CS 218 Design and Analysis of Algorithms

Nutan Limaye

Indian Institute of Technology, Bombay
nutan@cse.iitb.ac.in

Module 1: Basics of algorithms

Huffman Coding

Problem Description

- Given a file with data coming from an alphabet (say English alphabet).
- Convert it into a binary alphabet using as few bits as possible while keeping it uniquely decodable.

Suppose the file has the following letters $\Sigma = \{a, b, c, d, e\}$.

letters	a	b	c	d	e
frequency	100	3	5	45	20
Encoding	000	001	010	011	100

The file will be encoded into $(100 + 3 + 5 + 45 + 20) \times 3 = 519$ bits.

Huffman Coding

Problem Description

- Given a file with data coming from an alphabet (say English alphabet).
- Convert it into a binary alphabet using as few bits as possible while keeping it uniquely decodable.

Suppose the file has the following letters $\Sigma = \{a, b, c, d, e\}$.

letters	a	b	c	d	e
frequency	100	3	5	45	20
Encoding	0	100	010	1	01

The file will be encoded into
$100 \times 1 + 45 \times 1 + 20 \times 2 + 3 \times 3 + 5 \times 3 = 209$ bits.

But 0101 can be decoded as adad, or cd, or ee, or eda.

Prefix-free encoding of strings - No word is prefix of another

Encode the letters so that each has a unique prefix. Helps in decoding uniquely.

letters	a	b	c	d	e
frequency	100	3	5	45	20
Encoding	1	0000	0001	01	001

The file will be encoded into
$100 \times 1 + 45 \times 2 + 20 \times 3 + 3 \times 4 + 5 \times 4 = 282$ bits.

Moreover any string can be uniquely decoded.

E.g. 011 is decoded as da, 00101 is decoded as ed.

Designing a Greedy Strategy

Greedy Strategy 1

- Sort the frequencies in the non-increasing order $f_1 \geq f_2, \ldots \geq f_n$.
- Use i-length prefix-free string for f_i for each $i \in [n]$.

This almost seems to work for our example.

letters	a	b	c	d	e
frequency	100	3	5	45	20
Encoding	1	0000	0001	01	001

- But this is not optimal. Can you see why?

Designing a Greedy Strategy

Greedy Strategy 1

- Sort the frequencies in the non-increasing order $f_1 \geq f_2, \ldots, f_n$.
- Use i-length prefix-free string for f_i for each $i \in [n]$.

- But this is not optimal. Can you see why?

letters	a	b	c	d	e
frequency	100	20	40	80	45
Encoding 1	1	0000	0001	01	001
Encoding 2	11	000	001	10	01

Using encoding 1 we get:
$100 \times 1 + 80 \times 2 + 45 \times 3 + 40 \times 4 + 20 \times 4 = 635$.

Using encoding 2 we get:
$100 \times 2 + 80 \times 2 + 45 \times 2 + 40 \times 3 + 20 \times 3 = 630$.

Designing a Greedy Strategy

Greedy Strategy 2 (Top-down approach)

- Divide the letters into two sets of almost equal frequencies.
- Recursively code the two sets.

This almost seems to work for our example.

letters	a	b	c	d	e
frequency	100	3	5	45	20
Encoding	1	0000	0001	01	001

- But this is also not optimal.

Greedy strategy 2 and its tree

Tree associated with the top-down strategy.

We can naturally associate this strategy with a tree.

The nodes of the tree are subsets of the alphabet.

The leaves are single letters.

The depth of a leaf is the length of the code word assigned to it.

How do we extract the exact code word attached to a node?

Specifying the tree completely specifies the code generated by the strategy.

Designing a Greedy Strategy

Greedy Strategy 2 (Top-down approach)

- Divide the letters into two sets of almost equal frequencies.
- Recursively code the two sets.

- But this is not optimal.

letters	a	b	c	d	e
frequency	100	20	40	80	45
Encoding 1	11	000	10	01	001
Encoding 2	11	000	001	10	01

Using encoding 1 we get:
$100 \times 2 + 80 \times 2 + 45 \times 3 + 40 \times 2 + 20 \times 3 = 635$.

Using encoding 2 we get:
$100 \times 2 + 80 \times 2 + 45 \times 2 + 40 \times 3 + 20 \times 3 = 630$.

Is it a counter-example?

Can we recover Encoding 2 using the top-down approach?

letters	a	b	c	d	e
frequency	100	20	40	80	45
Encoding 1	11	000	10	01	001
Encoding 2	11	000	001	10	01

In fact, we can! This is not a valid counter-example.

A valid counter example.

letters	a	b	c	d
frequency	11	5	11	5
Top-down encoding	00	01	10	11
Optimal encoding	0	111	10	110