CS 112 Spring 2020

Huffman Coding

Mar 10

Huffman Tree

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

Input Symbols, with probabilities of occurrence in text
(IN INCREASING ORDER OF PROBABILITIES)

Symbols Queue and Trees Queue - Initialize

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

Initially, enqueue all symbols – taken in increasing order of probabilities -- into symbols queue. Trees queue is initially null. (Symbols are actually wrapped inside tree nodes, which are enqueued.)

Symbols queue

p f r s a t e

Trees queue

Building Huffman Tree – Step 1

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

Dequeue the first two symbols from the symbols queue, build a subtree out of them

Symbols queue

p f r s a t e

0.1

p f

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

Trees queue

This is also OK

0.1

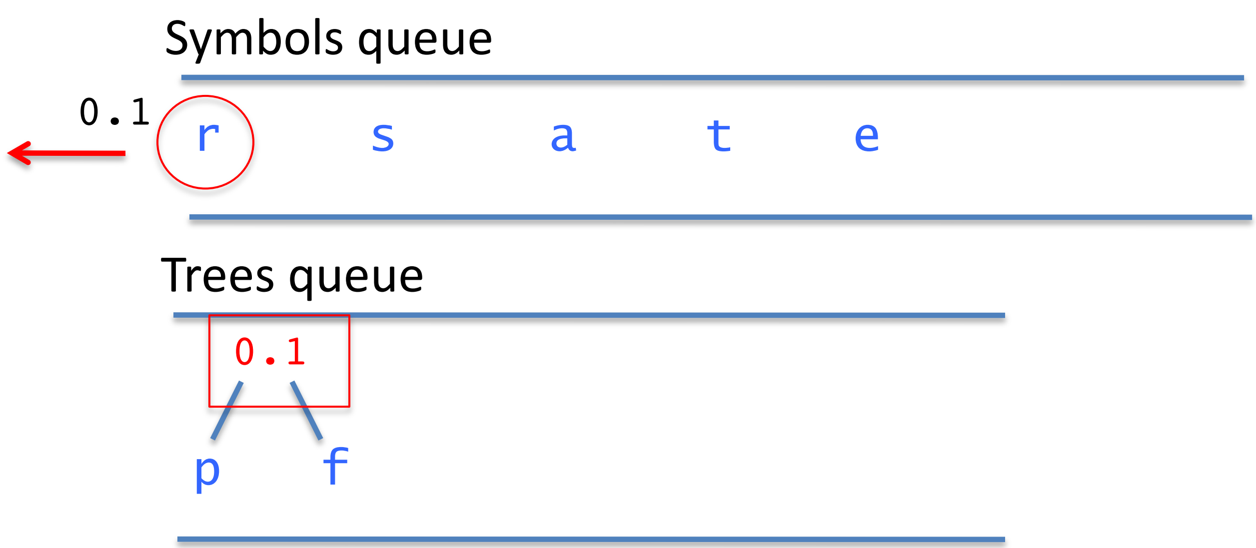
f p

Left and right subtrees are interchangeable

Building Huffman Tree – Step 2 (a)

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

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

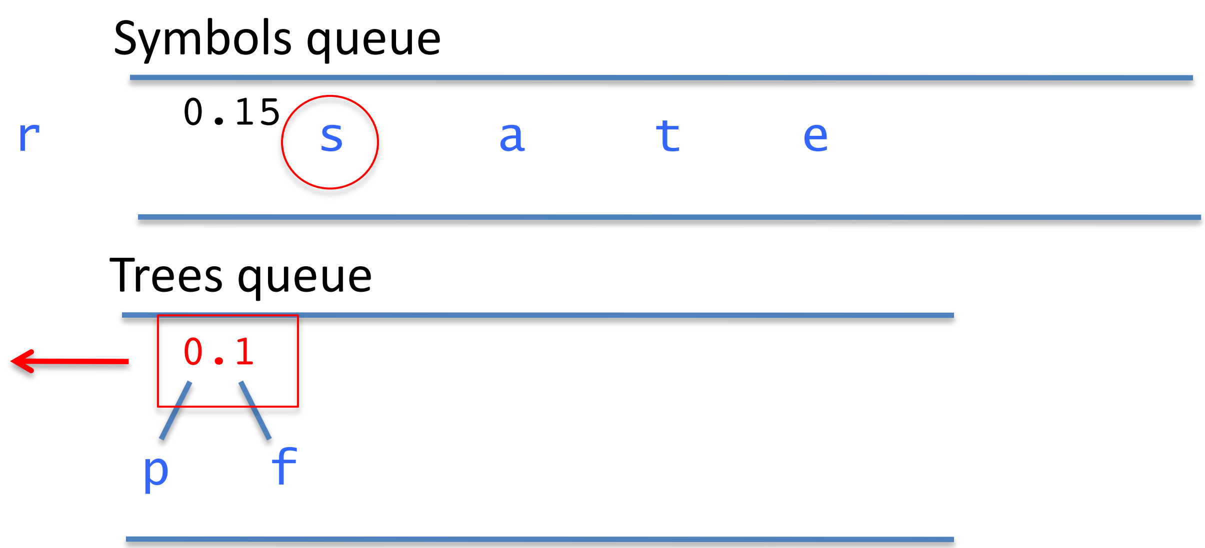


Dequeue the lesser of the two. Here, since they are both 0.1, either can be dequeued (pick arbitrarily). Say we pick `r` from the symbols queue, and dequeue it

