Great Theoretical Ideas In Computer Science

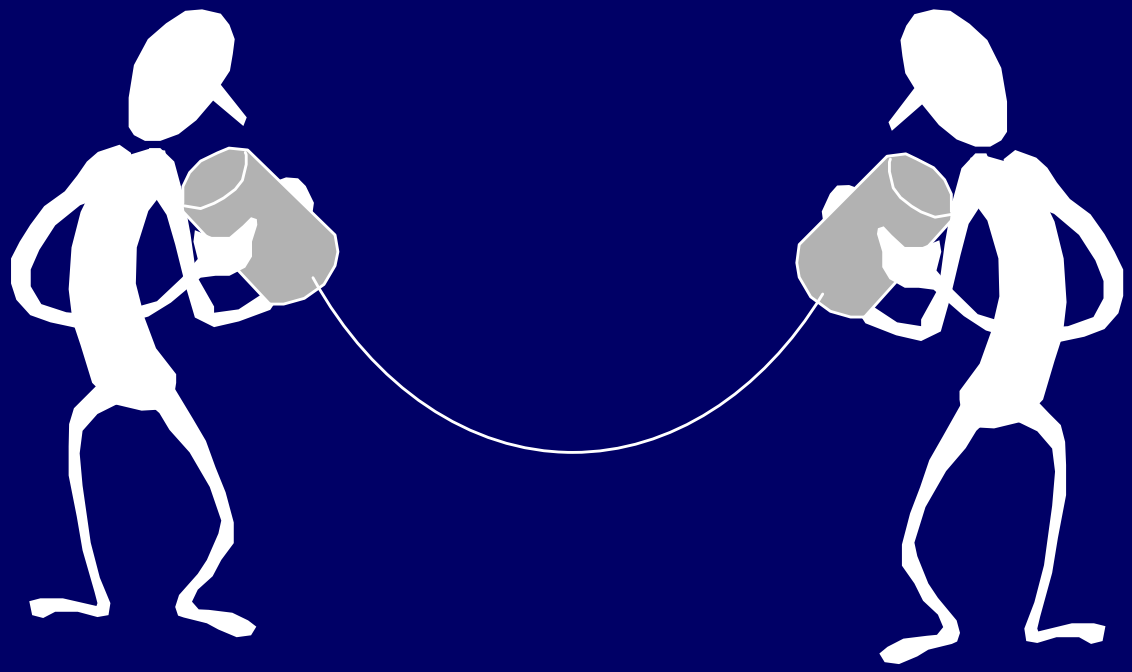
Anupam Gupta

CS 15-251 Fall 2005

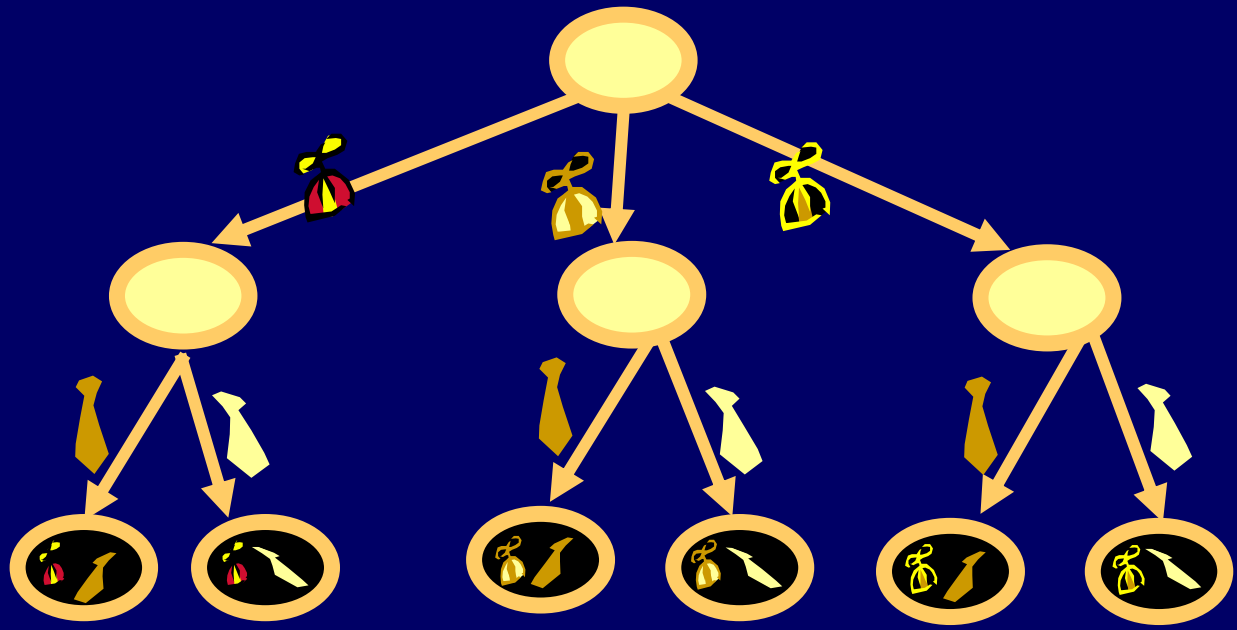
Lecture 24 Nov 17, 2005

Carnegie Mellon University

Decision Trees and Information: A Question of Bits

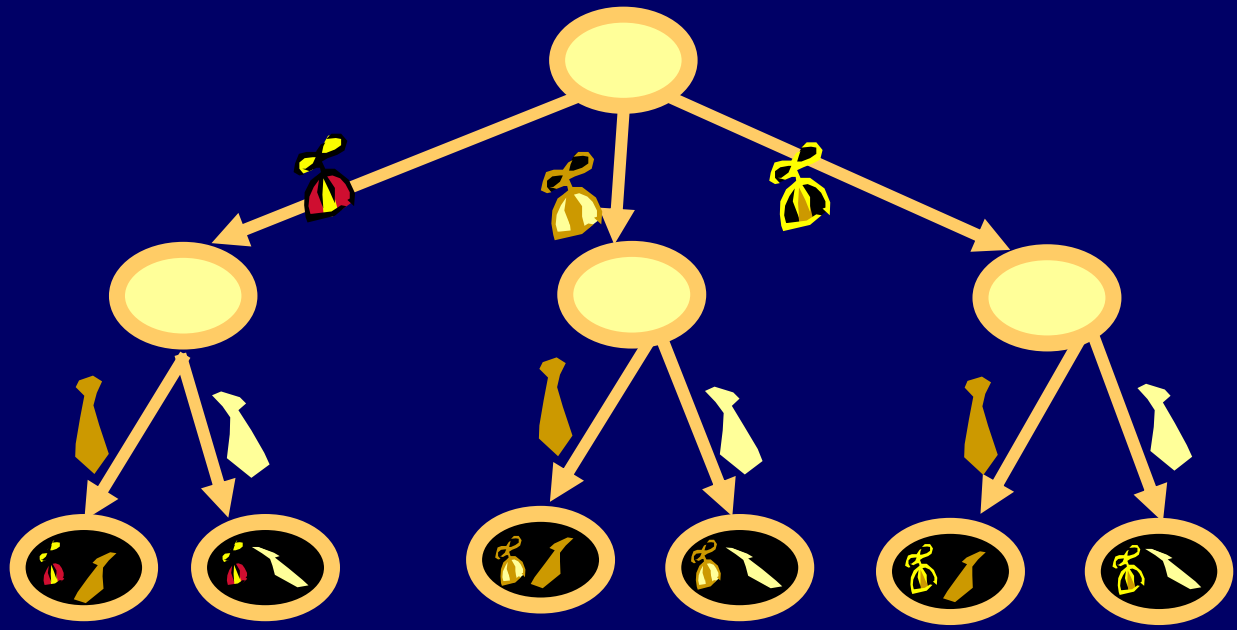


Choice Tree



A choice tree is a rooted, directed tree with an object called a "choice" associated with each edge and a label on each leaf.

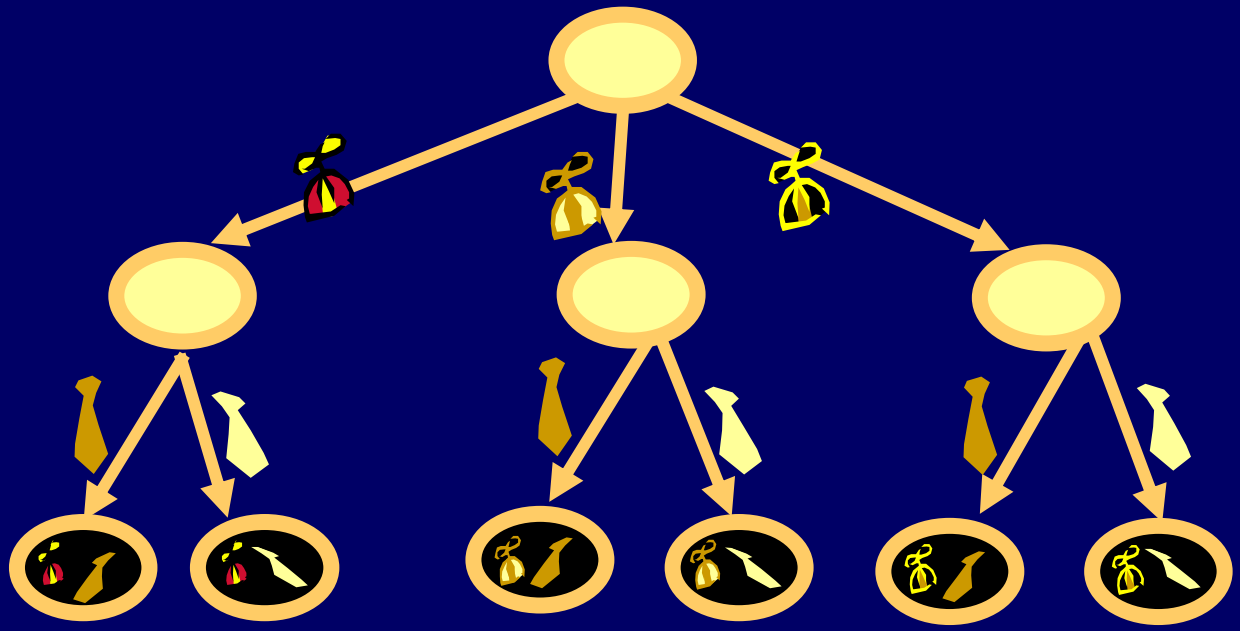
Choice Tree Representation of S



We satisfy these two conditions:

- Each leaf label is in S
- Each element from S on exactly one leaf.

