Lecture 4

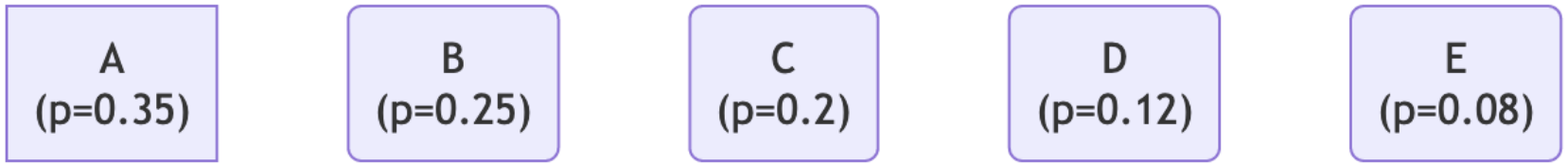
Huffman Codes + Prefix-free codes in practice

Huffman codes

- Sort the probabilities, and build singleton nodes

```
probs_dict = {"A": 0.35, "B": 0.25, "C": 0.2, "D": 0.12, "E": 0.08}
```

STEP 0: The initial step is to first build singleton nodes from these probabilities.



Huffman codes

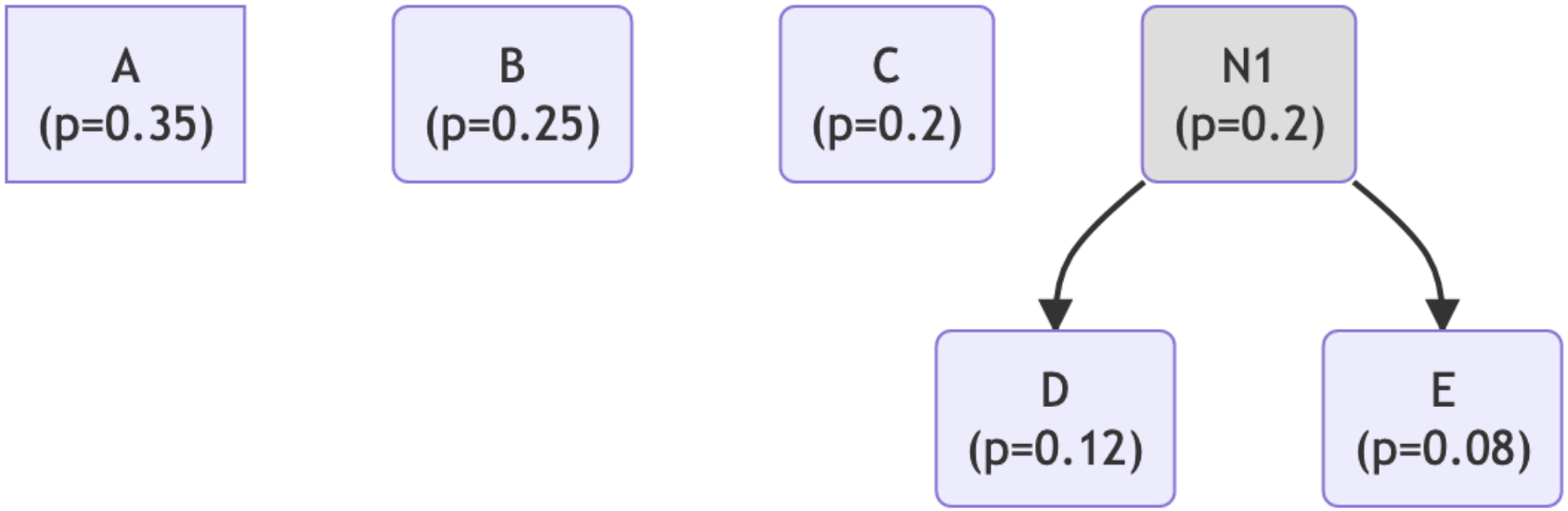
```
class Node:
    str symbol_name
    float prob
    Node left = None
    Node right = None
```

Each node has a `symbol_name` and a `prob` fields. The node also has `left` and `right` fields, which are pointers to its children as usual.

```
node_list = [Node(A, 0.35), Node(B, 0.25), Node(C, 0.2), Node(D, 0.12), Node(E, 0.08)]
```