Building Huffman Tree – Step 2 (b)

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



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

Building Huffman Tree – Step 2 (c)

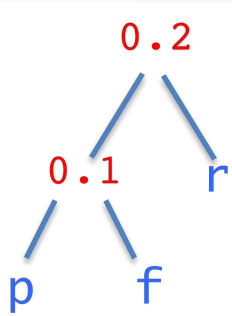
p	f	r	s	a	t	e
0.05	0.05	0.1	0.15	0.2	0.2	0.25

(c) Build a subtree out of the dequeued trees (symbols are single node trees), and enqueue into the trees queue

Symbols queue

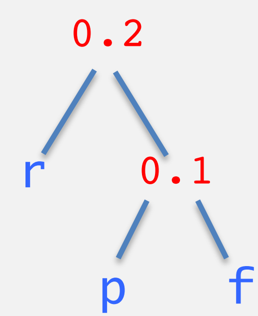
s a t e

Trees queue



Root node probability = 0.1 (r) + 0.1 (p-f subtree)

This is also OK

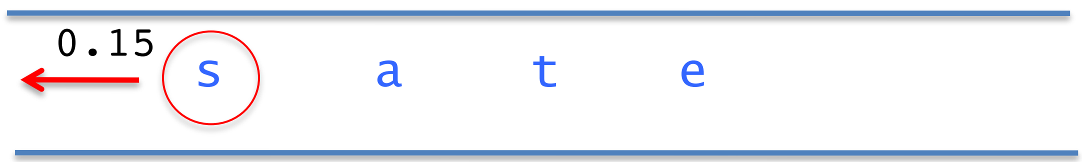


Building Huffman Tree – Step 3 (a)

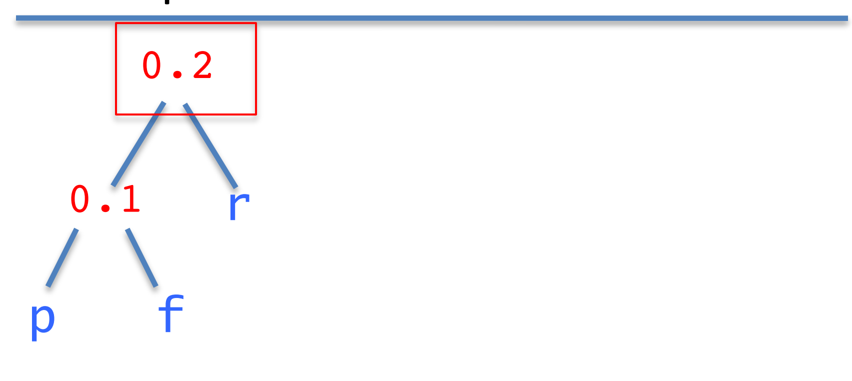
p	f	r	s	a	t	e
0.05	0.05	0.1	0.15	0.2	0.2	0.25

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

Symbols queue



Trees queue

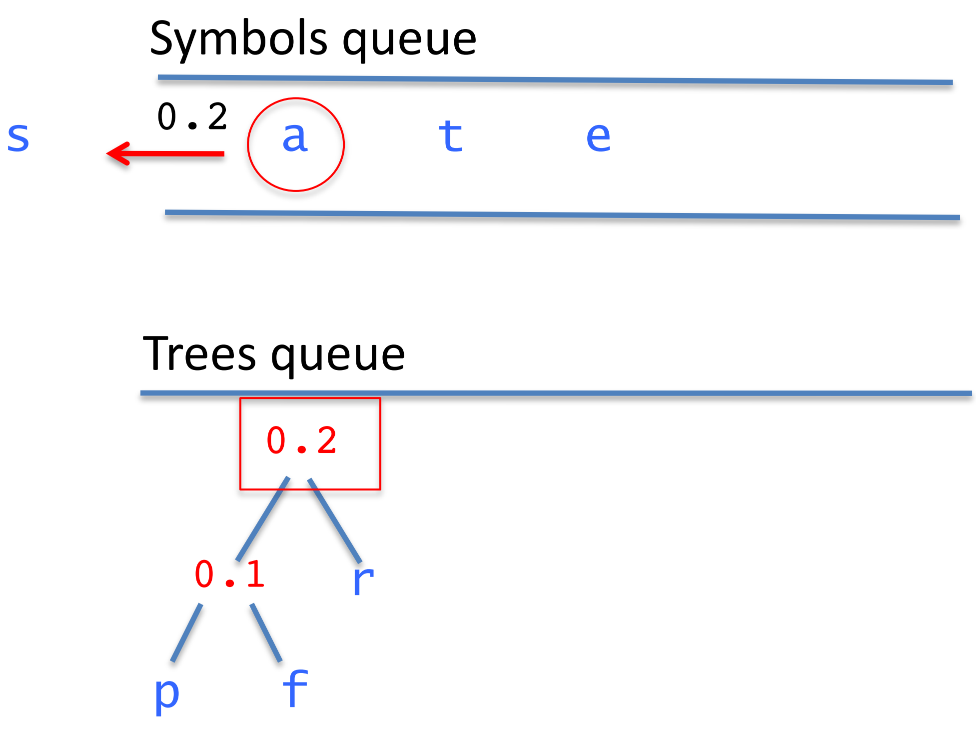


Dequeue the lesser of the two. Here s has a smaller probaility, so it is dequeued

