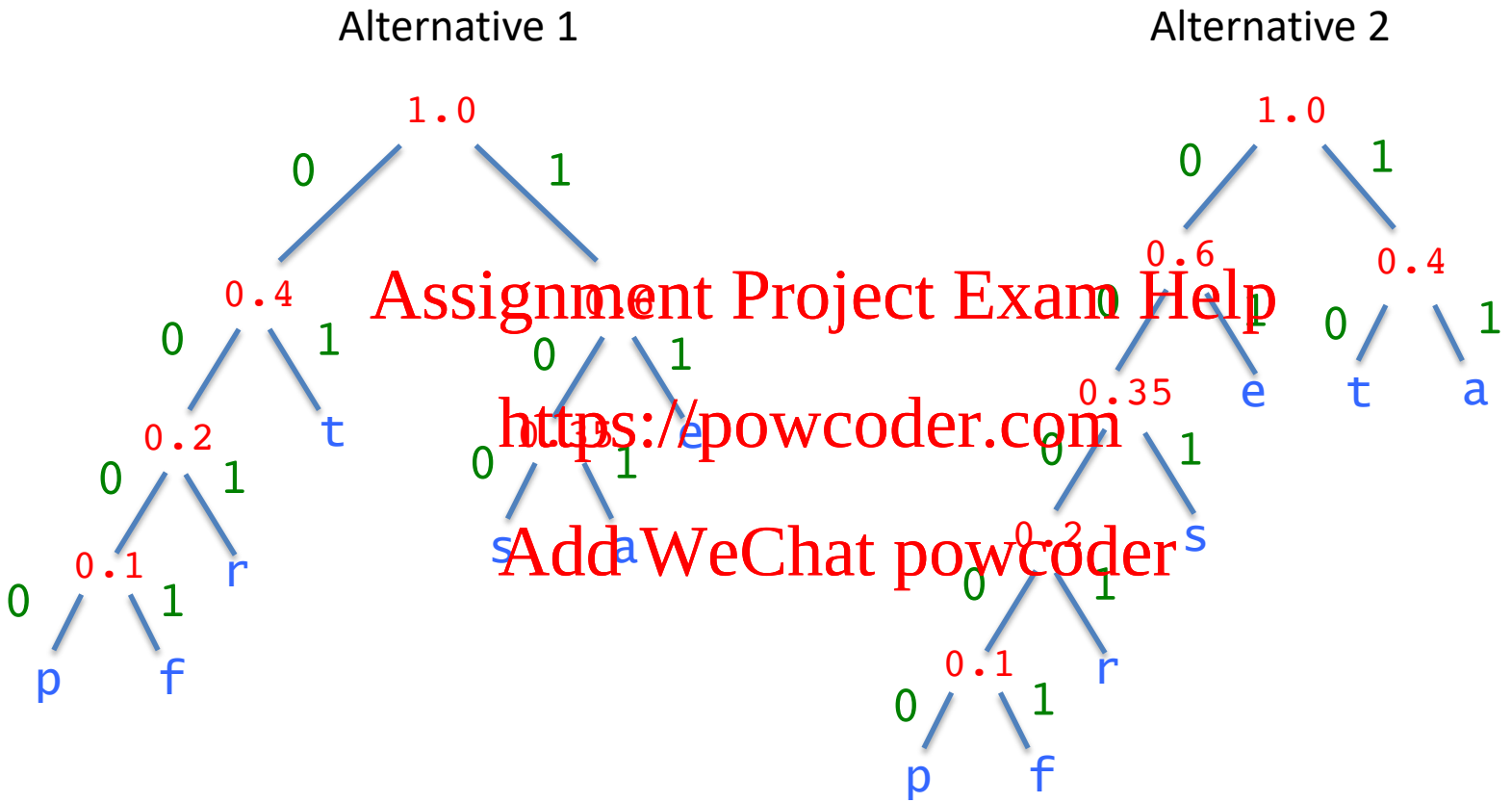
The number of probability comparisons for any subtree build for any Huffman tree is a constant (0,1,2).
And the time per probability comparison is O(1) .
So time for all probability comparisons for any subtree is O(1)

1. Enqueue in Symbols queue: O(1) per enqueue * n = O(n)

2. Dequeue from Symbols queue: O(1) per dequeue * n = O(n)

3. Build a subtree out of two other subtrees: O(1) per build * k = O(k)
4. Enqueue in Trees queue: O(1) per enqueue * k = O(k)
5. Compare probabilities: O(1) per subtree * k = O(k)
6. Dequeue from Trees queue: O(1) per dequeue * k = O(k)

What is value of k – how many subtrees?



These trees (alternatives for the same set of symbols) have very different shapes, but the number of subtrees (including final) is the same = 6!

What is value of k – how many subtrees?

Huffman tree is a special kind of binary tree in which every node has either 0 children or 2 children – this is called a strictly binary tree (in other words no node has exactly 1 child)

Theorem: If a strictly binary tree has n leaf nodes, then it has n-1 non-leaf (internal) nodes

Which means for a Huffman tree built for n symbols, so n leaf nodes, there are exactly n-1 subtrees (subtree roots are the internal nodes). In other words, k = n-1 (for our example, n=7 and k=6)

Total Running Time

Substitute n-1 for k

1. Enqueue in Symbols queue: O(1) per enqueue * n = O(n)
2. Dequeue from Symbols queue: O(1) per dequeue * n = O(n)
3. Build a subtree out of two other subtrees: O(1) per build * k = O(1) * (n-1) = O(n)
4. Enqueue in Trees queue: O(1) per enqueue * k = O(k) = O(n)
5. Compare probabilities: O(1) per comparison * k = O(k) = O(n)
6. Dequeue from Trees queue: O(1) per dequeue * k = O(k) = O(n)

Worst case running time to build a Huffman tree for n symbols is O(n)