Question Tree Representation of S

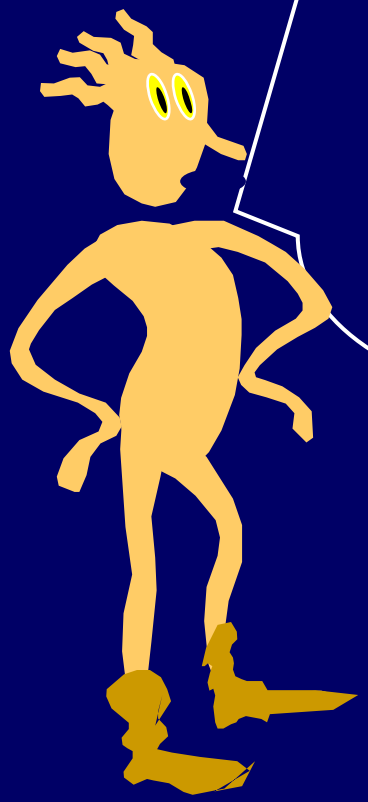


I am thinking of an outfit.
Ask me questions until you know which one.

What color is the beanie?
What color is the tie?

When a question tree has
at most 2 choices at each node,
we will call it a decision tree,
or a decision strategy.

Note: Nodes with one choices
represent stupid questions, but
we do allow stupid questions.



20 Questions

S = set of all English nouns

Game:

I am thinking of an element of S.

You may ask up to 20 YES/NO questions.

What is a question strategy for this game?

20 Questions

Suppose $S = \{a_0, a_1, a_2, \ldots, a_k\}$

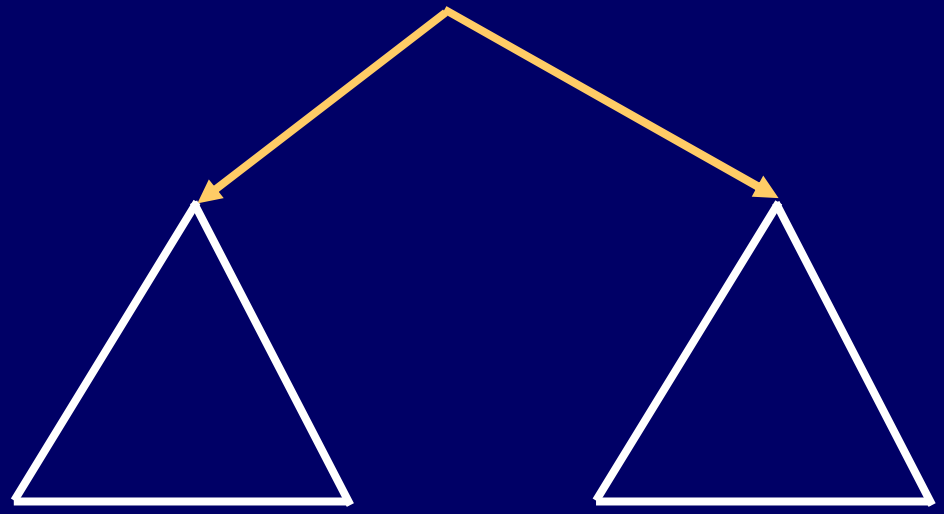
Binary search on S.

First question will be:

"Is the word in $\{a_0, a_1, a_2, \ldots, a_{(k-1)/2}\}$?"

20 Questions
Decision Tree Representation

A decision tree with depth at most 20, which has the elements of S on the leaves.



Decision tree for $\{a_0, a_1, a_2, \ldots, a_{(k-1)/2}\}$

Decision tree for $\{a_{(k+1)/2}, \ldots, a_{k-1}, a_k\}$

Decision Tree Representation

Theorem:
The binary-search decision tree for S with k+1 elements $\{ a_0, a_1, a_2, \ldots, a_k \}$ has depth

$$\lceil \log (k+1) \rceil$$
$$= \lfloor \log k \rfloor + 1$$
$$= |k|$$

"the length of k when written in binary"

Another way to look at it

Suppose you are thinking of the noun a_m in S
We ask about each bit of index m

Is the leftmost bit of m 0?

Is the next bit of m 0?

...

Theorem: The binary-search decision-tree for $S = \{ a_0, a_1, a_2, \ldots, a_k \}$ has depth

$$|k| = \lfloor \log k \rfloor + 1$$

A lower bound

Theorem: No decision tree for S (with $k+1$ elements) can have depth $d < \lfloor \log k \rfloor + 1$.

Proof:

A depth d binary tree can have at most 2^d leaves.

But $d < \lfloor \log k \rfloor + 1 \Rightarrow$ number of leaves $2^d < (k+1)$

Hence some element of S is not a leaf.

Tight bounds!

The optimal-depth decision tree
for any set S with $(k+1)$ elements has depth

$$\lfloor \log k \rfloor + 1 = |k|$$

Recall…

The minimum number of bits used to represent unordered 5 card poker hands =

$$\lceil \log_2 \binom{52}{5} \rceil$$

= 22 bits

= The decision tree depth for 5 card poker hands.

Prefix-free Set

Let T be a subset of $\{0,1\}^*$.

Definition:

T is prefix-free if for any distinct $x,y \in T$,
if $|x| < |y|$, then x is not a prefix of y

Example:

{000, 001, 1, 01} is prefix-free

{0, 01, 10, 11, 101} is not.

Prefix-free Code for S

Let S be any set.

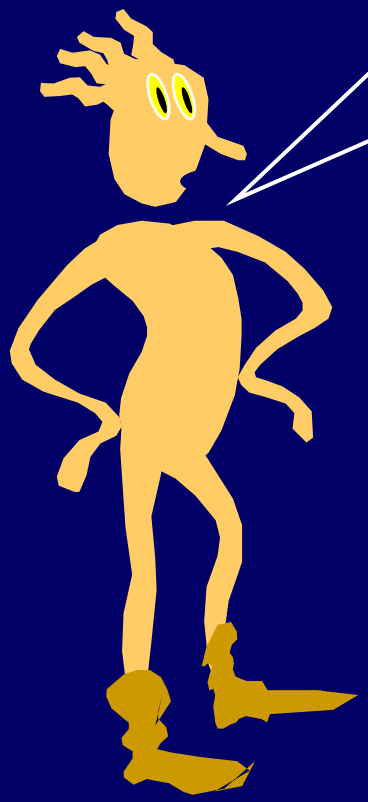
Definition: A prefix-free code for S is
a prefix-free set T and
a 1-1 "encoding" function f: S -> T.

The inverse function f^{-1} is called the "decoding function".

Example: S = {buy, sell, hold}.
T = {0, 110, 1111}.
f(buy) = 0, f(sell) = 1111, f(hold) = 110.

What is so cool about prefix-free codes?

Sending sequences of elements of S over a communications channel



Let T be prefix-free and f be an encoding function. Wish to send $\langle x_1, x_2, x_3, \ldots \rangle$

Sender: sends $f(x_1)\ f(x_2)\ f(x_3)\ldots$

Receiver: breaks bit stream into elements of T and decodes using f^{-1}

Sending info on a channel

Example: S = {buy, sell, hold}.

T = {0, 110, 1111}.

f(buy) = 0, f(hold) = 1111, f(sell) = 110.

If we see

00011011111100…

we know it must be

0 0 0 110 1111 110 0 …

and hence

buy buy buy sell hold sell buy…

Morse Code is not Prefix-free!

SOS encodes as …---…

A .-	F ..-.	K -.-	P .--.	U ..-	Z --..
B -...	G --.	L .-..	Q --.-	V ...-	
C -.-.	H	M --	R .-.	W .--	
D -..	I ..	N -.	S ...	X -..-	
E .	J .---	O ---	T -	Y -.--	

Morse Code is not Prefix-free!

SOS encodes as …---…

Could decode as: ..|.-|--|..|. = IAMIE

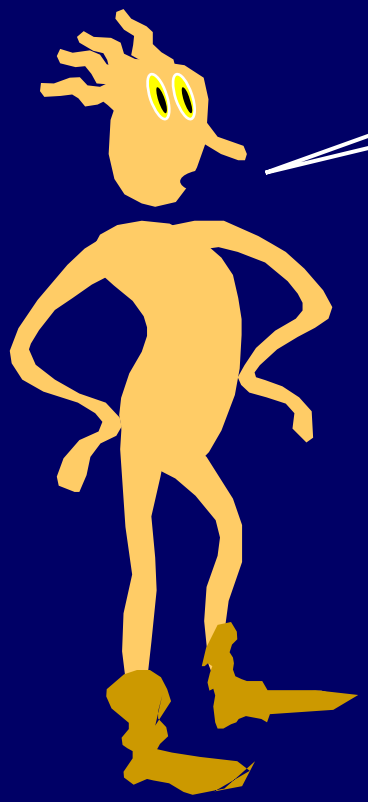
A .-	F ..-.	K -.-	P .--.	U ..-	Z --..
B -...	G --.	L .-..	Q --.-	V ...-	
C -.-.	H	M --	R .-.	W .--	
D -..	I ..	N -.	S ...	X -..-	
E .	J .---	O ---	T -	Y -.--	

Unless you use pauses

SOS encodes as … --- …

A .-	F ..-.	K -.-	P .--.	U ..-	Z --..
B -...	G --.	L .-..	Q --.-	V ...-	
C -.-.	H	M --	R .-.	W .--	
D -..	I ..	N -.	S ...	X -..-	
E .	J .---	O ---	T -	Y -.--	

Prefix-free codes are also called "self-delimiting" codes.



