CSE2310 Algorithm Design

Lecture/Q&A 4: Greedy Algorithms

Stefan Hugtenburg, Emir Demirović, and Mathijs de Weerdt

Algorithmics group — EEMCS — TU Delft

2023–2024 Q2

You are here

The course so far

- Introduction
- Greedy algorithms and proofs: scheduling, MSTs, clustering

Today's content

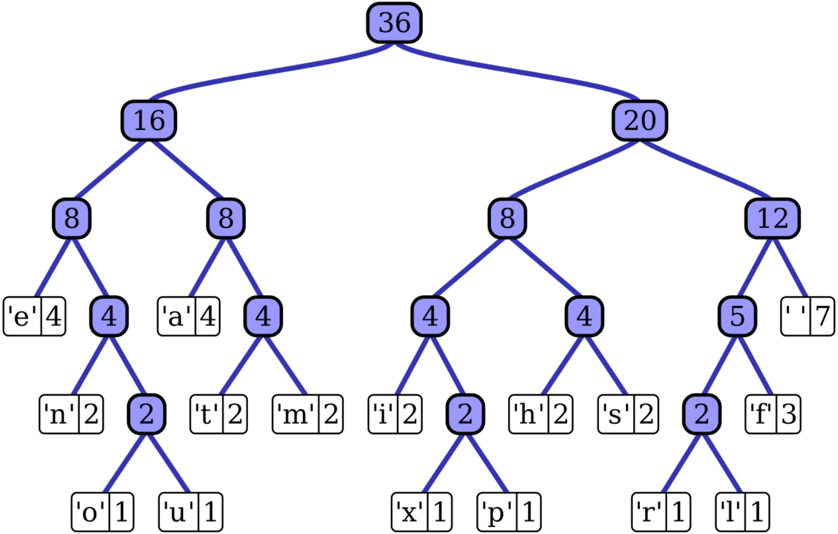
- Huffman's Optimal Encoding
- Some exam-level assignments
- Q&A

The future

- Divide & Conquer algorithms
- Dynamic programming
- Network Flow

Huffman codes

Image from Wikipedia



Encoding text

Efficiency in both runtime and output space!?

"example" $\longleftrightarrow$ "01100101 0111100 00110000 10110110 10111000 00110110 001100101"

Problem: Efficient encoding

Given a text, encode the text in binary as efficiently as possible, so that the encoding is non-ambiguous.

Encoding text

Efficiency in both runtime and output space!?

"example" $\longleftrightarrow$ "01100101 0111100 00110000 10110110 10111000 00110110 001100101"

Problem: Efficient encoding

Given a text, encode the text in binary as efficiently as possible, so that the encoding is non-ambiguous.

Answer: The best in it's own subclass

A Huffman encoding is the optimal encoding when encoding each symbol separately! It is a *prefix* coding.

Prefix codes?

Do not repeat your starts!

Definition (Prefix code)

A prefix code for a set S is a function $c : S \to \{0,1\}^+$ so that
$\forall x, y \in S : x \neq y \to c(x)$ is not the same as a prefix (first part) of $c(y)$.

Note that $\{0,1\}^+$ means any string of length ≥ 1 consisting of only zeroes and ones.

Prefix codes?

Do not repeat your starts!

Definition (Prefix code)

A prefix code for a set S is a function $c : S \to \{0,1\}^+$ so that $\forall x, y \in S : x \neq y \to c(x)$ is not the same as a prefix (first part) of $c(y)$.

Note that $\{0,1\}^+$ means any string of length ≥ 1 consisting of only zeroes and ones.

Question: Does this work?

Take $S = \{a, b, d\}$, and $c(a) = 01$, $c(b) = 010$, $c(d) = 1$. Is this a prefix code?

A. Yes
B. No
C. I don't know

Prefix codes?

Do not repeat your starts!

