

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 $O(n \log n)$

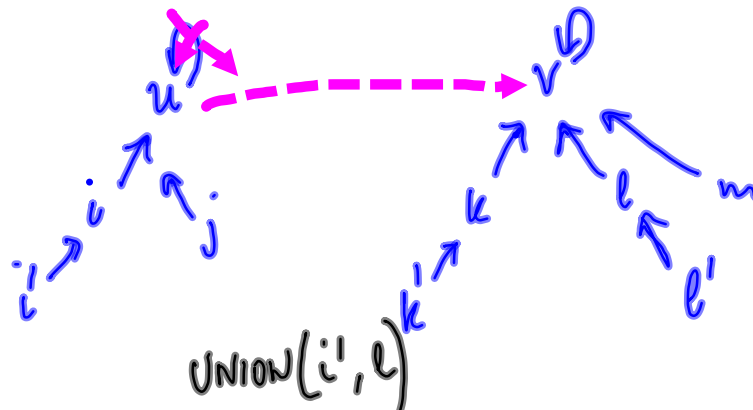
to merge n singletons in n steps into a single components

Slightly better

Maintain each component as a "tree"



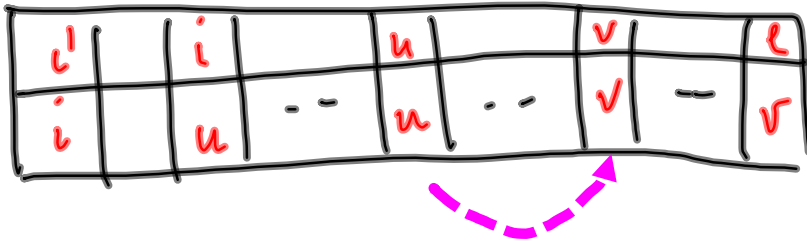
After some merges



$\text{FIND}(x)$ - follow path to root

$\text{UNION}(x, y)$

$\text{FIND}(x) \rightarrow u$
 $\text{FIND}(y) \rightarrow v$
 Make smaller of u, v
 child of other



Complexity of $\text{FIND}(x)$? — time proportional to
depth of x

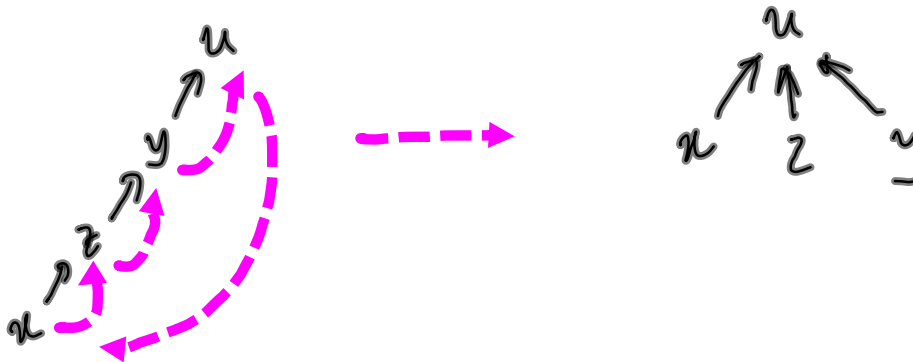
Merging smaller into bigger
ensures $\text{depth}(x)$ increases by 1 at most
 $\log n$ times

Clever trick

Path Compression

After every $\text{FIND}(x) = u$

Restart from x & set $\text{parent}(y) = u$ 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,

Doesn't reduce complexity $\alpha(n) \leq 4$
of Kruskal's algo $\rightarrow E \log E$ 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 better

Frequently occurring symbols have shorter encodings

Morse code \rightarrow dots, dashes
+ pauses

• — • • —
| | | | |
0 1 0 0 1

e \rightarrow • 0

t \rightarrow — 1

a \rightarrow • — 01

= ternary
alphabet!

0101 \rightarrow ?

et et

aa

et a

aet-

Unambiguous decoding

Coding is prefix free

Suppose γ is my
coding function

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

$\forall x, y \quad \gamma(x)$ 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 : $a \mapsto 11$

$b \mapsto 01$

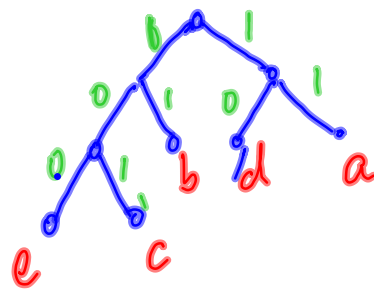
$c \mapsto 001$

$d \mapsto 10$

$e \mapsto 000$

001|000|10|01|
 c|e|d|b|

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 $|r(x)|$ bits

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

$ABL(r)$ average bit
length per symbol

How to find optimal γ (lowest $ABL(\gamma)$) ?

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

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

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

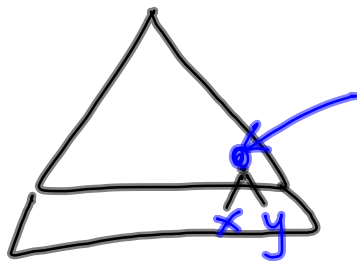
Swap $\gamma(x), \gamma(y) \rightarrow$ improves $ABL(\gamma)$

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

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