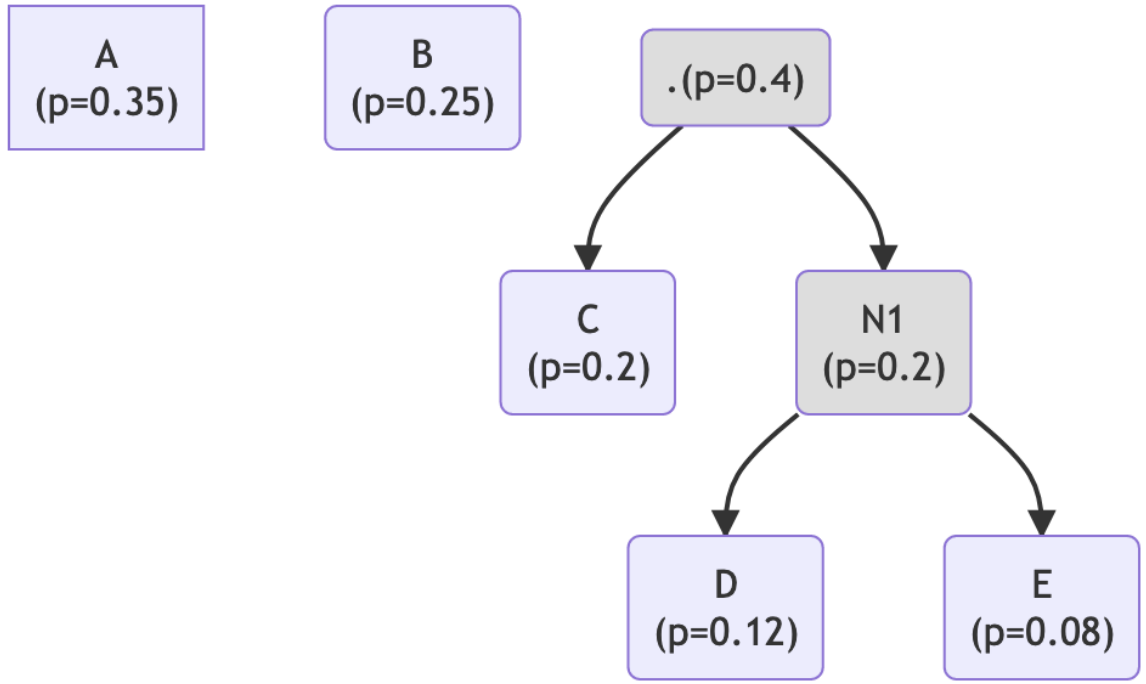
Huffman codes

```
## Iter 1
node_list = [Node(A, 0.35), Node(B, 0.25), Node(C, 0.2), Node(N1, 0.2)]
```