Definition (Prefix code)

A prefix code for a set S is a function $c : S \to \{0,1\}^+$ so that $\forall x, y \in S : x \neq y \to c(x)$ is not the same as a prefix (first part) of $c(y)$.

Note that $\{0,1\}^+$ means any string of length ≥ 1 consisting of only zeroes and ones.

Question: Does this work?

Take $S = \{a, b, d\}$, and $c(a) = 01$, $c(b) = 010$, $c(d) = 1$. Is this a prefix code?

A. Yes

B. No

C. I don't know

Answer: Nah!

Nope! $c(a)$ is a prefix of $c(b)$

Prefix codes?

Do not repeat your starts!

Definition (Prefix code)

A prefix code for a set S is a function $c : S \to \{0,1\}^+$ so that
$\forall x, y \in S : x \neq y \to c(x)$ is not the same as a prefix (first part) of $c(y)$.

Note that $\{0,1\}^+$ means any string of length ≥ 1 consisting of only zeroes and ones.

Problem: More complexity!

What does 1001000001 mean, given that
$c(a) = 000, c(e) = 01, c(k) = 11, c(n) = 10, c(t) = 001$?

Prefix codes?

Do not repeat your starts!

Definition (Prefix code)

A prefix code for a set S is a function $c : S \to \{0,1\}^+$ so that
$\forall x, y \in S : x \neq y \to c(x)$ is not the same as a prefix (first part) of $c(y)$.

Note that $\{0,1\}^+$ means any string of length ≥ 1 consisting of only zeroes and ones.

Problem: More complexity!

What does 1001000001 mean, given that
$c(a) = 000, c(e) = 01, c(k) = 11, c(n) = 10, c(t) = 001$?

Answer: Is that a good translation of 'leuk'?

neat

TU Delft

Towards optimal prefix codes!

Problem: How do we define optimal?

How do we measure a “good encoding”?

Towards optimal prefix codes!

Problem: How do we define optimal?

How do we measure a “good encoding”?

Answer: Average it out!

By looking at the average encoding length of text we want to encode!

Towards optimal prefix codes!

Problem: How do we define optimal?

How do we measure a “good encoding”?

Answer: Average it out!

By looking at the average encoding length of text we want to encode!

Problem: Average of what?

But what is the “average text”?

Towards optimal prefix codes!

Problem: How do we define optimal?

How do we measure a "good encoding"?

Answer: Average it out!

By looking at the average encoding length of text we want to encode!

Problem: Average of what?

But what is the "average text"?

Answer: Frequency analysis to the rescue!

We do a frequency analysis! So we formulate our question as: Given some letters S and the frequency of their use as a function f that sums to 1, what is the encoding function c that minimises the Average Bits per Letter: $ABL(c) = \sum_{x \in S} f(x) \cdot |c(x)|$?

Making it visual!

A binary tree as an encoding!

A binary tree represents a code where:

- Children are uniquely identified by an edge label (0 or 1)
- Nodes are labeled with symbol x iff the path from the root is labeled with the encoding $c(x)$.

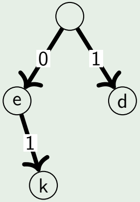
Making it visual!

A binary tree as an encoding!

A binary tree represents a code where:

- Children are uniquely identified by an edge label (0 or 1)
- Nodes are labeled with symbol x iff the path from the root is labeled with the encoding $c(x)$.

As an example...



$c(d) = 1, c(e) = 0, c(k) = 01$

$|c(x)|$ is now the depth of the node in the tree!

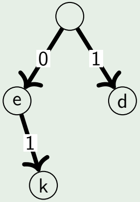
Making it visual!

A binary tree as an encoding!

A binary tree represents a code where:

- Children are uniquely identified by an edge label (0 or 1)
- Nodes are labeled with symbol x iff the path from the root is labeled with the encoding $c(x)$.

As an example...



$c(d) = 1, c(e) = 0, c(k) = 01$
$|c(x)|$ is now the depth of the node in the tree!