Representing prefix-free codes

A = 100

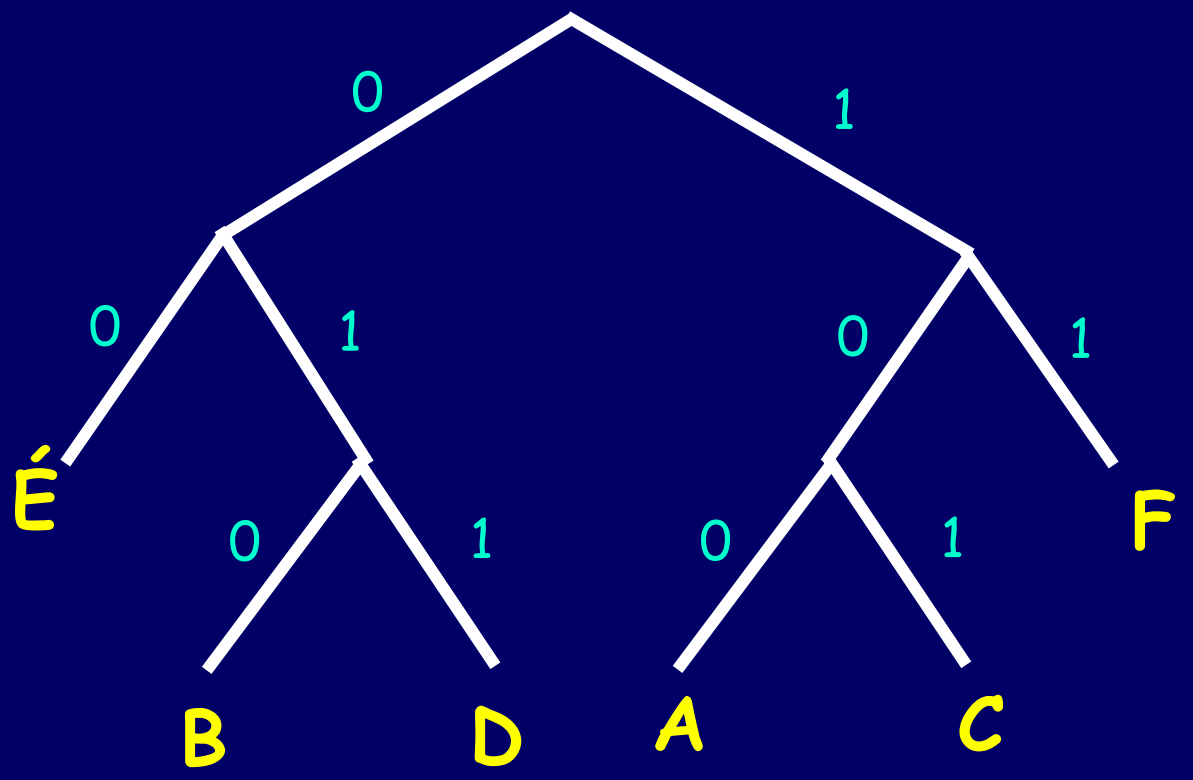
B = 010

C = 101

D = 011

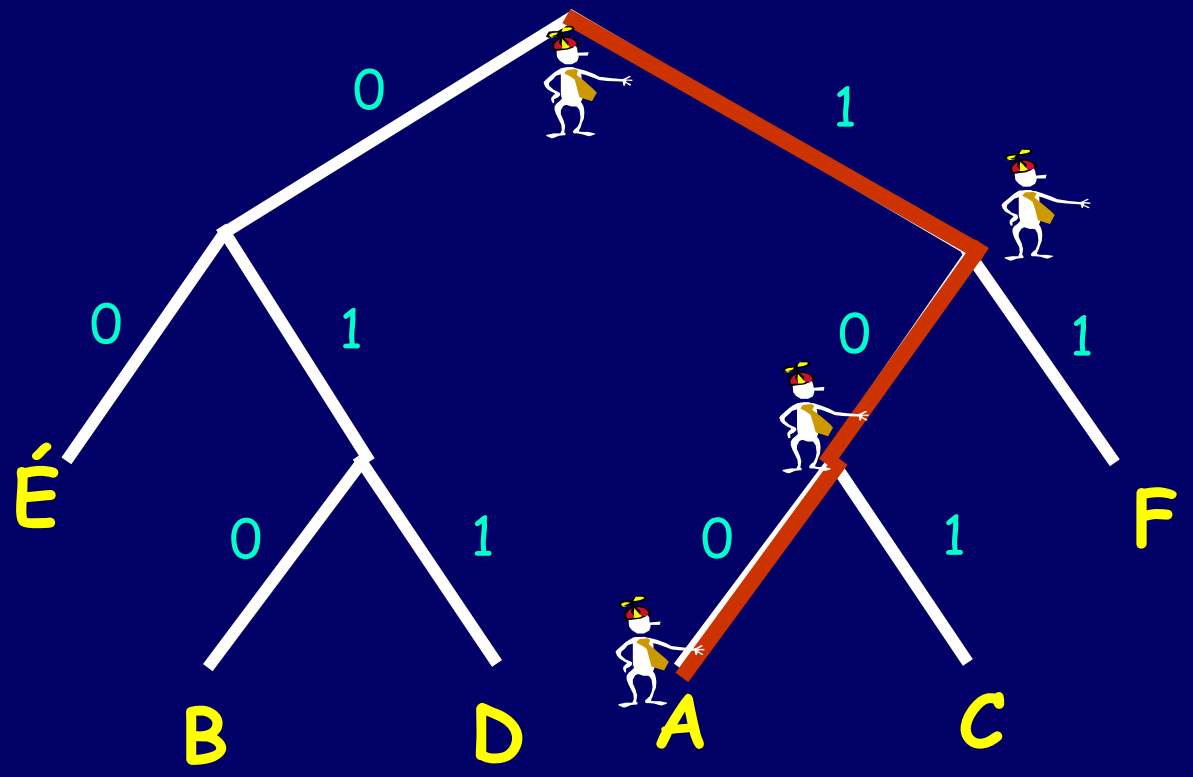
É = 00

F = 11



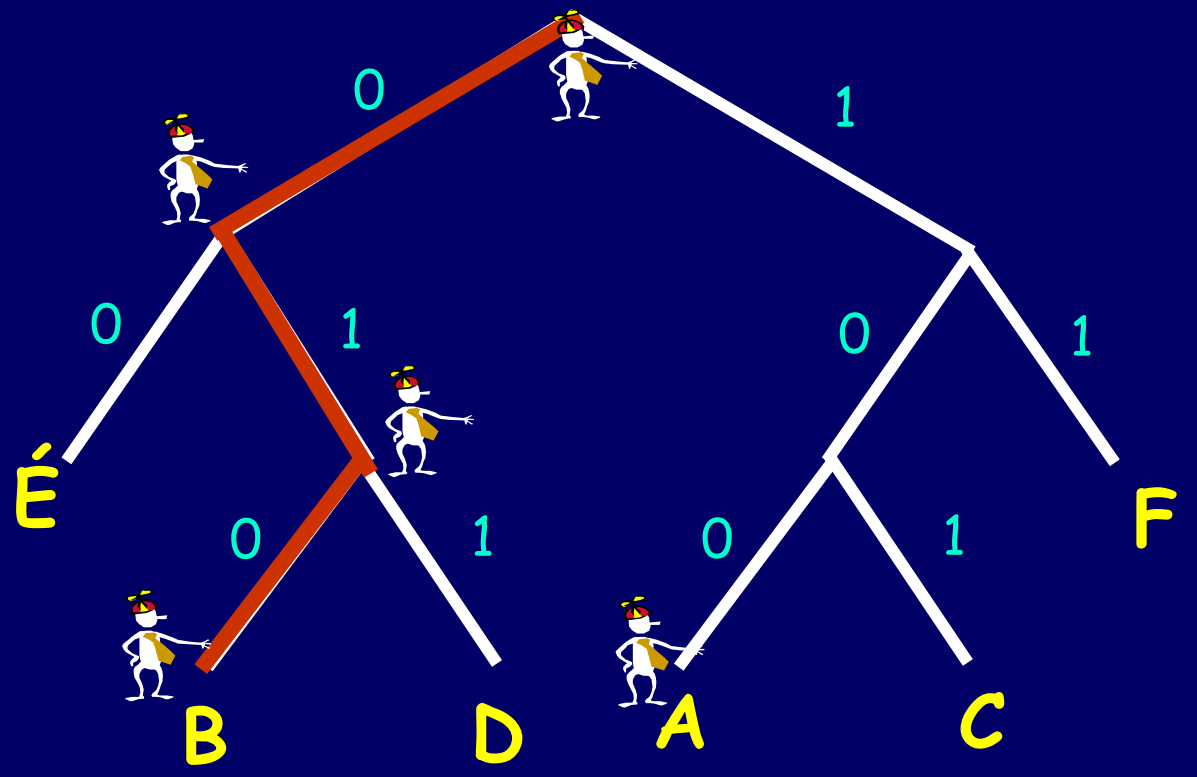
"CAFÉ" would encode as 1011001100

How do we decode 1011001100 (fast)?



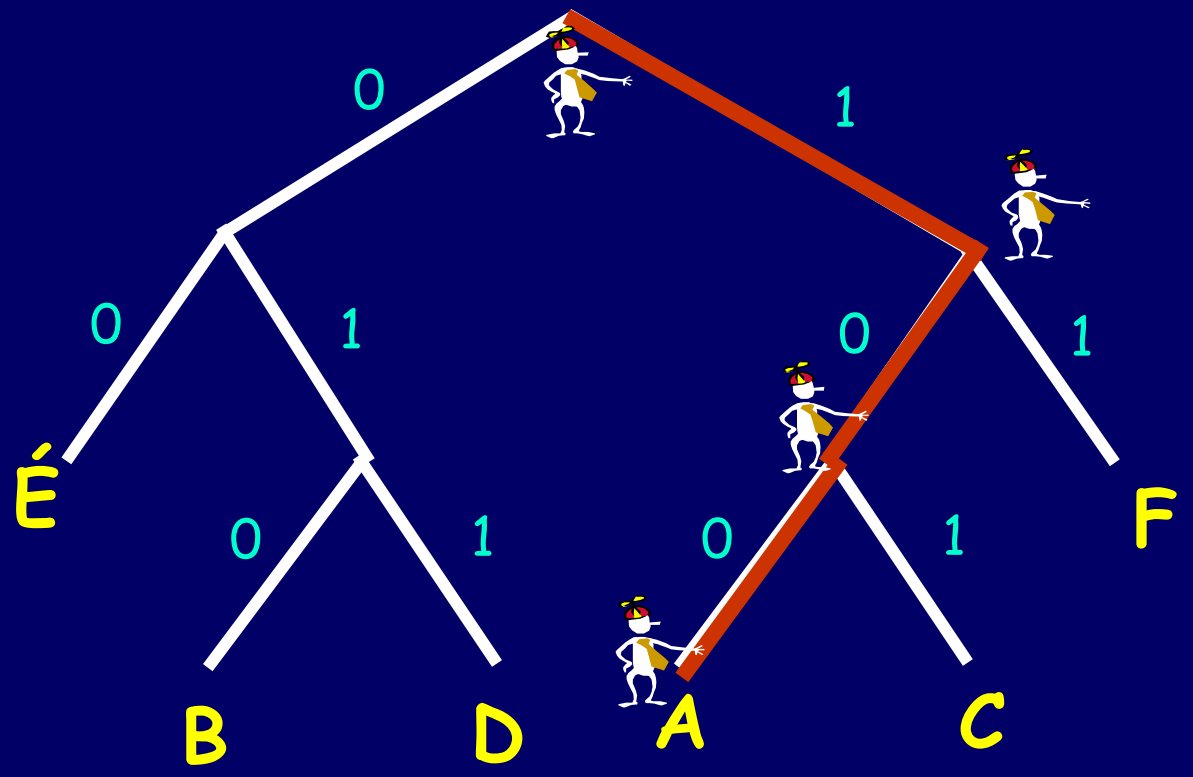
If you see: 10001010001110110011100

can decode as:



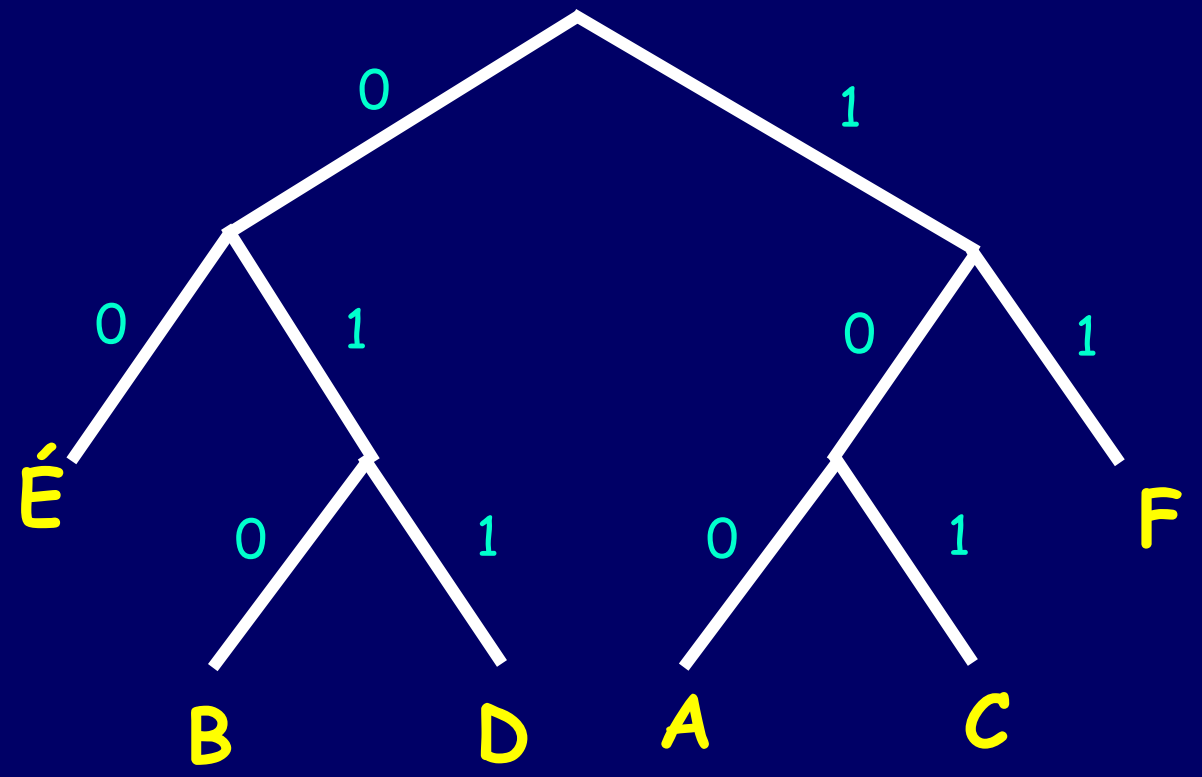
If you see: 100010100011101100 1

can decode as: A



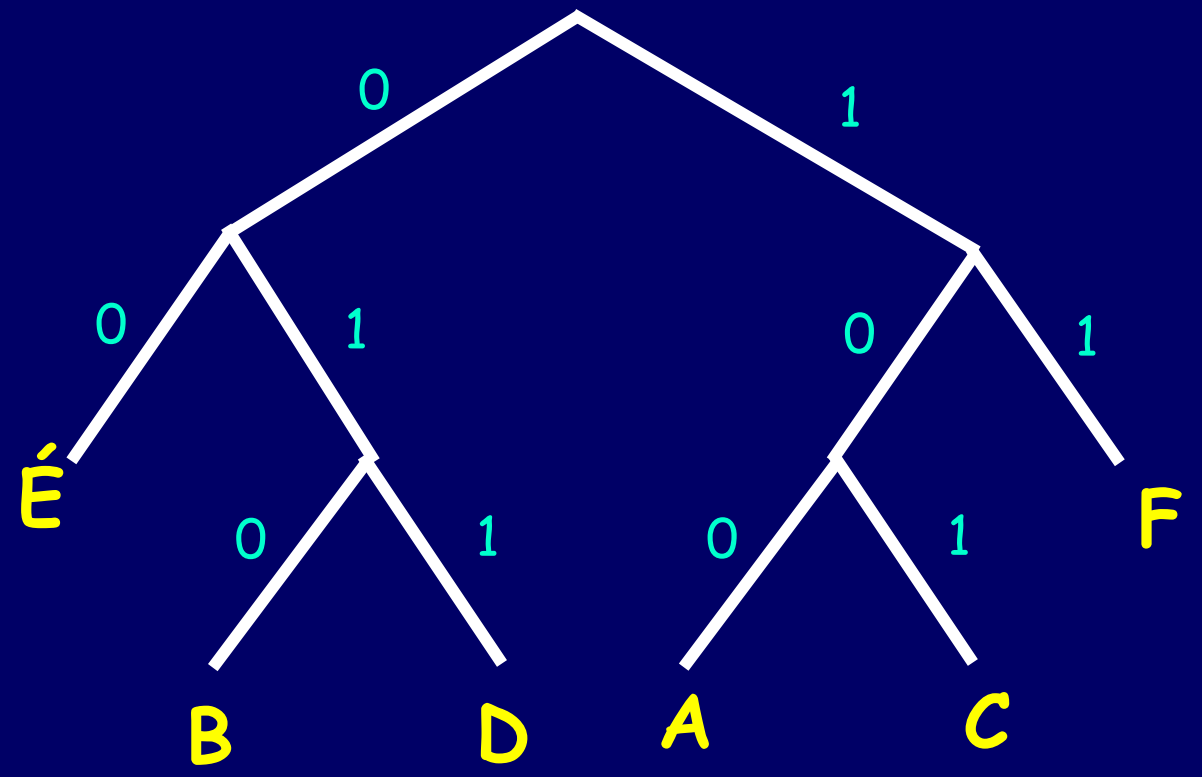
If you see: 10001010001110110011100

can decode as: AB



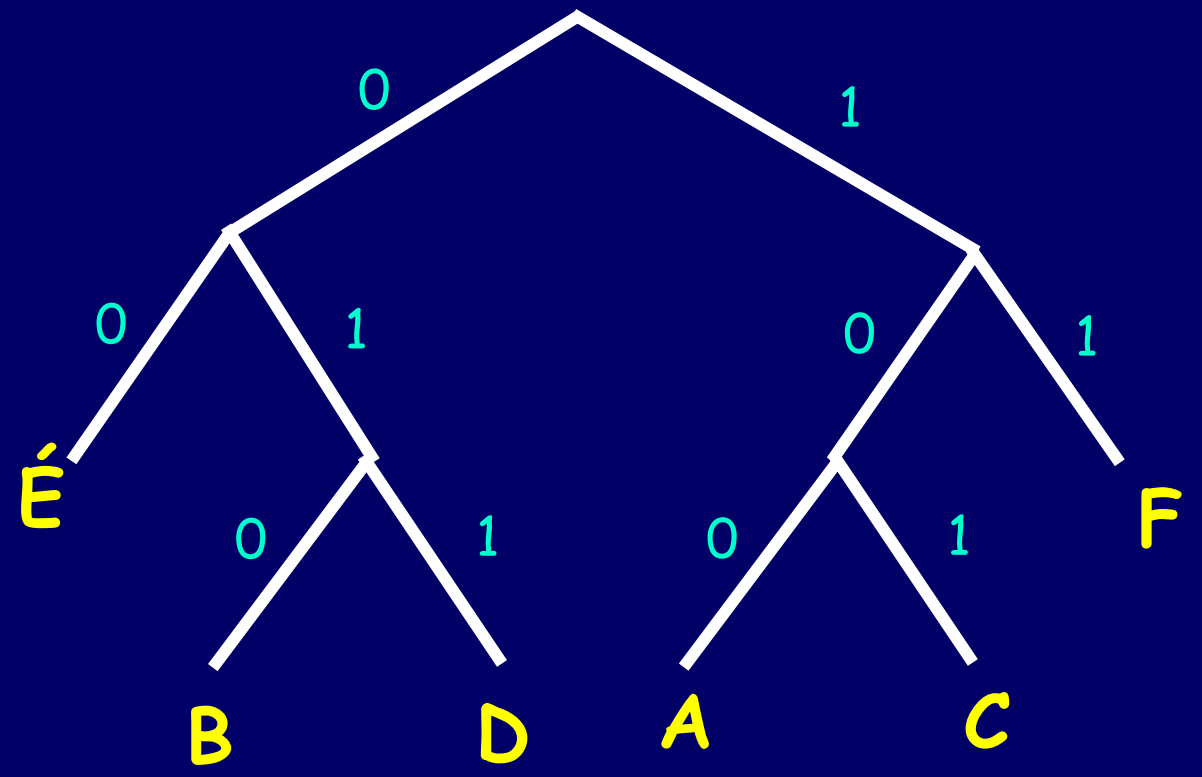
If you see: 100010100011101100 1100

can decode as: ABA



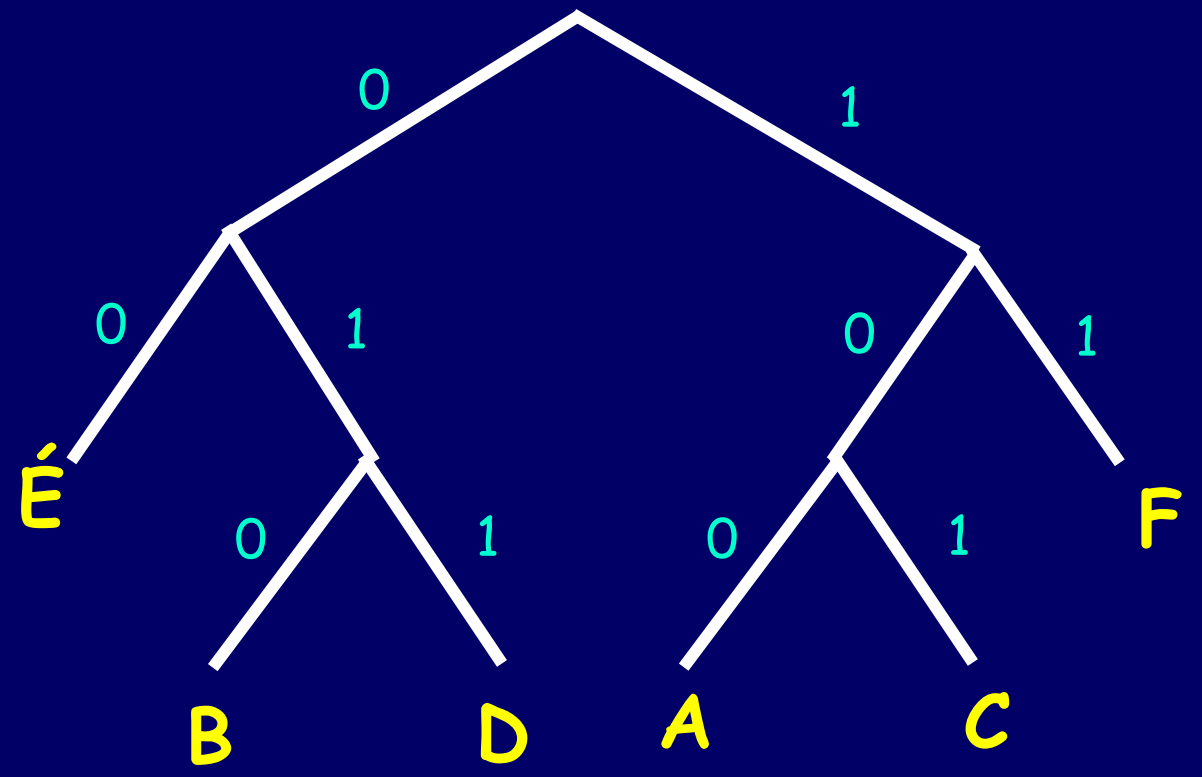
If you see: 10001010001110110011100

can decode as: ABAD



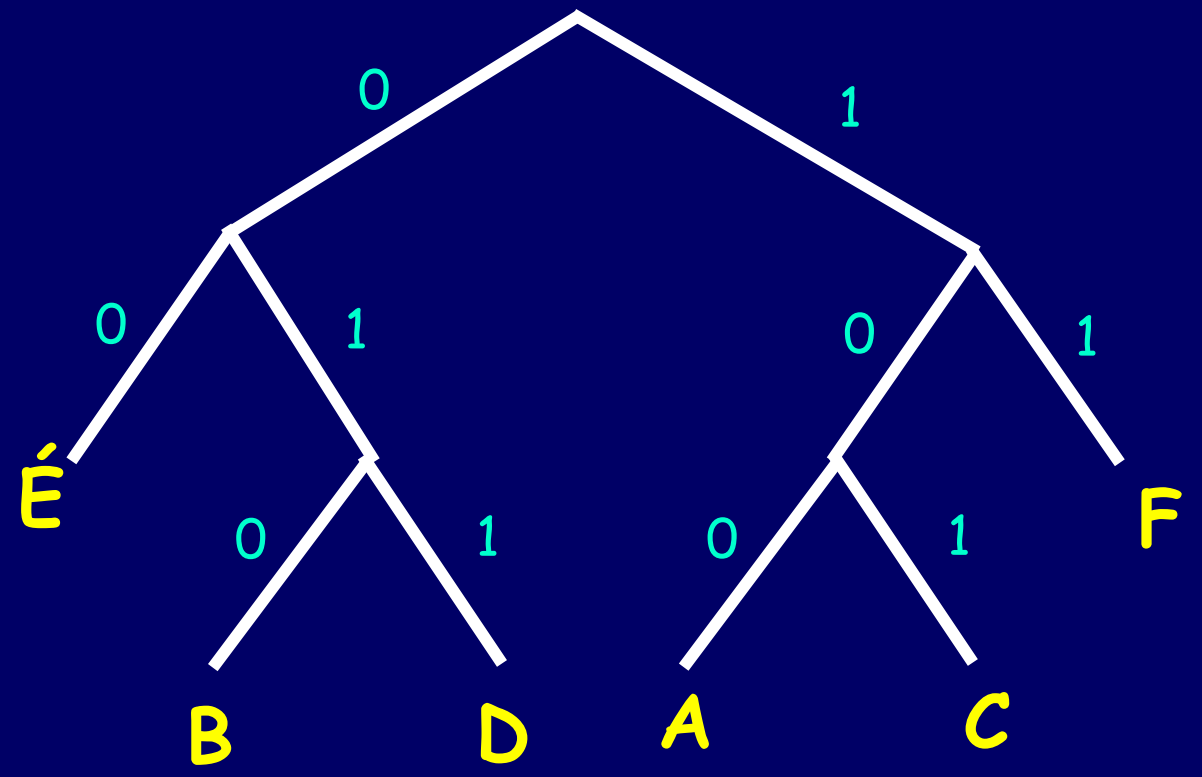
If you see: 10001010001110110011100

can decode as: ABADC



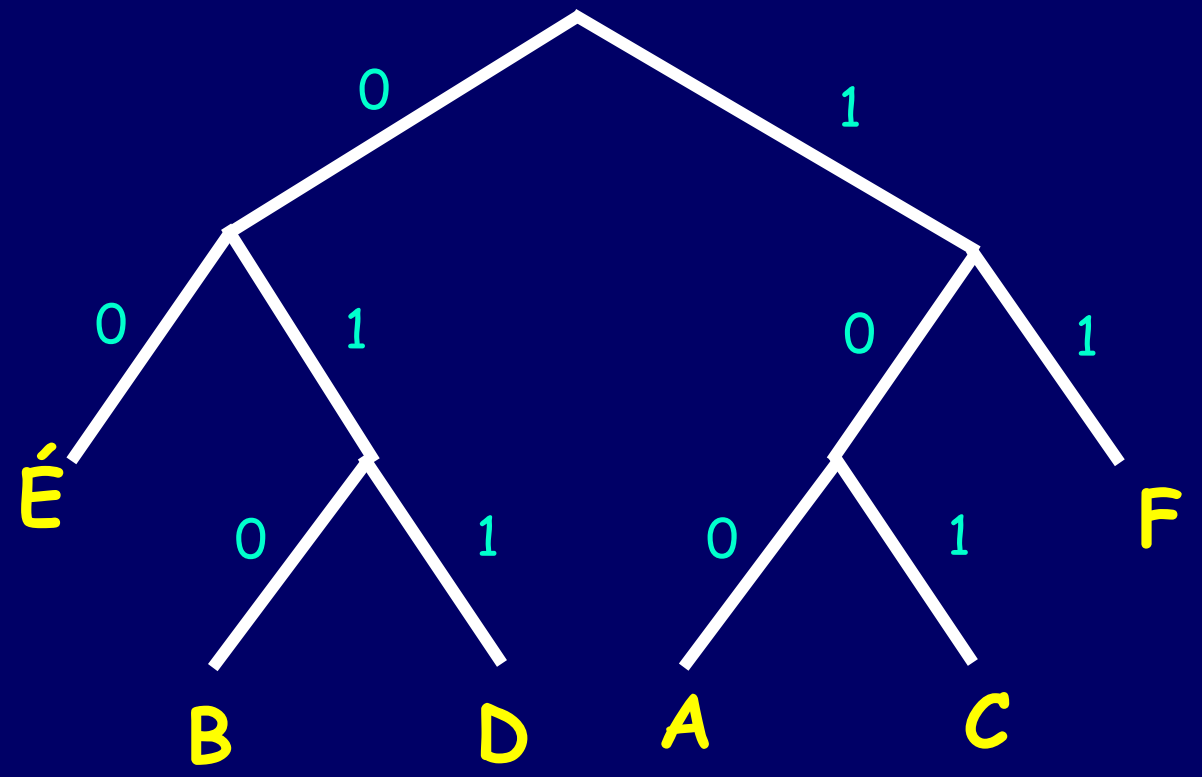
If you see: 10001010001110110011100

can decode as: ABADCA



If you see: 10001010001110110011100

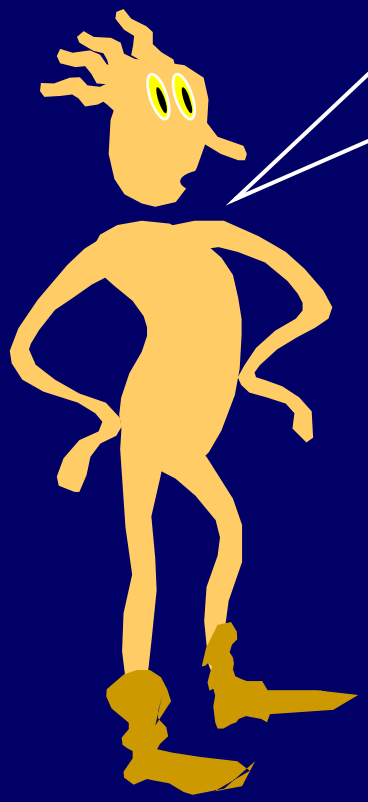
can decode as: ABADCAF



If you see: 10001010001110110011100

can decode as: ABADCAFÉ