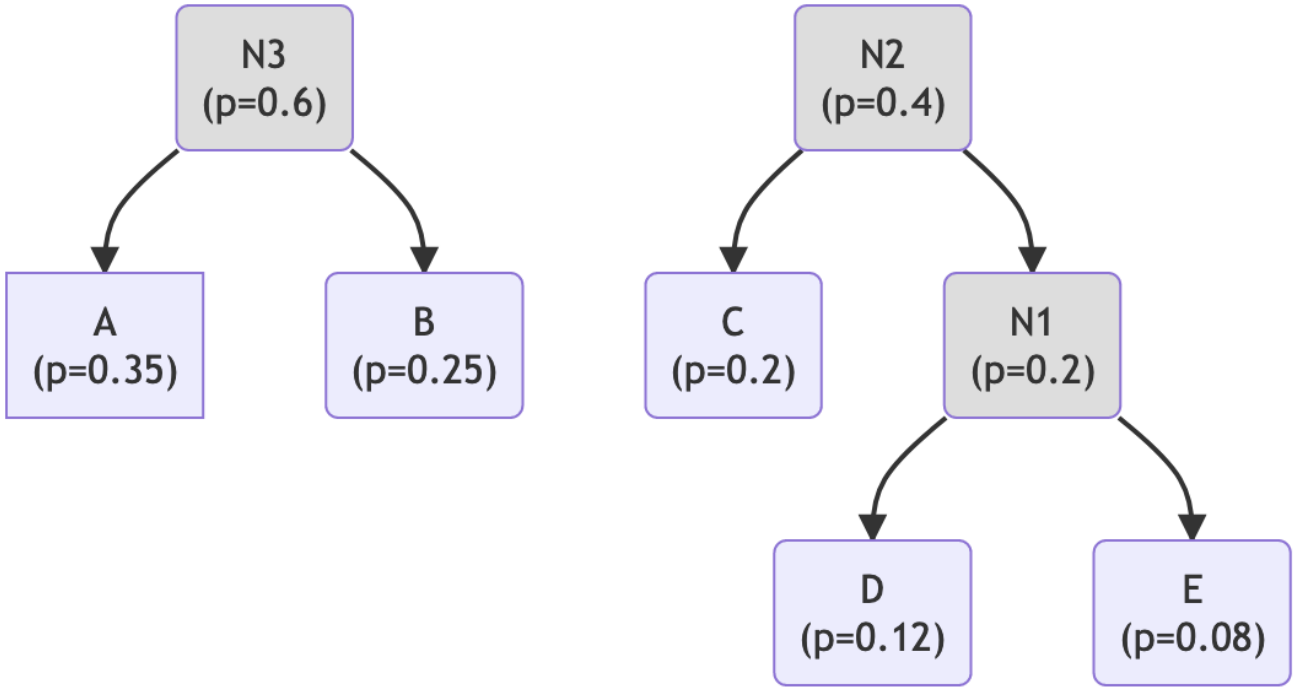
Huffman codes

```
## Iter 2
node_list = [Node(A, 0.35), Node(B, 0.25), Node(N2, 0.4)]
```