How do we see from the tree that this is not a prefix code?

Only leaves!

Or is it leafs, I always forget

An important observation

Only leaves can have a label in a prefix code!

Only leaves!

Or is it leafs, I always forget

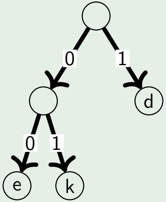
An important observation

Only leaves can have a label in a prefix code!

Proof.

If an internal node x has a label, its path is a prefix of another one, and...
The path of x is a prefix of the path of y *iff* its encoding is prefix of encoding of y. □

As an example...



elft

Only leaves!

Or is it leafs, I always forget

An important observation

Only leaves can have a label in a prefix code!

Question: Get out your pencils!

Draw the tree for the prefix encoding we had before:
$c(a) = 000, c(e) = 01, c(k) = 11, c(n) = 10, c(t) = 001$

Only leaves!

Or is it leafs, I always forget

An important observation

Only leaves can have a label in a prefix code!

Question: Get out your pencils!

Draw the tree for the prefix encoding we had before:
$c(a) = 000, c(e) = 01, c(k) = 11, c(n) = 10, c(t) = 001$

Question: Get out your pencils!

You get a 0010110 for this!

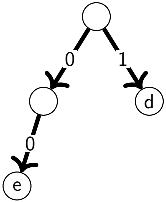
Full binary trees

Trust us, this will all come together :)

Definition (Full binary trees)

A binary tree is full if every node has either 2 or 0 children.

Claim: The binary tree corresponding to the optimal prefix code is full.



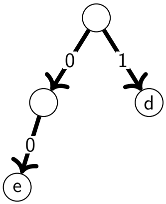
Full binary trees

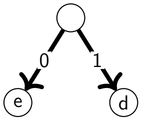
Trust us, this will all come together :)

Definition (Full binary trees)

A binary tree is full if every node has either 2 or 0 children.

Claim: The binary tree corresponding to the optimal prefix code is full.





Full binary trees

Trust us, this will all come together :)

Claim: The binary tree corresponding to the optimal prefix code is full.

Proof by contradiction.

- Suppose for the sake of contradiction that T is a *non-full* binary tree of an optimal prefix code.
- There must then be a node u with one child v. u does not have a label (no leaf).
- Now there are two options (division into cases!):
  - u is the root. Now create T' where we delete u and use v as the root.
  - u is not the root. Create T' where we delete u and let v be the child of w where w is the parent of u.
- In both cases the number of bits needed to encode any leaf in the subtree of v is decreased and the rest of the tree remains the same.
- Thus the ABL of T' is smaller than T, which contradicts our assumption that T is optimal.

elft

Okay, so it's full, now what?

Based on Shannon-Fano, 1949

Question: A greedy strategy

Where do the more common letters (highest frequencies) go?

Okay, so it's full, now what?

Based on Shannon-Fano, 1949

Question: A greedy strategy

Where do the more common letters (highest frequencies) go?

Answer: Like on a mountain

At the top!

Idea: Create the tree top-down. Split S into sets S_1 and S_2 with (almost) equal frequencies, then recursively build the tree for S_1 and S_2.

Okay, so it's full, now what?

Based on Shannon-Fano, 1949

Question: A greedy strategy

Where do the more common letters (highest frequencies) go?

Answer: Like on a mountain

At the top!

Idea: Create the tree top-down. Split S into sets S_1 and S_2 with (almost) equal frequencies, then recursively build the tree for S_1 and S_2.

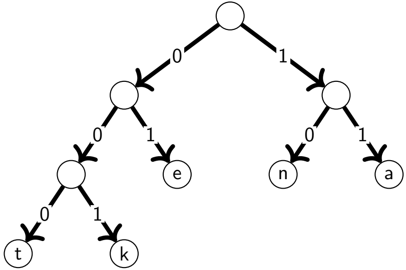
Question: Does it work?