Building Huffman Tree – Step 3 (b)

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



Dequeue the lesser of the two. Here, since they are both 0.2, either can be dequeued (pick arbitrarily). Say we pick a from the symbols queue, and dequeue it

Building Huffman Tree – Step 3 (c)

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

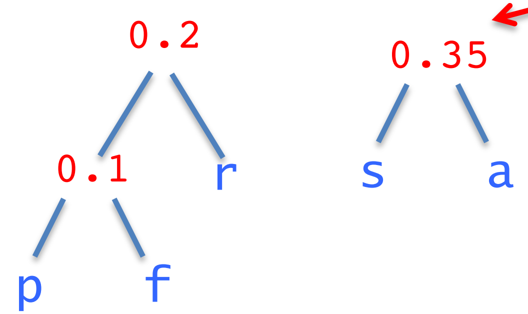
(c) Build a subtree out of the dequeued trees (symbols are single node trees), and enqueue into the trees queue

Symbols queue

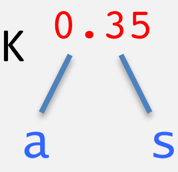
t e

Trees queue

Root node probability = 0.15 (s) + 0.2 (a)



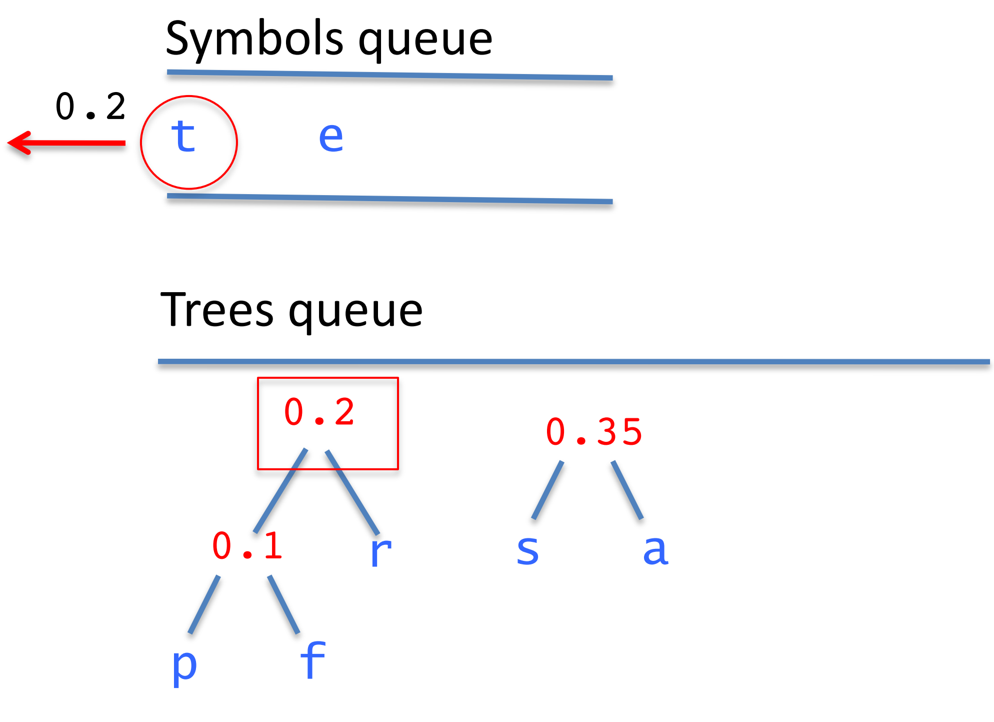
This is also OK



Building Huffman Tree – Step 4 (a)

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

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

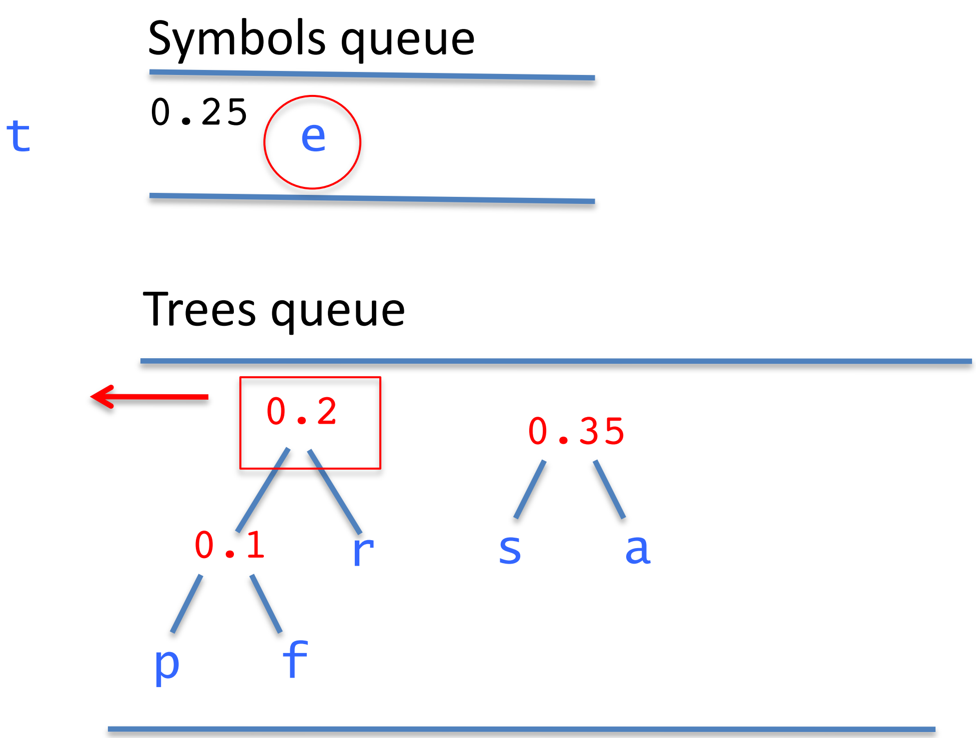


Dequeue the lesser of the two. Here, since they are both 0.2, either can be dequeued (pick arbitrarily). Say we pick t from the symbols queue, and dequeue it