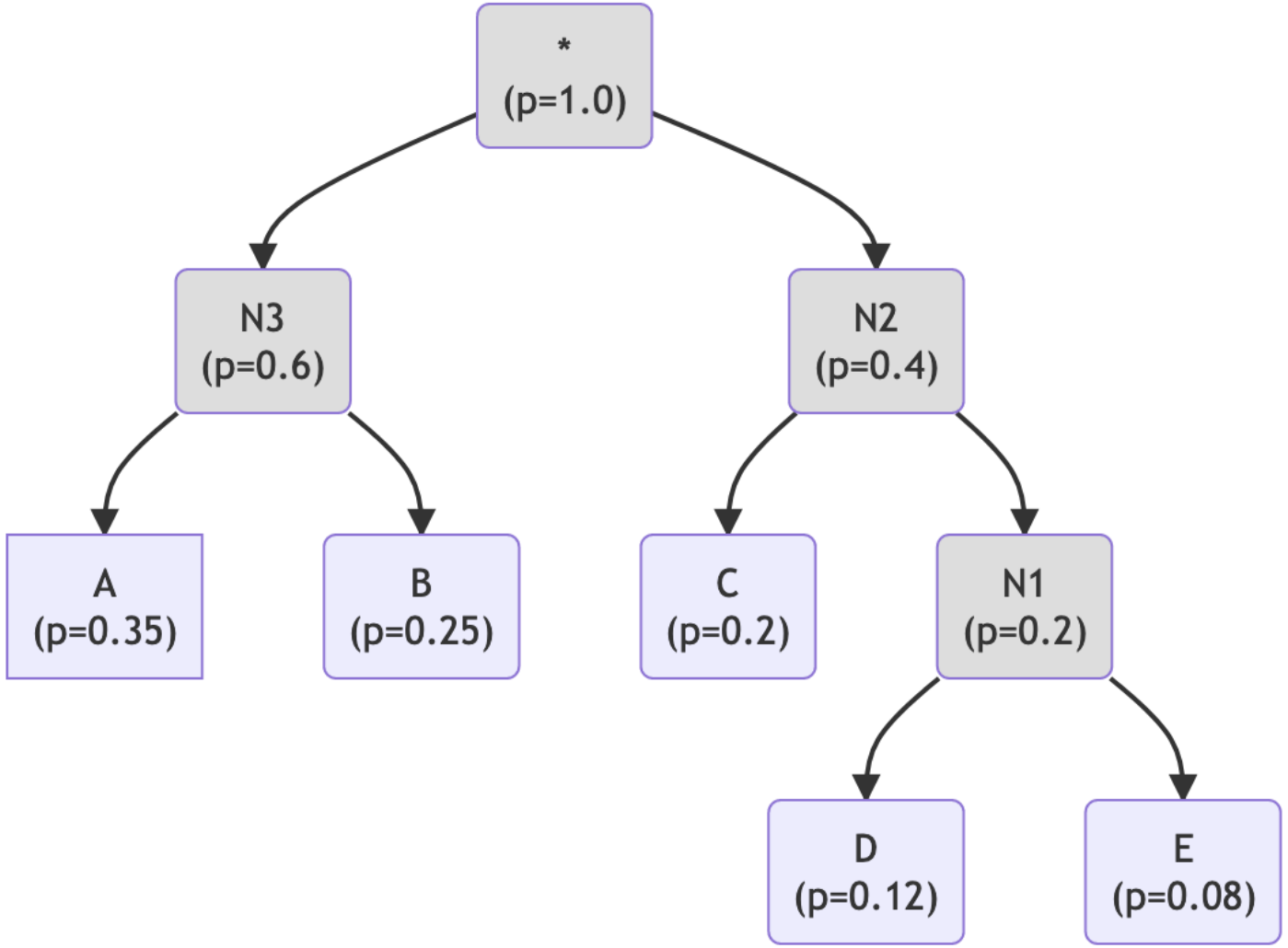


Huffman codes

```
## Iter 3
node_list = [Node(N2, 0.4), Node(N3, 0.6)]
```



```
## Iter 4
node_list = [Node("*", 1.0)]
```