Prefix-free codes are yet another representation of a decision tree.

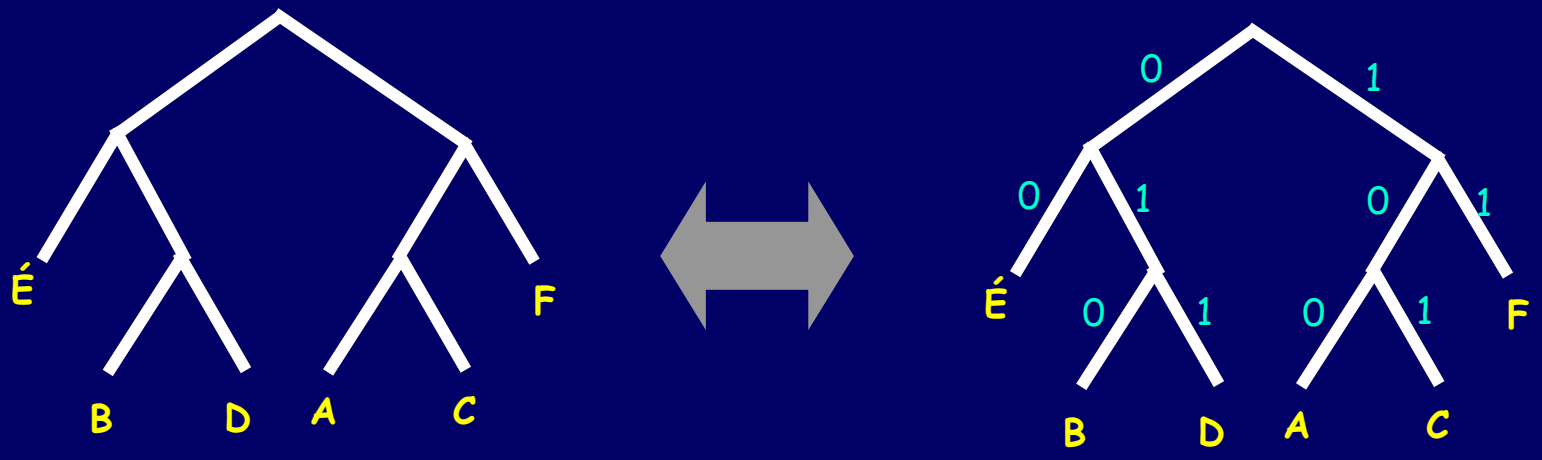


Theorem:

S has a decision tree of depth Δ

if and only if

S has a prefix-free code with all codewords bounded by length Δ



Theorem:

S has a decision tree of depth Δ

if and only if

S has a prefix-free code with all codewords bounded by length Δ

Extends to infinite sets

Let S is a subset of Σ^*

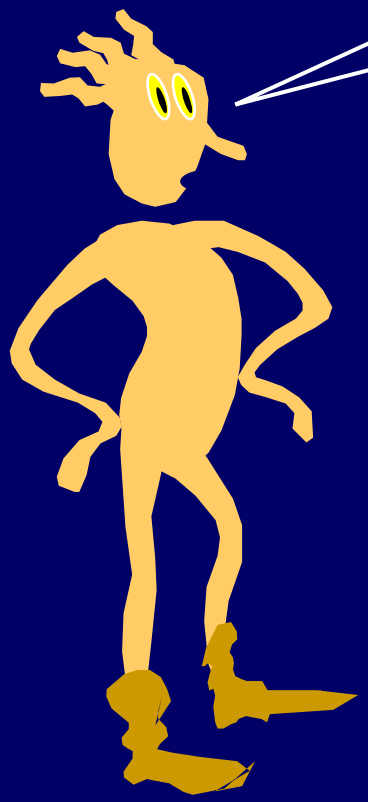
Theorem:

S has a decision tree where all length n elements of S have depth $\leq \Delta(n)$

if and only if

S has a prefix-free code where all length n strings in S have encodings of length $\leq \Delta(n)$

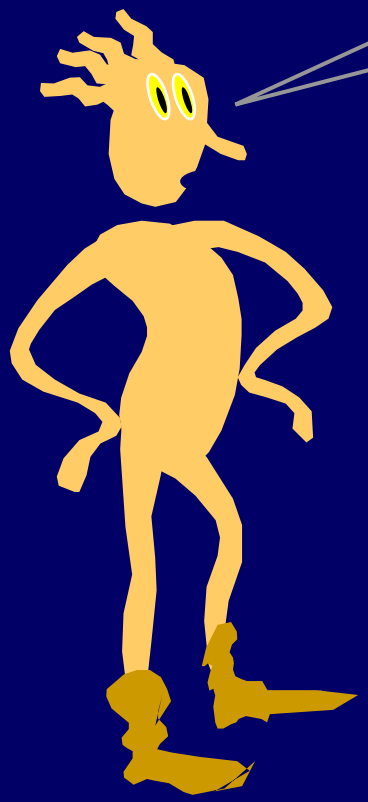
I am thinking of some natural number k.
ask me YES/NO questions in order to determine k.