Building Huffman Tree – Step 4 (b)

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



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

Building Huffman Tree – Step 4 (c)

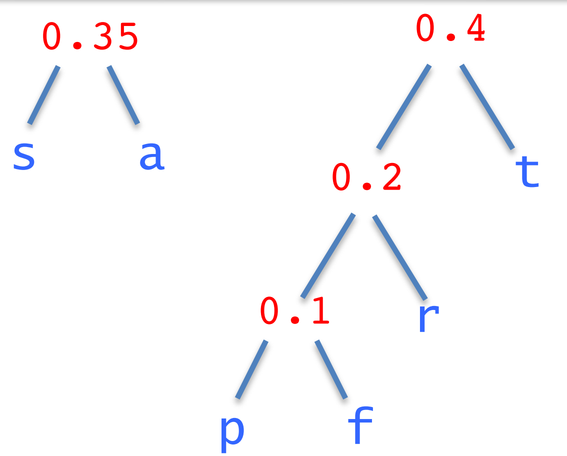
p	f	r	s	a	t	e
0.05	0.05	0.1	0.15	0.2	0.2	0.25

(c) Build a subtree out of the dequeued trees (symbols are single node trees), and enqueue into the trees queue

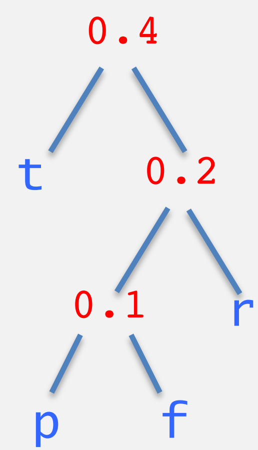
Symbols queue

e

Trees queue



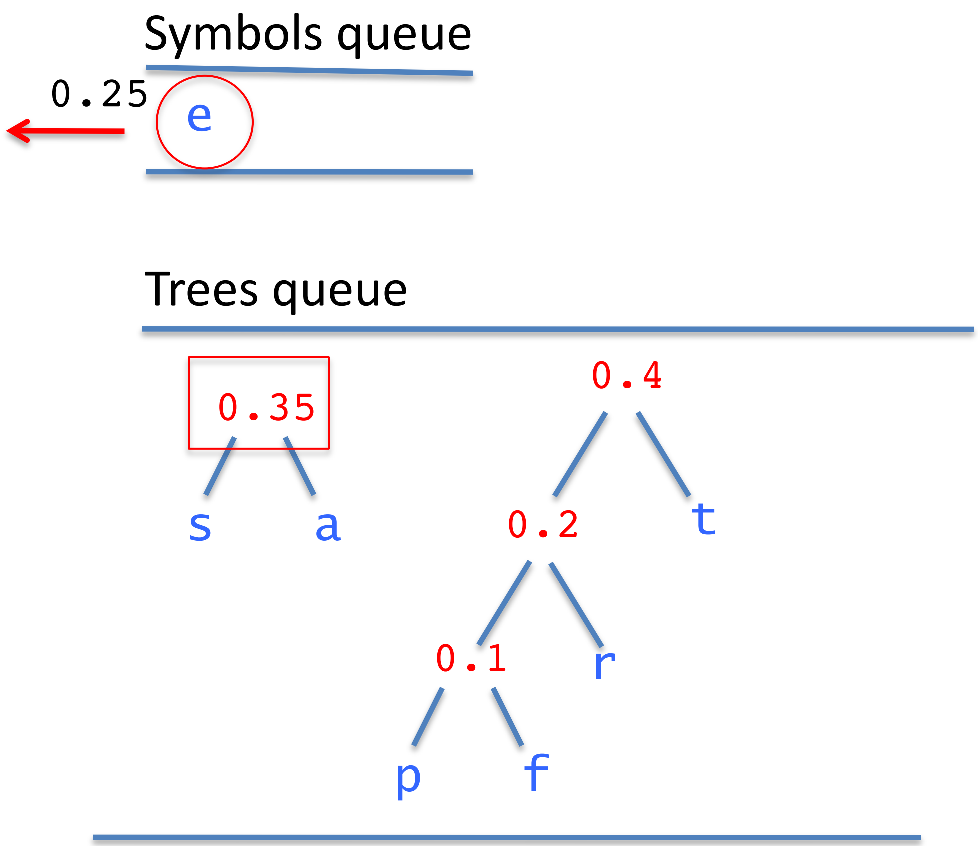
This is also OK



Building Huffman Tree – Step 5 (a)

