Codes

Bakh Khoussainov

Codes

We want to code texts as sequences of 0's and 1's.

A simplest way:

Use a fixed number of bits for each symbol.

Example: For texts written in English we have 26 letters and 5 punctuation characters (comma, period, question mark, exclamation, apostrophe) and the space. This totals to 32.

There are $2^5 = 32$ five-bit binary words; we can use each such word to encode the 32 symbols.

For instance

00000	for	a
00001	for	b
00010	for	c
⋮		

So, the code for the word abba becomes

00000 00001 00001 00000

This coding does not take into account the frequency of the letters.

If we want to compress our codes (of texts) we need to be more clever.

Idea:

Use short binary words to represent frequently used letters.

We need to be careful though.

Consider:

0 represents a

1 represents b

00 represents c

01 represents d.

The code for "bad" is 1001.

However 1001 can be decoded ambiguously:

baab, bcb, bad.

Prefix codes fix the ambiguity of decoding.

Let S be a set of letters. A prefix code of S is a rule f that maps each $x \in S$ to a binary word $f(x)$ such that for distinct letters $x, y \in S$ $f(x)$ and $f(y)$ are not prefixes of one AND the other.

Example.

$S = \{ a, b, c, d \}$

$d_1(a) = 11 \qquad d_2(a) = 11$

$d_1(b) = 01 \qquad d_2(b) = 10$

$d_1(c) = 001 \qquad d_2(c) = 01$

$d_1(d) = 10 \qquad d_2(d) = 001$

$d_1(e) = 000 \qquad d_2(e) = 000$

Let S be an alphabet AND f be a prefix code for S.

Then each text

$$X_1\ X_2\ X_3\ \ldots\ X_n$$

has a code

$$f(X_1)\,f(X_2)\,f(X_3)\ldots f(x_n).$$

When we decode this sequence we get back the original text (no ambiguity occurs):

$$X_1\ X_2\ X_3\ \ldots\ X_n\ .$$

Let x be a letter, and f_x be a number representing the frequency of x appearing in texts. So, a natural condition on f_x is

$$0 \leq f_x \leq 1$$

We postulate that the sum of frequencies of letters in S equals 1:

$$\sum_{x \in S} f_x = 1.$$

Example:

$$S = \{a, b, c, d, e\}$$

$$f_a = 0.32 \quad , \quad f_b = 0.25$$

$$f_c = 0.20 \quad , \quad f_d = 0.18 \, , \quad f_e = 0.05$$

So, $\quad f_a + f_b + f_c + f_d + f_e = 1.$

Consider the sum, over all $x \in S$, of the frequencies of x times the length $\gamma(x)$:

$$\sum_{x \in S} f_x \cdot |\gamma(x)| .$$

This represents the average number of bits required per letter.

Notation: $ABL(\gamma)$.

Example. $S = \{a, b, c, d, e\}$

$f_a = 0.32$, $f_b = 0.25$, $f_c = 0.2$, $f_d = 0.18$, $f_e = 0.05$.

$d_1(a) = 11$, $d_1(b) = 01$

$d_1(c) = 001$, $d_1(d) = 10$

$d_1(e) = 000$.

Then $ABL(d_1) =$

$0.32 \times 2 + 0.25 \times 2 + 0.2 \times 3 +$

$0.18 \times 2 + 0.05 \times 3 = 2.25$

Similarly, for δ_2:

$\delta_2(a) = 11$, $\delta_2(b) = 10$, $\delta_2(c) = 01$,

$\delta_2(d) = 001$, $\delta_2(e) = 000$

we have $ABL(\delta_2) =$

$0.32 \times 2 + 0.25 \times 2 + 0.2 \times 2 +$

$0.18 \times 3 + 0.05 \times 3 = 2.23.$

So δ_2 is more optimal than δ_1. The prefix code δ_2 saves more space (or gives a better compression) than δ_1

Problem:

Input: Alphabet S, frequencies f_x for all $x \in S$.

Output: A prefix code γ for S that minimizes the average number of bits per letter

$$ABL(\gamma) = \sum_{x \in S} f_x \cdot |\gamma(x)|.$$

Binary trees and prefix codes

Example.