Try it for $f_a = 0.32$, $f_e = 0.25$, $f_k = 0.2$, $f_n = 0.18$, $f_t = 0.05$. Does it work?

- A. Yes!
- B. No!
- C. Wait whut?

No dice, I'm afraid

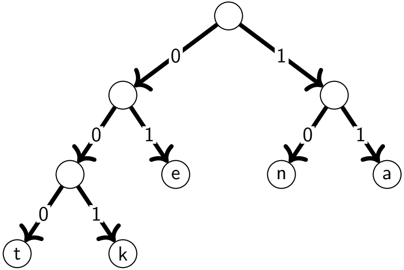
$f_a = 0.32,\ f_e = 0.25,\ f_k = 0.2,\ f_n = 0.18,\ f_t = 0.05.$



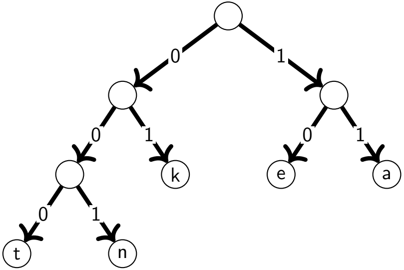
This is not optimal!
$ABL(t) = 0.05 \cdot 3 + 0.2 \cdot 3 + 0.25 \cdot 2 +$
$0.18 \cdot 2 + 0.32 \cdot 2 = 2.25$

No dice, I'm afraid

$f_a = 0.32,\ f_e = 0.25,\ f_k = 0.2,\ f_n = 0.18,\ f_t = 0.05.$



This is not optimal!
$ABL(t) = 0.05 \cdot 3 + 0.2 \cdot 3 + 0.25 \cdot 2 + 0.18 \cdot 2 + 0.32 \cdot 2 = 2.25$



This is better!
$ABL(t) = 0.05 \cdot 3 + 0.18 \cdot 3 + 0.20 \cdot 2 + 0.25 \cdot 2 + 0.32 \cdot 2 = 2.23$

Let's fix it!

Based on Huffman, 1952

Lemma

If u and v are leaves in T^* and $depth_{T^*}(u) < depth_{T^*}(v)$ then $f_u \geq f_v$.

Let's fix it!

Based on Huffman, 1952

Lemma

If u and v are leaves in T^* and $depth_{T^*}(u) < depth_{T^*}(v)$ then $f_u \geq f_v$.
(Proof by contradiction and exchange argument, showing decrease of ABL.)

Siblings claim

For every optimal prefix code T, there is an optimal T^* where the two lowest-frequency items are assigned to leaves that are siblings at the lowest level.

Let's fix it!

Based on Huffman, 1952

Lemma

If u and v are leaves in T^* and $depth_{T^*}(u) < depth_{T^*}(v)$ then $f_u \geq f_v$.
(Proof by contradiction and exchange argument, showing decrease of ABL.)

Siblings claim

For every optimal prefix code T, there is an optimal T^* where the two lowest-frequency items are assigned to leaves that are siblings at the lowest level.

Proof

- From Lemma we see that the lowest frequency item is assigned to the lowest level.

elft

Let's fix it!

Based on Huffman, 1952

Lemma

If u and v are leaves in T^* and $depth_{T^*}(u) < depth_{T^*}(v)$ then $f_u \geq f_v$.
(Proof by contradiction and exchange argument, showing decrease of ABL.)

Siblings claim

For every optimal prefix code T, there is an optimal T^* where the two lowest-frequency items are assigned to leaves that are siblings at the lowest level.

Proof

- From Lemma we see that the lowest frequency item is assigned to the lowest level.
- This leaf has a sibling (for $n > 1$) because trees are full.
- The order in which items appear in a level does not matter.
- So the two lowest frequency items can be made to appear next to each other.

Let's fix it!