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

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

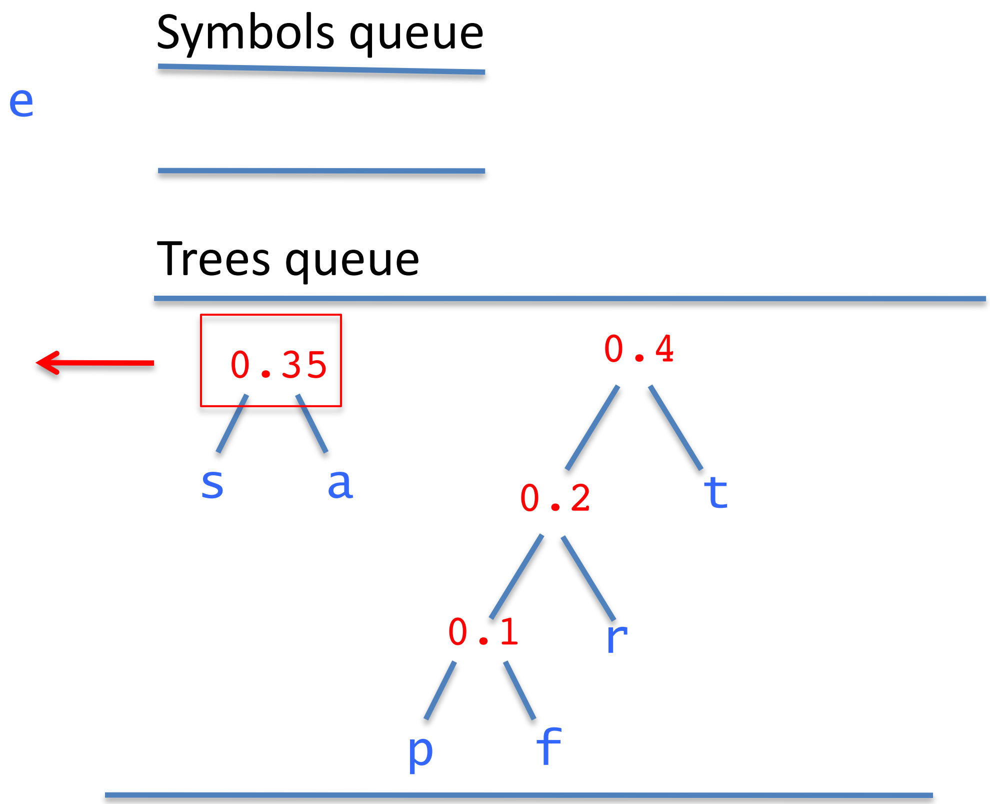


Dequeue the lesser of the two. Here e has a smaller probaility, so it is dequeued

Building Huffman Tree – Step 5 (b)

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. But the Symbols queue is empty, so the only option is to dequeue from the Trees queue



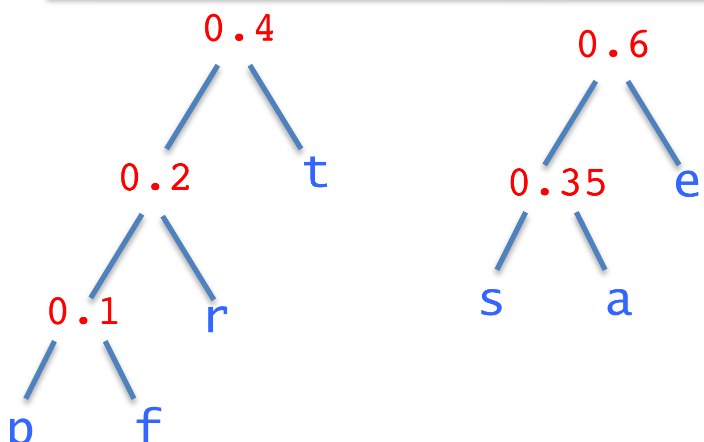
Building Huffman Tree – Step 5 (c)

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

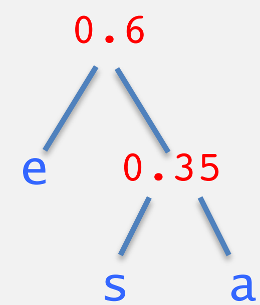
(c) Build a subtree out of the dequeued trees (symbols are single node trees), and enqueue into the trees queue

Symbols queue

Trees queue



This is also OK

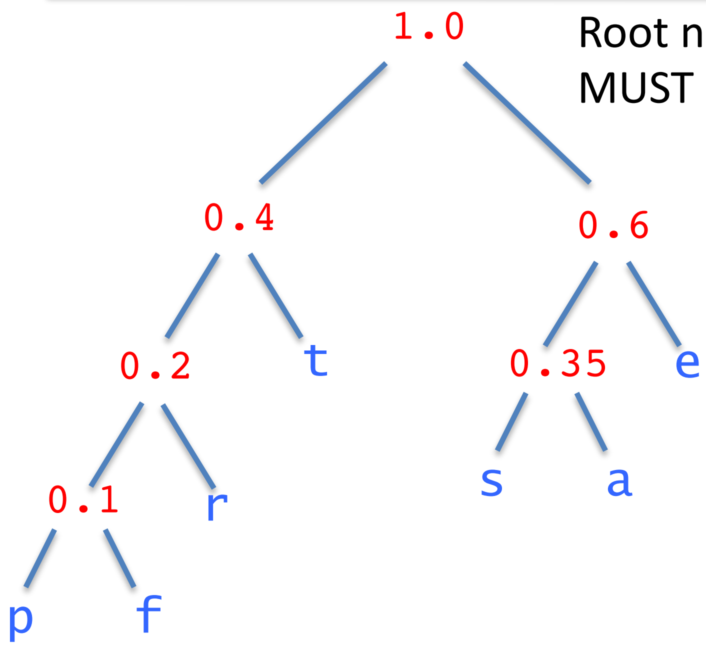


Building Huffman Tree – Step 6

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

Since Symbols queue is empty, dequeue pairs of trees from Trees queue, build subtree, and enqueue, until Trees queue has a single item

