UNION-FIND datastructure for sets

FIND(u) – return component containing u

UNION(u,v) – merge components of u & v

Keep each component as a list

Merge smaller into bigger

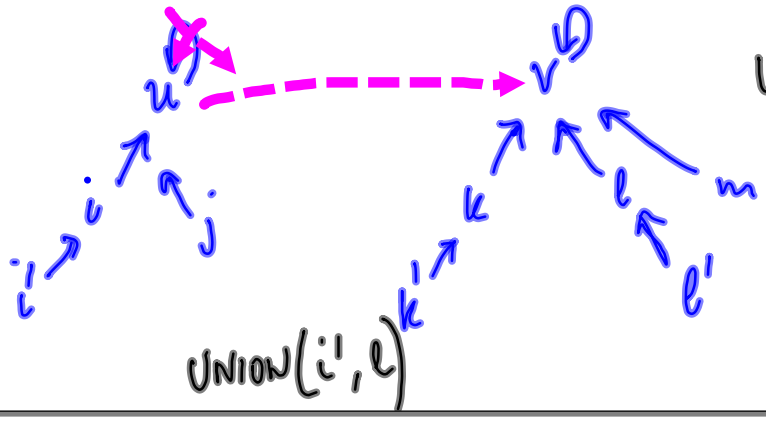
AMORTISED COMPLEXITY $O(n \log n)$

to merge n singleton in n steps into a single components

Slightly better

Maintain each component as a "tree"

After some merges



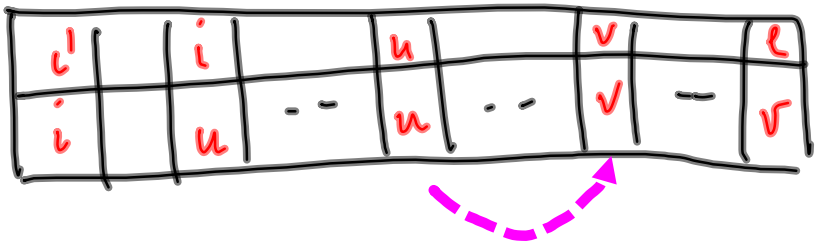
FIND(x) - follow path to root

UNION(x, y)

FIND(x) → u

FIND(y) → v

Make smaller of u, v child of other



Complexity of FIND(x)? - time proportional to depth of x

Merging smaller into bigger ensures depth(x) increases by 1 at most log n times

Clever trick

Path Compression

After every FIND(x) = u

Restart from x & set parent(y) = u for every element on path from x to u

After very careful accounting

path compression gives amortized overall complexity of $O(n \cdot \alpha(n))$

↓

inverse Ackermann function

For all likely values you will ever see,

$\alpha(n) \leq 4$

Doesn't reduce complexity of Kruskal's algo → E log E sort of edge weights

Data compression

Alphabet – set of symbols A

Text – string over this alphabet

Encode text in binary

Natural strategy : $|A| = n$

Use $\lceil \log n \rceil$ bits per symbol

b bits per symbol $\Rightarrow$ text of length m
encoding is $m \cdot b$ bits

Even Morse knew to do letter

Frequently occurring symbols have shorter encodings

Morse code → dots, dashes + pauses

. — . . —
1 1 11 1
0 1 0 0 1

e → . 0
t → — 1
a → .— 01

= ternary alphabet!

0101 → ?

etet
aa

eta
ael-

Unambiguous decoding

Coding is prefix free

Suppose γ is my coding function

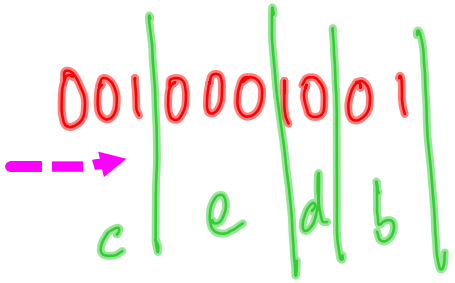
$\gamma : A \longrightarrow$ strings over $\{0,1\}$

$\forall x, y$ $\gamma(x)$ is not a prefix of $\gamma(y)$ & v.v.

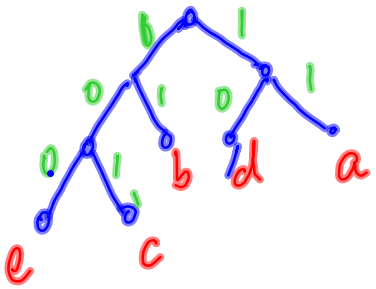
Morse code is not prefix free

$\gamma(e) = 0$ is a prefix of $\gamma(a) = 01$

Example: $A = \{a, b, c, d, e\}$ γ: $a \mapsto 11$, $b \mapsto 01$, $c \mapsto 001$, $d \mapsto 10$, $e \mapsto 000$



Represent a prefix free code as a tree



"full" binary tree
- any non leaf has 2 children

Any non-full tree can be "compressed" to a full tree

"Optimal" encoding

Requires some information about A

Assumption: $\forall x \in A$, f_x is relative frequency of x

$$\sum_{x \in A} f_x = 1$$

e.g. $A = \{a, b, c, d, e\}$ $f_a = 0.32$, $f_b = 0.25$, $f_c = 0.20$

$f_d = 0.18$, $f_e = 0.05$

Length of encoding of text with m symbols

$m \cdot f_x$ occurrences of x

each requires $|\gamma(x)|$ bits

$$\sum_{x \in A} (m \cdot f_x) |\gamma(x)| = m \sum_{x \in A} f_x \cdot |\gamma(x)|$$

$\sum_{x \in A} f_x \cdot |\gamma(x)|$: ABL(γ) average bit length per symbol

How to find optimal γ (lowest ABL(γ)) ?

Recall : γ can be represented as full binary tree with leaves labelled by A

Fix the skeleton of tree T - do not label the leaves

Suppose $f_x < f_y$ but $|\gamma(x)| < |\gamma(y)|$

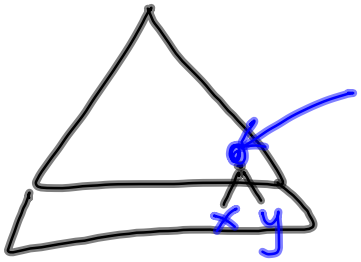
Swap $\gamma(x), \gamma(y)$ → improves ABL(γ)

Highest leaves have highest frequencies
lowest leaves have lowest frequencies

Huffman Encoding

Lowest leaf's sibling is also a leaf

Pick 2 lowest frequency letters, make them lowest leaves x, y



Dummy letter (xy) with frequency f(x)+f(y)

Recursively encode $A' = A \setminus \{x, y\} \cup \{(xy)\}$