Based on Huffman, 1952

Lemma

If u and v are leaves in T^* and $depth_{T^*}(u) < depth_{T^*}(v)$ then $f_u \geq f_v$.
(Proof by contradiction and exchange argument, showing decrease of ABL.)

Siblings claim

For every optimal prefix code T, there is an optimal T^* where the two lowest-frequency items are assigned to leaves that are siblings at the lowest level.

Now what?

Let's fix it!

Based on Huffman, 1952

Lemma

If u and v are leaves in T^* and $depth_{T^*}(u) < depth_{T^*}(v)$ then $f_u \geq f_v$.
(Proof by contradiction and exchange argument, showing decrease of ABL.)

Siblings claim

For every optimal prefix code T, there is an optimal T^* where the two lowest-frequency items are assigned to leaves that are siblings at the lowest level.

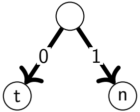
Now what?

Idea: Create tree bottom-up. Make two leaves for two lowest frequency letters y and z. Recursively build tree for the rest using a meta-letter for yz.

Let's try it out!

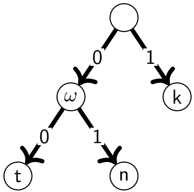
$f_a = 0.32, f_e = 0.25, f_k = 0.2, f_n = 0.18, f_t = 0.05$

Lowest frequencies: n and t, together 0.23



Let's try it out!

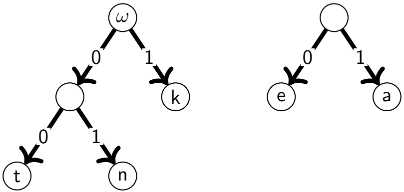
$f_a = 0.32$, $f_e = 0.25$, $f_k = 0.2$, $f_\omega = 0.23$
Lowest frequencies: k and ω, together 0.43



Let's try it out!

$f_a = 0.32$, $f_e = 0.25$, $f_\omega = 0.43$

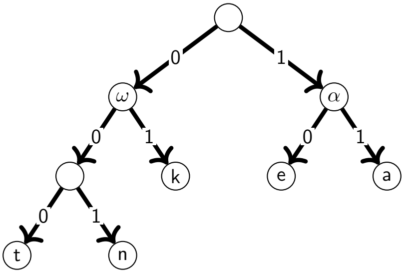
Lowest frequencies: e and a, together 0.57



Let's try it out!

$f_\alpha = 0.57$, $f_\omega = 0.43$

Lowest frequencies: ω and α



Getting it into a computer?

function HUFFMAN(S)
  if $|S| = 2$ **then**
    return tree with root and 2 leaves
  else
    let y and z be the lowest frequency letters in S
    $S' \leftarrow S - \{y, z\} \cup \{\omega\}$, so that $f_\omega = f_y + f_z$
    $T' \leftarrow \textsc{Huffman}(S')$
    $T \leftarrow$ add two children y and z to leaf ω in T'
    return T

Question: Efficient?

How do we implement this efficiently?

Answer: PQs galore!

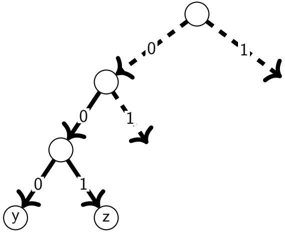
With a priority queue for S we can implement this in $O(n \log n)$ time!

But is it optimal?

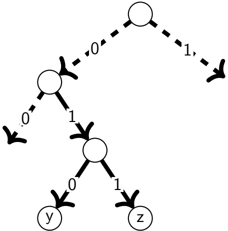
Well yes, but let us convince you!

Claim: Huffman code for S achieves the minimal ABL of any prefix code.

Huffman T:



Some optimal tree Z:

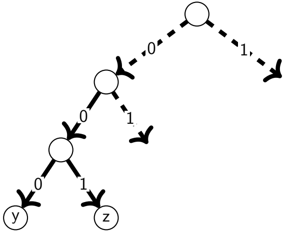


But is it optimal?

