Data vs. Drosselkom

Using huffman coding and trees to compress and decompress data

Enable native scrolling

Backstory

You are the new intern at a company that collects sensor data from remote places

Your company pays a lot of money per byte to transfer the sensor data into your datacenter

To impress your new boss you want to reduce the amount of data that has to be transfered

Huffman coding

Huffman coding is a method that uses a known probability distribution of symbols to encode messages with as few bits as possible

To generate the encodings of a huffman code a binary tree structure is used

A Binary tree is a tree where each node has exactly two children

On the next slide you will see the general algorithm to generate a huffman tree

Huffman coding

Sort the symbols by the probability of occurrence

```
─(17.4%)─► 'E'
─(9.78%)─► 'N'
─(4.76%)─► 'H'
─(4.35%)─► 'U'
```

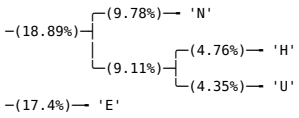

Huffman coding

Take the the two lowest ranking symbols and combine them into a tree node

```
─(17.4%)─► 'E'
─(9.78%)─► 'N'
          ┌─(4.76%)─► 'H'
─(9.11%)─┤
          └─(4.35%)─► 'U'
```

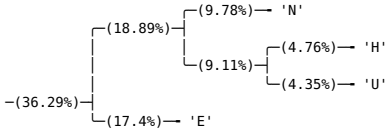
Huffman coding

Threat the node like a single symbol, resort the list and create a new node



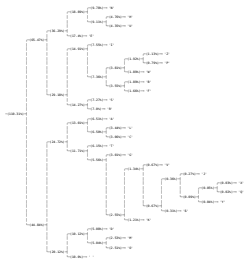
Huffman coding

Continue combining the lowest ranking nodes



Huffman coding

Until you are left with a single root node

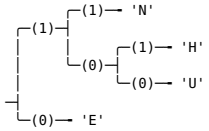


...

huffman_tree.py

Huffman coding

To get a binary encoding each side of a branch is assigned a '0' or '1'



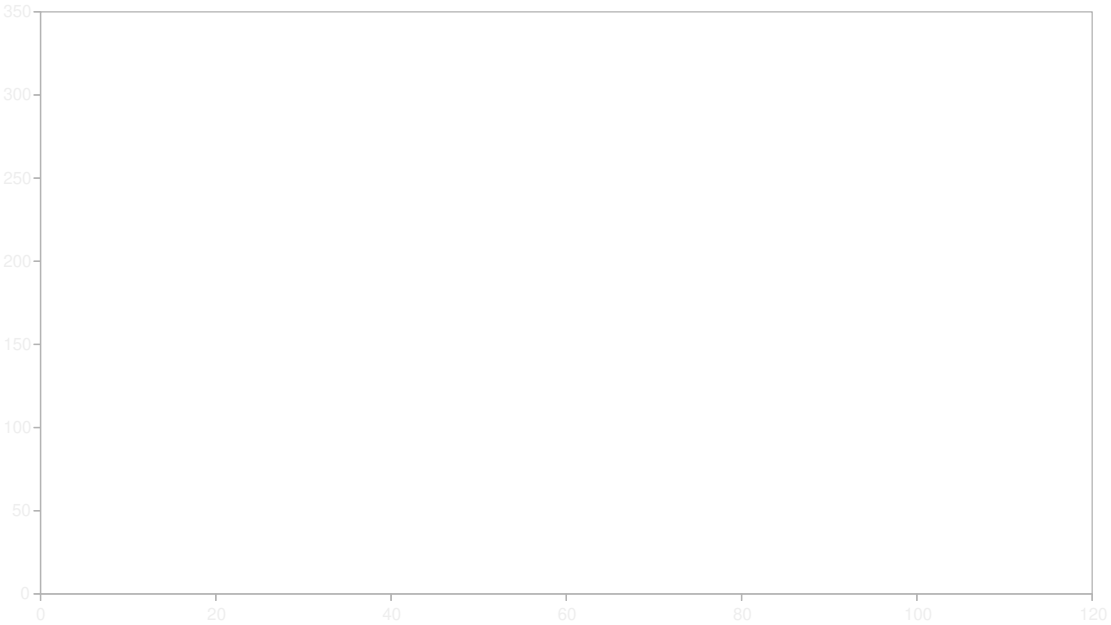
In the example above the encodings would be:

```
'E': 0, 'N': 11, 'H': 101, 'U': 100
```

Observation: No encoding is the prefix of another encoding!

Compressing sensor data

Sensor data that is sampled often enough is often mostly constant



To store the values above we would usually use a 16 bit wide variable per sample
We want to reduce that for transfer

Compressing (Idea #1)

We only transfer the difference between the current and the last value

Now most values are very close to zero

Compressing (Idea #2)

We know that values close to zero are more likely to occur

We can use huffman coding to reduce the average number of bits we have to transfer!

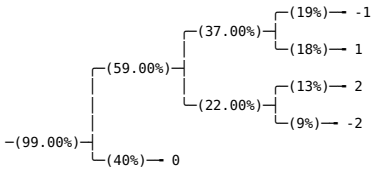
Difference compression

Assume that the difference values between two subsequent samples are known to have the following probability distribution:

Value	Probability
-2	9%
-1	19%
0	40%
1	18%
2	13%

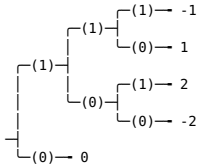
Construct a huffman tree from the table above (on paper)

Difference tree



... .py

Assigning numbers



Observation: For every measured value that does not differ from the previous value (difference zero) only a single bit has to be transfered

Using a tree in C

To encode a tree structure in C a `struct` like the following can be used

```
1 struct hm_node_t {
2   int16_t value;
3
4   struct hm_node_t *child_0;
5   struct hm_node_t *child_1;
6 };
```

Each node can either have two children (branch node) or a value (leaf node)

Using a tree in C

```
1 struct hm_node_t node_ch0 =
2   {.value= 0,
3    .child_0= NULL, .child_1= NULL};
4
5 struct hm_node_t node_root =
6   {.value= HM_NOVAL,
7    .child_0= &node_ch0, .child_1= &node_ch1};
```

The tree structure is then encoded using references to other nodes

...

Using the huffman codes

Download the example program from the previous slide
Upload it to your arduino and look at its output on the serial monitor

Using the huffman codes

Edit the `sequence` variable at the top of the program so that the following output is produced on the serial monitor:

```
Start walking
Accumulator: 0
Accumulator: 2
Accumulator: 4
Accumulator: 3
Accumulator: 2
```

Hint: The program uses the encoding developed on the previous slides