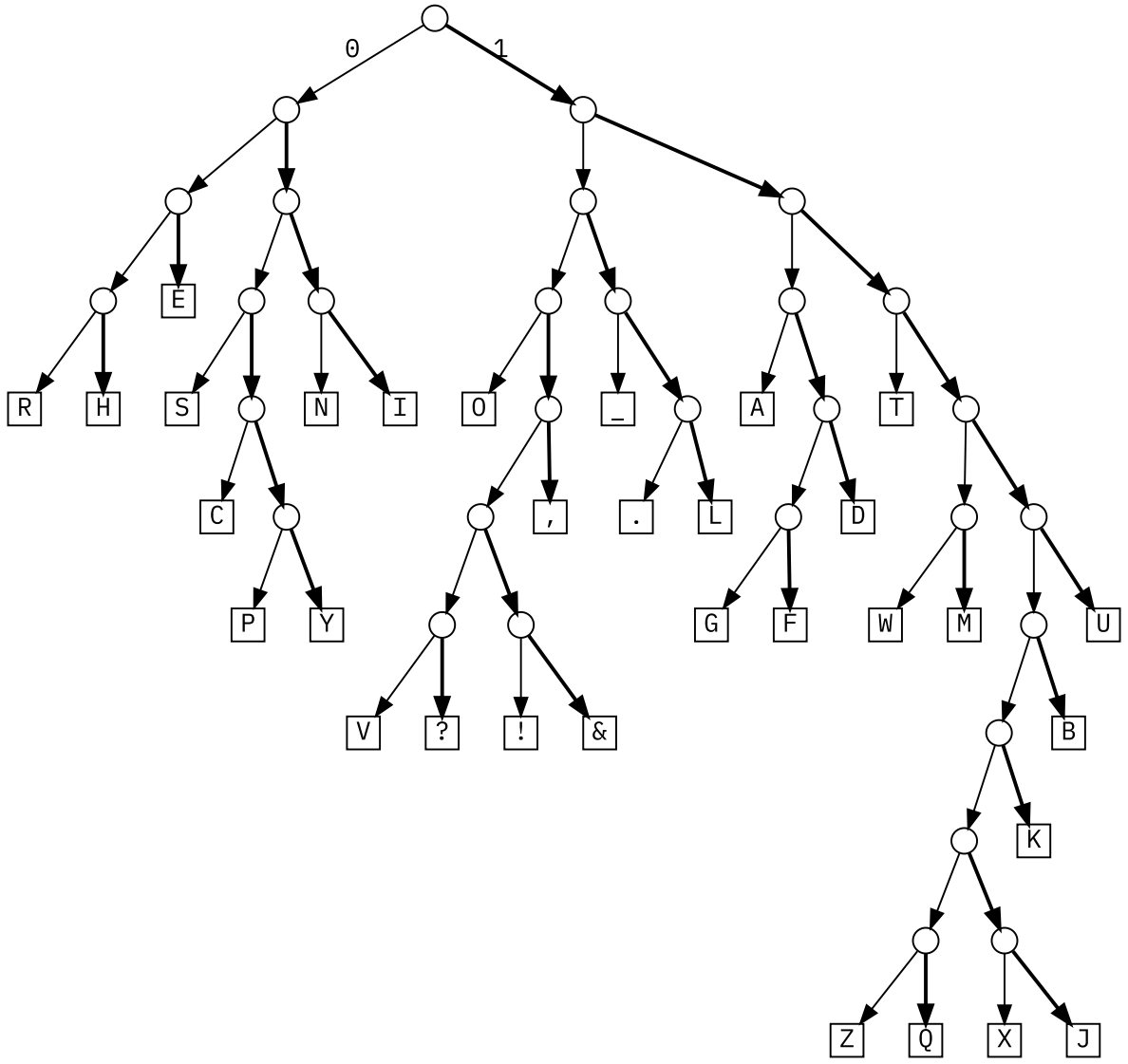


```
# Input -> given prob distribution
probs_dict = {A1: p_1, A2: p_2, ..., Ak: p_k} # p_1, .. are some floating point values

class Node:
    str symbol_name
    float prob
    Node left = None
    Node right = None


def build_huffman_tree(probs_array):
    ## STEP 0: initialize the node_list with singleton nodes
    node_list = [Node(s,prob) for s,prob in probs_array.items()]

    # NOTE: at each iter, we are reducing length of node_list by 1
    while len(node_list) > 1:
        ## STEP 1: sort node list based on probability,
        # and pop the two smallest nodes
        node_list = sort(node_list) # based on node.prob
        node_0, node_1 = node_list.pop(0, 1) # remove the smallest and second smallest e

        ## STEP 2: Merge the two nodes into a single node
        new_node_prob = node_0.prob + node_1.prob
        new_node = Node(symbol_name="", prob=new_node_prob, left=node_0, right=node_1)
        node_list.append(new_merged_node)

    # finally we will have a single node/tree.. return the node as it points to the
    # root node of our Huffman tree
    return node_list[0]
```

Huffman code construction

SCL Link:
https://github.com/kedartatwawadi/stanford_compression_library/blob/main/compressors/huffman_coder.py

Huffman codes in practice


0
1
E
R
H
S
N
I
O
_
A
T
C
,
.
L
D
P
Y
G
F
W
M
U
V
?
!
&
B
K
Z
Q
X
J


Simon Pampena
@mathemaniac
Letter Frequency Keyboard Histogram
4:16 AM - 11 Jan 2015
321 Retweets 288 Likes

Huffman codes in practice

Huffman code is typically used as the last step in quite a few compressors

- http/2 header compression: https://www.rfc-editor.org/rfc/rfc7541#appendix-B
- ZLIB, DEFLATE, GZIP, PNG compression: https://www.ietf.org/rfc/rfc1951.txt
- JPEG huffman coding tables: https://www.w3.org/Graphics/JPEG/itu-t81.pdf

K.3.1 Typical Huffman tables for the DC coefficient differences

Tables K.3 and K.4 give Huffman tables for the DC coefficient differences which have been developed from the average statistics of a large set of video images with 8-bit precision. Table K.3 is appropriate for luminance components and Table K.4 is appropriate for chrominance components. Although there are no default tables, these tables may prove to be useful for many applications.

Table K.3 – Table for luminance DC coefficient differences

Category	Code length	Code word
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

Beyond prefix-free codes?