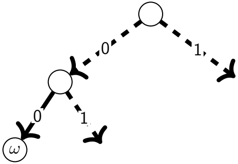
Well yes, but let us convince you!

Claim: Huffman code for S achieves the minimal ABL of any prefix code.

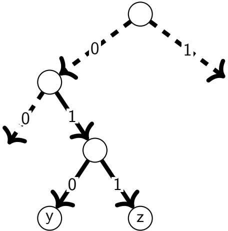
Huffman T:



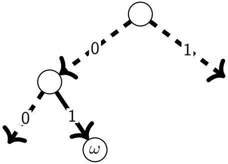
T'



Some optimal tree Z:



Z'



But is it optimal?

Well yes, but let us convince you!

Claim: Huffman code for S achieves the minimal ABL of any prefix code.

Proof by induction (sketch).

Base case ($n = 2$): there is no shorter code than a root and two leaves.
IH: The Huffman tree T' of any S' of size $n-1$ is optimal.
Induction step:

- Let Z be the optimal prefix code for S of size n, and T be the Huffman tree.
- Delete the lowest frequency items y and z from Z to create Z' of size $n-1$'.
- Same for T to create T' of size $n-1$.
- The induction hypothesis (T' is optimal) implies that $ABL(T') \leq ABL(Z')$.
- Question: how do $ABL(T')$ and $ABL(Z')$ relate to $ABL(T)$ and $ABL(Z)$?
- Then $ABL(T) \leq ABL(Z)$, and thus T is optimal.

$\square$

Quick side-step

Claim: $ABL(T') = ABL(T) - f_\omega$ when T' is T with y, z replaced with ω.

Quick side-step

Claim: $ABL(T') = ABL(T) - f_\omega$ when T' is T with y, z replaced with ω.

Proof.

$$
\begin{aligned}
ABL(T) &= \sum_{x \in S} f(x) \cdot depth_T(x) \\
&= f(y) \cdot depth_T(y) + f(z) \cdot depth_T(z) + \sum_{x \in S - \{y,z\}} f(x) \cdot depth_T(x) \\
&= (f_y + f_z) \cdot (1 + depth_T(\omega)) + \sum_{x \in S - \{y,z\}} f(x) \cdot depth_T(x) \\
&= f_\omega \cdot (1 + depth_T(\omega)) + \sum_{x \in S - \{y,z\}} f(x) \cdot depth_T(x) \\
&= f_\omega + \sum_{x \in S'} f(x) \cdot depth_T(x) \qquad \text{(including } \omega \text{ in the sum)} \\
&= f_\omega + ABL(T')
\end{aligned}
$$

Finishing our proof

Claim: Huffman code for S achieves the minimal ABL of any prefix code.

Proof by induction.

Base case ($n = 2$): there is no shorter code than a root and two leaves.
IH: The Huffman tree T' of any S' of size $n - 1$ is optimal.
Induction step:

- Let Z be the optimal prefix code for S of size n, and T be the Huffman tree.
- Using the siblings claim we may assume w.l.o.g. that the lowest frequency items y and z are siblings in Z (and they are by definition siblings in T).
- Let Z' and T' be the trees created by replacing y and z by ω.
- The induction hypothesis (T' is optimal) implies that $ABL(T') \leq ABL(Z')$.
- We know that $ABL(Z') = ABL(Z) - f_\omega$ and $ABL(T') = ABL(T) - f_\omega$.
- Thus also $ABL(T) \leq ABL(Z)$, and thus T is optimal.

□

Old exam question: Dr. Huffman

5 minutes (+5 minutes)

Question: Let's code it up

Dr. Huffman is given the following letters to encode using an optimal prefix code: $\{p, e, a, r, l\}$ with the following frequencies: $f_p = 0.2, f_e = 0.35, f_a = 0.08, f_r = 0.12, f_l = 0.25$. Which of the following statements about Huffman's optimal prefix code is **true** ?