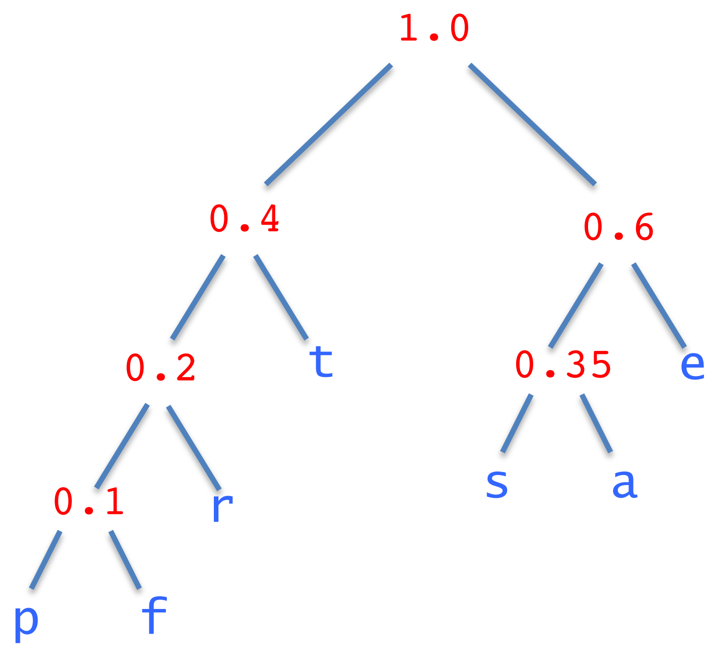
Trees queue



Root node probability of the final tree MUST be 1.0

Complete Huffman Tree

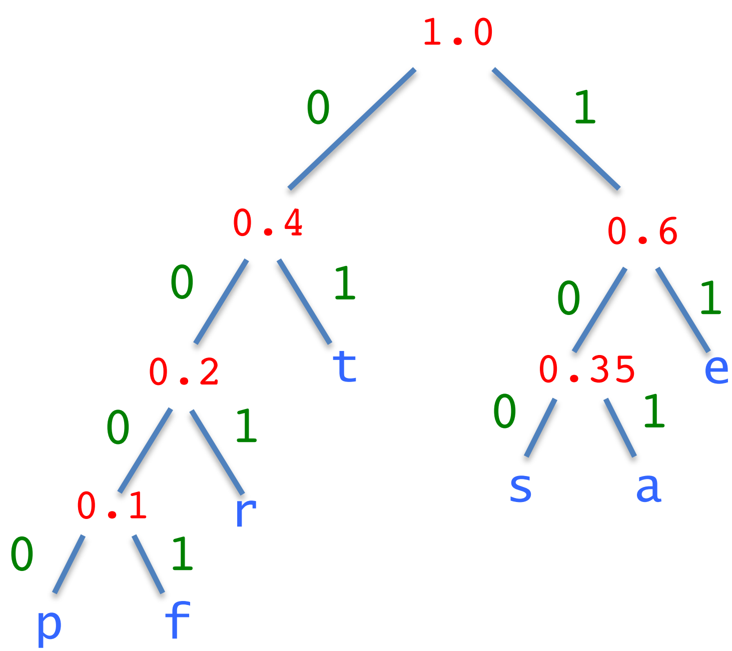
p	f	r	s	a	t	e
0.05	0.05	0.1	0.15	0.2	0.2	0.25



Assigning Bits to Branches

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

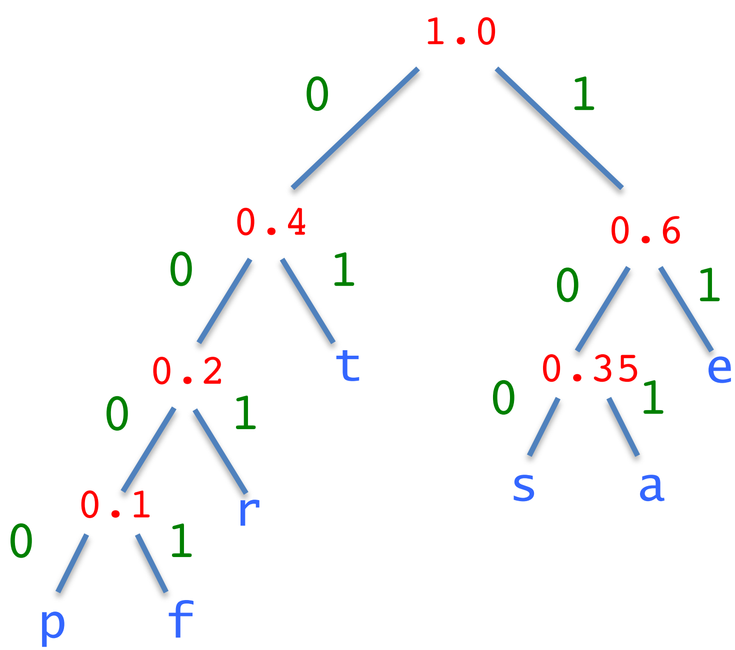
Each left branch is labeled with a 0 (bit) and right branch is labeled with a 1 (bit).