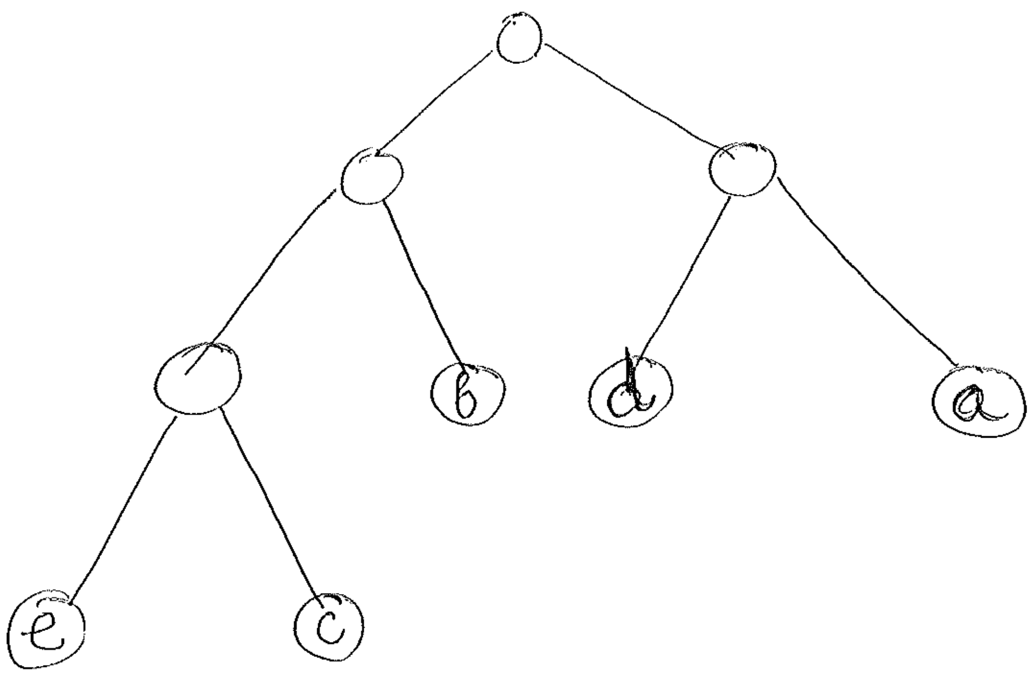
$$S = \{a, b, c, d, e\}$$

$$d_1(a) = 11, \quad d_1(b) = 01$$

$$d_1(c) = 001, \quad d_1(d) = 10$$

$$d_1(e) = 000$$

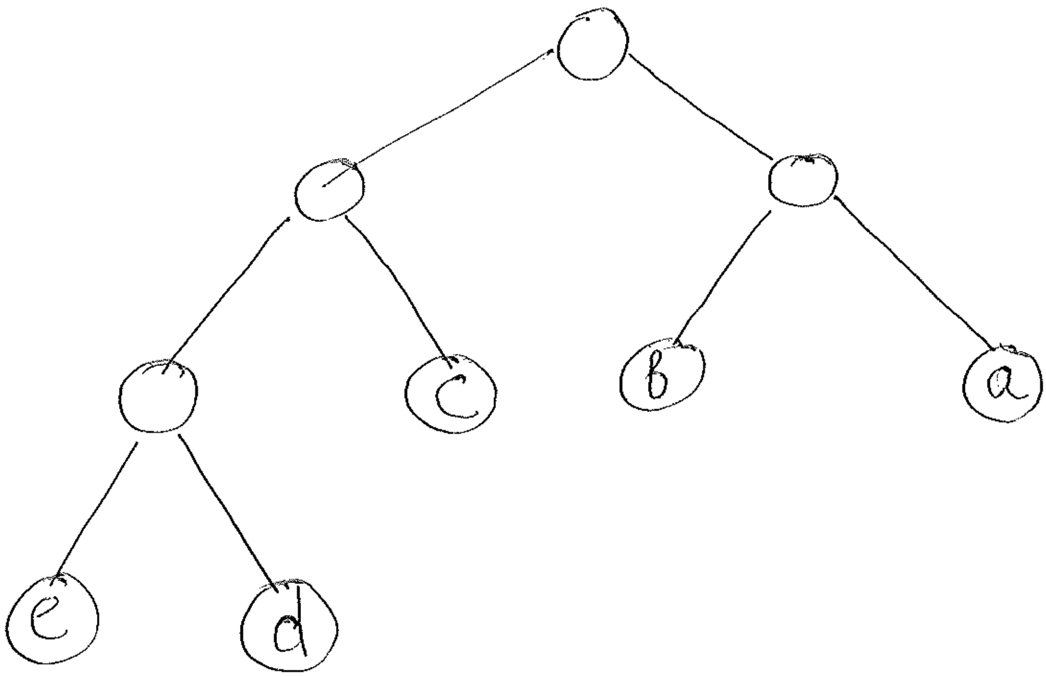
We can represent d_1 as a tree:



Similarly for δ_2:

$\delta_2(a)=11$, $\delta_2(b)=10$, $\delta_2(c)=01$

$\delta_2(d)=001$, $\delta_2(e)=000$



In general, every prefix code determines a binary tree whose leaves are labeled by letters.

Let T be a binary tree such that

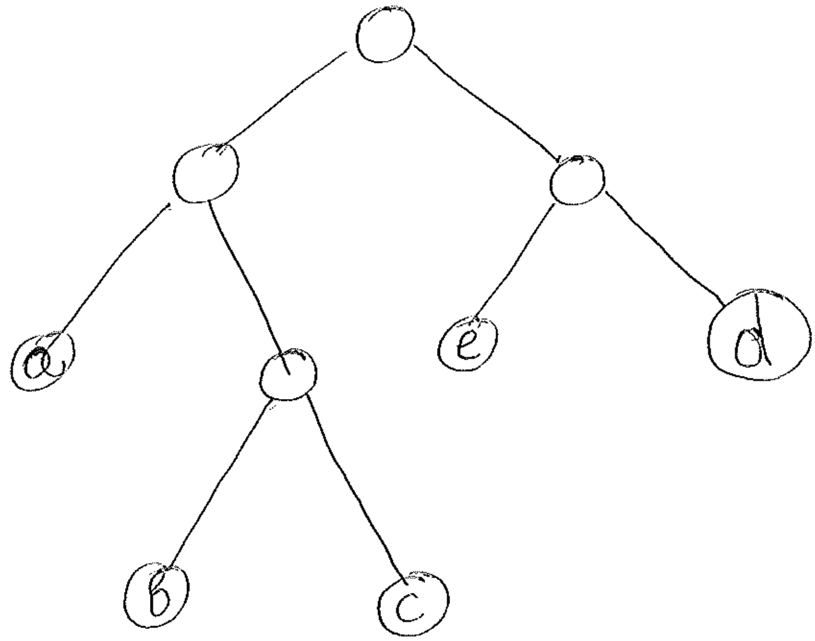
(1) The number of leaves of T equals the number of letters of S

(2) Leaves of T are labeled by distinct letters of S.

Fact. The tree above defines a prefix code of S.

Example

$S = \{a, b, c, d, e\}$



$f(00) = a \quad f(010) = b$

$f(011) = c \quad f(10) = e$

$f(11) = d$

Fact. The binary tree that corresponds to an optimal prefix code is full.

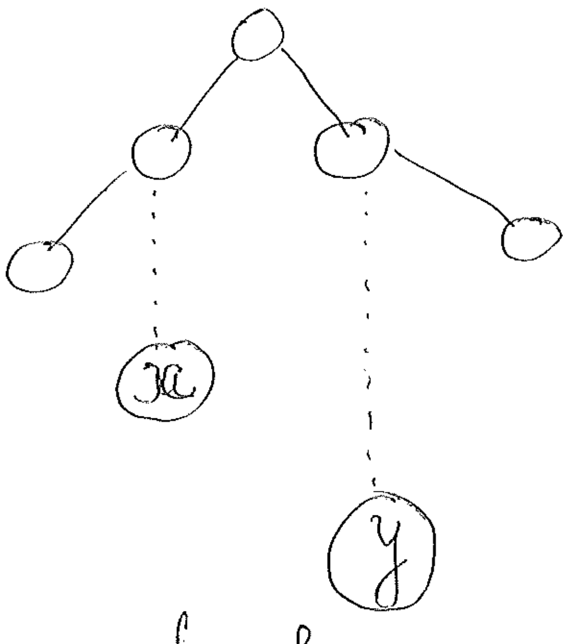
Indeed, let f be optimal. Let T be the tree built from the prefix code f.