Let d(k) be the number of questions that you ask when I am thinking of k.

Let $\Delta(n)$ = max { d(k) over n-bit numbers k }.

I am thinking of some natural number k - ask me YES/NO questions in order to determine k.



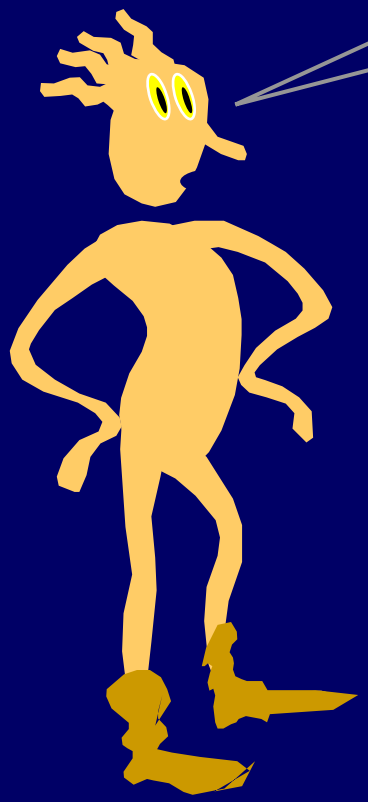
Naïve strategy: Is it 0? 1? 2? 3? …

$d(k) = k+1$

$\Delta(n) = 2^{n+1}$ since $2^{n+1} - 1$ uses only n bits.

Effort is exponential in length of k !!!

I am thinking of some natural number k - ask me YES/NO questions in order to determine k.

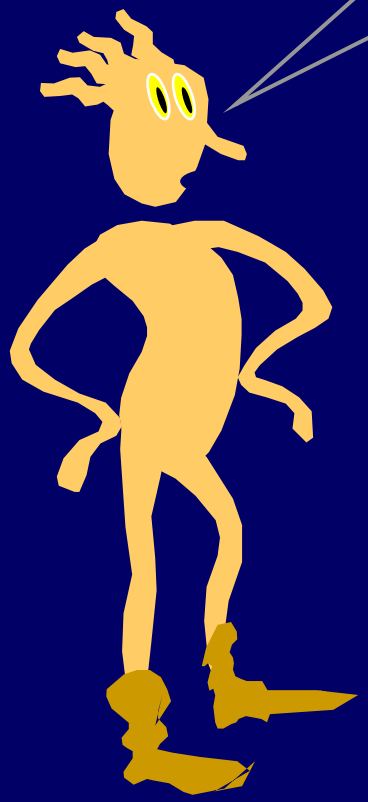


What is an efficient question strategy?

I am thinking of some natural number k...

k is number

$n = |k|$ = # of bits



Does k have length 1? NO

Does k have length 2? NO

Does k have length 3? NO

...

Does k have length n? YES

Do binary search on strings of length n.

$d(k) = |k| + |k|$

$= 2 (\lfloor \log k \rfloor + 1)$

$\Delta(n) = 2n$

Size First/ Binary Search

Does k have length 1? NO

Does k have length 2? NO

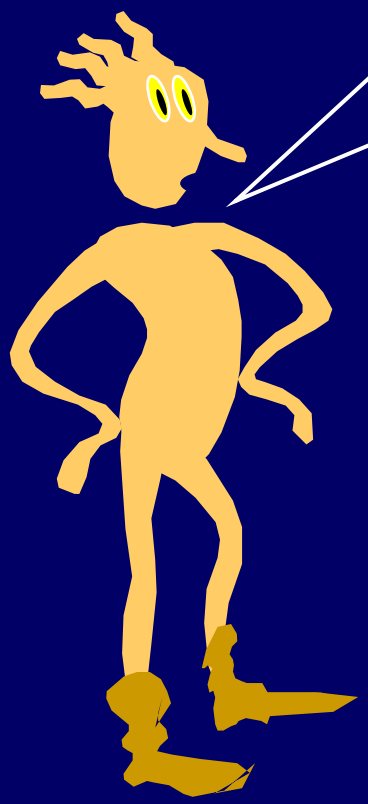
Does k have length 3? NO

...

Does k have length n? YES

Do binary search on strings of length n.

What prefix-free code corresponds to the Size First / Binary Search decision strategy?



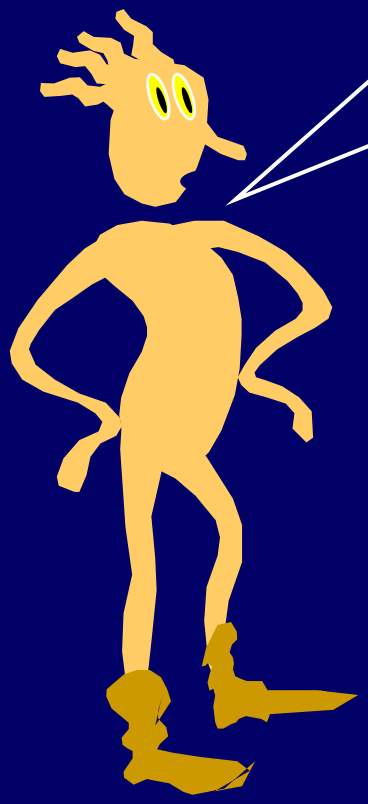
$k = 15 = (1111)_2$

$f(k) =$ 0001 1111

f(k) = (|k| - 1) zeros, followed by 1, and then by the binary representation of k

$|f(k)| = 2\,|k|$

What prefix-free code corresponds to the Size First / Binary Search decision strategy?



Or,

length of k in unary $\Rightarrow$ |k| bits
k in binary $\Rightarrow$ |k| bits

Another way to look at f

k = 27 = 11011, and hence |k| = 5

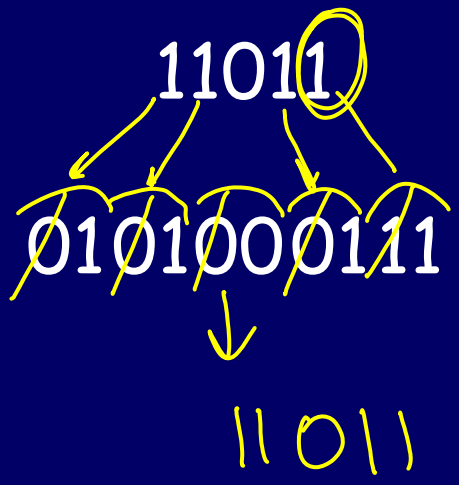
f(k) = 00001 11011

Another way to look at f

k = 27 = 11011, and hence |k| = 5

f(11011) = 00001 11011

g(11011) = 0101000111



Another way to look at the function g:

- g(final 0) -> 10
- g(final 1) -> 11
- g(all other 0's) -> 00
- g(all other 1's) -> 01

"Fat Binary" ⇔ Size First/Binary Search strategy

"Fat Binary"

FatBinary map does the following:

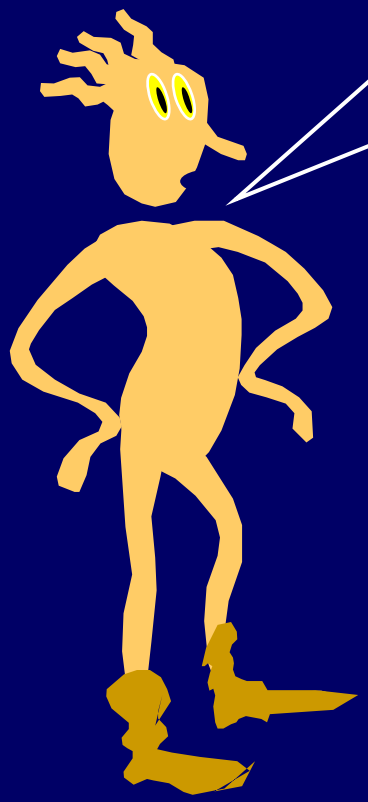
Map final bit b	->	1b
all previous bits b	->	0b

FatBin(00)	=	00 10
FatBin(1111)	=	01 01 01 11
FatBin(101010)	=	01 00 01 00 01 10

Is it possible to beat 2n questions
to find a number of length n?

Look at the prefix-free code…

Any obvious improvement suggest itself here?



the fat-binary map f concatenates

length of k in ~~unary~~ $\Rightarrow$ |k| bits

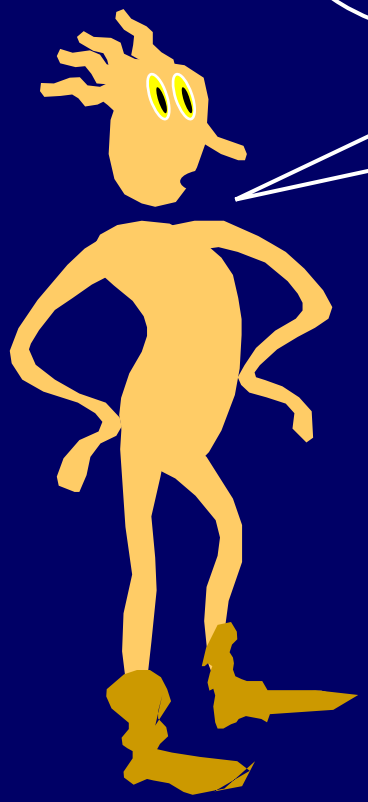
k in binary $\Rightarrow$ |k| bits

fat binary!