The left=0/right=1 choice is arbitrary. You could equally label left branches with 1's and right branches with 0's – the absolute code does not matter

Gathering Codes for Symbols

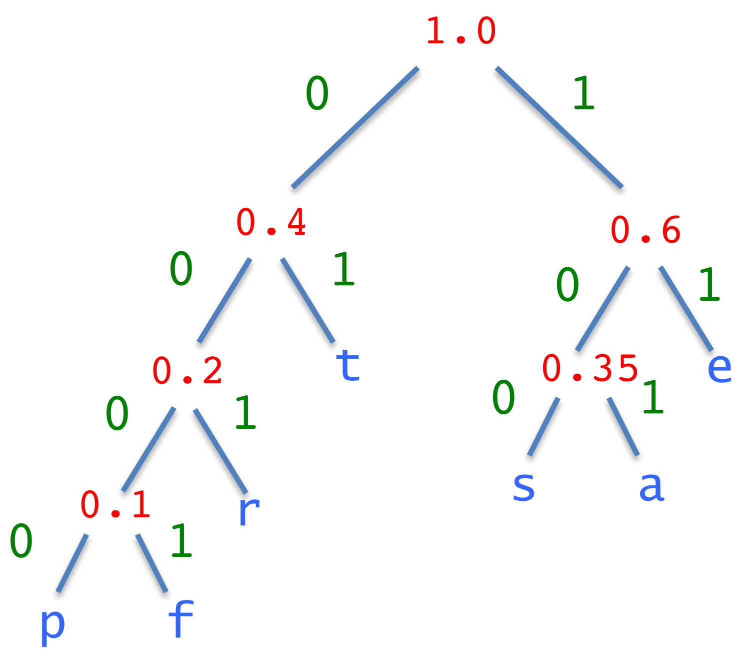
p	f	r	s	a	t	e
0.05	0.05	0.1	0.15	0.2	0.2	0.25



Code is sequence of bits along the path from from root to symbol

p	0000
f	0001
r	001
s	100
a	101
t	01
e	11

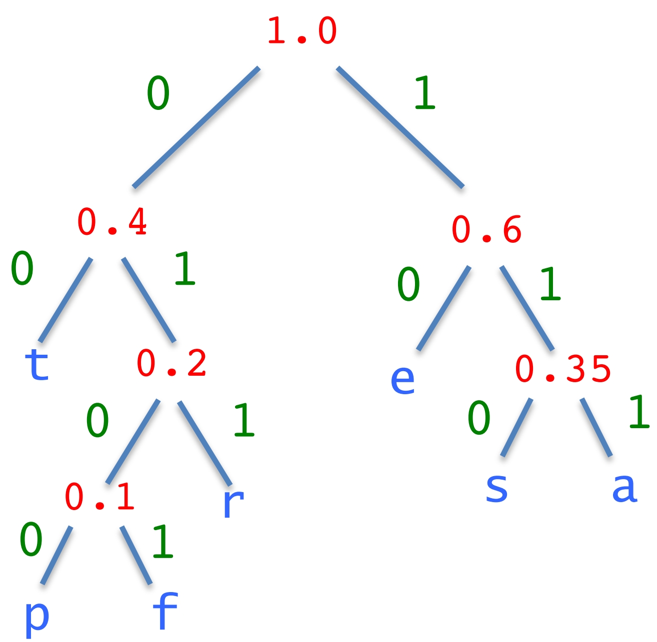
Codes for Symbols



p	0000	0.05
f	0001	0.05
r	001	0.1
s	100	0.15
a	101	0.2
t	01	0.2
e	11	0.25

Average code length
= 4*0.05 + 4*0.05 + 3*0.1 + 3*0.15 + 3*0.2 + 2*0.2 + 2*0.25
= 2.65

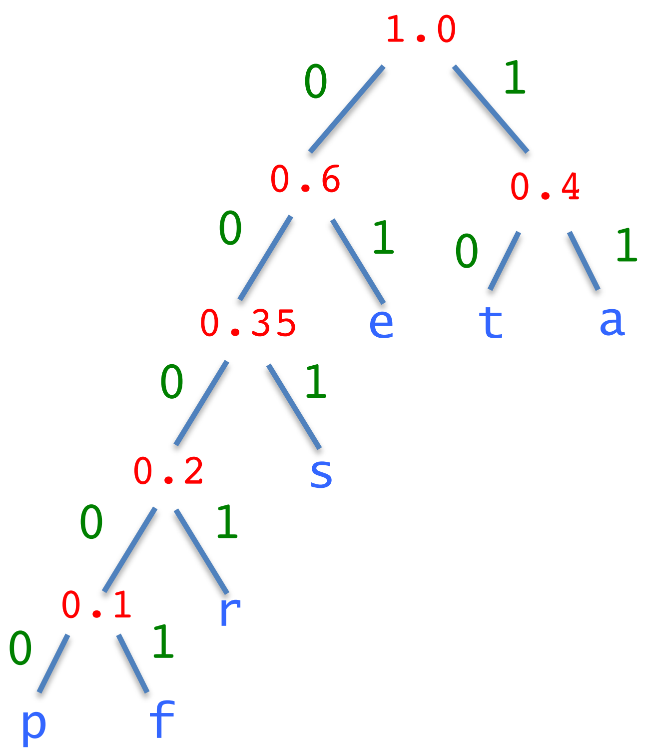
An Alternative Huffman Tree (subtrees switched)



p	0100	0.05
f	0101	0.05
r	011	0.1
s	110	0.15
a	111	0.2
t	00	0.2
e	10	0.25

Average code length
= 4*0.05 + 4*0.05 + 3*0.1 + 3*0.15 + 2*0.2 + 2*0.2 + 2*0.25
= 2.65