Let u be an internal node with exactly one child.

Case 1. v is the root of T. Delete v and obtain new tree T'. The prefix code built from T' is more optimal than f.

Case 2. v is not the root. Let w be the child of v. Remove w and make the children of w to be v's children. New tree defines more optimal code than f.

Let T^* be a binary tree that corresponds to an optimal prefix code. Let u, v be leaves of T^* such that depth$(u) <$ depth(v). Then for labels x and y of u and v we have $f_x \geq f_y$.



$f_x \geq f_y$

Indeed, suppose $f_x < f_y$.

Let γ^* be the code obtained from T^*.

We change T^x by labeling u with y and v with x.

The changed tree defines a new prefix code γ.

Now

$$\sum_{x' \in S} f_{x'} \cdot |\gamma^*(x)| - \sum_{x'} f_{x'} |\gamma(x)| =$$

$$= f_x |u| + f_y |v| - (f_y \cdot |u| + f_x |v|) =$$

$$(|u| - |v|)(f_x - f_y) < 0.$$

Let T^* be a tree as above.
Let v be a node in T^*
with the largest depth.
T^* is a full binary tree.
Hence v has a sibling w,
And w must be a leaf.

This implies:

Two lowest frequency letters
of S' are assigned to
leaves that are siblings in T^*.

Huffman's algorithm:

If S has two letters, encode them by 0, 1.

Else

Let y^*, z^* be two lowest-frequency letters.

Set $S' = (S - \{y^*, z^*\}) \cup \{\omega\}$,

$$f_\omega = f_{y^*} + f_{z^*}$$

Construct a prefix code γ' for S'.
Let T' be the tree for γ'.

Define a prefix code for S as follows:

Start with T'.
Take the leaf labeled by ω.
Add two children of ω.
Label them by y^* and z^*.

Stop.

Example: $S = \{a, b, c, d, e\}$,

$f_a = 0.32$, $f_b = 0.25$, $f_c = 0.2$, $f_d = 0.18$, $f_e = 0.05$.

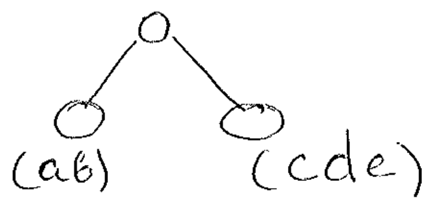
$S' = \{a, b, c, (de)\}$,

$f_{(de)} = 0.18 + 0.05 = 0.23$.

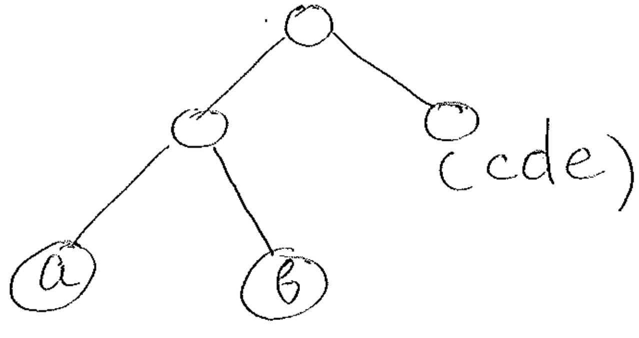
$S'' = \{a, b, (cde)\}$, $f_{(cde)} = 0.2 + 0.23 = 0.43$

$S''' = \{(ab), (cde)\}$.

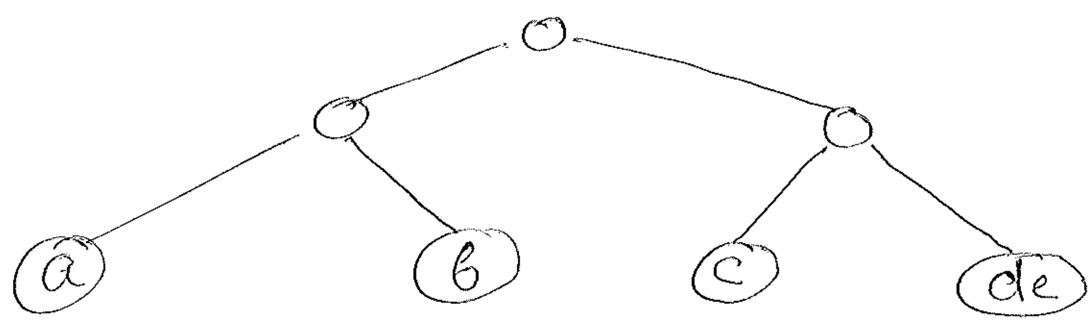
So: T''':