In fat-binary, $\Delta(n) \leq 2n$

Now $\Delta(n) \leq n + 2\,(\lfloor \log n \rfloor + 1)$

Can you do better?



better-than-Fat-Binary-code(k)
concatenates

length of k in fat binary $\Rightarrow$ 2||k|| bits
k in binary $\Rightarrow$ |k| bits

Hey, wait!

In a better prefix-free code

RecursiveCode(k) concatenates
RecursiveCode(|k|) & k in binary

better-t-better-thanFB

~~better than Fat Binary~~ code

	better-t-FB	||k|| + 2|||k|||
|k| in ~~fat binary~~	⇒	~~2|||k||| bits~~
k in binary	⇒	|k| bits

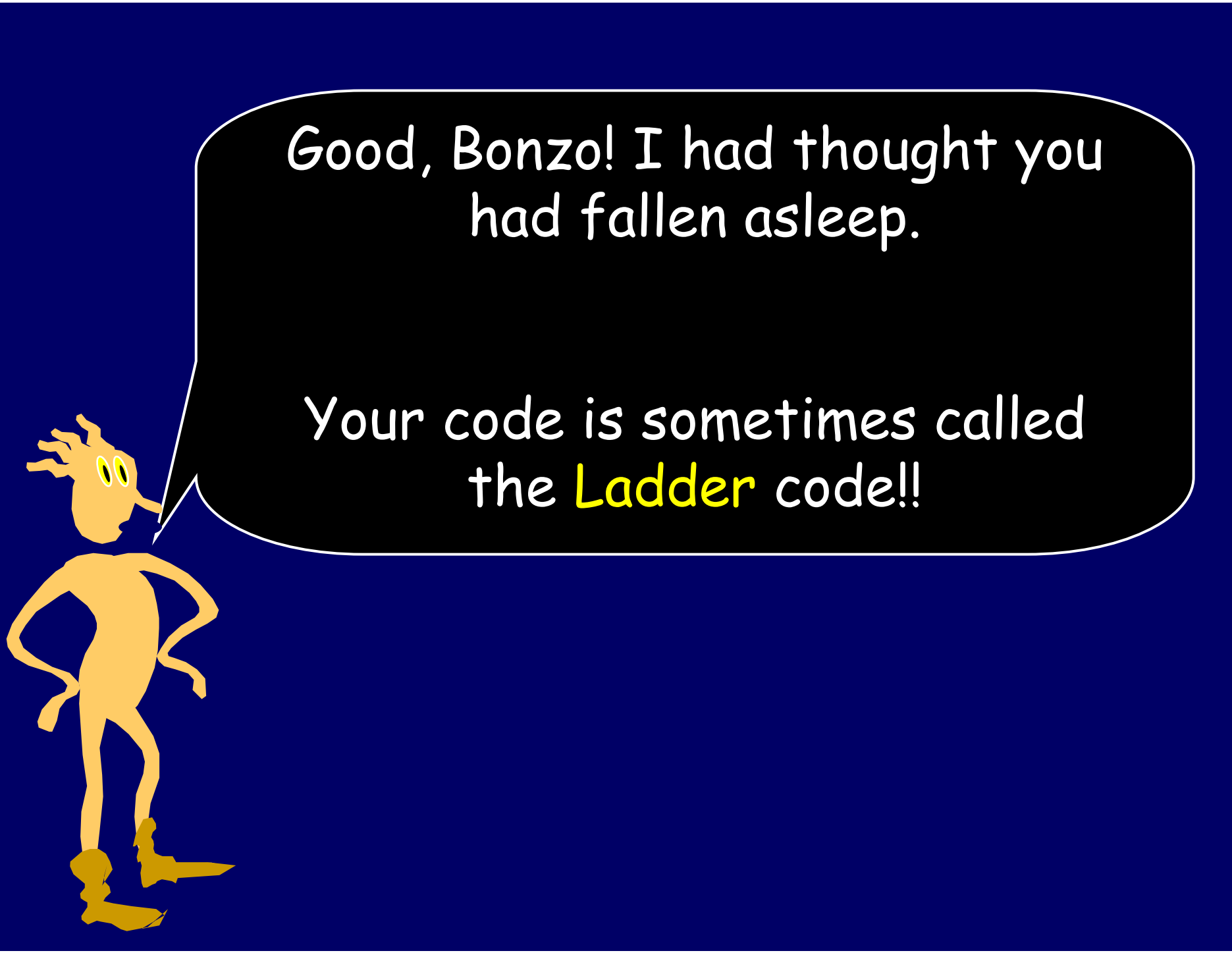
Oh, I need to remember how many levels of recursion r(k)

In the final code
F(k) = F(r(k)) . RecursiveCode(k)

r(k) = log* k

$(\log k + \log\log k + \log\log\log k + \ldots + 7 + \log^* k)$

Hence, length of F(k)
= $\overbrace{|k|}$ + ||k|| + |||k||| + … + 1
+ | log*k | + …


Good, Bonzo! I had thought you had fallen asleep.
Your code is sometimes called the Ladder code!!

Maybe I can do better…

Can I get a prefix code
for k with length ≈ log k ?

length

$\log k + 27$?

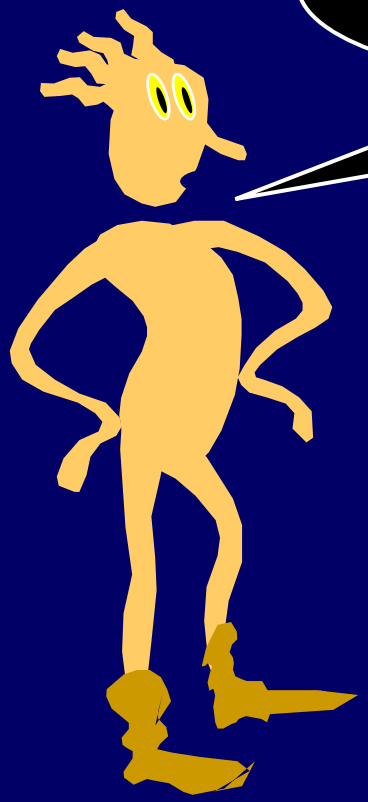
$\log k + \log\log k$?

$\log k + \log\log k + 2|||k|||$

$2\log\log\log k$

No!

Let me tell you why
length $\approx \log k$
is not possible



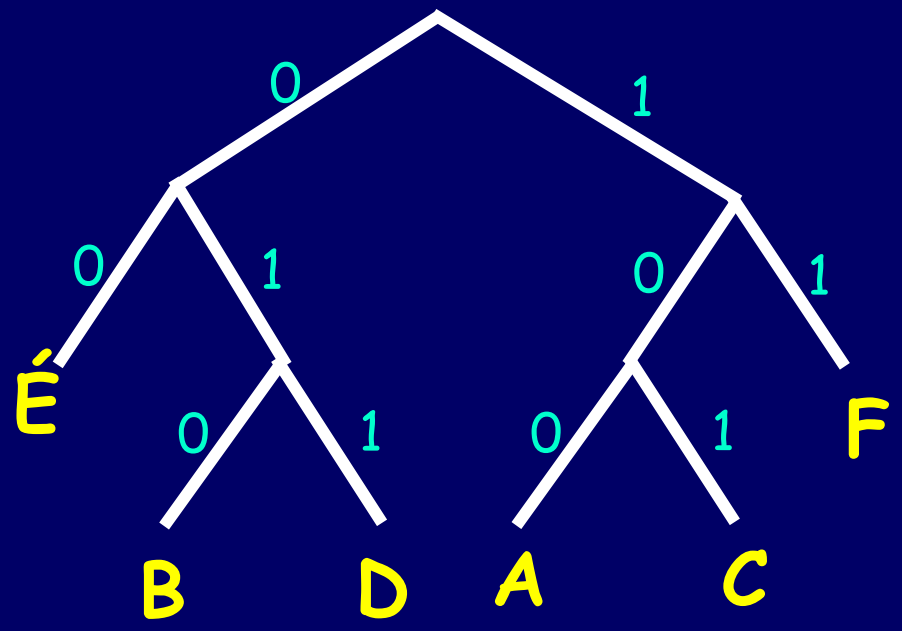
Decision trees have a natural probabilistic interpretation.

Let T be a decision tree for S.

Start at the root, flip a fair coin at each decision, and stop when you get to a leaf.

Each sequence **w** in S will be hit with probability $1/2^{|w|}$

Random walk down the tree



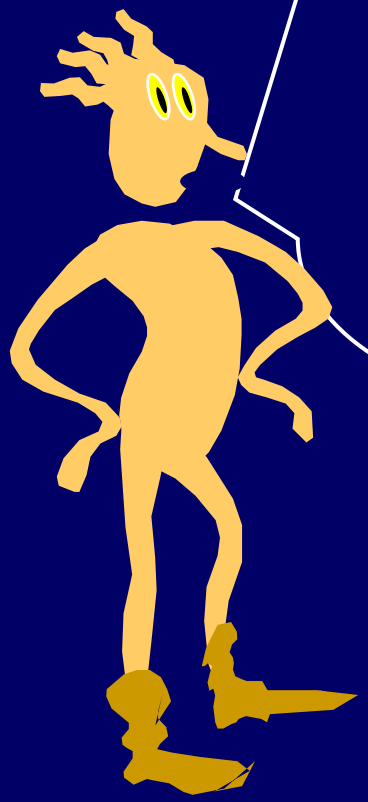
Each sequence w in S will
be hit with probability $1/2^{|w|}$

Hence, Pr(F) = $\frac{1}{4}$, Pr(A) = 1/8, Pr(C) = 1/8, …

Let T be a decision tree for S
(possibly countably infinite set)

The probability that some element in S is hit by a random walk down from the root is

$$\sum_{w \in S} 1/2^{|w|} \leq 1$$

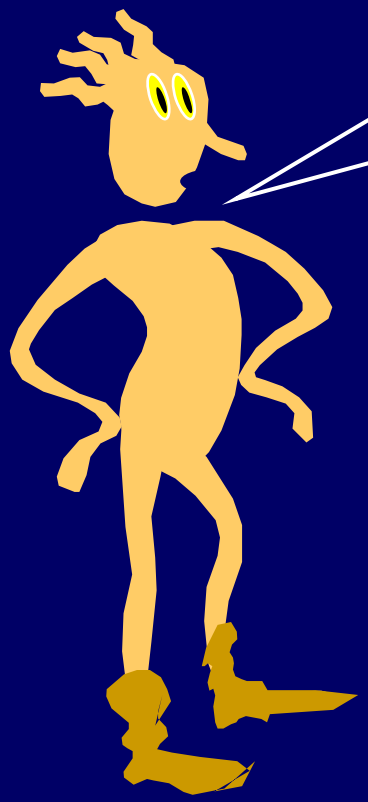


Kraft Inequality

Let S be any prefix-free code.