An Alternative Huffman Tree
(probability ties broken differently in Step 3(b))



p	00000	0.05
f	00001	0.05
r	0001	0.1
s	001	0.15
a	11	0.2
t	10	0.2
e	01	0.25

Average code length
$= 5*0.05 + 5*0.05 + 4*0.1 + 3*0.15 + 2*0.2 + 2*0.2 + 2*0.25$
$= 2.65$

Average Code Length is what matters

Various ways in which alternative Huffman trees can be obtained:

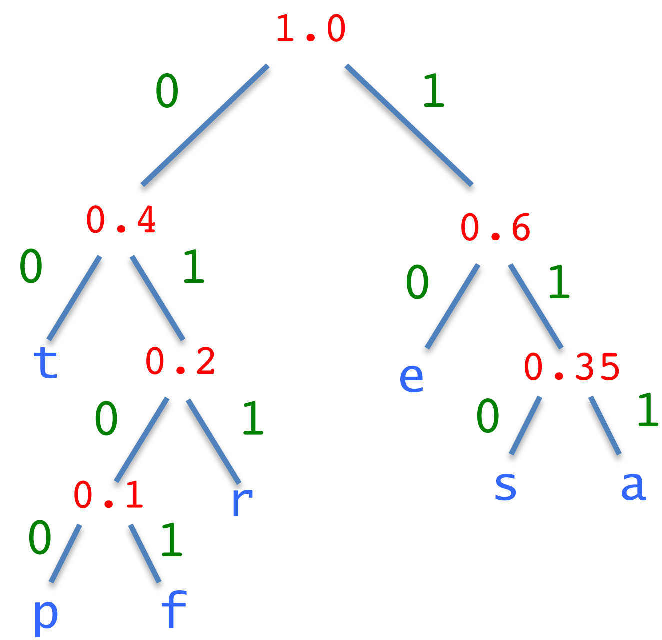
- Tie broken differently when dequeueing equal probability items from Symbols and Trees queues
- Left and right subtrees switched when building bigger tree
- Left and right branches switched when numbering with 0 or 1

Although the actual codes might differ, ALL Huffman trees for a given set of symbols+probabilities will have the SAME average code length!!!

Prefix Property

No two symbols can have codes such that one is the prefix of the other: for instance 0001 and 000 cannot both be codes since 000 is a prefix of 0001

This is true since all symbols are at the leaf nodes of the Huffman tree



It's critical that a code not be a prefix of another. WHY?

Real Example

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do. Once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, "and what is the use of a book," thought Alice, "without pictures or conversations?"

So she was considering in her own mind (as well as she could, for the day made her feel very sleepy and stupid), whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her.

There was nothing so very remarkable in that, nor did Alice think it so very much out of the way to hear the Rabbit say to itself, "Oh dear! Oh dear! I shall be too late!" But when the Rabbit actually took a watch out of its waistcoat-pocket and looked at it and then hurried on, Alice started to her feet, for it flashed across her mind that she had never before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and, burning with curiosity, she ran across the field after it and was just in time to see it pop down a large rabbit-hole, under the hedge. In another moment, down went Alice after it!

...

...

File has 6942 characters

The ASCII encoded file would be 6942 bytes long (each character is an 8-bit/1-byte code)

Characters, Probabilities, Huffman Codes

```
B|1.440507E-4|1001001010100
C|1.440507E-4|1001001010101
F|1.440507E-4|1001001010110
G|1.440507E-4|1001001010111
K|1.440507E-4|1001001011000
L|1.440507E-4|1001001011001
q|2.881014E-4|100100101101
:|4.3215213E-4|01010010110
H|4.3215213E-4|01010010111
...
...
g|0.01627773|100111
f|0.016565831|101000

|0.017862288|101001        New line character
u|0.018870642|110010
w|0.021031404|110110
d|0.03615673|10101
l|0.037453182|11000
r|0.039325844|11010
s|0.044655718|0000
i|0.047104582|0001
n|0.04897724|0100
h|0.05157015|0110
a|0.056179777|0111
o|0.061941803|1000
t|0.07346586|1011
e|0.097522326|001
 |0.17660616|111           Blank space character
```

Input text file was analyzed, and probabilities of occurrence were computed for all characters that appeared in the file

A Huffman tree was built, and codes computed

Encoded file

1001001111000000110010100111111011001110000111010111001100111000101000100000101001001111111110111000......

Alice was beginning to get very tired of sitting by her sister on the...

The encoded (compressed) file is a string of bits, total # of bits is 30887

The ASCII encoded file (uncompressed original) is 6942 bytes long =
6942*8 = 55536 bits

Ratio of compressed/uncompressed = 30887/55536 = 0.56

Compression factor = 1 - 0.56 = 0.44 = 44%!

NOTE: The encoded file must be written in binary form – you don't want each 1/0 to be written as a character!!