- A. The encodings for p, e, and l are all of the same length.
- B. The encodings for p, r, and a are all of the same length.
- C. The shortest encoding is of length 1 and is for the letter e.
- D. There is one letter with an encoding of length 4, which is for the letter a.

Old exam question: Dr. Huffman

5 minutes (+5 minutes)

Question: Let's code it up

Dr. Huffman is given the following letters to encode using an optimal prefix code: $\{p, e, a, r, l\}$ with the following frequencies:
$f_p = 0.2, f_e = 0.35, f_a = 0.08, f_r = 0.12, f_l = 0.25$. Which of the following statements about Huffman's optimal prefix code is **true** ?

- A. The encodings for p, e, and l are all of the same length.
- B. The encodings for p, r, and a are all of the same length.
- C. The shortest encoding is of length 1 and is for the letter e.
- D. There is one letter with an encoding of length 4, which is for the letter a.

Answer: Answer A

A possible correct encoding has:
$c(a) = 000, c(r) = 001, c(p) = 01, c(e) = 10, c(l) = 11$. So answer A is true.

Old exam question: Translated & slightly rephrased in the process.

10 minutes (+10 minutes)

Question: Placing pubs

There are houses along a road, which all want access to a pub. To ensure that people do not have to travel far after visiting a pub (this often leads to accidents), every house should have a pub within cycling distance, 5km. To minimise cost, we also want to minimise the number of pubs. Given distances $x_1, \ldots, x_n$, the municipality uses this algorithm to place the pubs:

Sort and relabel distances $x_1, \ldots, x_n$
$l \leftarrow -\infty; j \leftarrow 0$
for $i \leftarrow 1$ to n **do**
    if $|x_i - l| > 5$ **then**
        print $x_i + 5$
        $l \leftarrow x_i + 5$
        $j \leftarrow j + 1$

Prove the algorithm is optimal, using the greedy stays ahead proof strategy.

Greedily filling your backpack

10 minutes (+5 minutes)

Problem: The lazy fitness

You have decided to start training your upper body strength. To this end you want to carry a weight w around with you every day.

You have n categories of items, with num_i items per category and a weight of $weight_i$ weight per item for $1 \leq i \leq n$.

Implement a greedy strategy for determining as few items of each category as possible needed (with a greedy strategy) to get to weight w.

Greedily filling your backpack

10 minutes (+5 minutes)

Problem: The lazy fitness

You have decided to start training your upper body strength. To this end you want to carry a weight w around with you every day.
You have n categories of items, with num_i items per category and a weight of $weight_i$ weight per item for $1 \leq i \leq n$.
Implement a greedy strategy for determining as few items of each category as possible needed (with a greedy strategy) to get to weight w.

Hang on...

Does this greedy strategy lead to a minimal number of items for every input...?
Problem for another day I guess...?

TU Delft

You are here

The course so far

- Introduction
- Greedy algorithms and proofs: scheduling, MSTs, clustering

Today's content

- Huffman's Optimal Encoding
- Some exam-level assignments
- Q&A

The future

- Divide & Conquer algorithms
- Dynamic programming
- Network Flow

What is still unclear?

Question: After every lecture...

Give us some homework and tell us:
What is still unclear after attending today's lecture?

Homework for this week

- Before next lecture:
  - Study Chapter 4:
    - Huffman codes (Ch 4.8)
  - Do all skills of module Greedy (for your chosen path)

Homework for this week

- Before next lecture:
  - Study Chapter 4:
    - Huffman codes (Ch 4.8)
  - Do all skills of module Greedy (for your chosen path)
- Next TA check:
  - *Greedy Triathlon*: November 25 (tomorrow)

CSE2310 Algorithm Design

Lecture/Q&A 4: Greedy Algorithms

Stefan Hugtenburg, Emir Demirović, and Mathijs de Weerdt

Algorithmics group — EEMCS — TU Delft

2023–2024 Q2