Kraft Inequality:

$$\sum_{w \in S} 1/2^{|w|} \leq 1$$



Fat Binary:

f(k) has $2|k| \approx 2 \log k$ bits

$$\frac{1}{2^{2 \log k}} = \frac{1}{(2^{\log k})^2} = \frac{1}{k^2}$$

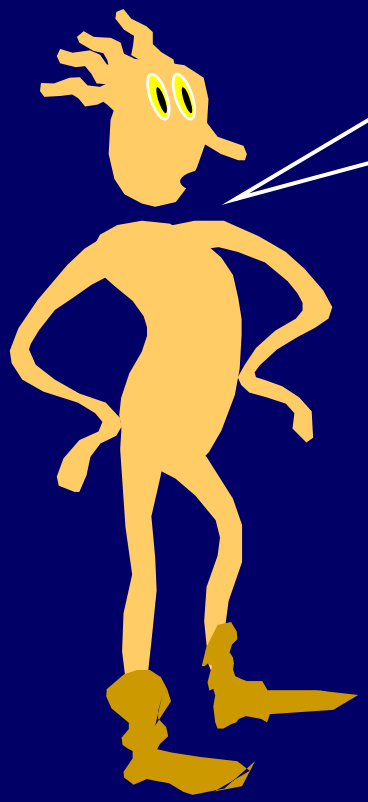
$$\sum_{k \in \mathbb{N}} \tfrac{1}{2}^{|f(k)|} \leq 1$$

$$\approx \sum_{k \in \mathbb{N}} 1/k^2 \rightsquigarrow \pi^2/6$$

Let S be any prefix-free code.

Kraft Inequality:

$\sum_{w \in S} 1/2^{|w|} \leq 1$



Better-than-FatB Code:

f(k) has $|k| + 2||k||$ bits

$\log k + 2 \log\log k$

$\sum_{k \in \mathbb{N}} \frac{1}{2}^{|f(k)|} \leq 1$

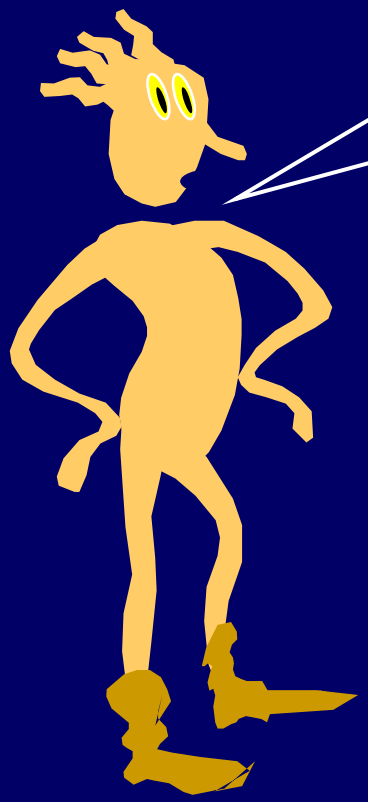
$\frac{1}{2^{(\log k + 2\log\log k)}} = \frac{1}{k(\log k)^2}$

$\approx \sum_{k \in \mathbb{N}} 1/(k\,(\log k)^2)$

Let S be any prefix-free code.

Kraft Inequality:

$$\sum_{w \in S} 1/2^{|w|} \leq 1$$



Ladder Code: k is represented by

$|k| + ||k|| + |||k||| + \ldots$ bits

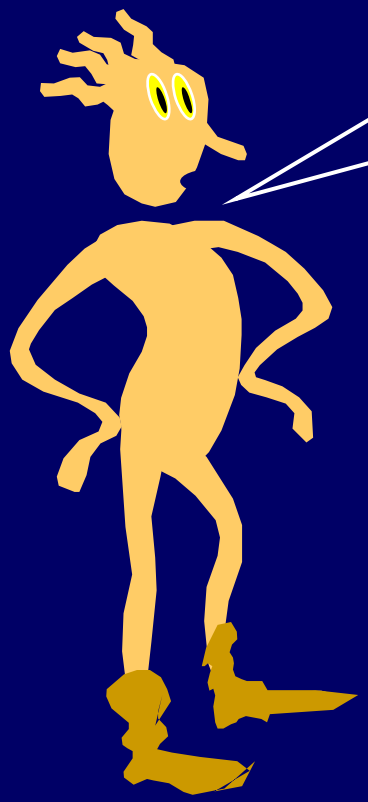
$$\sum_{k \in \mathbb{N}} \tfrac{1}{2}^{|f(k)|} \leq 1$$

$$\approx \sum_{k \in \mathbb{N}} 1/(k \log k \log\log k \ldots)$$

Let S be any prefix-free code.

Kraft Inequality:

$$\sum_{w \in S} 1/2^{|w|} \leq 1$$



Can a code that represents k by $|k| = \log k$ bits exist?

$= \log k + \log\log k$? $\longrightarrow \sum_{k \in \mathbb{N}} \frac{1}{k \log k}$

No, since $\sum_{k \in \mathbb{N}} 1/k$ diverges !!

So you can't get log n, Bonzo...

To Summarize

Used prefix-free codes to model decision trees.

We developed codes:

Fat-binary represents k using $2|k|$ bits.

Ladder code represents k using $|k| + ||k|| + \ldots + \log^* k$ bits.

Kraft Inequality

if S is prefix free code, then $\sum_{w \text{ in } S} 2^{|w|} \leq 1$

Cannot have prefix free codes with length $\approx |k|$ bits.

Back to compressing words

The optimal-depth decision tree
for any set S with (k+1) elements has depth

$$\lfloor \log_2 k \rfloor + 1$$

$$\Downarrow$$

The optimal prefix-free code
for A-Z + "space" has length

$$\lfloor \log_2 26 \rfloor + 1 = 5$$

English Letter Frequencies

But in English, different letters occur with different *frequencies*.

A 8.1%	F 2.3%	K .79%	P 1.6%	U 2.8%	Z .04%
B 1.4%	G 2.1%	L 3.7%	Q .11%	V .86%	
C 2.3%	H 6.6%	M 2.6%	R 6.2%	W 2.4%	
D 4.7%	I 6.8%	N 7.1%	S 6.3%	X .11%	
E 12%	J .11%	O 7.7%	T 9.0%	Y 2.0%	

ETAONIHSRDLUMWCFGYPBVKQXJZ

short encodings!

Why should we try to minimize the maximum length of a codeword?

If encoding A-Z, we will be happy if the average codeword is short.

Morse Code

A .-	F ..-.	K -.-	P .--.	U ..-	Z --..
B -...	G --.	L .-..	Q --.-	V ...-	
C -.-.	H	M --	R .-.	W .--	
D -..	I ..	N -.	S ...	X -..-	
E .	J .---	O ---	T -	Y -.--	

ETAONIHSRDLUMWCFGYPBVKQXJZ

Scrabble

A: 9, B: 2, C: 2, D: 4,
E: 12, F: 2, G: 3, H: 2,
I: 9, J: 1, K: 1, L: 4, M: 2, N: 6,
O: 8, P: 2, Q: 1, R: 6, S: 4, T: 6,
U: 4, V: 2, W: 2, X: 1, Y: 2, Z: 1,
blank: 2

Given frequencies for A-Z, what is the optimal prefix-free encoding of the alphabet?

I.e., one that minimizes the average code length

Huffman Codes: Optimal Prefix-free Codes Relative to a Given Distribution

Here is a Huffman code based on the English letter frequencies given earlier:

A 1011	F 101001	K 10101000	P 111000	U 00100
B 111001	G 101000	L 11101	Q 1010100100	V 1010101
C 01010	H 1100	M 00101	R 0011	W 01011
D 0100	I 1111	N 1000	S 1101	X 1010100101
E 000	J 1010100110	O 1001	T 011	Y 101011
				Z 1010100111

But Huffman coding uses only letter frequencies.

For any fixed language, we can use correlations!
E.g., Q is almost always followed by U…

Random words

Randomly generated letters from A-Z, space not using the frequencies at all:

XFOML RXKHRJFFJUJ ALPWXFWJXYJ
FFJEYVJCQSGHYD QPAAMKBZAACIBZLKJQD

Random words

Using only single character frequencies:

OCRO HLO RGWR NMIELWIS EU LL NBNESEBYA TH
EEI ALHENHTTPA OOBTTVA NAH BRL

Random words

Each letter depends on the previous letter:

ON IE ANTSOUTINYS ARE T INCTORE ST BE S
DEAMY ACHIN D ILONASIVE TUCOOWE AT
TEASONARE FUSO TIZIN ANDY TOBE SEACE CTISBE

Random words

Each letter depends on 2 previous letters:

IN NO IST LAT WHEY CRATICT FROURE BIRS
GROCID PONDENOME OF DEMONSTURES OF THE
REPTAGIN IS REGOACTIONA OF CRE

Random words

Each letter depends on 3 previous letters:

THE GENERATED JOB PROVIDUAL BETTER TRAND THE DISPLAYED CODE, ABOVERY UPONDULTS WELL THE CODERST IN THESTICAL IT DO HOCK BOTHEMERG.

(INSTATES CONS ERATION. NEVER ANY OF PUBLE AND TO THEORY. EVENTIAL CALLEGAND TO ELAST BENERATED IN WITH PIES AS IS WITH THE)

More examples

Order-1: t I amy, vin. id wht omanly heay atuss n macon aresethe hired boutwhe t, tl, ad torurest t plur I wit hengamind tarer-plarody thishand.

Order-2: Ther I the heingoind of-pleat, blur it dwere wing waske hat trooss. Yout lar on wassing, an sit." "Yould," "I that vide was nots ther.

Order-3: I has them the saw the secorrow. And wintails on my my ent, thinks, fore voyager lanated the been elsed helder was of him a very free bottlemarkable,

Order-4: His heard." "Exactly he very glad trouble, and by Hopkins! That it on of the who difficentralia. He rushed likely?" "Blood night that.

http://www.cs.bell-labs.com/cm/cs/pearls/sec153.html

Based on patterns of words

Order-1: The table shows how many contexts; it uses two or equal to the sparse matrices were not chosen. In Section 13.1, for a more efficient that ``the more time was published by calling recursive structure translates to build scaffolding to try to know of selected and testing and more robust and a binary search).

Order-2: The program is guided by verification ideas, and the second errs in the STL implementation (which guarantees good worst-case performance), and is especially rich in speedups due to Gordon Bell. Everything should be to use a macro: for *n=10,000*, its run time; that point Martin picked up from his desk

Order-3: A Quicksort would be quite efficient for the main-memory sorts, and it requires only a few distinct values in this particular problem, we can write them all down in the program, and they were making progress towards a solution at a snail's pace.

http://www.cs.bell-labs.com/cm/cs/pearls/sec153.html

Other Techniques

Lossless compression:

- Arithmetic Coding
- JPEG-LS, PPM
- Lempel Ziv encoding LZ77, LZW (gzip)
- Burrough-Wheeler (bzip)

Lossy compression:

- JPEG, MPEG
- Fractal compression

References

The Mathematical Theory of Communication, by C. Shannon and W. Weaver

http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html

Elements of Information Theory, by T. Cover and J. Thomas

Programming Pearls, by Jon Bentley

http://www.cs.bell-labs.com/cm/cs/pearls/