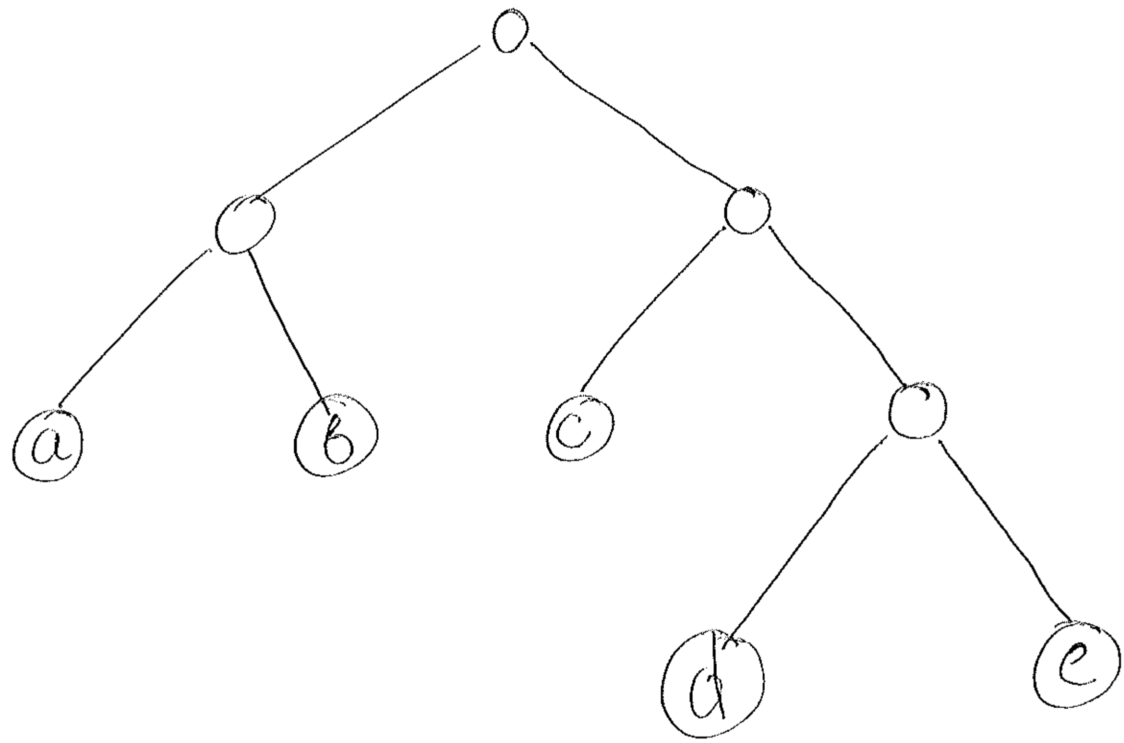
T'':



T':



T:



Now we explain why the algorithm produces an optimal solution.

This is done by induction on the size of S.

Clearly, the solution is optimal for alphabets S with exactly two letters.

Suppose S has $k+1$ letters. Let y^*, z^* be the lowest-frequency in S.

Then $S' = (S - \{y^*, z^*\}) \cup \{w\}$ with $f_w = f_{z^*} + f_{y^*}$ has k letters.

Run the algorithm on S'.
By induction it produces a tree T' determining an optimal prefix code f'.

By the algorithm T is obtained from T' by adding two leaves and labeling them by y^* and z^*.
Let f be the prefix code defined by T.

It is easy to see the following equality:

$$ABL(T) = f_\omega + ABL(T').$$

Let Z be a tree such that

$$ABL(Z) < ABL(T).$$

Define Z' from Z as we defined T' from T.

Making the same reasoning as we did for T, we have:

$$ABL(Z) = f_\omega + ABL(Z').$$

Thus

$$ABL(T) = f_\omega + ABL(T') >$$

$$> ABL(Z) = f_\omega + ABL(Z').$$

So, $ABL(T') > ABL(Z')$.

This contradicts the inductive assumption.